Large-Scale and Multi-Structured Databases

Neo4J: Introduction to Cypher

Prof Pietro Ducange

Ing. Alessio Schiavo

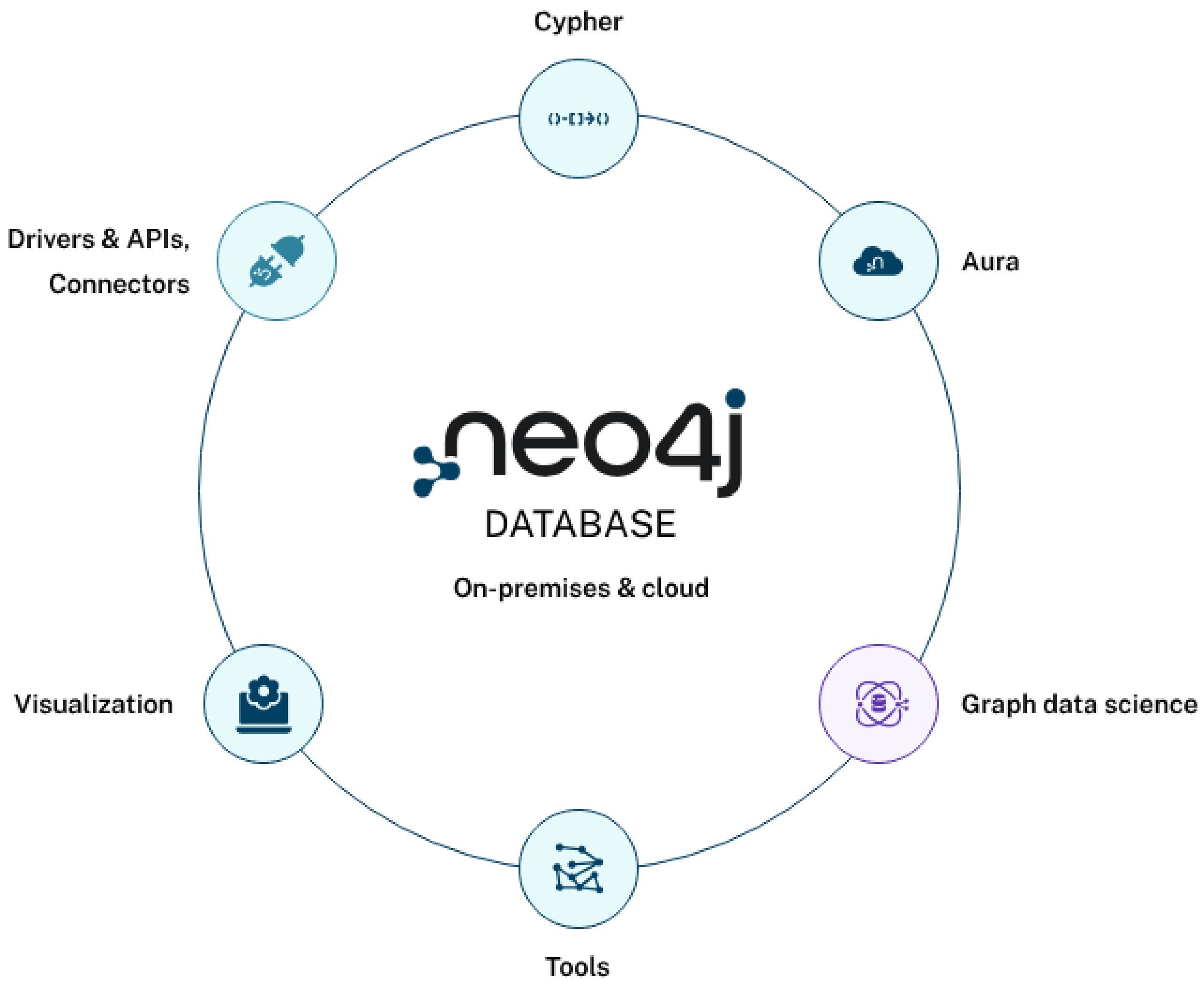
alessio.schiavo@phd.unipi.it

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

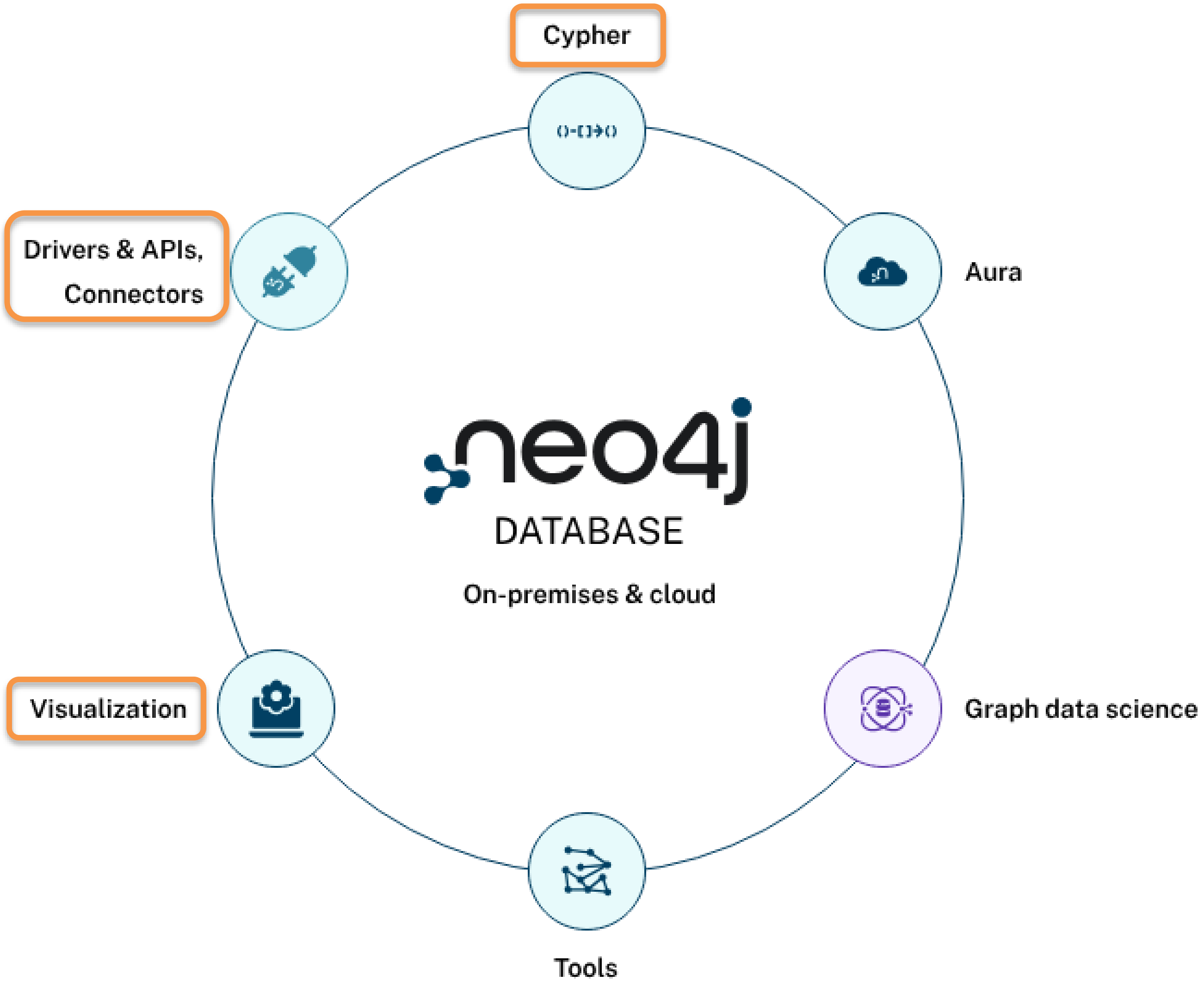
UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

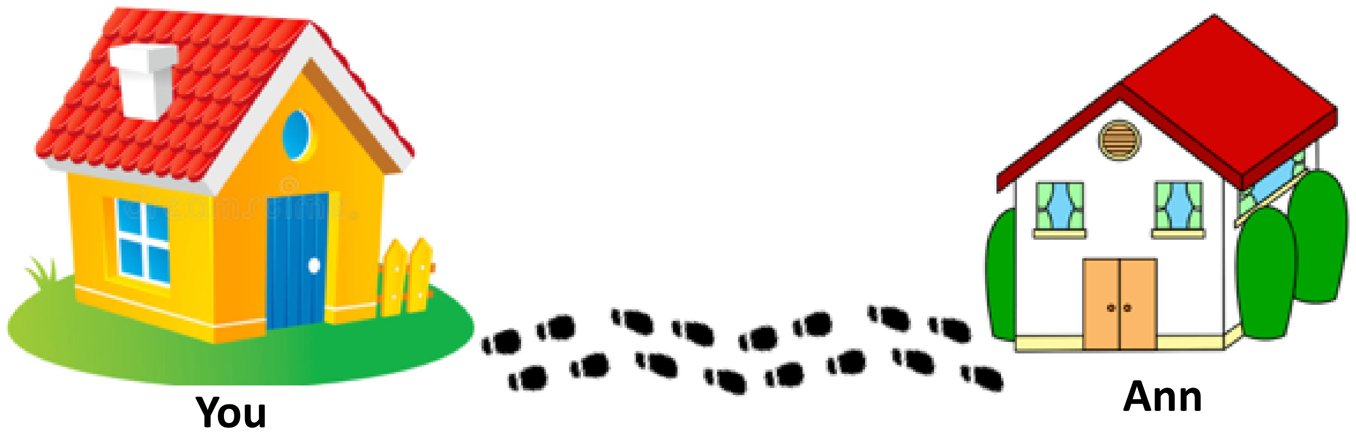
neo4J components



neo4J components



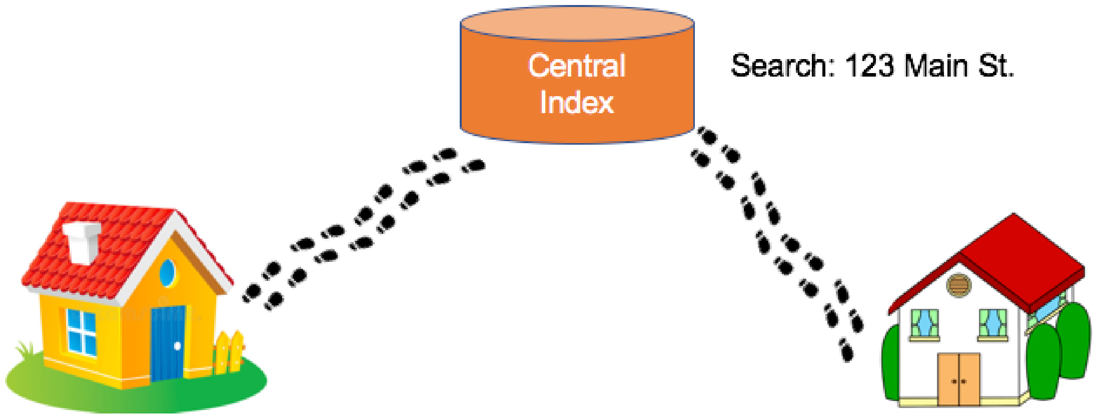
Index Free Adjacency (IFA)



1. You walk out the door.
2. You **point** towards Ann’s house.
3. You walk directly over to Ann’s house, which takes about 30 seconds, and you give her the hot apple pie — and she loves it! Happy ending!

Source: https://medium.com/@dmccreary/how-to-explain-index-free-adjacency-to-your-manager-1a8e68ec664a

Problems in Relational DB

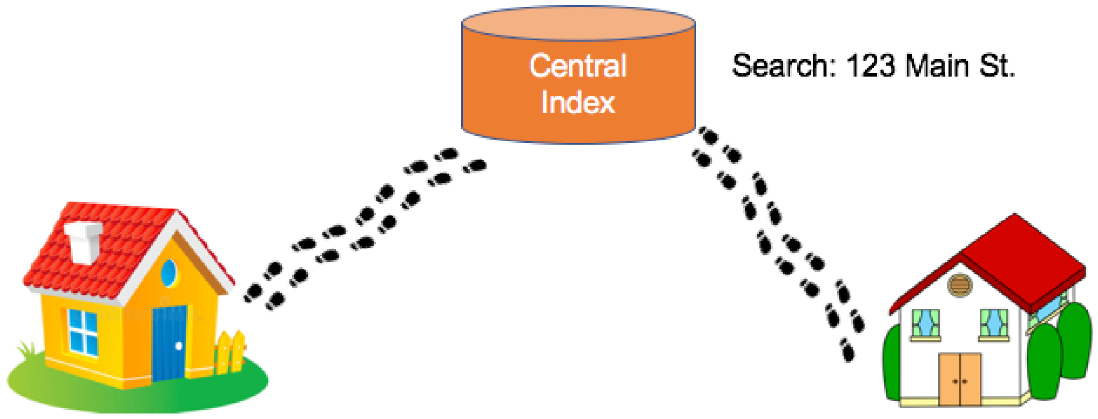


1. You walk out the door, but in the RDBMS world you are prohibited from walking to related items via pointers. You need to **go to the Central Index**.

2. Walk downtown to the government building where you get in line and wait your turn. Wait times are estimated at 6 hours.

3. When you get to the front of the line you go to the search agent counter.

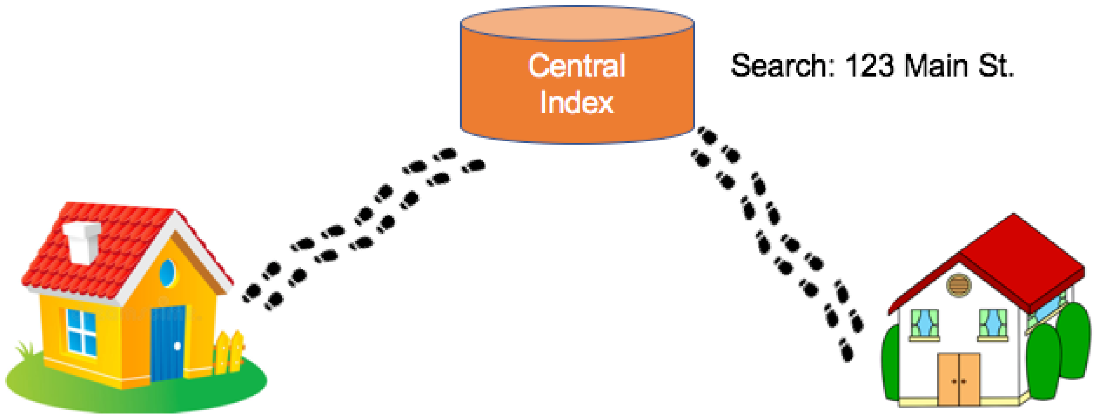
Problems in Relational DB



4. The search agent has a list of all the people in your town sorted by their address (**called the index**). They search the list (using a **binary search algorithm**). The problem is that **there are a lot of people in your town and the more people there are, the longer the search takes**. The search returns and they finally give you the GPS coordinates of your neighbor's house.

5. You take these GPS coordinates, enter them into your cell phone map and follow the directions to Ann's house. In total, it takes you 1,000 times a pointer hop (say 8 hours) to get there

Problems in Relational DB

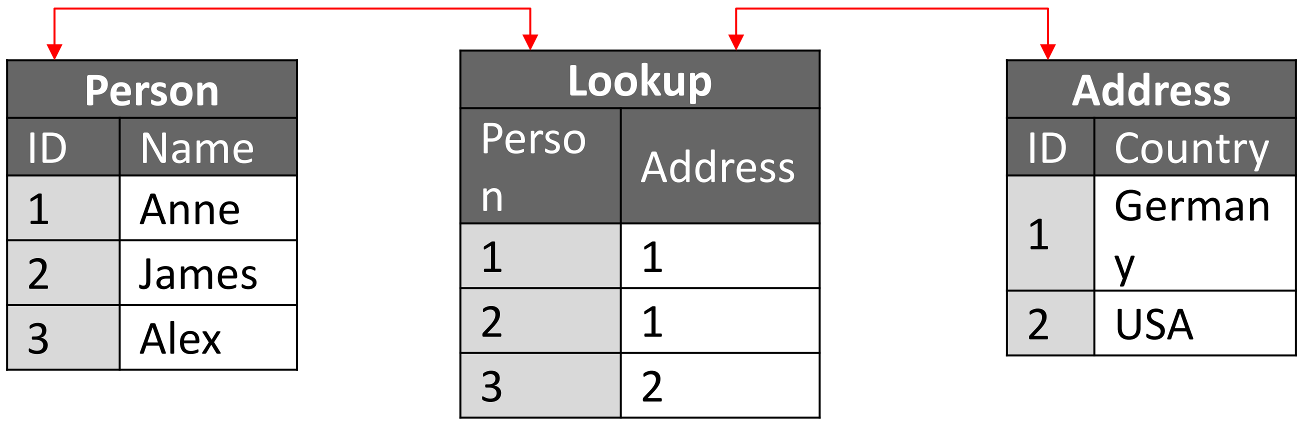


6. When you get there the pie is cold. She gives you a low Net Promoter Score on Neighborhood.com. She posts: “Nice guy but it takes him forever to deliver pies”.

Relationships in RDBMS

- Require foreign keys, and possibly a lookup table
- Traversing a foreign key requires an index lookup

The purpose of graphs is to do rapid traversal. The RDBMS model is too expensive for that.

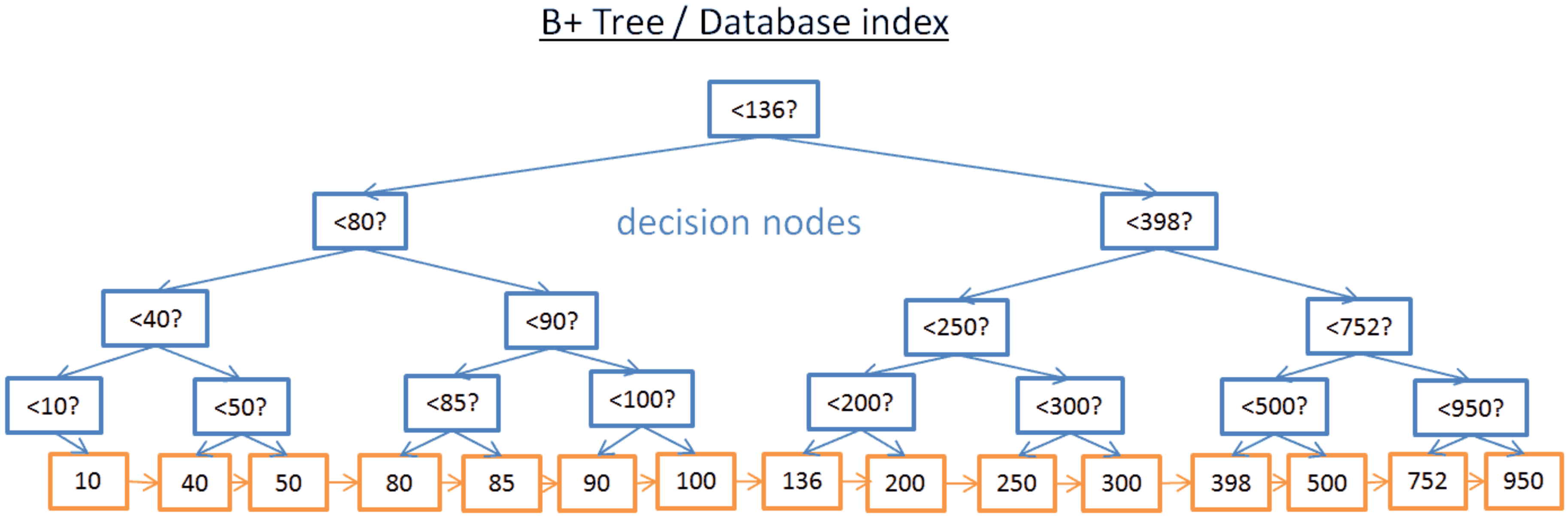


Person	
ID	Name
1	Anne
2	James
3	Alex

Lookup	
Person	Address
1	1
2	1
3	2

Address	
ID	Country
1	Germany
2	USA

RDBMS Index structure

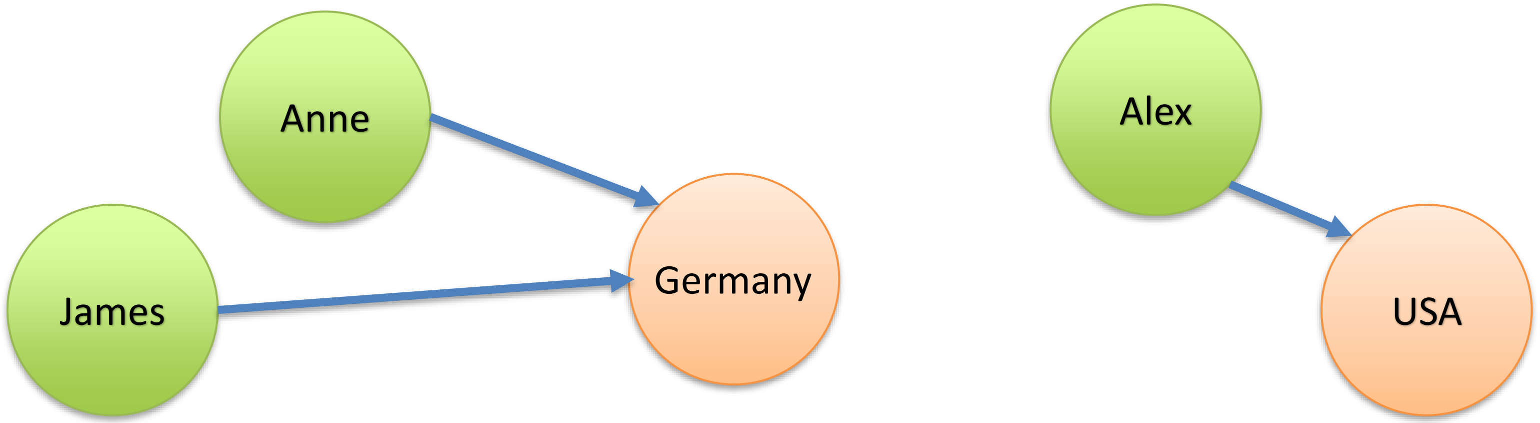


Pointers in IFA

IFA uses pointers to the target address

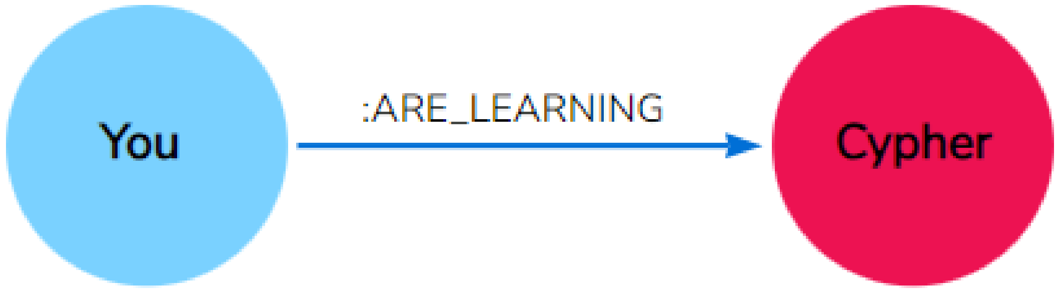
1		2		3		4		5	
Anne		USA		Alex		James		Germany	
Links	5	Links	3	Links	2	Links	5	Links	1, 4

- *No index lookups or table scans*
- *Reduced duplication of foreign keys*



What is Cypher?

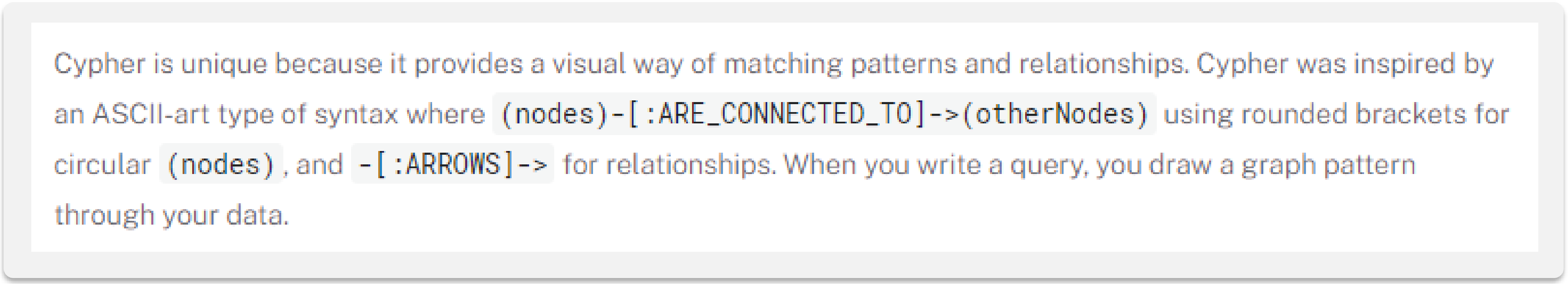
- Declarative query language
- Focuses on **what**, not on *how* to retrieve
- Uses keywords such as **MATCH**, **WHERE**, **CREATE** (SQL-like)
- Cypher **is heavily based on patterns** and is designed to recognize various versions of these patterns in data.



https://neo4j.com/docs/getting-started/cypher/

What is Cypher?

```
(:nodes)-[:ARE_CONNECTED_TO]->(:otherNodes)
```



Cypher is unique because it provides a visual way of matching patterns and relationships. Cypher was inspired by an ASCII-art type of syntax where `(nodes)-[:ARE_CONNECTED_TO]->(otherNodes)` using rounded brackets for circular `(nodes)`, and `-[:ARROWS]->` for relationships. When you write a query, you draw a graph pattern through your data.

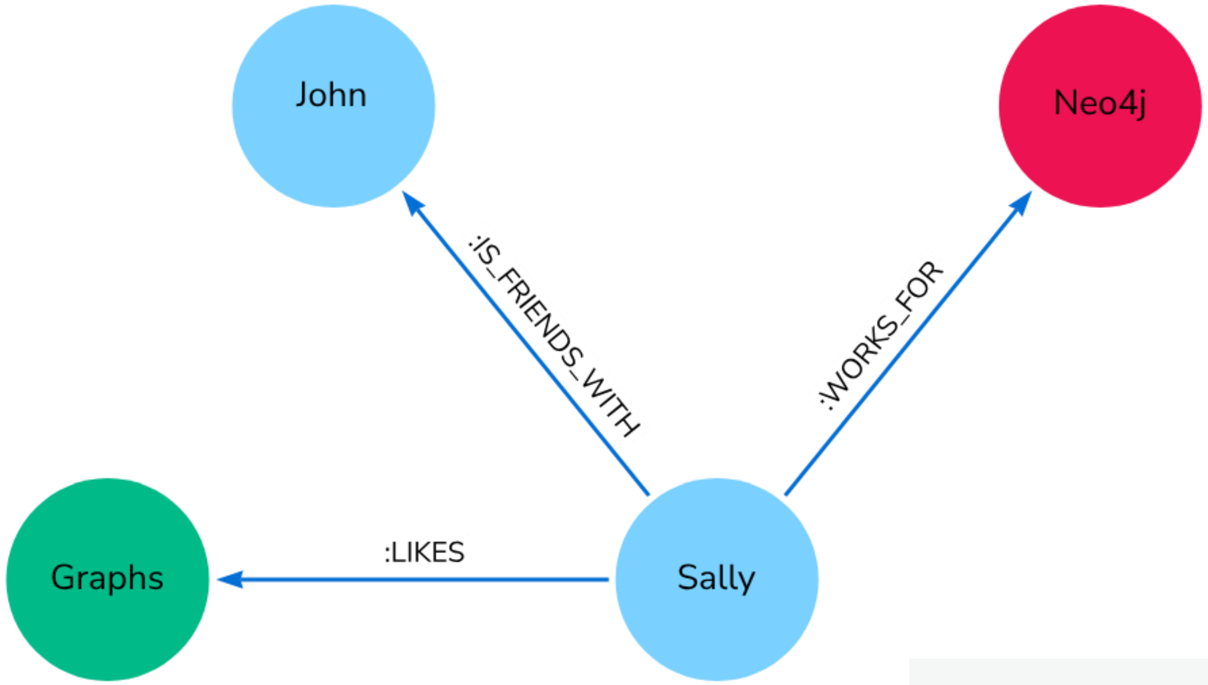
What is Cypher?

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

What is Cypher?



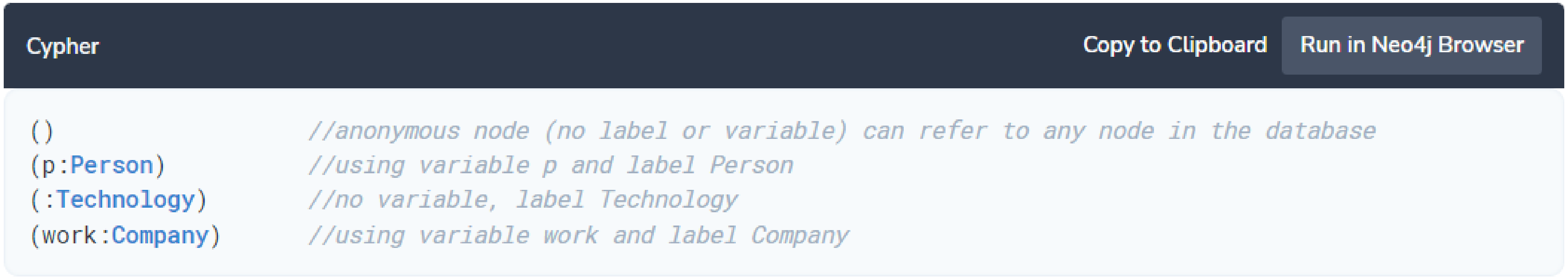
```
(:Sally)-[:LIKES]->(:Graphs)
(:Sally)-[:IS_FRIENDS_WITH]->(:John)
(:Sally)-[:WORKS_FOR]->(:Neo4j)
```

Creating a Node

In the following, we show an example for creating a node in a small social graph:

```
CREATE (ee:Person { name: "Emil", from: "Sweden", age: 29 })
```

- ***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



```
()                  //anonymous node (no label or variable) can refer to any node in the database
(p:Person)          //using variable p and label Person
(:Technology)       //no variable, label Technology
(work:Company)      //using variable work and label Company
```

Retrieving a Node

Given the following statement:

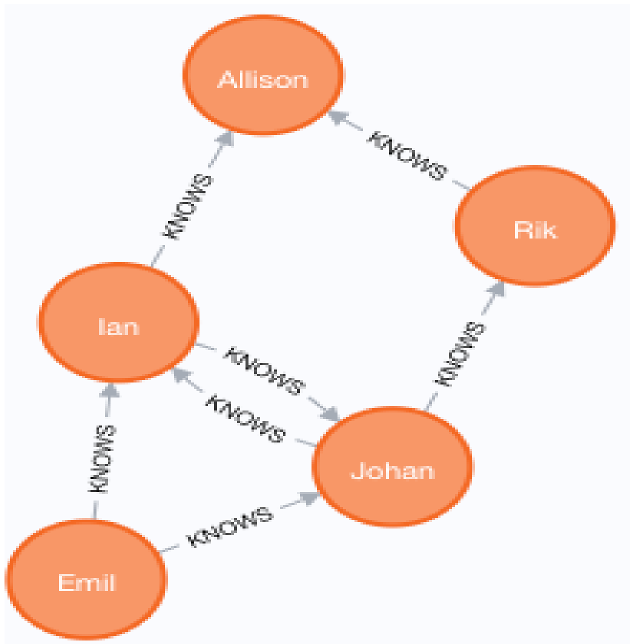
```
MATCH (ee:Person) WHERE ee.name = "Emil" RETURN ee;
```

- **MATCH clause** to specify a pattern of nodes and relationships
- (ee:Person) a single node pattern with label 'Person' which will assign matches to the variable 'ee'
- **WHERE clause** to constrain the results
- ee.name = "Emil" compares name property to the value "Emil"
- **RETURN clause** used to request particular results

Creating a Graph (1)

As shown in the following, ***CREATE clauses*** can create many nodes and relationships ***at once.*** Combining MATCH and CREATE allows **to attach a new sub-graph to existing nodes** of the current graph.

```
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)
```



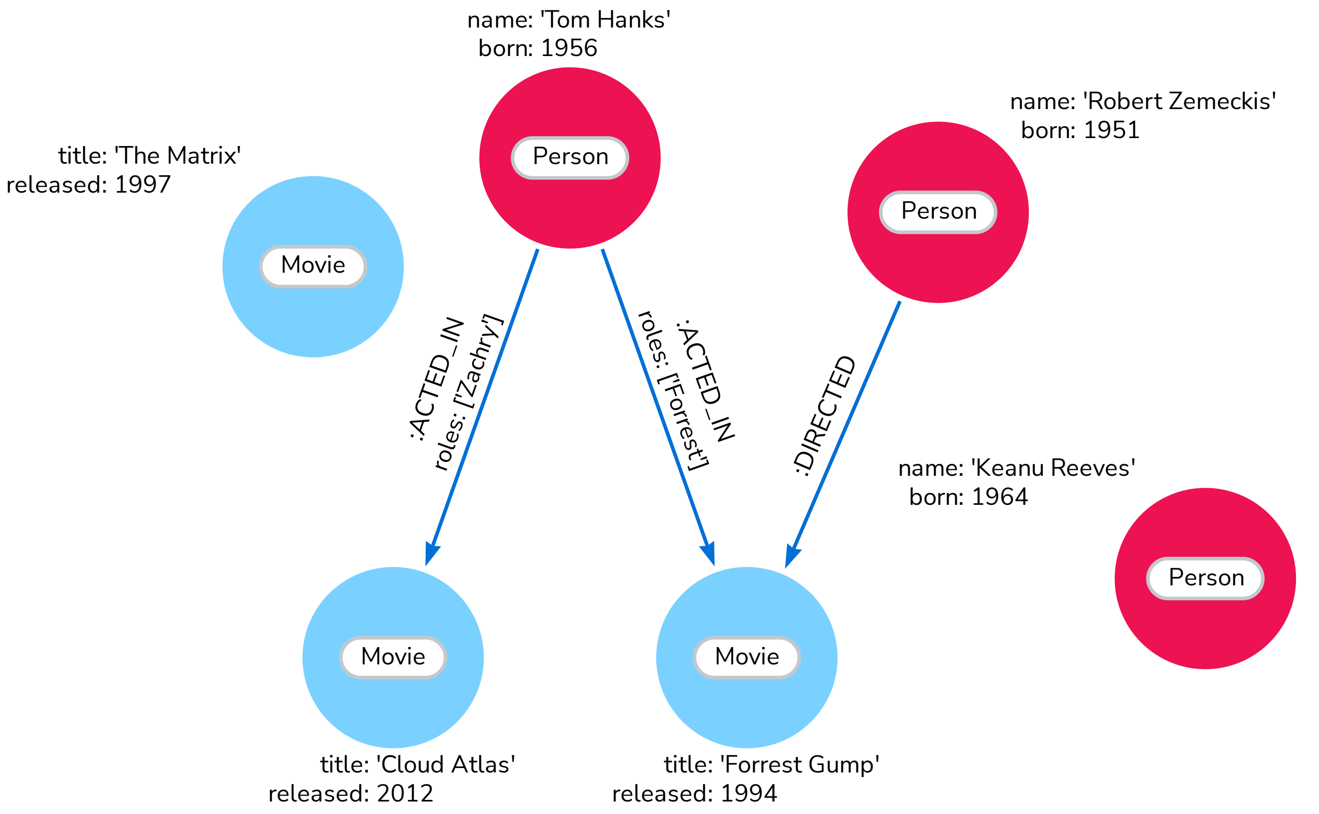
Creating a Graph (2)

As shown in the following, ***CREATE clauses*** can create many nodes and relationships ***at once.*** This time we create a graph from scratch without attaching it to existing nodes.

```
CREATE (matrix:Movie {title: 'The Matrix', released: 1997})
CREATE (cloudAtlas:Movie {title: 'Cloud Atlas', released: 2012})
CREATE (forrestGump:Movie {title: 'Forrest Gump', released: 1994})
CREATE (keanu:Person {name: 'Keanu Reeves', born: 1964})
CREATE (robert:Person {name: 'Robert Zemeckis', born: 1951})
CREATE (tom:Person {name: 'Tom Hanks', born: 1956})
CREATE (tom)-[:ACTED_IN {roles: ['Forrest']}]->(forrestGump)
CREATE (tom)-[:ACTED_IN {roles: ['Zachry']}]->(cloudAtlas)
CREATE (robert)-[:DIRECTED]->(forrestGump)
```

Creating a Graph (3)

As shown in the following, ***CREATE clauses*** can create many nodes and relationships ***at once.*** This time we create a graph from scratch without attaching it to existing nodes.



Adding a node and a relationship

- We may add a node to the graph as follows:

```
CREATE (pp:Person { name: "Pietro", from: "Italy",
learn: "neo4j" })
```

- We may add a relationship between two existing nodes as follows:

```
MATCH (ee:Person) WHERE ee.name = "Emil"
MATCH (pp:Person) WHERE pp.name = "Pietro"
CREATE (pp)-[:KNOWS {since: 2019}]->(ee)
```

Adding a node if not present

- We may add a node to the graph as follows:

```
MERGE (pp:Person { name: "Pietro", from: "Italy"})
```

- Merge a node and set properties if the node needs to be created

```
MERGE (keanu:Person { name: 'Keanu Reeves’ })
ON CREATE SET keanu.created = timestamp()
```

- Merge a node and set properties on found nodes

```
MERGE (pp:Person)
ON MATCH SET pp.found = TRUE
```

Pattern Matching

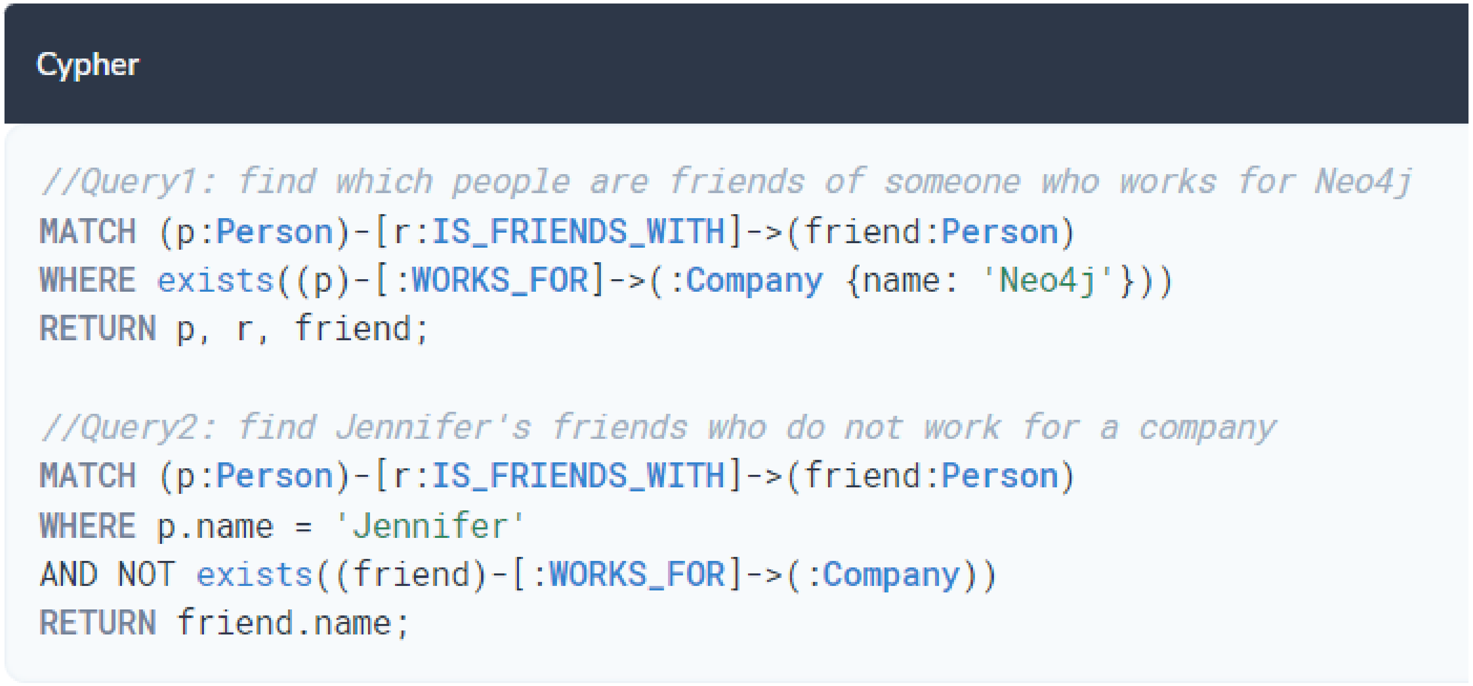
The following statement is used for finding Johan’s Friends:

```
MATCH (ee:Person)-[:KNOWS]-(friends)
WHERE ee.name = "Johan" RETURN 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 Johan’s friends

Filtering on patterns

One thing that makes graph unique is its **focus on relationships**: just as you can filter queries based on node labels or properties, you can also filter results based on relationships or patterns.



Cypher

```
//Query1: find which people are friends of someone who works for Neo4j
MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE exists((p)-[:WORKS_FOR]->(:Company {name: 'Neo4j'}))
RETURN p, r, friend;

//Query2: find Jennifer's friends who do not work for a company
MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE p.name = 'Jennifer'
AND NOT exists((friend)-[:WORKS_FOR]->(:Company))
RETURN friend.name;
```

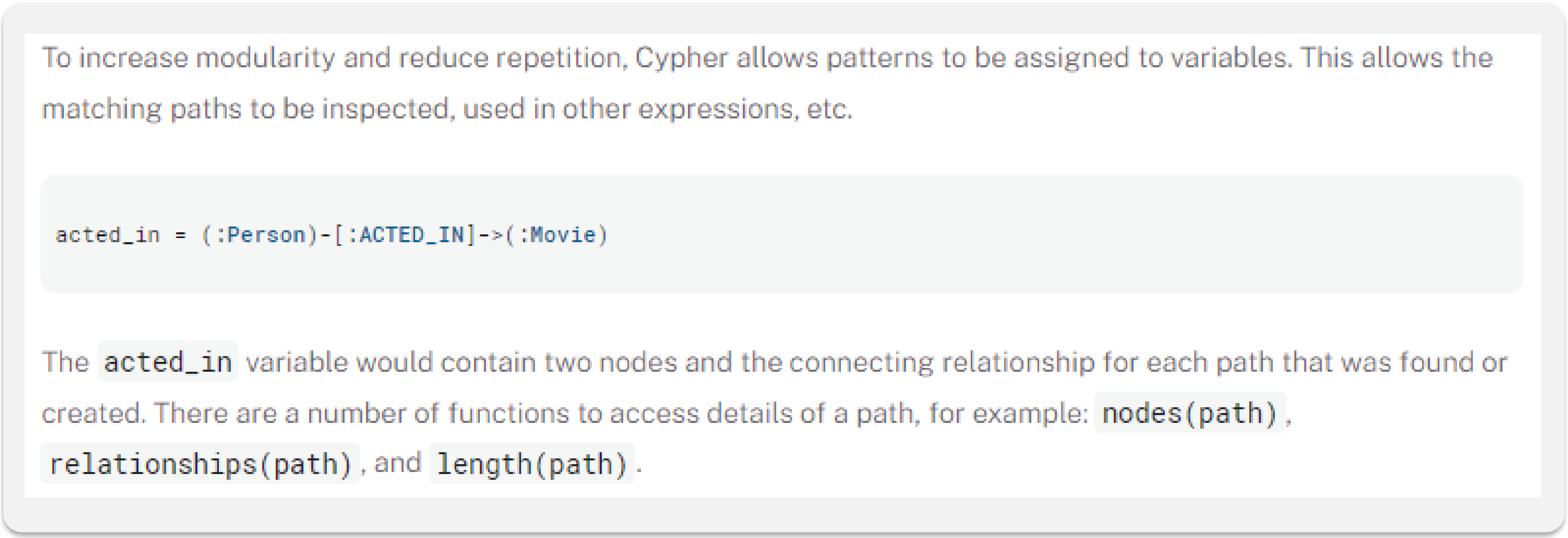
Recommending

Pattern matching can be used to make recommendations. Johan is learning to surf, so he may want to find a new friend who already does:

```
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

Patterns Variables



To increase modularity and reduce repetition, Cypher allows patterns to be assigned to variables. This allows the matching paths to be inspected, used in other expressions, etc.

```
acted_in = (:Person)-[:ACTED_IN]->(:Movie)
```

The `acted_in` variable would contain two nodes and the connecting relationship for each path that was found or created. There are a number of functions to access details of a path, for example: `nodes(path)`, `relationships(path)`, and `length(path)`.

Updating data with Cypher (1)

You may already have a node or relationship in the data, but you want to modify its properties. You can do this by matching the pattern you want to find and using the SET keyword to add, remove or update properties.

For example you could update Jennifer's node to add her birthdate:

Cypher

```
MATCH (p:Person {name: 'Jennifer'})
SET p.birthdate = date('1980-01-01')
RETURN p
```

If you want to change Jennifer's birthdate, you can use the same query above to find Jennifer's node again and put a different date in the SET clause.

Updating data with Cypher (2)

- In the following we show an example to ***set or update a property*** for a specific node:

```
MATCH (n:Person { name: 'Rik' })
SET n.surname = 'Taylor'
RETURN n.name, n.surname
```

- The setting may depend on to the result of a filtering:

```
MATCH (n :Person { name: 'Emil' })
SET (
    CASE
    WHEN n.from = 'Sweden’
    THEN n END ).worksIn = 'Malmo'
RETURN n.name, n.worksIn
```

Setting Properties on Relationships

The following statement shows how to set or modify the value of properties on relationships:

```
MATCH p=()-[r:KNOWS]->() SET r.since="2019" RETURN p
```

Setting Labels to Nodes

The following example shows **how to add a Label to a specific node**:

```
MATCH (ee:Person) WHERE ee.name = "Ian" SET ee:Young
RETURN labels(ee);
```

Removing a property

To ***remove*** a property, we can set the specific property to NULL.

```
MATCH (n:Person)
SET n.worksIn = NULL
RETURN n.name, n.worksIn
```

The example above removes the property worksIn from all the nodes.
Or, **equivalently**, you can remove the propery worksIn as follows:

```
MATCH (n:Person)
REMOVE n.worksIn
```

Delete Operations

- To delete a relationship of a node :

```
MATCH (n:Person { name: 'Ian' })-[r:KNOWS]->
      (c :Person { name:  'Allison' })
DELETE r
```

- To delete all relationships of a node :

```
MATCH (n { name: 'Ian'})-[r:KNOWS]-() DELETE r
```

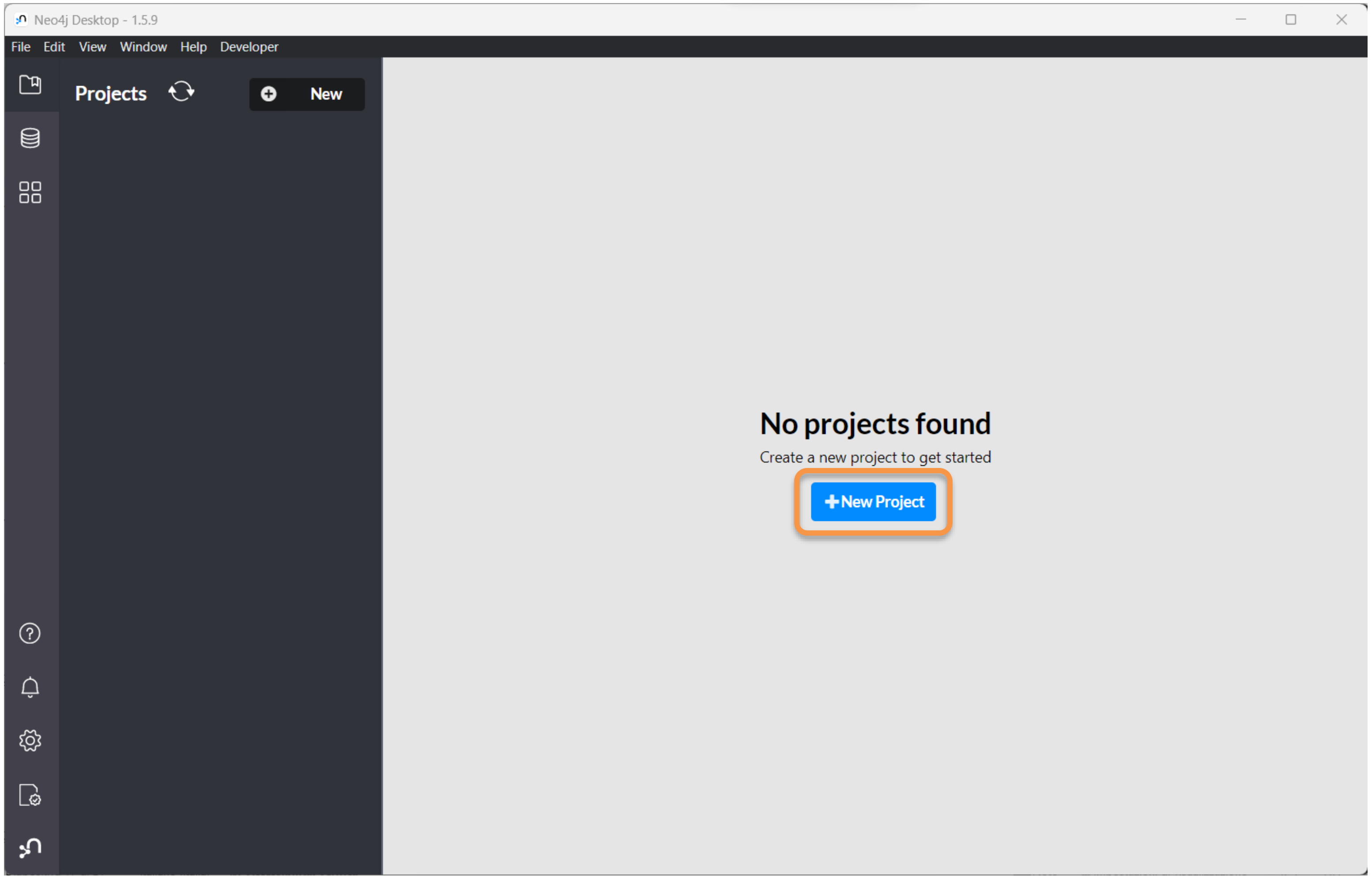
- To delete a node:

```
MATCH (n { name: 'Ian'}) DELETE n
```

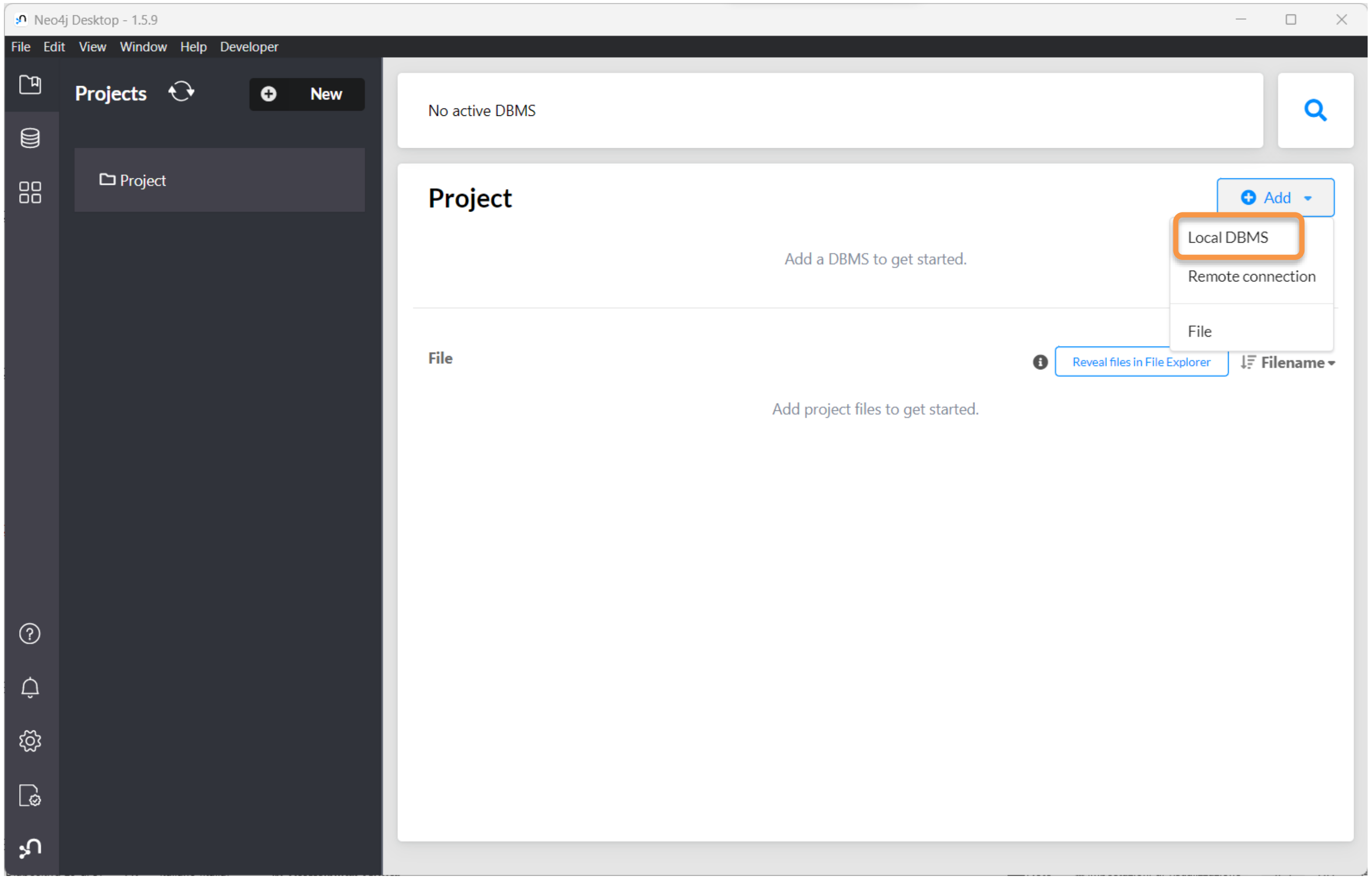
- To delete all nodes and relationships:

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

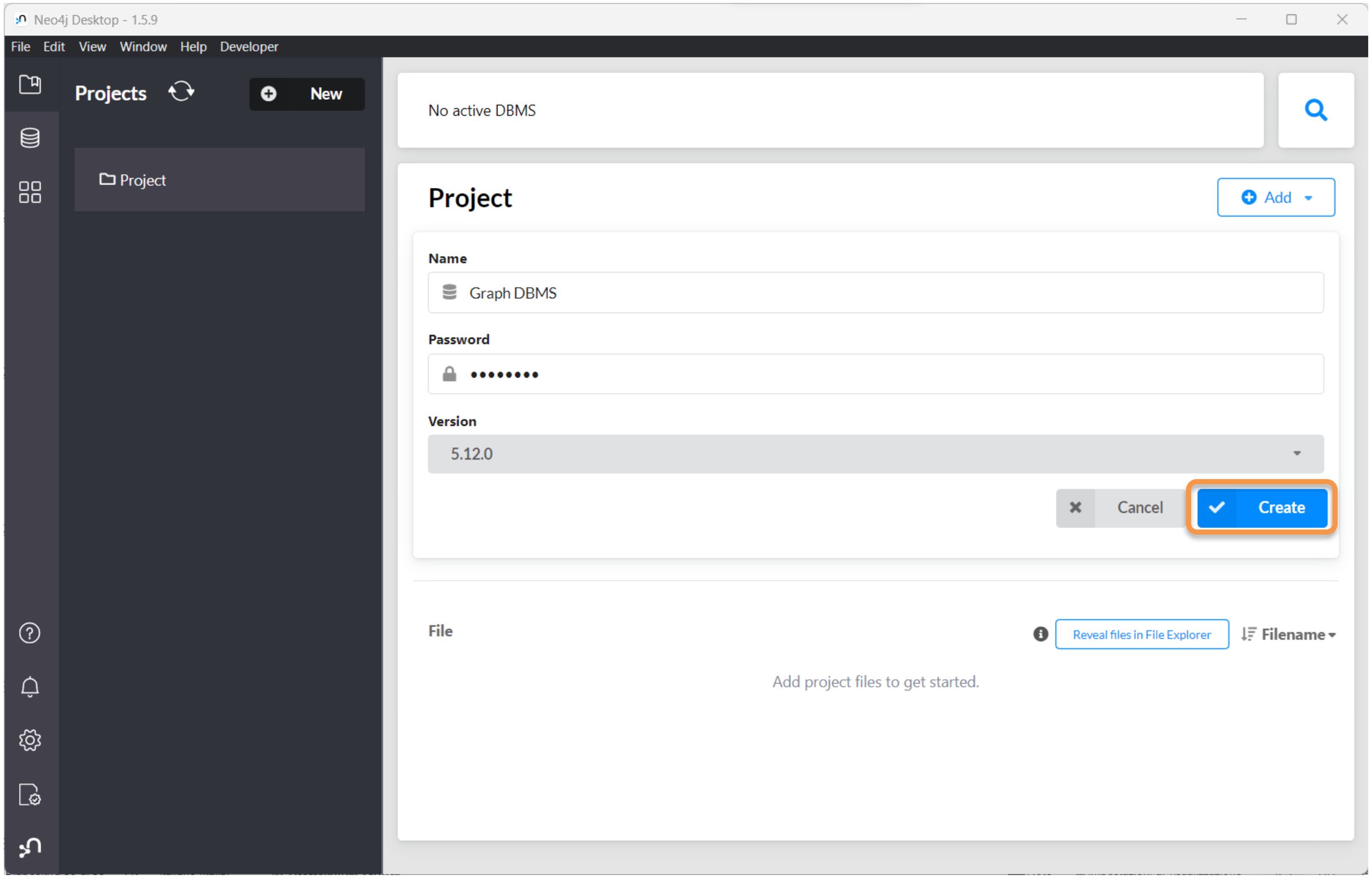
Setup Neo4j localhost deployment (I)


Neo4j Desktop - 1.5.9
File Edit View Window Help Developer
Projects
New
No projects found
Create a new project to get started
+New Project

Setup Neo4j localhost deployment (II)


Neo4j Desktop - 1.5.9
File Edit View Window Help Developer
Projects
New
Project
No active DBMS
Project
Add
Local DBMS
Remote connection
File
Add a DBMS to get started.
File
Reveal files in File Explorer
Filename
Add project files to get started.

Setup Neo4j localhost deployment (III)



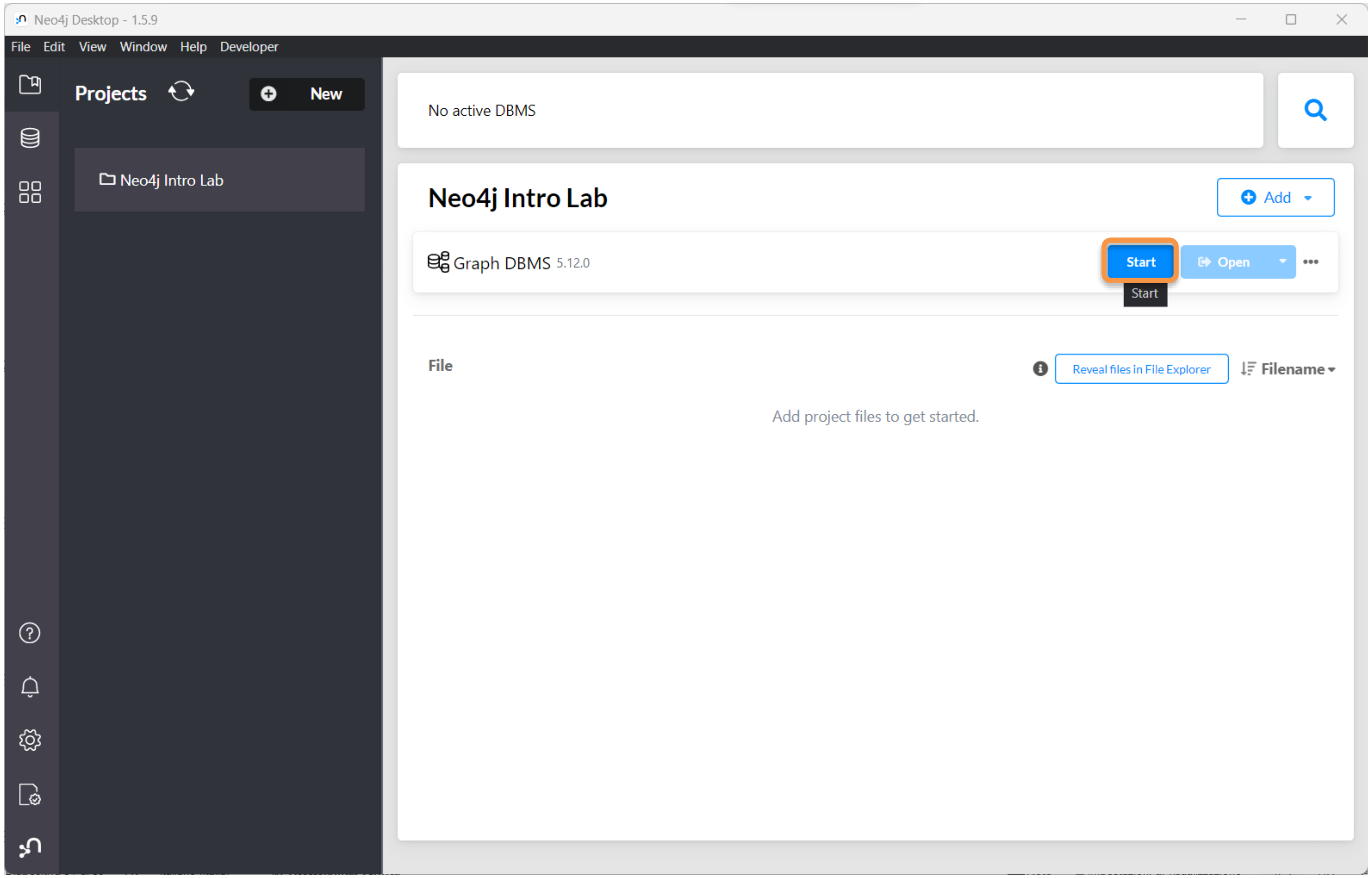
Name: Graph DBMS

Password: ••••••••

Version: 5.12.0

Add project files to get started.

Setup Neo4j localhost deployment (IV)



No active DBMS

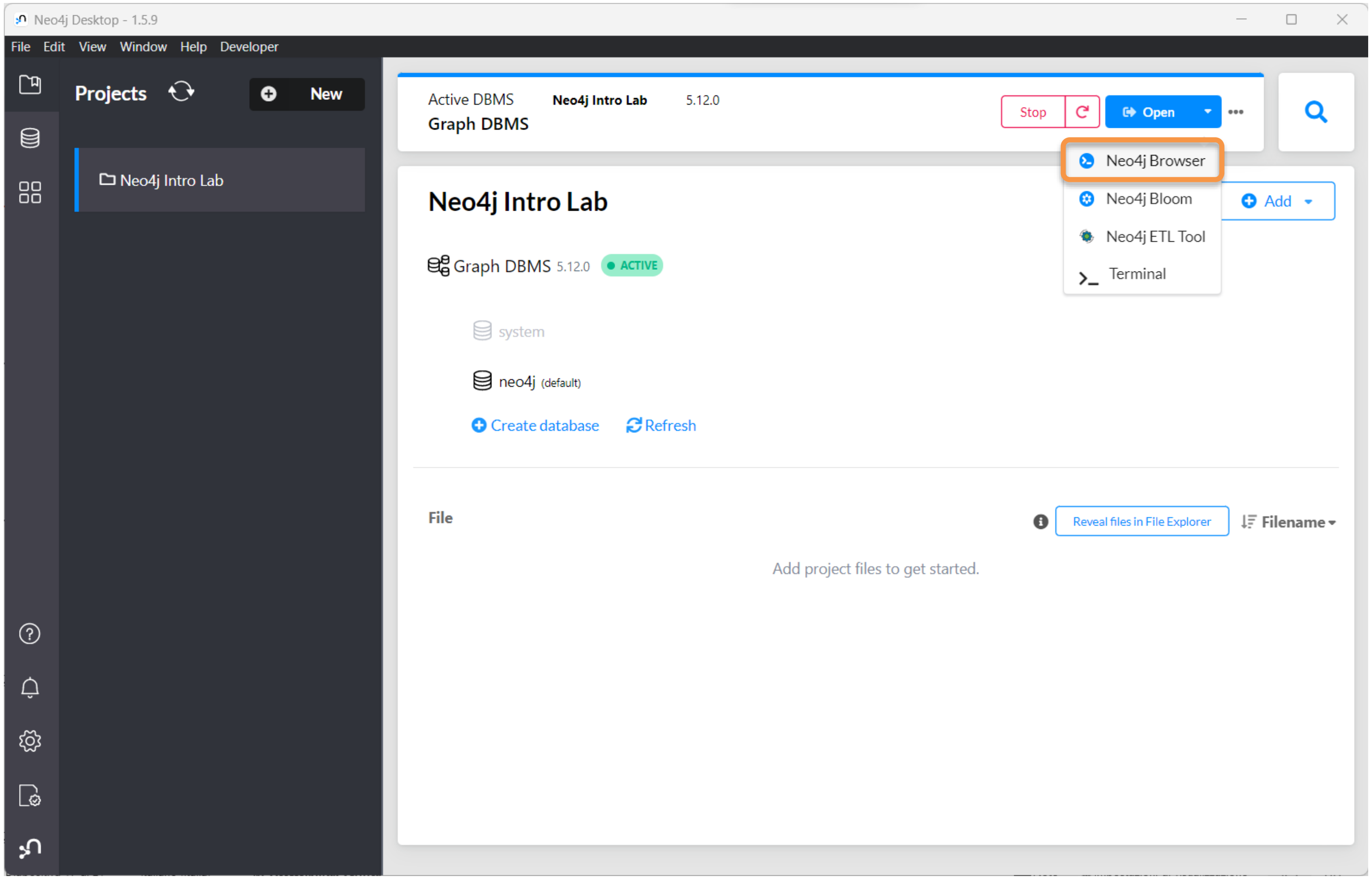
Neo4j Intro Lab

Graph DBMS 5.12.0

File

Add project files to get started.

Setup Neo4j localhost deployment (IV)



Neo4j Desktop - 1.5.9

File Edit View Window Help Developer

Projects New

Neo4j Intro Lab

Active DBMS Neo4j Intro Lab 5.12.0

Graph DBMS

Stop Open

Neo4j Browser

Neo4j Bloom

Neo4j ETL Tool

Terminal

Add

Neo4j Intro Lab

Graph DBMS 5.12.0 ACTIVE

system

neo4j (default)

Create database Refresh

File

Reveal files in File Explorer Filename

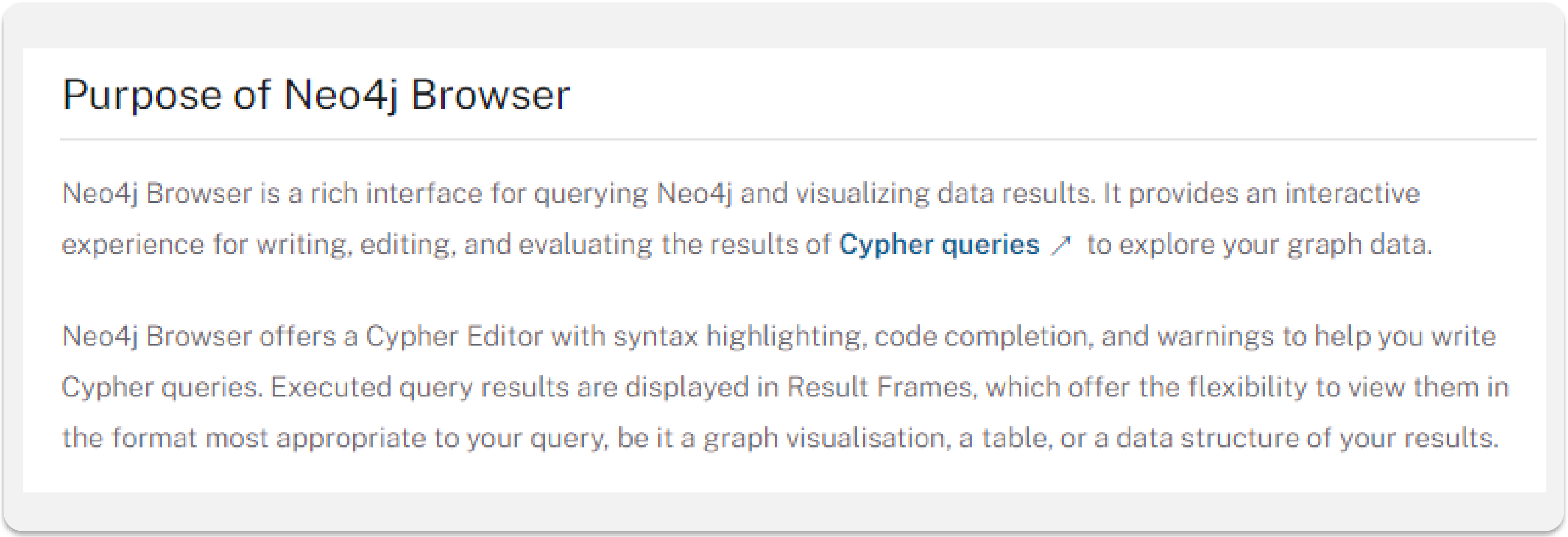
Add project files to get started.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Neo4j Browser User Interface

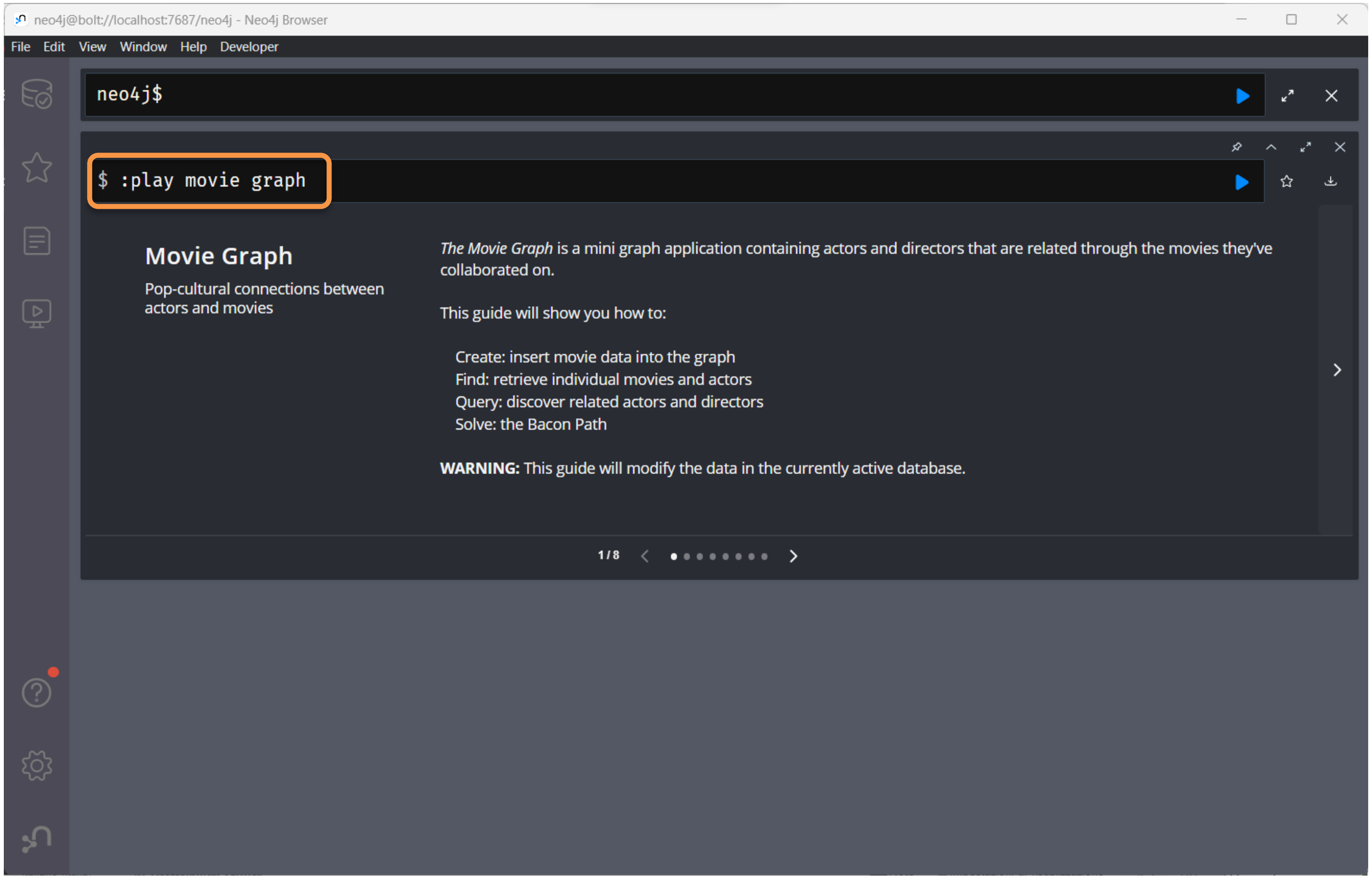


Purpose of Neo4j Browser

Neo4j Browser is a rich interface for querying Neo4j and visualizing data results. It provides an interactive experience for writing, editing, and evaluating the results of **Cypher queries** ↗ to explore your graph data.

Neo4j Browser offers a Cypher Editor with syntax highlighting, code completion, and warnings to help you write Cypher queries. Executed query results are displayed in Result Frames, which offer the flexibility to view them in the format most appropriate to your query, be it a graph visualisation, a table, or a data structure of your results.

Start playing with Movies GraphDB


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j$
1 / 8

```
$ :play movie graph
```

Movie Graph

Pop-cultural connections between actors and movies

The Movie Graph is a mini graph application containing actors and directors that are related through the movies they've collaborated on.

This guide will show you how to:

Create: insert movie data into the graph
Find: retrieve individual movies and actors
Query: discover related actors and directors
Solve: the Bacon Path

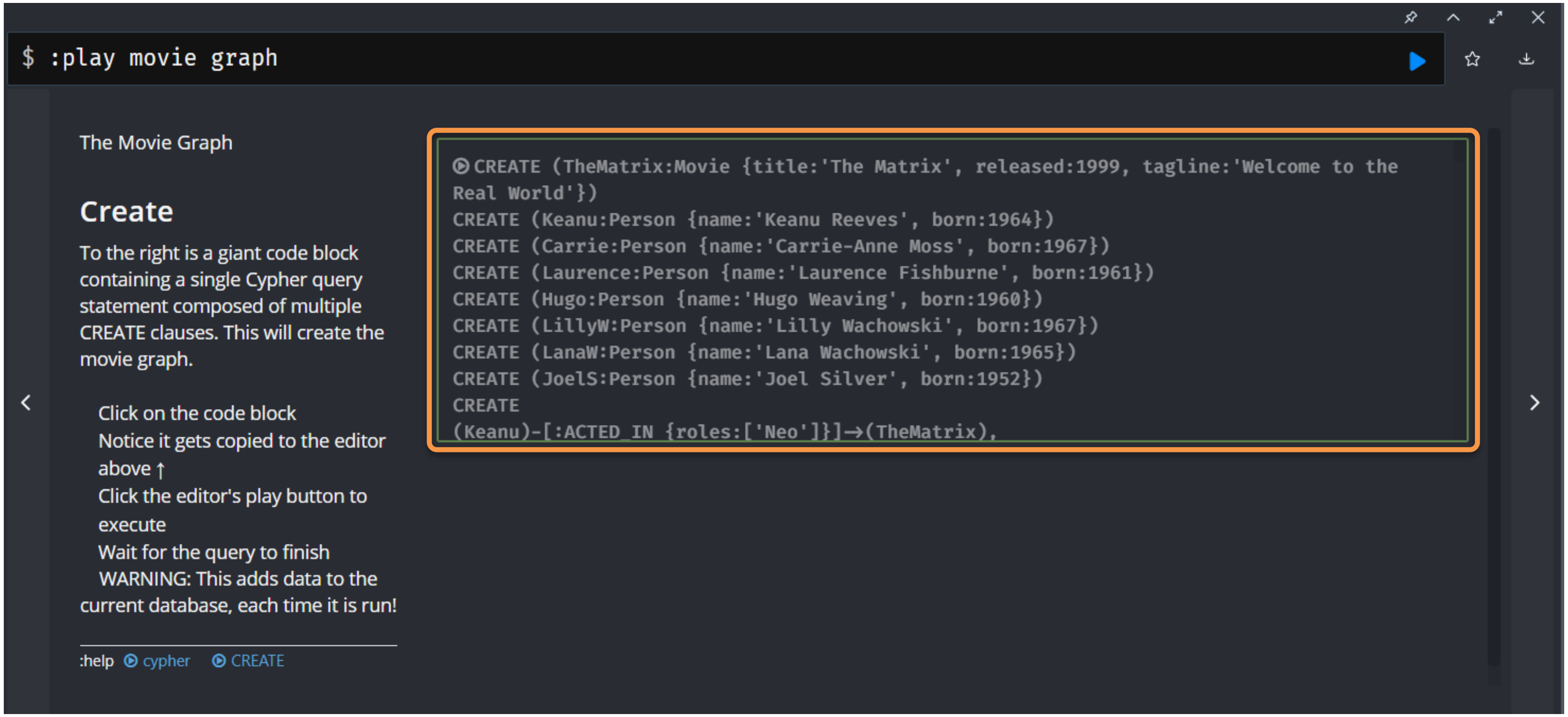
WARNING: This guide will modify the data in the currently active database.

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

IN SUPREMÆ DIGNITATIS 1343
UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Start playing with Movies GraphDB



```
$ :play movie graph
```

The Movie Graph

Create

To the right is a giant code block containing a single Cypher query statement composed of multiple CREATE clauses. This will create the movie graph.

Click on the code block
Notice it gets copied to the editor above ↑
Click the editor's play button to execute
Wait for the query to finish
WARNING: This adds data to the current database, each time it is run!

:help cypher CREATE

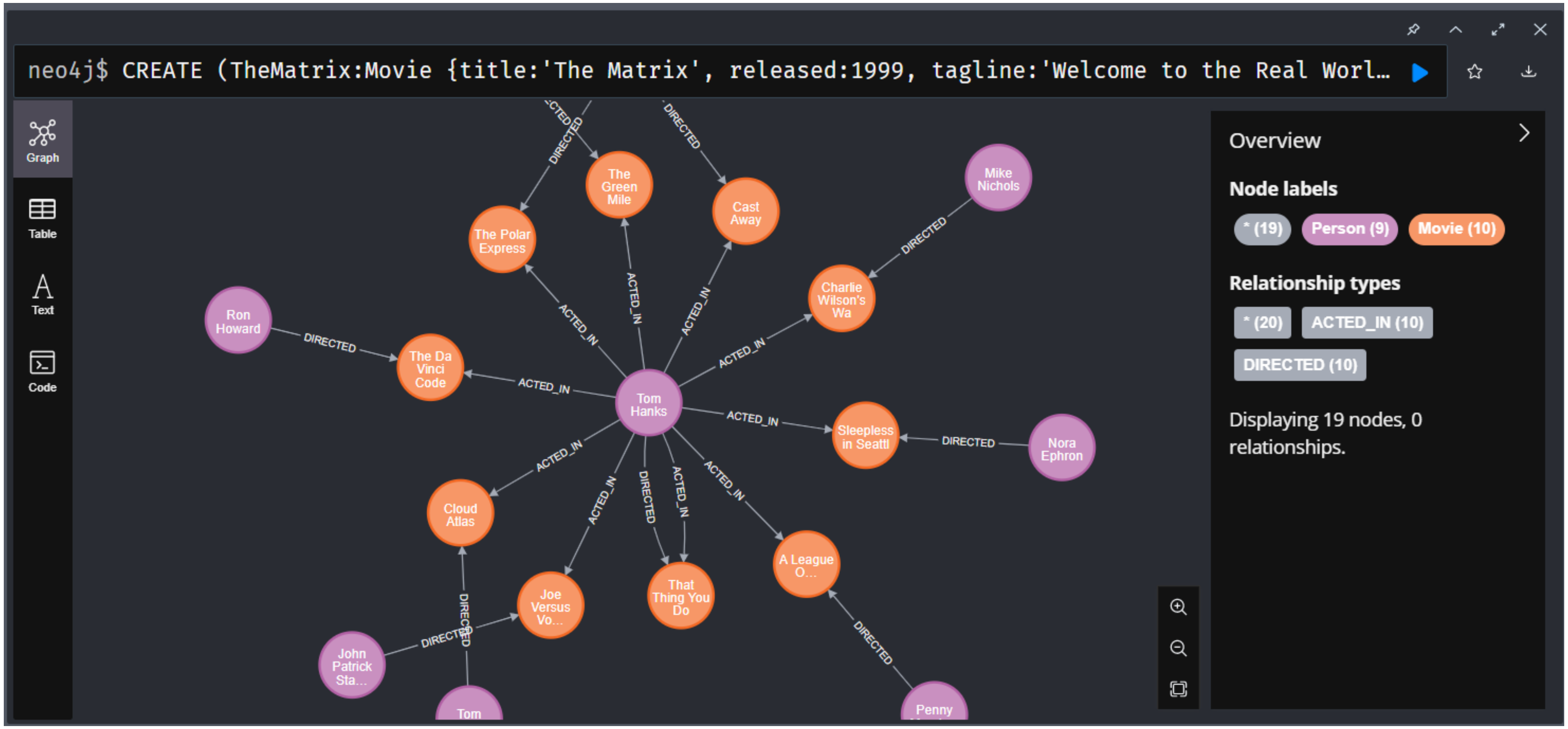
```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the
Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
(Keanu)-[:ACTED_IN {roles:['Neo']}]→(TheMatrix),
```

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Start playing with Movies GraphDB



```
neo4j$ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real Worl…
```

Overview

Node labels

* (19) Person (9) Movie (10)

Relationship types

* (20) ACTED_IN (10) DIRECTED (10)

Displaying 19 nodes, 0 relationships.

Examples (1)

- Find the actor named "Tom Hanks":
```
MATCH (tom {name: "Tom Hanks"}) RETURN tom
```

- Find 10 people:
```
MATCH (p:Person) RETURN p.name LIMIT 10
```

- Find movies released in either 1999 or 2003:
```
MATCH (m:Movie) WHERE m.released IN [1993, 2003]
RETURN m.title, m.released AS `Year of Release`
```

- List all movies that start with '*t*', ends with '*x*', and contains '*the*':
```
MATCH (m:Movie) WHERE m.title STARTS WITH 'T'
AND m.title ENDS WITH 'x' AND toLower(m.title)
CONTAINS 'the' RETURN m.title
```

Examples (2)

- Find Tom Hanks's co-actors:
  MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors) RETURN coActors.name

- Movies and actors up to 4 "hops" away from Kevin Bacon:
  MATCH (b:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
- **RETURN DISTINCT hollywood**

- Find the shortest path from Kevin Bacon to Meg Ryan:
  MATCH p = shortestPath((b:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"}))
- **RETURN p**

Examples (3)

- Extend Tom Hanks co-actors, to find co-co-actors who haven't worked with Tom Hanks:

  MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
- **(coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-(cocoActors)**
- **WHERE NOT (tom)-[:ACTED_IN]->()<-[:ACTED_IN]-(cocoActors) AND tom <> cocoActors**
- **RETURN cocoActors.name AS Recommended, count(*) AS Strength ORDER BY Strength DESC**

Examples (4)

- Find someone to introduce Tom Hanks to Tom Cruise:

```
MATCH (tom:Person {name:"Tom Hanks"})-
[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors),
  (coActors)-[:ACTED_IN]->(m2)<-[:ACTED_IN]-
(cruise:Person {name:"Tom Cruise"})
RETURN tom, m, coActors, m2, cruise
```

CHECK CYPHER CHEAT SHEET: https://neo4j.com/docs/cypher-cheat-sheet/current/

Exercises (1)

1. Find all the people who acted in the same movie that he or she directed. Return their names and movie titles
2. Find all the co-actors of Jack Nicholson. Return only co-actor names renamed as "Co-actor Name"
3. Find all the movies released only in the nineties
4. Find the shortest path between Christian Bale and Clint Eastwood
5. Find all movies that don’t include "Matrix" in the title, and where the movies were released in either 1999 or 2003. Return the title of those movies as Movie, the name of the actors as Name, and the movie. Order by movie title descending

CYPHER CHEAT SHEET: https://neo4j.com/docs/cypher-cheat-sheet/current/

Exercises (2)

6. Find all the *movies* that were released between 2000 and 2010 where the rating for those movies are either greater than 90 or less than 70. Return the movie title, rating, and movie released year. Order by rating descending and limit to 3 results.
7. Return distinct actors of movies that have two different reviewers. Sort by descending actor name and limit to 3 results.
8. Find all the movies where an actor played the role of "Neo" and the movie was released in 2003. (*Be careful, roles property is an array, use IN instead of =*)
9. Find all the movies Tom Hanks only acted in but did not direct and were released in 2004 or 2006. Return the movie title and movie released year as 'Year of release'

Further readings

A complete introductory tutorial on Cypher:
https://neo4j.com/graphacademy/

A reference card (collection of Cypher keywords and useful use cases:
https://neo4j.com/docs/cypher-refcard/current/

The complete Cypher manual
https://neo4j.com/docs/cypher-manual/current/