Cypher Query Language

Dr James Xue

James.xue@northampton.ac.uk

Cypher Clauses

- CREATE (nodes and relationships)
- DELETE (nodes and relationships)
- CONSTRAINT and INDEX (create and drop)
- SET and REMOVE (properties and labels)
- MERGE and CREATE UNIQUE (nodes and relationships)
- FOREACH (to update data within a list)
- LOAD CSV
- MATCH and OPTIONAL MATCH
- AGGREGATION
- WITH
- COLLECT and UNWIND
- CASE and LIST
- …

LOAD CSV

- LOAD CSV is used to import data from CSV files into Neo4j.
- It is required to specify a variable for the CSV data using AS.
- LOAD CSV supports resources compressed with *gzip*, *Deflate*, as well as *ZIP* archives.

LOAD CSV …

- CSV files can be stored on the database server (usually in the ~*Neo4j/default.graphdb/import* folder) and are then accessible using a file:/// URL.
- LOAD CSV also supports accessing CSV files via *HTTPS*, *HTTP*, and *FTP*.
  - E.g., loading data from Google sheet, Dropbox, Github is common practice
- LOAD CSV is often used in conjunction with the PERIODIC COMMIT clause for large dataset.

LOAD CSV From A Local File Without Headers

- Store the csv file in the *~Neo4j/default.graphdb/import* folder
- Run the following command in Neo4j browser.
- The *row* serves like an array, with index starting from 0
- The commands create all computing module nodes in Neo4j
- Ensure the order of the columns in the csv file matches the order of the attributes of the nodes to be created.

Computing_modules_without_headers.csv

```
CSY1017,Computer Communication,4,20
CSY1014,Computer Systems,4,20
CSY1018,Web Development,4,20
CSY1019,Software Engineering 1,4,20
CSY1020,Problem Solving and Programming,4,20
```

```
LOAD CSV FROM
"file:///computing_modules_without_headers.csv" as row
CREATE (m:Module{code:row[0],title:row[1],
level:toInteger(row[2]),credits:toInteger(row[3])})
```

Cypher function to convert string values into Integers.

Import Data From a CSV File Containing Headers

- When your CSV file has headers, you can view each row in the file as a map instead of as an array of strings.
- This time, the file starts with a single row containing column names. Indicate this using WITH HEADERS and you can access specific fields by their corresponding column name.

Column headers

Computing_modules.csv

```
code,title,level,credits
CSY1017,Computer Communication,4,20
CSY1014,Computer Systems,4,20
CSY1018,Web Development,4,20
CSY1019,Software Engineering 1,4,20
CSY1020,Problem Solving and Programming,4,20
```

```
LOAD CSV WITH HEADERS FROM
"file:///computing_modules.csv" as row
CREATE (m:Module{code:row.code,title:row.title,
level:toInteger(row.level),
credits:toInteger(row.credits)})
```

Import Data From a CSV File With a Custom Field Delimiter

- You can specify which delimiter your file uses using FIELDTERMINATOR.
- As values in this file are separated by a semicolon, a custom FIELDTERMINATOR is specified in the LOAD CSV clause.

Computing_modules_semicolon.csv

```
code;title;level;credits
CSY1017;Computer Communication;4;20
CSY1014;Computer Systems;4;20
CSY1018;Web Development;4;20
CSY1019;Software Engineering 1;4;20
CSY1020;Problem Solving and Programming;4;20
```

Columns (including the headers) are Separated by ;

```
LOAD CSV WITH HEADERS FROM
"file:///computing_modules_semicolon.csv" as row
FIELDTERMINATOR ';'
CREATE (m:Module{code:row.code,title:row.title,
level:toInteger(row.level),
credits:toInteger(row.credits)})
```

Convert Other Formats Into CSV

- Often, files are in other formats (e.g., json, XML), which needs to be converted using online tools (e.g., http://www.csvjson.com/csv2json)

Modules.json:

```
[
  {
    "code": "CSY1017",
    "title": "Computer Communication",
    "level": 4,
    "credits": 20
  },
  {
    "code": "CSY1014",
    "title": "Computer Systems",
    "level": 4,
    "credits": 20
  },
```

Modules.csv

```
code,title,level,credits
CSY1017,Computer Communication,4,20
CSY1014,Computer Systems,4,20
CSY1018,Web Development,4,20
CSY1019,Software Engineering 1,4,20
CSY1020,Problem Solving and Programming,4,20
```

```
LOAD CSV WITH HEADERS FROM
"file:///computing_modules.csv" as row
MERGE (m:Module{code:row.code, title:row.title,
level:toInteger(row.level),
credits:toInteger(row.credits)})


LOAD CSV WITH HEADERS FROM
"file:///computing_pre_requisites.csv" as row
MATCH (m1:Module{code:row.module1})
MATCH (m2:Module{code:row.module2})
MERGE (m1)-[:PRE_REQUISITE]->(m2)
```

computing_modules.csv — Edited

```
code,title,level,credits
CSY1017,Computer Communication,4,20
CSY1014,Computer Systems,4,20
CSY1018,Web Development,4,20
CSY1019,Software Engineering 1,4,20
CSY1020,Problem Solving and Programming,4,20
CSY1026,Databases 1,4,20
CSY1030,Digital Footprint,4,20
CSY1024,Game Techniques 1,4,20
CSY1025,Group Project 1 (Games),4,20
CSY2041,Quality and User-centered Systems,5,20
BUS2015,Service Management,5,20
CSY2042,Website Management,5,20
CSY2043,Website Design,5,20
CSY2001,Computer Networks,5,20
```

computing_pre_requisites.csv — Edited

```
module1,module2,relationship
CSY2042,CSY1018,PRE_REQUISITE
CSY2001,CSY1017,PRE_REQUISITE
CSY2002,CSY1014,PRE_REQUISITE
CSY2006,CSY1019,PRE_REQUISITE
CSY2008,CSY1019,PRE_REQUISITE
CSY2015,CSY1014,PRE_REQUISITE
CSY2028,CSY1018,PRE_REQUISITE
CSY2028,CSY1020,PRE_REQUISITE
CSY2030,CSY1020,PRE_REQUISITE
CSY2038,CSY1026,PRE_REQUISITE
```

Exercises

- Log on to NILE to find the csv files
- Using the LOAD CSV command and load the data and create graph nodes with corresponding attributes and relationships.
- Research on how to load data in other formats (e.g., XML).
- Research on how to load data from online storage, e.g., Google sheet, Dropbox, GibHub, etc.
- Log your learning activity and reflection in the diary

Read Clauses

MATCH

- The MATCH clause is used to search for the pattern described in it.
- MATCH is often coupled to a WHERE part which adds restrictions, or predicates, to the MATCH patterns, making them more specific.
- Cypher is declarative, and so usually the query itself does not specify the algorithm to use to perform the search. Neo4j will automatically work out the best approach to finding start nodes and matching patterns.

A Typical Cypher Query

```
MATCH (m:Movie)<-[:RATED]-(u:User)
WHERE m.title CONTAINS "Matrix"
WITH m.title AS movie, COUNT(*) AS reviews
RETURN movie, reviews
ORDER BY reviews DESC
LIMIT 5
```

find	`MATCH (m:Movie)<-[:RATED]-(u:User)`	Search for an existing graph pattern
filter	`WHERE m.title CONTAINS "Matrix"`	Filter matching paths to only those matching a predicate
aggregate	`WITH m.title AS movie, COUNT(*) AS reviews`	Count number of paths matched for each movie
return	`RETURN movie, reviews`	Specify columns to be returned by the statement
order	`ORDER BY reviews DESC`	Order by number of reviews, in descending order
limit	`LIMIT 5;`	Only return first 4 records

Should be 5

Basic Node Finding

- Get all nodes – by just specifying a pattern with a single node and no labels, all nodes in the graph will be returned.

```
MATCH (n)
RETURN n
```

Returns all the nodes in the database.

- Get all nodes with a label:

```
MATCH (movie:Movie)
RETURN movie.title
```

Returns all the movies in the database.

Match by Relationships

- Match by relationship types:

```
MATCH (matrix:Movie{title:'The Matrix'})<-[:ACTED_IN]-(actor)
RETURN actor.name
```

The query returns all actors that ACTED_IIN the Matrix movie

- Match by multiple relationship types:

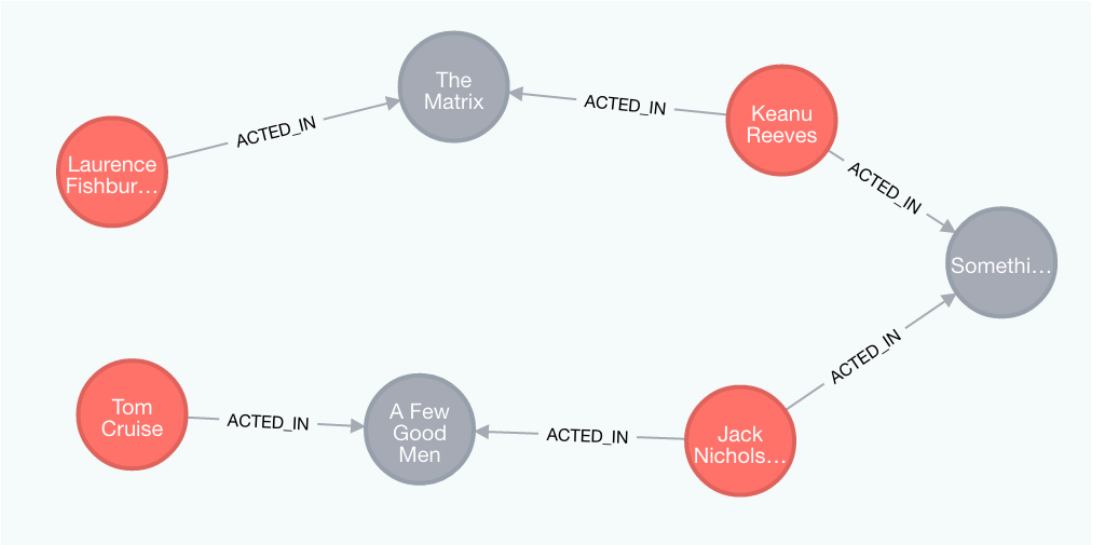
```
MATCH (tom:Person{name:'Tom Cruise'})-[:ACTED_IN]->
(movie)<-[:DIRECTED]-(director)
RETURN movie.title, director.name
```

The query returns all the movies ‘Tom Cruise’ ACTED_IN and the directors’ names.

Find Shortest Path

- Single shortest path, e.g., finding a single shortest path between two nodes, as long as the path is max 15 relationships long

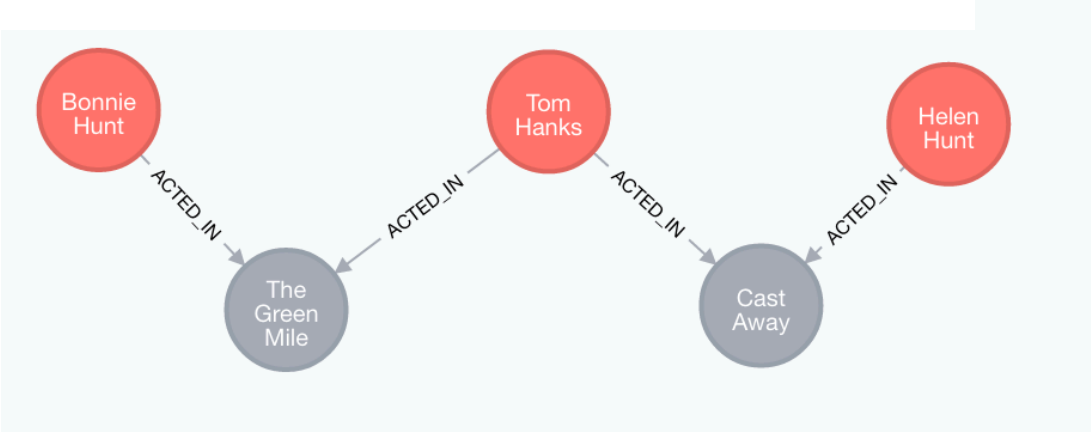
```
MATCH (tom:Person{name:"Tom Cruise"}),
(laurence:Person{name:"Laurence Fishburne"}),
p=shortestPath((tom)-[*..15]-(laurence))
RETURN p
```



Find Shortest Path with Predicates

- Find the shortest path between “Helen Hunt” and “Bonnie Hunt”, and the WHERE predicate will ensure that we don’t consider the Mother/daughter relationship between the two.

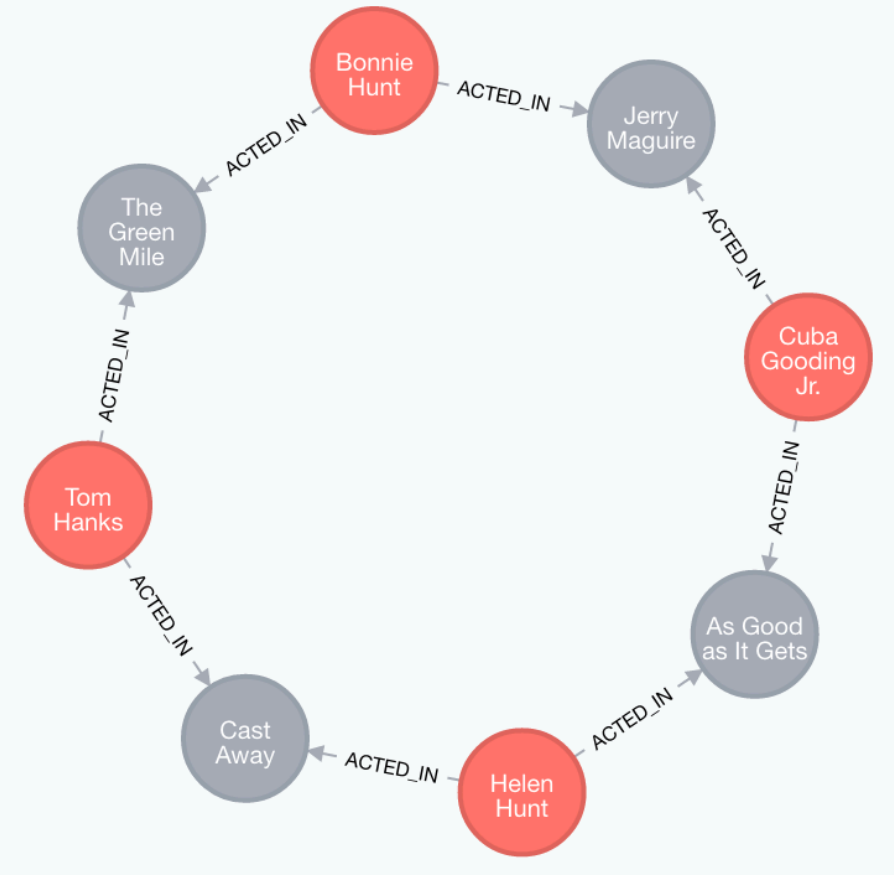
```
MATCH (helen:Person{name:"Helen Hunt"}),
(bonnie:Person{name:"Bonnie Hunt"}),
p=shortestPath((helen)-[*]-(bonnie))
WHERE NONE (r IN rels(p) WHERE type(r) = "MOTHER")
RETURN p
```



Find Shortest Path …

- All shortest paths between two nodes.

```
MATCH (helen:Person{name:"Helen Hunt"}),
(bonnie:Person{name:"Bonnie Hunt"}),
p=allShortestPaths((helen)-[*]-(bonnie))
RETURN p
```



OPTIONAL MATCH

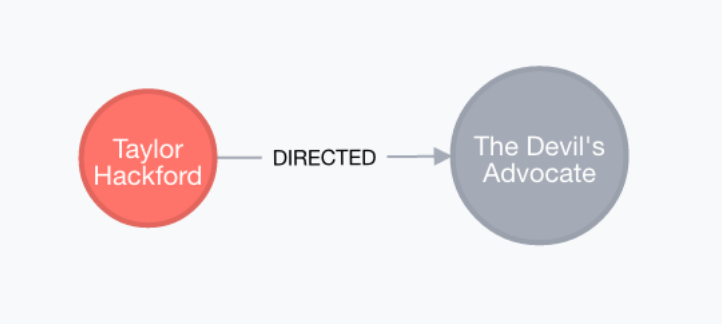
- The OPTIONAL MATCH clause is used to search for the pattern described in it
- NULL values are used for missing parts of the pattern.
- OPTIONAL MATCH could be considered the Cypher equivalent of the outer join in SQL.

OPTIONAL MATCH …

```
MATCH (taylor{name:'Taylor Hackford'})-[r]->() RETURN type(r)
```

Output:

type(r)
DIRECTED




```
MATCH (taylor{name:'Taylor Hackford'}) MATCH (taylor)-[r:ACTED_IN]->() RETURN r
```

Output: (no changes, no records)

```
MATCH (taylor{name:'Taylor Hackford'}) OPTIONAL MATCH (taylor)-[r:ACTED_IN]->() RETURN r
```

Output:

r
null

Exercises

- Clear the database.
- Load the Movie graph database.
- Run the commands to practice ”shortestPath()” and “allShortestPaths()” functions
- Make the following changes and run the codes again:
  - Change the path lengths from “*..15” to “*..5” in the first “shortestPath” example
  - Remove the NONE command in the second “shortestPath()” example
- Run the commands on the previous slides to practice the MATCH and OPTIONAL MATCH clause.
- Log your learning activities and reflection in the diary.