Neo4j Tutorial

Author: Atef Bader, PhD

TA: Prakhar Nag

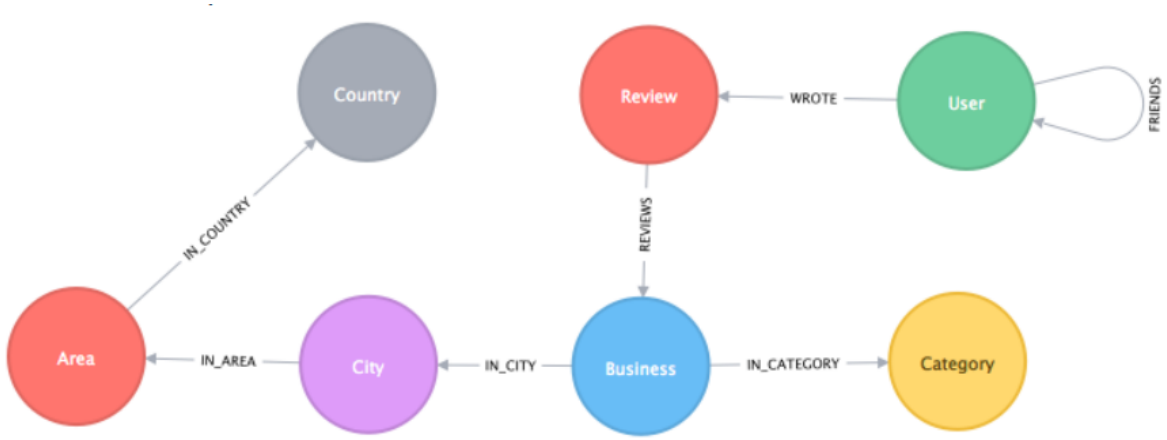
Material & References:

- What's a Graph Database?
  - https://neo4j.com/developer/graph-database/
- Neo4j Graph Database
  - https://neo4j.com/
- Neo4j Graph Data Science Library
  - https://neo4j.com/docs/graph-data-science/current/
  - Algorithms
    - Centrality
      - PageRank vs. ArticleRank
    - Community Detection
    - Similarity
      - Node
      - Jaccard
      - Cosine
    - Path Finding
      - Shortest path
      - Breadth first search
      - Depth fist search
    - Link Prediction
- How to Design Retail Recommendation Engines with Neo4j
  - https://www.youtube.com/watch?v=oMTmG4ClO5I
- Yelp dataset with Neo4j Graph Algorithms
  - https://neo4j.com/docs/graph-algorithms/current/yelp-example/
  - https://www.youtube.com/watch?v=7f2Tdn94JhY
  - https://github.com/mneedham/yelp-graph-algorithms

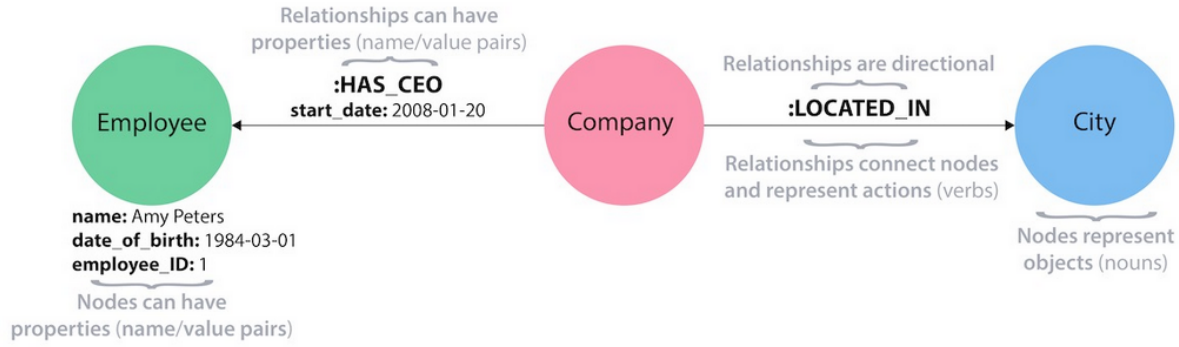
What can I do with Graph Database?

Create the knowledge graph from social, professional, transactional, etc. data.

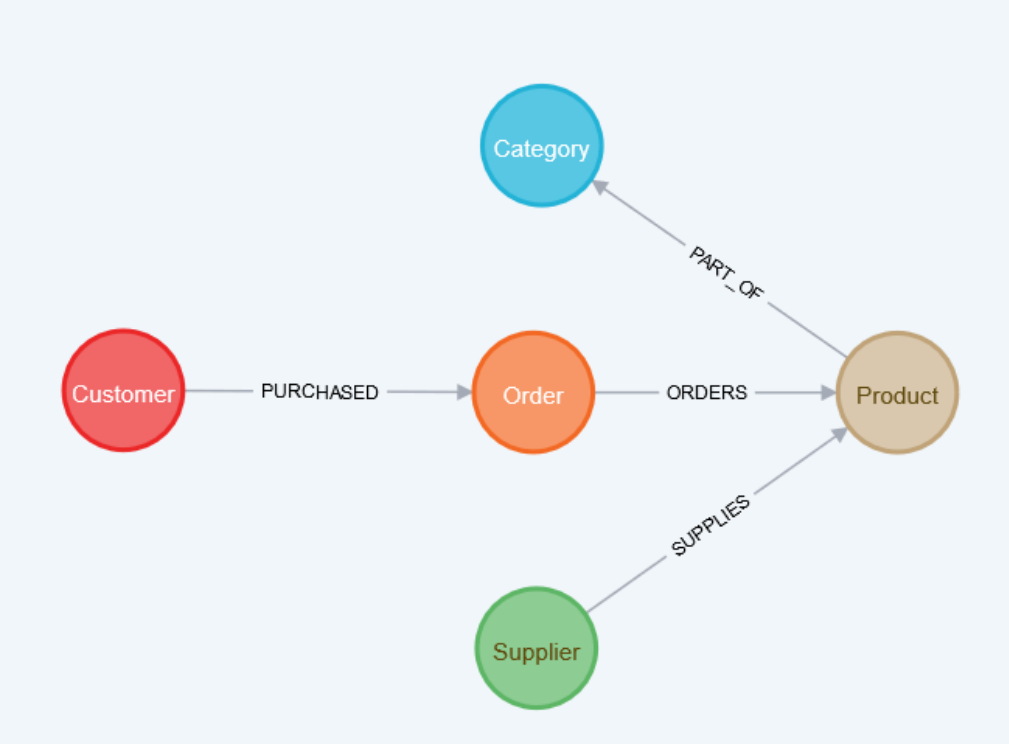
Consider, for example, the following Graph Model:



The graph has **nodes** with **labels** (User, Review, Area, etc.) that can have **relationship** of different **types** (FRIENDS, WROTE, IN_CATEGORY, etc.). All of the meta-data is stored as **properties** of nodes and relationships



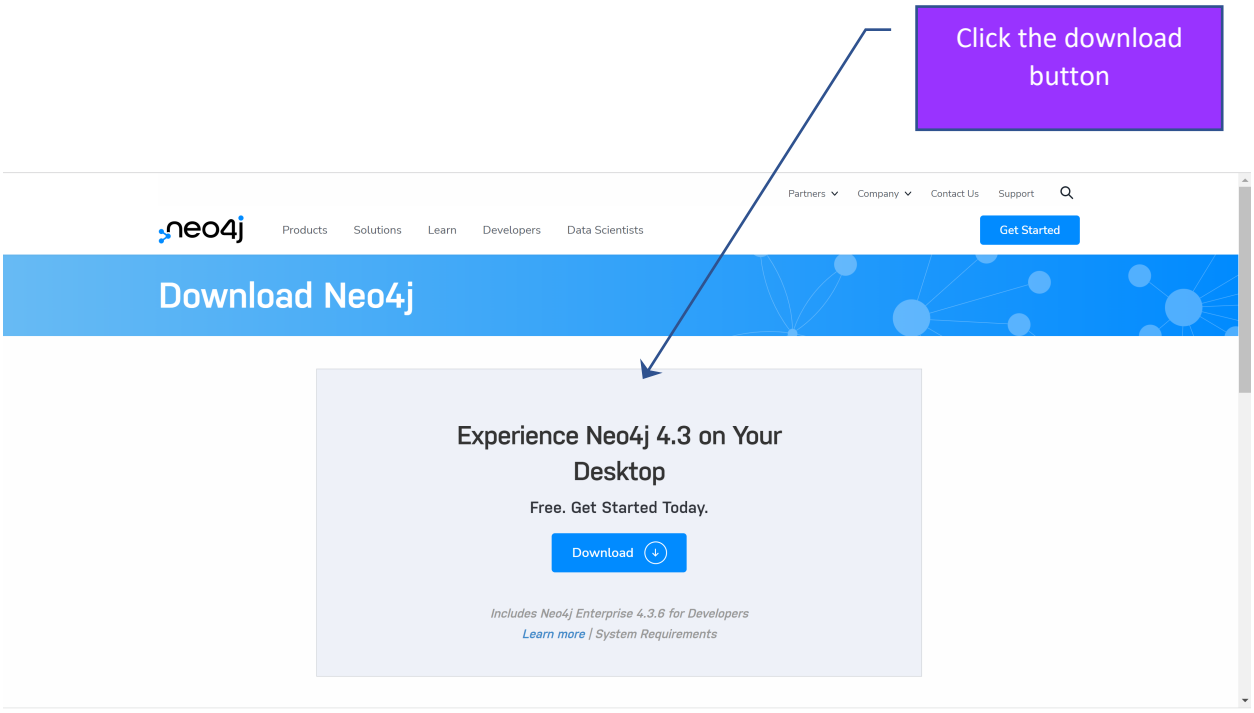
Graph model depends on the application in certain domain. The application will analyze the graph with graph algorithms.

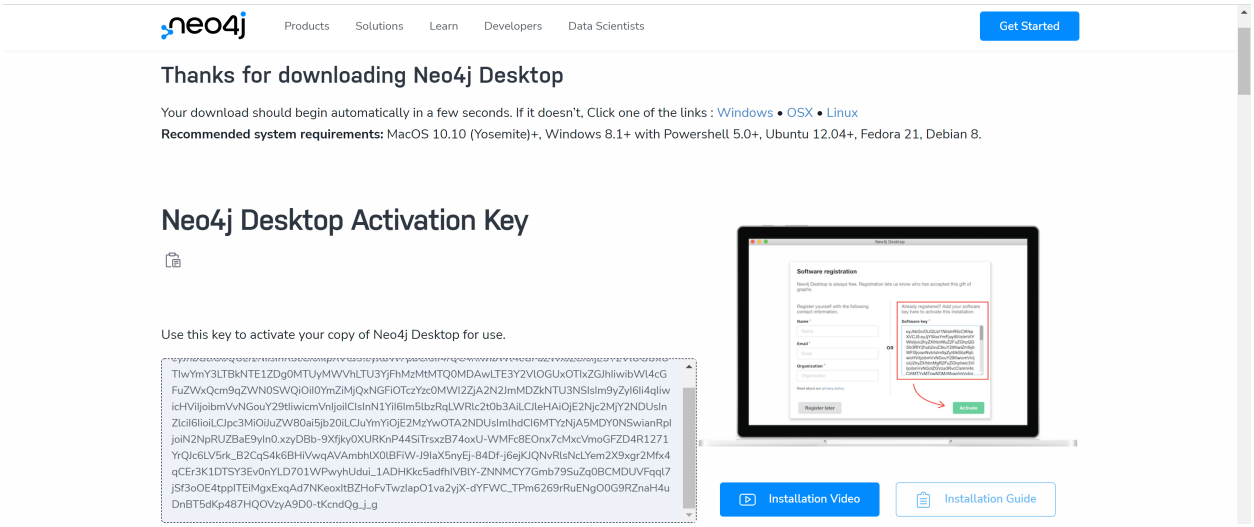


High Level Description

In this tutorial you will create and query a Graph database for retailer-based graph database application that has retailers, employees, customers, memberships for reward points, online personal accounts, and approved/disputed transactions.

For this tutorial, you need to download Neo4j **Desktop** onto your computer (Windows, Mac, Linux platforms are supported) from the following URL (https://neo4j.com/download/).





Copy YOUR generated key by Neo4j to activate your installation

Watch this video BEFORE you install Neo4j

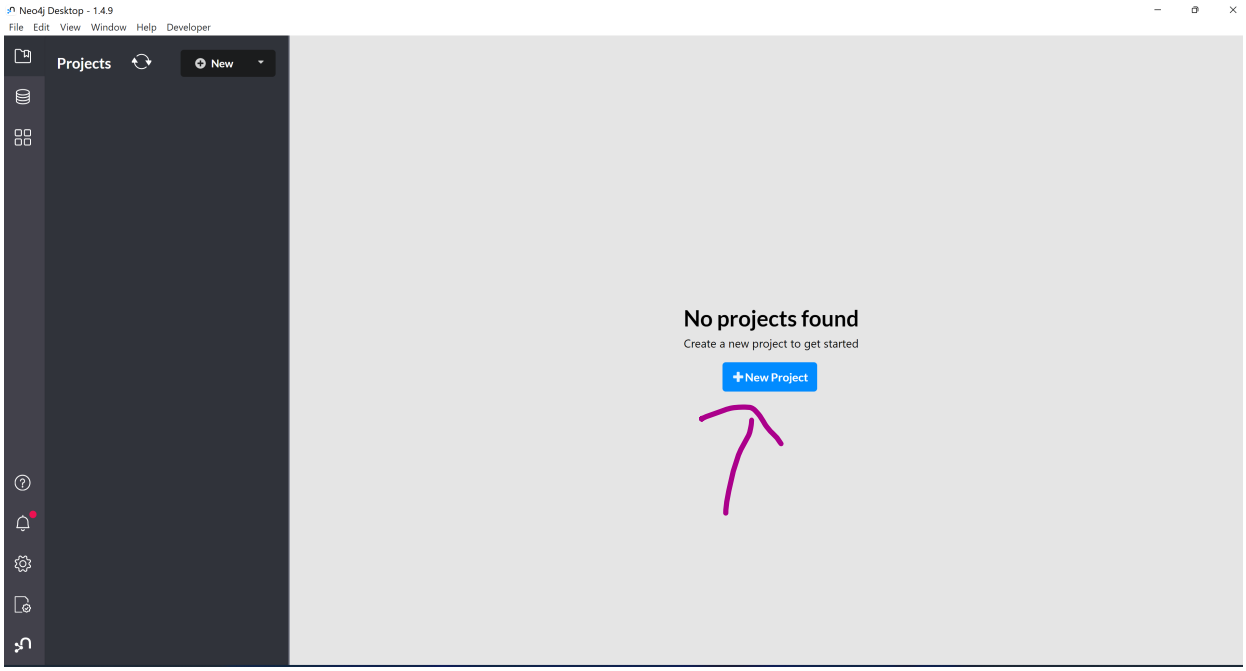
Graph Database Application - Requirements and Specifications

After successful installation of Neo4J on your personal computer, startup a new project and add a database with the name tutorial.

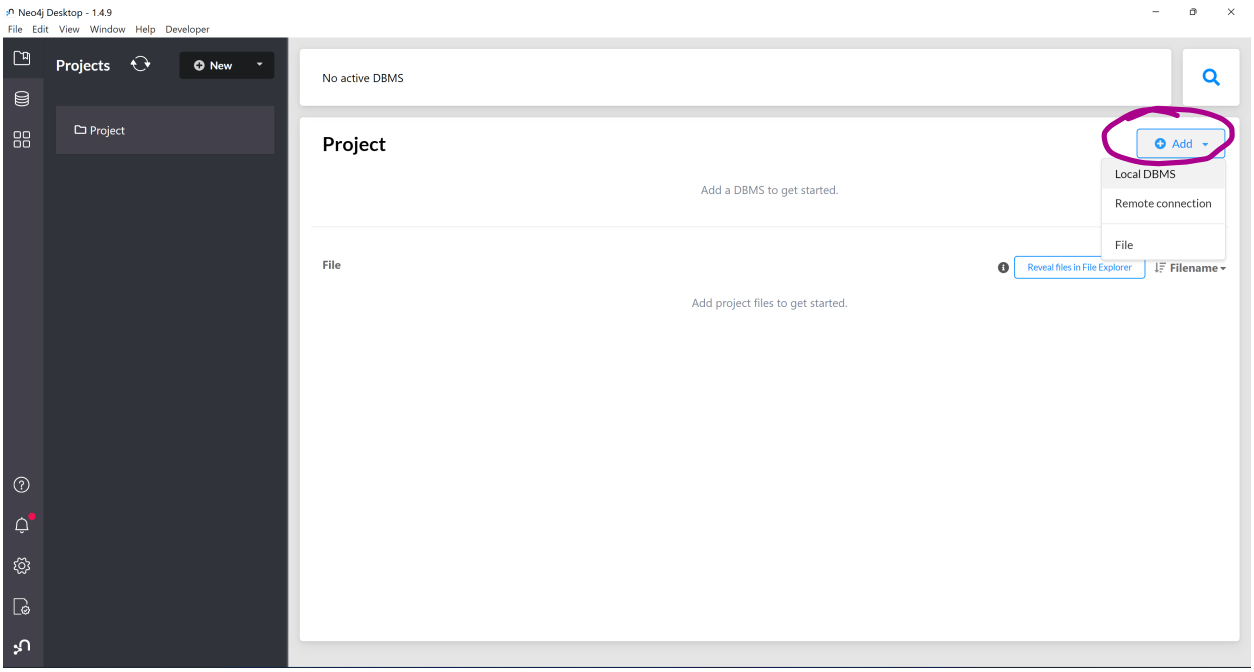
Create the Graph database for Tutorial based on the following business rules:

1. There are four Labels:
    a. Employee
    b. Retailer
    c. Customer
    d. Account
2. There are four relationships:
    a. WORKS_AT
        - Employee WORKS_AT Retailer
    b. SHOPPED_AT
        - Customer SHOPPED_AT Retailer
    c. HAS_MEMBERSHIP_AT
        - Customer HAS_MEMBERSHIP_AT Retailer
    d. USES_ACCOUNT
        - Customer USES_ACCOUNT Account
3. There are number of retailers that the customer can shop at.
4. The retailer has several employees.
5. The employee may work in more than one retailer.
6. The customer may shop at multiple retailers; hence, a customer might have one or more transactions with one or more retailers.
7. Every transaction has a date (timestamp) associated with it, and the transaction could be either Approved or Disputed.
8. The customer might have personal online accounts.
9. Customers might share the same online account; for example, a customer might share the account with the spouse.

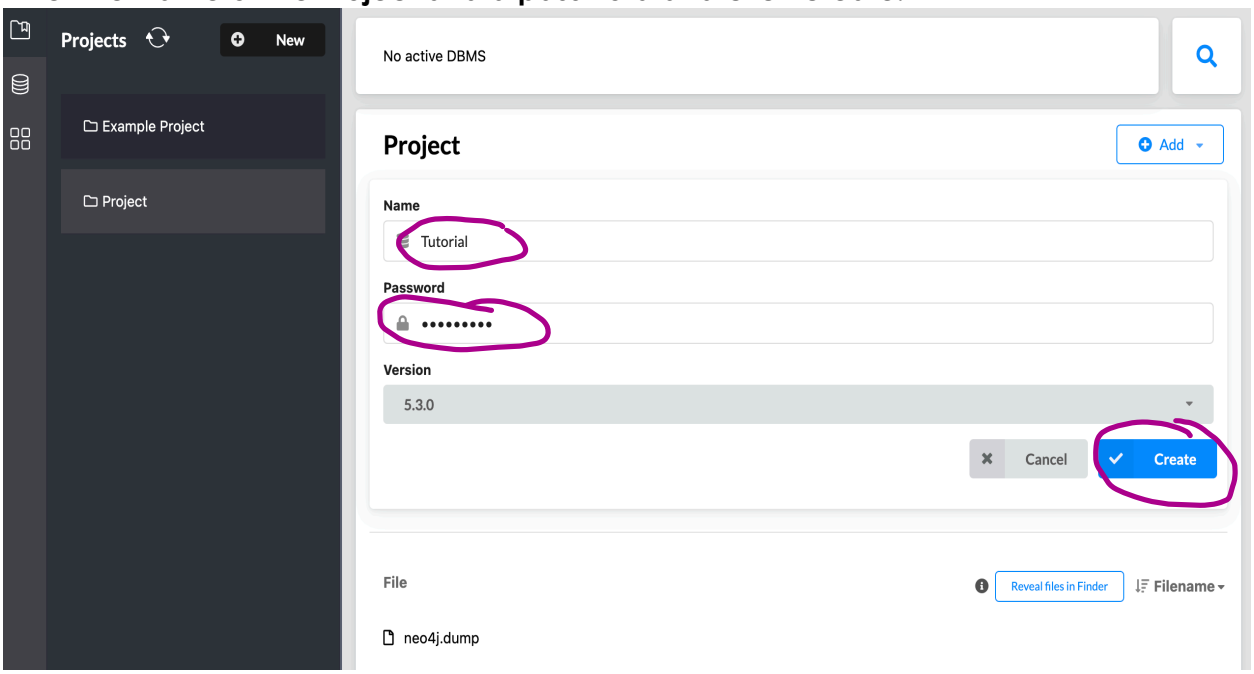
Create a New Project:



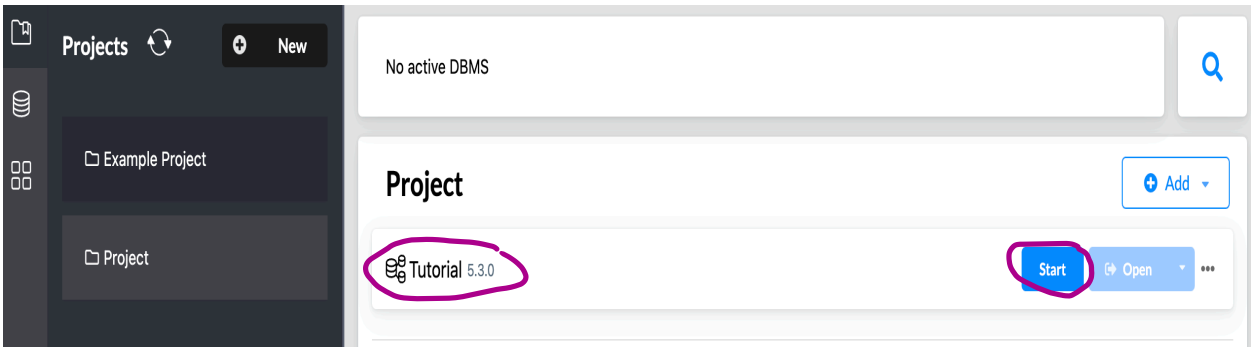
Click on Add button and choose Local DBMS



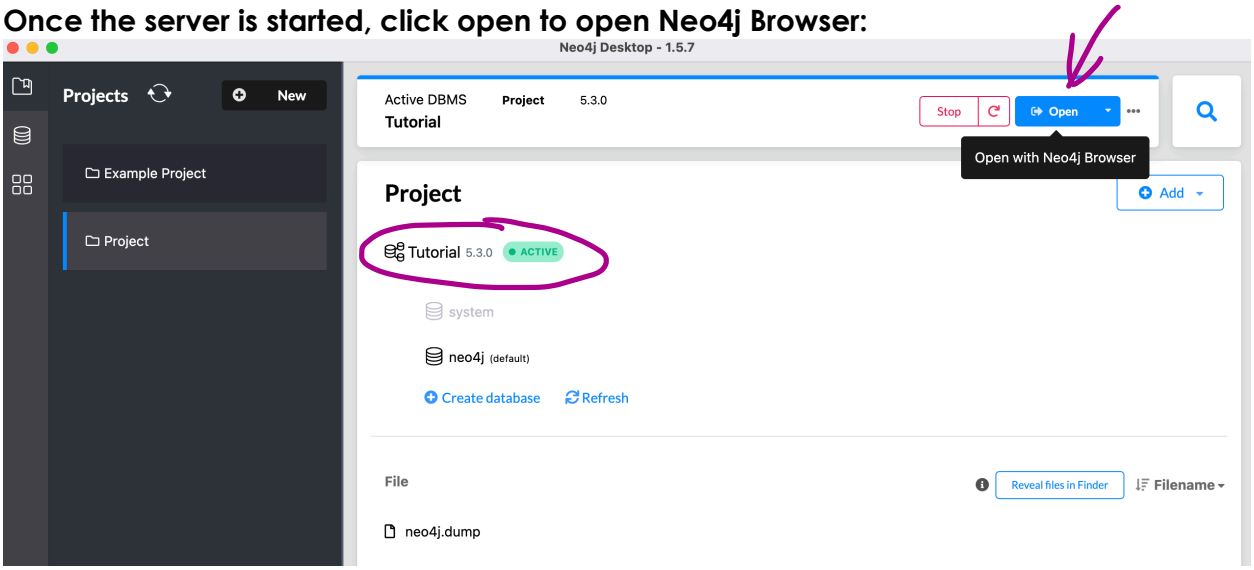
Enter the Name of the Project and a password and click create:



Once the Project is created, Click Start :



Once the server is started, click open to open Neo4j Browser:

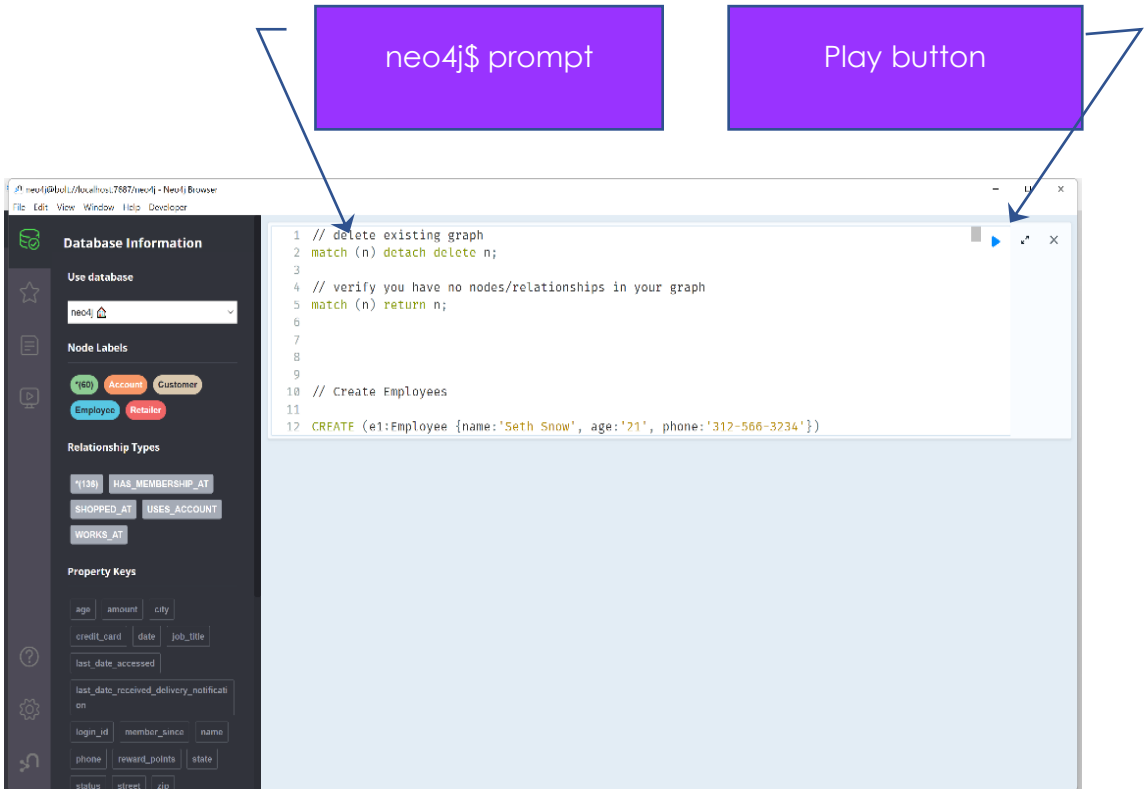


Instructions and Deliverable Requirements:

Before you start, make sure you bookmark the URL for Neo4j Cypher manual:

The Neo4j Cypher Manual v4.1
https://neo4j.com/docs/cypher-manual/current/

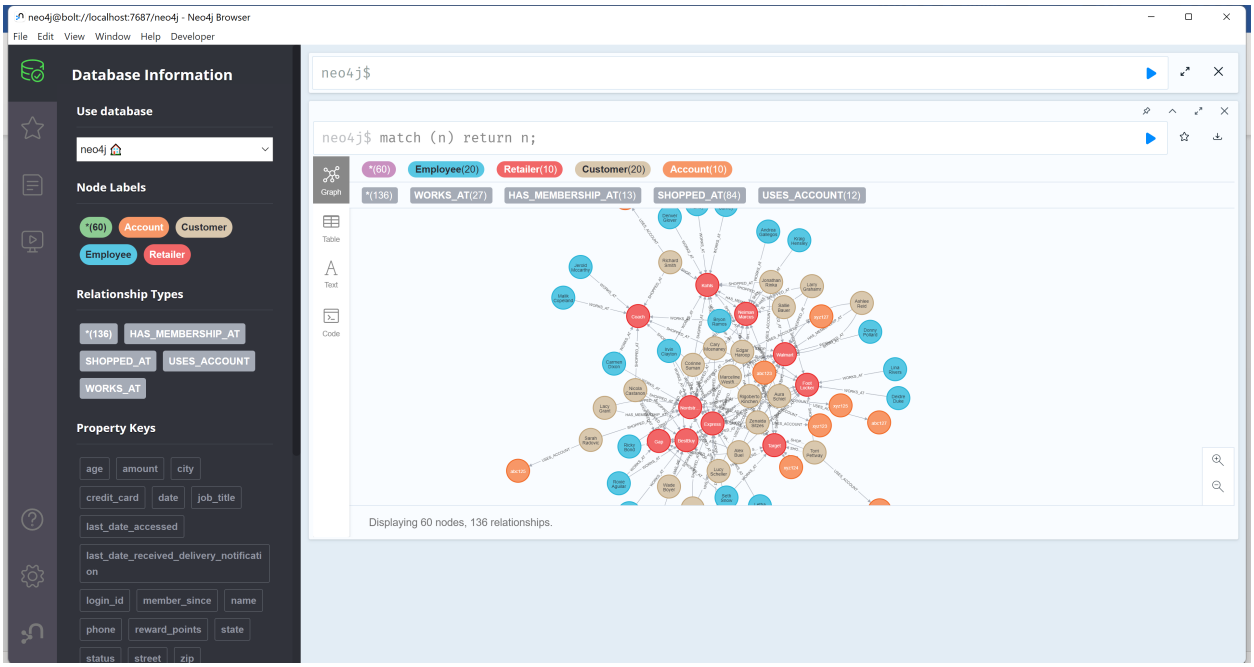
1. Use the provided Cypher script to create the graph database
   - You could use any names for your project and the graph database
   - Copy the **ENTIRE** Cypher code in the script and paste it in **neo4j$** prompt and then click the play button on the right. This will create your database. Now you can perform query on it.
     (**DO NOT copy and paste one line at a time**)



1. Execute the given Cypher code to get all the nodes of the database.

```
MATCH  (n)
RETURN (n);
```

The following screen-shot shows the complete graph database after you run your above code.



2. Execute the following Cypher code to get the list of retailers:

```
MATCH  (r:Retailer)
RETURN (r);
```

3. Execute the following Cypher code to get the list of employees:

```
MATCH  (e:Employee)
RETURN (e);
```

4. Execute the following Cypher code to get the list of customers:

```
MATCH  (c:Customer)
RETURN (c);
```

5. Execute the following Cypher code to get the list of all Disputed transactions

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed"

RETURN customer.name AS `Customer Name`, retailer.name AS `Retailer Name`,
transaction.amount AS `Transaction Amount`, transaction.date AS `Transaction date`

ORDER BY `Transaction date` DESC
```

6. Get a list of Approved transactions occurred before the disputed transaction

```
MATCH (customer:Customer)-[transaction1:SHOPPED_AT]->(retailer1)

WHERE transaction1.status = "Disputed"

MATCH (customer)-[transaction2:SHOPPED_AT]->(retailer2)

WHERE transaction2.status = "Approved" AND transaction2.date <
transaction1.date

WITH customer, retailer2, transaction2 ORDER BY transaction2.date DESC

RETURN customer.name AS `Customer Name`, retailer2.name AS `Retailer
Name`, transaction2.amount AS Amount, transaction2.date AS
`Transaction date`

ORDER BY `Transaction date` DESC
```

7. Get the number of times we see each Retailer in disputed transactions

```
MATCH (customer:Customer)-[transaction1:SHOPPED_AT]->(retailer1)

WHERE transaction1.status = "Disputed"

MATCH (customer)-[transaction2:SHOPPED_AT]->(retailer2)

WHERE transaction2.status = "Approved" AND transaction2.date <
transaction1.date

WITH customer, retailer2, transaction2 ORDER BY transaction2.date DESC

RETURN DISTINCT retailer2.name AS `Retailer`, count(DISTINCT transaction2)
AS Count, collect(DISTINCT customer.name) AS Customer

ORDER BY Count DESC
```

8. Get the number of disputed transactions for every store

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)
WHERE transaction.status = "Disputed"
WITH customer, retailer, transaction
RETURN DISTINCT retailer.name AS `Retailer`, count(DISTINCT transaction)
AS Count, collect(DISTINCT customer.name) AS Customer
ORDER BY Count DESC
```

9. Get the number of disputed transactions and the list of customer names for these disputed transactions for EVERY STORE

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)
WHERE transaction.status = "Disputed"
WITH customer, retailer,  transaction
RETURN retailer.name AS `Fraud Store`,  count(transaction) AS Count,
collect(customer.name) AS Customers
ORDER BY Count DESC
```

10. Get the number of disputed transactions for EVERY CUSTOMER that has more that one disputed transaction

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)
WHERE transaction.status = "Disputed"
WITH customer, count(*) AS number_of_disputed_transactions
WHERE number_of_disputed_transactions > 0
RETURN customer.name AS Customer, number_of_disputed_transactions
```

11. Get the top 3 customers that reported the maximum number of disputed transactions

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)
WHERE transaction.status = "Disputed"
WITH customer, count(*) AS number_of_disputed_transactions
WHERE number_of_disputed_transactions > 0
```

```
WITH customer,  number_of_disputed_transactions

ORDER BY number_of_disputed_transactions DESC LIMIT 3

RETURN customer.name AS Customer, number_of_disputed_transactions
```

12. Get the list of stores on LaSalle street that have disputed tranasctions and the number of disputed transactions for every store

```
// The store list must be sorted by store name in ascending order.

MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed" AND retailer.street =~ '.* LaSalle.*'

WITH retailer, count(*) AS number_of_disputed_transactions

WHERE number_of_disputed_transactions > 0

WITH retailer, number_of_disputed_transactions

ORDER BY retailer.name ASC

RETURN retailer.name AS Retailer , number_of_disputed_transactions
```

13. Get the list of customers and stores that have disputed transactions where Jonathan reported a disputed transaction

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed" AND customer.name = "Jonathan Rinka"

MATCH (customer2)-[transaction2:SHOPPED_AT]->(retailer2)

WHERE transaction2.status = "Disputed" AND retailer.name = retailer2.name AND customer2.name <> "Jonathan Rinka"

WITH customer, customer2, retailer, retailer2

RETURN customer.name AS `Customer`, customer2.name AS `Other Customer`, retailer2.name AS `Retailer`
```

14. Get the list of customers who have MEMBERSHIP in those stores but have disputed transactions.

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed"

MATCH (customer)-[membership:HAS_MEMBERSHIP_AT]->(retailer)
```

```
WITH customer, membership, retailer

RETURN customer.name AS `Customer Name`, membership.reward_points
AS `Reward Points`, retailer.name AS `Retailer Name`
```

15. Get the list of customers who have online accounts and have disputed transaction

```
MATCH (customer:Customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed"

MATCH (customer)-[used_account:USES_ACCOUNT]->(account)

WITH customer, account,  used_account, retailer, transaction

RETURN customer.name AS `Customer`, account.login_id AS `Account
Used`, used_account.last_date_accessed AS `Account Accesed On`,
retailer.name AS `Retailer Name`, transaction.date AS `Disputed
Transaction Date`
```

16. Get the list of Employees who work in at least 2 stores where disputed transactions reported these stores

```
MATCH (customer)-[transaction:SHOPPED_AT]->(retailer)

WHERE transaction.status = "Disputed"

MATCH (employee1:Employee)-[:WORKS_AT]->(retailer1)

WHERE retailer.name = retailer1.name

MATCH (employee2:Employee)-[:WORKS_AT]->(retailer2)

WHERE  employee1.name = employee2.name AND retailer1.name <>
retailer2.name

WITH employee1, retailer1, retailer2, customer, retailer

Return employee1.name AS `Employee`, retailer1.name AS `Retailer1`,
collect(DISTINCT retailer2.name)    AS `Retailer2`, customer.name,
retailer.name
```