Big Graph Processing Systems

Part II: Property Graphs

▶ **Chapter 1:** A Concrete Query Language

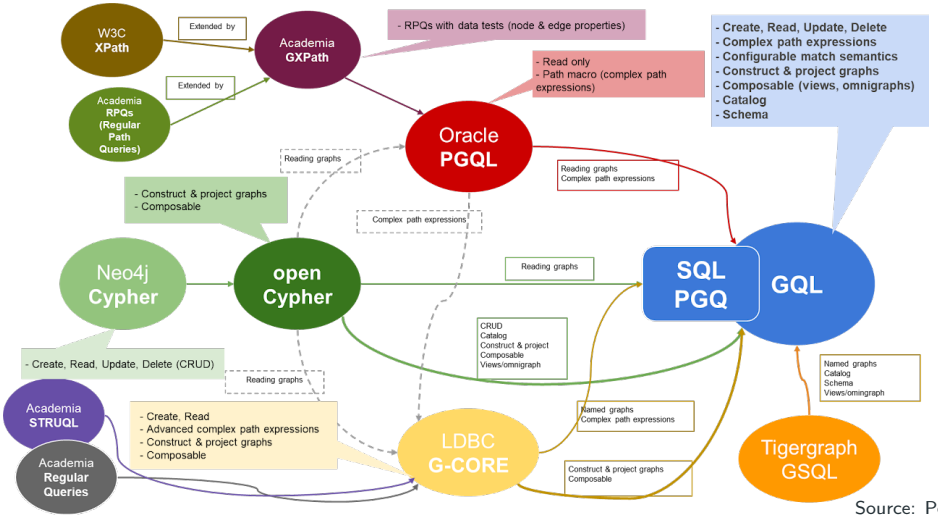
Christopher Spinrath

CNRS – LIRIS – Lyon 1 Université

DISS Master 2025

This presentation is an adaption of slides from Angela Bonifati

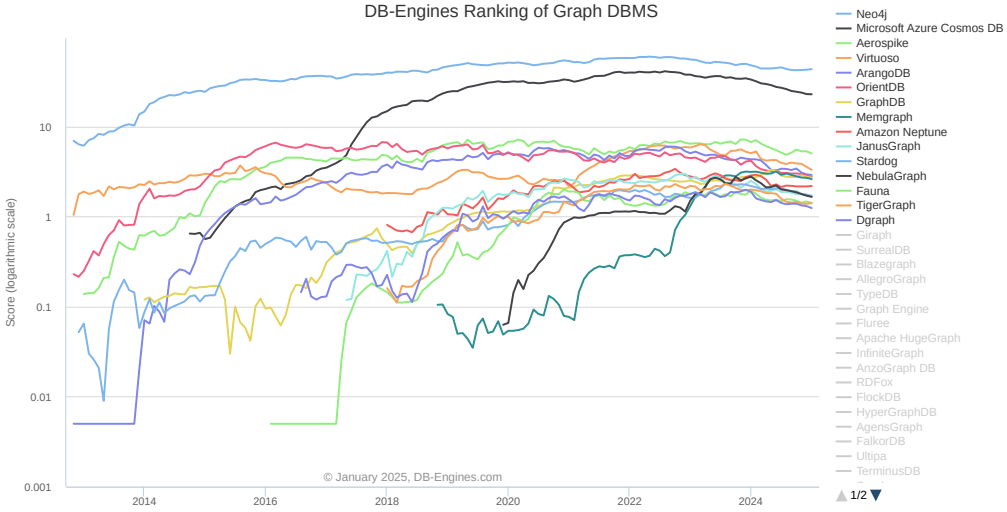
Graph Query Languages



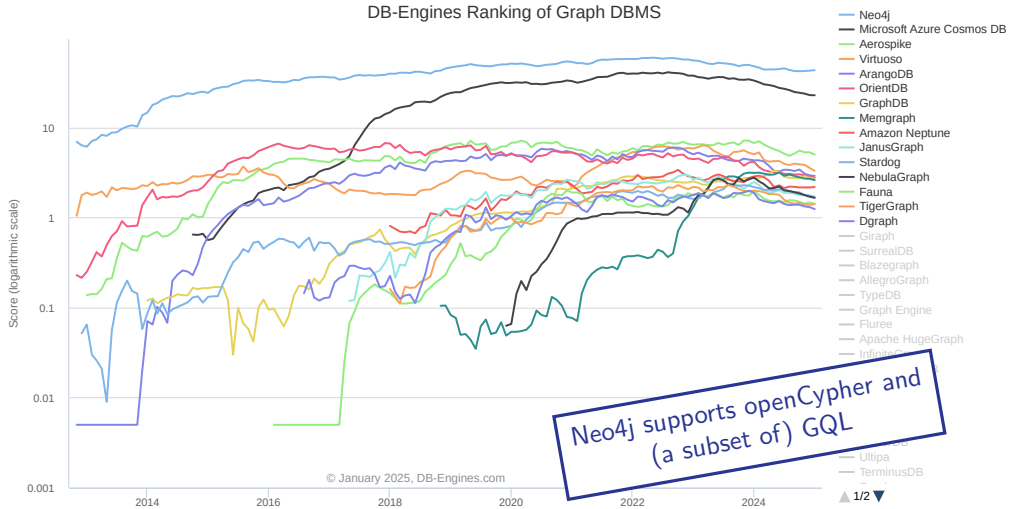
Source: Petra Selmer

[https://www.gqlstandards.org/existing-languages, 10/01/2025, Copyright © 2018-2024 JCC Consulting, Inc., licensed under the Apache License, Version 2.0]

Graph Database Engines



Graph Database Engines



GQL and openCypher

openCypher

[https://opencypher.org]

- Declarative language for property graphs
- open source specification
- aims to be human readable
- Implemented by various database, e.g.
  - Amazon Neptune, CAPS, Memgraph, Neo4j, Redisgraph, SAP HANA Graph, ...
- major influence for GQL
  - openCypher "evolves" towards GQL

GQL and openCypher

openCypher
[https://opencypher.org]

- Declarative language for property graphs
- open source specification
- aims to be human readable
- Implemented by various database, e.g.
  - Amazon Neptune, CAPS, Memgraph, Neo4j, Redisgraph, SAP HANA Graph, ...
- major influence for GQL
  - openCypher "evolves" towards GQL

GQL
[https://www.gqlstandards.org/]

- ISO Standard of a property graph query language
  - ISO/IEC 39075:2024
- First version published in April 2024
- 610 pages
- Syntax for specifying graph patterns is shared with the new SQL Standard SQL/PGQ for graph queries

GQL and openCypher

openCypher
[https://opencypher.org]

- Declarative language for property graphs
- open source specification
- aims to be human readable
- Implemented by various database, e.g.
  - Amazon Neptune, CAPS, Memgraph, Neo4j, Redisgraph, SAP HANA Graph, ...
- major influence for GQL
  - openCypher "evolves" towards GQL

GQL
[https://www.gqlstandards.org/]

- ISO Standard of a property graph query language
  - ISO/IEC 39075:2024
- First version published in April 2024
- 610 pages
- Syntax for specifying graph patterns is shared with the new SQL Standard SQL/PGQ for graph queries

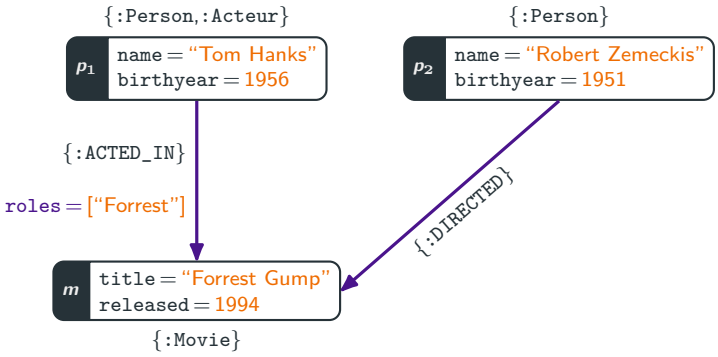
Data Model of Neo4j: Property Graphs

[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

consist of

Example



Data Model of Neo4j: Property Graphs

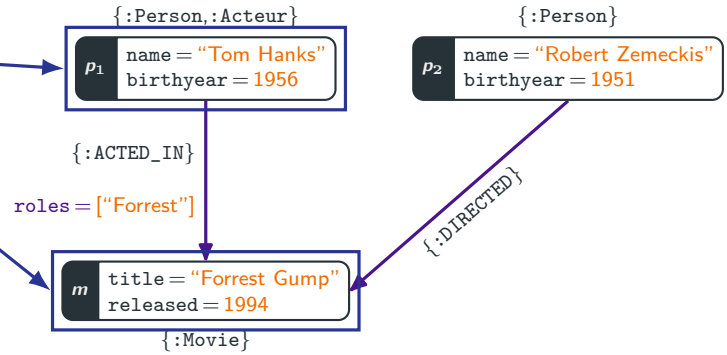
[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

consist of

- nodes,

Example



Data Model of Neo4j: Property Graphs

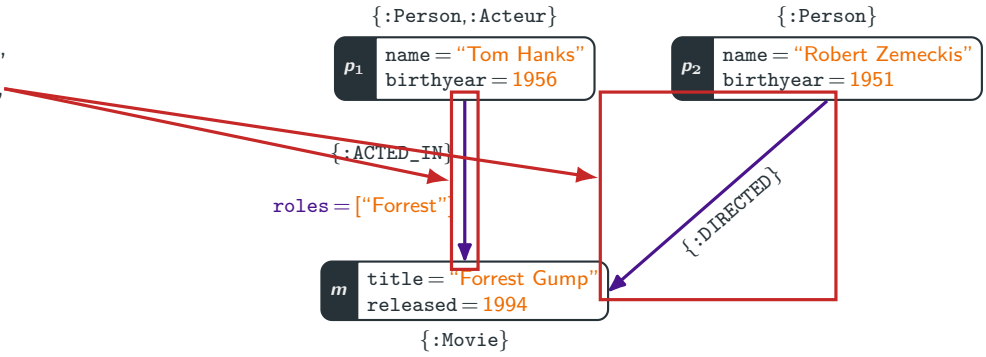
[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

consist of

- nodes,
- edges,

Example



Data Model of Neo4j: Property Graphs

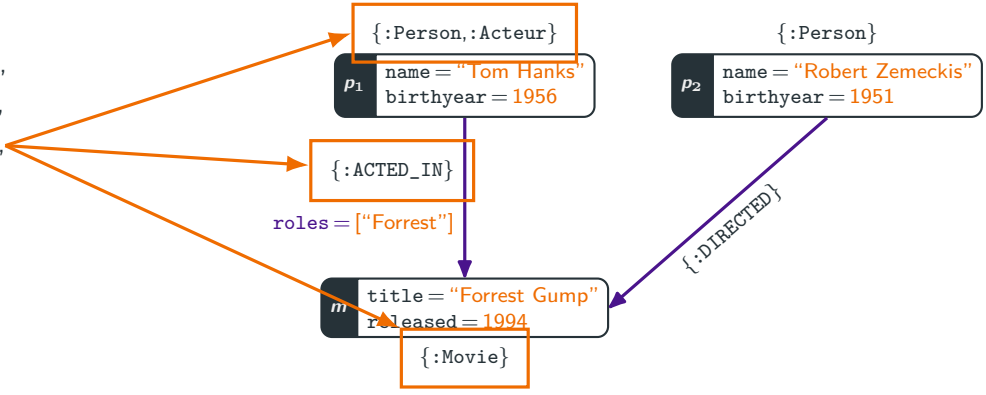
[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

consist of

- nodes,
- edges,
- labels,

Example



Data Model of Neo4j: Property Graphs

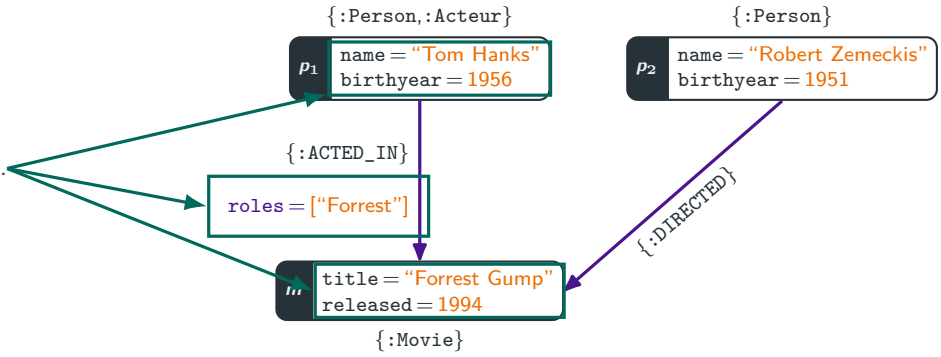
[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

consist of

- nodes,
- edges,
- labels,
- properties.

Example



Data Model of Neo4j: Property Graphs

[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

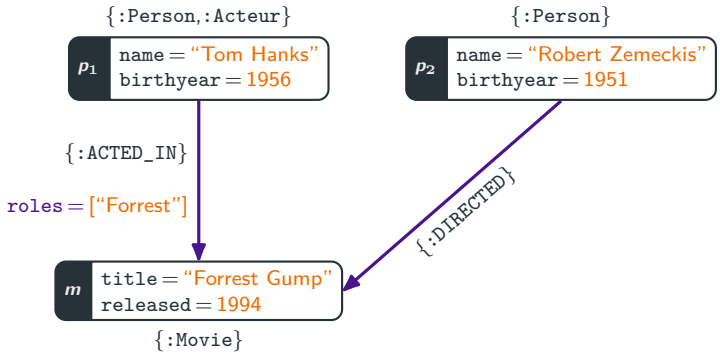
consist of

- nodes,
- ~~edges~~ relationships,
- labels,
- properties.

Neo4j Terminology

- Edges are called relationships

Example



Data Model of Neo4j: Property Graphs

[https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/]

Property Graphs

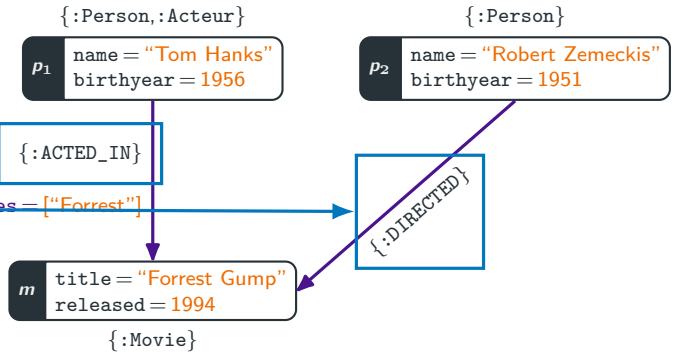
consist of

- nodes,
- ~~edges~~ relationships,
- labels,
- properties,
- types.

Neo4j Terminology

- Edges are called relationships
- Every relationship has *exactly one* label, which is its type

Example

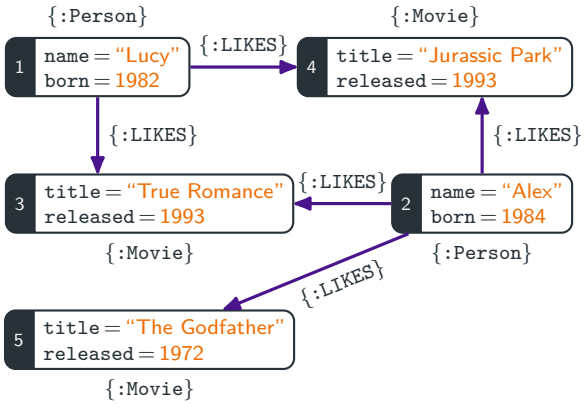


A Simple Example

Example

"The name of all persons and the release year of movies they like"

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
RETURN p.name, m.released
```



A Simple Example

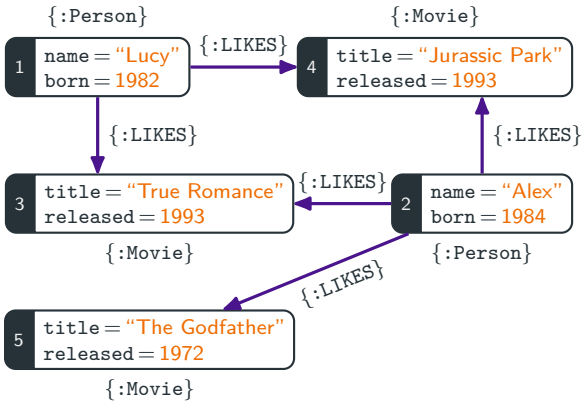
Example

"The name of all persons and the release year of movies they like"

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
RETURN p.name, m.released
```

Ingredients

- A pattern consisting of...
  - ...vertex patterns `(p:Person)`, `(m:Movie)`
  - ...an edge pattern `-[:LIKES]->`
- A `RETURN` clause



Pattern Syntax – Vertex Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Vertex Pattern

Pattern	Description
`()`	unidentified/anonymous vertex
`(matrix)`	vertex identified by/bound to variable `matrix`
`(:Movie)`	unidentified vertex with label Movie
`(:Movie|(Series & !Cancelled))`	vertex with complex label expression
`(matrix:Movie {title: "The Matrix"})`	property title has value "The Matrix"
`(matrix:Movie {title: "The Matrix", released: 1997})`	...and property released equals the integer 1997
`(matrix:Movie WHERE matrix.released >= 1997)`	more verbose syntax

In a Neo4j database each node can have an arbitrary number of labels

Pattern Syntax – Relationship Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Relationship (Edge) Pattern

Pattern	Description
`--`	unidentified edge, matches edges in either direction
`-->`	unidentified edge, matches in forward direction
`<--`	unidentified edge, matches in reverse direction
`-[:LIKES]->`	unidentified edge with type LIKES
`-[role]->`	forward edge bound to variable role
`-[role:ACTED_IN]->`	forward edge bound to variable role with type `ACTED_IN`
`-[role:ACTED_IN WHERE role.name = "Neo"]->`	...and property name has value "Neo"

Pattern Syntax – Relationship Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Relationship (Edge) Pattern

Pattern	Description
`--`	unidentified edge, matches edges in either direction
`-->`	unidentified edge, matches in forward direction
`<--`	unidentified edge, matches in reverse direction
`-[:LIKES]->`	unidentified edge with type LIKES
`-[role]->`	forward edge bound to variable role
`-[role:ACTED_IN]->`	forward edge bound to variable role with type `ACTED_IN`
`-[role:ACTED_IN WHERE role.name = "Neo"]->`	…and property name has value "Neo"

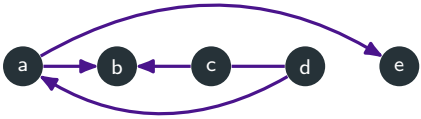
In a Neo4j database each relationship has exactly one type

Pattern Syntax – Path Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Fixed-length Path Patterns

- String of alternating vertex and edge pattern
- Starting and ending with a vertex pattern
- `(a)-->(b)<--(c)--(d)-->(a)-->(e)`

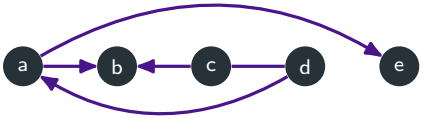


Pattern Syntax – Path Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Fixed-length Path Patterns

- String of alternating vertex and edge pattern
- Starting and ending with a vertex pattern
- `(a)-->(b)<--(c)--(d)-->(a)-->(e)`



Example

```
(p:Actor WHERE p.name = "Keanu Reeves")
  -[role:ACTED_IN WHERE role.name = "Neo"]->
(m:Movie WHERE m.title = "The Matrix")
```

Pattern Syntax – Variable-length Path Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns

- Path of varying/unknown length can be matched by adding a quantifier $\{n, m\}$
- n is a lower, and m a upper bound for the number of repetitions

Pattern Syntax – Variable-length Path Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns

- Path of varying/unknown length can be matched by adding a quantifier $\{n, m\}$
- n is a lower, and m a upper bound for the number of repetitions
- `(a)-->(b)(()<--()--()-->()){1,2}()-->(e)` is equivalent to the "union of"
  - `(a)-->(b)<--()--()-->()-->(e)` and
  - `(a)-->(b)<--()--()-->()<--()--()-->()-->(e)`

Pattern Syntax – Variable-length Path Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns

- Path of varying/unknown length can be matched by adding a quantifier $\{n, m\}$
- n is a lower, and m a upper bound for the number of repetitions
- `(a)-->(b)(()<--()--()-->()){1,2}()-->(e)` is equivalent to the "union of"
  - `(a)-->(b)<--()--()-->()-->(e)` and
  - `(a)-->(b)<--()--()-->()<--()--()-->()-->(e)`
- Both upper and lower bound are optional:
  - At most two repetitions: `(p1)((:Post)-[:REPLY_TO]->(:Post)){,2}(p2)`
  - At least three repetitions: `(p1)((:Post)-[:REPLY_TO]->(:Post)){3,}(p2)`

Pattern Syntax – Variable-length Path Patterns Cont'd

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns – Shorthands

- Shorthands for common cases:
  - Zero or more repetitions (Kleene star): `(p1)((:Post)-[:REPLY_TO]->(:Post))*(p2)`
  - At least one repetition: `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`

Pattern Syntax – Variable-length Path Patterns Cont'd

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns – Shorthands

- Shorthands for common cases:
  - Zero or more repetitions (Kleene star): `(p1)((:Post)-[:REPLY_TO]->(:Post))*(p2)`
  - At least one repetition: `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`
- Shorthand for repeating a single edge pattern:
  - `(p1:Post)-[:REPLY_TO]->*(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())*(p2:Post)`

Pattern Syntax – Variable-length Path Patterns Cont'd

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns – Shorthands

- Shorthands for common cases:
  - Zero or more repetitions (Kleene star): `(p1)((:Post)-[:REPLY_TO]->(:Post))*(p2)`
  - At least one repetition: `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`
- Shorthand for repeating a single edge pattern:
  - `(p1:Post)-[:REPLY_TO]->*(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())*(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->+(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())+(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->{2,4}(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->()){2,4}(p2:Post)`

Pattern Syntax – Variable-length Path Patterns Cont'd

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns – Shorthands

- Shorthands for common cases:
  - Zero or more repetitions (Kleene star): `(p1)((:Post)-[:REPLY_TO]->(:Post))*(p2)`
  - At least one repetition: `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`
- Shorthand for repeating a single edge pattern:
  - `(p1:Post)-[:REPLY_TO]->*(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())*(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->+(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())+(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->{2,4}(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->()){2,4}(p2:Post)`

Question?

Are the following patterns equivalent?

- `(p1:Post)-[:REPLY_TO]->+(p2:Post)`
- `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`

Pattern Syntax – Variable-length Path Patterns Cont'd

[https://neo4j.com/docs/cypher-manual/current/patterns/variable-length-patterns/]

Quantified Path Patterns – Shorthands

- Shorthands for common cases:
  - Zero or more repetitions (Kleene star): `(p1)((:Post)-[:REPLY_TO]->(:Post))*(p2)`
  - At least one repetition: `(p1)((:Post)-[:REPLY_TO]->(:Post))+(p2)`
- Shorthand for repeating a single edge pattern:
  - `(p1:Post)-[:REPLY_TO]->*(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())*(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->+(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->())+(p2:Post)`
  - `(p1:Post)-[:REPLY_TO]->{2,4}(p2:Post)` instead of `(p1:Post)(()-[:REPLY_TO]->()){2,4}(p2:Post)`

Old Cypher Syntax

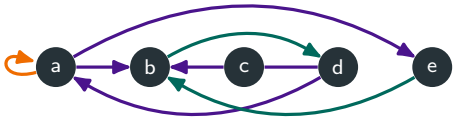
- Between two and four repetitions: `(p1:Post)-[:REPLY_TO*2..4]->(p2:Post)`
- equivalent to `(p1:Post)-[:REPLY_TO]->{2,4}(p2:Post)`
- Can only be used with single edge patterns
- Does not conform to GQL

Pattern Syntax – Graph Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Graph Patterns

- One or multiple path pattern
- separated by commata



Example

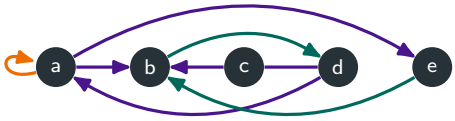
```
(a)-->(b)<--(c)--(d)-->(a)-->(e), (e)-->(b)-->(d), (a)-->(a)
```

Pattern Syntax – Graph Patterns

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Graph Patterns

- One or multiple path pattern
- separated by commata



Example

```
(a)-->(b)<--(c)--(d)-->(a)-->(e), (e)-->(b)-->(d), (a)-->(a)
```

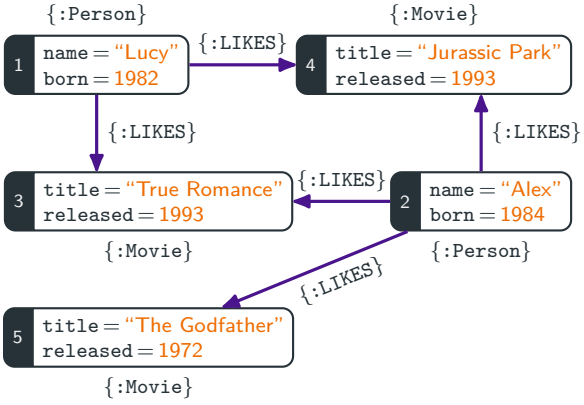
"Warning"

- Path patterns should have at least one shared variable
- Without shared variable the graph pattern is disconnected
  - Results in a cross-product of the results for connected sub patterns
  - Quadratic blow up in result size and computational complexity

Pattern Semantics

Semantics (of Neo4j)

- Homomorphism-like semantics
- but every edge can only be matched once
- GQL: Different edges matching semantics
- openCypher: trail semantics



Pattern Semantics

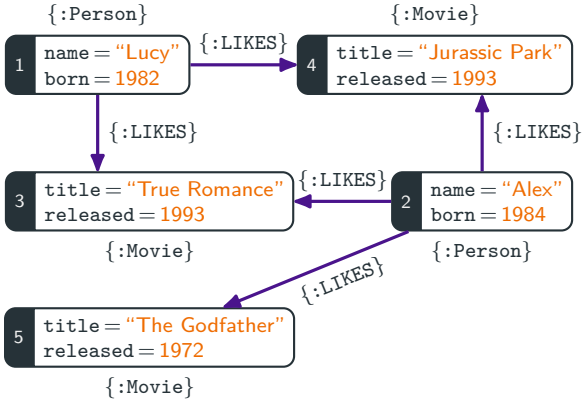
Semantics (of Neo4j)

- Homomorphism-like semantics
- but every edge can only be matched once
- GQL: Different edges matching semantics
- openCypher: trail semantics

Example

How many matches does the following graph pattern have?

```
(p1:Person)
  -[:LIKES]->
  (m:Movie {title: "The Godfather"}),
(p2:Person)-[:LIKES]->(m)
```



Pattern Syntax

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Which of these strings are patterns?

A: `(a,b:Movie)-[:SHOWN_IN]->(e),(f)`

B: `(a:Movie)-[:SHOWN_IN]->*()`

C: `(:Movie)-[:SHOWN_IN]->`

D: `()<--(a:Movie)`

Pattern Syntax

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Which of these strings are patterns?

A: ~~`(a,b:Movie)-[:SHOWN_IN]->(e),(f)`~~

B: `(a:Movie)-[:SHOWN_IN]->*()`

C: ~~`(:Movie)-[:SHOWN_IN]->`~~

D: `()<--(a:Movie)`

Pattern Syntax

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Which of these strings are patterns?

A: ~~`(a,b:Movie)-[:SHOWN_IN]->(e),(f)`~~

B: `(a:Movie)-[:SHOWN_IN]->*()`

C: ~~`(:Movie)-[:SHOWN_IN]->`~~

D: `()<--(a:Movie)`

Which patterns specify a loop?

A:
```
(a:Movie WHERE a.name = "Matrix")-->(a)
```

B:
```
(a:Movie WHERE a.name = "Matrix")-->(b:Movie WHERE b.name = "Matrix")
```

C:
```
(a:Movie WHERE a.name = "Matrix")-->(a:Movie WHERE a.name = "Matrix")
```

D:
```
(a:Movie WHERE name = "Matrix")-->({name: "Matrix"})
```

Pattern Syntax

[https://neo4j.com/docs/cypher-manual/current/patterns/]

Which of these strings are patterns?

A: ~~`(a,b:Movie)-[:SHOWN_IN]->(e),(f)`~~

B: `(a:Movie)-[:SHOWN_IN]->*()`

C: ~~`(:Movie)-[:SHOWN_IN]->`~~

D: `()<--(a:Movie)`

Which patterns specify a loop?

A: `(a:Movie WHERE a.name = "Matrix")-->(a)`

B: `(a:Movie WHERE a.name = "Matrix")-->(b:Movie WHERE b.name = "Matrix")`

C: `(a:Movie WHERE a.name = "Matrix")-->(a:Movie WHERE a.name = "Matrix")`

D: `(a:Movie WHERE name = "Matrix")-->({name: "Matrix"})`

Matching, Filtering, Result Definition

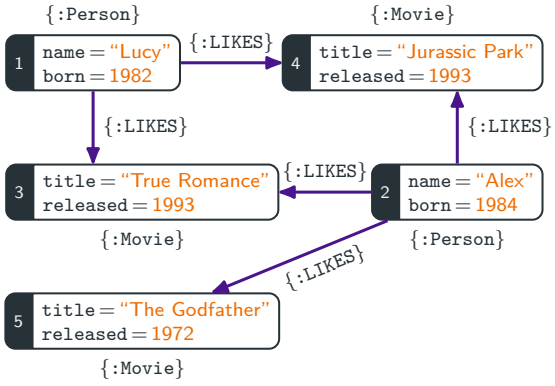
Matching

MATCH clause

- Primary way of querying Neo4j
- Takes a subgraph pattern
- and binds variables to matches

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
RETURN p.name, m.released
```



Matching

MATCH clause

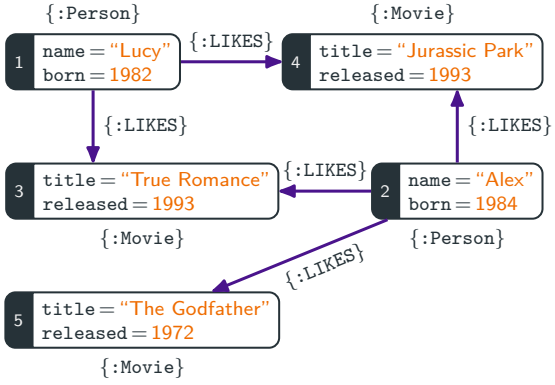
- Primary way of querying Neo4j
- Takes a subgraph pattern
- and binds variables to matches

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
RETURN p.name, m.released
```

Question

How many answers does the query return?



Matching

MATCH clause

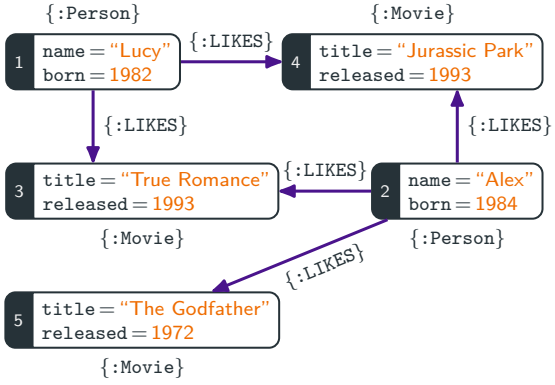
- Primary way of querying Neo4j
- Takes a subgraph pattern
- and binds variables to matches

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
RETURN p.name, m.released
```

Result

p.name	m.released
Lucy	1993
Lucy	1993
Alex	1993
Alex	1993
Alex	1972



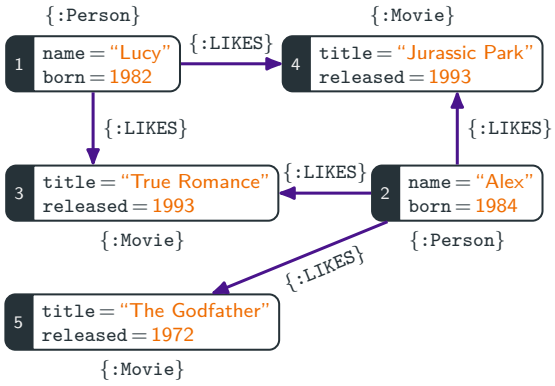
Matching

Multiple MATCH clauses

- A query can have multiple `MATCH` clauses
- Variable bindings are "passed" to the next `MATCH`

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
MATCH (p:Person)-[:LIKES]->(o:Movie)
WHERE m.released = o.released
RETURN m.title, o.title
```



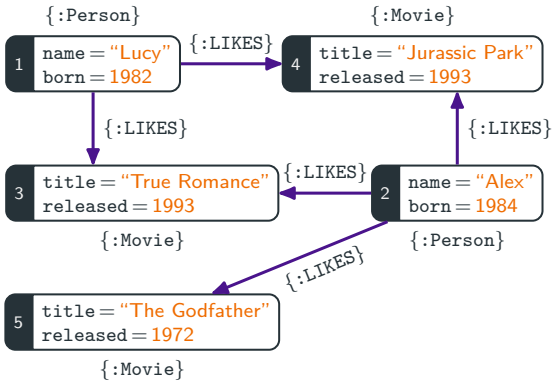
Matching

Multiple MATCH clauses

- A query can have multiple `MATCH` clauses
- Variable bindings are "passed" to the next `MATCH`

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
MATCH (p:Person)-[:LIKES]->(o:Movie)
WHERE m.released = o.released
RETURN m.title, o.title
```



Question

How many answers does the query return?

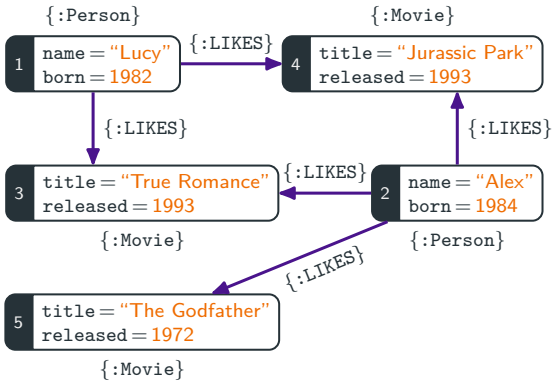
Matching

Multiple MATCH clauses

- A query can have multiple `MATCH` clauses
- Variable bindings are "passed" to the next `MATCH`

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
MATCH (p:Person)-[:LIKES]->(o:Movie)
WHERE m.released = o.released
RETURN m.title, o.title
```



Question

How many answers does the query return?

9 answers

Matching

Multiple MATCH clauses

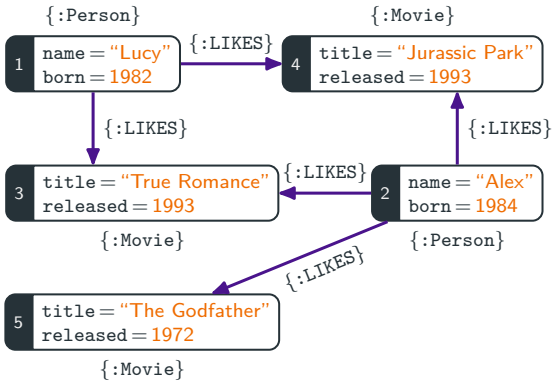
- A query can have multiple `MATCH` clauses
- Variable bindings are "passed" to the next `MATCH`

Example

```
MATCH (p:Person)-[:LIKES]->(m:Movie)
MATCH (p:Person)-[:LIKES]->(o:Movie)
WHERE m.released = o.released
RETURN m.title, o.title
```

Question

How many answers does the query return?
9 answers



- Every edge can only be matched once per `MATCH` clause

Optional Match

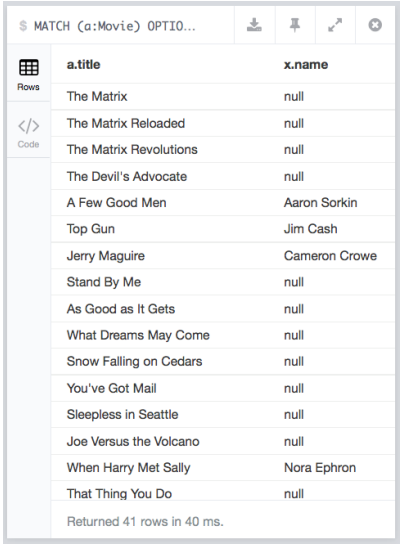
[https://neo4j.com/docs/cypher-manual/current/clauses/optional-match/]

Optional Match clause

- Matches patterns, just like `MATCH`
- Matches the complete pattern or not
- If no matches are found, `OPTIONAL MATCH` will use `null`s as bindings
- Like relational left outer join

Example

```
MATCH (a:Movie)
OPTIONAL MATCH (a)<-[:WROTE]-(x)
RETURN a.title, x.name
```



Filtering

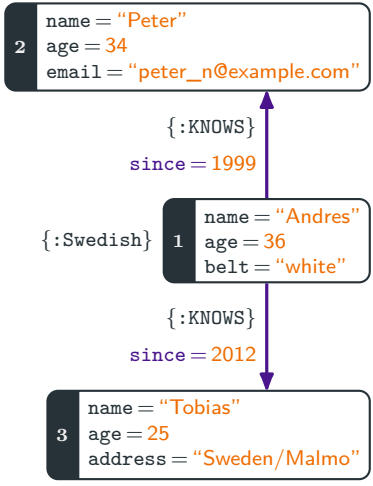
[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

WHERE clause

- After an (`OPTIONAL`) `MATCH`, it adds constraints to the (optional) match
- After a `WITH` clause, it just filters the result

Example

```
MATCH (n)
WHERE n.name = "Peter"
  OR (n.age < 30 AND n.name = "Tobias")
  OR NOT (n.name = "Tobias" OR n.name="Peter")
RETURN n
```



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

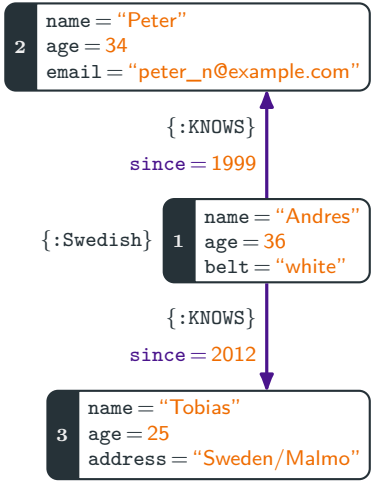
WHERE clause

- After an (`OPTIONAL`) `MATCH`, it adds constraints to the (optional) match
- After a `WITH` clause, it just filters the result

Example

```
MATCH (n)
WHERE n.name = "Peter"
  OR (n.age < 30 AND n.name = "Tobias")
  OR NOT (n.name = "Tobias" OR n.name="Peter")
RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}



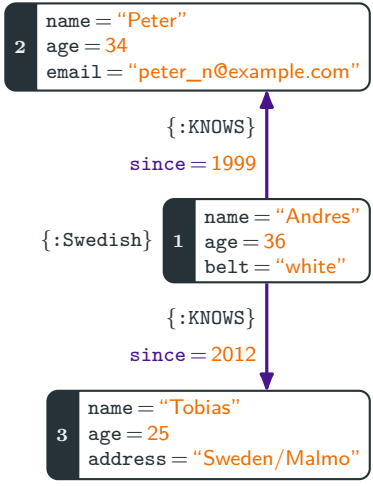
Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on node label

```
MATCH (n) WHERE n:Swedish RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on node label

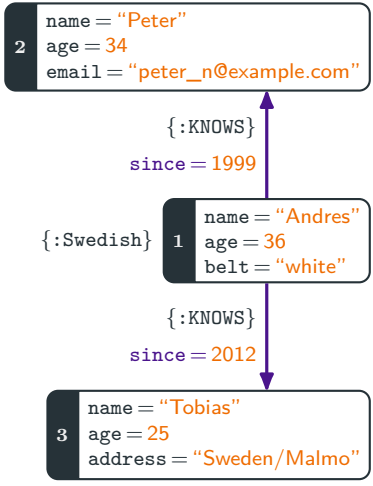
```
MATCH (n) WHERE n:Swedish RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- Filter on a node property

```
MATCH (n) WHERE n.age < 30 RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on node label

```
MATCH (n) WHERE n:Swedish RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- Filter on a node property

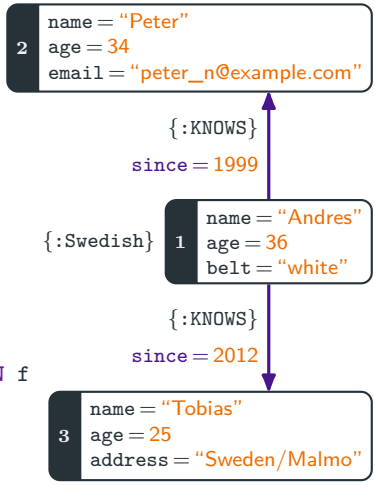
```
MATCH (n) WHERE n.age < 30 RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

- Filter on a relationships

```
MATCH (n)-[k]->(f) WHERE k:KNOWS AND k.since < 2000 RETURN f
```

f
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}



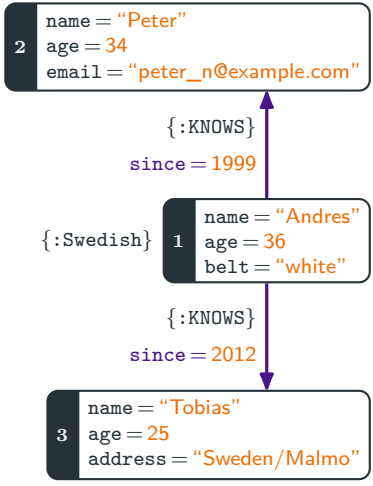
Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on lists

```
MATCH (n) WHERE n.name IN ["Peter", "Tobias"] RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

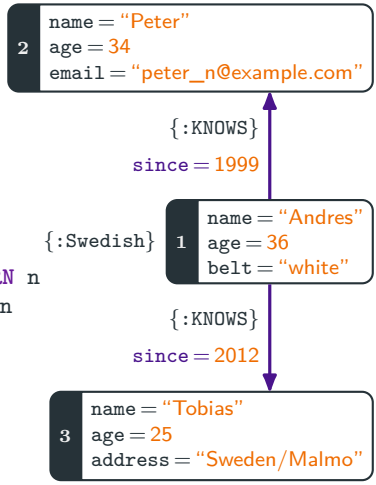
- Filter on lists

```
MATCH (n) WHERE n.name IN ["Peter", "Tobias"] RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

- Filter on string ...
  - properties: `MATCH (n) WHERE n.name = 'Peter' RETURN n`
  - prefixes: `MATCH (n) WHERE n.name STARTS WITH 'Pet' RETURN n`
  - suffixes: `MATCH (n) WHERE n.name ENDS WITH 'ter' RETURN n`
  - infixes: `MATCH (n) WHERE n.name CONTAINS 'ete' RETURN n`
  - regex: `MATCH (n) WHERE n.name =~ 'P[et]+r?' RETURN n`

n
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

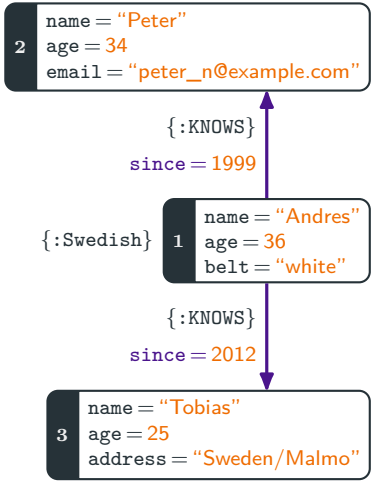


Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on property existence
  `MATCH (n) WHERE n.belt IS NOT NULL RETURN n`
  (default value for missing properties is `NULL`)

n
Node[0]{name:"Andres",age:36,belt:"white"}



Filtering

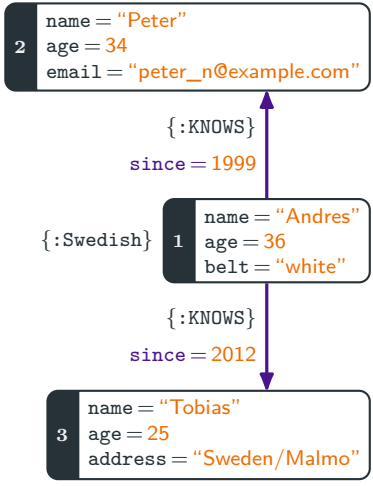
[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on property existence
  `MATCH (n) WHERE n.belt IS NOT NULL RETURN n`
  (default value for missing properties is `NULL`)

n
Node[0]{name:"Andres",age:36,belt:"white"}

- Filter on property absence/non-existence
  `MATCH (n) WHERE n.belt IS NULL RETURN n`

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}



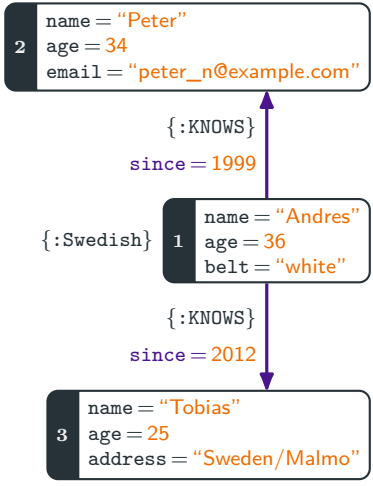
Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on patterns

```
MATCH (t { name: 'Tobias' }), (others)
WHERE others.age > 30 AND (tobias)<--(others)
RETURN others
```

others
Node[0]{name:"Andres",age:36,belt:"white"}



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on patterns

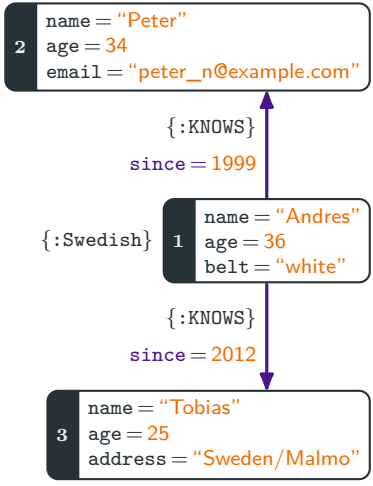
```
MATCH (t { name: 'Tobias' }), (others)
WHERE others.age > 30 AND (tobias)<--(others)
RETURN others
```

others
Node[0]{name:"Andres",age:36,belt:"white"}

- ...with negation

```
MATCH (persons), (p {name: 'Peter'})
WHERE NOT (persons)-->(p)
RETURN persons
```

persons
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}



Filtering

[https://neo4j.com/docs/cypher-manual/current/clauses/where/]

- Filter on patterns

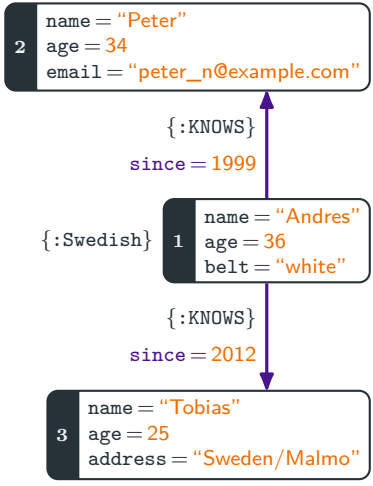
```
MATCH (t { name: 'Tobias' }), (others)
WHERE others.age > 30 AND (tobias)<--(others)
RETURN others
```

others
Node[0]{name:"Andres",age:36,belt:"white"}

- ...on existence

```
MATCH (person) WHERE EXISTS((person)-->())
RETURN person
```

person
Node[0]{name:"Andres",age:36,belt:"white"}



Projection

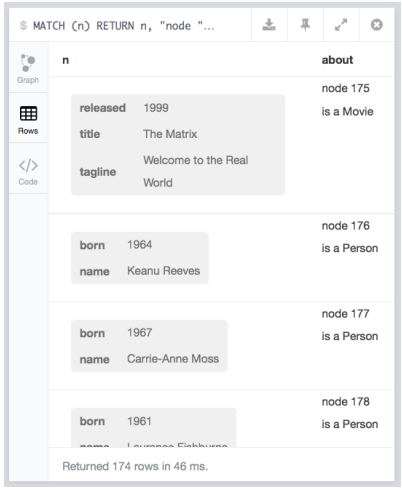
[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

RETURN clause

- Defines what to include in the query result set
- Comparable with relational projection
- Only once per query
- Allows to return nodes, edges, properties, or any expressions
- Column can be rename using `AS` <new name>

Example

```
MATCH (n)
RETURN n, "node " + id(n) + " is " +
  CASE WHEN n.title IS NOT NULL THEN "a Movie"
       WHEN EXISTS(n.name) THEN "a Person"
       ELSE "something unknown"
  END AS about
```



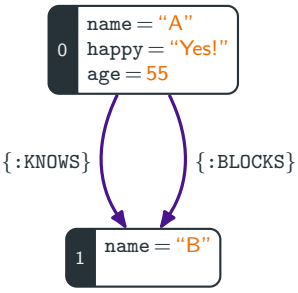
Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Return nodes

```
MATCH (n { name: "B" }) RETURN n
```

n
Node[1]{name:"B"}



Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Return nodes

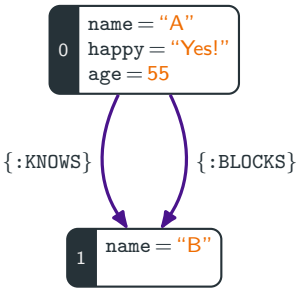
```
MATCH (n { name: "B" }) RETURN n
```

n
Node[1]{name:"B"}

- Return relationships

```
MATCH (n { name: "A" })-[r:KNOWS]->(c) RETURN r
```

r
:KNOWS[0]{}



Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Return nodes

```
MATCH (n { name: "B" }) RETURN n
```

n
Node[1]{name:"B"}

- Return relationships

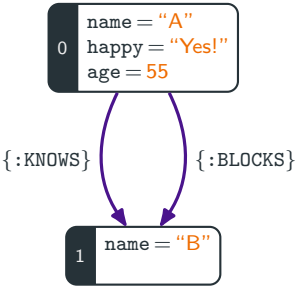
```
MATCH (n { name: "A" })-[r:KNOWS]->(c) RETURN r
```

r
:KNOWS[0]{}

- Return properties

```
MATCH (n { name: "A" }) RETURN n.name
```

n.name
"A"



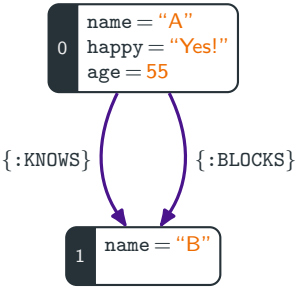
Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Column alias

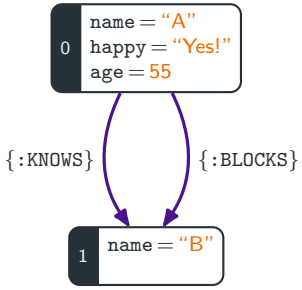
```
MATCH (a { name: "A" })
RETURN a.age AS SomethingTotallyDifferent
```

SomethingTotallyDifferent
55



Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]



- Column alias

```
MATCH (a { name: "A" })
RETURN a.age AS SomethingTotallyDifferent
```

SomethingTotallyDifferent
55

- Return all bounded elements

```
MATCH p=(a { name: "A" })-[r]->(b) RETURN *
```

a	b	p	r
Node[0]{name:"A",happy:"Yes!",age:55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age: 55},:BLOCKS[1]{},Node[1]{name:"B"}]	:BLOCKS[1]{}
Node[0]{name:"A",happy:"Yes!",age: 55}	Node[1]{name:"B"}	[Node[0]{name:"A",happy:"Yes!",age: 55},:KNOWS[0]{},Node[1]{name:"B"}]	:KNOWS[0]{}

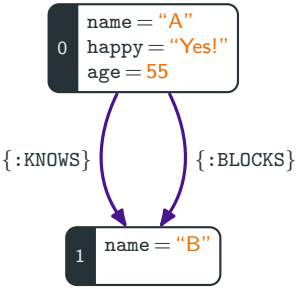
Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Returning optional properties

```
MATCH (n) RETURN n.age
```

n.age
55
null



Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]

- Returning optional properties

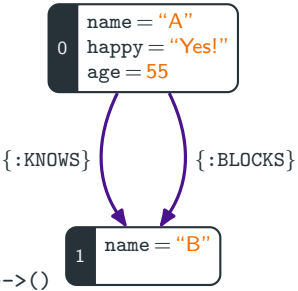
```
MATCH (n) RETURN n.age
```

n.age
55
null

- Other expressions

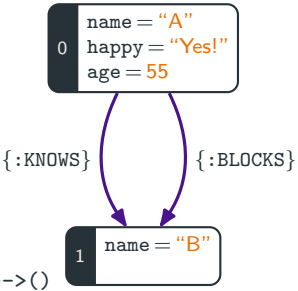
```
MATCH (a { name: "A" }) RETURN a.age > 30, "I'm a literal", (a)-->()
```

a.age > 30	"I'm a literal"	(a)-->()
true	"I'm a literal"	[[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{},Node[1]{name:"B"}], [Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{},Node[1]{name:"B"}]]



Projection

[https://neo4j.com/docs/cypher-manual/current/clauses/return/]



- Returning optional properties

```
MATCH (n) RETURN n.age
```

n.age
55
null

- Other expressions

```
MATCH (a { name: "A" }) RETURN a.age > 30, "I'm a literal", (a)-->()
```

a.age > 30	"I'm a literal"	(a)-->()
true	"I'm a literal"	[[Node[0]{name:"A",happy:"Yes!",age:55},:BLOCKS[1]{},Node[1]{name:"B"}], [Node[0]{name:"A",happy:"Yes!",age:55},:KNOWS[0]{},Node[1]{name:"B"}]]

- Unique results

```
MATCH (a { name: "A" })-->(b) RETURN DISTINCT b
```

b
Node[1]{name:"B"}

Result Modification – Sorting

[https://neo4j.com/docs/cypher-manual/current/clauses/order-by/]

ORDER BY clause

- Sub-clause following `RETURN` or `WITH`
- Specifies how the output should be sorted
- Can only sort on properties,
  not nodes or relationships
- `null` will come last in ascending order (`ASC`),
  and first in descending order (`DESC`)

Result Modification – Sorting

[https://neo4j.com/docs/cypher-manual/current/clauses/order-by/]

ORDER BY clause

- Sub-clause following `RETURN` or `WITH`
- Specifies how the output should be sorted
- Can only sort on properties,
  not nodes or relationships
- `null` will come last in ascending order (`ASC`),
  and first in descending order (`DESC`)

Example (Order by property)

```
MATCH (n) RETURN n ORDER BY n.name
```

n
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}
Node[2]{name:"C",age:32,length:185}

Result Modification – Sorting

[https://neo4j.com/docs/cypher-manual/current/clauses/order-by/]

ORDER BY clause

- Sub-clause following `RETURN` or `WITH`
- Specifies how the output should be sorted
- Can only sort on properties,
  not nodes or relationships
- `null` will come last in ascending order (`ASC`),
  and first in descending order (`DESC`)

Example (Order by multiple property)

```
MATCH (n) RETURN n ORDER BY n.age, n.name
```

n
Node[2]{name:"C",age:32,length:185}
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}

Example (Order by property)

```
MATCH (n) RETURN n ORDER BY n.name
```

n
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}
Node[2]{name:"C",age:32,length:185}

Result Modification – Sorting

[https://neo4j.com/docs/cypher-manual/current/clauses/order-by/]

ORDER BY clause

- Sub-clause following `RETURN` or `WITH`
- Specifies how the output should be sorted
- Can only sort on properties,
  not nodes or relationships
- `null` will come last in ascending order (`ASC`),
  and first in descending order (`DESC`)

Example (Order by property)

```
MATCH (n) RETURN n ORDER BY n.name
```

n
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}
Node[2]{name:"C",age:32,length:185}

Example (Order by multiple property)

```
MATCH (n) RETURN n ORDER BY n.age, n.name
```

n
Node[2]{name:"C",age:32,length:185}
Node[0]{name:"A",age:34,length:170}
Node[1]{name:"B",age:34}

Example (Order nodes in descending order)

```
MATCH (n) RETURN n ORDER BY n.name DESC
```

n
Node[2]{name:"C",age:32,length:185}
Node[1]{name:"B",age:34}
Node[0]{name:"A",age:34,length:170}

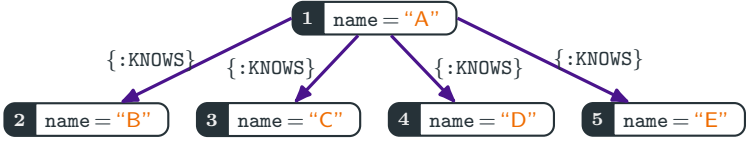
Result Modification – LIMIT clause

[https://neo4j.com/docs/cypher-manual/current/clauses/limit/]

LIMIT clause

- Constrains the number of rows in the output
- Accepts any expression that evaluates to a positive integer
- Expression cannot refer to nodes or relationships
- Return first from the top

```
MATCH (n) RETURN n ORDER BY n.name LIMIT 3
```



n
Node[0]{name:"A"}
Node[0]{name:"B"}
Node[0]{name:"C"}

Result Modification – LIMIT clause

[https://neo4j.com/docs/cypher-manual/current/clauses/limit/]

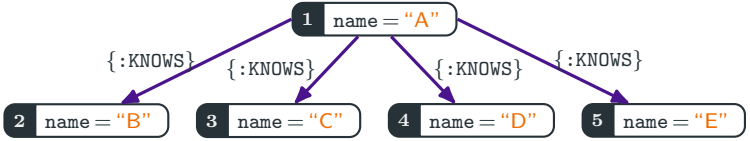
LIMIT clause

- Constrains the number of rows in the output
- Accepts any expression that evaluates to a positive integer
- Expression cannot refer to nodes or relationships
- Return first from the top

```
MATCH (n) RETURN n ORDER BY n.name LIMIT 3
```

- Return first from expression

```
MATCH (n) RETURN n ORDER BY n.name LIMIT toInt(3 * rand()) + 1
```



Result Modification – OFFSET clause

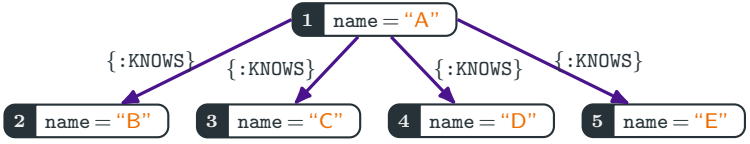
[https://neo4j.com/docs/cypher-manual/current/clauses/skip/]

OFFSET clause

- Defines from which row to start including the rows in the output
- Result set will get trimmed from the top
- Same rules as for LIMIT
- Skip first three

```
MATCH (n) RETURN n ORDER BY n.name OFFSET 3
```

- `SKIP` is an alias supported by Neo4j



n
Node[0]{name:"D"}
Node[0]{name:"E"}

Aggregation

Aggregation

[https://neo4j.com/docs/cypher-manual/current/functions/aggregating/]

Group by/Aggregation

- Implicit group by (that is, there is no keyword!)
  - Expressions without an aggregation function will be the group keys
  - Expressions with an aggregation function will produce aggregates
- `DISTINCT` within the aggregation function removes duplicates in a group before the aggregation
- `ALL` aggregates duplicates (default)

Example

```
MATCH (p:Person {name: "Ann"})-->(friend:Person)-->(fof:Person)
RETURN p.name, count(DISTINCT fof), count(ALL fof), count(fof)
```

Aggregation

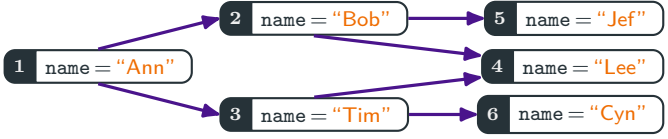
[https://neo4j.com/docs/cypher-manual/current/functions/aggregating/]

Group by/Aggregation

- Implicit group by (that is, there is no keyword!)
  - Expressions without an aggregation function will be the group keys
  - Expressions with an aggregation function will produce aggregates
- `DISTINCT` within the aggregation function removes duplicates in a group before the aggregation
- `ALL` aggregates duplicates (default)

Example

```
MATCH (p:Person {name: "Ann"})-->(friend:Person)-->(fof:Person)
RETURN p.name, count(DISTINCT fof), count(ALL fof), count(fof)
```



Aggregation

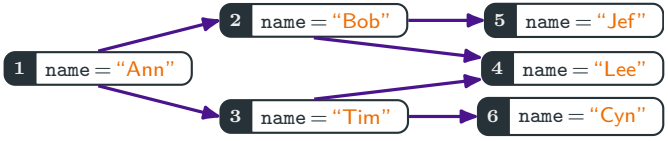
[https://neo4j.com/docs/cypher-manual/current/functions/aggregating/]

Group by/Aggregation

- Implicit group by (that is, there is no keyword!)
  - Expressions without an aggregation function will be the group keys
  - Expressions with an aggregation function will produce aggregates
- `DISTINCT` within the aggregation function removes duplicates in a group before the aggregation
- `ALL` aggregates duplicates (default)

Example

```
MATCH (p:Person {name: "Ann"})-->(friend:Person)-->(fof:Person)
RETURN p.name, count(DISTINCT fof), count(ALL fof), count(fof)
```



Result

p.name	DISTINCT count	ALL count	count
Ann	3	4	4

Aggregation

[https://neo4j.com/docs/cypher-manual/current/functions/aggregating/]

Some Common Aggregation Functions

Function	Description
`avg()`	Returns the average of a numeric column.
`collect()`	Returns a list containing all collected values.
`count()`	Returns the number of rows.
`max()`	Returns the highest value in a numeric column.
`min()`	Returns the lowest value in a numeric column.
`percentileCont()`	Returns the percentile of a given value over a group using linear interpolation.
`percentileDisc()`	Returns the nearest value to a given percentile over a group using a rounding method.
`stDev()`	Returns the standard deviation for a given value over a group for a sample of a population.
`stDevP()`	Returns the standard deviation for a given value over a group for an entire population.
`sum()`	Returns the sum of a numeric column.

Composition

Query Composition

[https://neo4j.com/docs/cypher-manual/current/clauses/with/]

WITH clause

- Like `RETURN` followed by a process pipe
- Chains subqueries together, piping the results from one to be used as starting points in the next
- Like `RETURN`, `WITH` defines – including aggregation – the output before it is passed on

Query Composition

[https://neo4j.com/docs/cypher-manual/current/clauses/with/]

WITH clause

- Like `RETURN` followed by a process pipe
- Chains subqueries together, piping the results from one to be used as starting points in the next
- Like `RETURN`, `WITH` defines – including aggregation – the output before it is passed on

Example (Friends of five best friends)

Limit search space based on order of properties or aggregates

```
MATCH (p)-[f:FRIENDS]->(p2)
WITH f, p2 ORDER BY f.rating DESC LIMIT 5
MATCH (p2)-[:FRIENDS]->(p3)
RETURN DISTINCT p3
```

Query Composition

[https://neo4j.com/docs/cypher-manual/current/clauses/with/]

WITH clause

- Like `RETURN` followed by a process pipe
- Chains subqueries together, piping the results from one to be used as starting points in the next
- Like `RETURN`, `WITH` defines – including aggregation – the output before it is passed on

Example (Average age of the youngest player in each team)

Aggregation of aggregates

```
MATCH (p:Player)-[:PLAYS]->(t:Team)
WITH t, min(p.age) AS age
RETURN avg(age)
```

Query Composition

[https://neo4j.com/docs/cypher-manual/current/clauses/with/]

WITH clause

- Like `RETURN` followed by a process pipe
- Chains subqueries together, piping the results from one to be used as starting points in the next
- Like `RETURN`, `WITH` defines – including aggregation – the output before it is passed on

Example (Teams whose players are on average younger than 25)

Filter on aggregates

```
MATCH (p:Player)-[:PLAYS]->(t:Team)
WITH t, avg(p.age) AS age WHERE age < 25
RETURN t
```

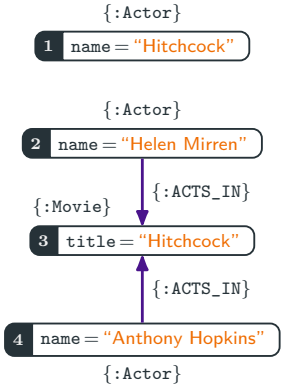
Query Composition – Unions

[https://neo4j.com/docs/cypher-manual/current/clauses/union/]

UNION DISTINCT or UNION

Combines two query results and removes duplicates

```
MATCH (n:Actor)
RETURN n.name AS name
UNION DISTINCT
MATCH (n:Movie)
RETURN n.title AS name
```



Query Composition – Unions

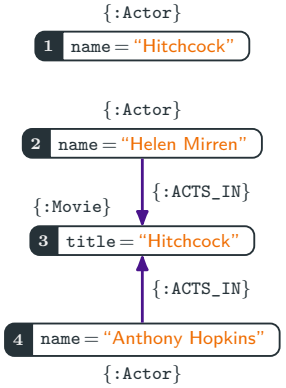
[https://neo4j.com/docs/cypher-manual/current/clauses/union/]

UNION DISTINCT or UNION

Combines two query results and removes duplicates

```
MATCH (n:Actor)
RETURN n.name AS name
UNION DISTINCT
MATCH (n:Movie)
RETURN n.title AS name
```

name
"Anthony Hopkins"
"Helen Mirren"
"Hitchcock"



Query Composition – Unions

[https://neo4j.com/docs/cypher-manual/current/clauses/union/]

UNION DISTINCT or UNION

Combines two query results and removes duplicates

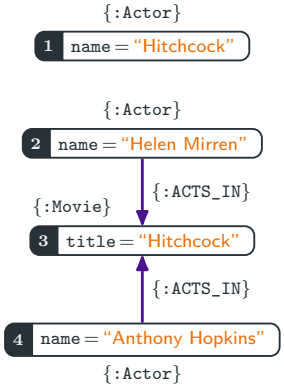
```
MATCH (n:Actor)
RETURN n.name AS name
UNION DISTINCT
MATCH (n:Movie)
RETURN n.title AS name
```

name
"Anthony Hopkins"
"Helen Mirren"
"Hitchcock"

UNION ALL

Combines two query results and retains duplicates

```
MATCH (n:Actor)
RETURN n.name AS name
UNION ALL
MATCH (n:Movie)
RETURN n.title AS name
```



Query Composition – Unions

[https://neo4j.com/docs/cypher-manual/current/clauses/union/]

UNION DISTINCT or UNION

Combines two query results and removes duplicates

```
MATCH (n:Actor)
RETURN n.name AS name
UNION DISTINCT
MATCH (n:Movie)
RETURN n.title AS name
```

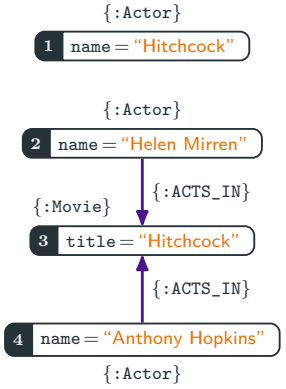
name
“Anthony Hopkins”
“Helen Mirren”
“Hitchcock”

UNION ALL

Combines two query results and retains duplicates

```
MATCH (n:Actor)
RETURN n.name AS name
UNION ALL
MATCH (n:Movie)
RETURN n.title AS name
```

name
“Anthony Hopkins”
“Helen Mirren”
“Hitchcock”
“Hitchcock”



Lists

Lists

[https://neo4j.com/docs/cypher-manual/current/values-and-types/lists/]

- Lists can be stored as properties (if all elements have the same type)

Example

Lists

[https://neo4j.com/docs/cypher-manual/current/values-and-types/lists/]

- Lists can be stored as properties (if all elements have the same type)

Example

- All movies available in English and French

```
MATCH (m:Movie)
WHERE "en" IN m.lang AND "fr" IN m.lang
RETURN m.title
```

m.title
"True Romance"

Lists

[https://neo4j.com/docs/cypher-manual/current/values-and-types/lists/]

- Lists can be stored as properties (if all elements have the same type)

Example

- All movies available in English and French

```
MATCH (m:Movie)
WHERE "en" IN m.lang AND "fr" IN m.lang
RETURN m.title
```

m.title
"True Romance"

- All movies and the languages they are available in

```
MATCH (m:Movie)
UNWIND m.lang AS language
RETURN m.title, language
```

m.title	language
"True Romance"	"en"
"True Romance"	"fr"
"True Romance"	"de"

Filtering on Lists

[https://neo4j.com/docs/cypher-manual/current/functions/predicate/]

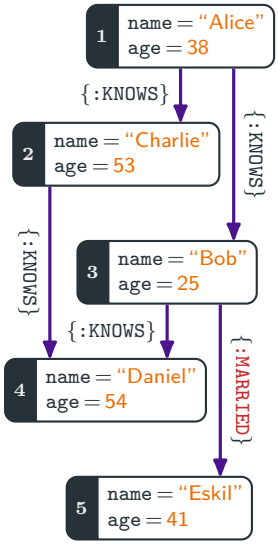
Predicate Functions for Lists

- Tests whether a predicate holds for all elements of this list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE all(b IN bs WHERE b.age > a.age)
RETURN a.name
```

Result:

a.name
Charlie
Bob



Filtering on Lists

[https://neo4j.com/docs/cypher-manual/current/functions/predicate/]

Predicate Functions for Lists

- Tests whether a predicate holds for all elements of this list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE all(b IN bs WHERE b.age > a.age)
RETURN a.name
```

Result:

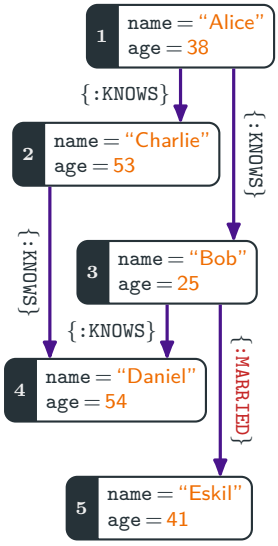
a.name
Charlie
Bob

- Tests whether a predicate holds for at least one element in the list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE any(b IN bs WHERE b.age < a.age)
RETURN a.name
```

Result:

a.name
Alice



Filtering on Lists Cont'd

[https://neo4j.com/docs/cypher-manual/current/functions/predicate/]

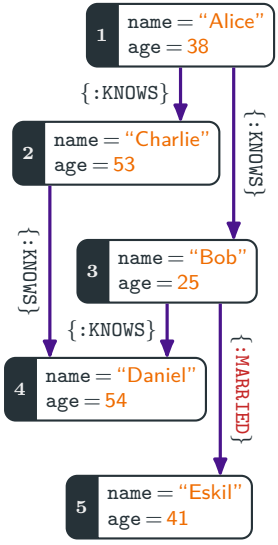
Predicate Functions for Lists

- Tests whether a predicate holds for exactly one element in the list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE single(b IN bs WHERE b.age < a.age)
RETURN a.name
```

Result:

a.name
Alice



Filtering on Lists Cont'd

[https://neo4j.com/docs/cypher-manual/current/functions/predicate/]

Predicate Functions for Lists

- Tests whether a predicate holds for exactly one element in the list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE single(b IN bs WHERE b.age < a.age)
RETURN a.name
```

Result:

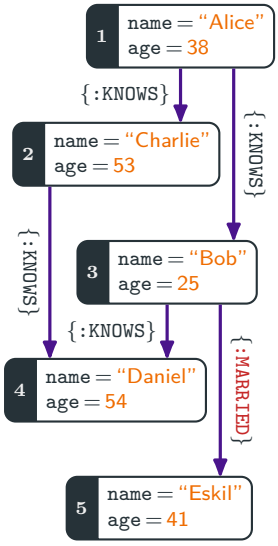
a.name
Alice

- Tests whether a predicate holds for no element in the list

```
MATCH (a)-[:KNOWS]->(b) WITH a, collect(b) AS bs
WHERE none(b IN bs WHERE b.age < a.age)
RETURN a.name
```

Result:

a.name
Charlie
Bob



Paths

Path Variables

[https://neo4j.com/docs/cypher-manual/current/clauses/match/#find-paths]

Path Variables

- Matched paths can be assigned to variables for further processing
- `UNWIND` can be used to access nodes and edges on the path
- Paths can be returned

Example

```
MATCH p = (a:Author)-[:WROTE]->(:Post)((:Post)-[:REPLY_TO]->(:Post))+
UNWIND nodes(p) AS post
RETURN p, post.date
```

Path Variables

[https://neo4j.com/docs/cypher-manual/current/clauses/match/#find-paths]

Path Variables

- Matched paths can be assigned to variables for further processing
- `UNWIND` can be used to access nodes and edges on the path
- Paths can be returned

Example

```
MATCH p = (a:Author)-[:WROTE]->(:Post)((:Post)-[:REPLY_TO]->(:Post))+
UNWIND nodes(p) AS post
RETURN p, post.date
```

- When working with path pattern care should to be taken: they easily match a large number of paths (exponential blow-up)

Shortest Paths

[https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/]

Shortest Paths

- Path between two nodes with minimum number of edges

Example

- Match all shortest paths

```
MATCH p = ALL SHORTEST
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```

Shortest Paths

[https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/]

Shortest Paths

- Path between two nodes with minimum number of edges

Example

- Match all shortest paths

```
MATCH p = ALL SHORTEST
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```

- Are the queries above and below equivalent?

```
MATCH p = ALL SHORTEST (start:City)-[:TRAIN]->+(dest:City)
WHERE start.name = "Lyon" AND dest.name = "Berlin"
RETURN p
```

Shortest Paths

[https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/]

Shortest Paths

- Path between two nodes with minimum number of edges

Example

- Match all shortest paths

```
MATCH p = ALL SHORTEST
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```

- Are the queries above and below equivalent?

```
MATCH p = ALL SHORTEST (start:City)-[:TRAIN]->+(dest:City)
WHERE start.name = "Lyon" AND dest.name = "Berlin"
RETURN p
```

- No, for the second query all shortest paths between any two cities are computed and then filtered

Shortest Paths

[https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/]

Shortest Paths – Variants

Example

- Match the top k shortest paths

```
MATCH p = SHORTEST 5
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```

Shortest Paths

[https://neo4j.com/docs/cypher-manual/current/patterns/shortest-paths/]

Shortest Paths – Variants

Example

- Match the top k shortest paths

```
MATCH p = SHORTEST 5
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```

- Match an arbitrary shortest path (same as `SHORTEST 1`)

```
MATCH p = ANY
(start:City {name: "Lyon"})-[:TRAIN]->+(dest:City {name: "Berlin"})
RETURN p
```