

# AT82.02

## DATA MODELING AND MANAGEMENT

---

### LAB8: NEO4J - GRAPH MODEL

# Outline

---

- Neo4J Graph database Overview

- CRUD Operations

- 1. CREATE

- 1.1 Create Node

- 1.2. Create many Nodes and Relationships at once.

- 2. QUERY

- 2.1 Basic Query

- 2.2 Make Recommendations

- 2.3 Aggregate

- 3. UPDATE

- 3.1 Update Property of Node or Relationship

- 3.2 Update Label

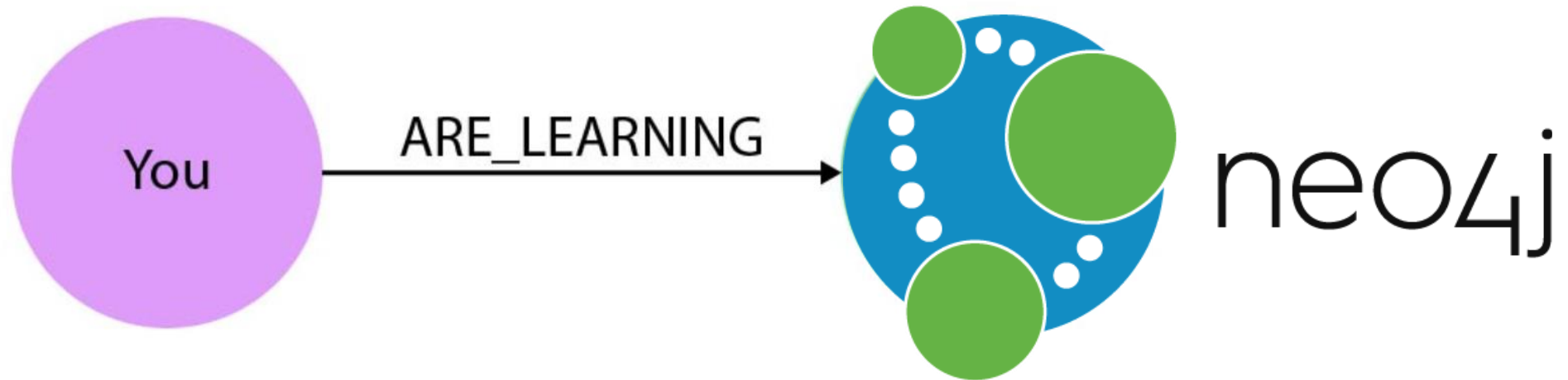
- 4. DELETE

- 4.1 Delete a specific node

- 4.2 Delete a specific relationship

- 4.3 Remove Label from a node

- 4.4 Remove a property





- ◎ Neo4j is an open-source, NoSQL graph database.
- ◎ **Property Graph** data model
- ◎ **Cypher Graph** query language



# Property Graph Model

## Nodes

- Represent the objects in the graph
- Can be *labeled*

## Relationships

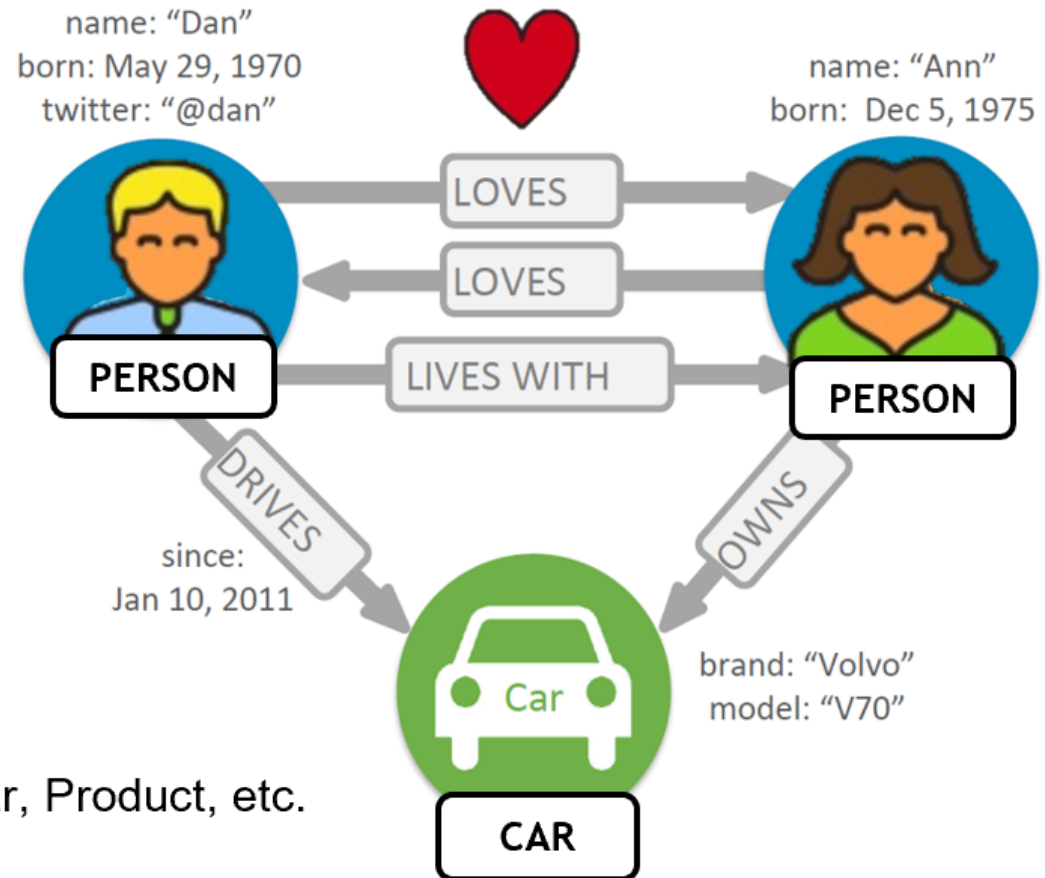
- Relate nodes by *type* and *direction*

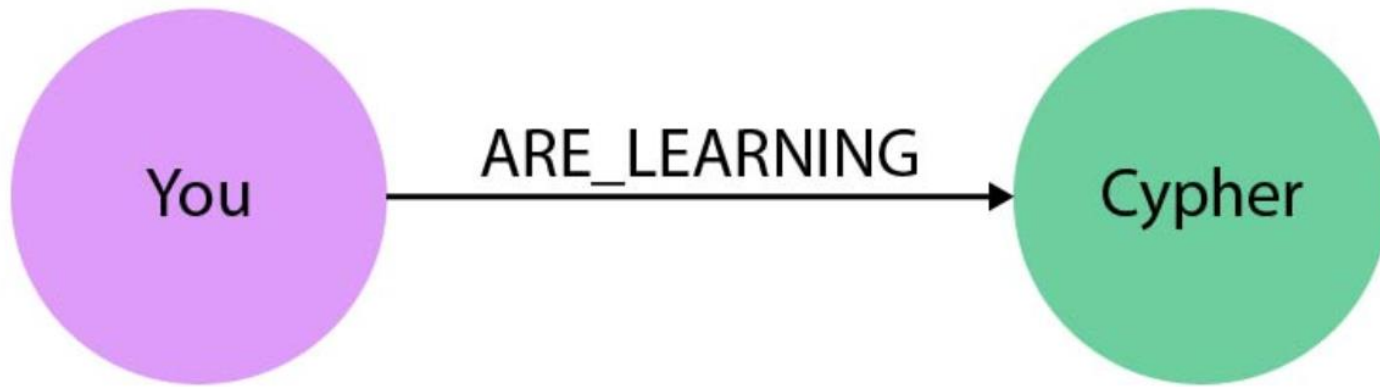
## Properties

- Name-value pairs that can go on nodes and relationships.

## Label

- Labels describe the types of data.
- These are typically nouns like Person, Car, Product, etc.
- Associate a set of nodes.
- A node can have zero or more labels
- Labels do not have any properties



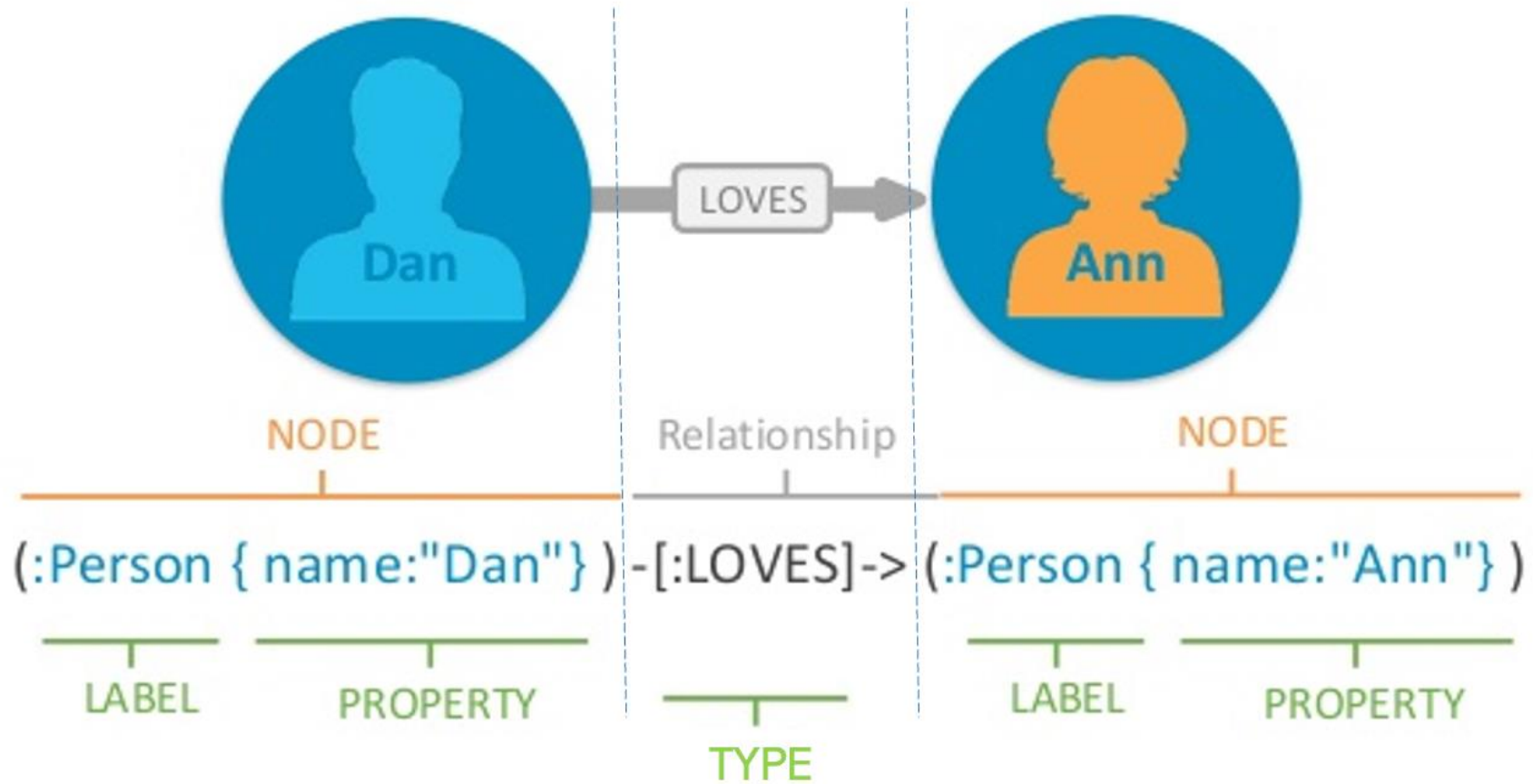


# **Cypher** - Neo4j's graph query language

---

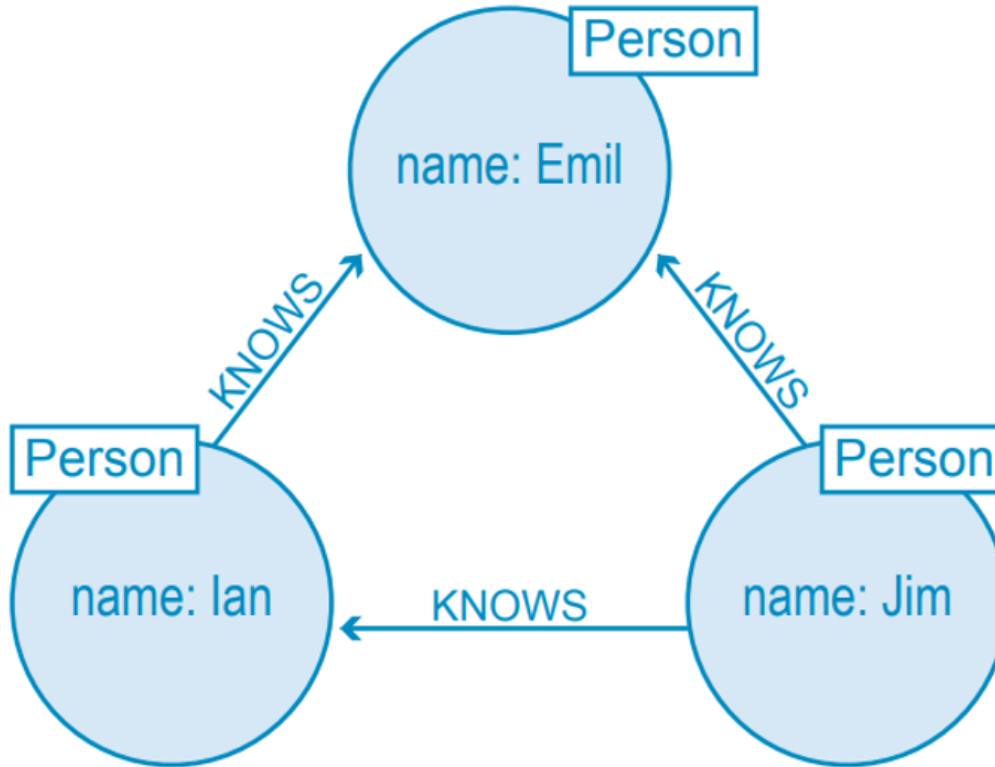
- ◎ uses patterns to describe graph data
- ◎ familiar SQL-like clauses
- ◎ declarative, describing what to find, not how to find it

# Pattern





**Example** MATCH (a:Person {name:'Jim'})-[:KNOWS]->(b)-[:KNOWS]->(c),  
(a)-[:KNOWS]->(c)  
RETURN b, c



# CRUD operations

---

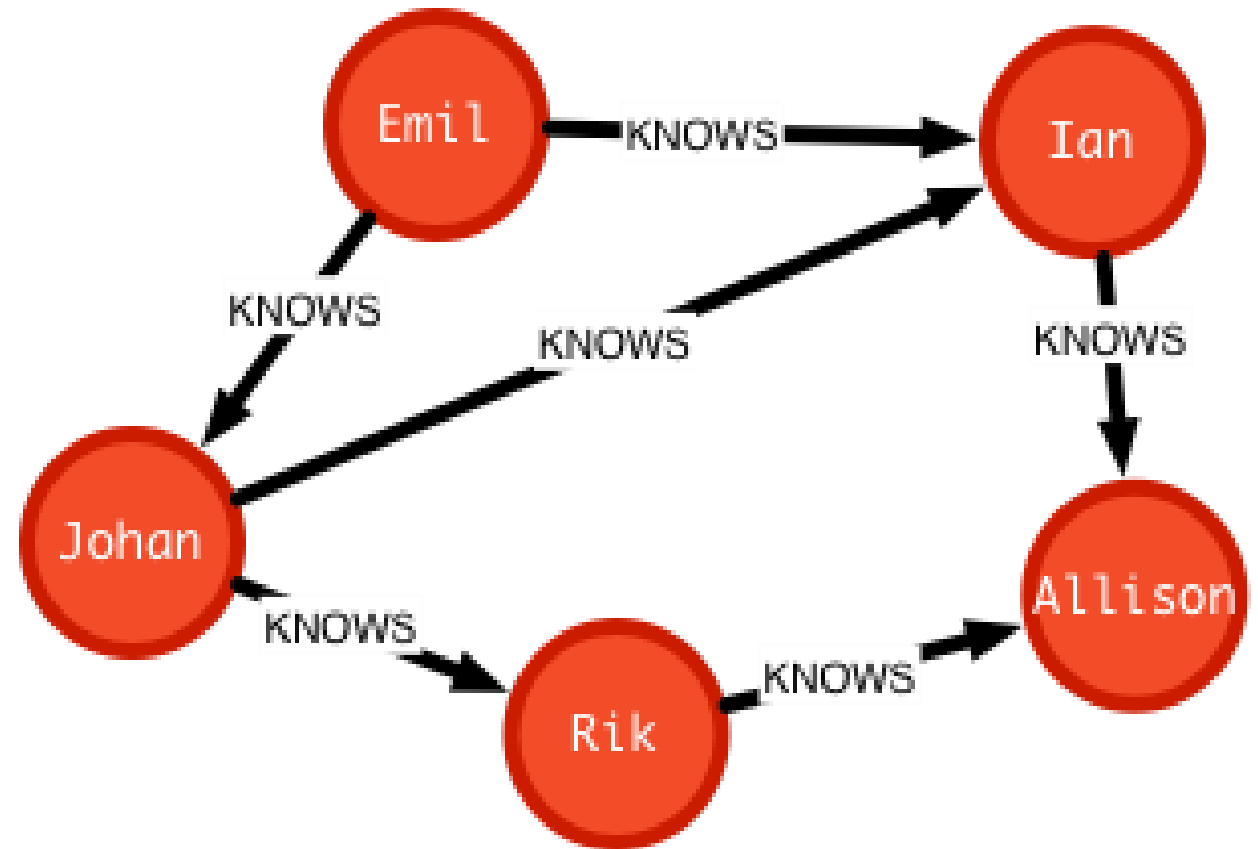
1. Create/Insert
2. Read/Query
3. Update
4. Delete

# Scenario

---

## Small Social Network Graph

- Emil KNOWS Johan and Ian
- Johan KNOWS Ian and Rik
- Rik and Ian KNOWS Allison



# 1. CREATE

---

## 1.1 Create Node

Let's use Cypher to generate a small social graph.

```
▶ CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

- **CREATE** clause to create data
- **()** parenthesis to indicate a node
- **ee:Person** a variable 'ee' and label 'Person' for the new node
- **{}** brackets to add properties to the node

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })  
Return ee
```

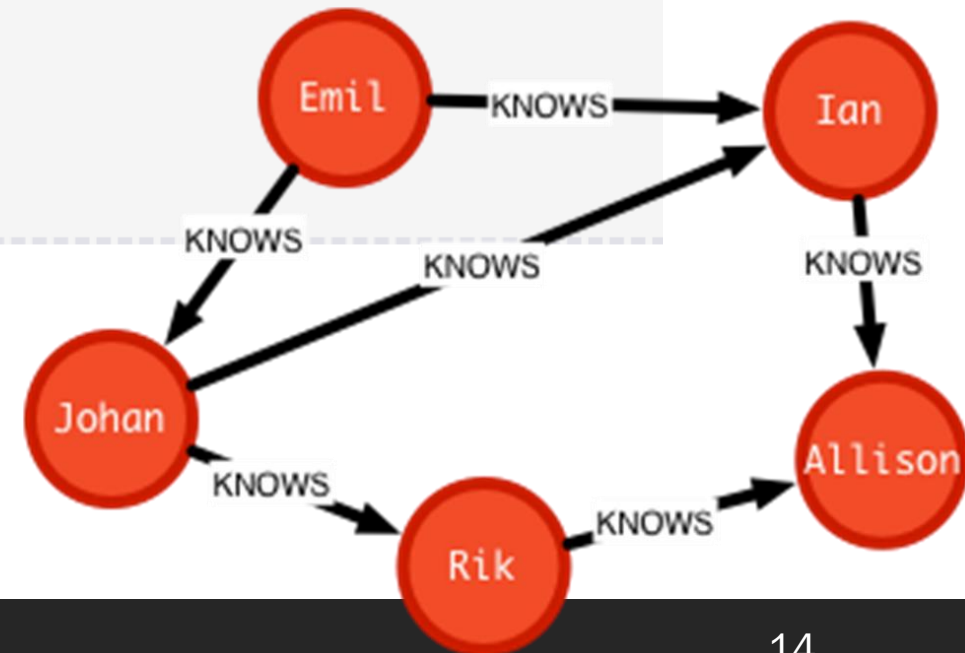
The screenshot shows the Neo4j Browser interface. The browser's address bar displays the URL `10-0-1-10-37567.neo4jsandbox.com/browser/`. Below the address bar, there are several browser tabs, including "Coursera | Online C...", "Best JSON Pretty Pr...", "JSON - Scratch Wiki", "Storytelling Student...", and "DigitalOcean - scrat...".

The main interface is divided into a left sidebar and a main content area. The sidebar contains three icons: a database icon, a star icon, and a magnifying glass icon. The main content area is divided into two sections. The top section is a text editor where the Cypher query `1 CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })` and `2 Return ee` is entered. The bottom section displays the query results. It shows the query `$ CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 }) Return ee` and the results in a graph view. The graph view shows a single node labeled "Emil" with a purple label `*(1)` and an orange label `Person(1)`. The node is represented by an orange circle with the name "Emil" inside. Below the graph view, there are three tabs: "Graph", "Table", and "Text". The "Graph" tab is currently selected.

## 1.2. CREATE clauses can create many Nodes and Relationships at once.

```
▶ MATCH (ee:Person) WHERE ee.name = "Emil"  
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" } ),  
(ir:Person { name: "Ian", from: "England", title: "author" } ),  
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" } ),  
(ally:Person { name: "Allison", from: "California", hobby: "surfing" } ),  
(ee)-[:KNOWS {since: 2001}]->(js), (ee)-[:KNOWS {rating: 5}]->(ir),  
(js)-[:KNOWS]->(ir), (js)-[:KNOWS]->(rvb),  
(ir)-[:KNOWS]->(js), (ir)-[:KNOWS]->(ally),  
(rvb)-[:KNOWS]->(ally)
```

MERGE

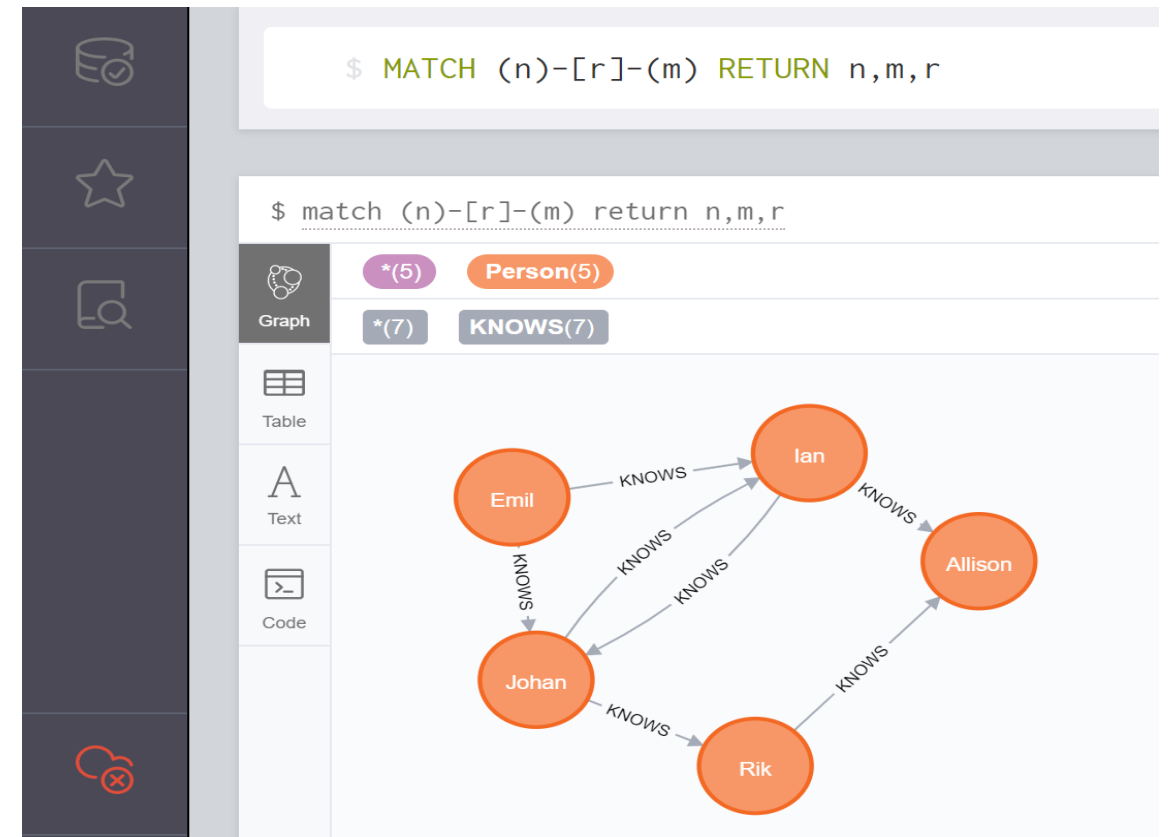




# Verify by MATCH and RETURN

`MATCH (n)-[r]-(m) RETURN n,m,r`

means Show all Nodes and Relationships



# 2. QUERY

---

## 2.1 Basic query the graph (Pattern matching )

### EX.1 Find Emil's Friends

```
▶ MATCH (ee:Person)-[:KNOWS]-(friends)  
WHERE ee.name = "Emil" RETURN ee, friends
```

- **MATCH** clause to describe the pattern from known Nodes to found Nodes
- **(ee)** starts the pattern with a Person (qualified by WHERE)
- **-[:KNOWS]-** matches "KNOWS" relationships (in either direction)
- **(friends)** will be bound to Emil's friends

## EX.2 Find Immediate Friends

MATCH (ee:Person)-[:KNOWS]-(friends)

```
WHERE ee.name = "Emil" RETURN ee, friends
```

\$ MATCH (ee:Person)-[:KNOWS]-(friends) WHERE ee.name = "Emil" RETURN ee...

The screenshot shows the Cypher Playground interface. At the top, a query is entered: `$ MATCH (ee:Person)-[:KNOWS]-(friends) WHERE ee.name = "Emil" RETURN ee...`. Below the query, there are two tabs: "Graph" and "Table". The "Table" tab is selected, showing a table with two columns: "ee" and "friends". The table contains two rows of JSON data. The first row shows a person named Emil from Sweden with a klout of 99, and a friend named Ian, an author from England. The second row shows the same person Emil, but with a friend named Johan, who is a surfer from Sweden.

ee	friends
<pre>{   "name": "Emil",   "from": "Sweden",   "klout": 99 }</pre>	<pre>{   "name": "Ian",   "title": "author",   "from": "England" }</pre>
<pre>{   "name": "Emil",   "from": "Sweden",   "klout": 99 }</pre>	<pre>{   "name": "Johan",   "learn": "surfing",   "from": "Sweden" }</pre>

\$ MATCH (ee:Person)-[:KNOWS]-(friends) WHERE

Graph

\* (3) Person (3)

\* (4) KNOWS (4)

Table

A Text

>\_ Code

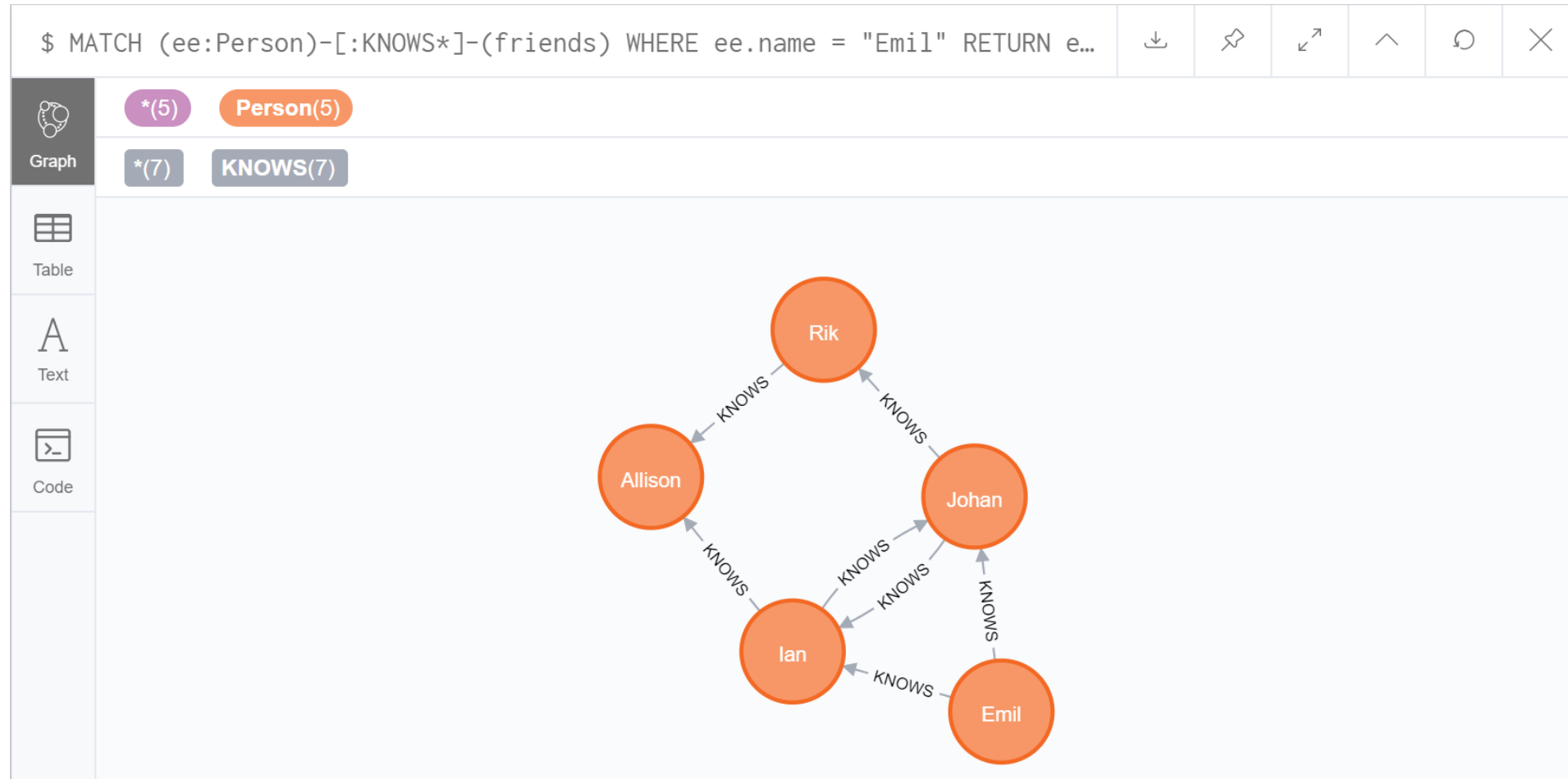
```
graph TD; Johan((Johan)) -- KNOWS --> Ian((Ian)); Ian -- KNOWS --> Johan; Johan -- KNOWS --> Emil((Emil)); Emil -- KNOWS --> Ian;
```

Displaying 3 nodes, 4 relationships.

## EX.3 Find Friends of Friends

```
MATCH (ee:Person)-[:KNOWS*]-(friends)
```

```
WHERE ee.name = "Emil" RETURN ee, friends
```



## 2.2 Make Recommendations (Pattern matching )

EX.4 Johan is learning to surf, so he may want to find a new friend who already does. Recommend him!!

```
▶ MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

- `()` empty parenthesis to ignore these nodes
- `DISTINCT` because more than one path will match the pattern
- `surfer` will contain Allison, a friend of a friend who surfs

```
MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

\$ MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer) WHERE js.name = "Joha...		↓	📌
 Graph	<b>surfer</b>		
 Table	<pre>{   "name": "Allison",   "from": "California",   "hobby": "surfing" }</pre>		
 Text			
 Code			
Started streaming 1 records in less than 1 ms and completed after 2 ms.			



# More Patterns

## Patterns

`(n:Person)`

Node with `Person` label.

`(n:Person:Swedish)`

Node with both `Person` and `Swedish` labels.

`(n:Person {name: $value})`

Node with the declared properties.

`()-[r {name: $value}]-()`

Matches relationships with the declared properties.

`(n)->(m)`

Relationship from `n` to `m`.

`(n)--(m)`

Relationship in any direction between `n` and `m`.

`(n:Person)->(m)`

Node `n` labeled `Person` with relationship to `m`.

`(m)<-[:KNOWS]-(n)`

Relationship of type `KNOWS` from `n` to `m`.

`(n)-[:KNOWS|:LOVES]->(m)`

Relationship of type `KNOWS` or of type `LOVES` from `n` to `m`.

`(n)-[r]->(m)`

Bind the relationship to variable `r`.

`(n)-[*1..5]->(m)`

Variable length path of between 1 and 5 relationships from `n` to `m`.

`(n)-[*]->(m)`

Variable length path of any number of relationships from `n` to `m`. (See Performance section.)

`(n)-[:KNOWS]->(m {property: $value})`

A relationship of type `KNOWS` from a node `n` to a node `m` with the declared property.

`shortestPath((n1:Person)-[*..6]-(n2:Person))`

Find a single shortest path.

EX.5 Johan is learning to surf, so he may want to find a new friend who already does. Recommend him [Only the shortestPath](#)

```
MATCH (js:Person)-[:KNOWS*]->(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
MATCH path = shortestPath((js)-[:KNOWS*]->(surfer) )
RETURN surfer, path
```

*Warning: Using shortest path with an unbounded pattern will likely result in long execution times. It is recommended to **use an upper limit to the number of node hops** in your pattern. So, we change to*

```
MATCH (js:Person)-[:KNOWS*]->(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
MATCH path = shortestPath((js)-[:KNOWS*..5]->(surfer) )
RETURN surfer, path
```

```
1 MATCH (js:Person)-[:KNOWS*]-(surfer)
2 WHERE js.name = "Johan" AND surfer.hobby = "surfing"
3 MATCH path = shortestPath( (js)-[:KNOWS*..5]-(surfer) )
4 RETURN surfer, path
```

\$ MATCH (js:Person)-[:KNOWS\*]-(surfer) WHERE js.name = "Johan" AND surf...



Graph

\*(3)

Person(3)

\*(3)

KNOWS(3)



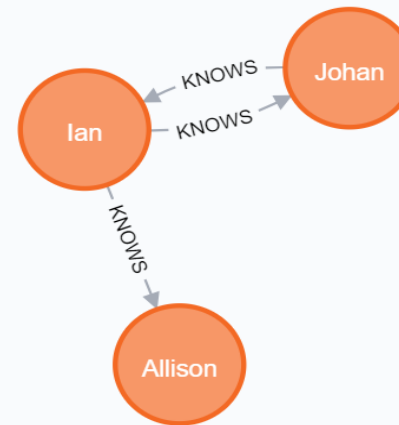
Table



Text



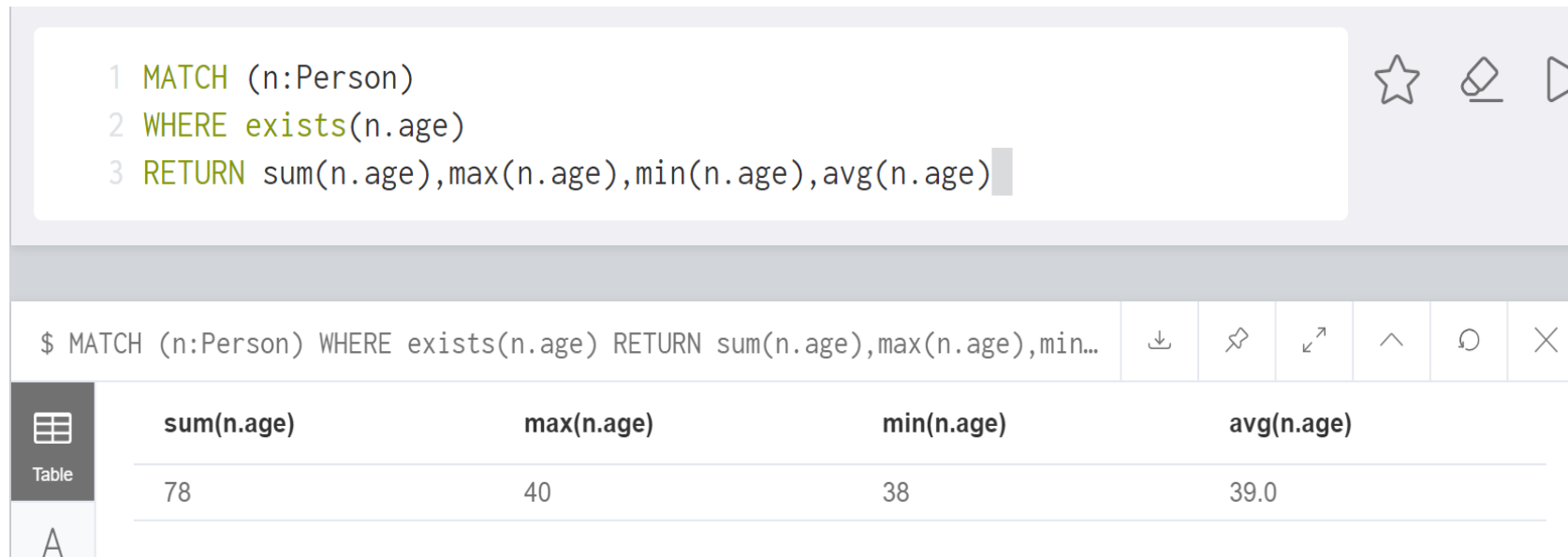
Code



## 2.3 Aggregate

EX.6 Find Total, Average, Minimum, Maximum age of all people

```
MATCH (n:Person) WHERE exists(n.age)
RETURN sum(n.age),max(n.age),min(n.age),avg(n.age)
```



The screenshot shows a query editor interface. At the top, a code editor contains a Cypher query with line numbers 1, 2, and 3. To the right of the code editor are three icons: a star, a pencil, and a play button. Below the code editor is a toolbar with a text input field containing the query, followed by icons for download, favorite, expand, zoom in, zoom out, and close. Below the toolbar is a table with four columns: sum(n.age), max(n.age), min(n.age), and avg(n.age). The table has one data row with values 78, 40, 38, and 39.0. On the left side of the table is a sidebar with a 'Table' button and a letter 'A'.

```
1 MATCH (n:Person)
2 WHERE exists(n.age)
3 RETURN sum(n.age),max(n.age),min(n.age),avg(n.age)
```

\$ MATCH (n:Person) WHERE exists(n.age) RETURN sum(n.age),max(n.age),min...

sum(n.age)	max(n.age)	min(n.age)	avg(n.age)
78	40	38	39.0

# Note.

```
1 MATCH (n:Person)
2 WHERE exists(n.age)
3 RETURN n
```

```
$ MATCH (n:Person) WHERE exists(n.age) RETURN n
```



Graph



Table



Text



Code

```
{
  "name": "Johan",
  "from": "Sweden",
  "learn": "surfing",
  "surname": "Taylor",
  "age": 40
}
```

```
{
  "name": "Ian",
  "from": "England",
  "title": "author",
  "pet": "Bingo",
  "age": 38
}
```

## EX.7 Find Total number of persons

MATCH (n:Person) RETURN count(n)

```
1 MATCH (n:Person)
2 RETURN count(*)
```

\$ MATCH (n:Person) RETURN count(\*)

	count(*)
	5

Table

Text



## EX.8 Find Total number of persons who have pet

MATCH (n:Person) WHERE exists (n.pet) RETURN count(n)

```
1 MATCH (n:Person)
2 WHERE exists(n.pet)
3 RETURN count(*)
```

\$ MATCH (n:Person) WHERE exists(n.pet) RETURN count(\*)



Table

**count(\*)**

2



Text

# 3. UPDATE

---

## 3.1 Update Node or Relationship Property

EX.9 Set Johan's surname to be 'Taylor' and age =40

```
MATCH (js {name: 'Johan'})  
SET js.surname = 'Taylor', js.age = 40  
RETURN js
```

The screenshot shows a Cypher query editor with the following query:

```
1 MATCH (js {name: 'Johan'})  
2 SET js.surname = 'Taylor'  
3 RETURN js.name, js.surname
```

Below the query editor, the executed query is shown: `$ MATCH (js {name: 'Johan'}) SET js.surname = 'Taylor' RETURN js.name, ...`. To the right of the query are icons for favorite, copy, and run.

The results are displayed in a table with two columns: **js.name** and **js.surname**. The table contains one row with the values "Johan" and "Taylor".

js.name	js.surname
"Johan"	"Taylor"

(1) If you set a property with NULL value = removing the property

EX.10 Set Johan's surname to be NULL means removing the surname

```
MATCH (js {name: 'Johan'})
SET js.surname = NULL
RETURN js
```

## BEFORE

```
1 MATCH (js {name: 'Johan'})
2 SET js.surname = 'Taylor'
3 RETURN js.name, js.surname
```

```
$ MATCH (js {name: 'Johan'}) SET js.surname = 'Taylor' RETURN js
```

	js.name	js.surname
	"Johan"	"Taylor"

## AFTER

```
$ MATCH (js {name: 'Johan'}) SET js.surname = NULL F
```



Graph



Table



Text



js

```
{
  "name": "Johan",
  "learn": "surfing",
  "from": "Sweden"
}
```

## (2) Set mutate properties using +=

- Any properties in the map that are not on the node or relationship will be added.
- Any properties not in the map that are on the node or relationship will be left as is.
- Any properties that are in both the map and the node or relationship will be replaced in the node or relationship.
- However, if any property in the map is null, it will be removed from the node or relationship.

## EX.11 Update Ian age and pet using +=

```
MATCH (ir { name: 'Ian' })
SET ir += { age: 38, pet: 'Bingo' }
RETURN ir.name, ir.age, ir.pet
```

### BEFORE

```
➤ MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" }),
(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby: "surfing" }),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]->(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),
(rvb)-[:KNOWS]->(ally)
```

### AFTER

```
1 MATCH (ir { name: 'Ian' })
2 SET ir += { age: 38, pet: 'Bingo' }
3 RETURN ir.name, ir.age, ir.pet
```

```
$ MATCH (ir { name: 'Ian' }) SET ir += { age: 38, pet: 'Bingo' } RETURN...
```



Table

A

ir.name

ir.age

ir.pet

"Ian"

38

"Bingo"

## EX.12 Update relationship KNOWS to specify that Johan has known Rik since 2018

```
MATCH (:Person {name: 'Johan'})-[rel:KNOWS]-(:Person {name: 'Rik'})
SET rel.startYear = date({year: 2018})
RETURN rel
```

### BEFORE

```
➤ MATCH (ee:Person) WHERE ee.name = "Emil"
CREATE (js:Person { name: "Johan", from: "Sweden", learn: "surfing" }),
(ir:Person { name: "Ian", from: "England", title: "author" }),
(rvb:Person { name: "Rik", from: "Belgium", pet: "Orval" }),
(ally:Person { name: "Allison", from: "California", hobby: "surfing" }),
(ee)-[:KNOWS {since: 2001}]->(js),(ee)-[:KNOWS {rating: 5}]->(ir),
(js)-[:KNOWS]->(ir),(js)-[:KNOWS]->(rvb),
(ir)-[:KNOWS]->(js),(ir)-[:KNOWS]->(ally),
(rvb)-[:KNOWS]->(ally)
```

### AFTER

```
1 MATCH (:Person {name: 'Johan'})-[rel:KNOWS]-(:Person {name: 'Rik'})
2 SET rel.startYear = date({year: 2018})
3 RETURN rel
```

```
$ MATCH (:Person {name: 'Johan'})-[rel:KNOWS]-(:Person {name: 'Rik'}) SET rel.startYe
```



Table



Text



rel

```
{
  "startYear": "2018-01-01"
}
```



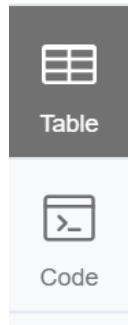
## 3.2 Update Node Label

Use SET to set **Label(s)** to a node

EX.13 Update Label for Johan to be Parent and Employee

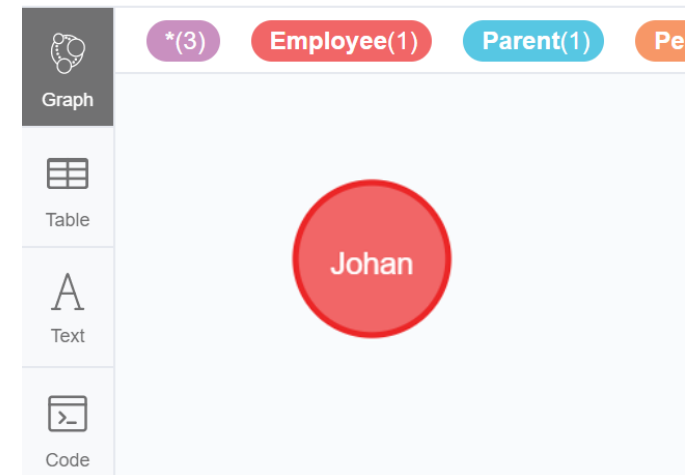
```
MATCH (js {name: 'Johan'})  
SET js:Parent:Employee
```

```
$ MATCH (js {name: 'Johan'}) SET js:Parent:Employee
```



Added 2 labels, completed after 14 ms.

```
$ match (js {name:'Johan'}) return js
```



# 4. DELETE

---

`DELETE n, r`

Delete a node and a relationship.

`DETACH DELETE n`

Delete a node and all relationships connected to it.

`MATCH (n)`

`DETACH DELETE n`

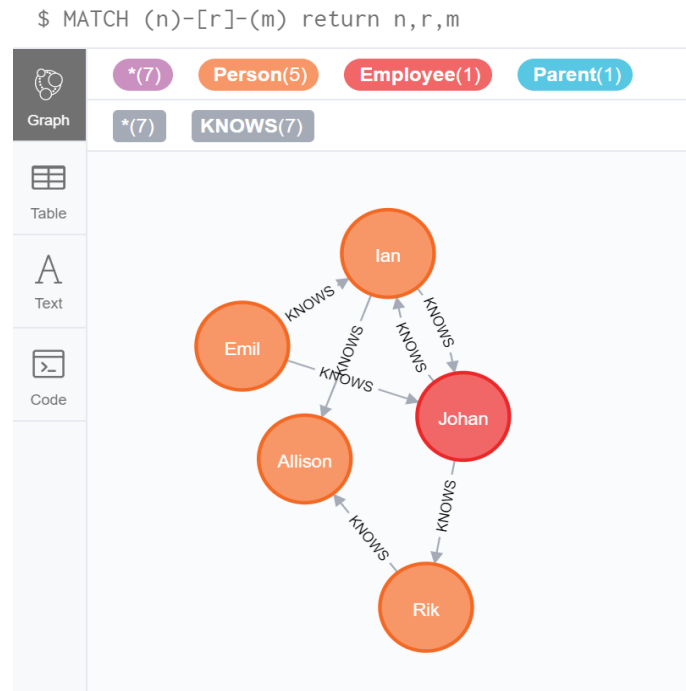
Delete all nodes and relationships from the database.

## 4.1 Delete a specific node

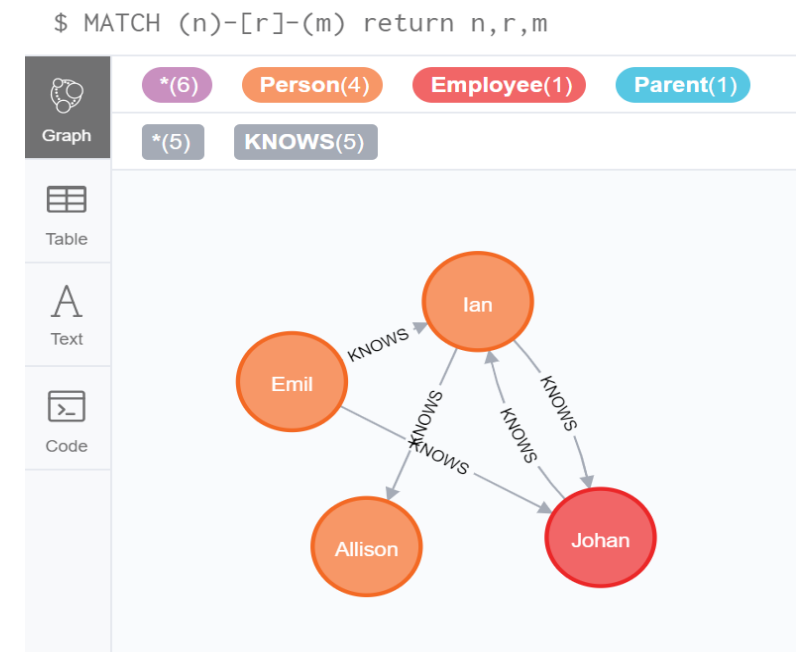
### EX.14 Delete Rik node

```
MATCH (x {name: 'Rik'})  
DETACH DELETE x
```

BEFORE



AFTER

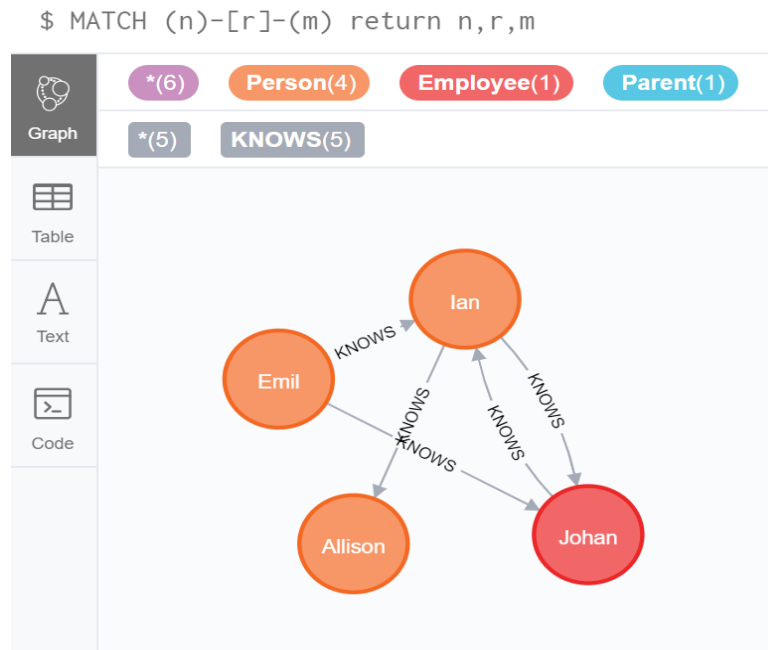


## 4.2 Delete a specific relationship

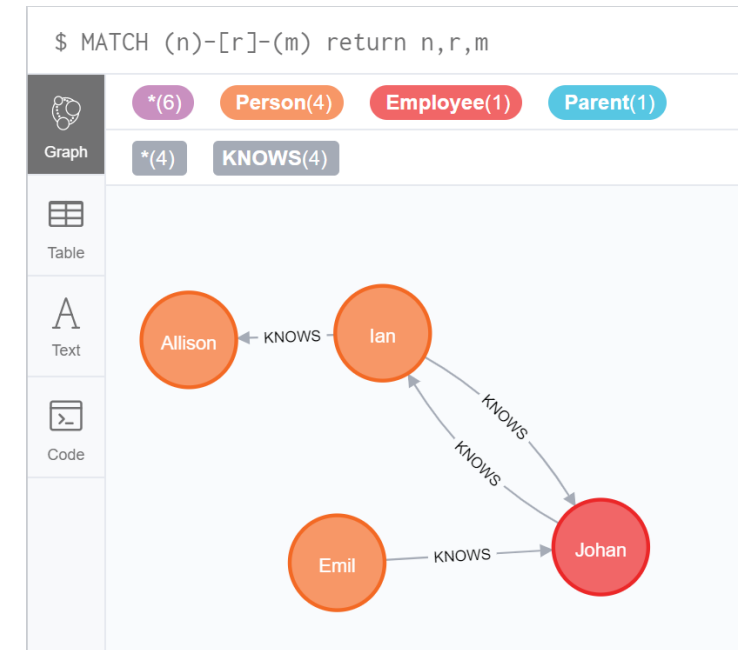
### EX.15 Delete Rik node

```
MATCH (y {name: 'Emil'}) -[r:KNOWS]->(ir{name: 'Ian'})
DELETE r
```

BEFORE



AFTER



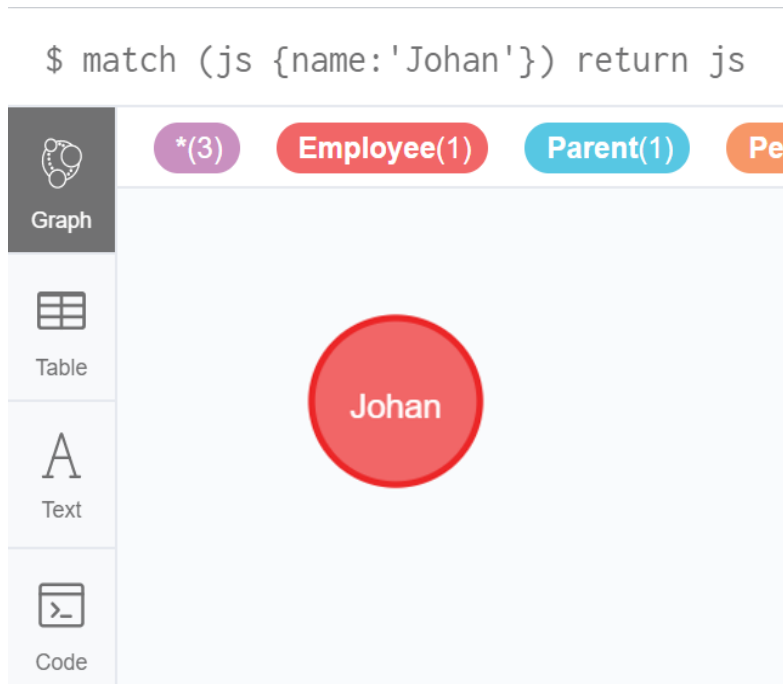
## 4.3 Remove Label from a node

### EX.16 Remove Label Employee from Johan

```
MATCH (js {name: 'Johan'})  
REMOVE js:Employee  
RETURN js
```

BEFORE

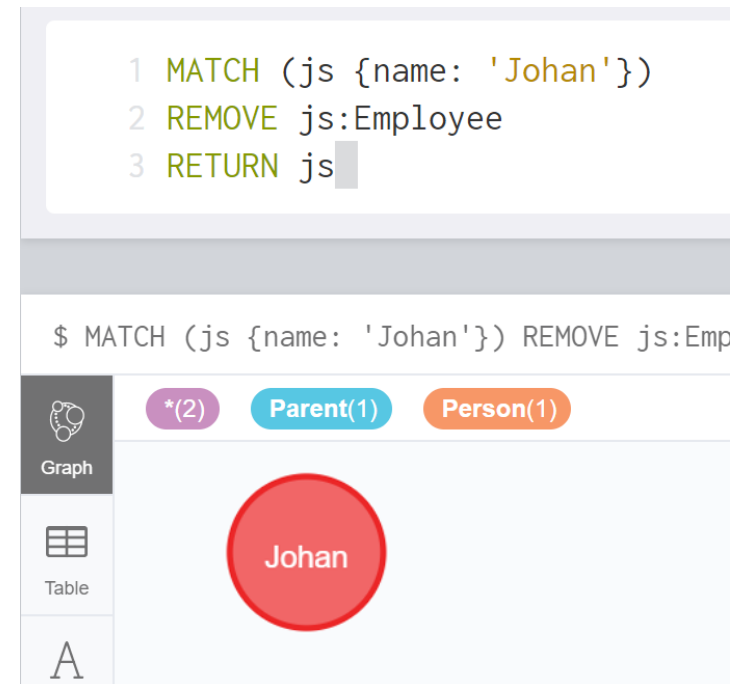
\$ match (js {name:'Johan'}) return js



AFTER

```
1 MATCH (js {name: 'Johan'})  
2 REMOVE js:Employee  
3 RETURN js
```

\$ MATCH (js {name: 'Johan'}) REMOVE js:Emp



## 4.4 Remove a property

### EX.17 Remove age property of Johan

```
MATCH (js {name: 'Johan'})  
REMOVE js.age  
RETURN js
```



The screenshot displays a Neo4j Cypher query interface. At the top, the query is entered: `$ MATCH (js {name: 'Johan'}) REMOVE js.age RETURN js`. Below the query, a sidebar on the left contains four icons: a graph icon labeled 'Graph', a table icon labeled 'Table' (which is highlighted), a text icon labeled 'Text', and a code icon labeled 'Code'. The main area of the interface, titled 'js', shows the resulting JSON object: `{ "name": "Johan", "learn": "surfing", "from": "Sweden" }`.

## 4.5 Delete ALL nodes

### EX.18 Delete ALL nodes

```
MATCH (n)
DETACH DELETE n
```

```
1 MATCH (n)
2 DETACH DELETE n
```

```
$ MATCH (n) DETACH DELETE n
```



Table

Deleted 4 nodes, deleted 3 relationships, completed after 2 ms.

# Assignment Deadline

---

NEXT THURS 11.59 PM

SUBMIT on **GOOGLE CLASSROOM**

## Submission Procedure

1. Submit in Google Form then you will get a confirmation email.
2. Turn in the confirmation email in Google Classroom.



# References

---

- <https://neo4j.com/docs/cypher-refcard/current/>
- <https://neo4j.com/developer/>
- <https://neo4j.com/docs/cypher-manual/current/clauses>
- <https://neo4j.com/docs/cypher-manual/current/introduction/#cypher-introduction>
- <https://neo4j.com/developer/guide-sql-to-cypher/>



Thank You.

---