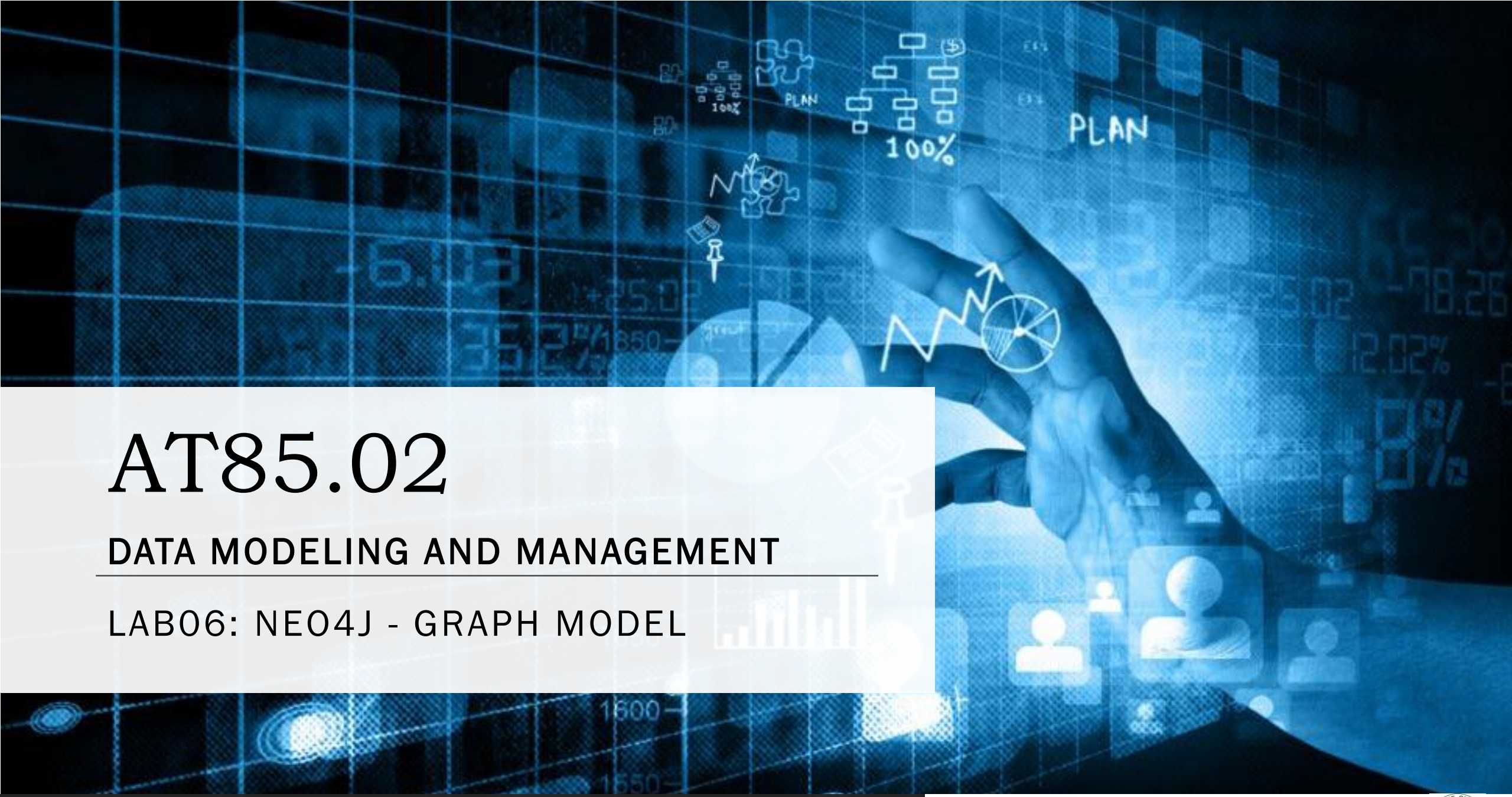


AT85.02

DATA MODELING AND MANAGEMENT

LAB06: NEO4J - GRAPH MODEL

Co-funded by the Erasmus+ Programme of the European Union

DS&AI

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

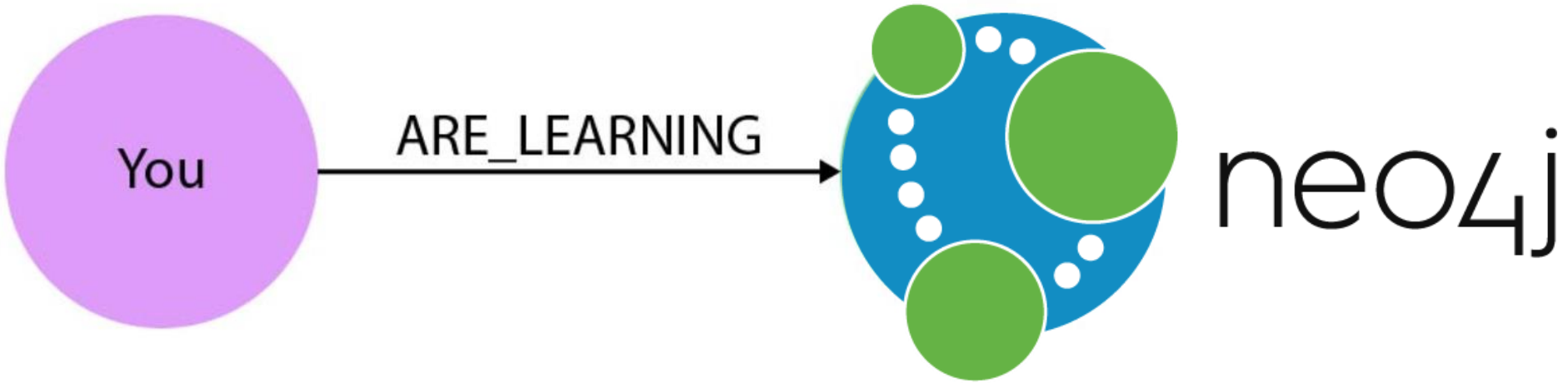
4. DELETE

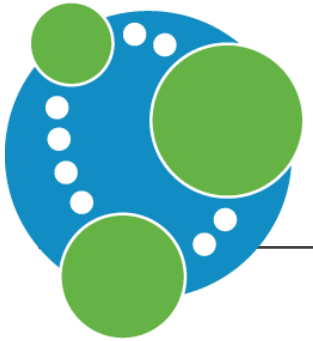
4.1 Delete a specific node

4.2 Delete a specific relationship

4.3 Remove Label from a node

4.4 Remove a property


You
ARE_LEARNING
neo4j



neo4j

- 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

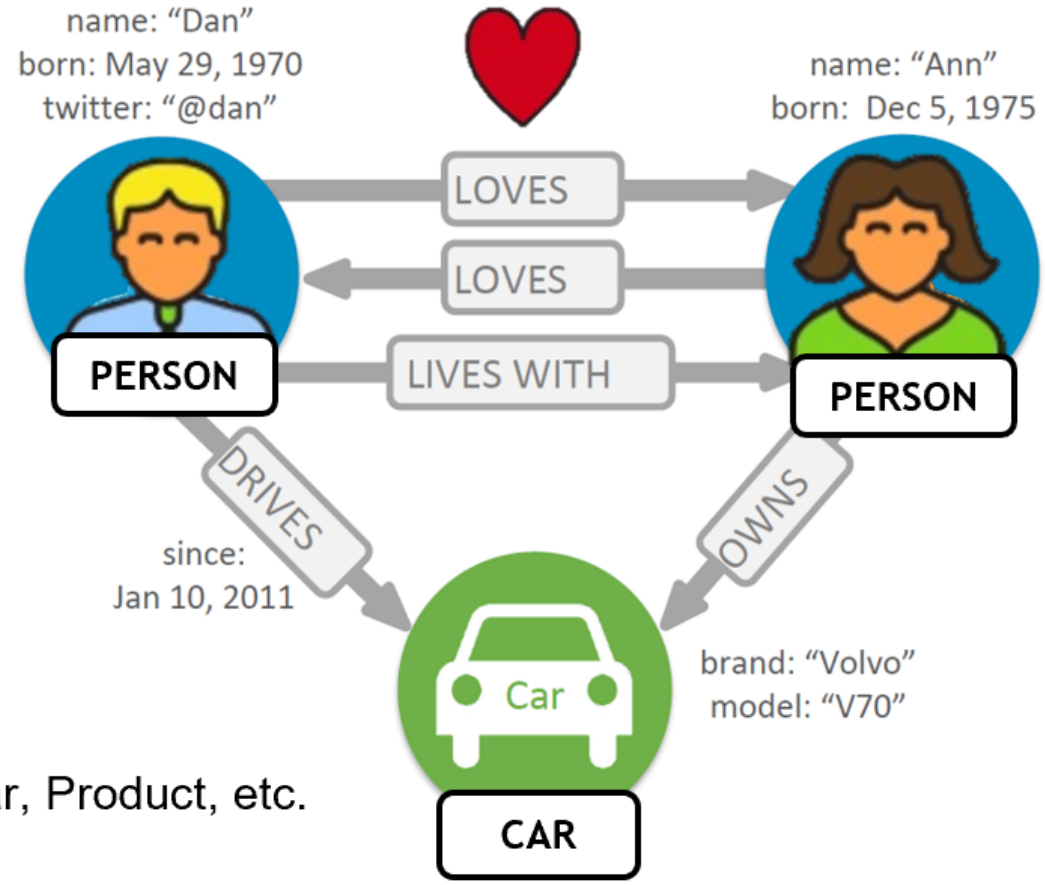
- Relate nodes by *type* and *direction*

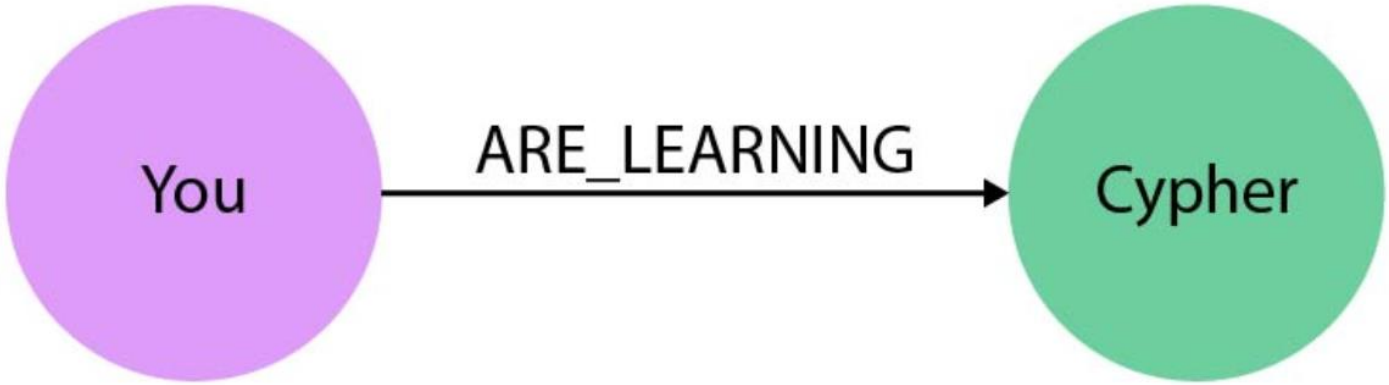
Properties

- 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

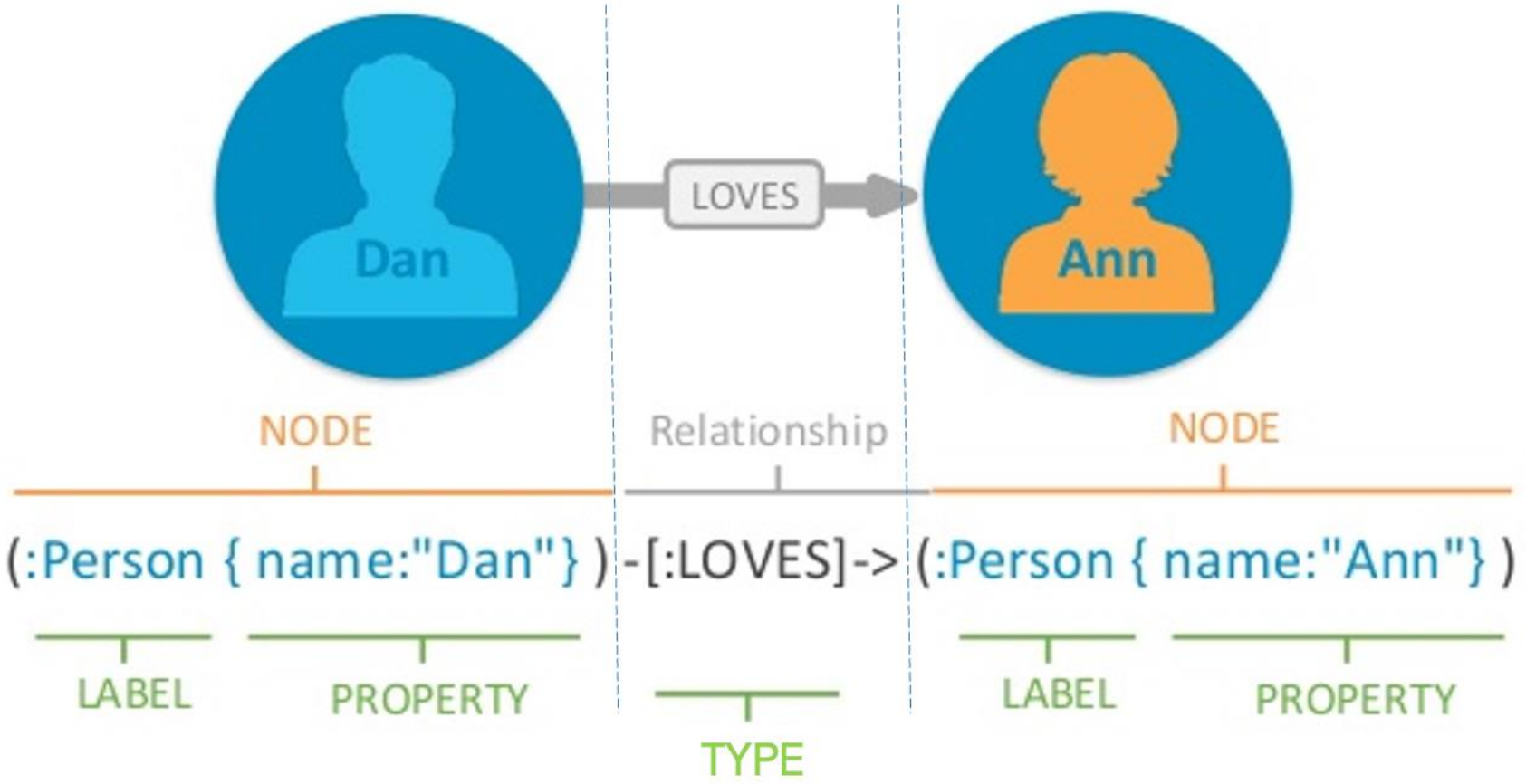



You
ARE_LEARNING
Cypher

Cypher Query Language (CQL)

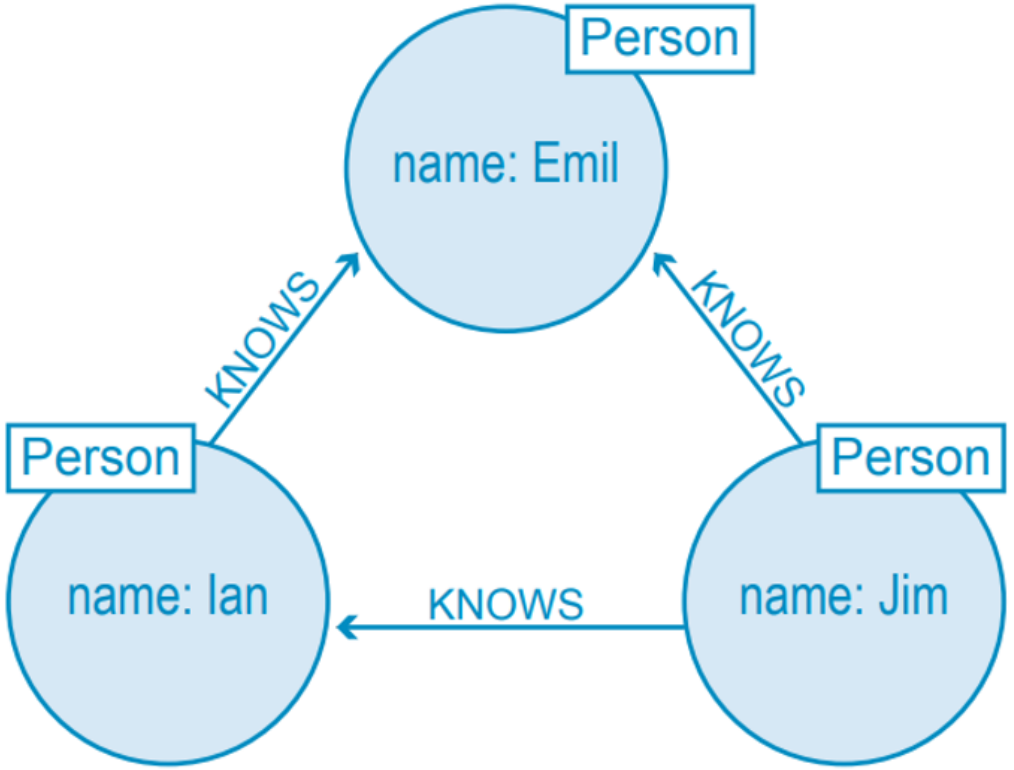
- uses patterns to describe graph data
- familiar SQL-like clauses
- declarative, describing what to find, not how to find it

Cypher Query Language Syntax



Example

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



Getting Started

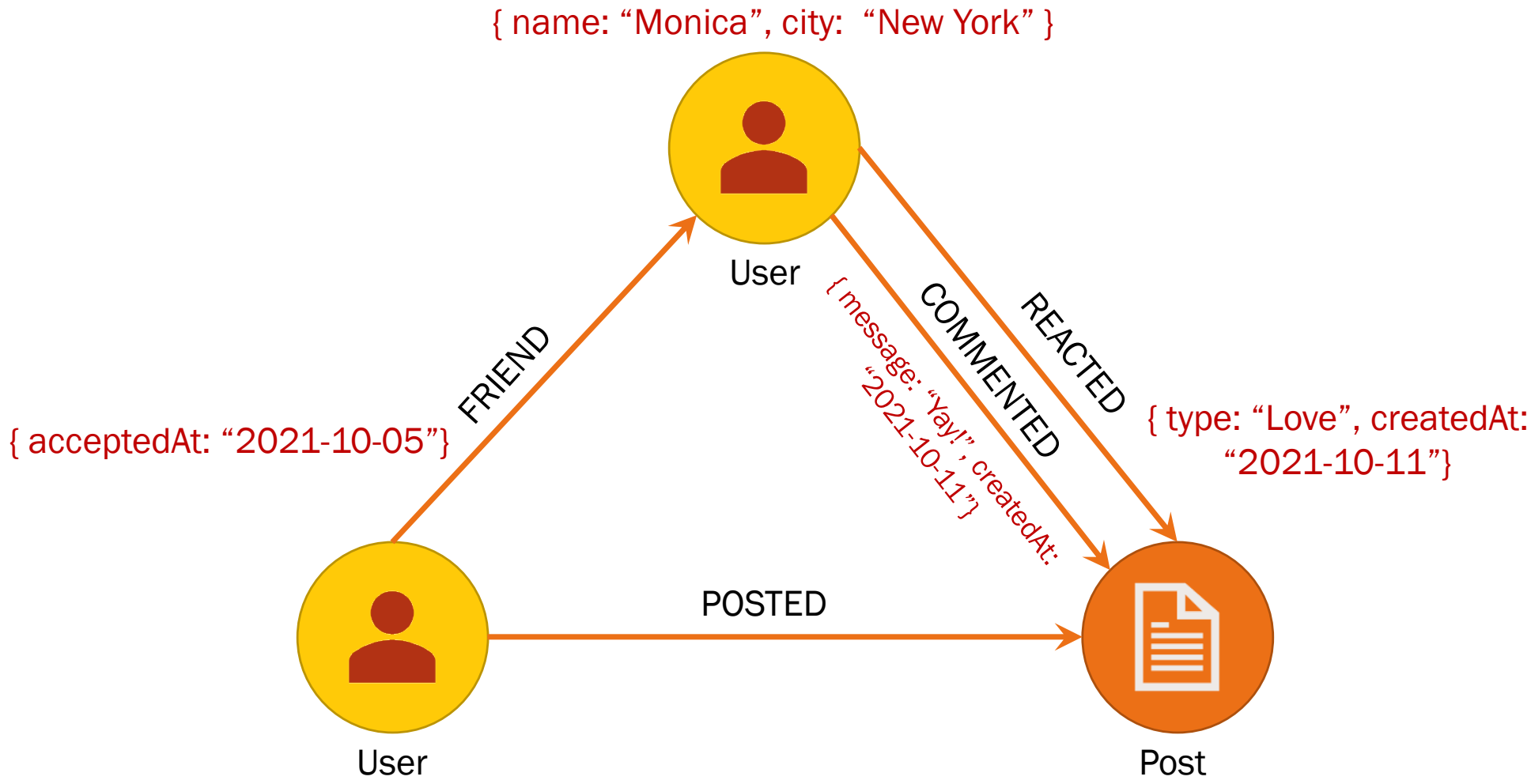
NEO4J SANDBOX (DEMONSTRATION)

HTTPS://SANDBOX.NEO4J.COM/

CRUD operations

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

Case Study: Social Network

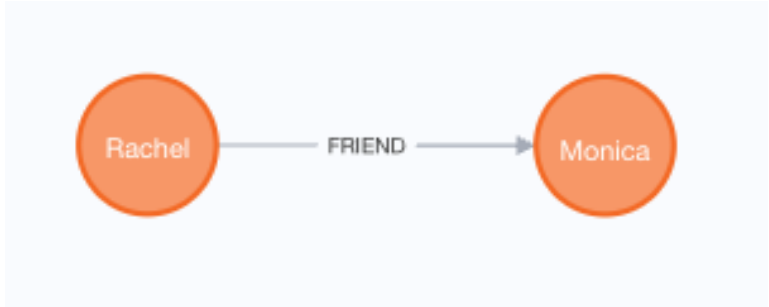


1. CREATE

Create Nodes with a relationship

```
CREATE (r:User { name: "Rachel", city: "London" }),
       (m:User { name: "Monica", city: "New York" }),
       (r)-[:FRIEND {acceptedAt:"2021-10-05"}]-> (m)
```

```
CREATE (r:User { name: "Rachel", city: "London" }),
       (m:User { name: "Monica", city: "New York" }),
       (r)-[f:FRIEND {acceptedAt:"2021-10-05"}]-> (m)
RETURN r,f,m
```



Graph View

r

```
{
  "identity": 0,
  "labels": [
    "User"
  ],
  "properties": {
"name": "Rachel",
"city": "London"
  }
}
```

f

```
{
  "identity": 0,
  "start": 0,
  "end": 1,
  "type": "FRIEND",
  "properties": {
"acceptedAt": "2021-10-05"
  }
}
```

m

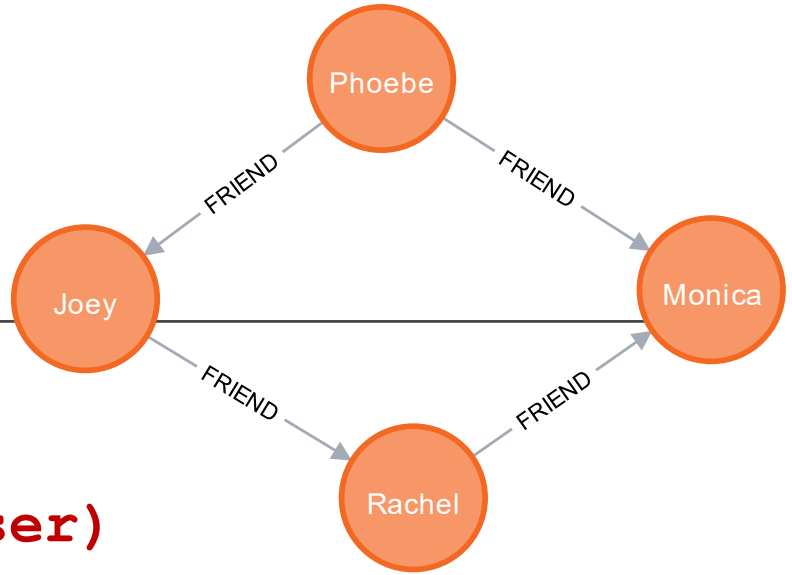
```
{
  "identity": 1,
  "labels": [
    "User"
  ],
  "properties": {
"name": "Monica",
"city": "New York"
  }
}
```

Table View

Your Turn (1)

- Create Cypher commands that insert two more users and make them as friends. Try add other properties than city.
- Make each users you added as friends with the 2 previous users (Rachel and Monica).

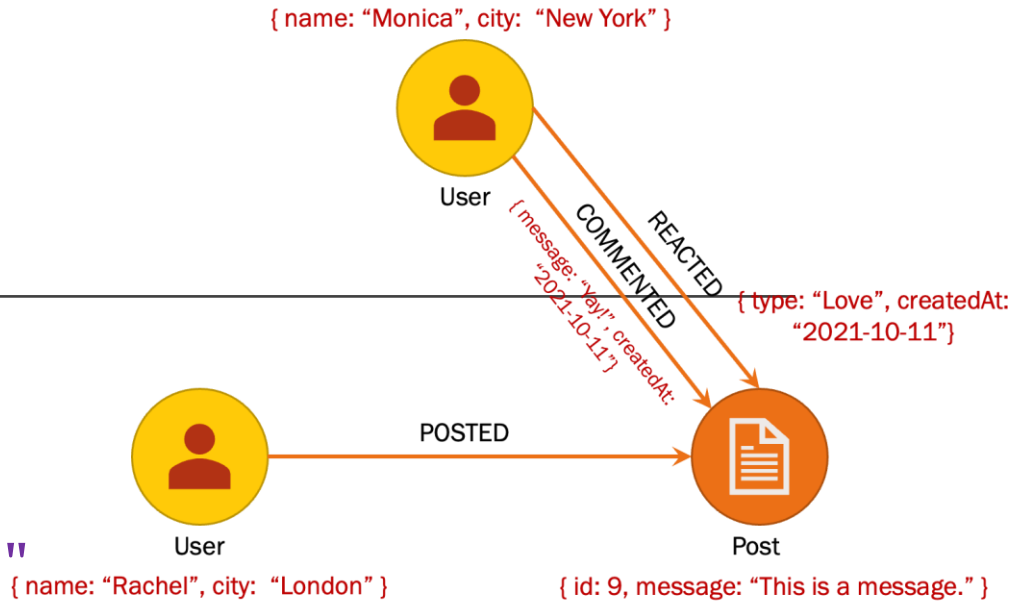
1. CREATE



Create more friendships

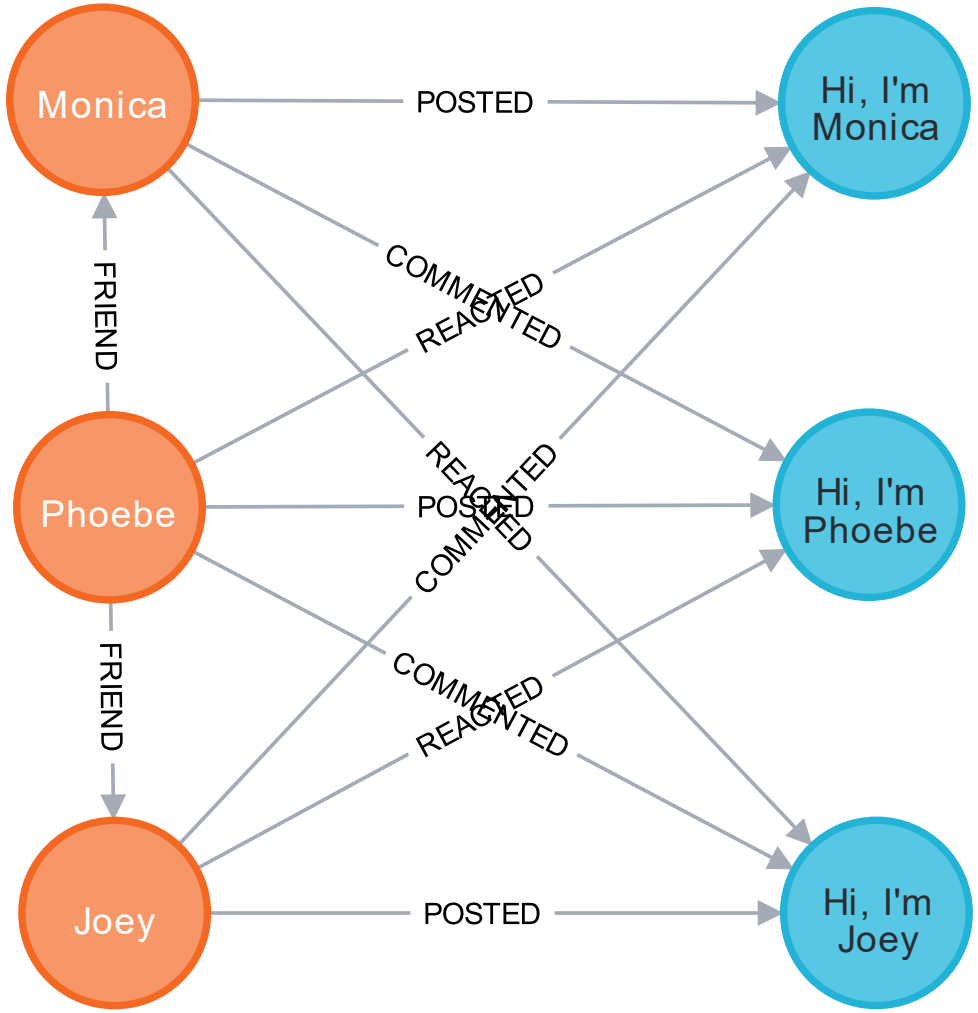
```
MATCH (r:User), (m:User),(ph:User),(j:User)
WHERE r.name="Rachel" AND m.name="Monica" AND ph.name =
"Phoebe" AND j.name="Joey"
CREATE (ph)-[:FRIEND {acceptedAt: date() }]-> (m),
       (j)-[:FRIEND {acceptedAt: date() }]-> (r)
RETURN r,m,ph,j;
```

1. CREATE



Create a post , a comment and a reaction.

```
MATCH (r:User), (m:User)
WHERE r.name="Rachel" AND m.name="Monica"
CREATE (p:Post { id:"9", message: "This is a message." } ),
(r)-[:POSTED]->(p),
(m)-[:REACTED { type: "Love", createdAt: date() }]->(p),
(m)-[:COMMENTED { message: "Yay!", createdAt: date() }]->(p)
RETURN r,m,p;
```



Your Turn (2)

- Create 3 more posts.
- Each must be posted by existing users and get at least a reaction and a comment from other users.

2. QUERY

2.1 Basic query the graph (Pattern matching)

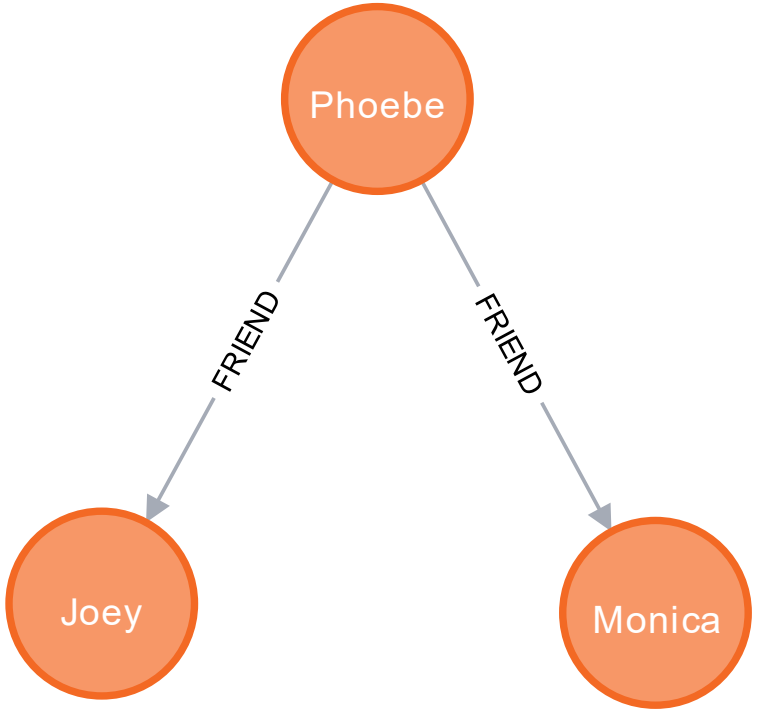
EX.1 Find Emil's Friends

```
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

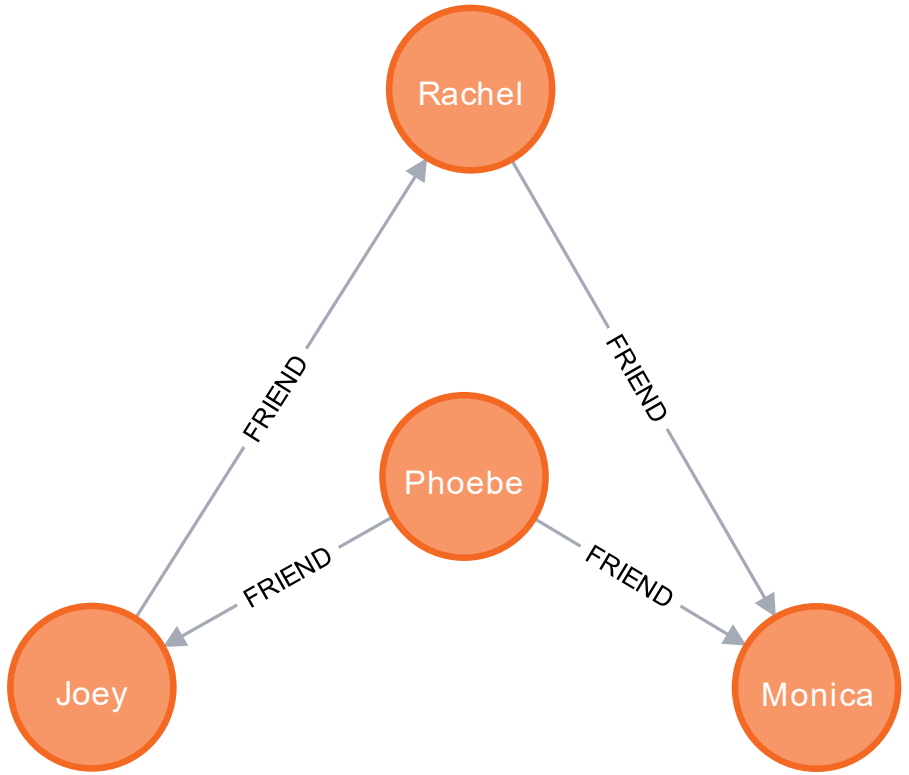
EX.2 Find Immediate Friends

```
MATCH (u:User)-[:FRIEND]->(f)
WHERE u.name = "Phoebe"
RETURN u,f;
```



EX.3 Find Friends of Friends

```
MATCH (u:User)-[:FRIEND*]->(f)
WHERE u.name = "Phoebe"
RETURN u,f;
```



2.2 Make Recommendations (Pattern matching)

EX.4 Rachel is planning to go to Newark for a business trip, so she is looking for her friends who know someone living that city.

```
MATCH (r:User)-[:FRIEND]->()-[:FRIEND]-(newarkian)
WHERE r.name="Rachel" AND newarkian.city="Newark"
RETURN DISTINCT newarkian;
```

- `()` empty parenthesis to ignore these nodes
- `DISTINCT` because more than one path will match the pattern

```
MATCH (r:User)-[:FRIEND]->()-[:FRIEND]-(newarkian)
WHERE r.name="Rachel" AND newarkian.city="Newark"
RETURN DISTINCT newarkian;
```


```
{
  "identity": 2,
  "labels": [
    "User"
  ],
  "properties": {
"name": "Phoebe",
"gender": "female",
"city": "Newark",
"lastname": "Buffay"
  }
}
```

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.

Your Turn (3)

1. Find who reacted all Monica's post.
2. Modify the above command to find all friends of the reactor of Monica's post.

2.3 Aggregation

EX.*5-1* Find total, average, minimum and maximum age of all users

```
MATCH (u:User) WHERE exists(u.age)
RETURN sum(u.age), max(u.age), min(u.age), avg(u.age);
```

```
neo4j$ MATCH (u:User) WHERE exists(u.age) RETURN sum(u.age), max(u.age), min(u.age), avg(u.age);
```

Table

Text

	sum(u.age)	max(u.age)	min(u.age)	avg(u.age)
1	365	87	24	60.83333333333333

2.3 Aggregate

EX.5-2 Find Total, Average, Minimum, Maximum age of all people grouped by users' gender.

```
MATCH (u:User) WHERE exists(u.age)
RETURN u.gender, sum(u.age), max(u.age), min(u.age), avg(u.age);
```

	u.gender	sum(u.age)	max(u.age)	min(u.age)	avg(u.age)
1	*null*	186	82	47	62.0
2	"female"	155	87	68	77.5
3	"male"	24	24	24	24.0

EX.6 Find total number of users

```
// Without a grouping key
MATCH (u:User) RETURN count(u);
```

	count(u)
1	6

```
// With a grouping key
MATCH (u:User)
RETURN u.city, count(u);
```

u.city	count(u)
"London"	1
"New York"	1
"Newark"	2
"Arkansas"	1
"Texas"	1

EX.7 Find Total number of persons who have pet

```
MATCH (u:User) WHERE u.age >=60
RETURN u.city AS City,
u.gender AS gender,
count(u) AS NoOfUsers;
```

City	gender	NoOfUsers
"New York"	*null*	1
"Newark"	"female"	2

EX.8 Enumerate all cities of users

```
MATCH (u:User)
RETURN collect(u.city), collect(DISTINCT u.city);
```

Your Turn (4)

1. Count all posts grouped by each user.
2. List all friends' name of each user. Show friends names in the same column.

3. UPDATE

3.1 Update Node or Relationship Property

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

```
MATCH (r:User) WHERE r.name="Rachel"
SET r.lastname = "Berry", r.age=34
RETURN r.name, r.lastname, r.age;
```

"r.name"	"r.lastname"	"r.age"
"Rachel"	"Berry"	34

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

EX.10 Remove Rachel's age.

```
MATCH (r:User {name:"Rachel"})
SET r.age = NULL
RETURN r;
```

"r"
{"name":"Rachel","city":"London","lastname":"Berry"}

(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 Monica's age and workplace using +=

```
MATCH (m:User { name: "Monica" })
SET m += { age: 39, workplace: "World Bank" }
RETURN m.name, m.age, m.workplace;
```

"m.name"	"m.age"	"m.workplace"
"Monica"	39	"World Bank"

EX.12 Update Phoebe's reaction on the Monica's post by changing the reaction type to 'Love' and recording a current date-time.

```
MATCH (:User { name : "Phoebe's"})-[re:REACTED]->(p:Post)<-
      [:POSTED]-(:User { name : "Monica" })
SET re.type="Love", re.createdAt=datetime()
RETURN re.type, re.createdAt;
```

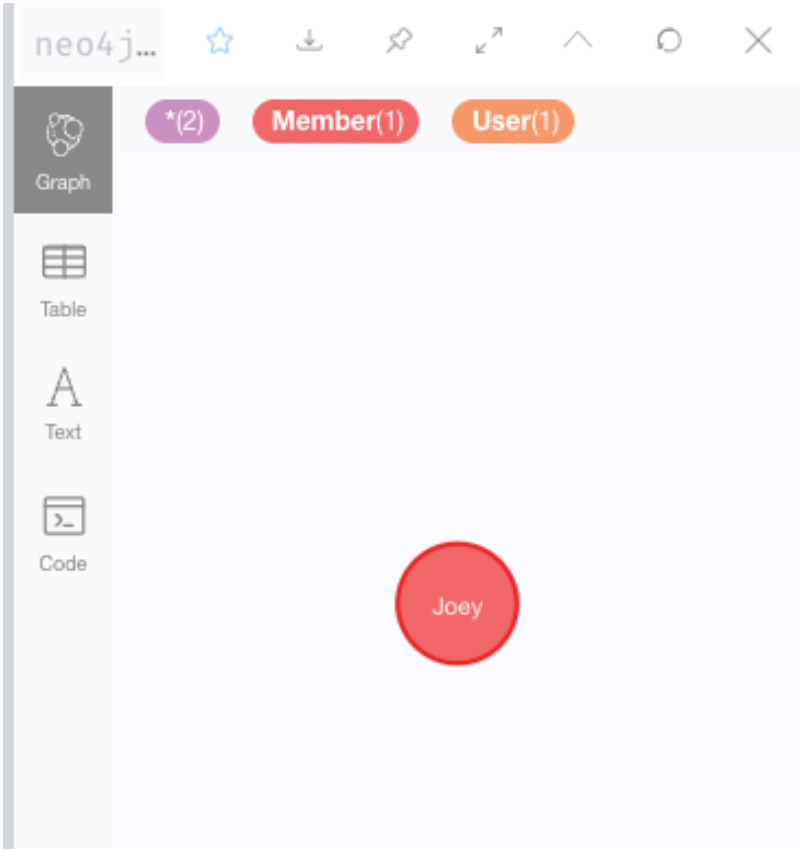
re.type	re.createdAt
"Love"	"2021-10-21T10:51:01.449000000Z"

3.2 Update a node label

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

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

```
MATCH (j {name:"Joey"})
SET j:User:Member
RETURN j;
```



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 Kim’s node

```
MATCH (k {name: 'Kim'})
DETACH DELETE k;
```

4.2 Delete a specific relationship

EX.15 Undo a reaction of Monica on a Joey's post.

```
MATCH (m{name: 'Monica'})-[re:REACTED]->(p)<-[:POSTED]-
      (j{name:'Joey'})
DELETE re;
```

4.3 Remove Label from a node

EX.16 Remove label Member from Joey

```
MATCH (j {name: 'Joey'})
REMOVE j:Member
RETURN j;
```


```
{
  "identity": 3,
  "labels": [
    "User"
  ],
  "properties": {
"name": "Joey",
"city": "Arkansas",
"dob": "1988-01-01",
"age": 40
  }
}
```

4.4 Remove a property

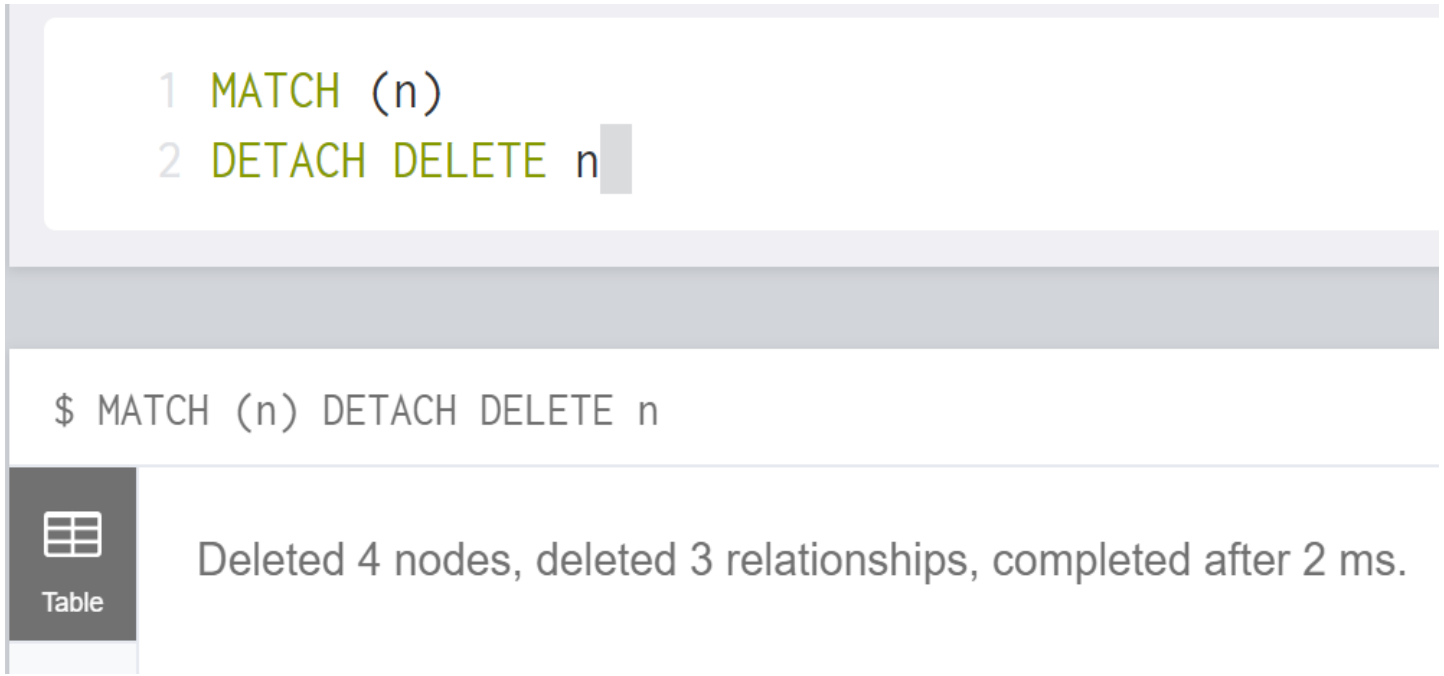
EX.17 Remove DOB property Joey's node

```
MATCH (j {name: 'Joey'})
REMOVE j.dob
RETURN j;
```

4.5 Delete ALL nodes

EX.18 Delete ALL nodes

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



References

- Migrating SQL to Cypher https://neo4j.com/developer/guide-sql-to-cypher/
- Patterns Matching https://neo4j.com/docs/cypher-manual/current/syntax/patterns/
- Aggregating functions https://neo4j.com/docs/cypher-manual/current/functions/aggregating/
- SET property using MAP https://neo4j.com/docs/cypher-manual/current/clauses/set/#set-setting-properties-using-map
- Importing Data from CSV Files https://neo4j.com/developer/guide-import-csv/
- Setting Up Neo4j on AWS EC2 https://www.inoutcode.com/aws/how-to-setup-neo4j-on-aws-ec2/


Thank You.