Modelado y Procesamiento de Grandes Volumenes de Datos

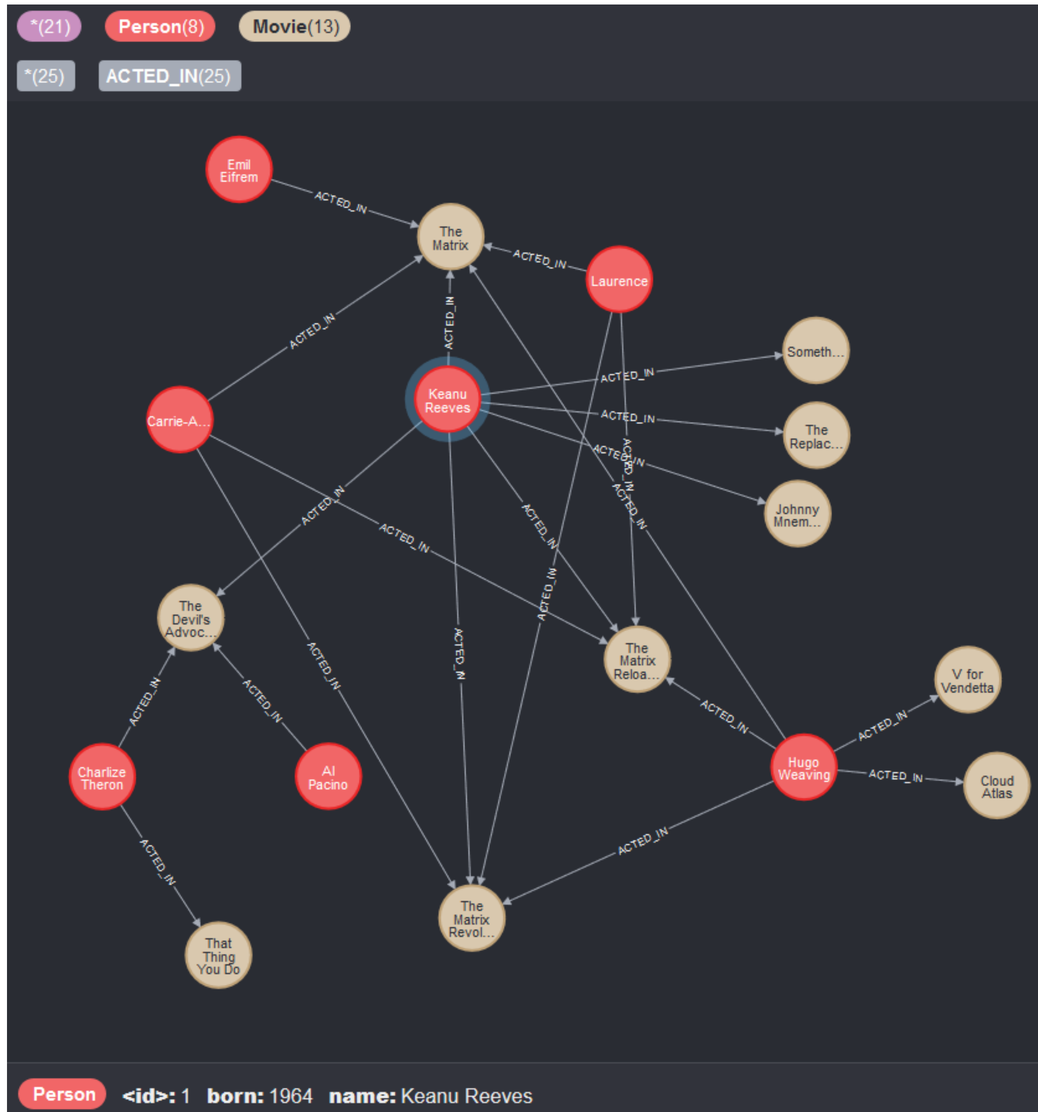
Neo4j – Cypher

CPAP, FING, UdelaR – 2022

Cypher Language

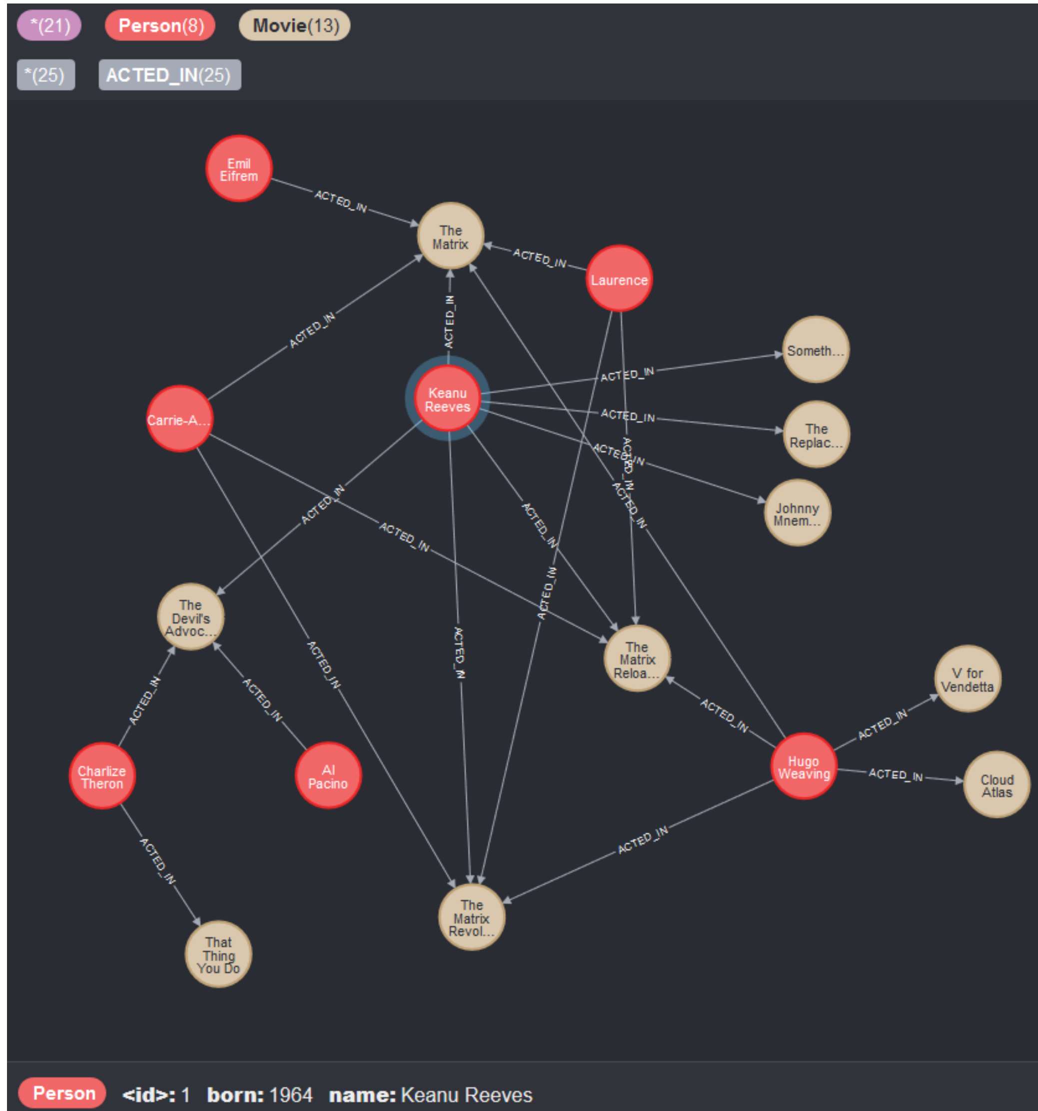
- Declarative query language that allows users to state what actions to perform upon their graph data
- Inspired by SQL with the concept of pattern matching taken from SPARQL (SPARQL Protocol and RDF Query Language)
- Open sourced in 2015 by Neo4j, Inc. aiming that it becomes the "SQL for graphs" (openCypher)

Property Graphs in Neo4j



- **Nodes**:
  - grouped by labels `Person` and `Movie`:
    - Keanu Reeves (`Person`)
    - The Matrix (`Movie`)
- **Relationships**:
  - `ACTED_IN`: from `Person` nodes to `Movie` nodes

Property Graphs in Neo4j



- **Properties**:
  - Nodes:
    - `Person`:
      - `born` (integer)
      - `name` (string)
    - `Movie`:
      - `released` (integer)
      - `title` (string)
      - `tagline` (string)
  - Relationships:
    - `ACTED_IN`:
      - `roles` (array of string)

Cypher Patterns - ASCII Art in Nodes

- **Nodes**:
  - surrounded by parenthesis
    `()` or `(p1)`
- **Labels**:
  - start with colon
    `(p1:Person)`
- **Properties**:
  - wrapped with braces and property name separated by a colon from property value
    `(p1:Person {name: “Keanu Reeves”})`

Cypher Patterns – ASCII Art in Relationship

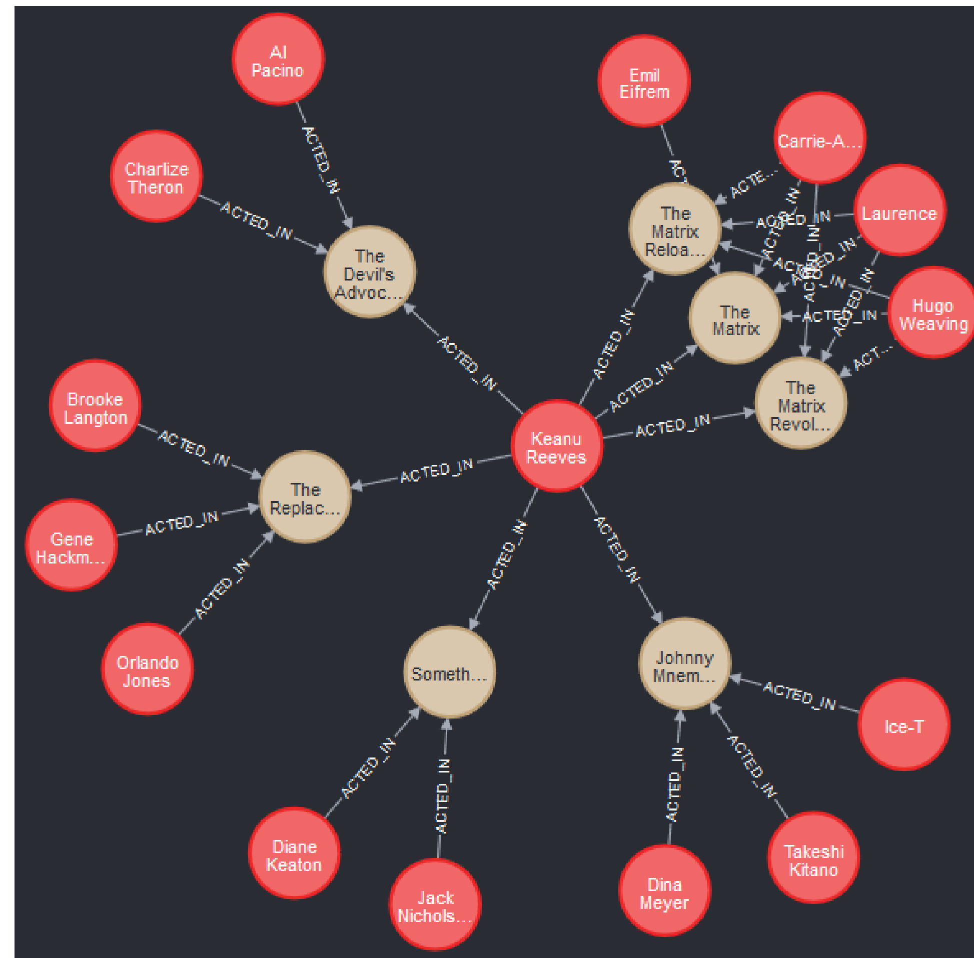
- **Relationships**:
  - wrapped with hyphens or square brackets
    `-->` or `-[a:ACTED_IN]->`

- **Direction**:
  - specified by `<` or `>`
    `(p1)-[:ACTED_IN]->(m)` or
    `(m)<-[:ACTED_IN]-(p2)`

- **Properties**:
  - analogus to nodes
    `-[:ACTED_IN {roles: “Neo”}]->`

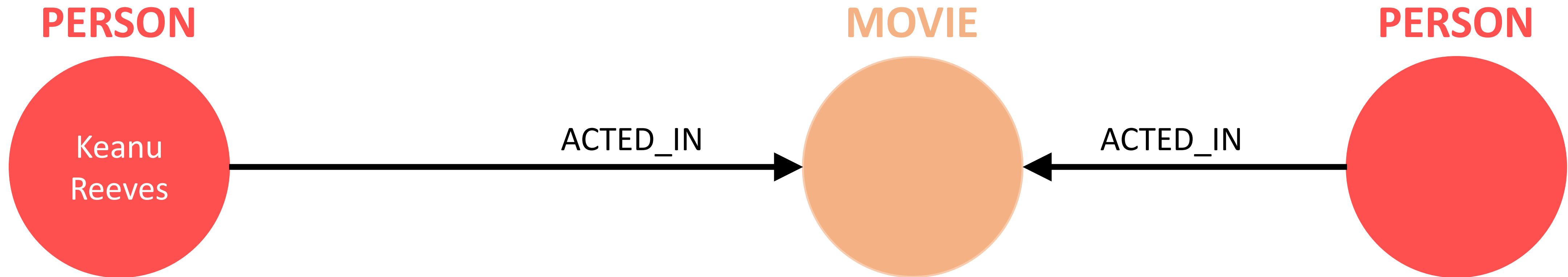
Cypher Patterns

- Get all persons who co-acted with Keanu Reeves.



Cypher Patterns

- Get all persons who co-acted with Keanu Reeves.




```
(p1:Person {name: “Keanu Reeves”}) -[:ACTED_IN]-> (m:Movie) <-[:ACTED_IN]- (p2:Person)
```

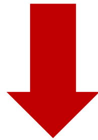
PATTERN:

```
(p1:Person {name: "Keanu Reeves"})-[:ACTED_IN]→(m:Movie)←[:ACTED_IN]-(p2:Person)
```

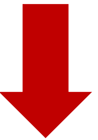
Cypher - CREATE (Node)

- **`AddNode(G,x)`** : adds node x to graph G

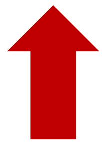
LABELS



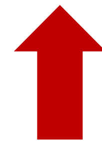
RETURN created node (optional)



```
CREATE (p:Person {name: "Keanu Reeves", born: "1964"}) RETURN p
```



VARIABLE (temporary store node)



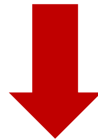
PROPERTIES

* Multiple nodes can be created in one statement separeted by a colon “ , ”

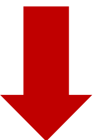
Cypher - CREATE (Relationship)

- **`Add(G,x,y,l)`**: adds an edge to graph `G` between nodes node `x` and `y` with label `l`

RELATIONSHIP LABEL



RETURN created relationship (optional)



```
CREATE (p:Person {name: "Keanu Reeves"})-[a:ACTED_IN {roles: ["Neo"]}]->(m:Movie {title: "The Matrix"}) RETURN p,a,m
```

NODE PATTERN

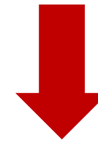
PROPERTIES

NODE PATTERN

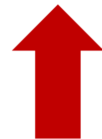
Cypher - DELETE (Node)

- **`DeleteNode(G,x)`**: deletes the node x from graph G

DETACH inbound/outbound relationships (optional)



```
MATCH (p:Person {name: "Keanu Reeves"}) DETACH DELETE p
```



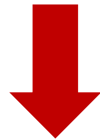
NODE PATTERN

* You cannot delete a node without also deleting relationships that start or end on said node.

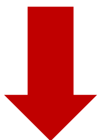
Cypher - DELETE (Relationship)

- **`Delete(G,x,y,l)`**: deletes an edge from graph `G` between nodes node `x` and `y` with label `l`

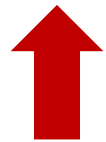
RELATIONSHIP LABEL



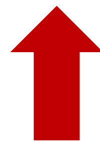
RETURN created relationship (optional)



```
MATCH (p:Person {name: "Keanu Reeves"})-[a:ACTED_IN]->(m:Movie {title: "The Matrix"}) DELETE a
```



NODE PATTERN



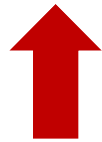
RELATIONSHIP PATTERN

NODE PATTERN

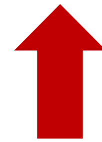
Cypher - ADJACENT

- **`Adjacent(G,x,y)`** : tests if there is an edge from `x` to `y` in graph `G`

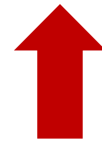
```
MATCH (p:Person {name: "Keanu Reeves"}),(m:Movie {title: "The Matrix"}) RETURN EXISTS((p)⟶(m))
```



NODE PATTERN



NODE PATTERN

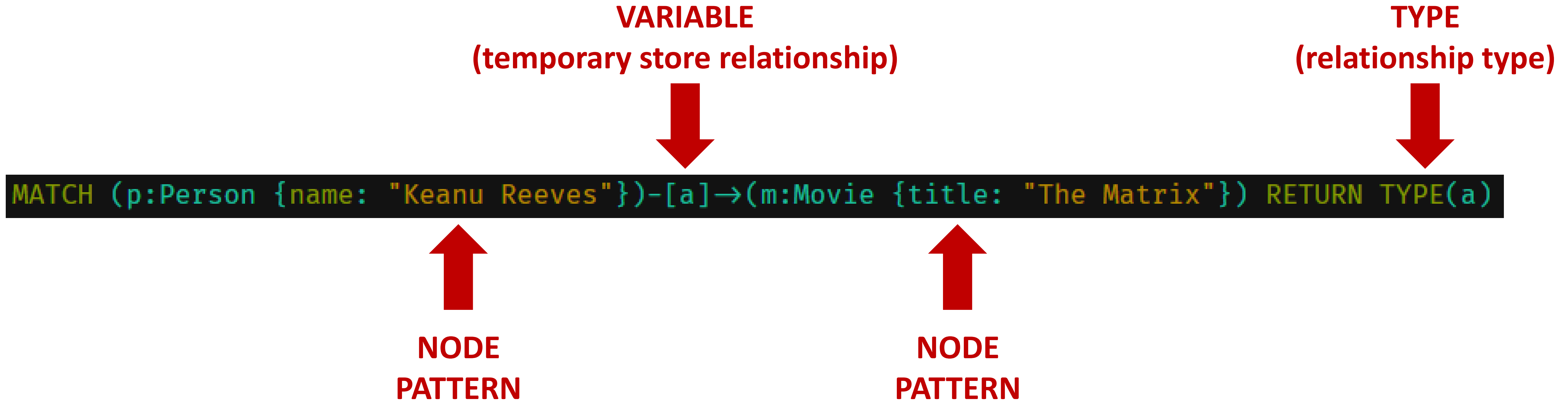


VARIABLE (temporary store path)

* `EXISTS` returns true if a match for the pattern exists in the graph, or if the specified property exists in the node, relationship or map.

Cypher - ADJACENT EDGES

- **`AdjacentEdges(G,x,y)`** : set of labels of edges from `x` to `y` in graph `G`

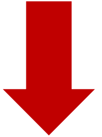


* `TYPE` returns the string representation of the relationship type.

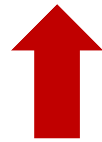
Cypher - REACH

- **`Reach(G,x,y)`**: tests if there is a path from `x` to `y` in graph `G`

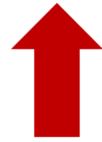
EXISTS (path)



```
MATCH (p1:Person {name: "Keanu Reeves"}),(p2:Person {name: "Lana Wachowski"}) RETURN EXISTS((p1)-[*]→(p2))
```



NODE PATTERN



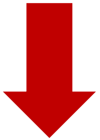
NODE PATTERN

* `EXISTS` returns true if a match for the pattern exists in the graph, or if the specified property exists in the node, relationship or map.

Cypher - PATH

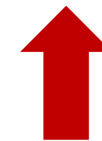
- **`Path(G,x,y)`**: a shortest path from `x` to `y` in graph `G`

SHORTESTPATH
(a single shortest path)



```
MATCH r=SHORTESTPATH((p1:Person {name: "Keanu Reeves"})-[*]→(p2:Person {name: "Lana Wachowski"})) RETURN r
```

NODE PATTERN



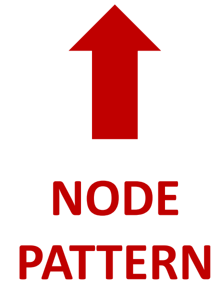
NODE PATTERN

* `SHORTESTPATH` finds a single shortest path.
* `ALLSHORTESTPATHS` finds all shortest paths.

Cypher - N-HOP

- **`N-hop(G,x)`** : set of nodes y where exists a path of length n from x to y in graph G

```
MATCH r=((p:Person {name: "Keanu Reeves"})-[*3]->()) RETURN r
```



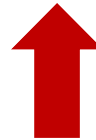
NODE PATTERN

N
(number of hops)

Cypher - More…

- Match on multiple relationship types:

```
MATCH r=(m:Movie {title: "The Matrix"})-[:DIRECTED|:PRODUCED]-() RETURN r
```



Match multiple relationship types

Cypher - More…

- Variable length relationships

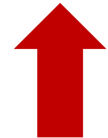
```
MATCH r=(p:Person {name: "Keanu Reeves"})-[*2..4]-() RETURN r
```

Variable length path:
minHops: 2
maxHops: 4

Cypher - More…

- Relationship variable in variable length relationships (return edges)

```
MATCH r=(p:Person {name: "Keanu Reeves"})-[*2..3]-() RETURN RELATIONSHIPS(r)
```



List relationships

* `RELATIONSHIPS` returns a list of relationships comprising a variable length path between to nodes.

Cypher - More…

- Return only properties:

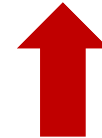
```
MATCH r=((p:Person)--(m:Movie {released:1999})) RETURN p.name, m.title
```

Return properties

Cypher - SET (Properties)

- Set `rating` of `9.9` for title `The Matrix`:

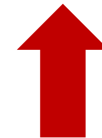
```
MATCH (m:Movie {title: "The Matrix"}) SET m.rating=9.9 RETURN m
```



Rating value

- Remove `rating` property for title `The Matrix`:

```
MATCH (m:Movie {title: "The Matrix"}) SET m.rating=NULL RETURN m
```

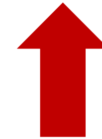


Removing property

Cypher - SET (Labels)

- Set `Favorite` label to title `The Matrix`:

```
MATCH (m:Movie {title: "The Matrix"}) SET m:Favorite RETURN m
```

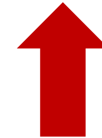


LABEL

Cypher - REMOVE (Properties/Labels)

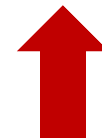
- Remove `rating` property from title `The Matrix`:

```
MATCH (m:Movie {title: "The Matrix"}) REMOVE m.rating RETURN m
```



- Remove `:Favorite` label from title `The Matrix`:

```
MATCH (m:Movie {title: "The Matrix"}) REMOVE m:Favorite RETURN m
```



Cypher - WHERE

- All titles released between 2000 and 2010:

```
MATCH (m:Movie) WHERE m.released ≥ 2000 AND m. released≤ 2010 RETURN m
```

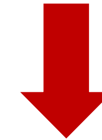
- It is also possible to nest existential subqueries (must return true or false):

```
MATCH (m:Movie)
WHERE EXISTS {
    MATCH (m)-[]-(p:Person)
    WHERE p.name ENDS WITH "Wachowski"
}
RETURN m
```

Cypher - Aggregating Functions

- Analogus to SQL `GROUP BY`:
  - `avg`
  - `count`
  - `max`
  - `min`
  - `sum`
  - etc

Aggregating Function



```
MATCH (m:Movie) RETURN m.released, COUNT(m)
```

Grouping key

Cypher - Aggregating Functions

- `collect`: returns a list containing the values returned by an expression

```
MATCH (m:Movie {released: 2006}) RETURN COLLECT(m.title)
```

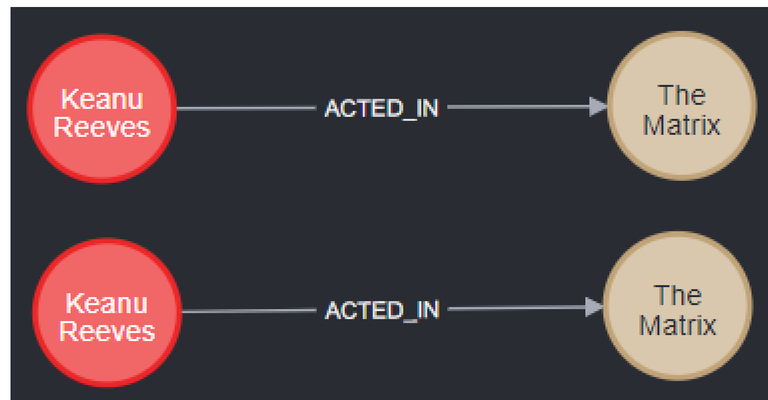
- Result:

"COLLECT(m.title)"
["RescueDawn","The Da Vinci Code","V for Vendetta"]

Cypher - Internal IDs

- it is possible to create two distinct relationships with the same label between two nodes:

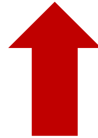
- it is also possible to create two distinct nodes with the same label and same properties:



Cypher - Internal IDs

- `id`: returns the internal ID of the relationship or node.

```
MATCH (p) WHERE ID(p) = 1 RETURN p
```



Internal ID

Cypher - Constraints

Constraint	Description	Neo4j Edition
Unique node property	ensures that property values are unique for all nodes with a specific label	Community
Node property existence	ensures that a property exists for all nodes with a specific label	Enterprise
Relationship property existence	ensures that a property exists for all relationships with a specific type	Enterprise
Node key	ensures that, for a given label and set of properties: i. all the properties exist on all the nodes with that label ii. the combination of the property values is unique	Enterprise

https://neo4j.com/docs/cypher-manual/current/constraints/

Cypher - Unique Node Property Constraints

- Adding a unique property constraint on a property will also add a single-property index on that property

```
CREATE CONSTRAINT movie_title_unique
ON (m:Movie)
ASSERT m.title IS UNIQUE
```

DEPRECATED

```
CREATE CONSTRAINT FOR (m:Movie)
    REQUIRE m.title IS UNIQUE
```

```
$ CALL db.indexes
```

"id"	"name"	"state"	"populationPercent"	"uniqueness"	"type"	"entityType"	"labelsOrTypes"	"properties"	"provider"
2	"movie_title_unique"	"ONLINE"	100.0	"UNIQUE"	"BTREE"	"NODE"	["Movie"]	["title"]	"native-btree-1.0"

Cypher - Indexes

- Single-property:

```
CREATE INDEX person_name_index FOR (p:Person) ON (p.name)
```

- Composite:

```
CREATE INDEX movie_released_title_index FOR (m:Movie) ON (m.released, m.title)
```

Cypher - Query Execution

1. Convert the input query string into an abstract syntax tree (AST)
   - query string is first tokenized and then parsed into an AST
   - perform semantic checking of the variable types and scoping of variables within the tree

2. Optimize and normalize the AST
   - simple optimizations and normalizations, i.e:
     - moving all labels and types from the `MATCH` clause to `WHERE`
     - suppressing redundant `WITH`
     - expanding aliases: `RETURN * => RETURN x AS x, y AS y`
     - folding of constants: `1+2*4 => 9`
     - naming anonymous pattern nodes: `MATCH () => MATCH (n)`
     - converting the equality operator to an `IN`:
       `MATCH (n) WHERE id(n)=12 => MATCH n WHERE id(n) IN [12]`
     - other normalizations

https://neo4j.com/blog/introducing-new-cypher-query-optimizer

Cypher - Query Execution

3. Create a query graph from the normalized AST
   - more abstract, high level representation of the query
   - allows to compute costs and perform optimizations far more effectively

4. Create a logical plan from X
   - a logical plan is produced in a step-by-step fashion following a bottom-up approach for each query graph (depending on the query, a query graph may consist of sub query graphs)
   - the cost of a logical plan is an estimate of the amount of work the database will have to do in order to execute it (dominated by I/O reads from the store and indices, and in-memory computational work such as expanding the graph by traversing more relationships and hence gathering more nodes)

5. Rewrite the logical plan
   - the logical plan is now rewritten using un-nesting, merging and simplification of various components

https://neo4j.com/blog/introducing-new-cypher-query-optimizer

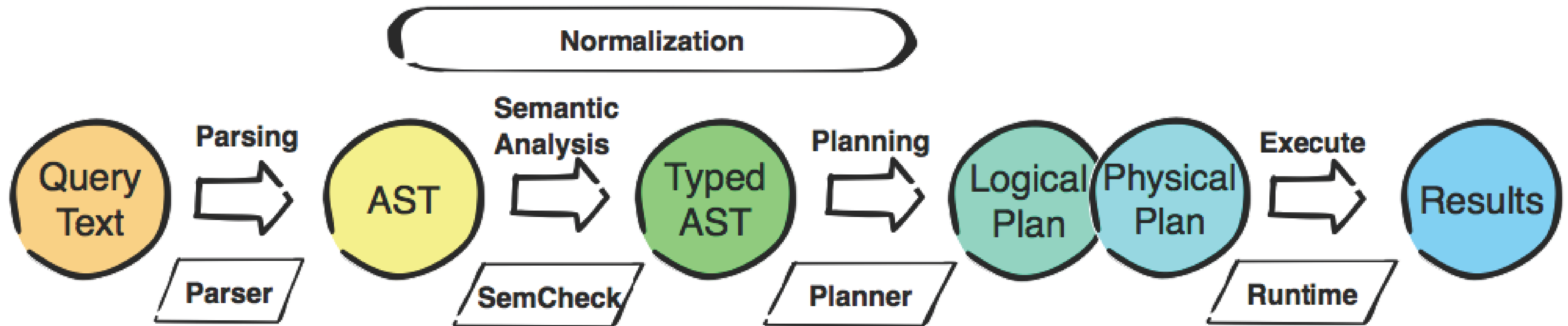
Cypher - Query Execution

6. Create an execution plan from the logical plan
    - choose a physical implementation for logical operators

7. Execute the query using the execution plan

https://neo4j.com/blog/introducing-new-cypher-query-optimizer

Cypher - Query Execution



https://neo4j.com/blog/introducing-new-cypher-query-optimizer

Cypher - Query Profiling

- `EXPLAIN`:
  - see the execution plan but not run the statement
  - return an empty result and make no changes to the database

- `PROFILE`:
  - run the statement and see which operators are doing most of the work

References

- Neo4j Cypher Refcard
  https://neo4j.com/docs/cypher-refcard/current/
- Cypher Query Language
  https://neo4j.com/developer/cypher/
- The Neo4j Cypher Manual v4.4
  https://neo4j.com/docs/cypher-manual/current/
- Introducing the new Cypher Query Optimizer
  https://neo4j.com/blog/introducing-new-cypher-query-optimizer/