**Aayush Shrestha (UoN ID: 21422064)**

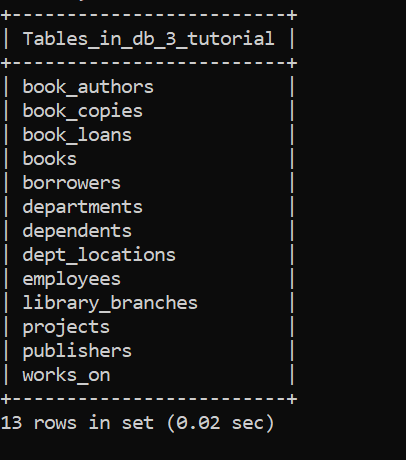
**PORTFOLIO**

**Week 1**

* In week 1, I revised some basics of SQL.
* Performed some basics queries.
* Done some exercises on JOIN, GROUP BY, EXISTS, WHERE.
* Revised on the usage of nested queries.
* Revised some previous years slides of database.

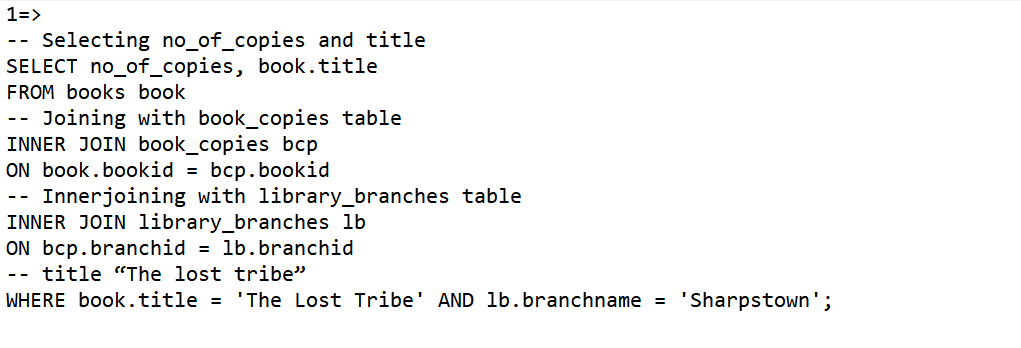
**Appendix**

Following tables will be created:

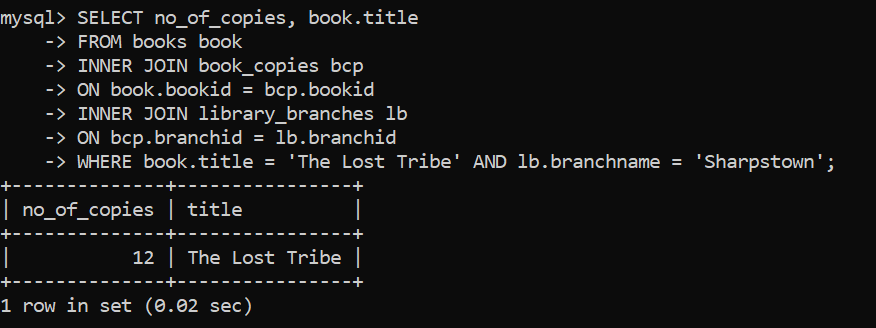


**Solution:**

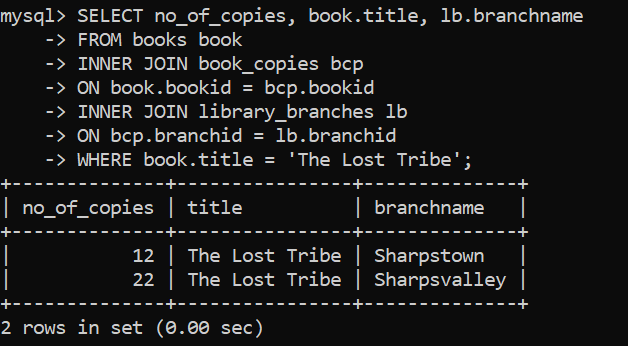
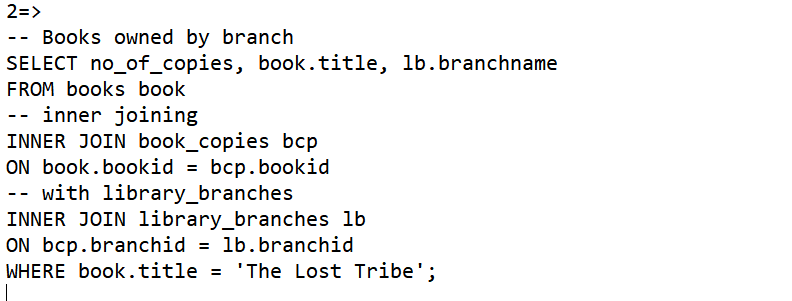
1. “How many copies of the book titled The Lost Tribe are owned by the library branch whose name is "Sharpstown"?”



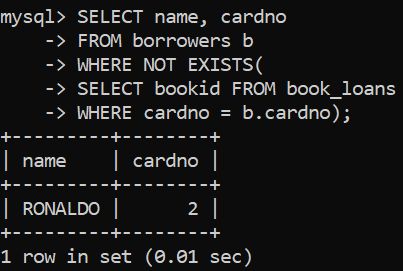
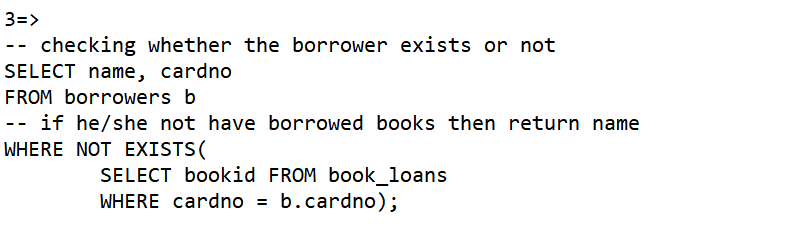
Output:



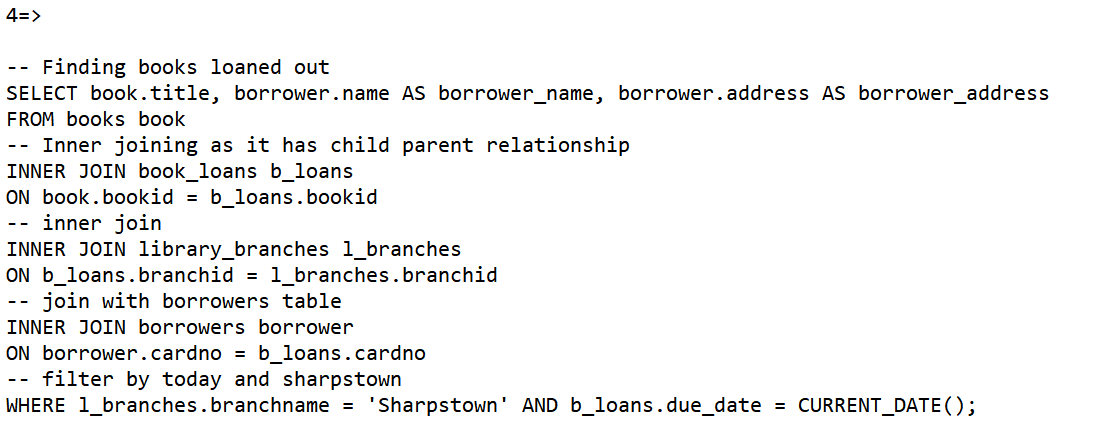
2) “How many copies of the book titled The Lost Tribe are owned by each library branch?”



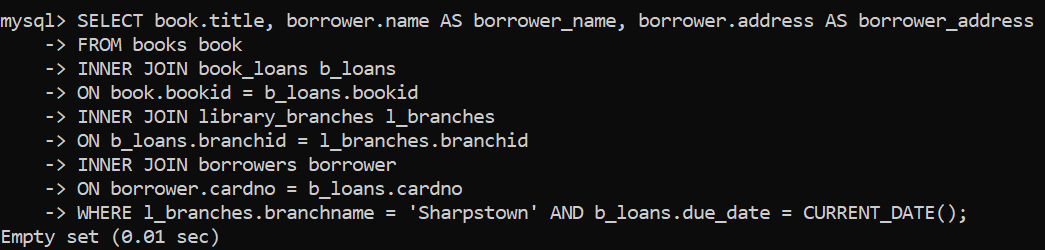
1. “Retrieve the names of all borrowers who do not have any books checked out .”



4) “For each book that is loaned out from the "Sharpstown" branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.”

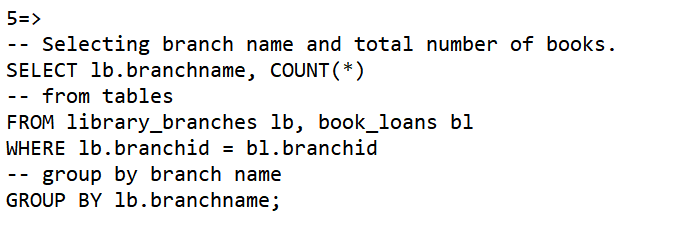


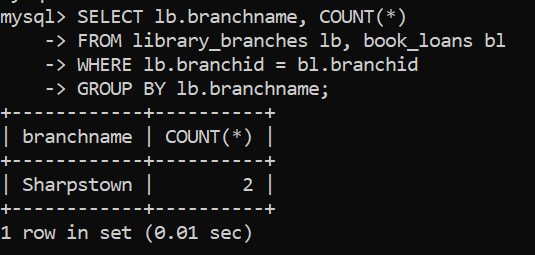
Output: As there is no record containing today date so no record found



5) “For each library branch, retrieve the branch name and the total number of books loaned out from that

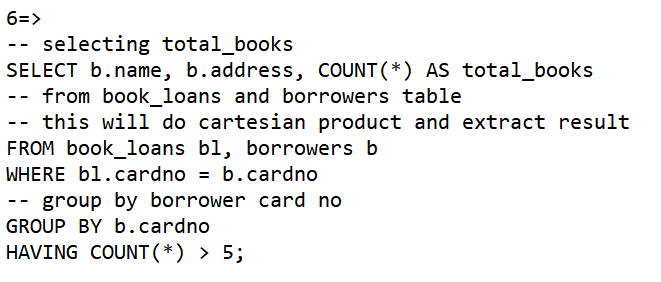
branch.”

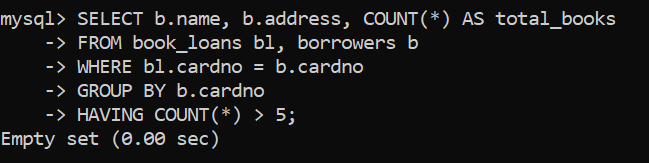




6) “Retrieve the names, addresses, and number of books checked out for all borrowers who have more than

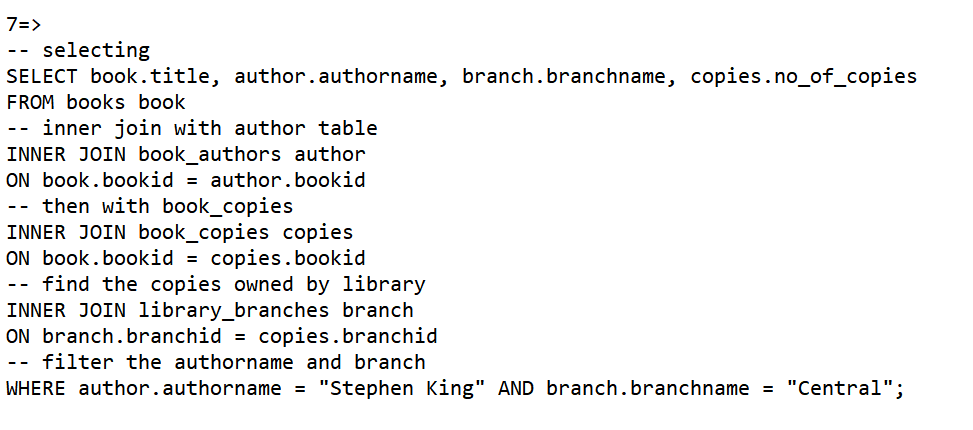
five books checked out”

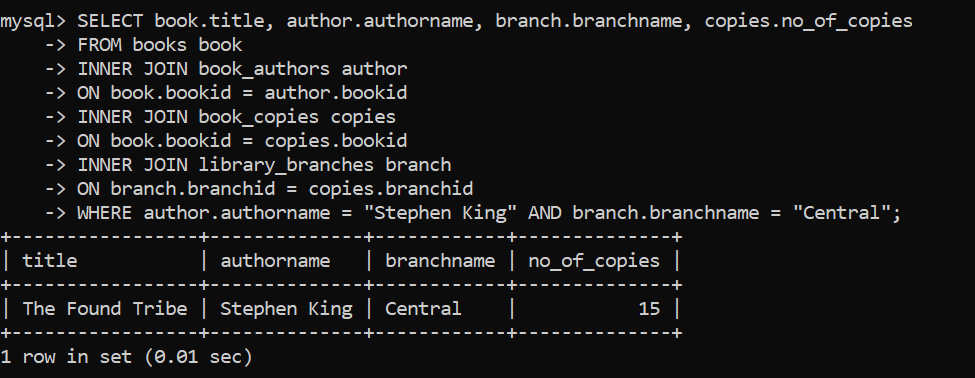




7) “For each book authored (or co-authored) by "Stephen King", retrieve the title and the number of copies

owned by the library branch whose name is "Central"”

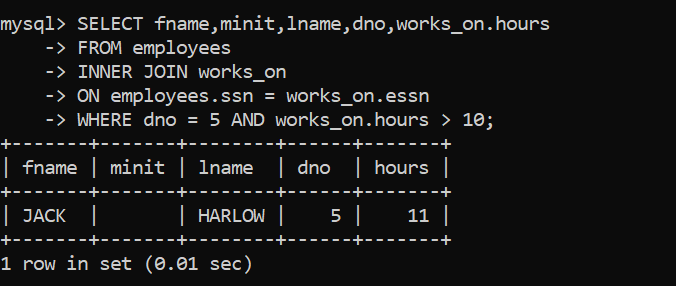
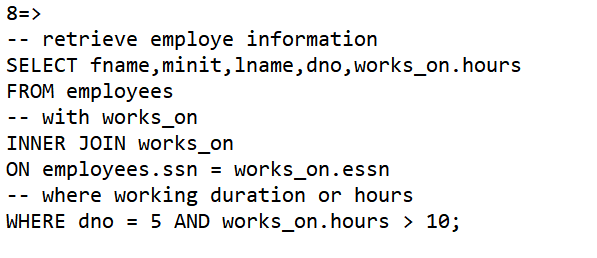




8) “Retrieve the names of employees in department 5 who work more than 10 hours per week on the

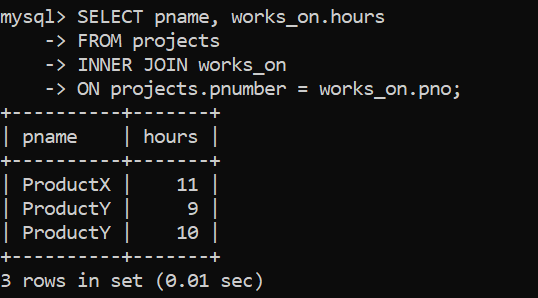
'ProductX' project.”

=> SOLUTION,

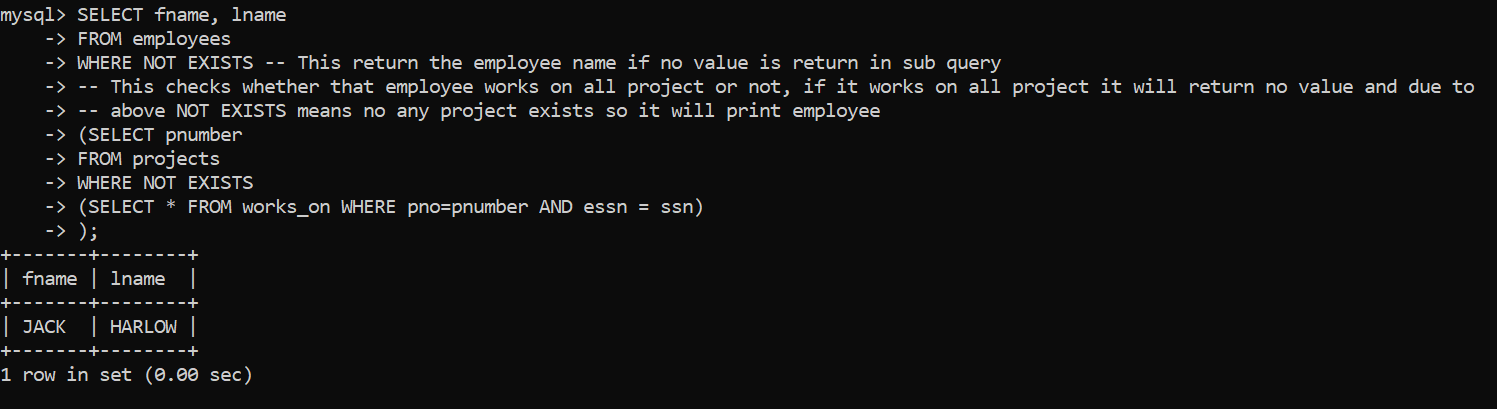
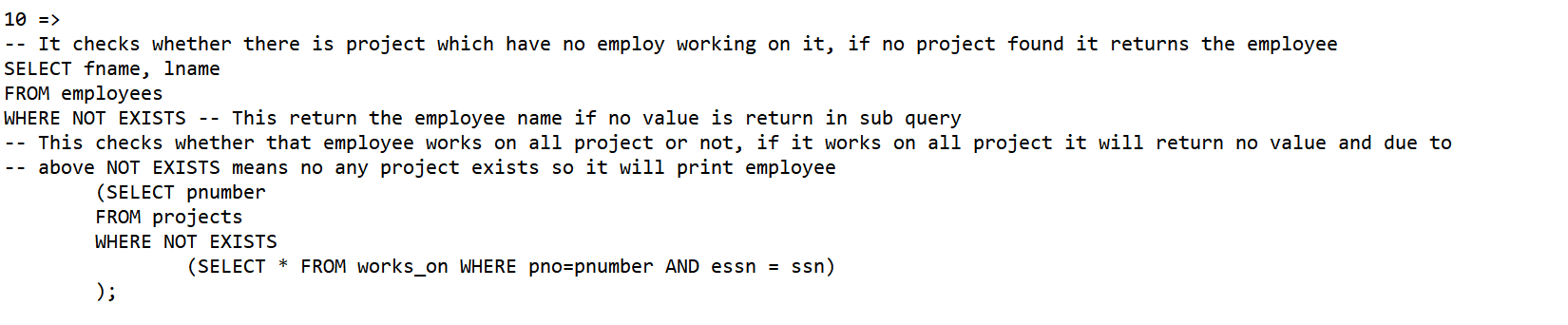


9) “For each project, list the project name and the total hours per week (by all employees) spent on that

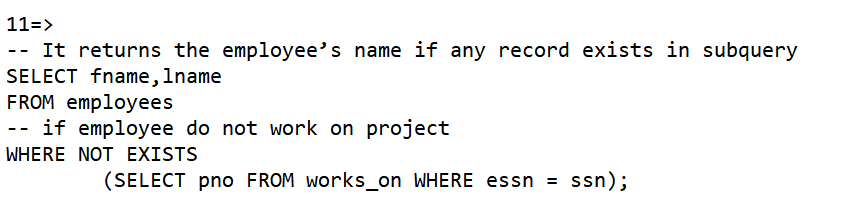
project.”

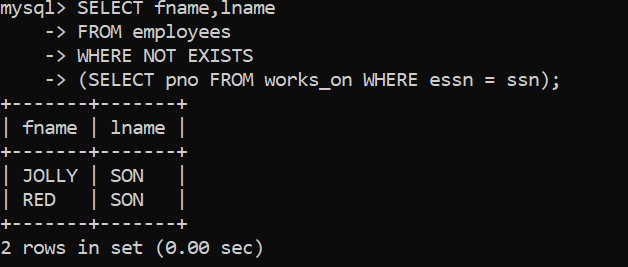


10) “Retrieve the names of employees who work on every project.”



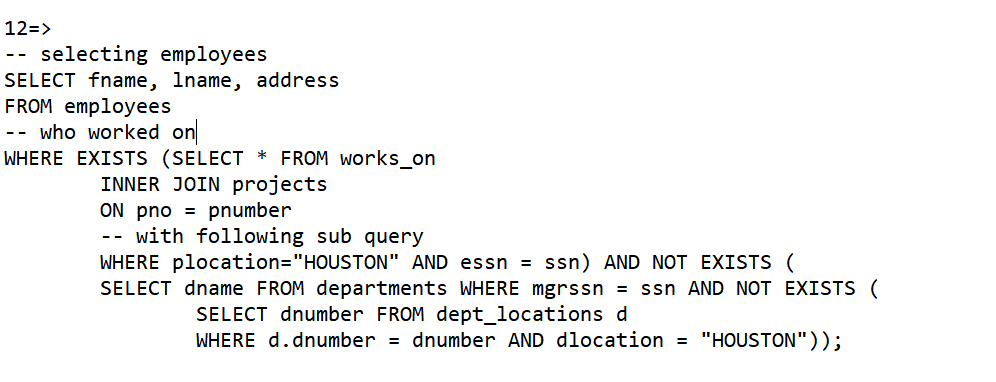
11) “Retrieve the names of employees who do not work on any project.”

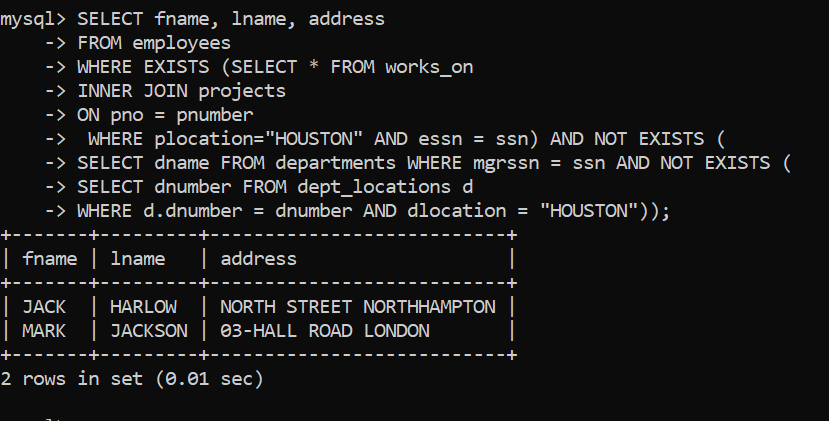




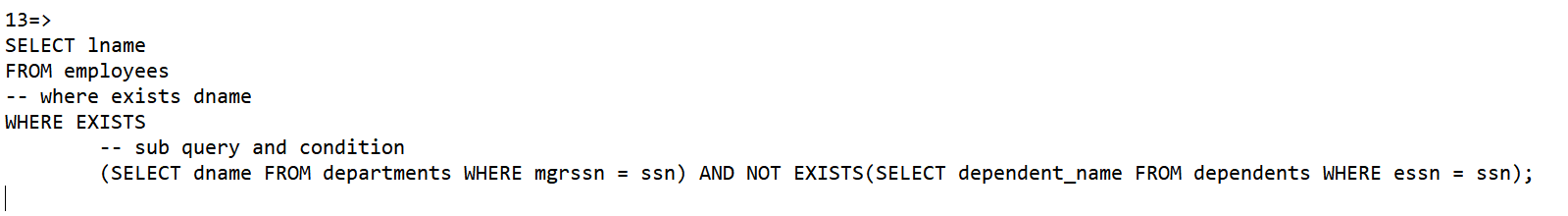
12) “Find the names and addresses of employees who work on at least one project located in Houston but

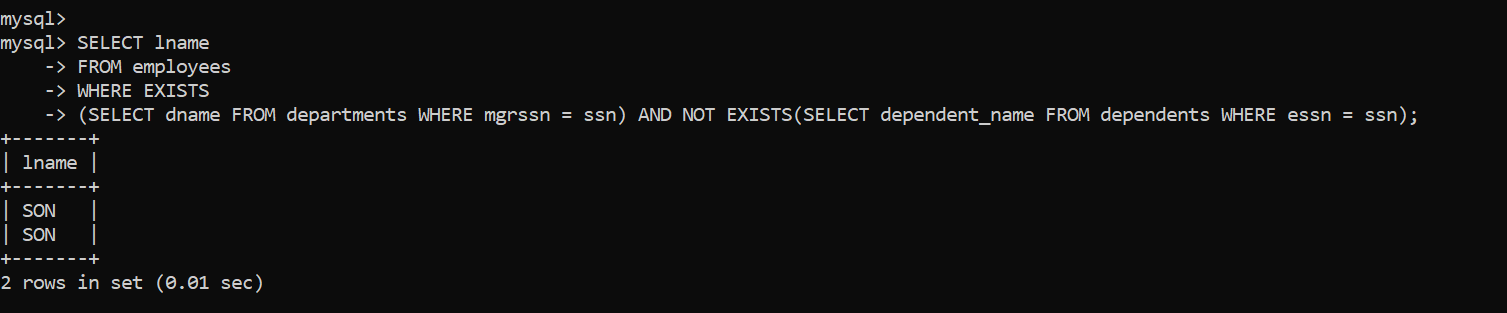
whose department has no location in Houston.”





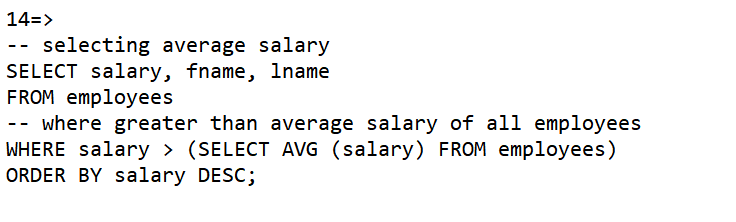
13) “List the last names of department managers who have no dependents”

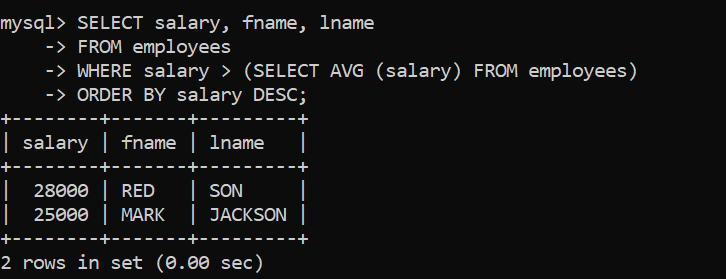




14) “Find details of those employees whose salary is > the average salary for all employees. Output salary in

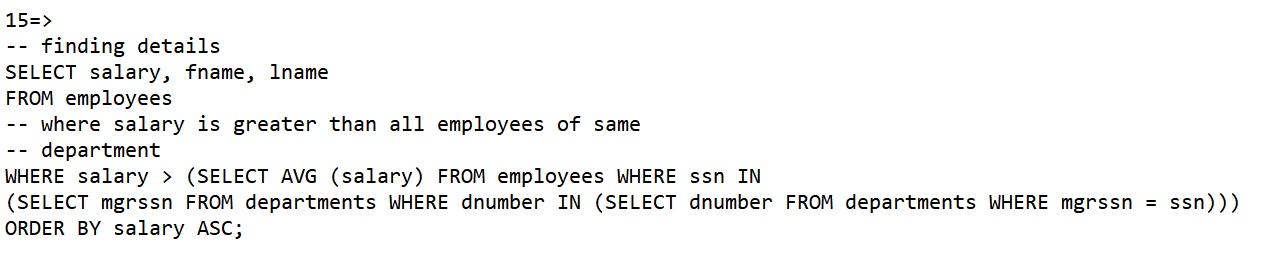
descending order.”

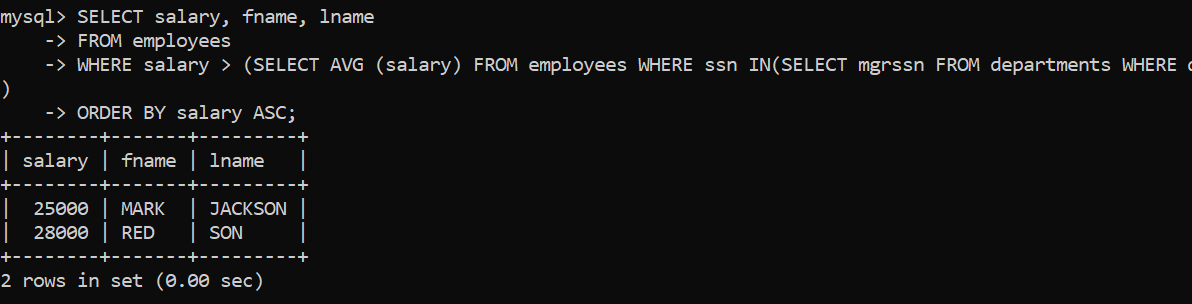




15) “Find details of those employees whose salary is > the average salary for all employees in his/her

department. Output salary in ascending order.”





**Problem arised:**

While doing these student activities, different problem had arisen, there was problem while running MYSQL. Questions were tricky to solve, with nested queries were needed to be implemented for solving. Syntax error was occurring while working with JOIN and Nested queries. Finally I solved my all activities.

**Week 2**

* Learned about NoSQL.
* Learned about the performance, scalability, usage, limitations of NoSQL.
* Learned disadvantages of RDBMS.
* Compared NoSQL with RDBMS.
* Researched on performance of NoSQL.
* Observed the future scope of NoSQL.

1. What is NoSQL?

NoSQL Databases are non-tabular databases that are designed to store distributed data that requires extremely large storage.

1. The rationale of NoSQL movement.

The main aim of NoSQL movement is to preserve and storing growing data, converting unstructured data to structure form for easy tracking and covering the limitations of RDBMS.

1. Different kinds of NoSQL data stores are:
2. Key-Value stores:

They are simplest and stores data in identifiers(key) and value form. Key can be simple strings or lists. While value can be BLOB or any kind of data. Data searches can be only performed on the basis of keys, which are unique identifiers so that its performance and scalability is high.

Examples: Amazon DynamoDB, Redis, Oracle NoSQL

1. Document Databases:

They store and handle documents, in standard data format such as BSON, JSON or XML. A column can contain several types of attributes , and types and quantity of attributes may vary according to row. Query can be performed based on both key and values and they are suited for application that uses JSON.

Examples: CouchDB, MongoDB

1. Column-Family Stores:

It consists of column-oriented, distributed data that consists of many attributes per key, designed for storage of large quantity of data, with high read, write performance and availability. It works on multiple servers, so small data is preferred to be stored in Key-value or document databases.

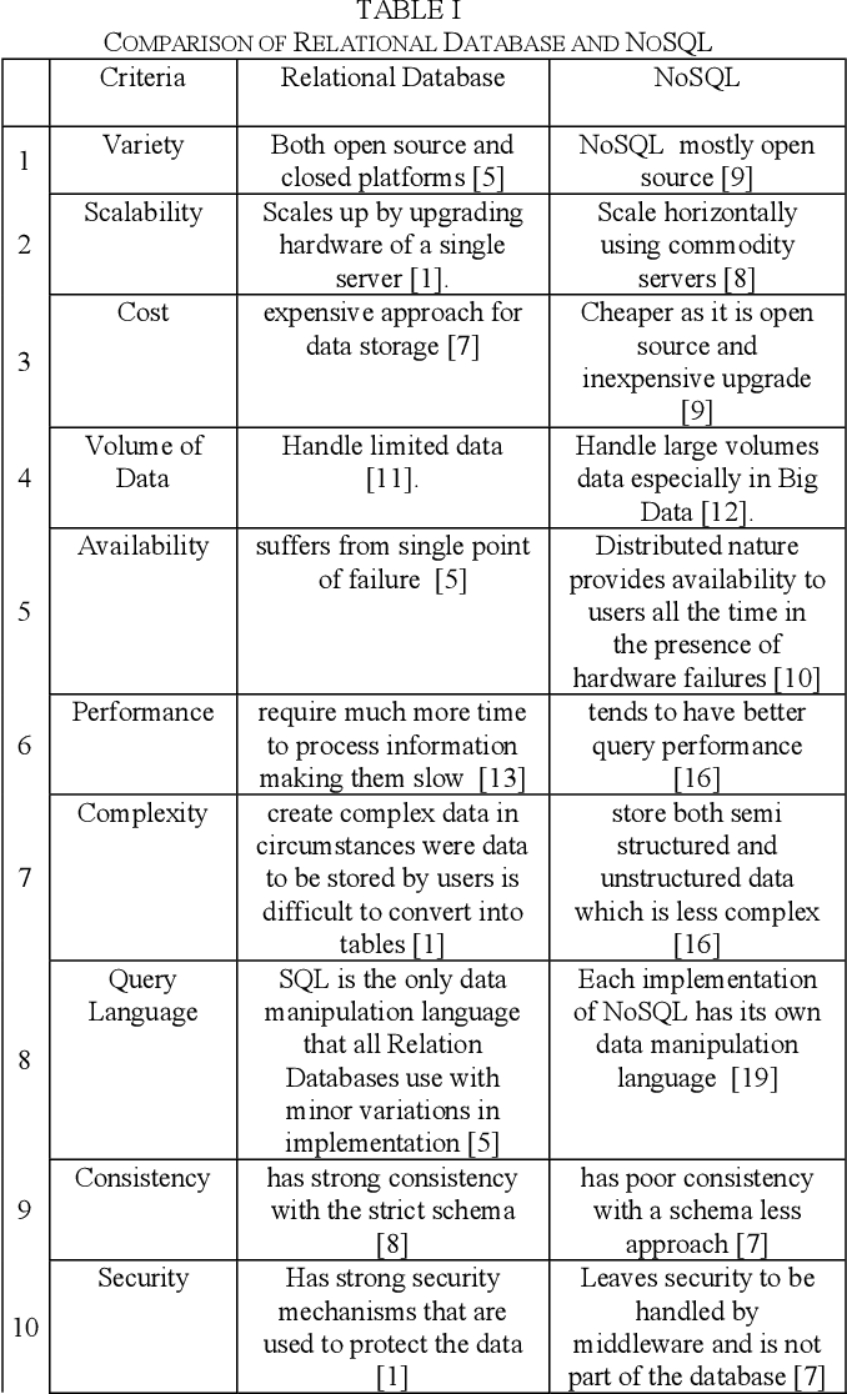
Examples: Google BigTable, Cassandra

1. Graph Databases:

They can replace traditional database tables by storing data in different structured relational graphs bounded by key-value pairings. They are used when there are relationships between entities. They consist of object-oriented networks of nodes, relationships, and properties. They focus of visual presentation of data maintaining relationships among them.

Examples: Neo4j, InfoGrid, InfiniteGraph

1. Comparison of NoSQL and relational database.



1. Some challenges faced by relational database while adopting NoSQL are:

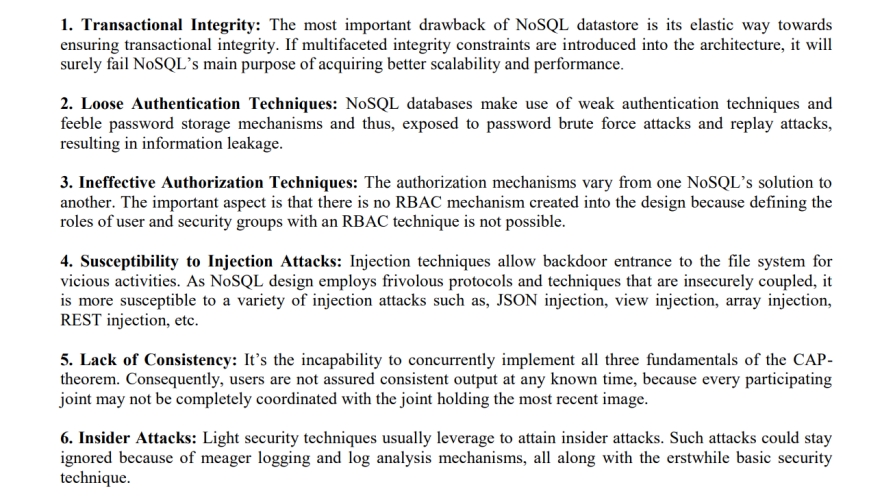
- Lack of consistency

- Insider attacks

- Loose Authentication Techniques

- Transactional Integrity

- Ineffective Authorization Techniques



**Week3:**

In week 3 I have done some revision of some SQL queries. I revised back to the Week 1 and week 2 slides and tutorial. Searched some aims and objectives of NoSQL movement. Learned comparison between NoSQL and RDBMS.

**Week 4:**

I learned how to start Neo4j Desktop, first I ran movie database and followed guide provided there. I learned basic flow of Neo4j and some basic syntax of Cypher Query language like MATCH statement.

- Learned about nodes, labels, relationship, variables, and so on.

- To create nodes and relationship.

E.g.

// CREATE Teacher node with name "JACK" and assign TEACHES relationship with course Node

CREATE (n:Teacher{name:"Jack"})-[t:TEACHES]->(c:course{name:"AI"})

RETURN n,t,c;

- Find the node and assign the relationship.

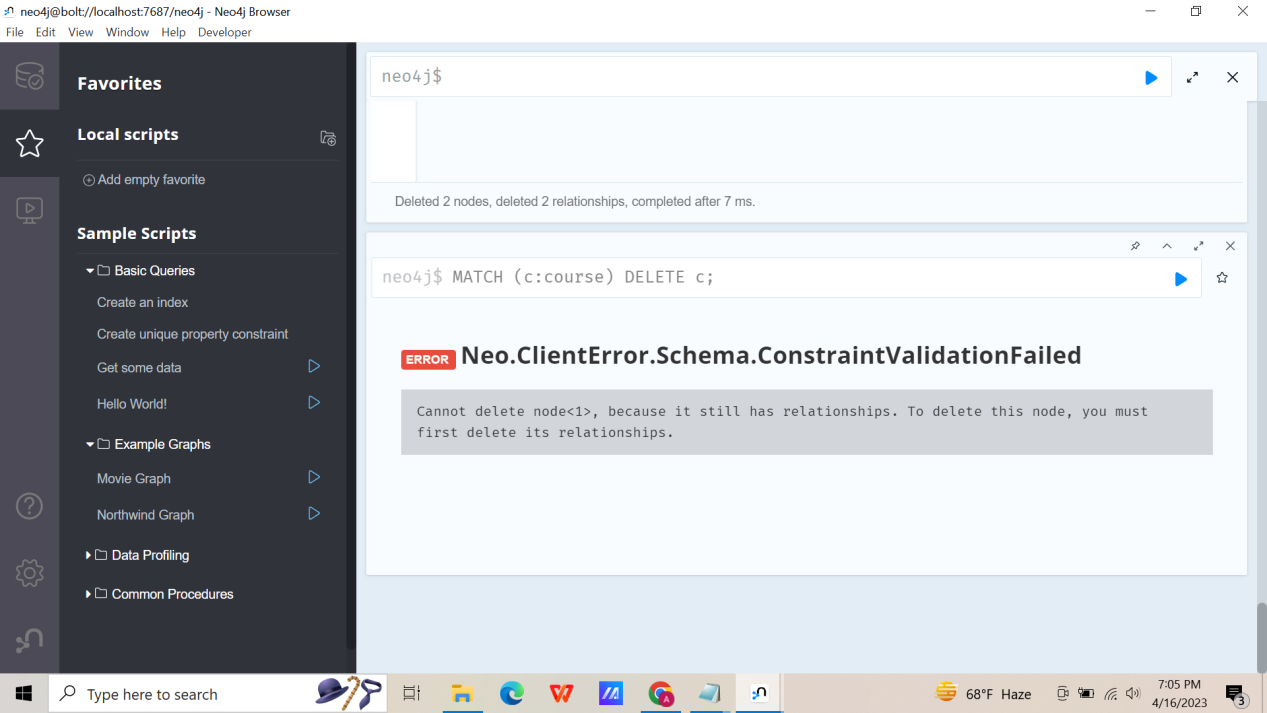
- Optional Relationship is denoted with question mark in earlier versions of Neo4j.

- Three different types of operators I.e. mathematical (like +,-, \*,/), comparision (like =, <>, <,>), special operator (like ?, !) are used in cypher.

- pattern can be assign to variable like q = (m)-[\*3..5]->(n)

- Learned to delete or detach node and its relationship.

- Difficulties arrise:



Some errors saying its relationship still exists had arise while deleting the nodes.

To solve this issue:

- Delete its relationship first and delete node

- or use detach delete command to delete node regardless of its relationship

**Appendix:**

// TO delete all nodes

MATCH (n) DETACH DELETE(n);

// Creates node n with label teacher and name attribute Jack

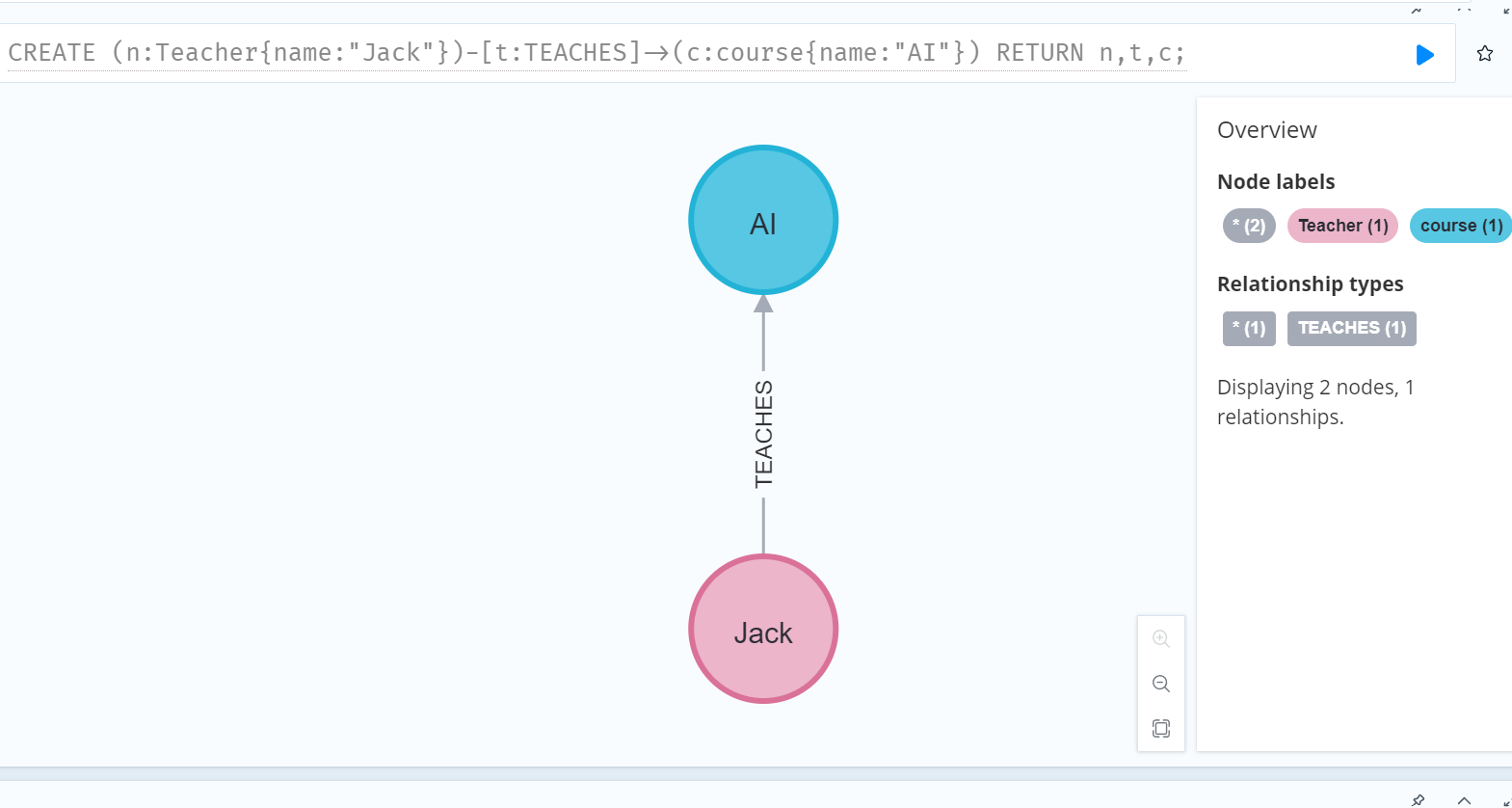
CREATE (n:Teacher{name:"Jack"});

// CREATE Teacher node with name "JACK" and assign TEACHES relationship with course Node

CREATE (n:Teacher{name:"Jack"})-[t:TEACHES]->(c:course{name:"AI"})

RETURN n,t,c;

**Result:**



// CREATING NODE AND ASSIGNING RELATIONSHIp

CREATE (teacher:Teacher{name: "HARI SPARROW"});

CREATE (module:Module{code: "CSY 2028", name: "WEB 2" });

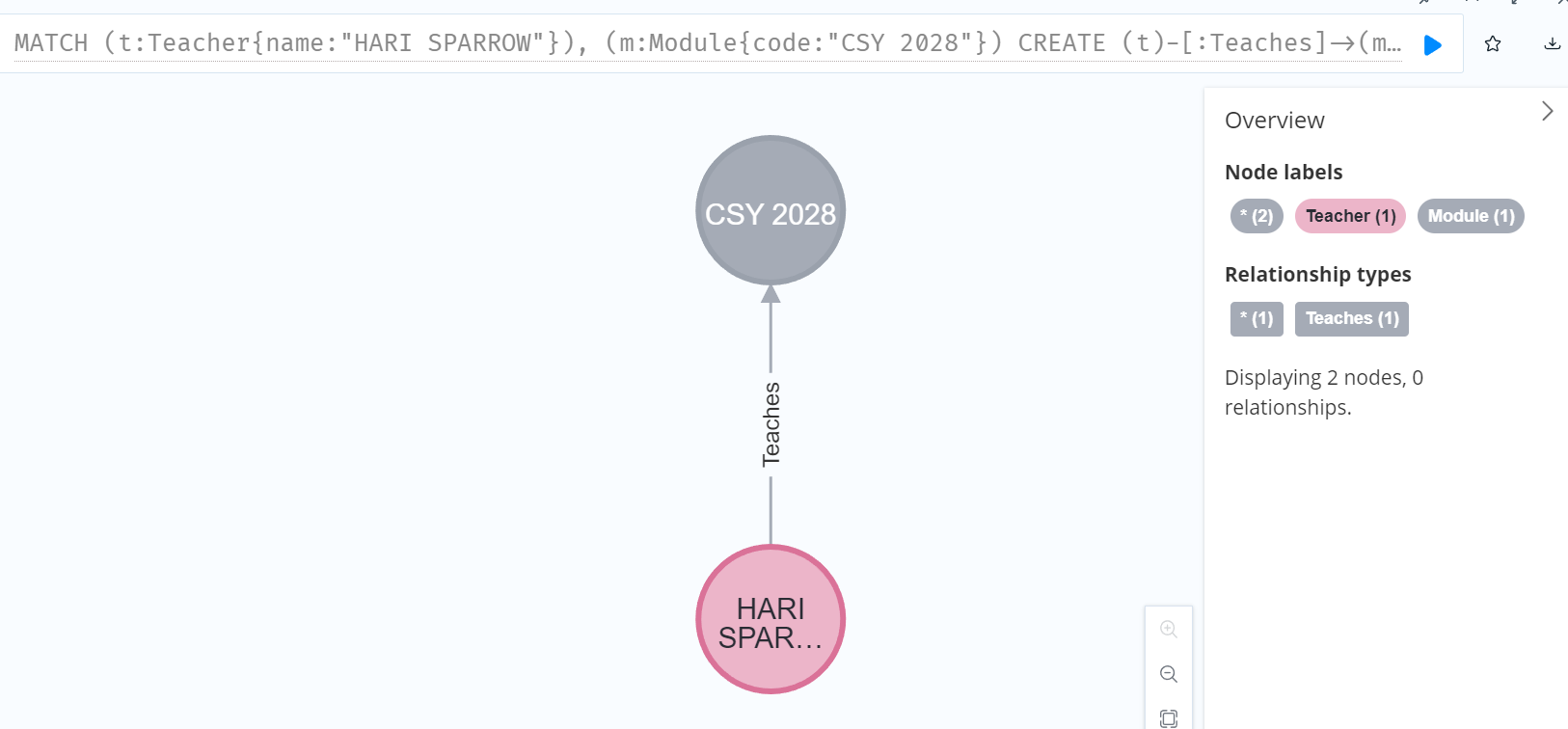
CREATE (pathway:Pathway{code: "C", name: "Engineering"});

MATCH (t:Teacher{name:"HARI SPARROW"}), (m:Module{code:"CSY 2028"})

CREATE (t)-[:Teaches]->(m)

RETURN t, m;

**Result:**

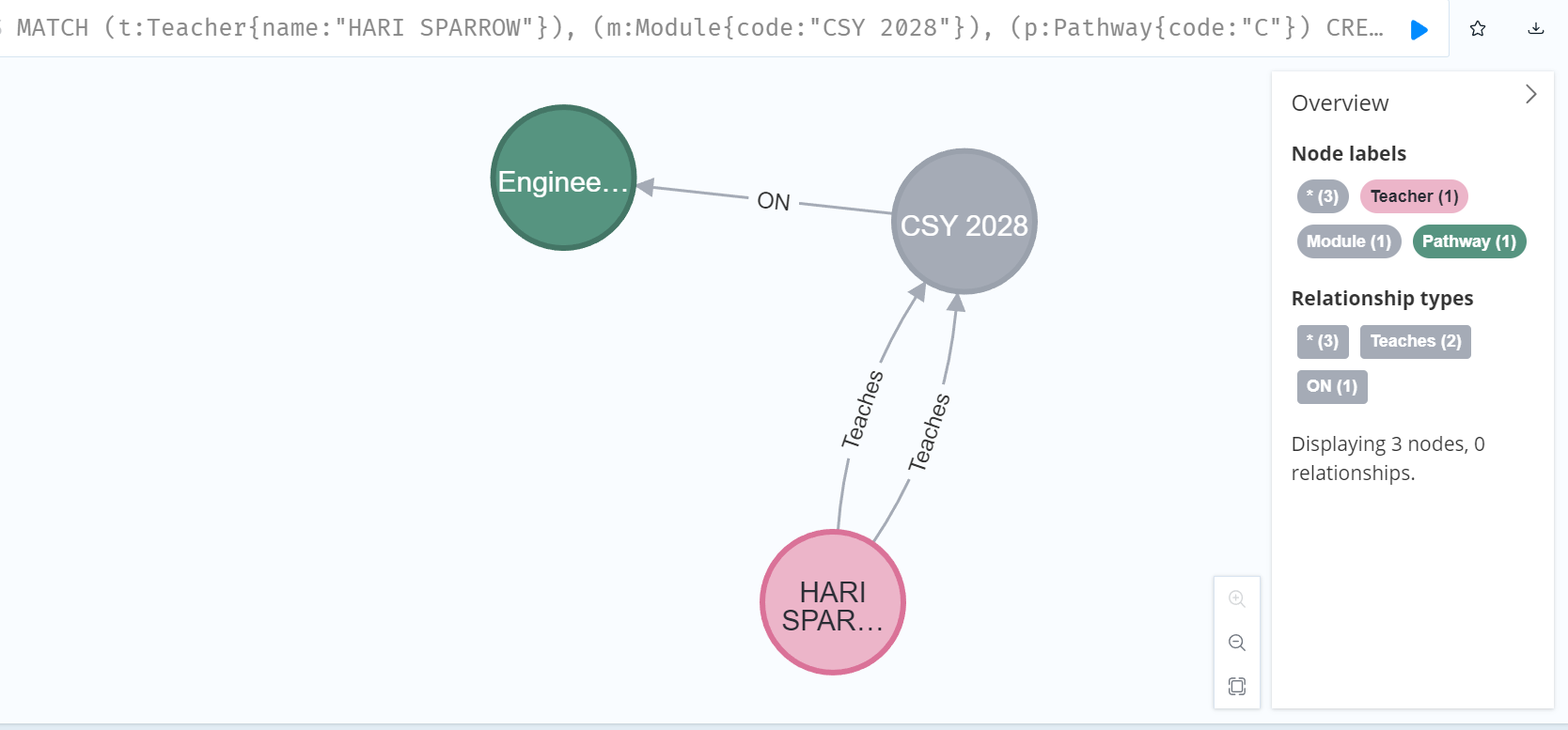


MATCH (t:Teacher{name:"HARI SPARROW"}), (m:Module{code:"CSY 2028"}), (p:Pathway{code:"C"})

CREATE (t)-[:Teaches{since:"2018"}]->(m)-[:ON]->(p)

RETURN t, m, p;

**Result:**

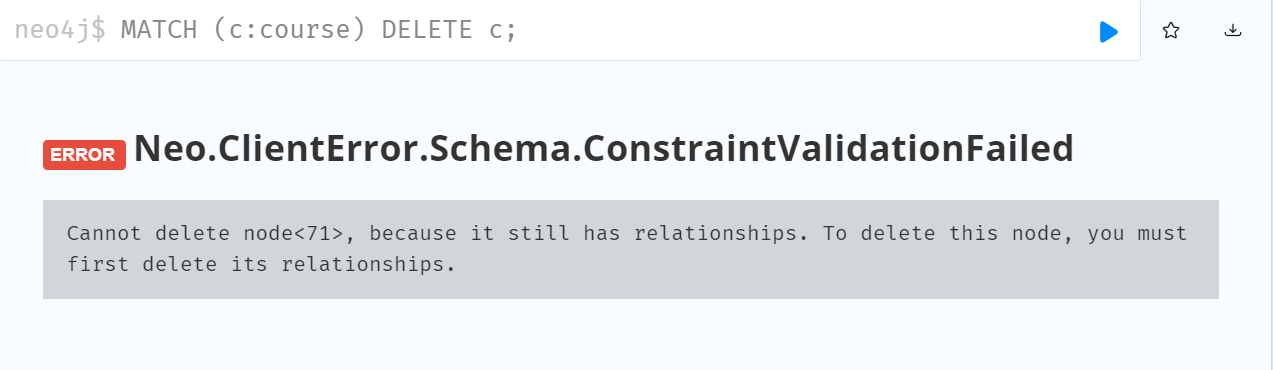


//Deleting single node

// This do not work if you have relationship

MATCH (c:course) DELETE c;

**Result:**



// This will delete node with its relationship

MATCH (c:course) DETACH DELETE c;

**Extra:**

// What if one TEACHES relationship has ’since’ property,

//The other hasn’t?

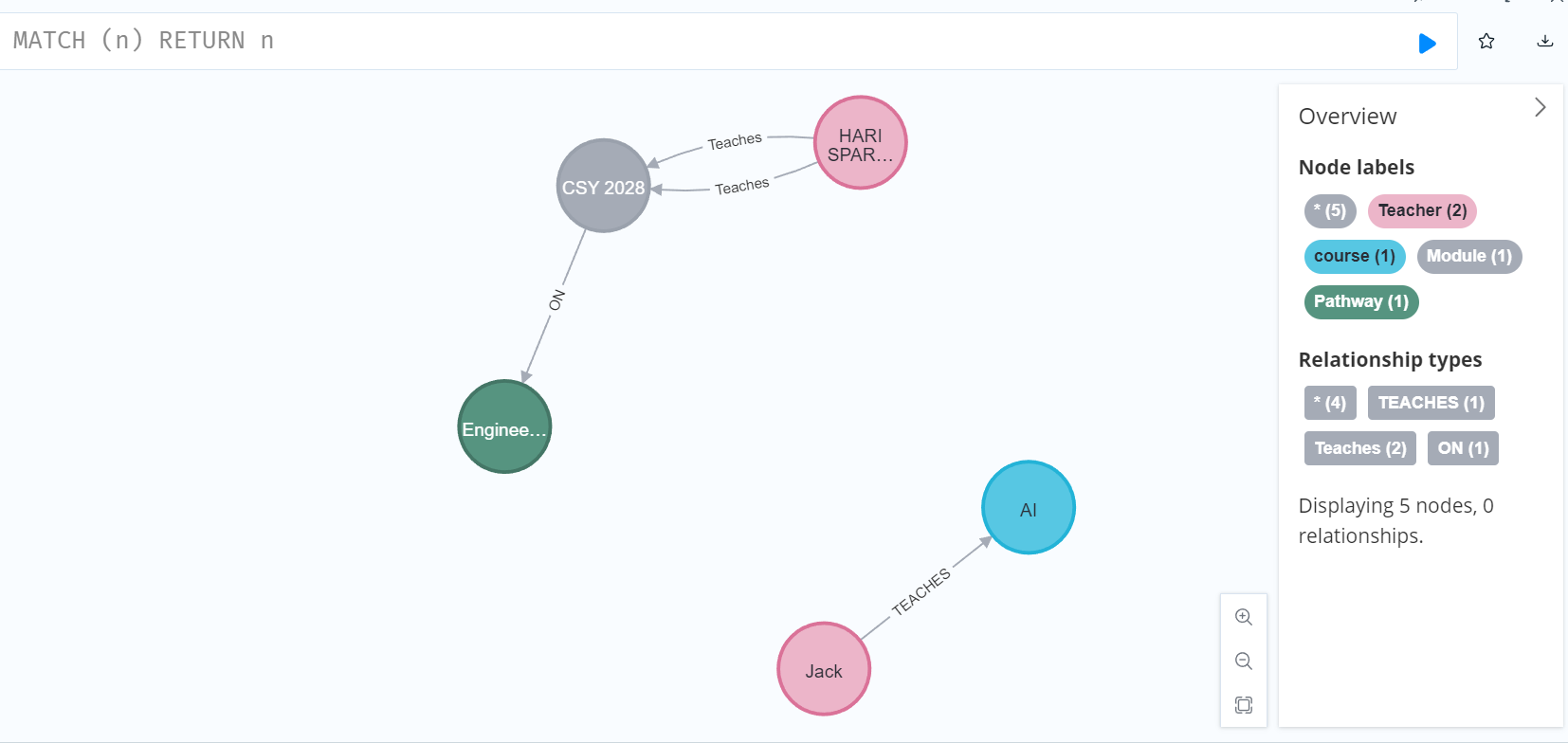
// => It will delete all relationship

MATCH (t:Teacher{name:"HARI SPARROW"})-[teaches:Teaches]->(m:Module{code:"CSY 2028"})

DELETE teaches;

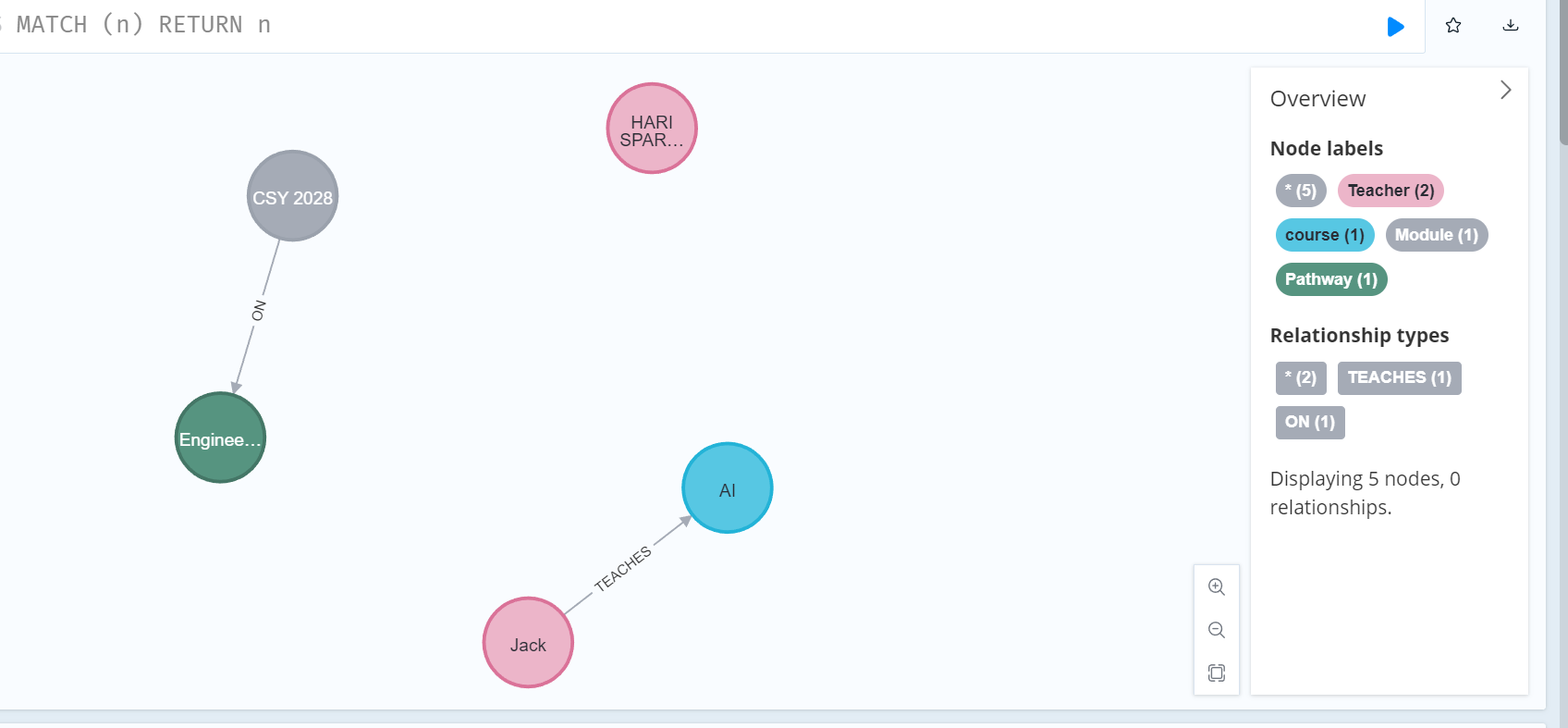
**Result:**

**Before Deleting:**



**After deleting:**

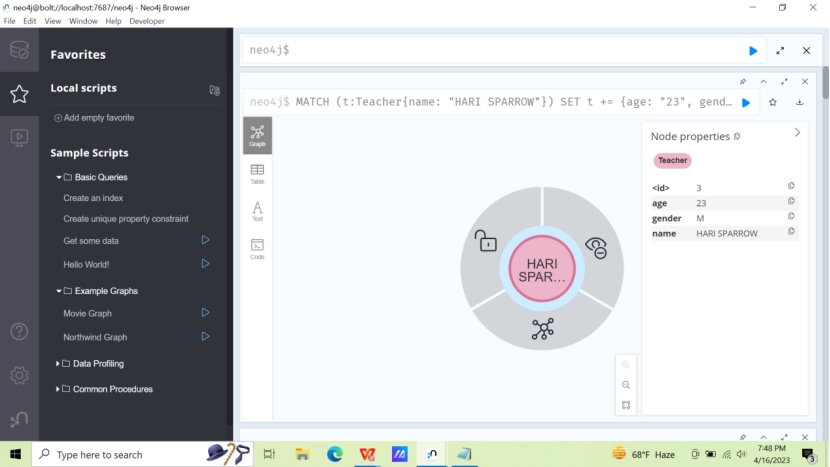
All [:teaches] relationships are removed.

****

**Week 5:**

- Learned to create constraints, index.

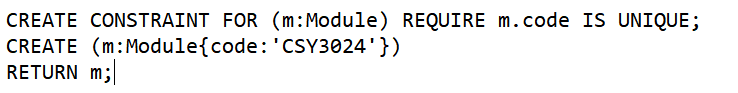
- Learned to set attribute of node already create node



Here age and gender properties is added of teacher

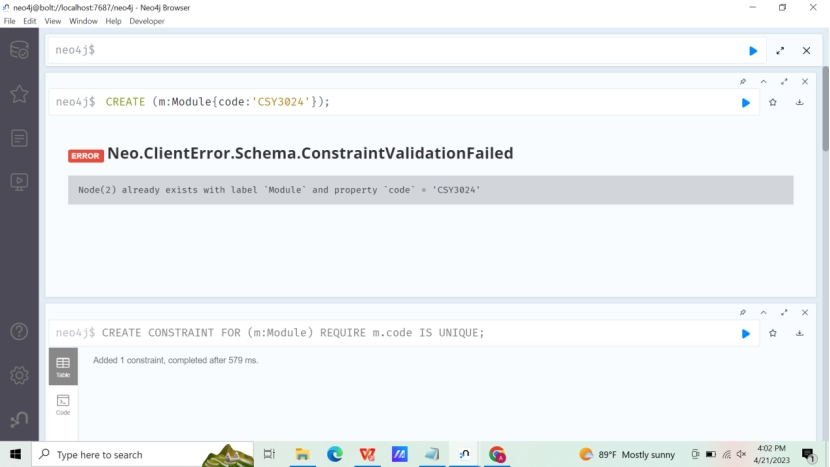
**Try It Out:**

- Created unique constraint for node Module having unique code.



- It shows following error on duplicate creation of Module node having same code. It is due to unique constraint added to this property.

CREATE (n:Module{code:'CSY3024'});

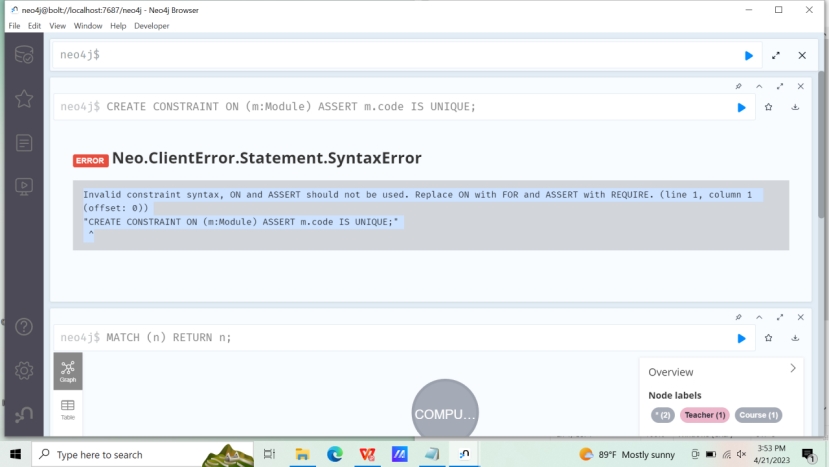


**Analysis:**

- Code provided in slide did not work for my version of Neo4j or new version.

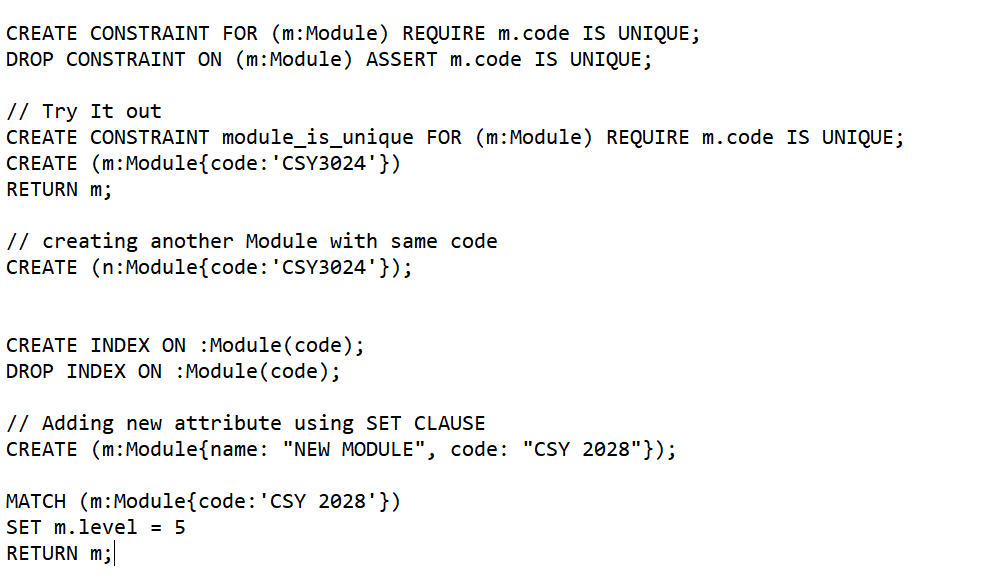


It shows following error:



“CREATE CONSTRAINT FOR (m:Module) REQUIRE m.code IS UNIQUE;” worked for me.

**Appendix:**



**Result:**



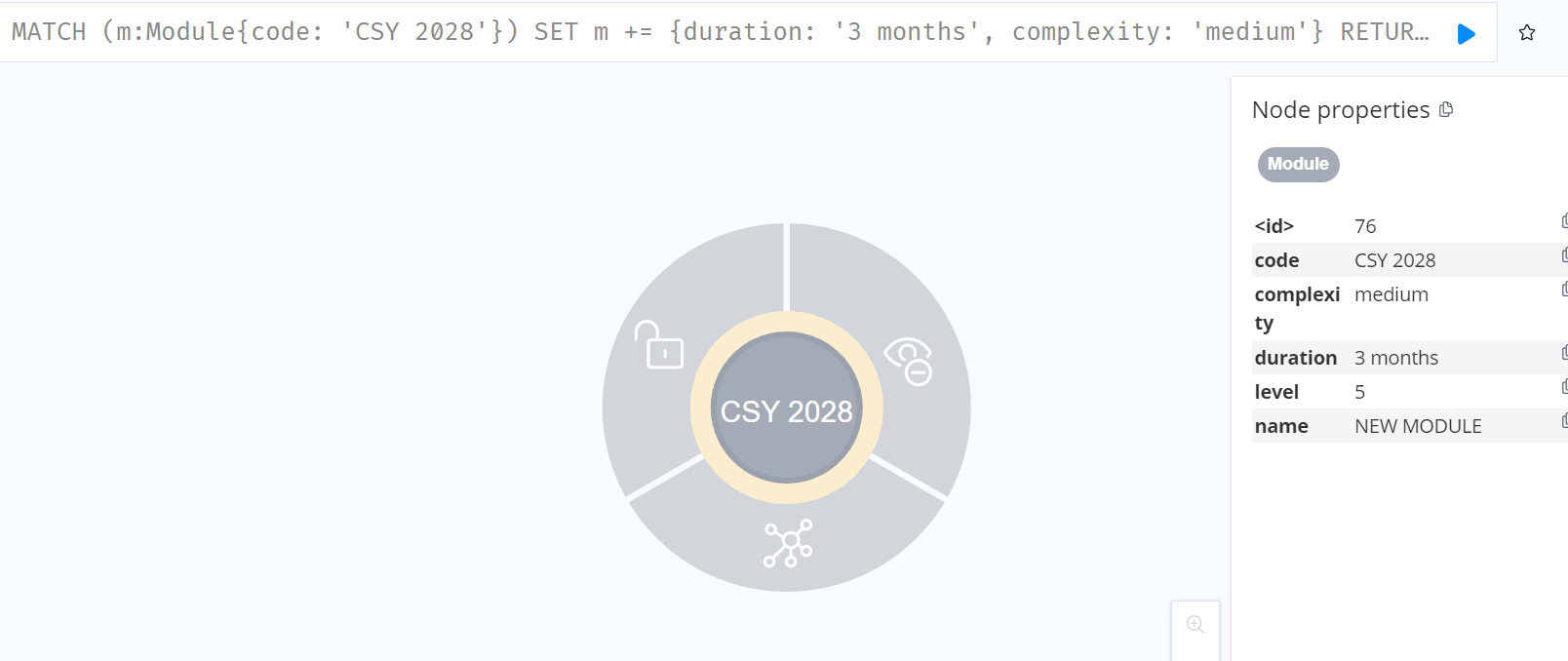
// Adding properties from map

MATCH (m:Module{code: 'CSY 2028'})

SET m += {duration: '3 months', complexity: 'medium'}

RETURN m;

**Result:**

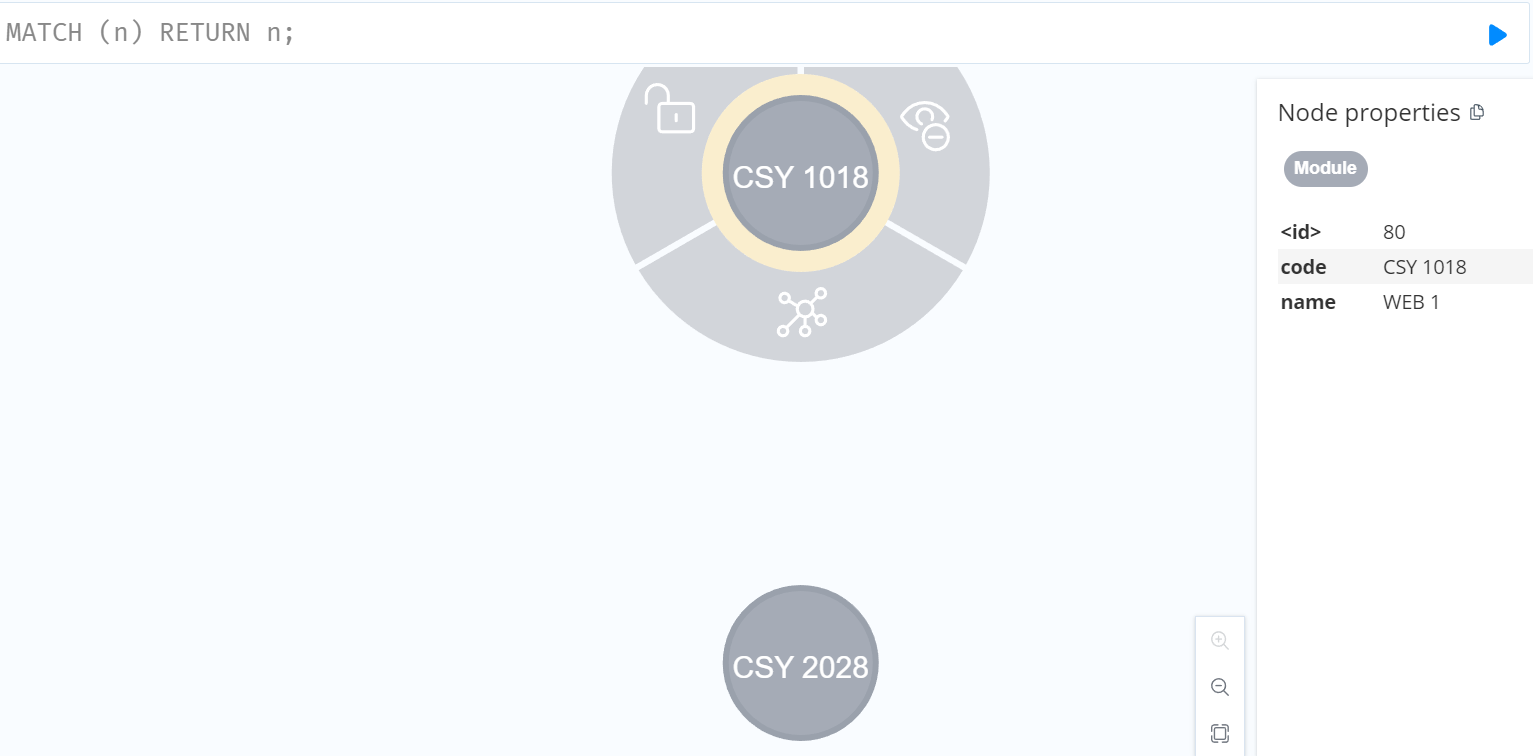


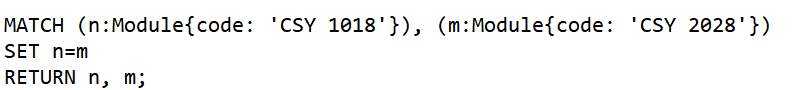
// copying properties from one node to anther

MERGE (n:Module{code: 'CSY 1018', name: 'WEB 1'});

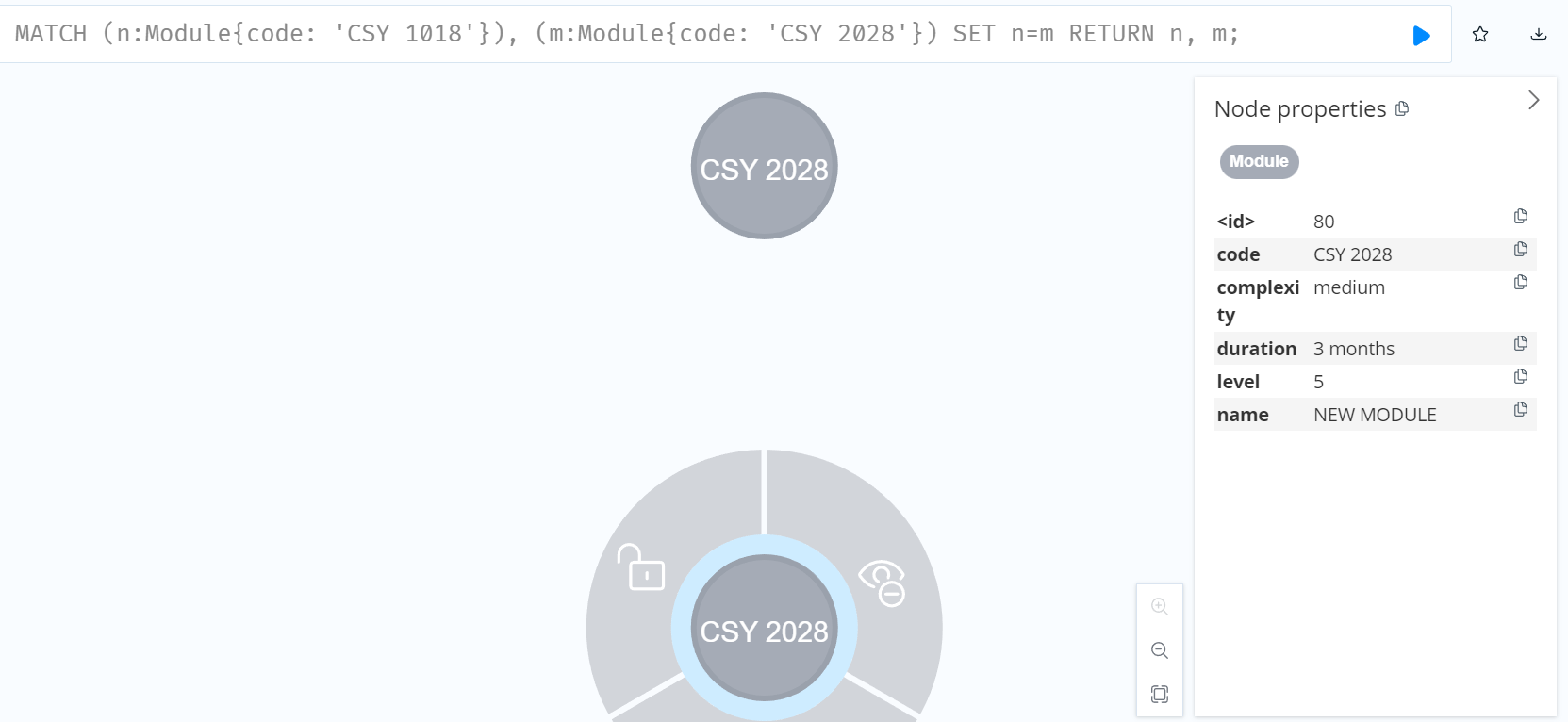
MATCH (n) RETURN n;

Result:

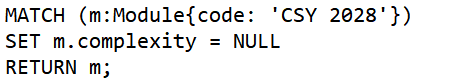




**Result**: It will copy all property of ‘CSY 2028’ to ‘CSY 1018’ module



// removing property complexity by using NULL



**Result:**

Attribute complexity is removed.



// Updating label

// This will add new label in existing label

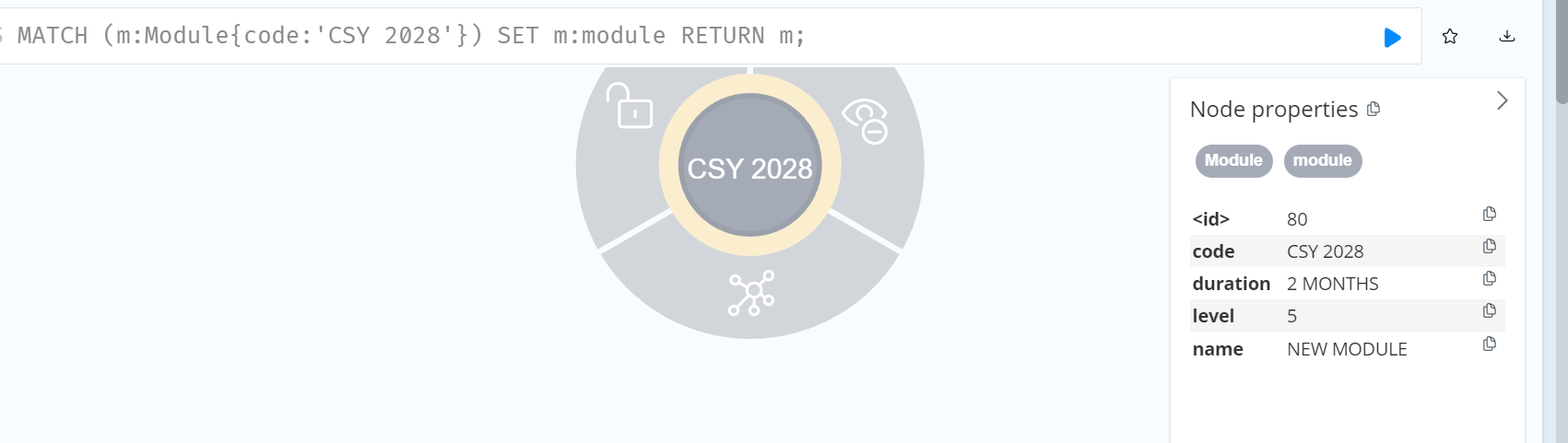
MATCH (m:Module{code:'CSY 2028'})

SET m:module

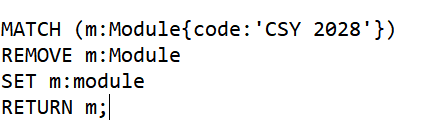
RETURN m;

**Result:**

New label ‘module’ is added on existing label.



// to update previous label first delete previous label and add new label



**RESULT:**

A picture containing diagram

Description automatically generated

// adding age and gender to previous week node

MATCH (t:Teacher{name: "HARI SPARROW"})

SET t += {age: "23", gender: "M"}

RETURN t;

**Result:**

**Chart

Description automatically generated**

**Week 6:**

* In week 6, I learned about MERGE command, MERGE command is simply the combination of MATCH or CREATE command.
* MERGE creates new node or relationships if node or relationships do not exists, if already exists then it returns them.
* We cannot track whether create or match done by MERGE command by using MERGE with ON MATCH SET or ON CREATE SET.
* I learned to create relationships with MERGE.
* I practiced 8 cases of MERGE RELATIONSHIPS and completed exercises

**Appendix:**

CREATE (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'})-[:Teaches]->(course: Course{name: 'COMPUTING'})

RETURN teacher, course;

CREATE (c:Car{no: 123, brand: 'Mercedes'});

CREATE CONSTRAINT FOR (food:Food) REQUIRE food.name IS UNIQUE;

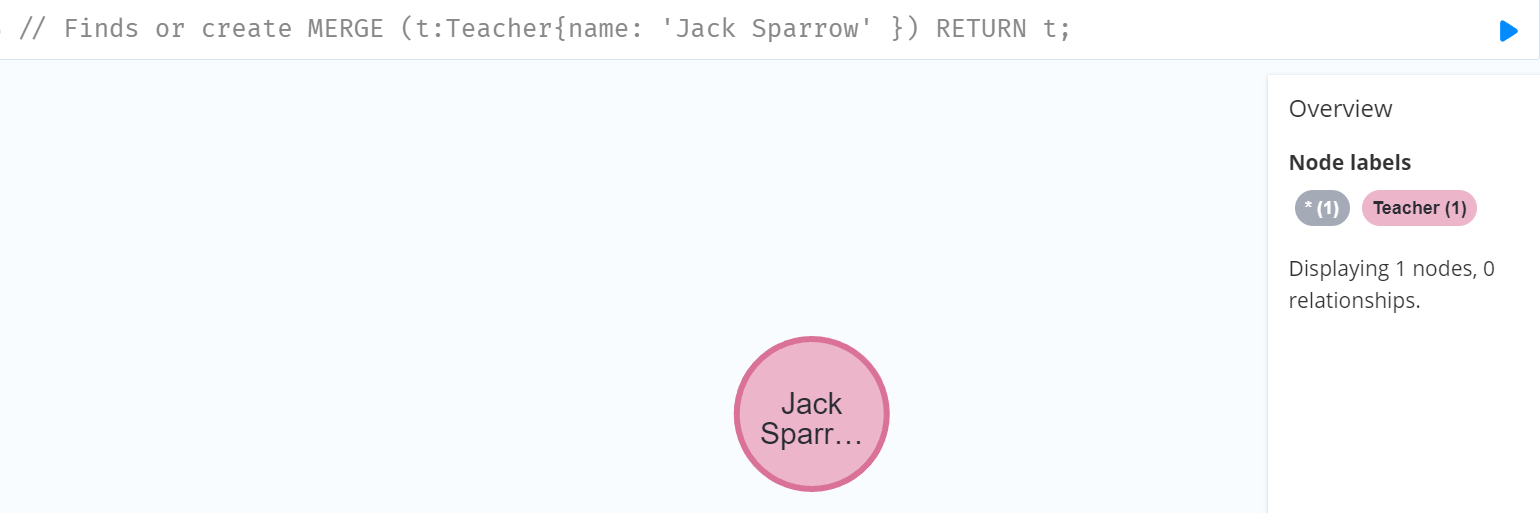
CREATE (food:Food{name: 'PIZZA'});

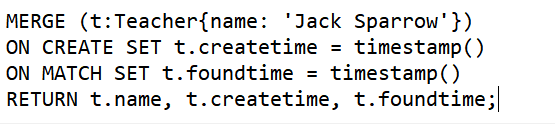
// Finds or create

MERGE (t:Teacher{name: 'Jack Sparrow' })

RETURN t;

RESULT:





RESULT: Here it finds the node and set found time as it works as MATCH in this case.



// MERGE RELATIONSHIPS

//Case 1: When relationship exist,

MERGE (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'})-[:Teaches]->(course: Course{name: 'COMPUTING'});

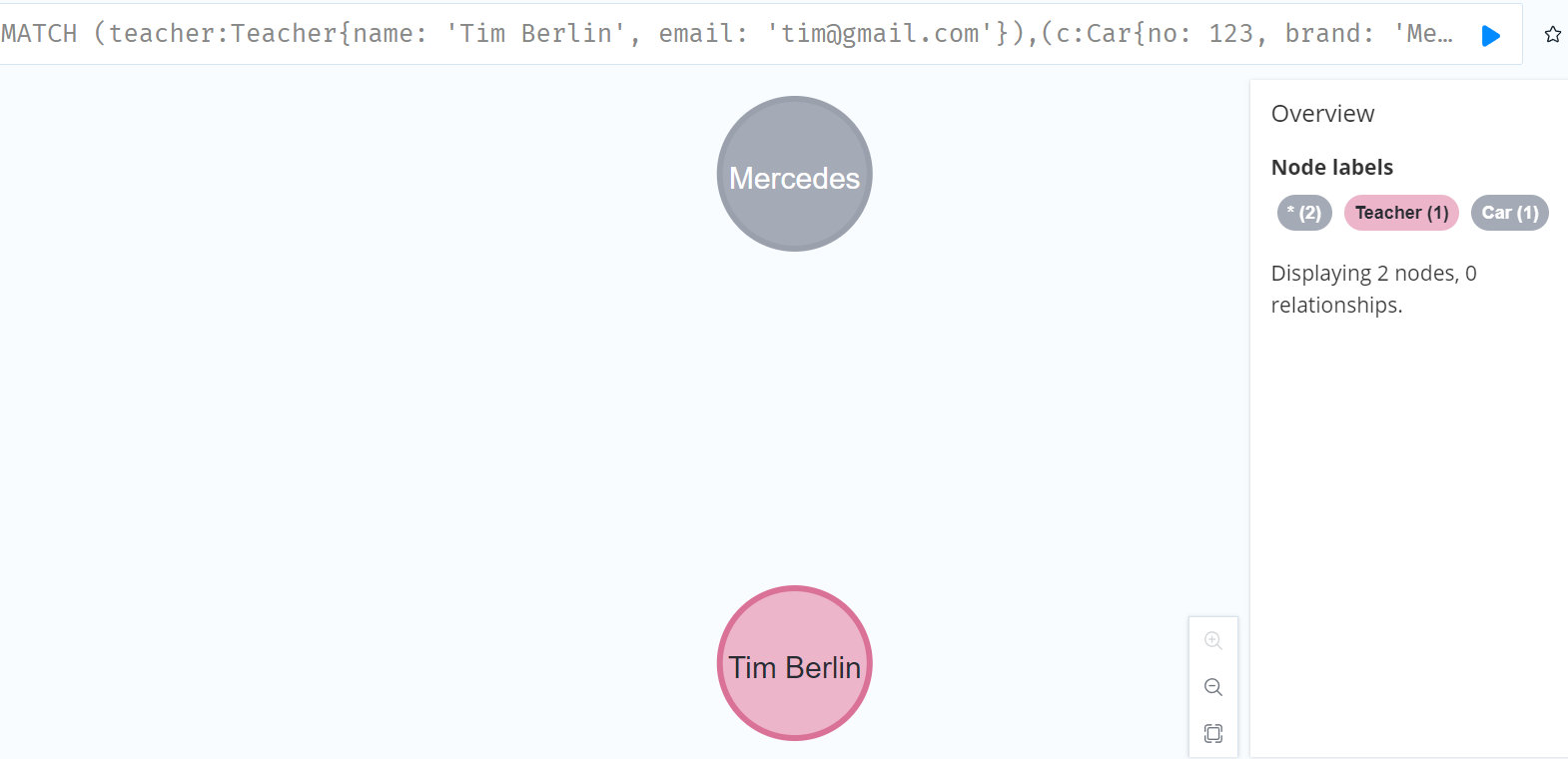
**RESULT:** It will not change any record or relationship



// Case 2: When two nodes exist, but not the relationship

MATCH (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'}),(c:Car{no: 123, brand: 'Mercedes'})

RETURN teacher, c;



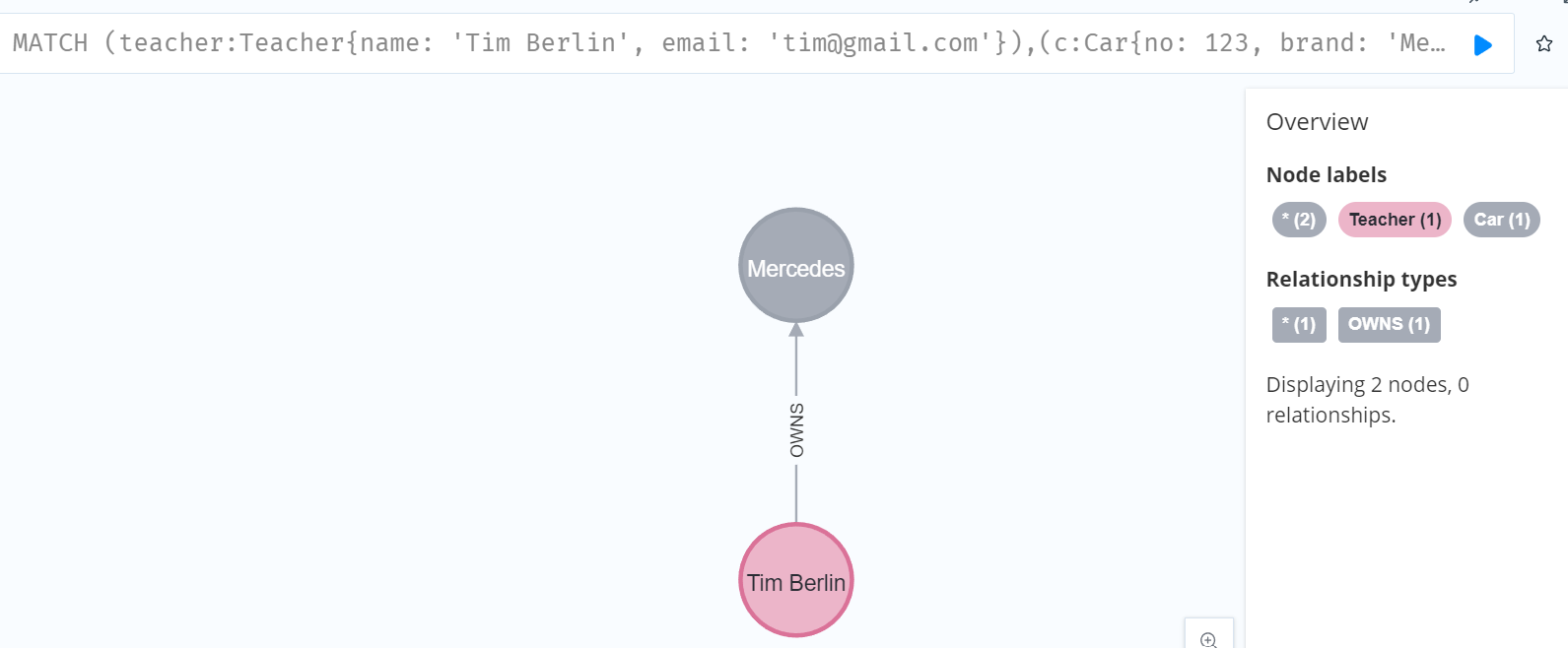
MATCH (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'}),(c:Car{no: 123, brand: 'Mercedes'})

MERGE (teacher)-[:OWNS]->(c);

MATCH (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'}),(c:Car{no: 123, brand: 'Mercedes'})

RETURN teacher, c;

RESULT:



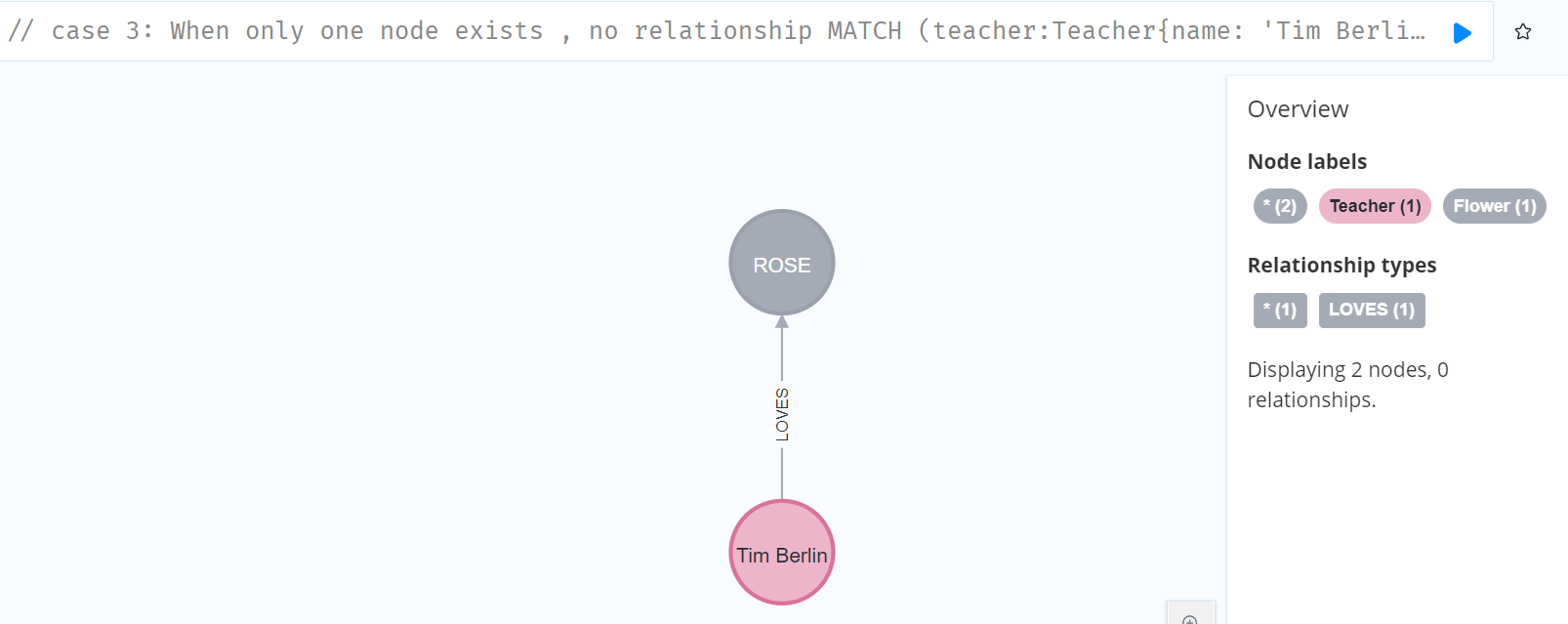
// case 3: When only one node exists , no relationship

MATCH (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'})

MERGE (teacher) -[:LOVES]->(f:Flower{name:'ROSE'})

RETURN teacher, f;

Result: It will create new node and relationship.

// case 4: When some node exists and there is a unique constraint on some property of node

MATCH (teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'})

MERGE (teacher) -[:EATS]->(food:Food{name: 'PIZZA'});

RESULT: It do not create relationship due to unique constraint property



// case 5: two nodes and relationship exists but has different direction

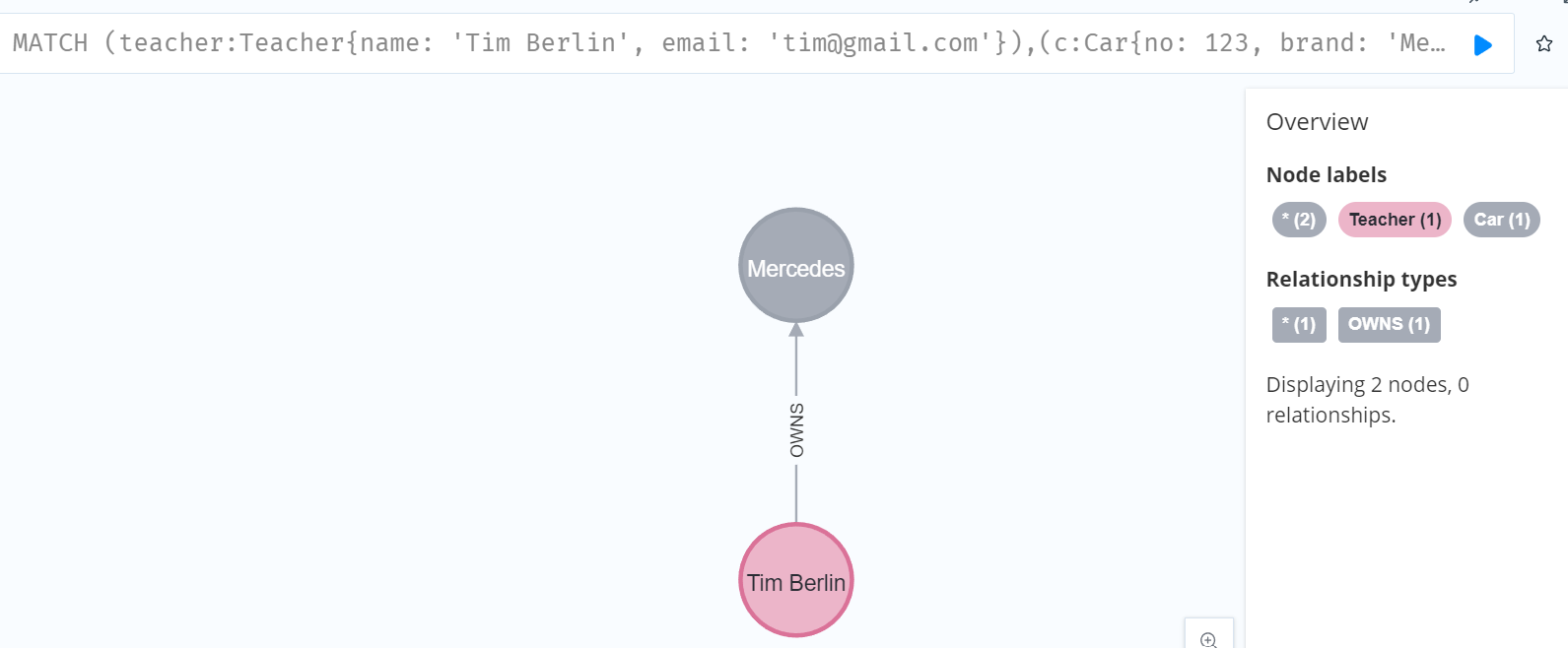
// HERE 2 new nodes and 1 relationship is created

MERGE (c:Car{no: 123, brand: 'Mercedes'})-[:OWNS]->(teacher:Teacher{name: 'Tim Berlin', email: 'tim@gmail.com'})

RETURN c, teacher;

**RESULT:**

Before it was,



After running the code the direction of relationship is changed

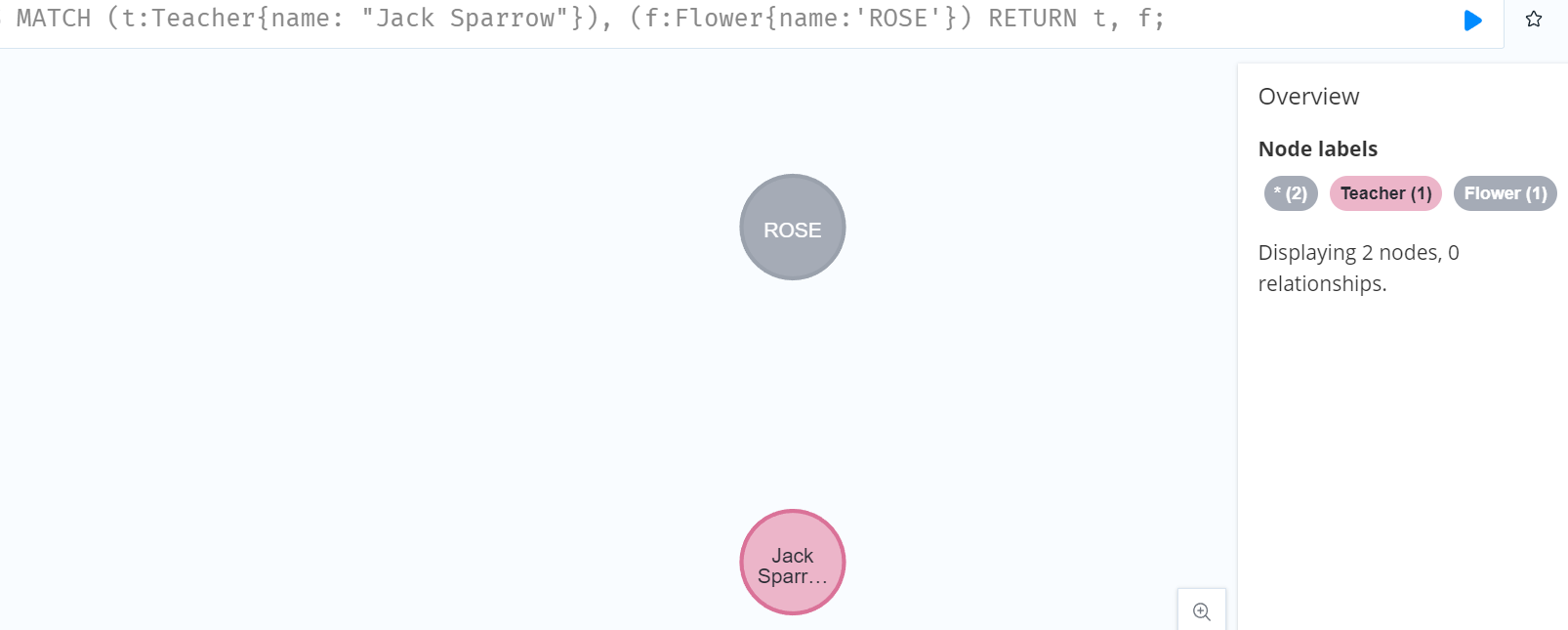


// case 6: Two nodes exist without relationship, merge a un-directional relationship

MATCH (t:Teacher{name: "Jack Sparrow"}), (f:Flower{name:'ROSE'})

RETURN t, f;

BEFORE:



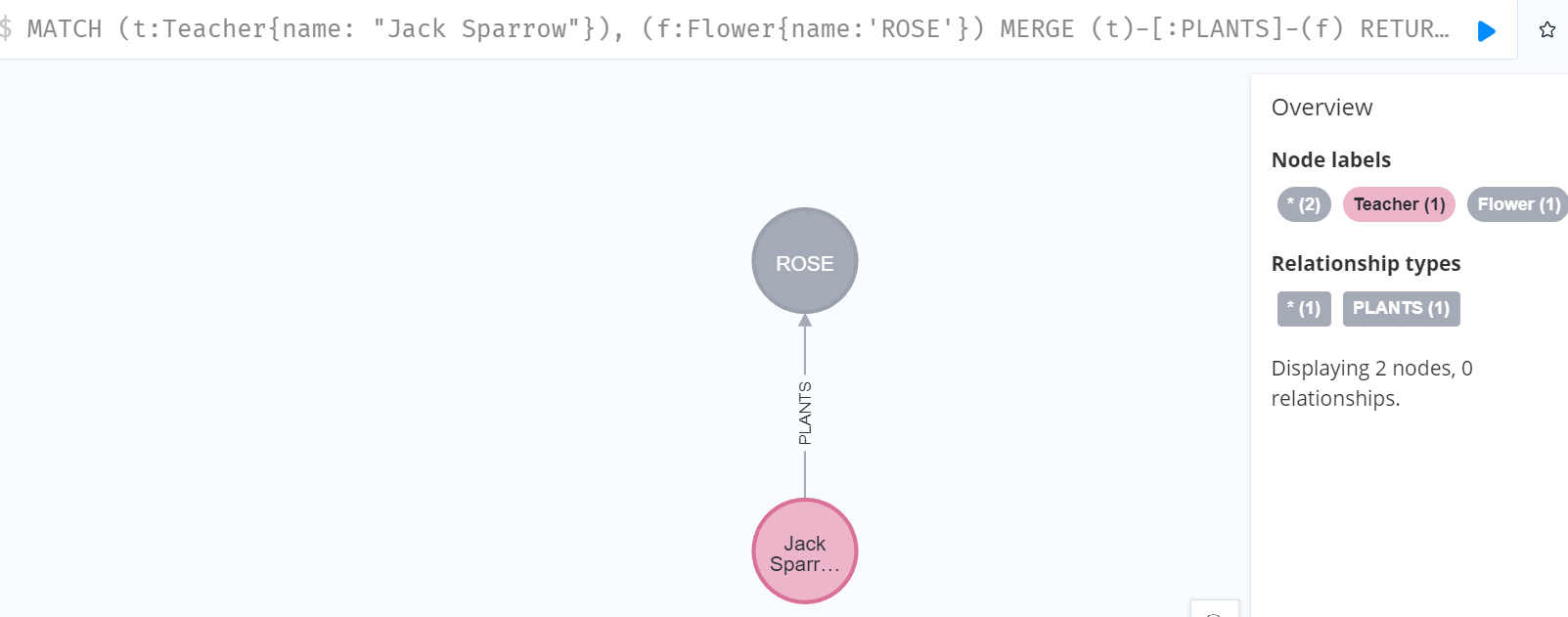
//As There is no connection between nodes, relationship will be created and direction of the created relationship is random (specially left to right)

MATCH (t:Teacher{name: "Jack Sparrow"}), (f:Flower{name:'ROSE'})

MERGE (t)-[:PLANTS]-(f)

RETURN t, f;

RESULT: After adding the unidirectional relationship.

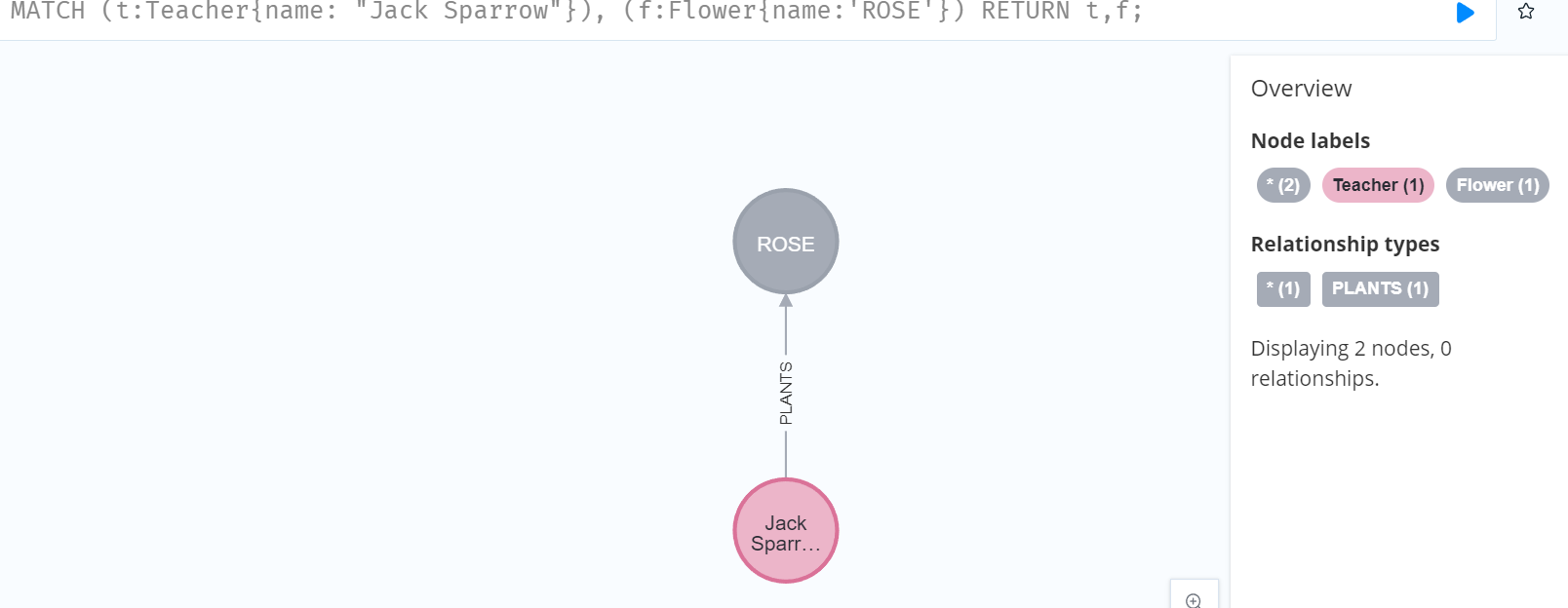


Case 7:

//case 7: When two nodes exists with relationship, MERGE a un-directional relationship

MATCH (t:Teacher{name: "Jack Sparrow"}), (f:Flower{name:'ROSE'})

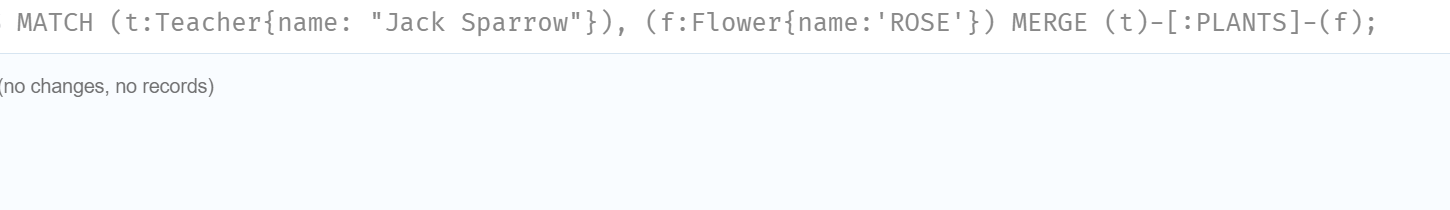
RETURN t,f;



MATCH (t:Teacher{name: "Jack Sparrow"}), (f:Flower{name:'ROSE'})

MERGE (t)-[:PLANTS]-(f);

// Result: No change



// case 8: Two nodes and relationship exists and merge different direction relationship to that

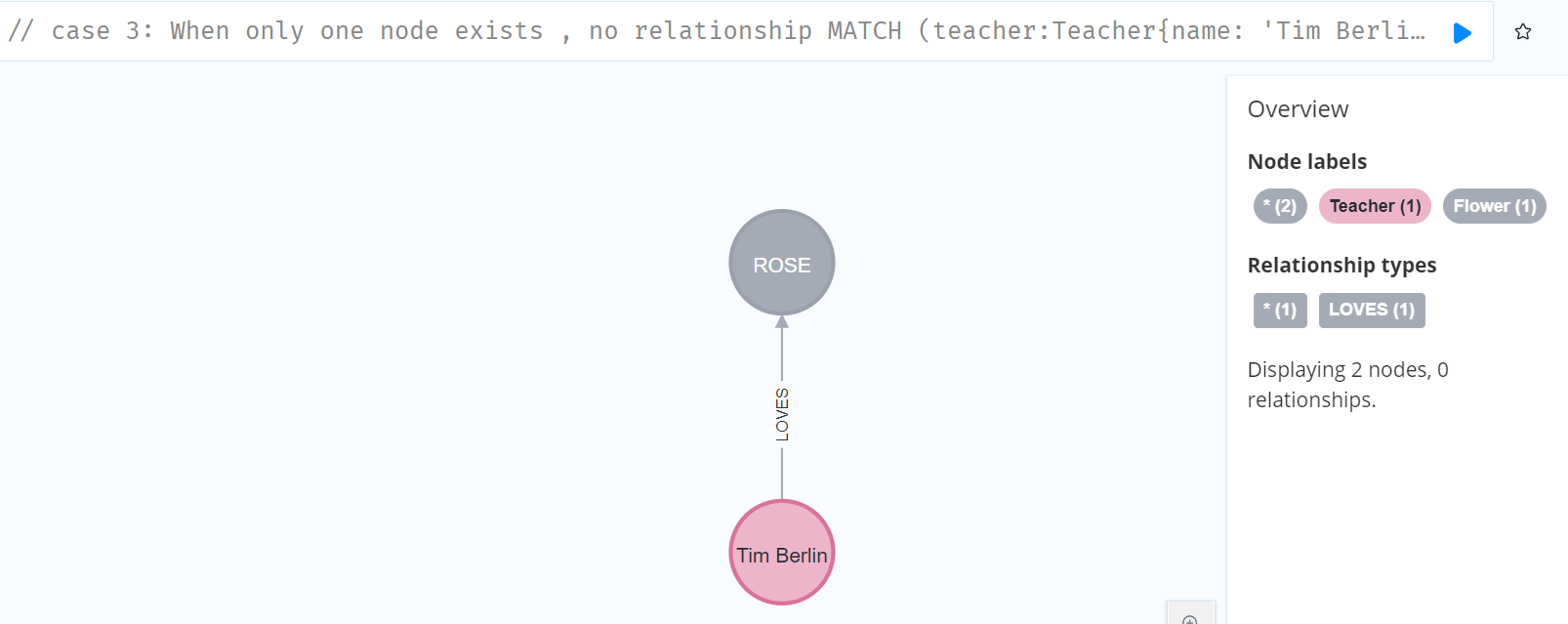
// Here 1 new relation of given direction is added in existing nodes

MATCH (teacher:Teacher{name: 'Tim Berlin'}), (f:Flower{name:'ROSE'})

MERGE (f)-[:LOVES]->(teacher);

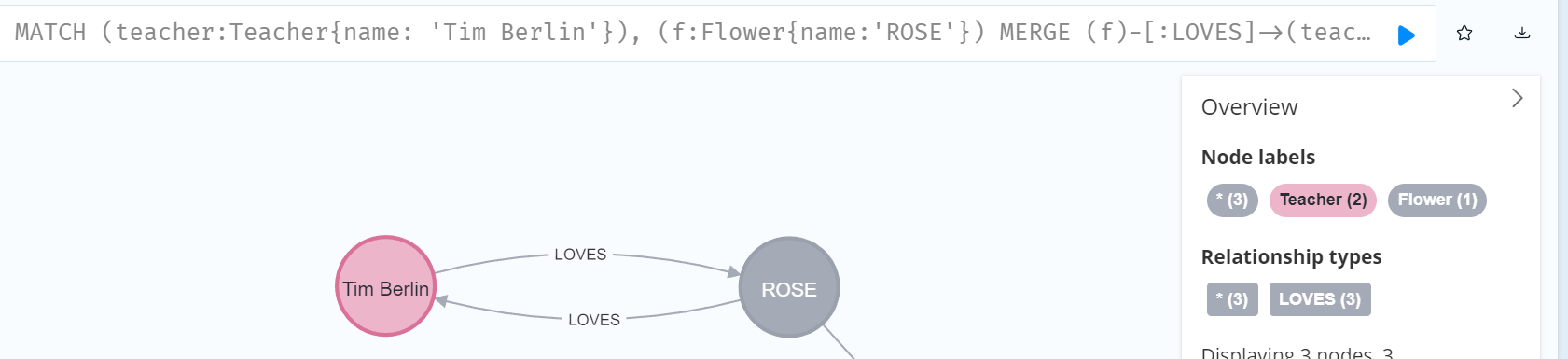
**RESULT:**

Before,



**After,**

Adds opposite direction relationship between that node.

****

**Analysis:**

The difference between case 5 and case 8:

* In case 5, existing nodes and relationship are updated with new direction of relationship.
* In case 8: Two nodes are searched and new relationship with opposite direction of relationship is added.

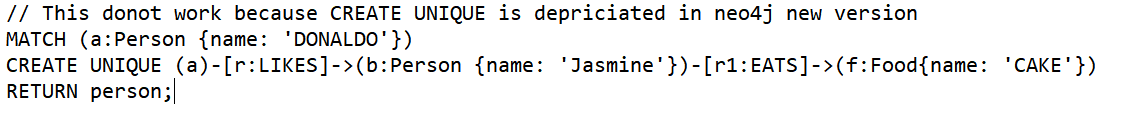
**Week 7:**

* Learned about the CREATE UNIQUE, it is mix of MATCH AND CREATE
* BUT in latest version of NEO4J CREATE UNIQUE does not works, MERGE can be used instead of CREATE UNIQUE
* Learned about FOREACH to update data with a list.

**Appendix:**

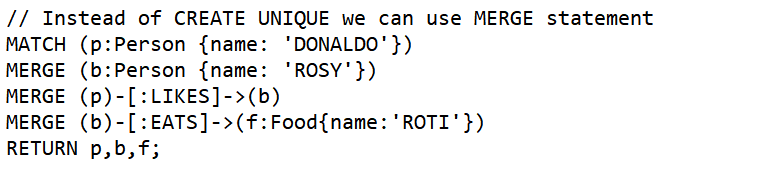
MERGE (person:Person {name: 'DONALDO'})

RETURN person;

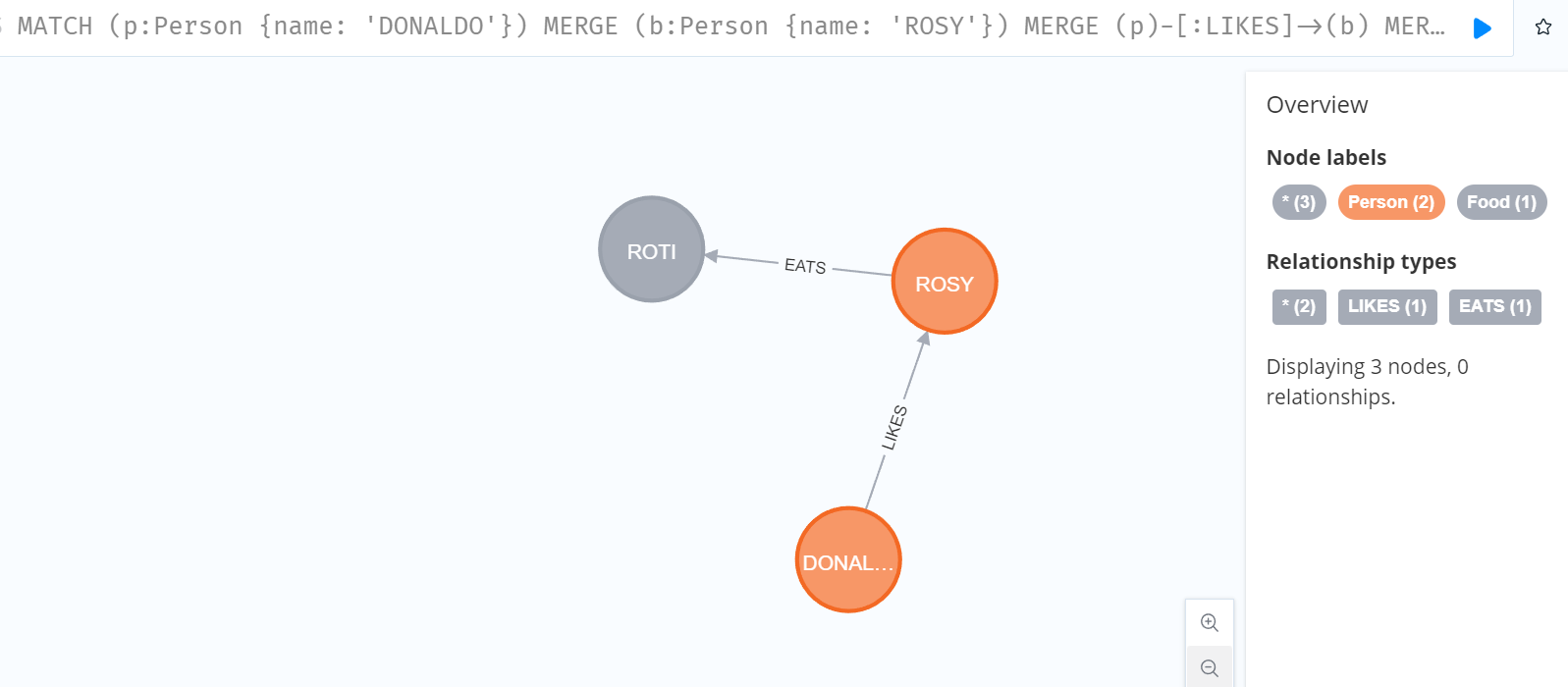


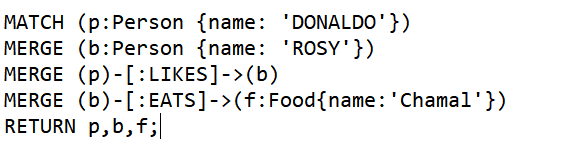
**RESULT:** CREATE UNIQUE is not recognized by earlier version of Neo4j.





**Result:** Works same as CREATE UNIQUE.





**Result:**



// This prevent duplication of node

MERGE (c:Cloth {name: "Suit Pant"})

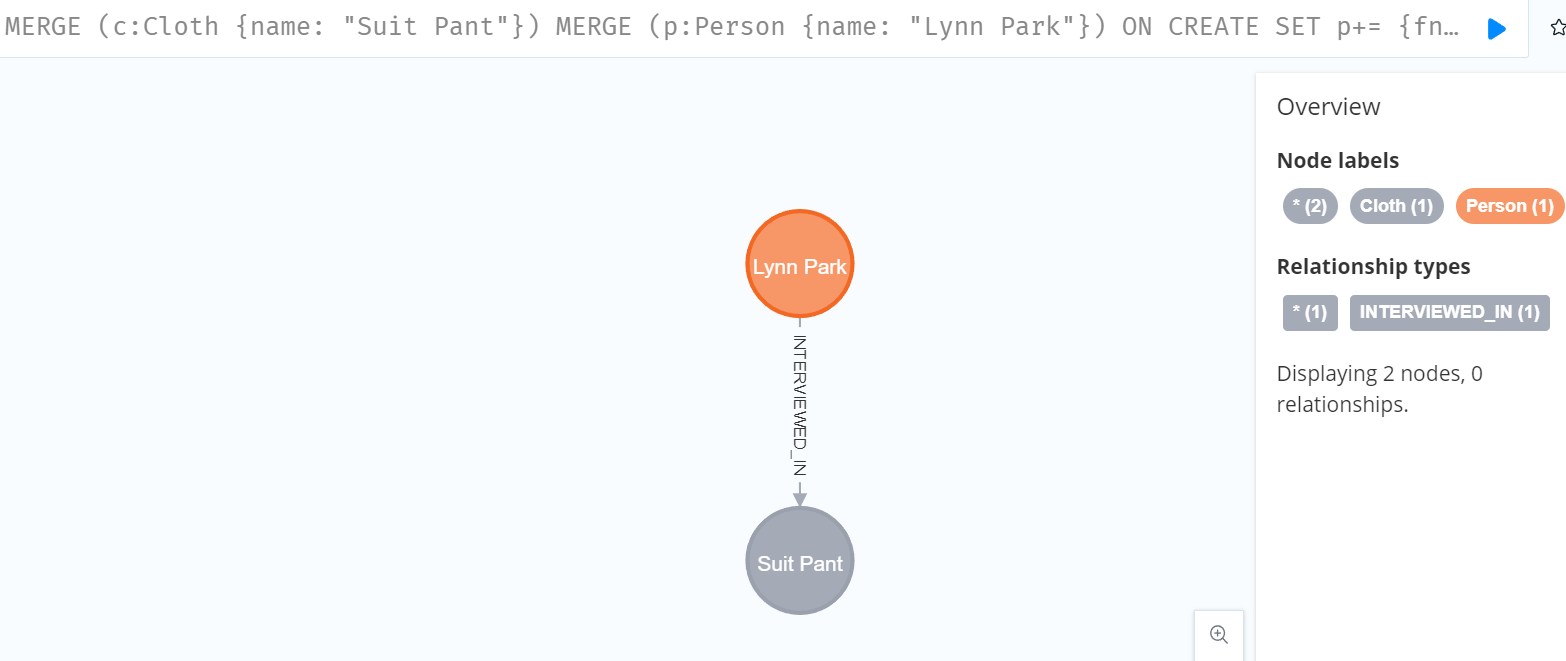
MERGE (p:Person {name: "Lynn Park"})

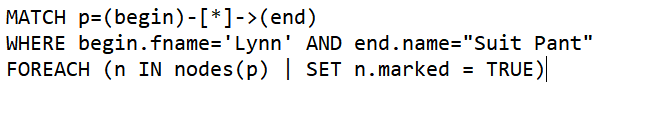
ON CREATE SET p+= {fname: "Lynn", lname: 'Park'}

MERGE (c) <-[:INTERVIEWED\_IN]-(p)

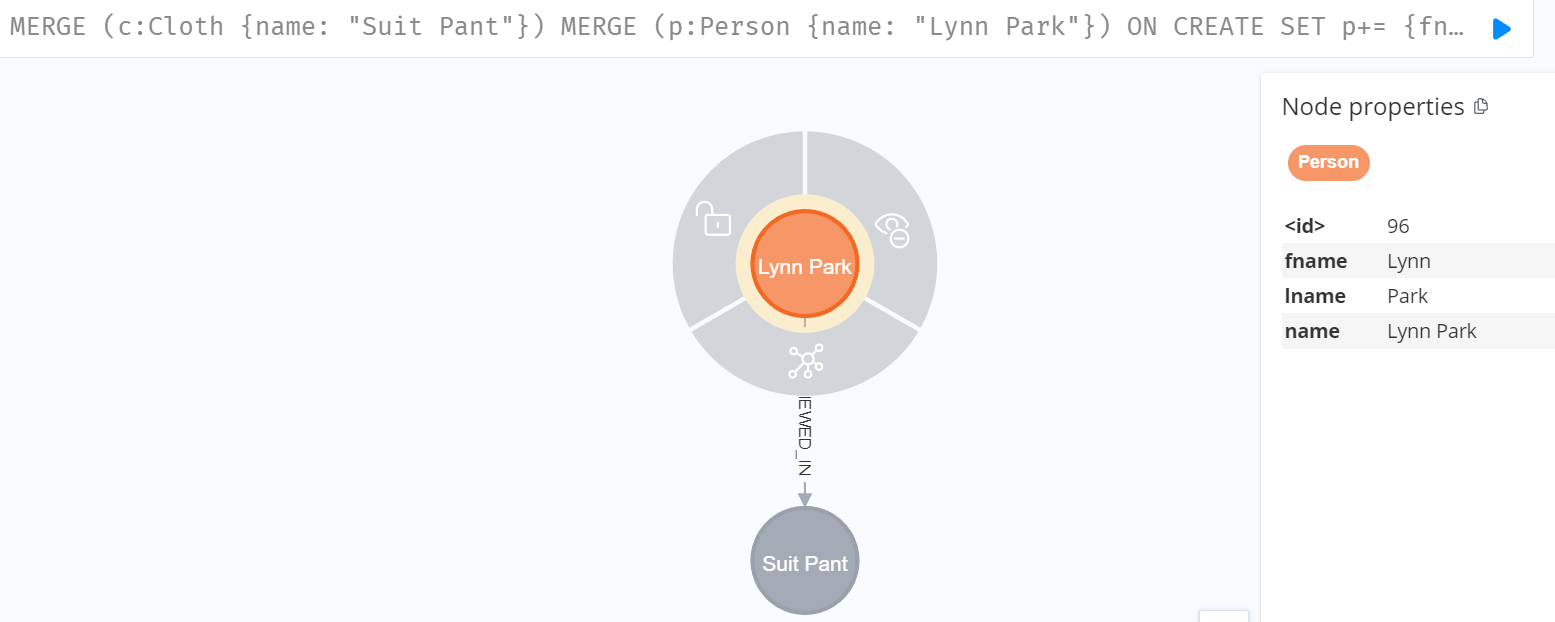
RETURN c,p;

**Result:** To prevent duplication, we can use MERGE statement.

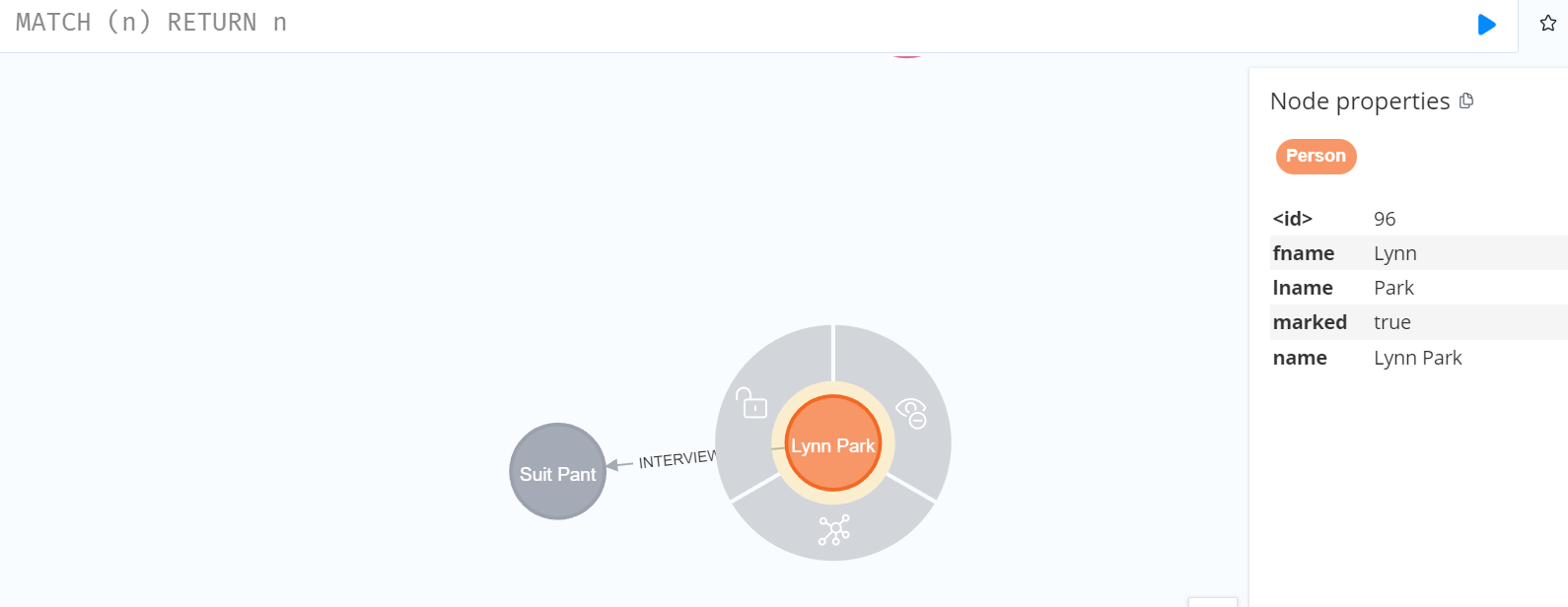




**Result:** Before no marked attribute is present.



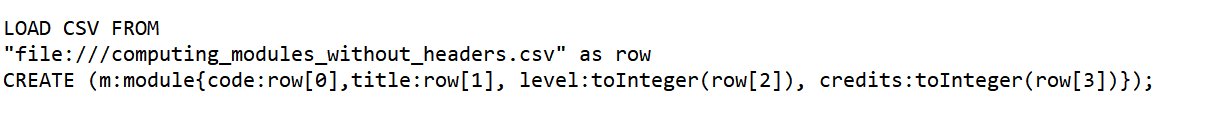
After running code, marked attribute is added.



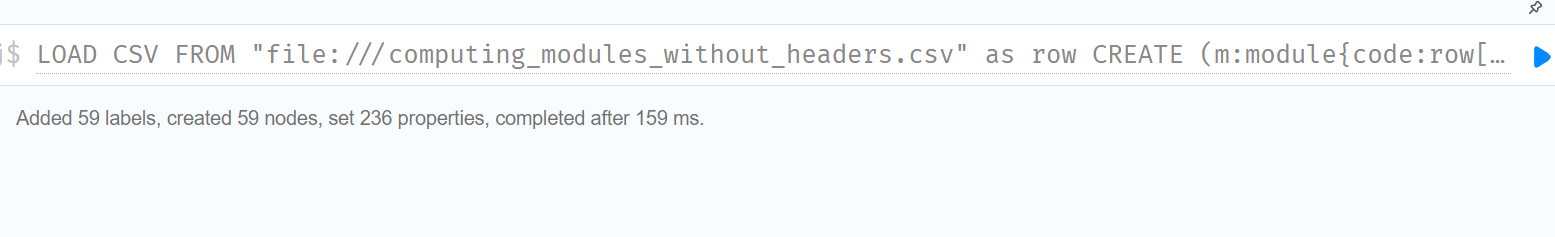
**Week 8:**

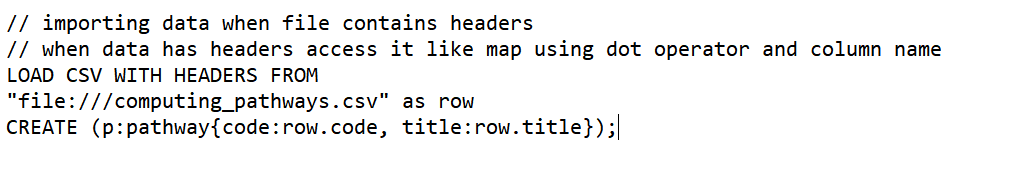
* I learned to load CSV files. Learned to load nodes from csv file with or without headers.
* Learned to load data from different online sources.
* Loaded movie database, learned to find shortest path between nodes.
* Learned about use of OPTIONAL MATCH, it returns NULL if no record is found.

**Appendix:**



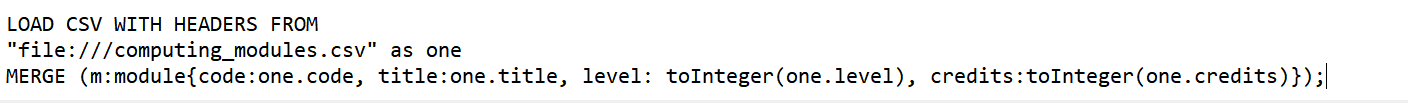
**Result: Successfully loaded from dataset without headers**

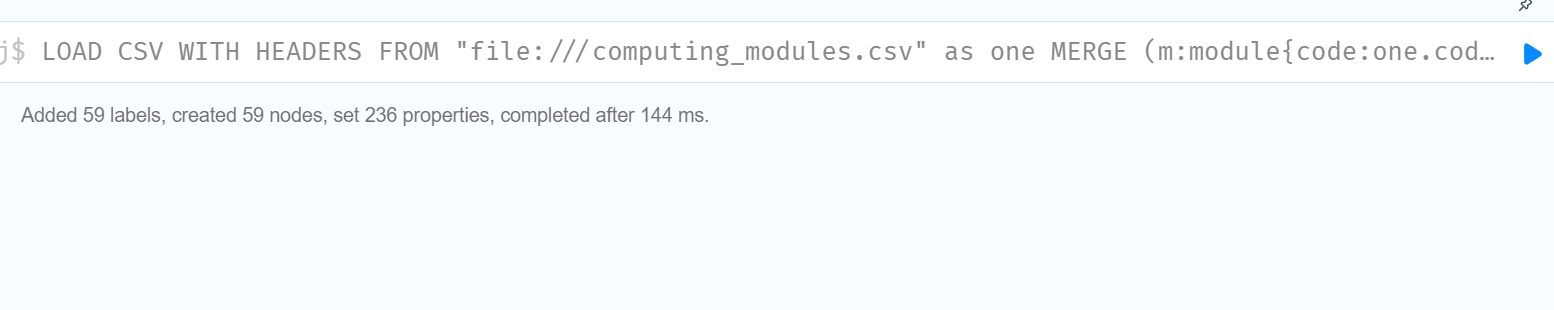




**Result:**

****



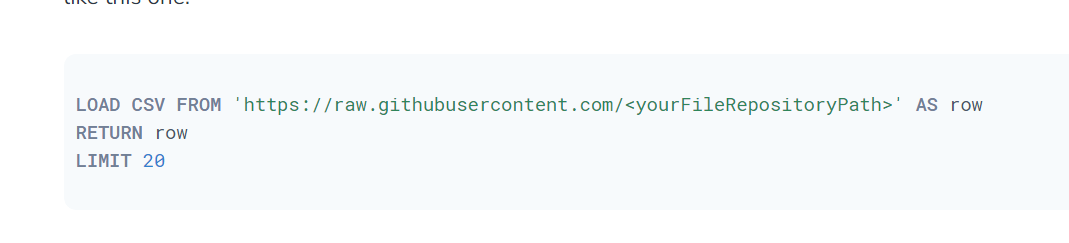
****

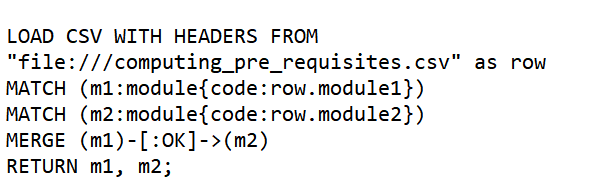
**We can load data from otherformats (LIKE XML)**

****

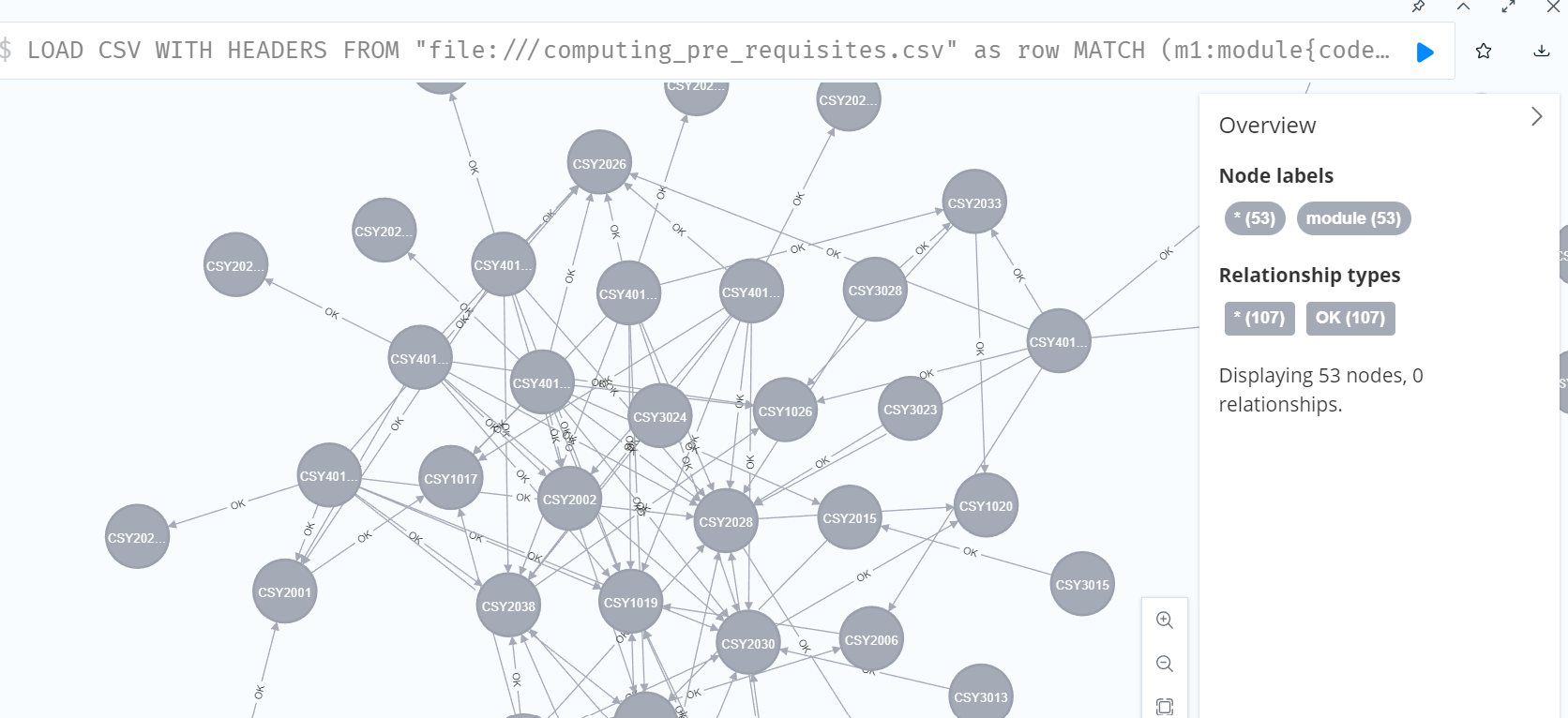
**LOAD DATA FROM ONLINE SOURCES(LIKE GITHUB):**

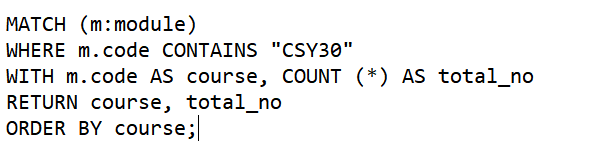
Just use url after FROM.

****

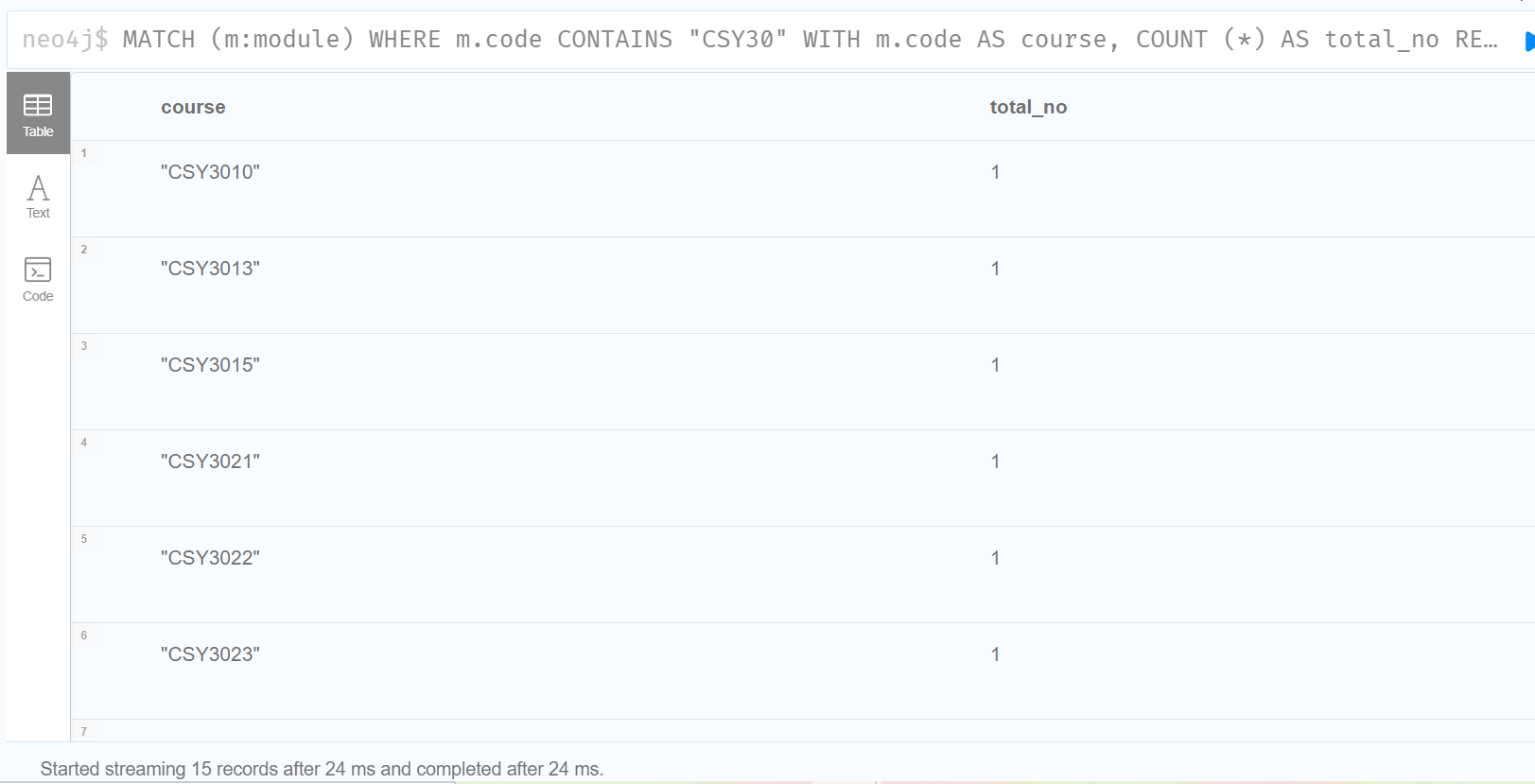


**Result:** OK relationship is added which is prerequisite to another module

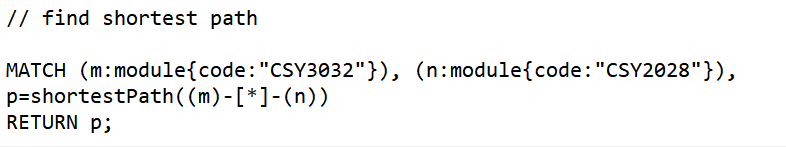
****



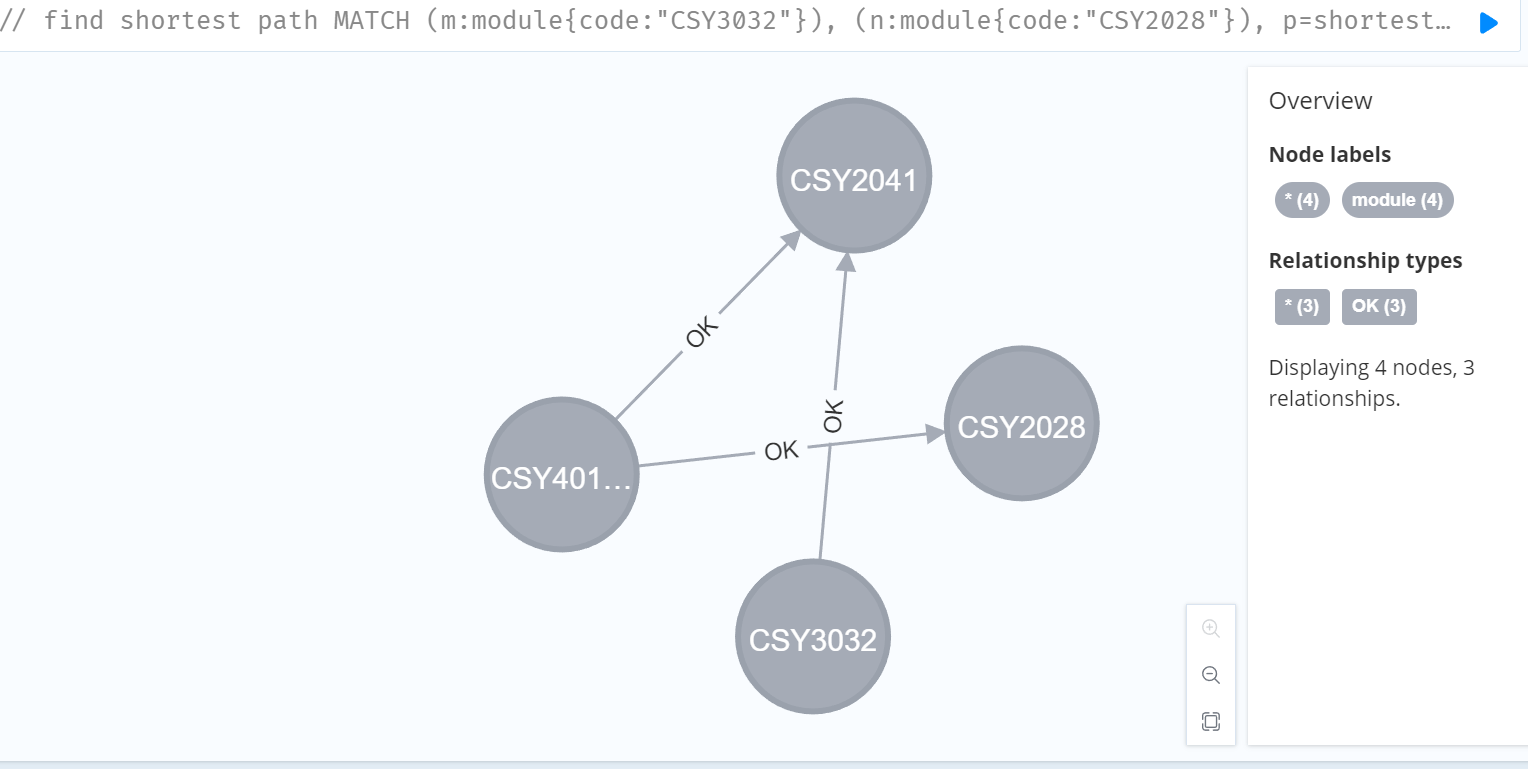
**Result:** Returns all modulde That contains CSY30

****

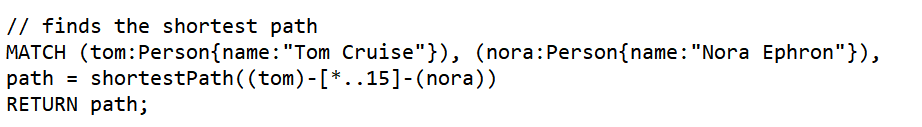
Finding Shortest Path:



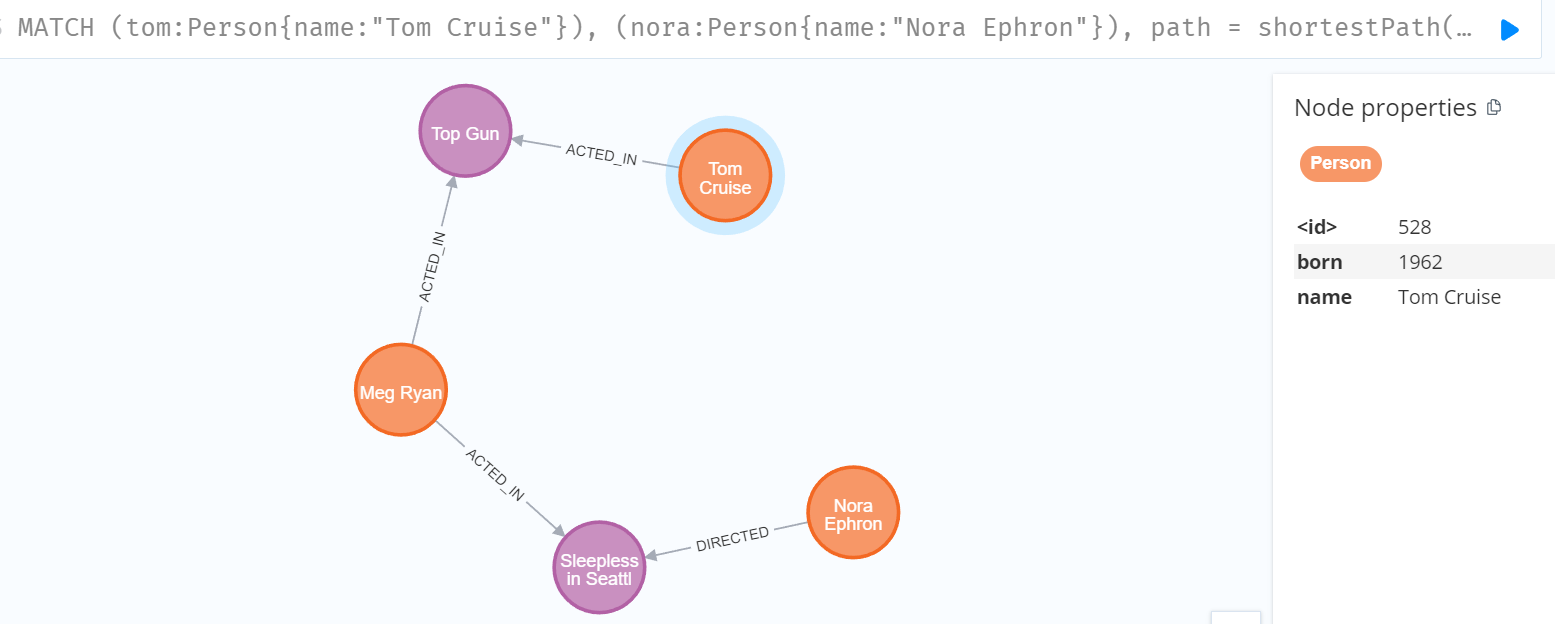
**Result:**

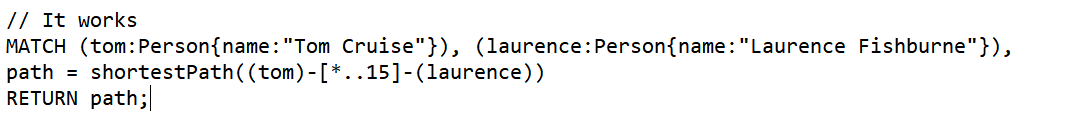


**Loading movie database and finding shortest Paths**

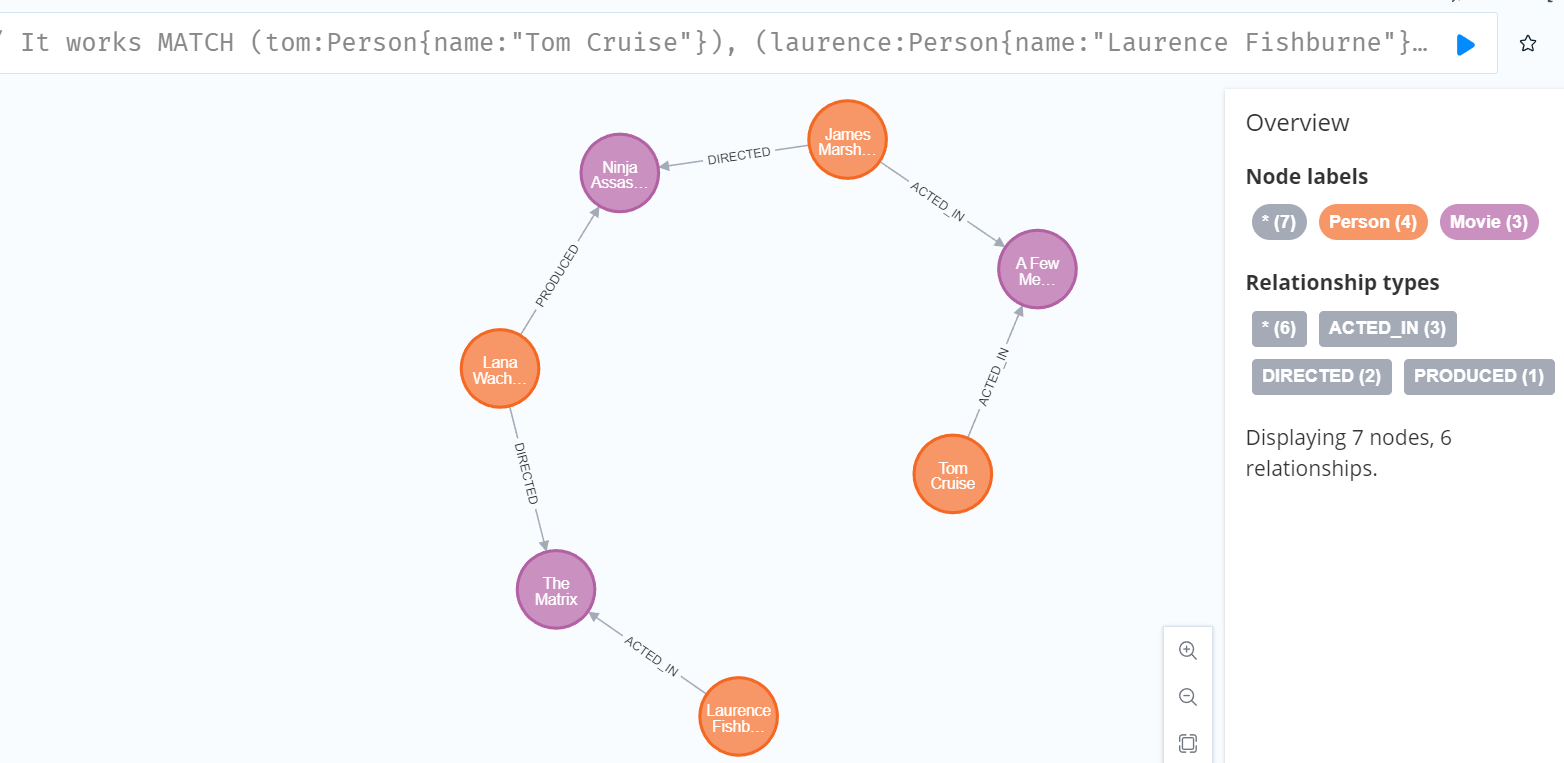


**Result: It finds shortest path upto 15 path long**

****

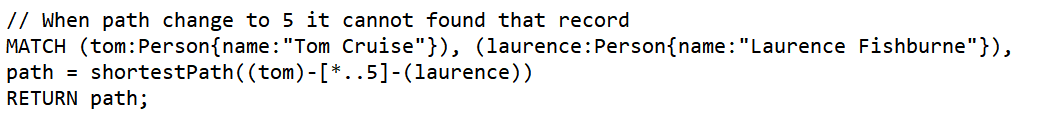


**Result:**

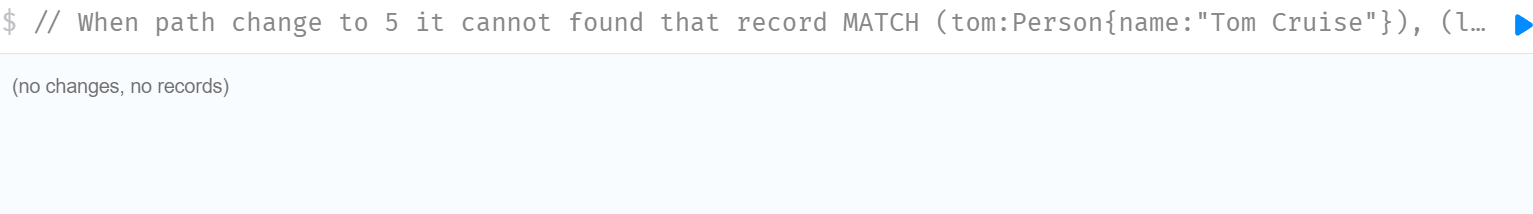
****

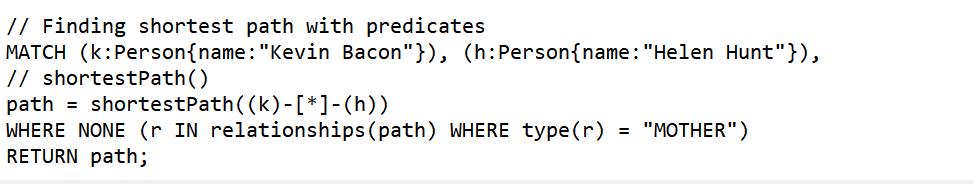
**When the path is change to 5 in above code.**

**=>**



**Result:**



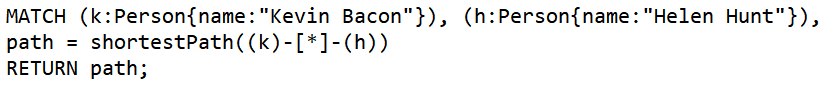


**Result:**

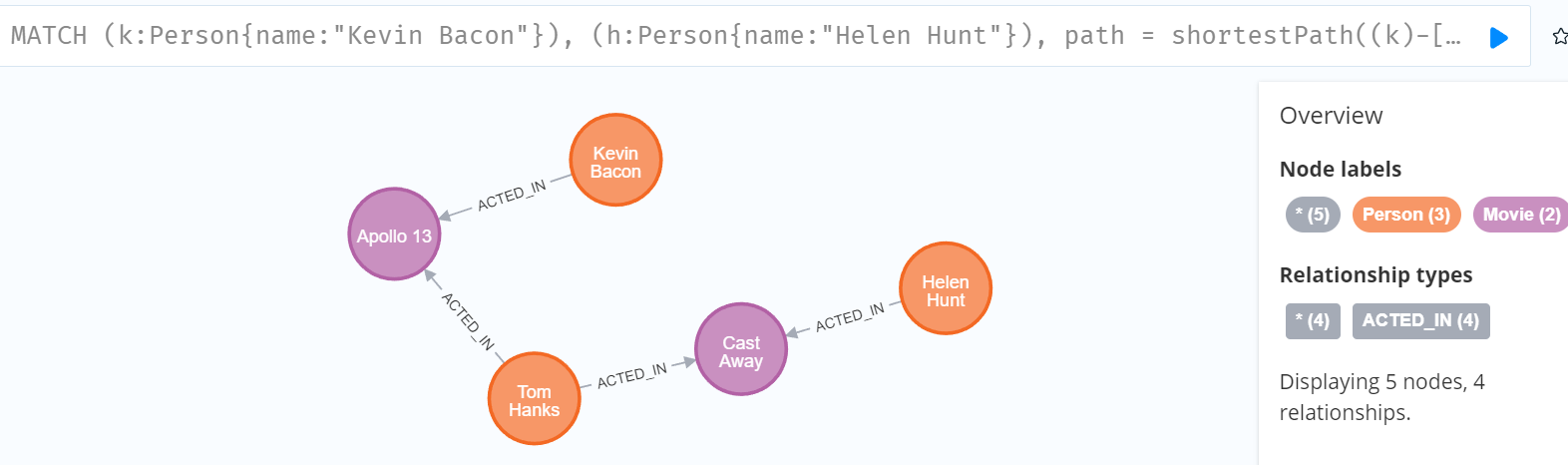


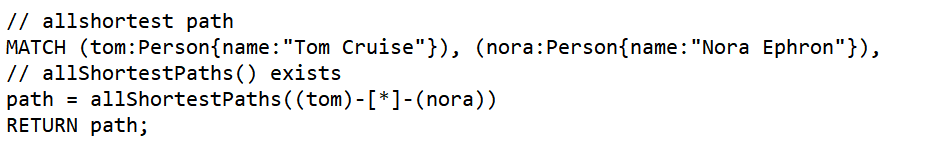
**// REMOVING NONE command, will give same result in this movie database**

**=>**

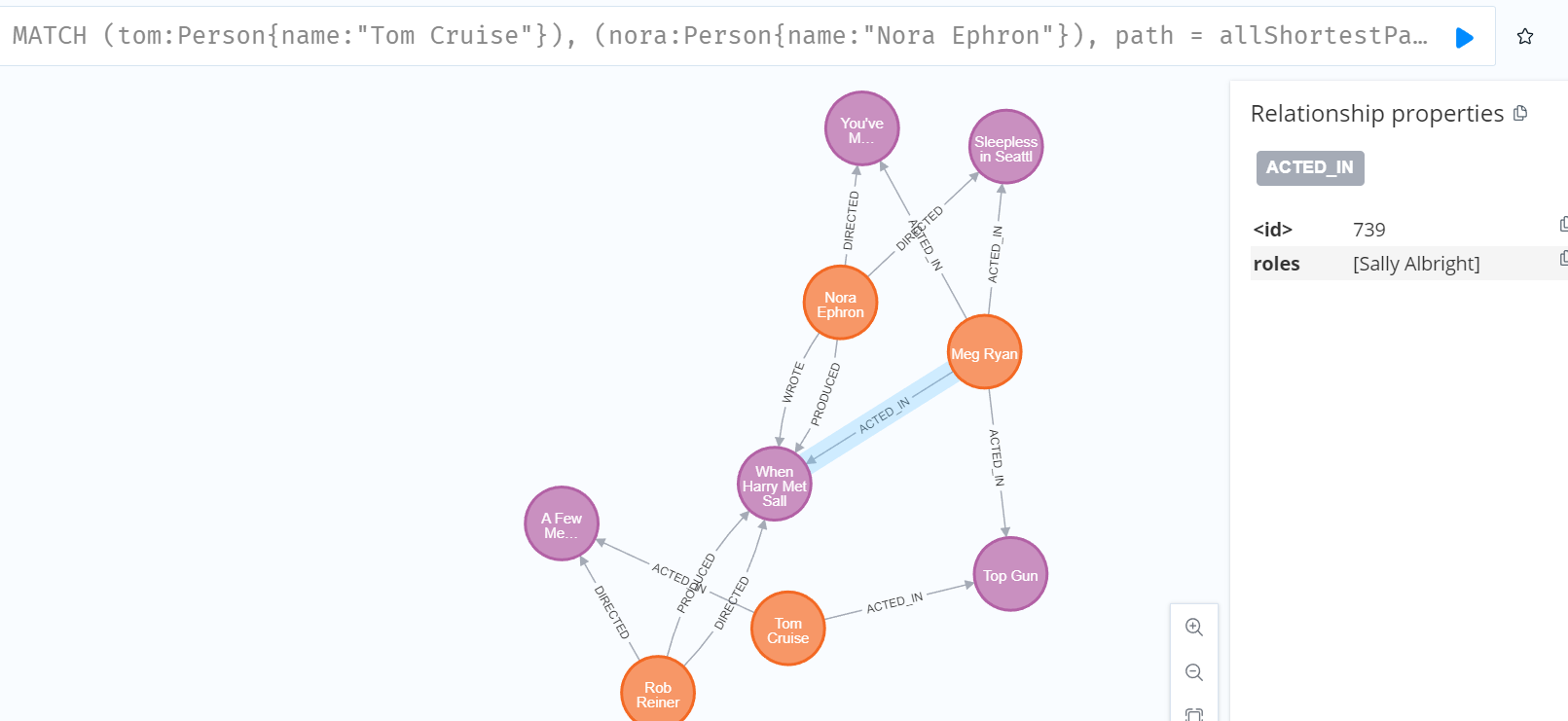


**Result:**





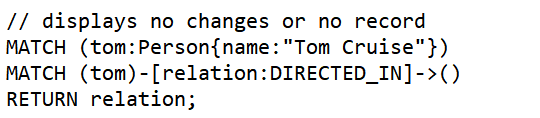
Result:

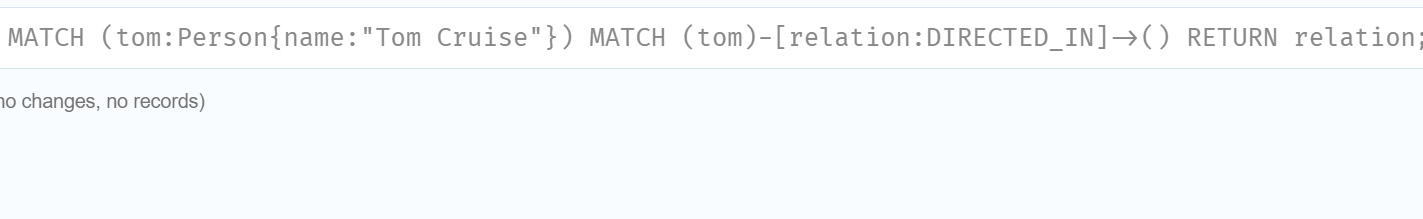


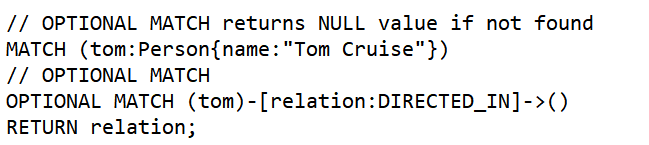


Result:





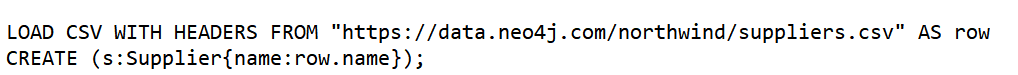




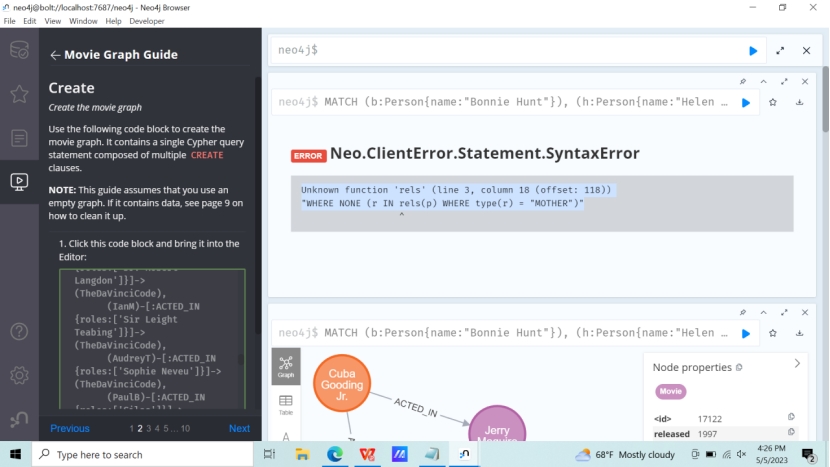
Result:



You can also load from online by just writing url in place of file name



**Problem arised:**



According to slide,

MATCH (b:Person{name:"Bonnie Hunt"}), (h:Person{name:"Helen Hunt"}),

p = shortestPath((b)-[\*]-(h))

WHERE NONE (r IN rels(p) WHERE type(r) = "MOTHER")

RETURN p;

rels() was given in our lecture slide, which showed above error. This error was solved by using relationships() instead of rels().

**Week 9:**

**Appendix:**

MERGE (p:Person {name: 'DONALDO', gender: 'M', age: 29, status: 'Unmarried'})

MERGE (b:Person {name: 'ROSY', gender: 'F', age: 27})

MERGE (c:Person {name: 'HALLY', gender: 'F', age: 22})

MERGE (p)-[:LIKES{since: 2018}]->(b)

MERGE (p)-[:TEACHES{since: 2015}]->(c)

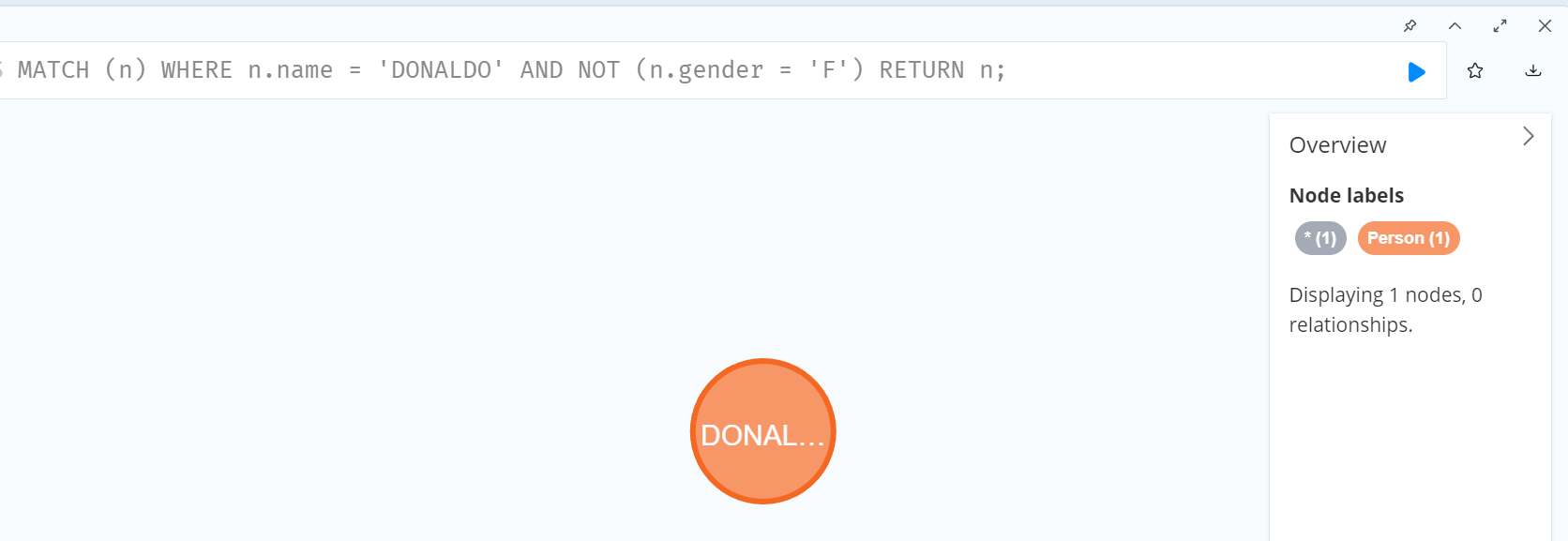
MERGE (b)-[:EATS{type: 'HEALTHY'}]->(f:Food{name:'ROTI'})

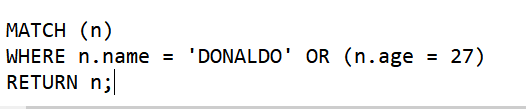
RETURN p,b;

//Boolean Operation

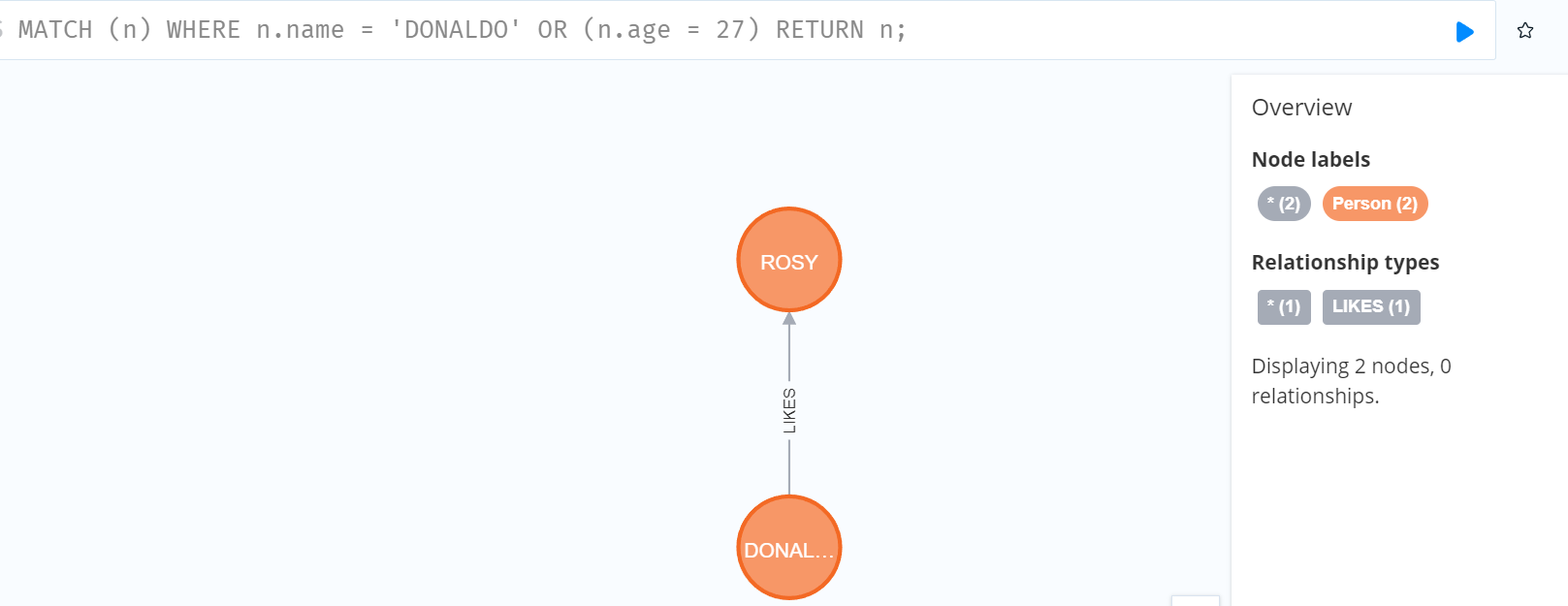


**Result:**



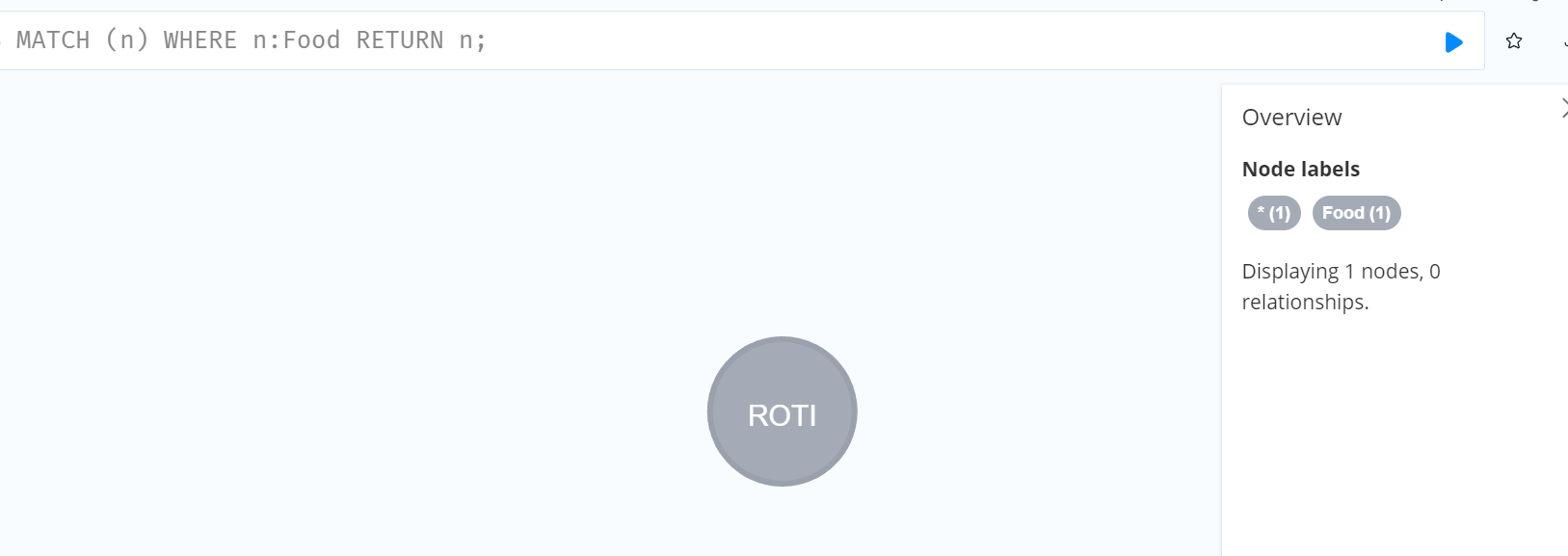


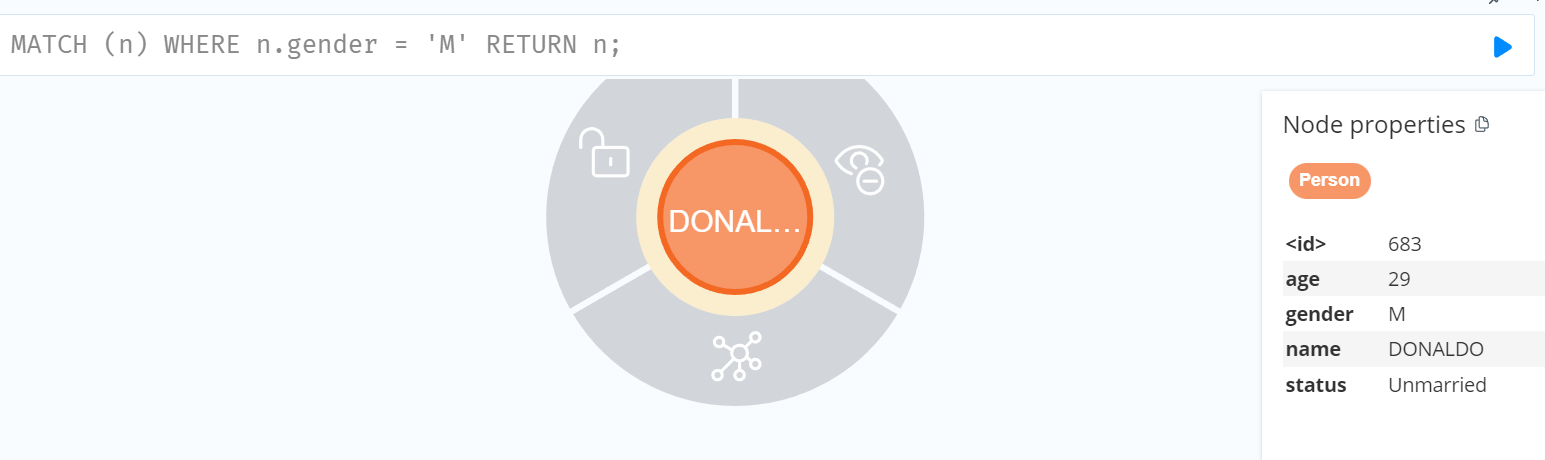
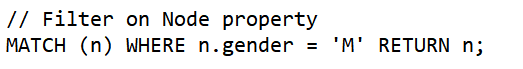
**Result:**



// Filter On Node label

MATCH (n) WHERE n:Food RETURN n;



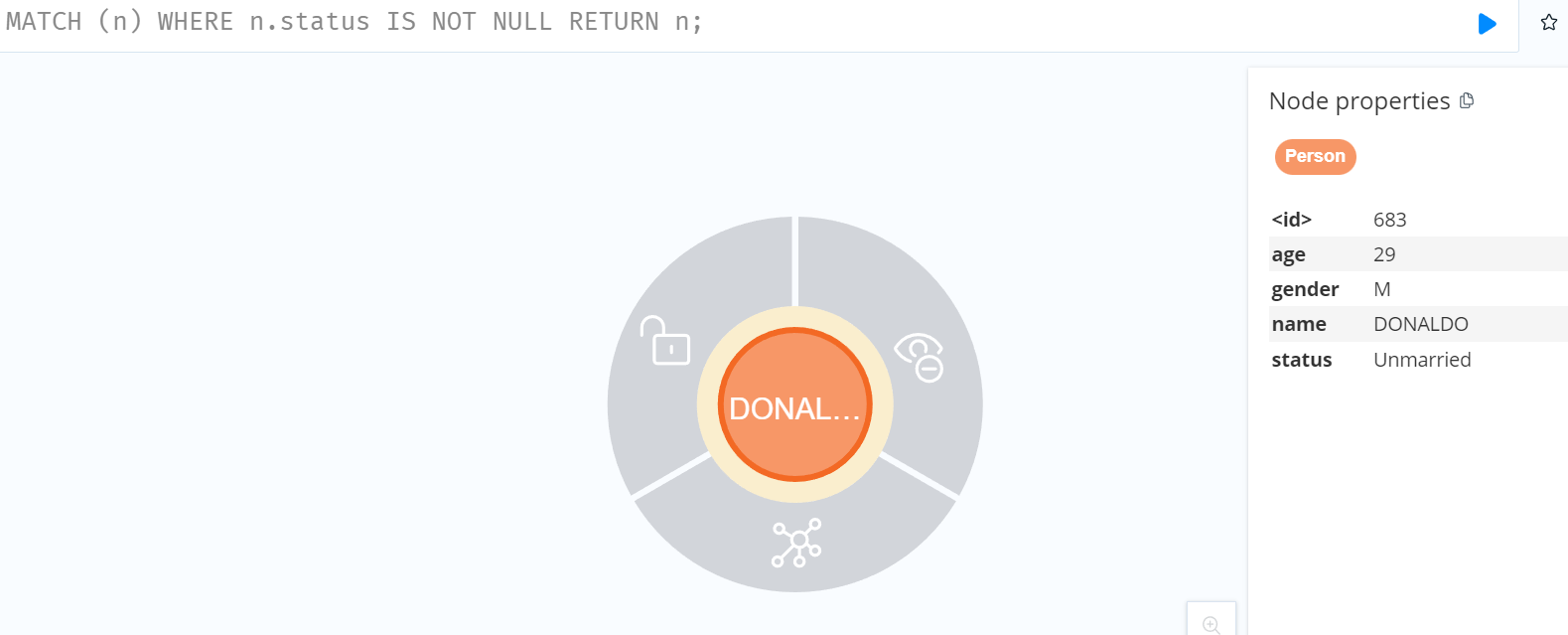
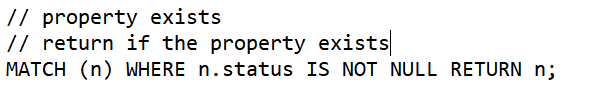


// Filter on relationship property

MATCH (n)-[e:EATS]->(food)

WHERE e.type = 'HEALTHY' RETURN food;



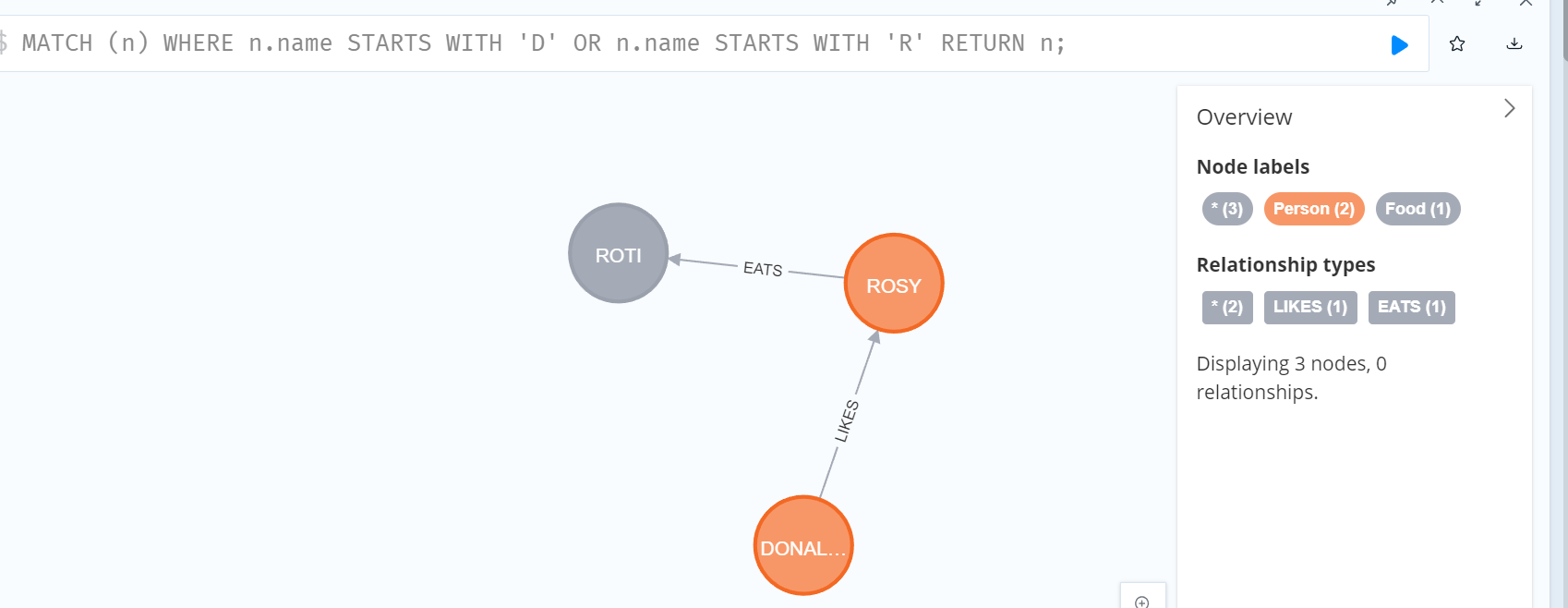


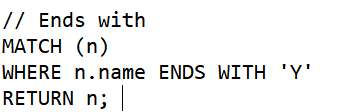
// Start of String

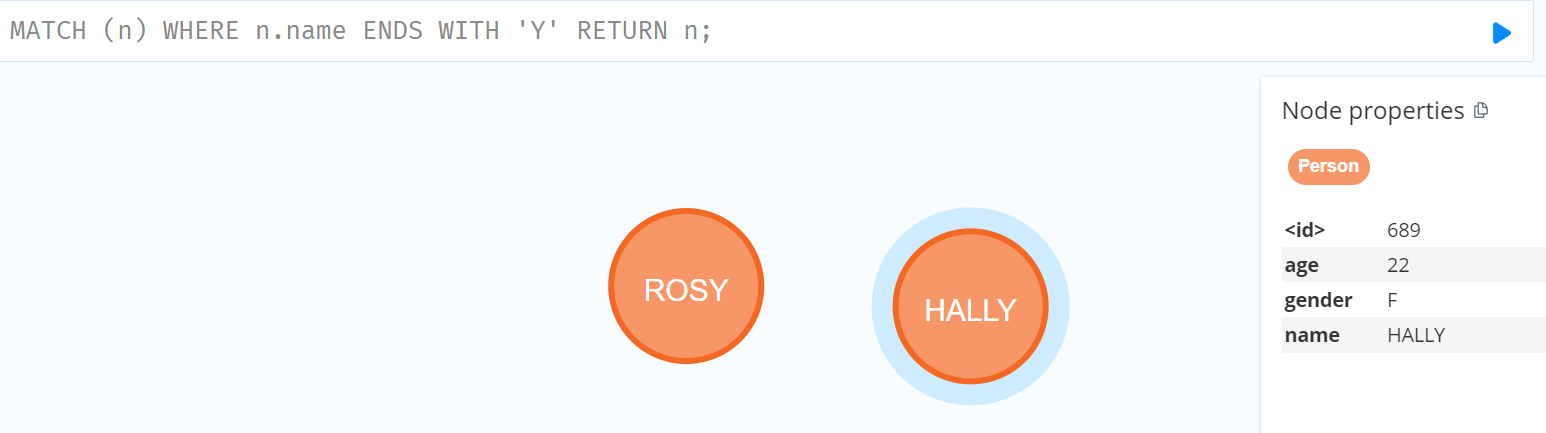
MATCH (n)

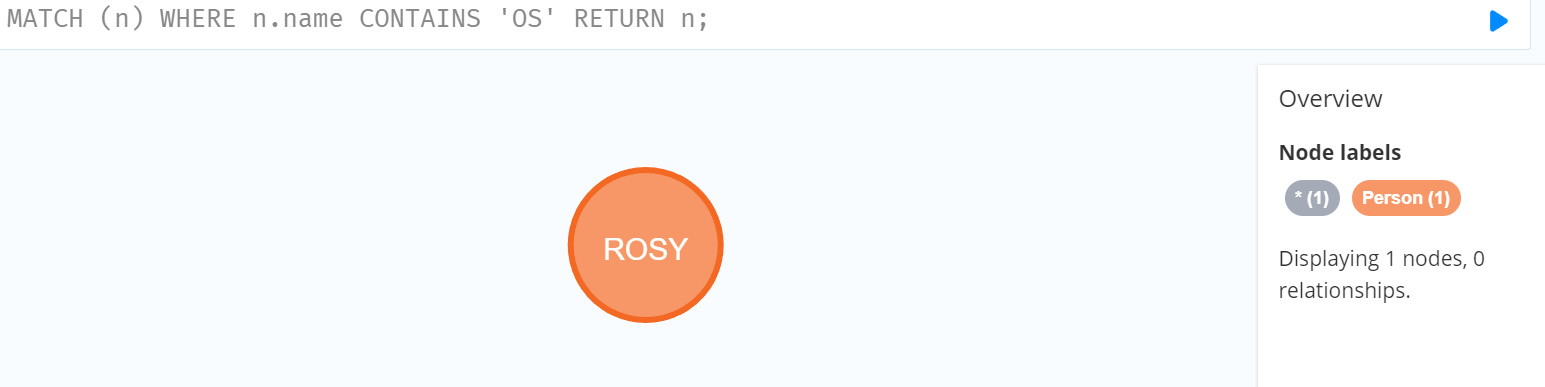
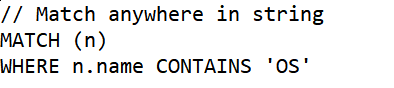
WHERE n.name STARTS WITH 'D' OR n.name STARTS WITH 'R'

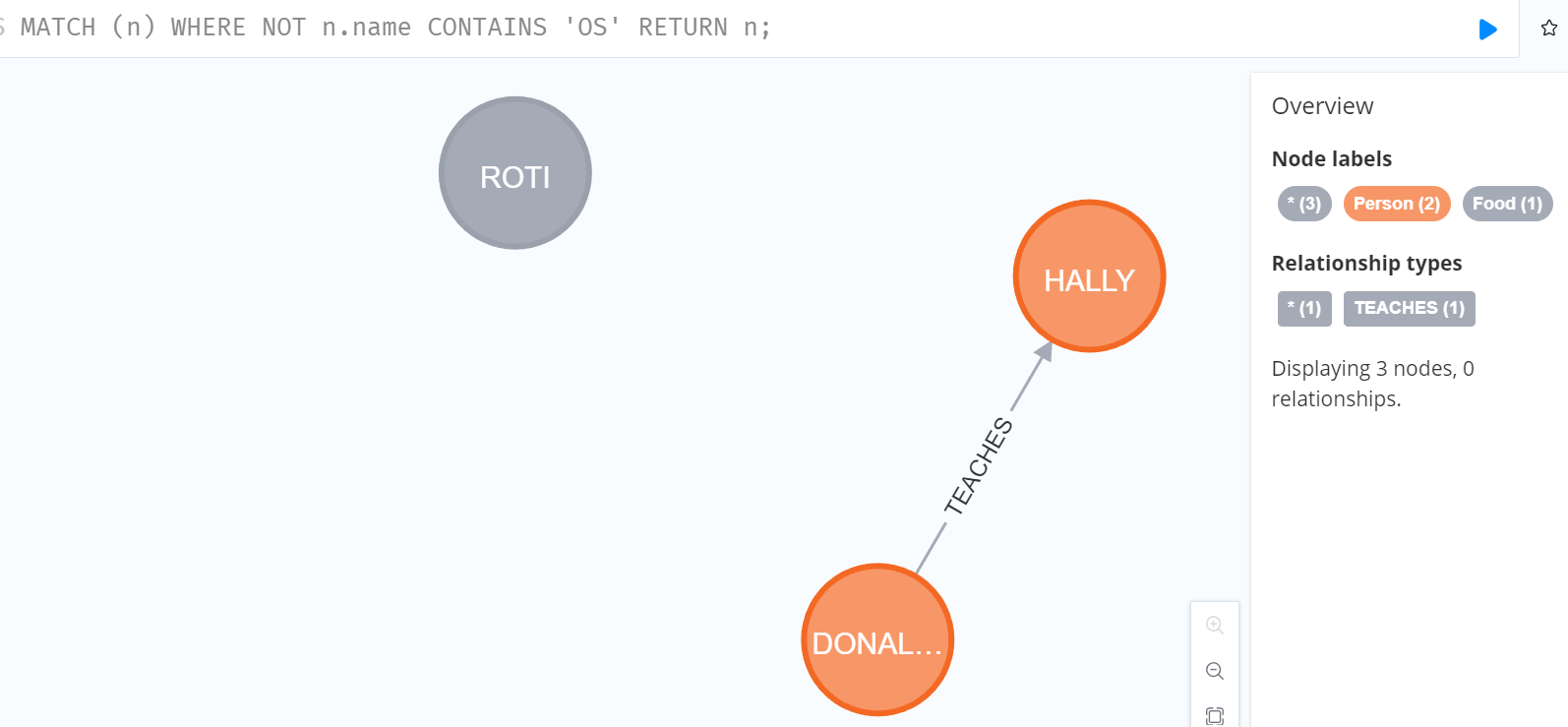
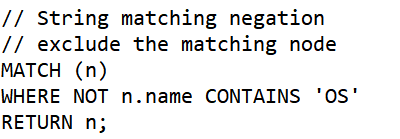
RETURN n;

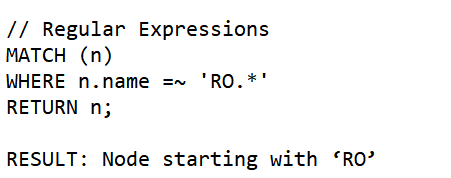




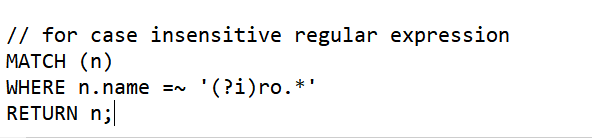




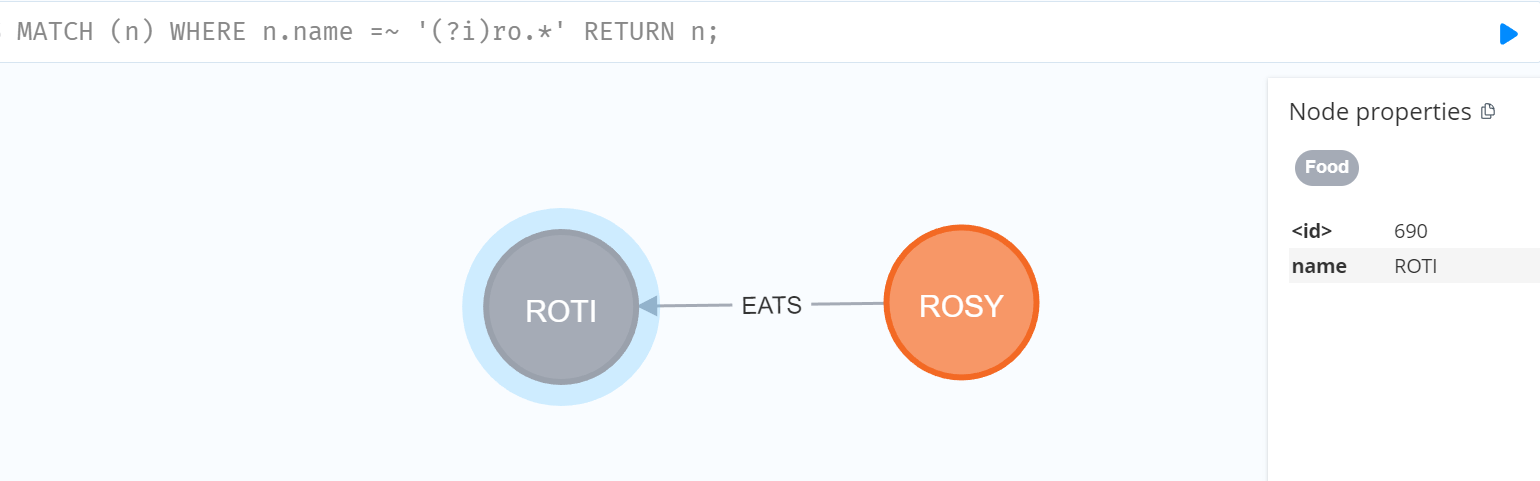








Result: gives case insensitive expressions.



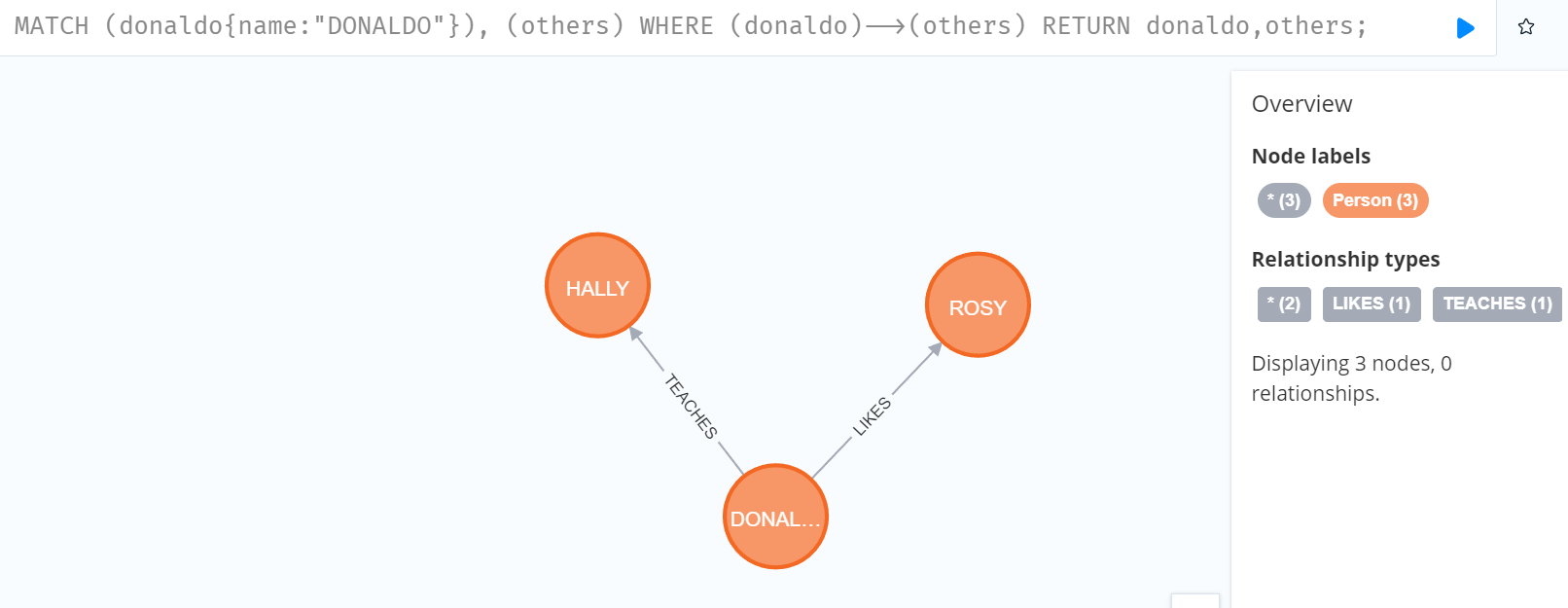
// Filter using patterns in where

// This return all node linked by donaldo

MATCH (donaldo{name:"DONALDO"}), (others)

WHERE (donaldo)-->(others)

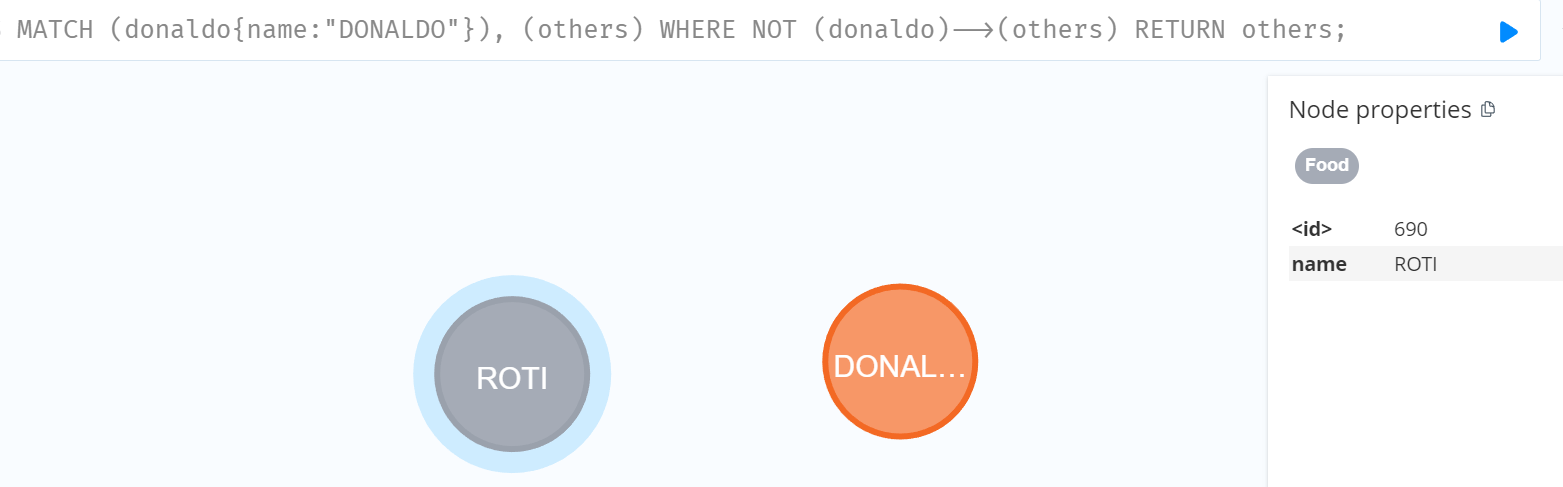
RETURN donaldo,others;



MATCH (donaldo{name:"DONALDO"}), (others)

WHERE NOT (donaldo)-->(others)

RETURN others;



// filter on relationship type

// using special property type

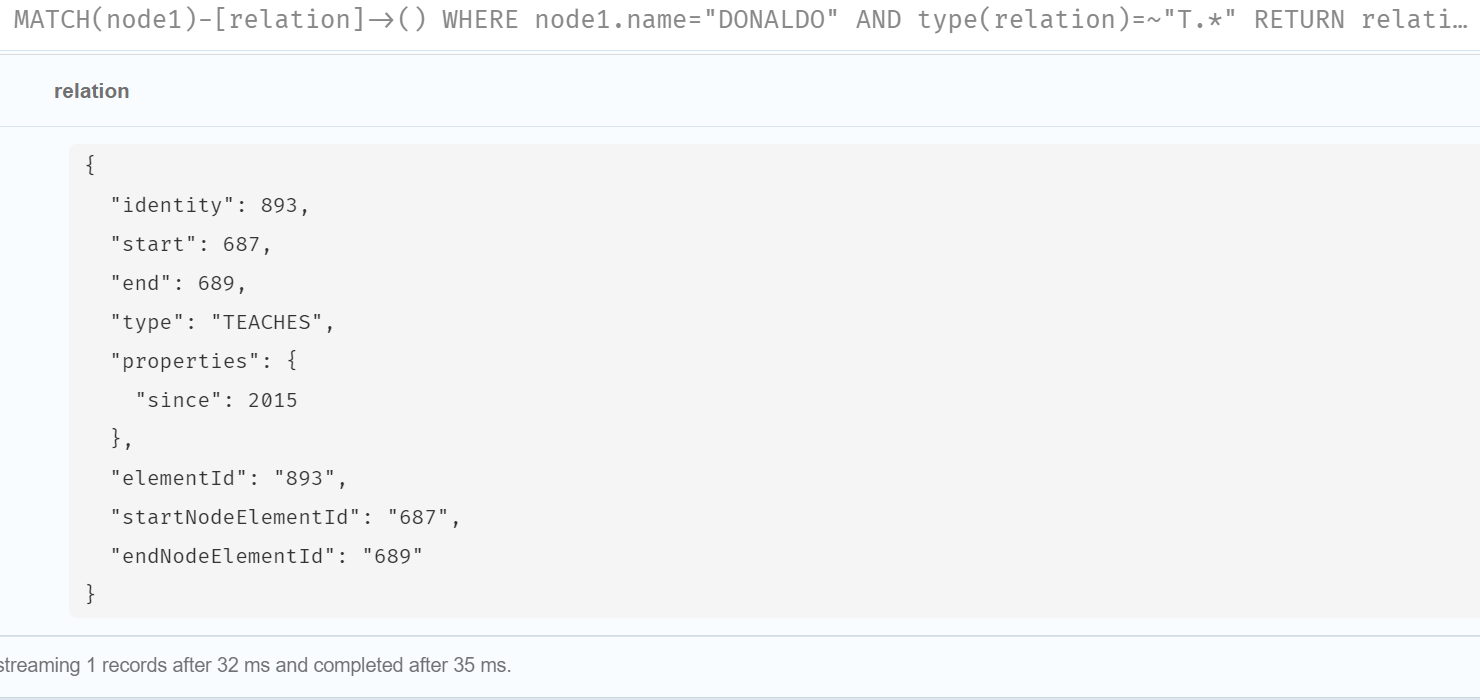
// RETURN all the relation of DONALDO which starts with 'T'

MATCH(node1)-[relation]->()

WHERE node1.name="DONALDO" AND type(relation)=~"T.\*"

RETURN relation;

//Here type() function is used to track relationship



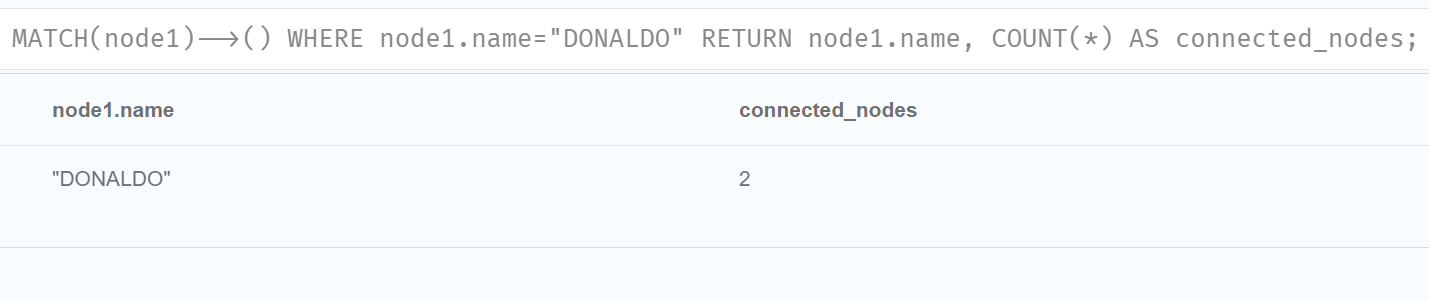
// COUNT

// count no of node connected to it

MATCH(node1)-->()

WHERE node1.name="DONALDO"

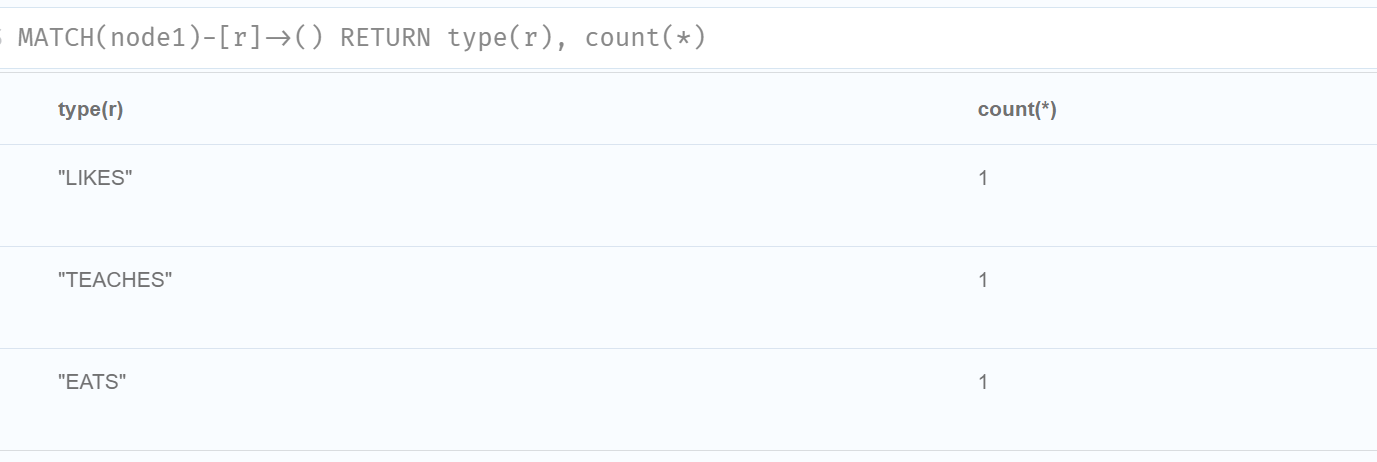
RETURN node1.name, COUNT(\*) AS connected\_nodes;



// Count groups of relationship types

MATCH(node1)-[r]->()

RETURN type(r), count(\*)

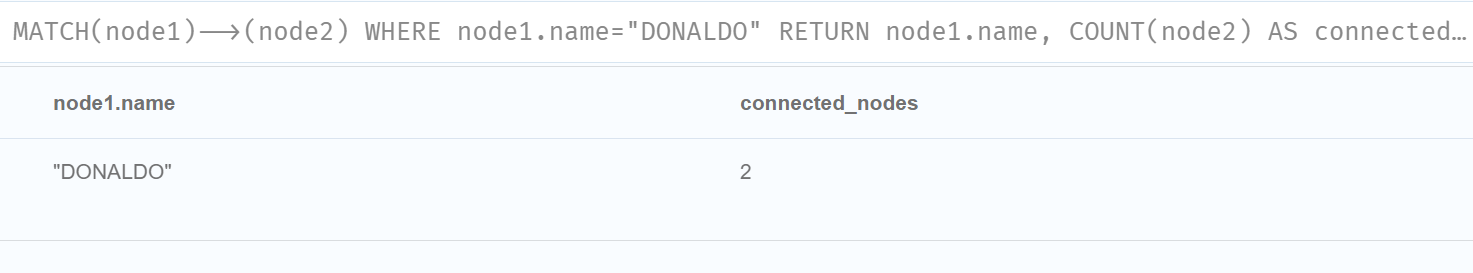


// count entities

MATCH(node1)-->(node2)

WHERE node1.name="DONALDO"

RETURN node1.name, COUNT(node2) AS connected\_nodes;



// count no null values of node.

// It counts total no of nodes that has following property

MATCH(n:Person)

RETURN COUNT(n.gender);



**NOTE:**

Exists function no longer supported in latest version of Neo4j. USE IS NOT NULL instead of EXISTS.

