POLITECNICO
MILANO 1863

Exercise Session - Neo4j

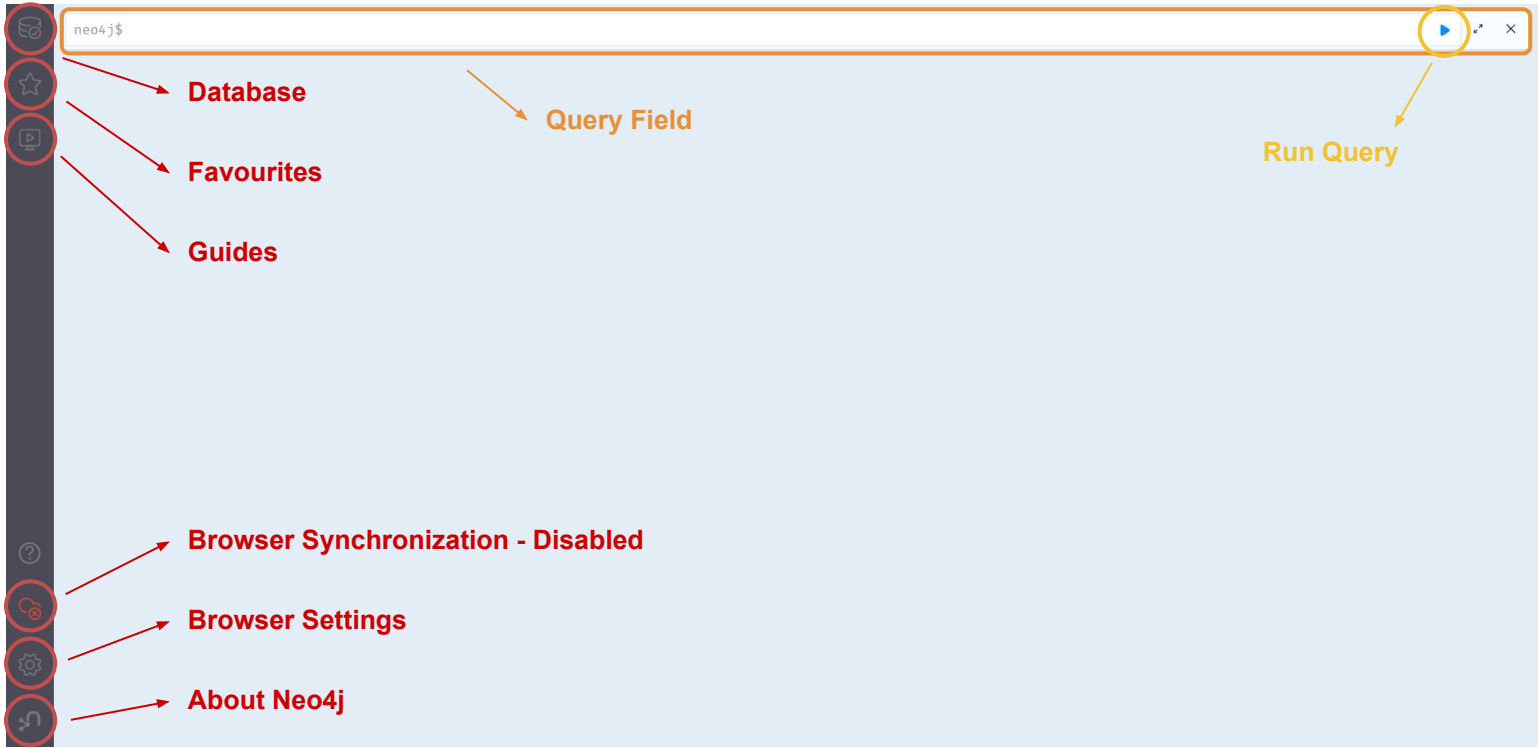
Andrea Tocchetti
andrea.tocchetti@polimi.it

Introduction to Neo4j

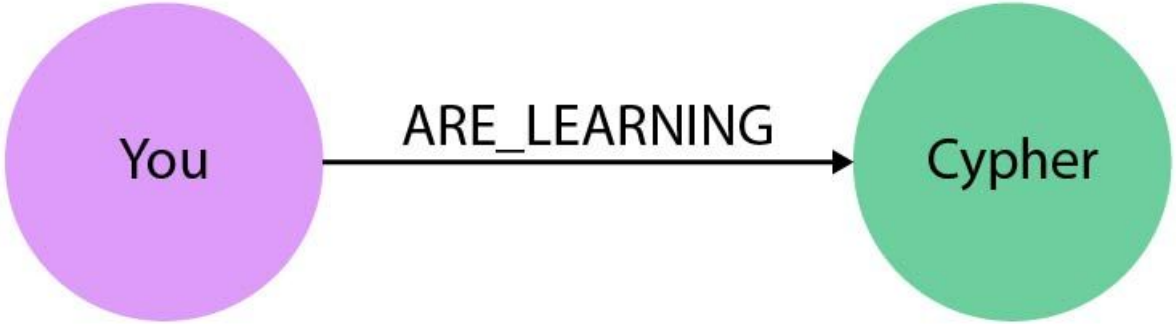
Neo4j is a graph database management system implemented in Java.

It is **A.C.I.D.** compliant. It has its own query language, named **Cypher**.

Neo4j Interface



Introduction to Cypher



Cypher is used to query a Neo4j graph, as well as update it.

The syntax is easy to understand and makes queries self-explanatory.

It focuses on the clarity of expressing what to retrieve from a graph, not on how to retrieve it.

Introduction to Cypher - CREATE Node

Variable : Node Label Node Properties

```
neo4j$ CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 })
```

Node - Identified by the parenthesis

The **CREATE** clause allows the creation of nodes and relationships.

Introduction to Cypher - CREATE Node

It is possible to **CREATE** multiple nodes and/or relationships at once.

```
CREATE (ee:Person { name: "Emil", from: "Sweden", klout: 99 }),
       (a:Person { name: "Richard", from: "UK", klout: 80 }),
       (j:Person { name: "Francis", from: ["Sweden","UK"], klout: 75 })
```

It is possible to **CREATE** simpler or even more complex nodes.

```
neo4j$ CREATE (ee:Person:Fisherman), (a)
```

The **RETURN** clauses return nodes and/or the properties of a node after its creation.

```
neo4j$ CREATE (a {name: 'Andy'}) RETURN a.name
```

Introduction to Cypher - MATCH

Variable : Node Label

Content Returned

```
neo4j$ MATCH (ee:Person) WHERE ee.name = "Emil" RETURN ee;
```

Condition

The **MATCH** clause returns all the nodes that match the conditions in query.

The **WHERE** clause identify all the conditions the nodes should match.

The **RETURN** clause identify what the query should return to the user.

Introduction to Cypher - CREATE Relationship

CREATE two Person(s) named Mark and Mike.

```
neo4j$ CREATE (e:Person {name: "Mark"}), (a:Person {name: "Mike"})
```

CREATE the relationship **KNOWS** between Mark and Mike.

MATCH Person(s)

```
MATCH
  (a:Person),
  (b:Person)
WHERE a.name = 'Mike' AND b.name = 'Mark'
CREATE (a)-[r:KNOWS]→(b)
RETURN type(r)
```

RETURN the type() of the relationship

CREATE the relationship

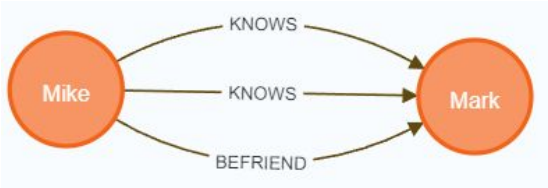
Introduction to Cypher - CREATE Relationship

It is possible to **CREATE** multiple relationships between the same entities.

Relationships can also be created with properties.

```
MATCH
  (a:Person),
  (b:Person)
WHERE a.name = 'Mike' AND b.name = 'Mark'
CREATE (a)-[r:KNOWS {since: 2015}]→(b), (a)-[h:BEFRIEND]→(b)
RETURN type(r)
```

N.B. The two **KNOWS** relationships we created are different even though they have the same name



Introduction to Cypher - DELETE

Delete Operation

```
neo4j$ MATCH (n) DETACH DELETE n
```

Match all Nodes

The **DELETE** clause allows the removal of nodes and relationships.

DETACH removes all the relationships before removing the nodes.

Introduction to Cypher - DELETE Node

Delete Operation

```
neo4j$ MATCH (n:Person {name: 'Ben'}) DELETE n
```

Match the Person(s)

The **DELETE** clause allows the removal of nodes and relationships.

Introduction to Cypher - DELETE Relationship

Delete Operation

```
neo4j$ MATCH (n {name: 'Ben'})-[r:KNOWS]→() DELETE r
```

Match the Relationship

The **DELETE** clause allows the removal of nodes and relationships.

Introduction to Cypher - Pattern Matching

Let’s find all of Emil’s friends

Condition to find Emil

```
neo4j$ MATCH (ee:Person)-[:KNOWS]-(friends) WHERE ee.name = "Emil" RETURN ee, friends
```

Match the relationship between Emil and his Friends

Return Emil and his Friends

Introduction to Cypher - Pattern Matching

Let’s find a **new** friend that can teach surf to Johan

Match the friend of a friend of Johan

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

Return the elements

Matching condition

Introduction to Cypher - DISTINCT

What would happen if we remove the **DISTINCT** clause from the query?

Let's use the **COUNT** clause to get the amount of nodes returned by the query.

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

```
RETURN COUNT(DISTINCT surfer)
```

1

```
RETURN COUNT(surfer)
```

3

Introduction to Cypher - PROFILE

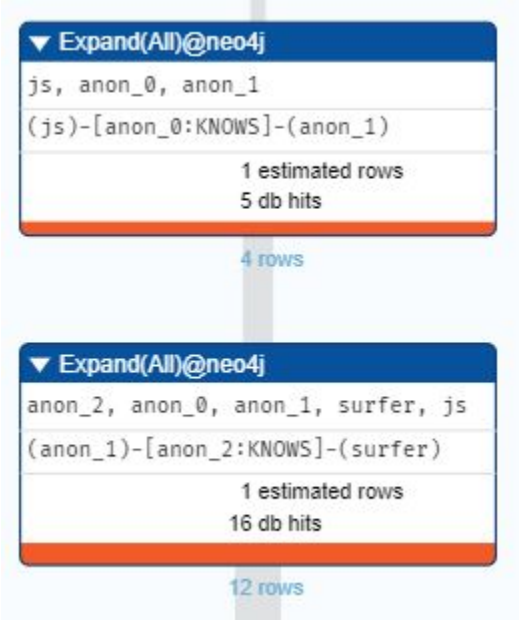
PROFILE provides the complete set of operations provided to perform the query.

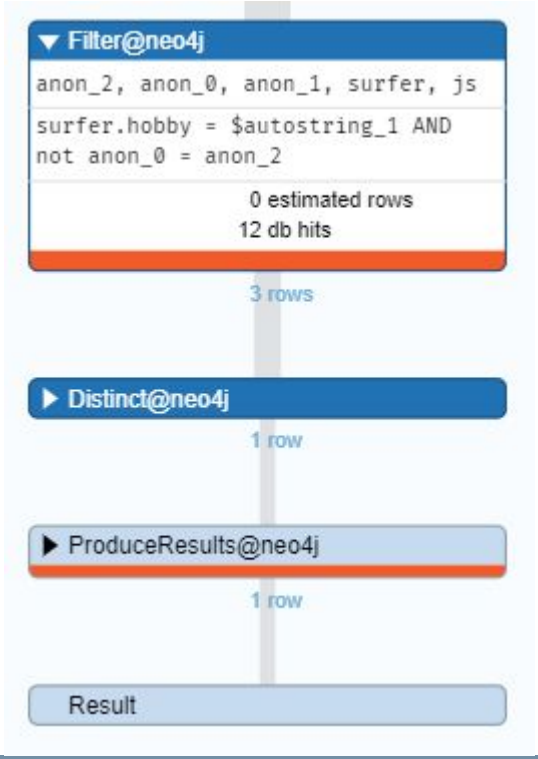
```
PROFILE MATCH (js:Person)-[:KNOWS]-()-[:KNOWS]-(surfer)
WHERE js.name = "Johan" AND surfer.hobby = "surfing"
RETURN DISTINCT surfer
```

PROFILE clauses

Query

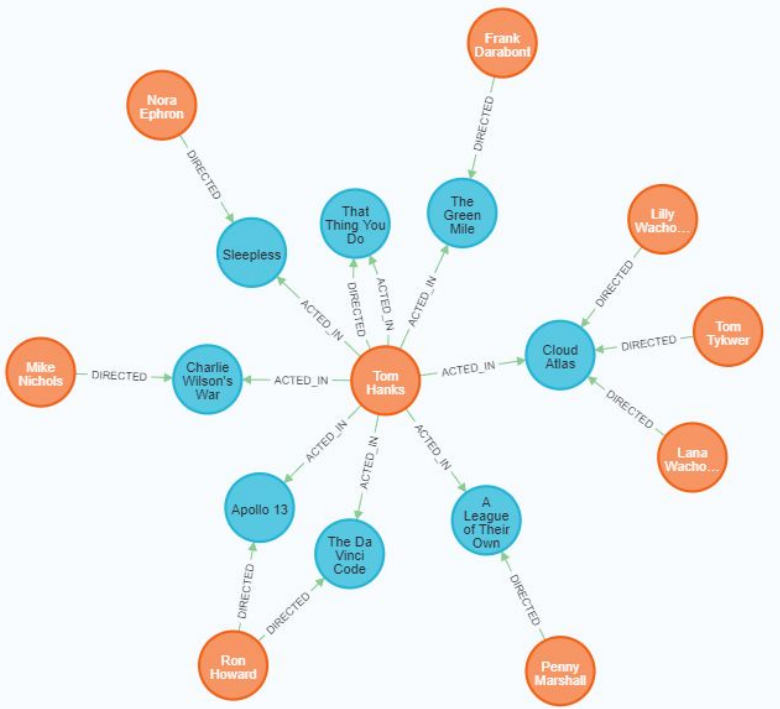
Introduction to Cypher - PROFILE






ANY
Questions?

Exercise - Movies and Actors



Nodes

- Actors
- Directors
  - name, born
- Films
  - title, release, tagline

Relationships

- ACTED_IN
  - roles
- DIRECTED
- PRODUCED
- WROTE

Exercise - Nodes Query

Find the actor named “Tom Hanks”.

```
neo4j$ MATCH (tom {name: "Tom Hanks"}) RETURN tom
```

Find the movie with title “Cloud Atlas”.

```
neo4j$ MATCH (cloudAtlas {title: "Cloud Atlas"}) RETURN cloudAtlas
```

Find 10 People.

```
neo4j$ MATCH (people:Person) RETURN people.name LIMIT 10
```

Exercise - Nodes Query

Find movies released in the 1990s.

```
MATCH (nineties:Movie)
WHERE nineties.released ≥ 1990 AND nineties.released < 2000
RETURN nineties.title
```

Find the number of people born before 1970.

```
MATCH (person:Person)
WHERE person.born < 1970
RETURN COUNT(person.name)
```

Exercise - Pattern Query

List all “Tom Hanks” movies.

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]→(tomHanksMovies)
RETURN tom,tomHanksMovies
```

Who directed “Cloud Atlas”?

Match “Cloud Atlas”

Inverse Relationship

```
MATCH (cloudAtlas {title: "Cloud Atlas"})←[:DIRECTED]-(directors)
RETURN directors.name
```

Return the name of the director

Exercise - Pattern Query

Find Tom Hanks' co-actors.

Match the film Tom Hanks acted in

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]→(m)←[:ACTED_IN]-(coActors)
RETURN coActors.name
```

Return the name of the co-actors

Match the actors who acted in the same movies as Tom Hanks

Exercise - Pattern Query

Find how people are related to “Cloud Atlas”.

Match all people related to “Cloud Atlas”

```
MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
RETURN people.name, Type(relatedTo), relatedTo
```

Return the name, type of relation and relation

```
MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
RETURN people.name, Type(relatedTo), relatedTo
```

Only variable name (any relationship)

No variable name

Introduction to Cypher - Nodes Distance

Movies and actors up to 4 "hops" away from “Kevin Bacon”.

Distance computed w.r.t. relationships

Distance from 1 to 4

Any node

```
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

Distinct clause, as many paths may lead to the same node

Introduction to Cypher - Built in Functions

Find the shortest path of any relationships from “Kevin Bacon” to “Meg Ryan”

Invoke shortest path function

```
MATCH p = shortestPath(
(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Return start node, end node and path

Shortest path pattern

Introduction to Cypher - Built in Functions

Find the shortest path of ACTED_IN relationships from “Kevin Bacon” to “Al Pacino”

Match starting and end node

Invoke shortest path function

```
MATCH (KevinB:Person {name: 'Kevin Bacon'} ),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[:ACTED_IN*]-(Al))
WHERE all(r IN relationships(p) WHERE r.role IS NOT NULL)
RETURN p
```

Return start node, end node and path

Condition (discussed later)

Introduction to Cypher - Predicate Functions

Let's take a look at the previous **WHERE** condition

```
WHERE all(r IN relationships(p) WHERE r.role IS NOT NULL)
```

all() function

Returns **TRUE** if the predicate holds for all elements in the given list.

IN operator

Access the elements within the list of relationships

relationship() function

Returns a list containing all the relationships in a path.

Introduction to Cypher - WITH

Using **WITH**, you can manipulate the output before it is passed on to the following query parts. it is usually combined with other clauses, like

- **ORDER BY** - Sort the result of the query.
- **LIMIT** - Limit the amount of results provided by the query.

It can also be used to

- Introduce aggregates which can then be used in predicates in **WHERE**.
- Alias expressions that are introduced into the results using the aliases **AS** the binding name.
- Separate reading from updating of the graph.

Let’s see some examples to learn how **WITH** works.

Introduction to Cypher - WITH

FILTER - Filter w.r.t. an aggregate function.

```
MATCH (person:Person)-[ACTED_IN]→(movie)
WITH person,        count(*) AS movieCount
WHERE movieCount > 1
RETURN person, movieCount
```

Alias

Count the movies in which each person acted in

Extract the actors who acted in at least 2 movies

N.B. The query will **only** be able to read the variables stated within the **WITH** clause. If the *person* variable wouldn't have been included within the clause, I wouldn't have been able to **RETURN** it.

Introduction to Cypher - WITH

ORDER BY & LIMIT - Filter w.r.t. an aggregate function.

```
MATCH (n:Person)
WITH n AS person
ORDER BY person.name DESC
LIMIT 3
RETURN collect(person.name)
```

Alias

Sort by name

Limit the returned values

collect() function

Return a collection with the results

Introduction to Cypher - Warning

```
MATCH (KevinB:Person {name: 'Kevin Bacon'} ),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[:ACTED_IN*]-(Al))
WHERE all(r IN relationships(p) WHERE r.role IS NOT NULL)
RETURN p
```

```
MATCH
  (a:Person),
  (b:Person)
WHERE a.name = 'Mike' AND b.name = 'Mark'
CREATE (a)-[r:KNOWS]→(b)
RETURN type(r)
```

```
MATCH p = shortestPath(
(bacon:Person {name:"Kevin Bacon"})-[*]-(meg:Person {name:"Meg Ryan"})
)
RETURN p
```

Sometimes when Cypher queries are performed, **WARNINGS** may arise.

They may arise for different reasons. For example, some operators may be deprecated or the query is performing a lot of operations e.g., cartesian products among the nodes or unbound number of relationships.

Advanced Cypher - Shortest Path

Depending on the predicates to be evaluated, Neo4j plans the shortest path in different ways.

By default, Neo4j uses a **Fast Bidirectional Breadth-first Search Algorithm** if the conditions can be evaluated whilst searching for the path.

e.g., all nodes must have the *Person* label.

e.g., no nodes should have a *name* property.

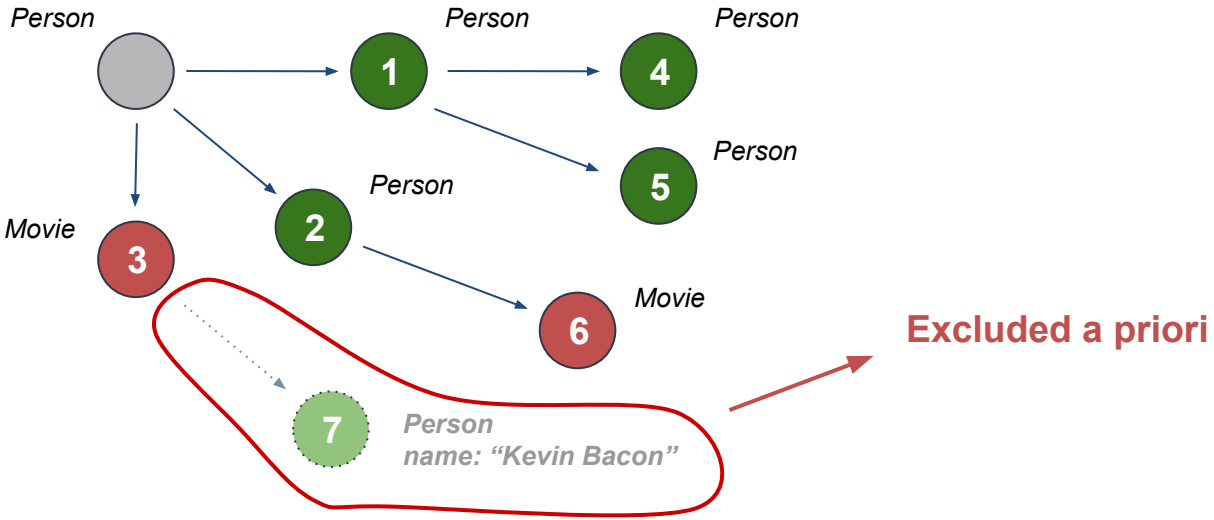
If the predicates need to inspect the whole path before deciding on whether it is valid or not, Neo4j may have to resort to using a **Slower Exhaustive Depth-first Search Algorithm** to find the path.

e.g., at least one node contains the property *name* = *"Kevin Bacon"*.

When the **Exhaustive Search** is planned, it is still only executed when the **Fast Algorithm** fails to find any matching paths.

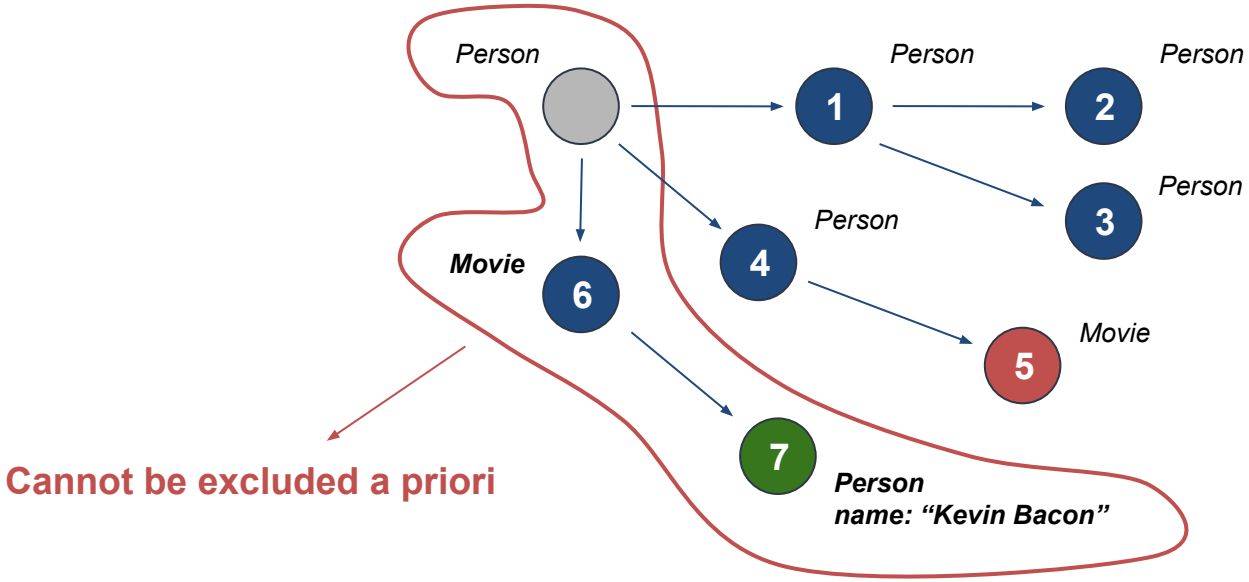
Advanced Cypher - Shortest Path

Fast Algorithm - All nodes must have the *Person* label.



Advanced Cypher - Shortest Path

Exhaustive Algorithm - At least one node contains the property *name* = *"Kevin Bacon"*.



Advanced Cypher - Shortest Path

```
MATCH (KevinB:Person {name: 'Kevin Bacon'} ),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[:ACTED_IN*]-(Al))
WHERE all(r IN relationships(p) WHERE r.role IS NOT NULL)
RETURN p
```

Fast Algorithm - Conditions can be evaluated as the function is performed.

```
MATCH (KevinB:Person {name: 'Kevin Bacon'}),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[*]-(Al))
WHERE length(p) > 1
RETURN p
```

Documentation - Predicates used in the **WHERE** clause that apply to the shortest path pattern are evaluated before choosing the shortest matching path.

Advanced Cypher - Shortest Path

```
MATCH (KevinB:Person {name: 'Kevin Bacon'}),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[*]-(Al))
WHERE length(p) > 1
RETURN p
```

Exhaustive Algorithm - Must check that the whole path follows the predicate before validity is evaluated.

```
MATCH (KevinB:Person {name: 'Kevin Bacon'}),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[*]-(Al))
WITH p
WHERE length(p) > 1
RETURN p
```

The inclusion of the **WITH** clause means that the query plan will not include the fallback to the **Slower Exhaustive Search Algorithm**.

Advanced Cypher - Shortest Path

```
MATCH (KevinB:Person {name: 'Kevin Bacon'} ),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[:ACTED_IN*]-(Al))
WHERE all(r IN relationships(p) WHERE r.role IS NOT NULL)
RETURN p
```

Fast Algorithm
Always returns a result, if exists

```
MATCH (KevinB:Person {name: 'Kevin Bacon'}),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[*]-(Al))
WHERE length(p) > 1
RETURN p
```

Exhaustive Algorithm
Always returns a result, if exists

```
MATCH (KevinB:Person {name: 'Kevin Bacon'}),
      (Al:Person {name: 'Al Pacino'}),
      p = shortestPath((KevinB)-[*]-(Al))
WITH p
WHERE length(p) > 1
RETURN p
```

Fast Algorithm
Not guaranteed to return a result, even if it exists.

Graph DB Approaches - Recommendation

Find actors that “Tom Hanks” hasn't yet worked with, but his co-actors have. Find someone who can introduce Tom to his potential co-actor.

Match “Tom Hanks” co-actors

Match the actors who acted in movies with “Tom Hanks” co-actors

```
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
```

Check that the coco-actors do not acted with “Tom Hanks”

Ignore “Tom Hanks”

Count the number of times the coco-actor is found and **ORDER BY** the count

Graph DB Approaches - Recommendation

Find someone to introduce Tom Hanks to Tom Cruise.

Match “Tom Hanks” co-actors

Match co-actors who acted with “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
```

Return the nodes that defines the paths


ANY
Questions?