



NEO4J

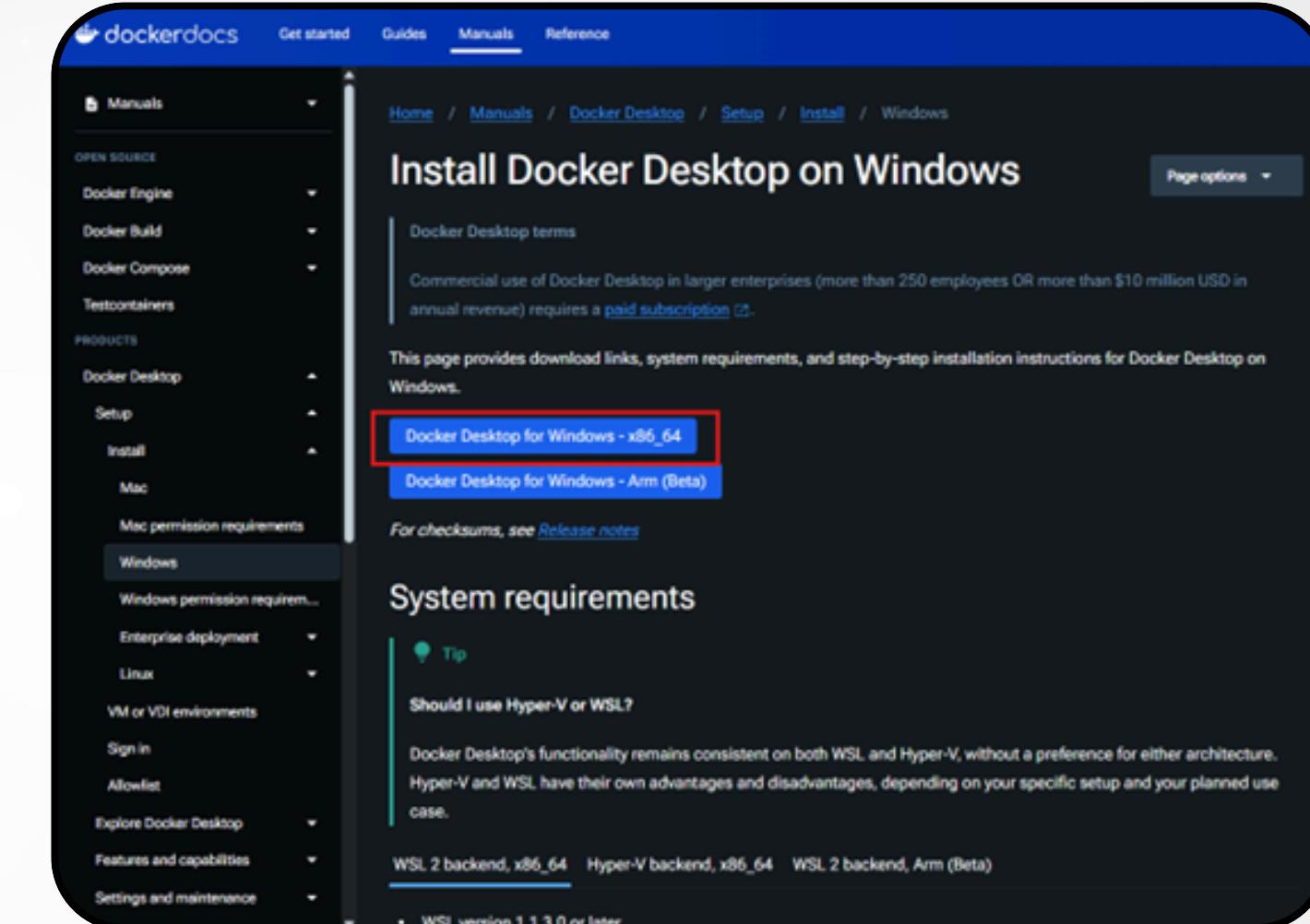
GRAPH DATABASE

นำเสนօอาจารย์ : ดร.วรากรณ์ วิyanabhak

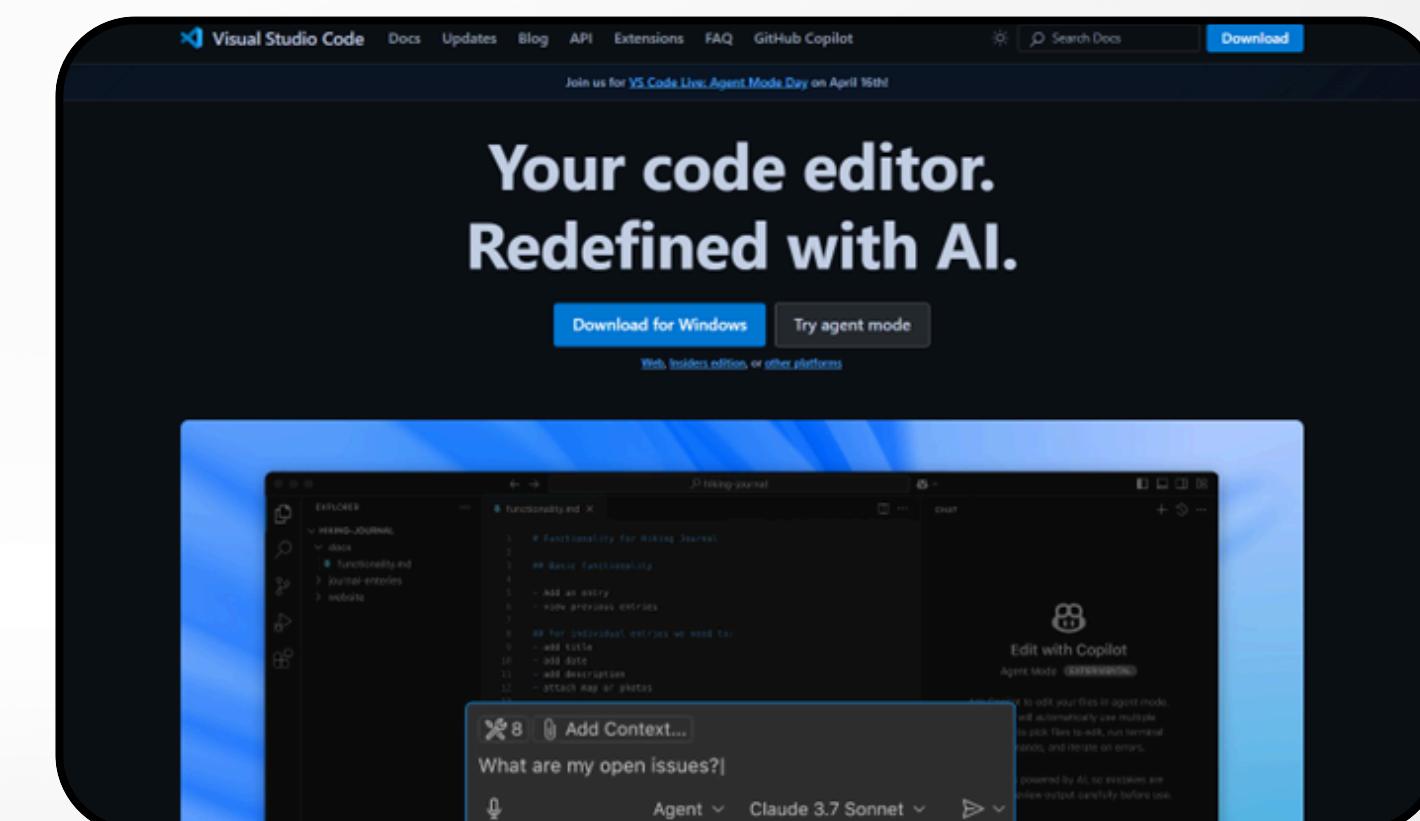
ขั้นตอนการติดตั้ง Neo4j Database

1

ติดตั้ง Docker desktop
<https://www.docker.com/products/docker-desktop/>



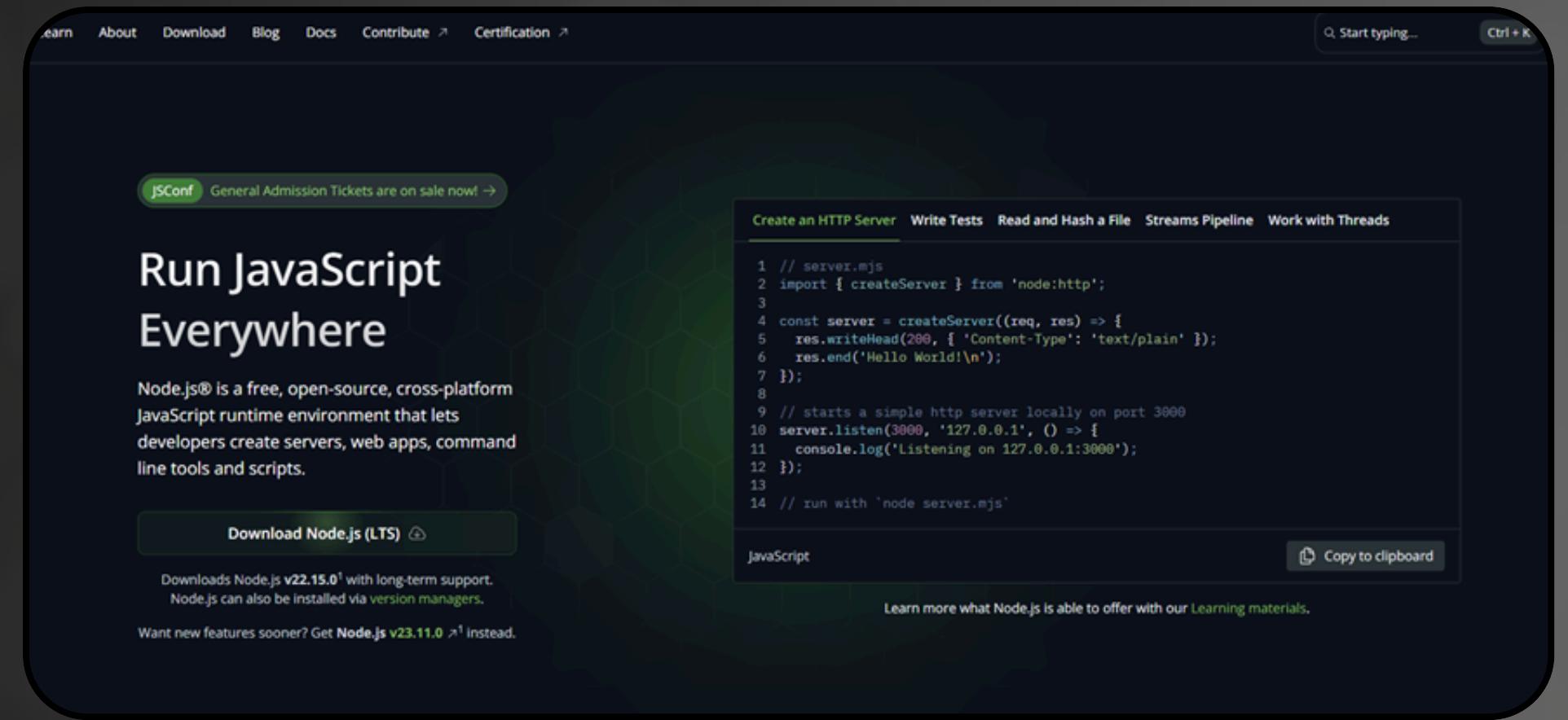
The screenshot shows the 'Install Docker Desktop on Windows' page from dockerdocs.org. The left sidebar has sections for OPEN SOURCE (Docker Engine, Docker Build, Docker Compose, Testcontainers) and PRODUCTS (Docker Desktop, Setup, Install, Mac, Mac permission requirements, Windows, Windows permission requirements, Enterprise deployment, Linux, VM or VDI environments, Sign in, Allowlist, Explore Docker Desktop, Features and capabilities, Settings and maintenance). The main content area has a heading 'Install Docker Desktop on Windows'. It includes a note about commercial use, download links, system requirements, and step-by-step installation instructions. A 'Docker Desktop for Windows - x86_64' button is highlighted with a red box.



The screenshot shows the Visual Studio Code landing page with the heading 'Your code editor. Redefined with AI.' and download links for Windows and agent mode. Below it is a screenshot of the VS Code interface with an AI-powered code completion suggestion 'Edit with Copilot' and an 'Agent Mode' status bar.

2

ติดตั้ง Visual Studio Code
<https://code.visualstudio.com/>



```
services:  
  neo4j:  
    image: neo4j:latest  
    container_name: neo4j-container  
    volumes:  
      - ./neo4j_database/neo4j_data:/data  
      - ./neo4j_database/neo4j_logs:/logs  
      - ./neo4j_database/neo4j_plugins:/plugins  
    environment:  
      - NEO4J_AUTH=neo4j/Neo4j12345*  
  ports:  
    - "7474:7474"  
    - "7687:7687"  
  restart: always
```

3

ติดตั้ง Node.js
<https://nodejs.org/en>

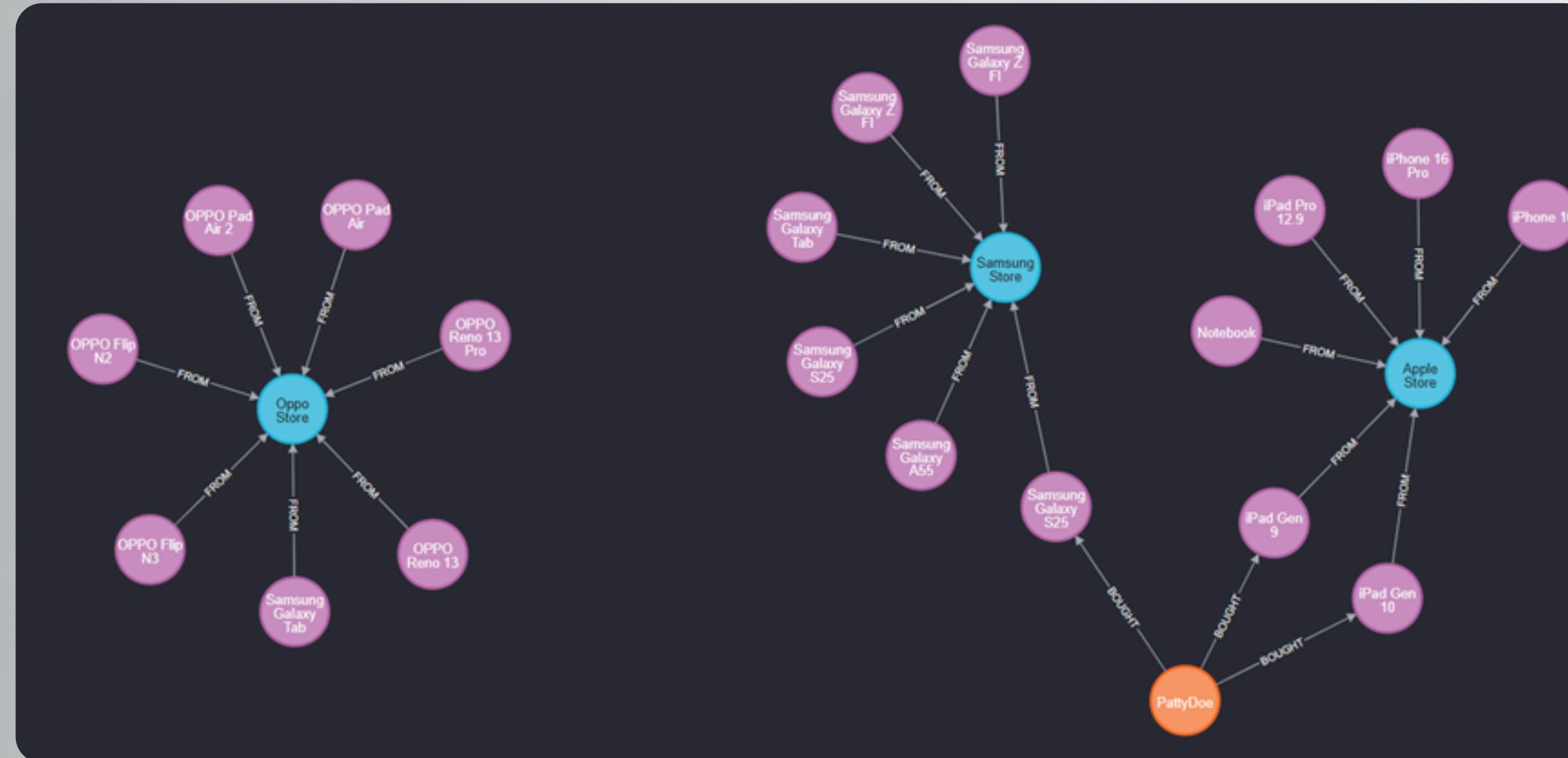
4

ติดตั้งระบบฐานข้อมูลลงบนเครื่อง โดย
ใช้ Docker Desktop และ Visual
Studio Code และกำหนดค่าเริ่มต้นใน
docker-compose.yml

ขั้นตอนการติดตั้ง Neo4j Database

CRUD OPERATION

ລັກເມນະຂອງ Database



ແປ່ງເປັນ **CRUD** ສໍາຮັບ **NODE** ແລະ **RELATIONSHIP (EDGE)**

CRUD OPERATION

การกำเบี้องตื้นด้วย **Neo4j Driver** จาก **Node.js** และ **Postman**

สร้างไฟล์ชื่อ `server.js` และเขียนโค้ดด้านล่างนี้

```
const express = require('express');
const bodyParser = require('body-parser');
const neo4j = require('neo4j-driver');

const app = express();
app.use(bodyParser.json());

const driver = neo4j.driver(
  'bolt://localhost:your_port',
  neo4j.auth.basic('your_username', 'your_password')
);
const session = driver.session();

app.get('/', async (req, res) => {
  try {
    await session.run('RETURN 1');
    res.send('Connected to Neo4j successfully!');
  } catch (error) {
    res.status(500).send(`Failed to connect to Neo4j: ${error.message}`);
  }
});

const PORT = 3000;
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

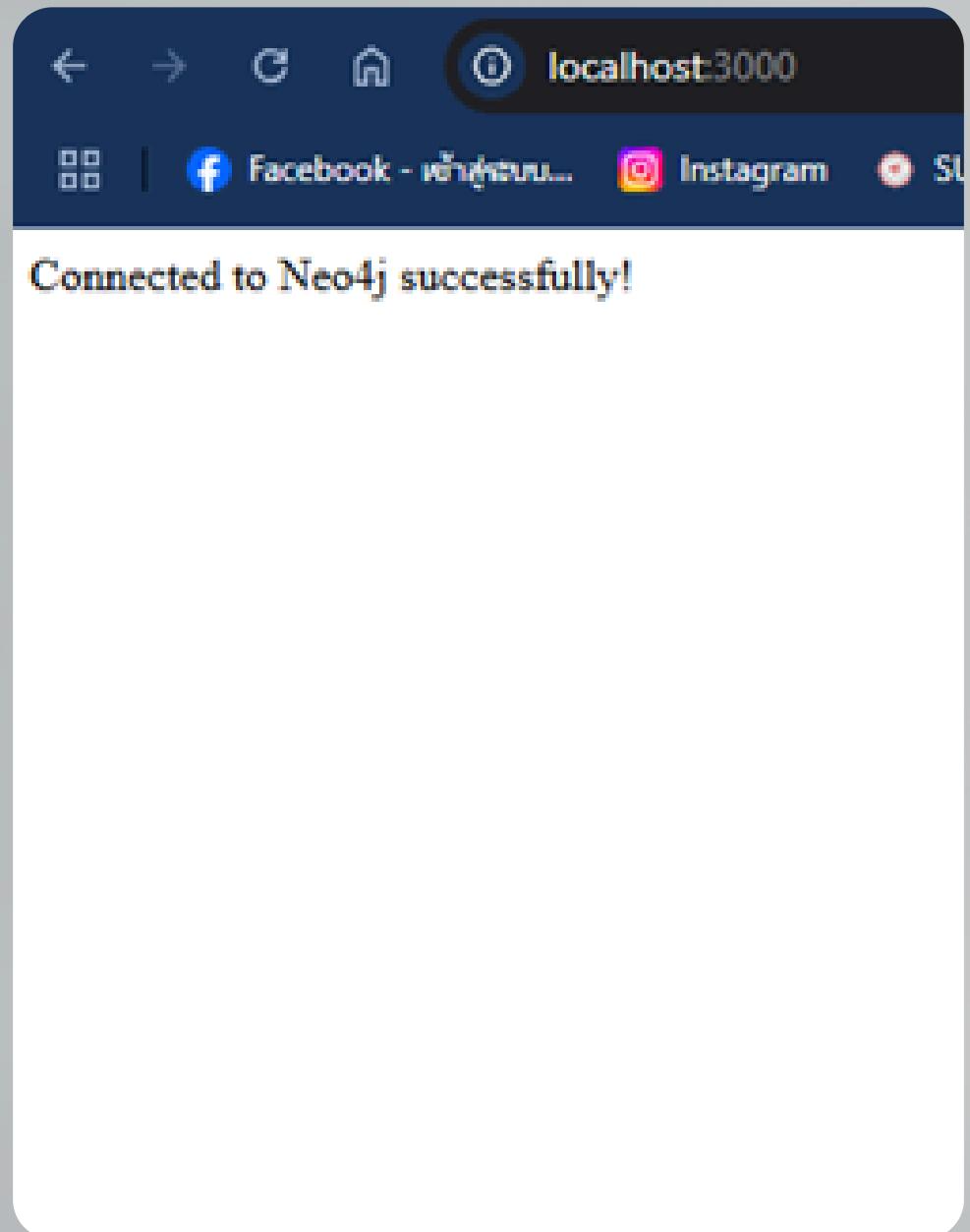
Package ที่จำเป็น

```
npm init
npm install express neo4j-driver nodemon body-parser
```

CRUD OPERATION

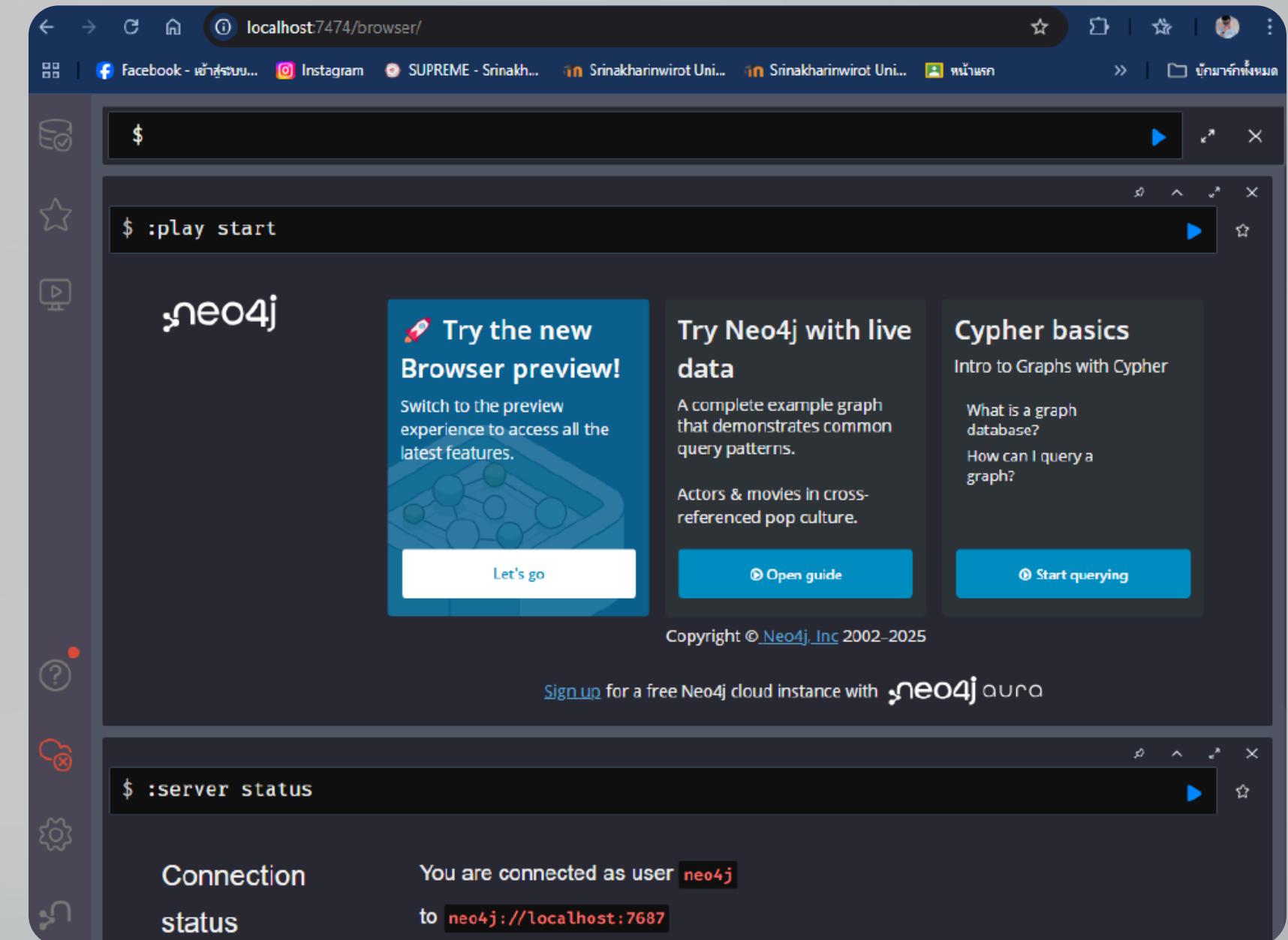
การกำเบี้องตื้นด้วย **Neo4j Driver** จาก **Node.js** และ **Postman**

`nodemon server.js`



localhost:3000

`docker-compose up -d`



localhost:7474/browser/

CRUD OPERATION

การกำเบี้องตื้นด้วย **Neo4j Driver** จาก **Node.js** และ **Postman** จัดการ **NODE** เก่าบ้าน

CREATE

```
app.post('/nodes', async (req, res) => {
  const { label, properties } = req.body;
  try {
    const result = await session.run(
      `CREATE (n:${label} ${props}) RETURN n`,
      { props: properties }
    );
    res.json(result.records[0].get('n').properties);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

UPDATE

```
app.put('/nodes/:label/:id', async (req, res) => {
  const { label, id } = req.params;
  const { properties } = req.body;
  try {
    const result = await session.run(
      `MATCH (n:${label} {id: $id})
        SET n += $props
        RETURN n`,
      { id, props: properties }
    );
    res.json(result.records[0].get('n').properties);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

READ

```
app.get('/nodes/:label', async (req, res) => {
  const label = req.params.label;
  try {
    const result = await session.run(`MATCH
(n:${label}) RETURN n`);
    const nodes = result.records.map(record =>
record.get('n').properties);
    res.json(nodes);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

DELETE

```
app.delete('/nodes/:label/:id', async (req, res) => {
  const { label, id } = req.params;
  try {
    await session.run(
      `MATCH (n:${label} {id: $id}) DETACH DELETE n`,
      { id }
    );
    res.json({ message: 'Node deleted' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

CRUD OPERATION

CREATE NODE ผ่าน Neo4jBrowser



The screenshot shows the Neo4j Browser interface. In the main window, a Cypher query is being run:

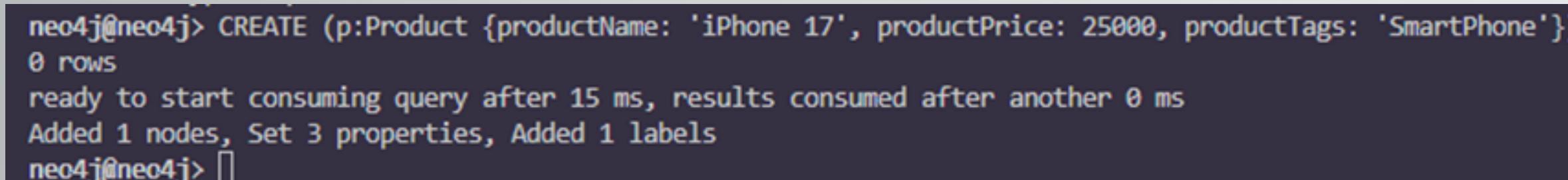
```
neo4j$ CREATE (p:Product {productName: 'iPhone 15', productPrice: 15000, productTags: 'SmartPhone'});
```

Below the query, the results are displayed:

Added 1 label, created 1 node, set 3 properties, completed after 11 ms.

The sidebar on the left has two tabs: "Table" (selected) and "Code".

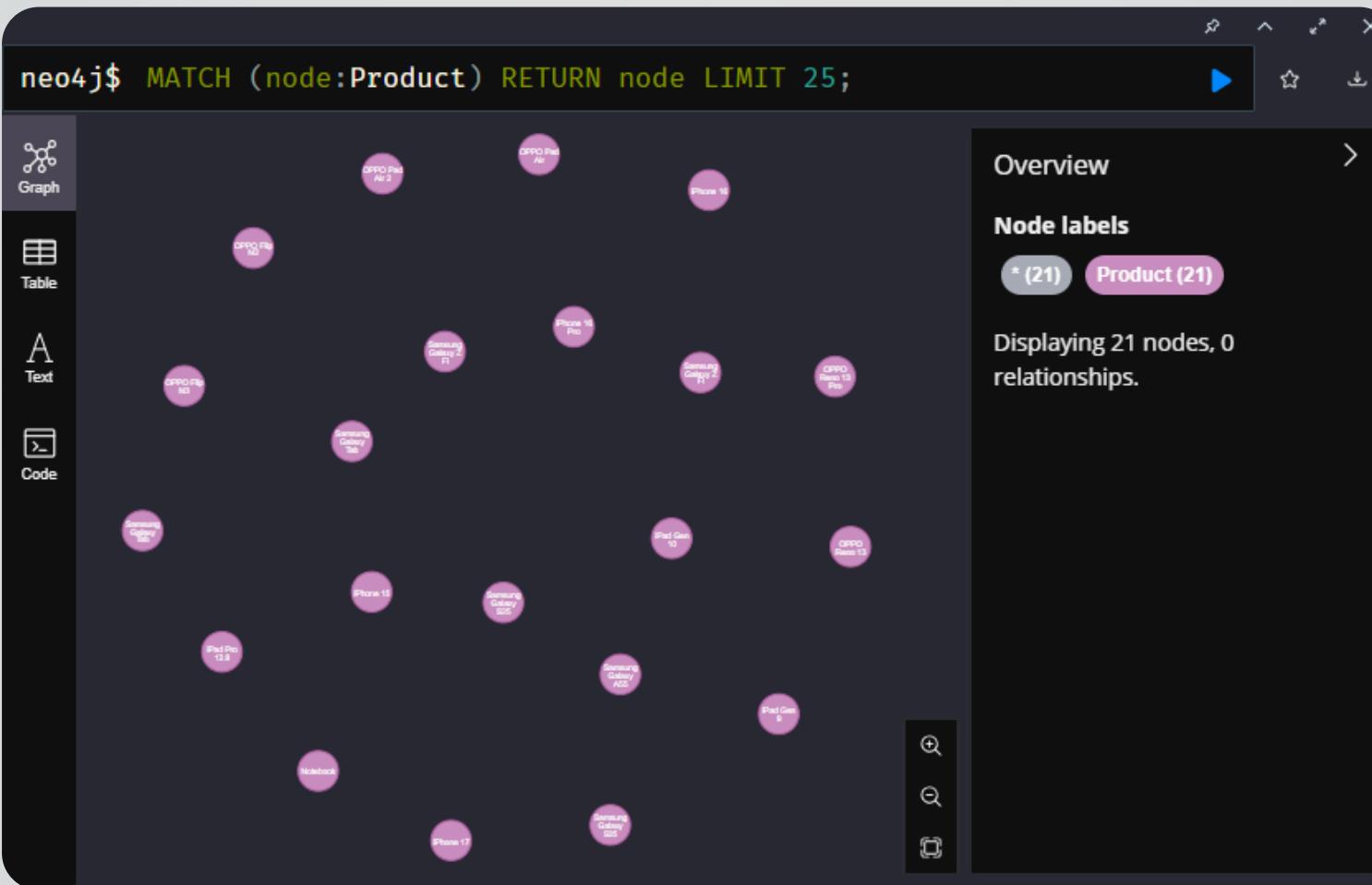
CREATE NODE ผ่าน Neo4j Cypher-Shell บน Docker



```
neo4j@neo4j:~> CREATE (p:Product {productName: 'iPhone 17', productPrice: 25000, productTags: 'SmartPhone'});  
0 rows  
ready to start consuming query after 15 ms, results consumed after another 0 ms  
Added 1 nodes, Set 3 properties, Added 1 labels  
neo4j@neo4j:> █
```

CRUD OPERATION

READ NODE ผ่าน Neo4jBrowser

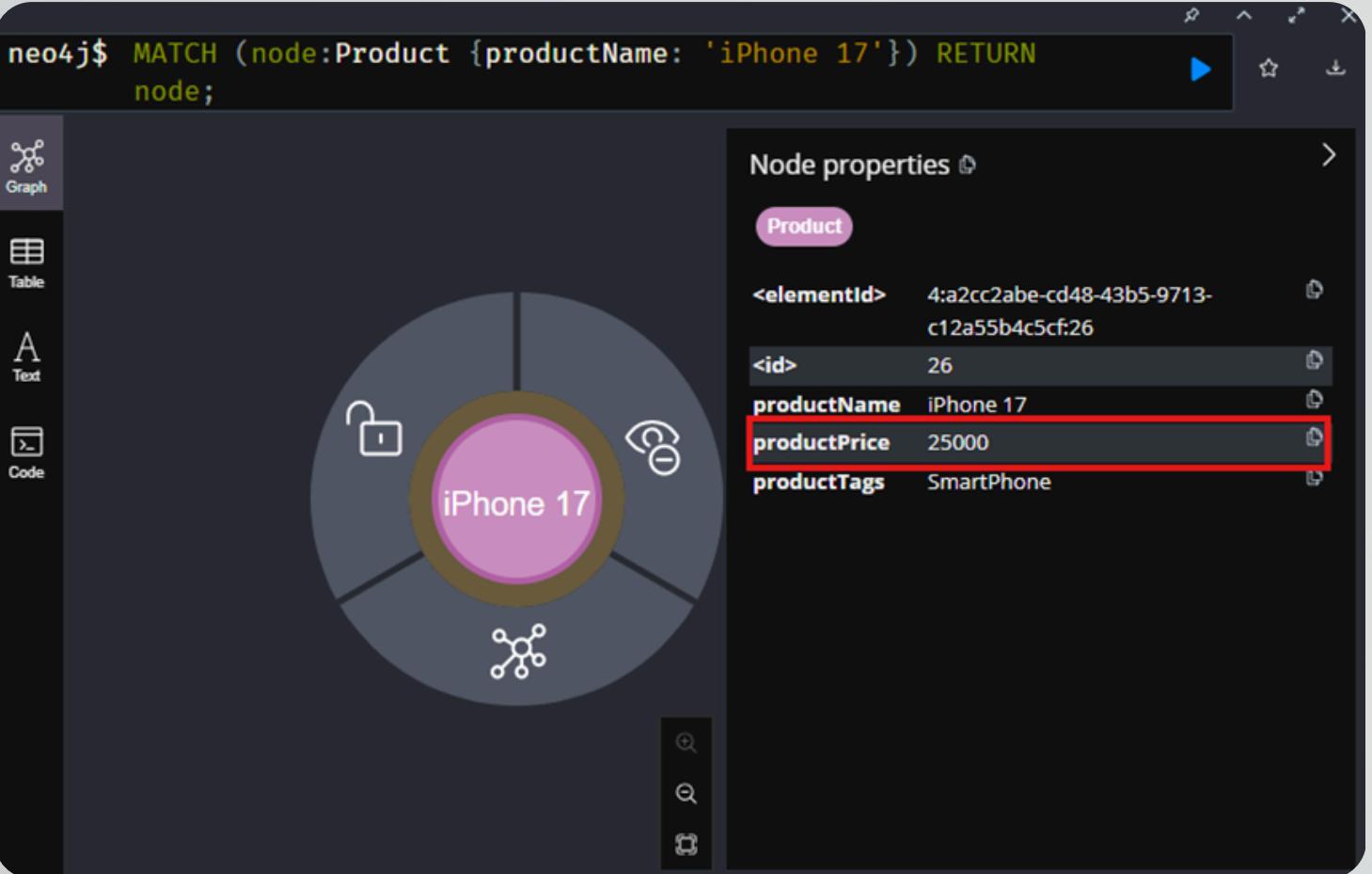


READ NODE ผ่าน Neo4j Cypher-Shell บน Docker

```
neo4j@neo4j:~$ MATCH (node:Shop) RETURN node LIMIT 25;
+-----+
| node
+-----+
| (:Shop {shopName: "Samsung Store", shopPhone: "0998881248", id: "1745643181263", shopAddress: "114 Sukhumvit 23, Bangkok 10110, Thailand."}) |
| (:Shop {shopName: "Apple Store", shopPhone: "0998887777", id: "1745643212838", shopAddress: "789 Green Lane, Eco Town"}) |
| (:Shop {shopName: "Oppo Store", shopPhone: "0998887755", id: "1745643228818", shopAddress: "123 Main Street, Cityville"}) |
| (:Shop {shopName: "ShuJShop", shopPhone: "0999999999", id: "1745648165481", shopAddress: "114 Sukhumvit 23, Bangkok 10110, Thailand."}) |
+-----+
4 rows
ready to start consuming query after 43 ms, results consumed after another 1 ms
neo4j@neo4j:~$
```

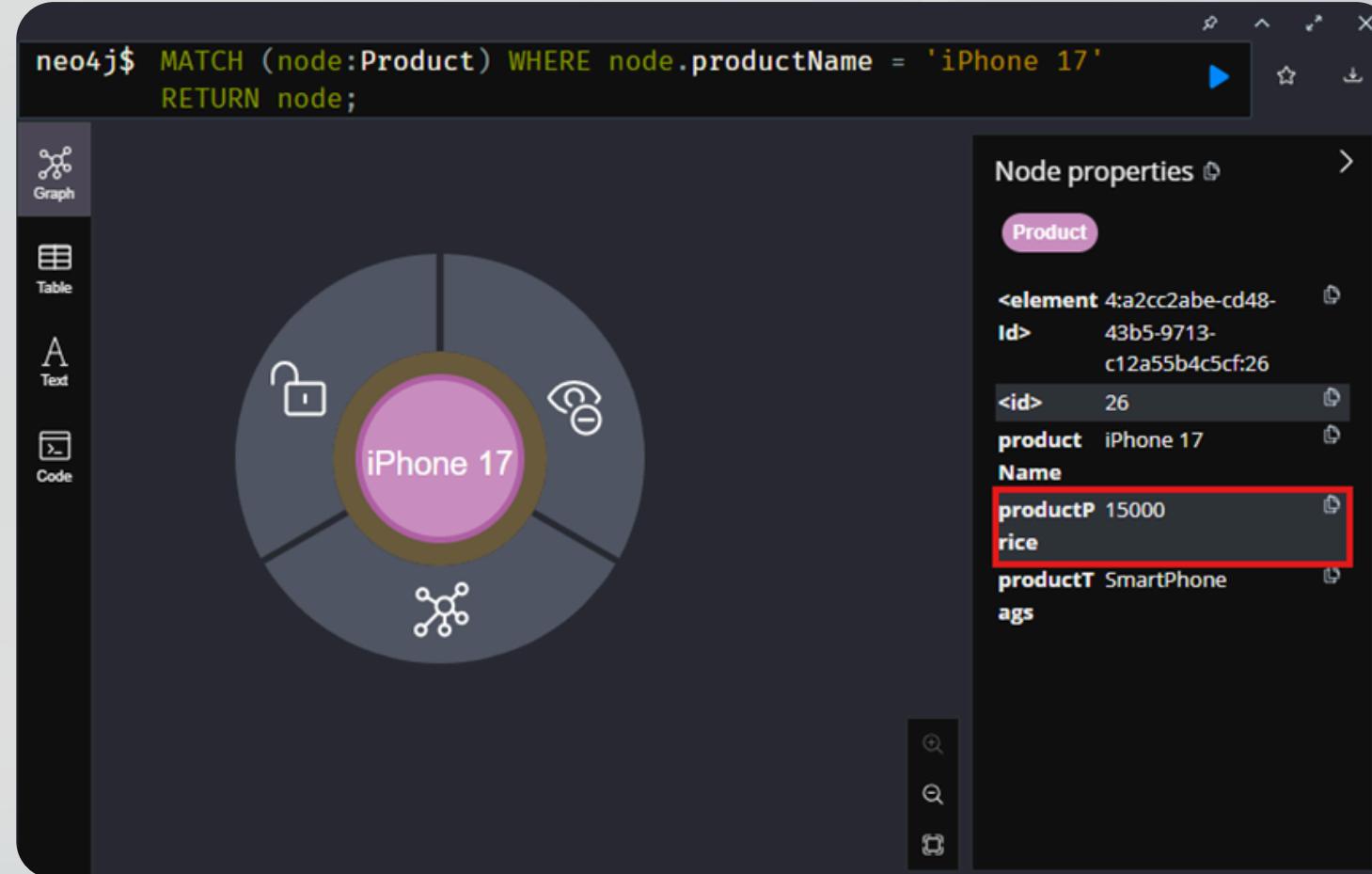
CRUD OPERATION

UPDATE NODE ผ่าน Neo4jBrowser



```
1 MATCH (node:Product {productName: 'iPhone 17'})  
2 SET node.productPrice = 15000;
```

Set 1 property, completed after 62 ms.



```
neo4j$ MATCH (node:Product) WHERE node.productName = 'iPhone 17'  
RETURN node;
```

BEFORE

AFTER

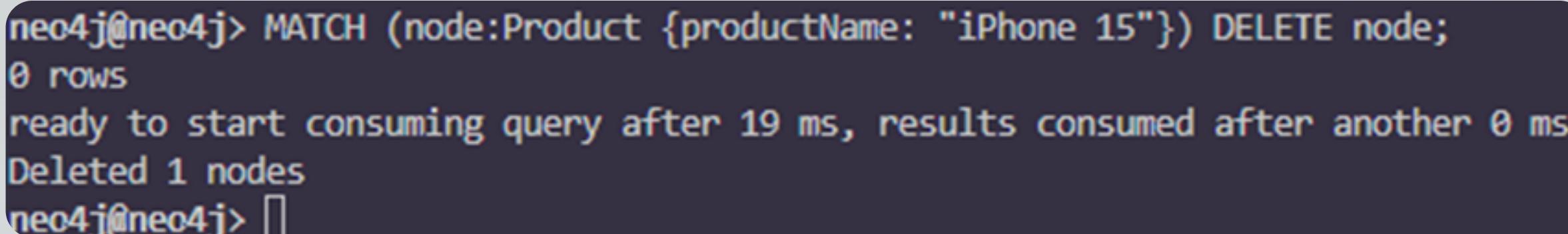
CRUD OPERATION

DELETE NODE ជាមួយ Neo4jBrowser



A screenshot of the Neo4j Browser interface. The query entered is: `neo4j$ MATCH (node:Product {productName: "iPhone 17"}) DELETE node;`. Below the query, the result is displayed: `Deleted 1 node, completed after 22 ms.`. A small icon labeled "Table" is visible on the left side of the results area.

DELETE NODE ជាមួយ Neo4j Cypher-Shell ឬ Docker



```
neo4j@neo4j:~> MATCH (node:Product {productName: "iPhone 15"}) DELETE node;
0 rows
ready to start consuming query after 19 ms, results consumed after another 0 ms
Deleted 1 nodes
neo4j@neo4j:> []
```

CRUD OPERATION

CREATE RELATIONSHIP ผ่าน Neo4jBrowser

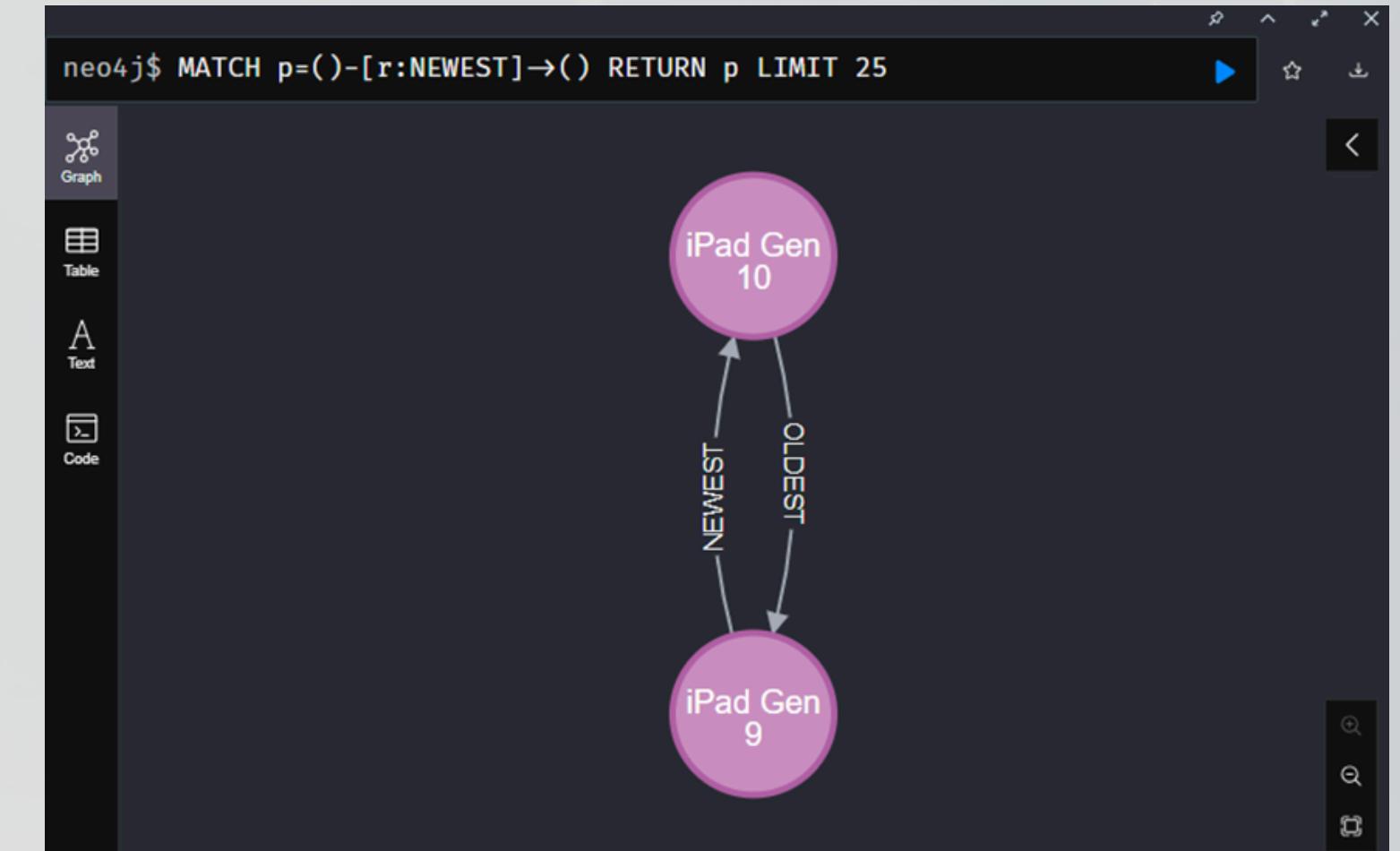
สร้าง **RELATIONSHIP** จาก **NODE** แรกไปยัง **NODE** กี่สอง

```
1 MATCH (p1:Product {productName: 'iPad Gen 9'})  
2 MATCH (p2:Product {productName: 'iPad Gen 10'})  
3 MERGE (p1)-[:NEWEST]→(p2);  
  
Created 1 relationship, completed after 19 ms.
```

สร้าง **RELATIONSHIP** จาก **NODE** กี่สองกลับมายัง **NODE** แรก

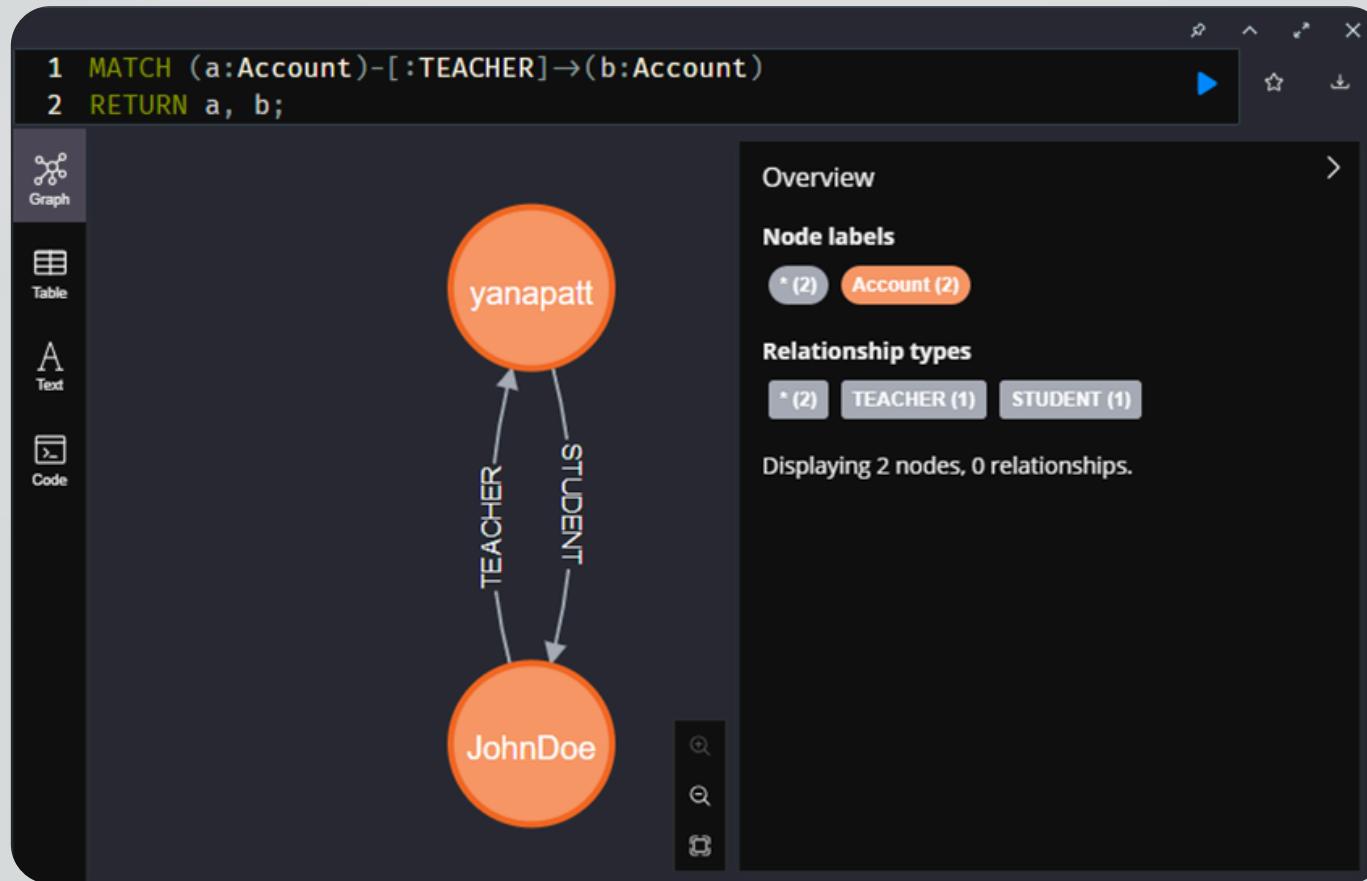
```
1 MATCH (p1:Product {productName: 'iPad Gen 9'})  
2 MATCH (p2:Product {productName: 'iPad Gen 10'})  
3 MERGE (p2)-[:OLDEST]→(p1);  
  
Created 1 relationship, completed after 68 ms.
```

ผลลัพธ์



CRUD OPERATION

READ ក្នុង NODE កំណត់ RELATIONSHIP ដោយ Neo4jBrowser



READ ក្នុង NODE កំណត់ RELATIONSHIP ដោយ Neo4j Cypher-Shell ឬ Docker

```
neo4j@neo4j:~$ MATCH (a:Account)-[:TEACHER]-(b:Account)
neo4j@neo4j:~$ RETURN a, b;
+-----+-----+
| a | b |
+-----+-----+
| (:Account {password: "123", role: "User", id: "1745670144931", email: "john.doe@example.com", username: "JohnDoe"}) | (:Account {password: "123", role: "User", id: "1745670138311", email: "yanapatt@example.com", username: "yanapatt"}) |
+-----+-----+
1 row
```

The screenshot shows the output of a Cypher query in the Neo4j Cypher-Shell. The query matches nodes 'a' and 'b' where 'a' is labeled 'Account' and has a 'TEACHER' relationship to 'b'. The results show two rows of data, each representing a node with its properties: password, role, id, email, and username. The first node is 'JohnDoe' and the second is 'yanapatt'. Both nodes have a 'User' role and a password of '123'. The 'Cypher' tab at the bottom of the shell shows the executed query.

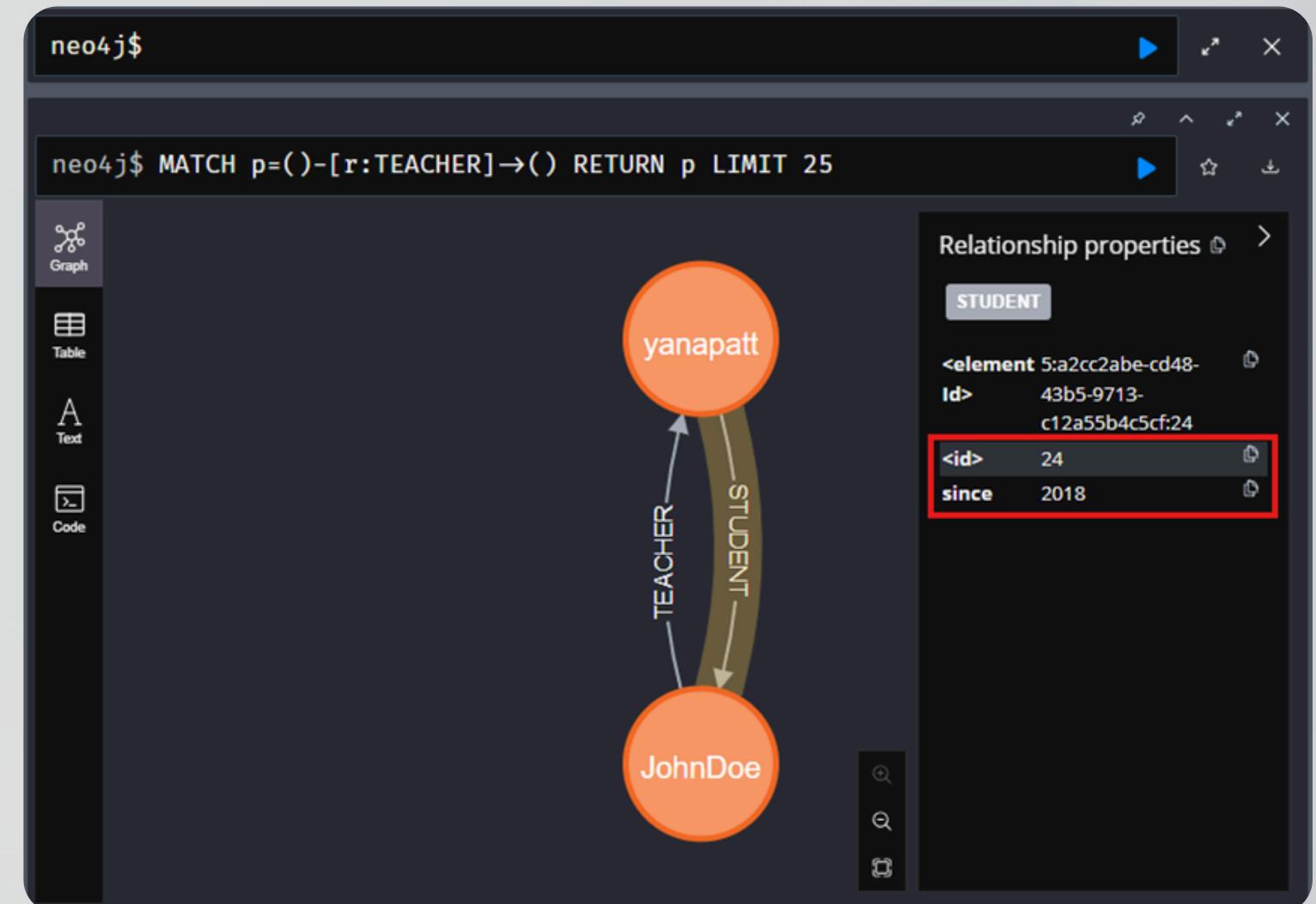
CRUD OPERATION

UPDATE PROPERTY ของ **RELATIONSHIP** ผ่าน **Neo4jBrowser**

PROPERTY ของ **RELATIONSHIP** จาก **NODE** แรกไป
ยัง **NODE** กี่สอง

ผลลัพธ์

```
neo4j$  
  
1 MATCH (a1:Account)-[r:STUDENT]->(a2:Account)  
2 WHERE a1.username = 'yanapatt' AND a2.username = 'JohnDoe'  
3 SET r.since = 2018  
4 RETURN r;  
  
Table  
A Text  
Code  
  
MAX COLUMN WIDTH: 
```



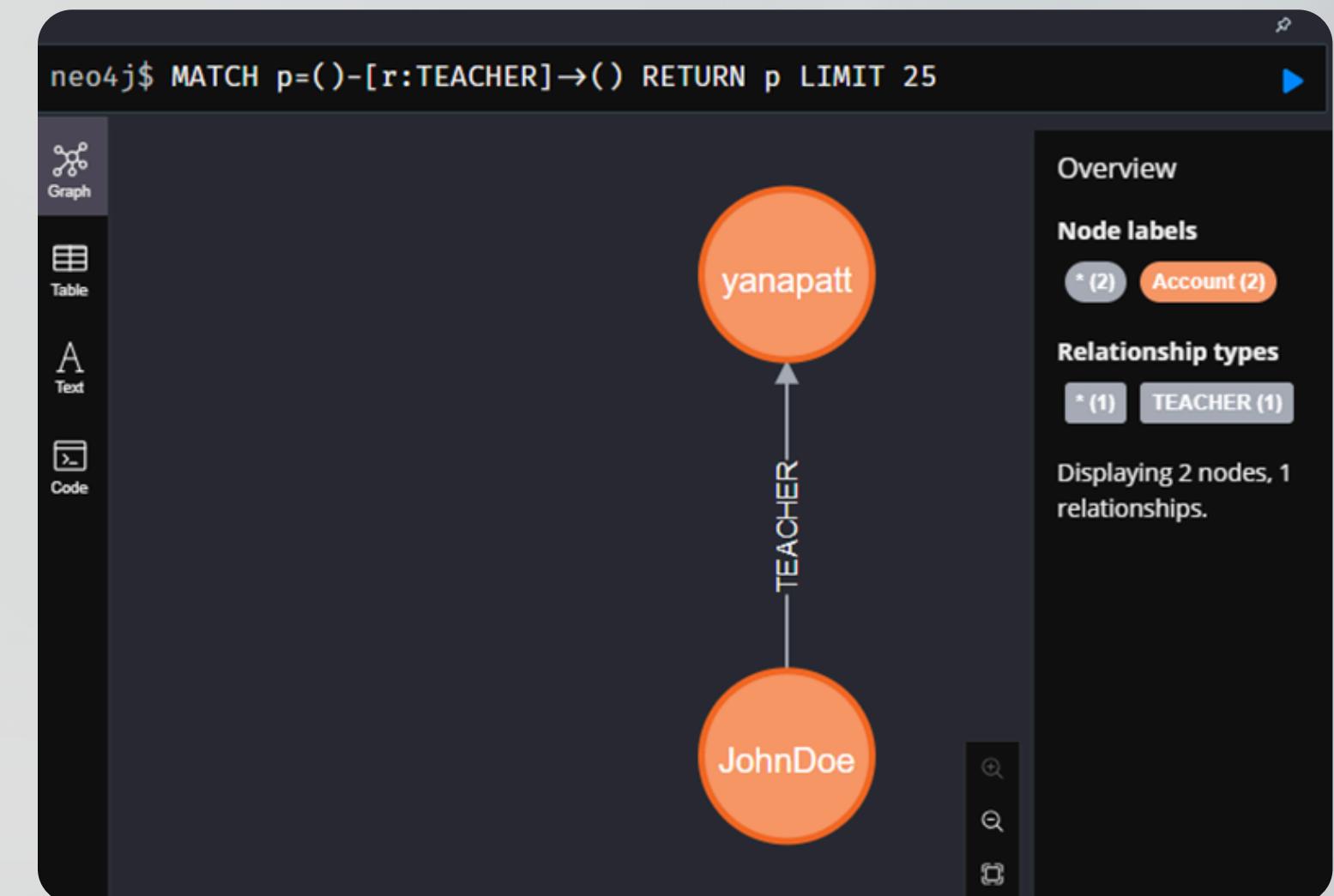
CRUD OPERATION

DELETE RELATIONSHIP នៅវាទេរកគ្នា **NODE** កីត្តិភាព ដោយ **Neo4jBrowser**

DELETE RELATION ចាប់ពី **NODE** នៅវាទេរកគ្នា **NODE** កីត្តិភាព

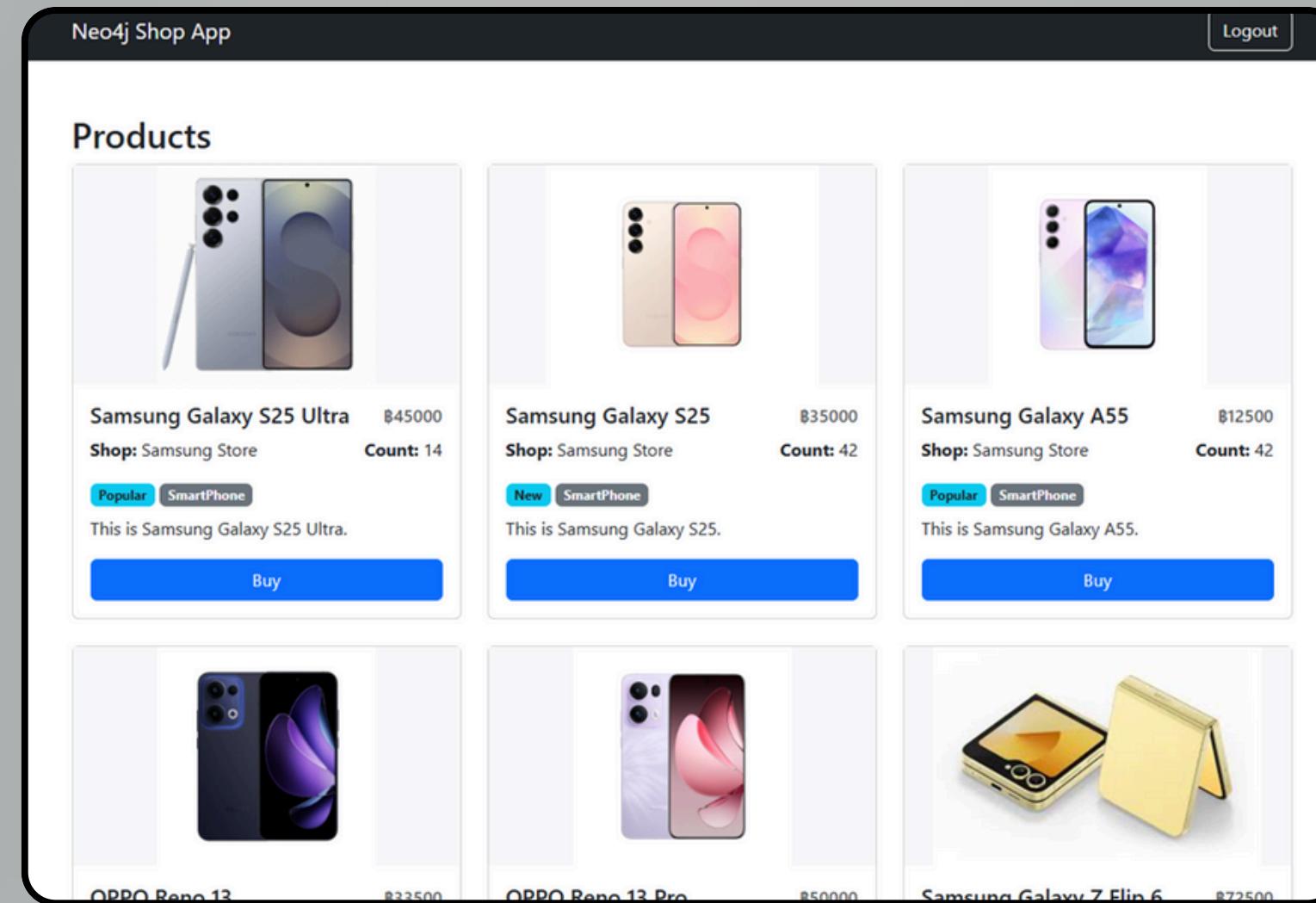
លទ្ធផល

```
neo4j$  
  
1 MATCH (a)-[r:STUDENT]->(b)  
2 DELETE r;  
  
Deleted 1 relationship, completed after 8 ms.
```



USE CASE

WEBSITE แนะนำสินค้า



แนะนำสินค้าที่มีหมวดหมู่เดียวกัน ที่ผู้ใช้คนนั้นซื้อเยอะที่สุด
แนะนำสินค้าที่มาจากร้านค้าเดียวกัน ที่ผู้ใช้คนนั้นซื้อเยอะที่สุด

จัดทำโดย

นายอิรักช์ อุ่คงคา

66102010233

นายษานกัตร ปานเกษม

66102010236

นายรัชดาสตร์ จันทร์

66102010244



THANK YOU