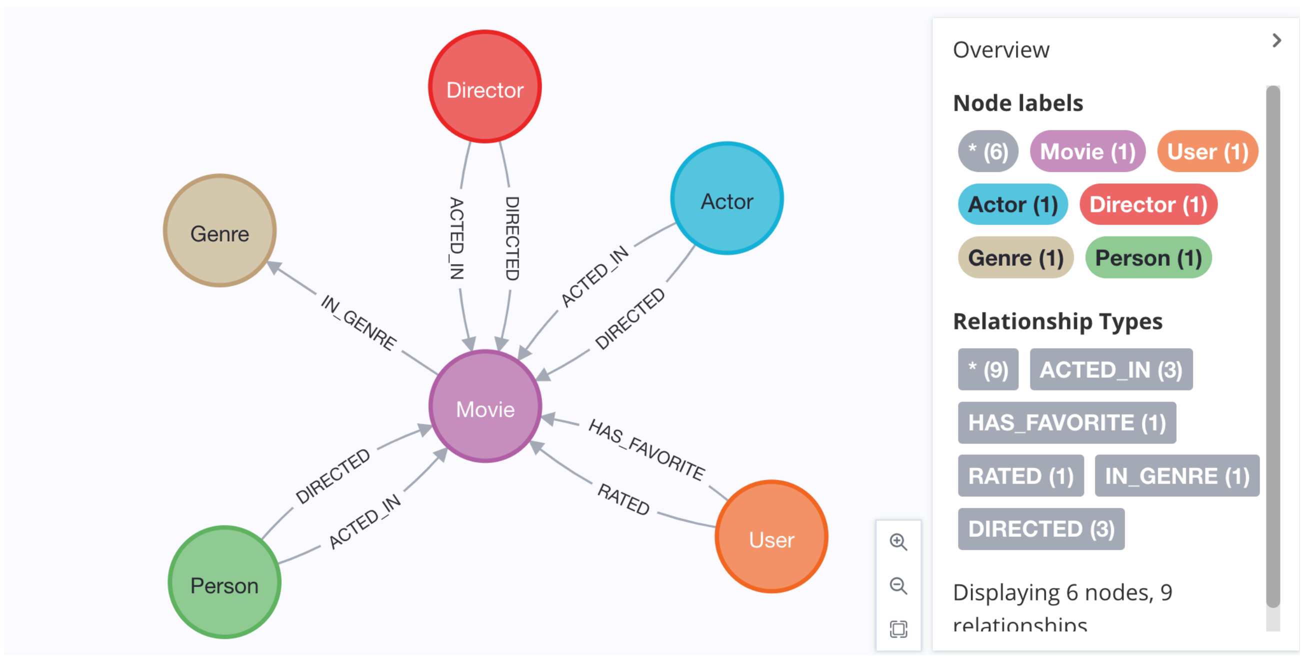
Intermediate Cypher Queries (Part1)

1. Data Model

```
CALL db.schema.visualization()
```



Overview

Node labels

* (6) Movie (1) User (1) Actor (1) Director (1) Genre (1) Person (1)

Relationship Types

* (9) ACTED_IN (3) HAS_FAVORITE (1) RATED (1) IN_GENRE (1) DIRECTED (3)

Displaying 6 nodes, 9 relationships

- View the property types for nodes in the graph:

```
CALL db.schema.nodeTypeProperties()
```

- View the property types for relationships in the graph:

```
CALL db.schema.relTypeProperties()
```

```
╒═══════════════════╤══════════════╤═══════════════╤═══════════╕
│"relType"          │"propertyName"│"propertyTypes"│"mandatory"│
╞═══════════════════╪══════════════╪═══════════════╪═══════════╡
│":`IN_GENRE`"      │null          │null           │false      │
├───────────────────┼──────────────┼───────────────┼───────────┤
│":`RATED`"         │"rating"      │["Double"]     │true       │
├───────────────────┼──────────────┼───────────────┼───────────┤
│":`RATED`"         │"timestamp"   │["Long"]       │true       │
├───────────────────┼──────────────┼───────────────┼───────────┤
│":`ACTED_IN`"      │"role"        │["String"]     │false      │
├───────────────────┼──────────────┼───────────────┼───────────┤
│":`DIRECTED`"      │"role"        │["String"]     │false      │
├───────────────────┼──────────────┼───────────────┼───────────┤
│":`HAS_FAVORITE`"  │"createdAt"   │["DateTime"]   │true       │
└───────────────────┴──────────────┴───────────────┴───────────┘
```

- View the uniqueness constraint indexes

```
SHOW CONSTRAINTS
```

```
╒════╤═════════════════════╤════════════╤════════════╤═══════════════╤════════════╤══════════════╕
│"id"│"name"               │"type"      │"entityType"│"labelsOrTypes"│"properties"│"ownedIndexId"│
╞════╪═════════════════════╪════════════╪════════════╪═══════════════╪════════════╪══════════════╡
│13  │"constraint_3b27b0"  │"UNIQUENESS"│"NODE"      │["User"]       │["userId"]  │11            │
├────┼─────────────────────┼────────────┼────────────┼───────────────┼────────────┼──────────────┤
│3   │"constraint_3d5fcb7f"│"UNIQUENESS"│"NODE"      │["Movie"]      │["movieId"] │1             │
├────┼─────────────────────┼────────────┼────────────┼───────────────┼────────────┼──────────────┤
│44  │"constraint_4499eae9"│"UNIQUENESS"│"NODE"      │["Person"]     │["tmdbId"]  │41            │
├────┼─────────────────────┼────────────┼────────────┼───────────────┼────────────┼──────────────┤
│37  │"constraint_737d9c1d"│"UNIQUENESS"│"NODE"      │["Movie"]      │["tmdbId"]  │34            │
├────┼─────────────────────┼────────────┼────────────┼───────────────┼────────────┼──────────────┤
│8   │"constraint_f8689281"│"UNIQUENESS"│"NODE"      │["Genre"]      │["name"]    │6             │
└────┴─────────────────────┴────────────┴────────────┴───────────────┴────────────┴──────────────┘
```

2. Main References

- Neo4j Cypher Refcard
- Neo4j Cypher Manual

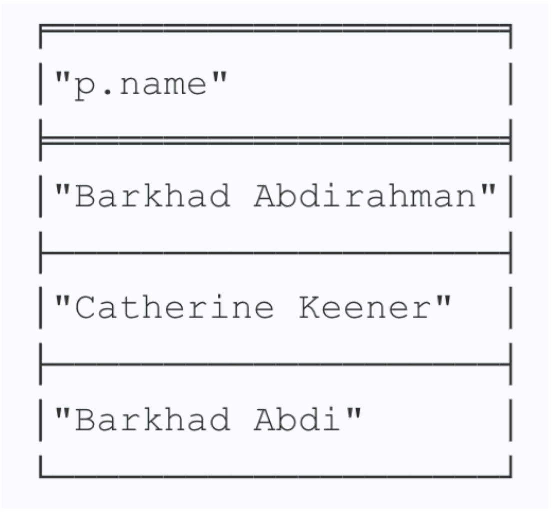
3. Basic Cypher Queries

Testing Inequalities

- Test inequality of a property use <> predicate.

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name <> 'Tom Hanks'
AND m.title = 'Captain Phillips'
RETURN p.name
```

This query returns the names of all actors that acted in the movie Captain Phillips, where Tom Hanks is excluded.

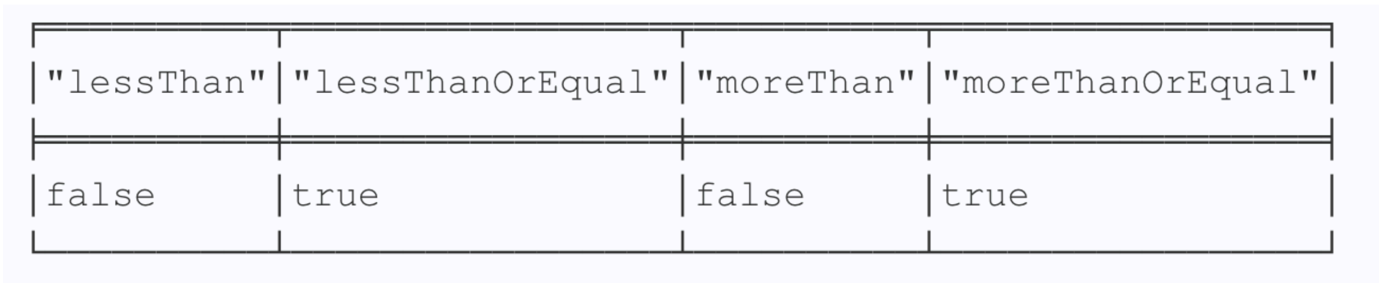


```
╒════════════════════╕
│"p.name"            │
╞════════════════════╡
│"Barkhad Abdirahman"│
├────────────────────┤
│"Catherine Keener"  │
├────────────────────┤
│"Barkhad Abdi"      │
└────────────────────┘
```

Testing less than or greater than

- Test both *numbers* and *strings* for values less than < or greater than > a value

```
MATCH (m:Movie) WHERE m.title = 'Toy Story'
RETURN
    m.year < 1995 AS lessThan,              //  Less than (false)
    m.year <= 1995 AS lessThanOrEqual,      // Less than or equal(true)
    m.year > 1995 AS moreThan,              // More than (false)
    m.year >= 1995 AS moreThanOrEqual       // More than or equal (true)
```

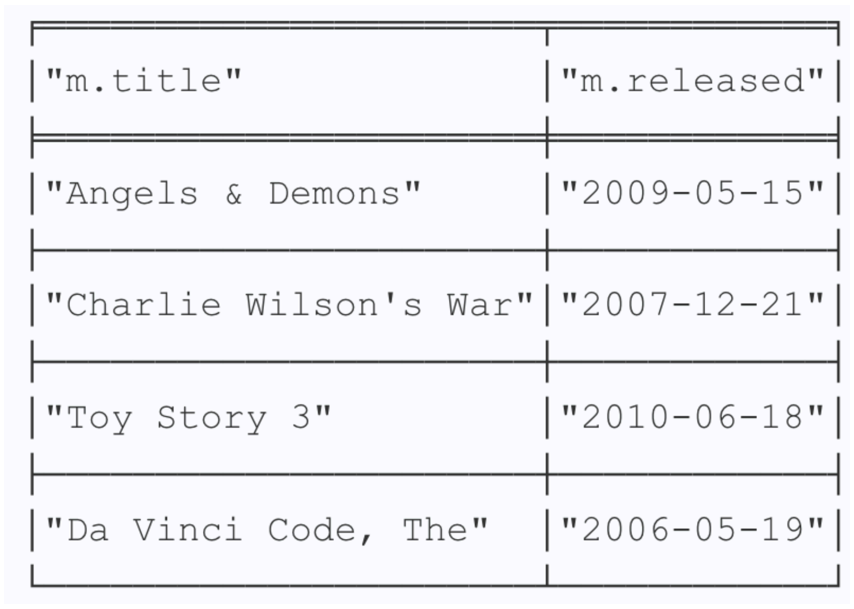


```
╒══════════╤═════════════════╤══════════╤═════════════════╕
│"lessThan"│"lessThanOrEqual"│"moreThan"│"moreThanOrEqual"│
╞══════════╪═════════════════╪══════════╪═════════════════╡
│false     │true             │false     │true             │
└──────────┴─────────────────┴──────────┴─────────────────┘
```

Testing Ranges

- Use a combination of less than and greater than.

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name = 'Tom Hanks'
AND  2005 <= m.year <= 2010
RETURN m.title, m.released
```



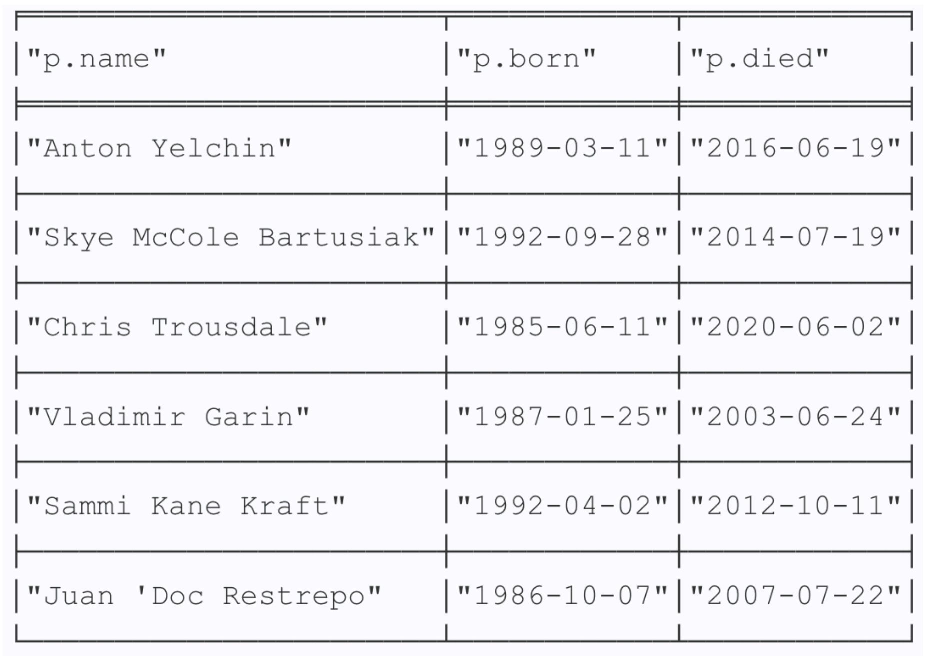
```
╒═════════════════════╤════════════╕
│"m.title"            │"m.released"│
╞═════════════════════╪════════════╡
│"Angels & Demons"    │"2009-05-15"│
├─────────────────────┼────────────┤
│"Charlie Wilson's War"│"2007-12-21"│
├─────────────────────┼────────────┤
│"Toy Story 3"        │"2010-06-18"│
├─────────────────────┼────────────┤
│"Da Vinci Code, The" │"2006-05-19"│
└─────────────────────┴────────────┘
```

Returns the four movies that Tom Hanks acted in between 2005 and 2010, inclusive.

Testing `null` property values

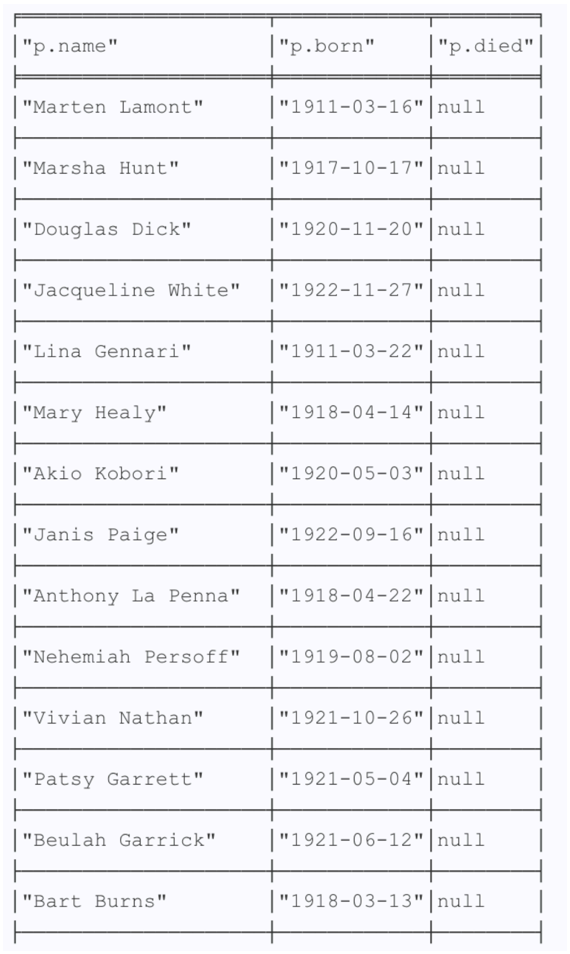
- A property of a node or relationship is null **if it does not exist**.
    - Test the existence of a property for a node using the `IS NOT NULL` predicate.
    - Test if a property exists using the `IS NULL` predicate

```
MATCH (p:Person)
WHERE p.died IS NOT NULL
AND p.born.year >= 1985
RETURN p.name, p.born, p.died
```



```
╒═══════════════════════╤════════════╤════════════╕
│"p.name"               │"p.born"    │"p.died"    │
╞═══════════════════════╪════════════╪════════════╡
│"Anton Yelchin"        │"1989-03-11"│"2016-06-19"│
├───────────────────────┼────────────┼────────────┤
│"Skye McCole Bartusiak"│"1992-09-28"│"2014-07-19"│
├───────────────────────┼────────────┼────────────┤
│"Chris Trousdale"      │"1985-06-11"│"2020-06-02"│
├───────────────────────┼────────────┼────────────┤
│"Vladimir Garin"       │"1987-01-25"│"2003-06-24"│
├───────────────────────┼────────────┼────────────┤
│"Sammi Kane Kraft"     │"1992-04-02"│"2012-10-11"│
├───────────────────────┼────────────┼────────────┤
│"Juan 'Doc Restrepo"   │"1986-10-07"│"2007-07-22"│
└───────────────────────┴────────────┴────────────┘
```

```
MATCH (p:Person)
WHERE p.died IS NULL
AND p.born.year <= 1922
RETURN p.name, p.born, p.died
```



```
╒═════════════════════╤════════════╤════════╕
│"p.name"             │"p.born"    │"p.died"│
╞═════════════════════╪════════════╪════════╡
│"Marten Lamont"      │"1911-03-16"│null    │
├─────────────────────┼────────────┼────────┤
│"Marsha Hunt"        │"1917-10-17"│null    │
├─────────────────────┼────────────┼────────┤
│"Douglas Dick"       │"1920-11-20"│null    │
├─────────────────────┼────────────┼────────┤
│"Jacqueline White"   │"1922-11-27"│null    │
├─────────────────────┼────────────┼────────┤
│"Lina Gennari"       │"1911-03-22"│null    │
├─────────────────────┼────────────┼────────┤
│"Mary Healy"         │"1918-04-14"│null    │
├─────────────────────┼────────────┼────────┤
│"Akio Kobori"        │"1920-05-03"│null    │
├─────────────────────┼────────────┼────────┤
│"Janis Paige"        │"1922-09-16"│null    │
├─────────────────────┼────────────┼────────┤
│"Anthony La Penna"   │"1918-04-22"│null    │
├─────────────────────┼────────────┼────────┤
│"Nehemiah Persoff"   │"1919-08-02"│null    │
├─────────────────────┼────────────┼────────┤
│"Vivian Nathan"      │"1921-10-26"│null    │
├─────────────────────┼────────────┼────────┤
│"Patsy Garrett"      │"1921-05-04"│null    │
├─────────────────────┼────────────┼────────┤
│"Beulah Garrick"     │"1921-06-12"│null    │
├─────────────────────┼────────────┼────────┤
│"Bart Burns"         │"1918-03-13"│null    │
├─────────────────────┼────────────┼────────┤
```

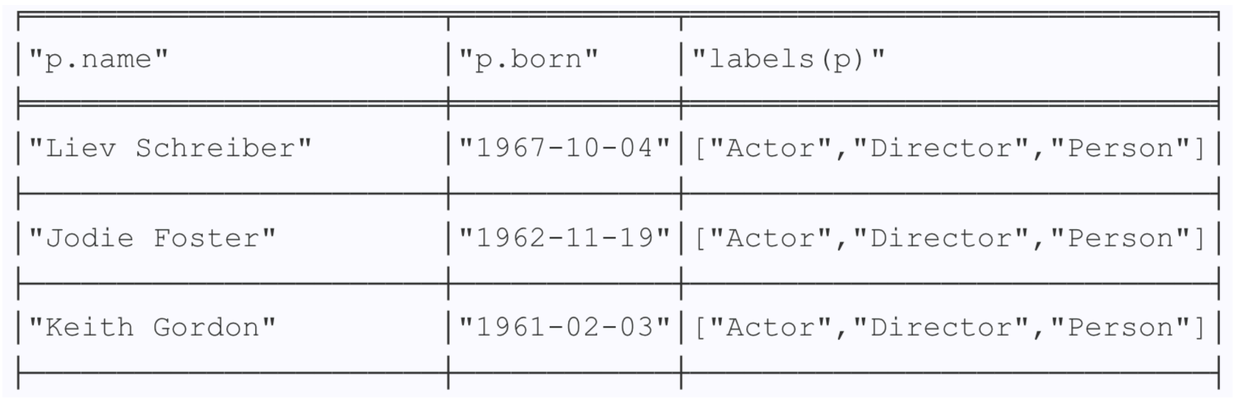
This query returns all people born before 1923 who do not have a died property value.

Testing labels or patterns

- Test for a label's existence on a node using the `{alias}:{label}`

```
MATCH (p:Person)
WHERE  p.born.year > 1960
AND p:Actor
AND p:Director
RETURN p.name, p.born, labels(p)
```

The `labels()` function returns the list of labels for a node.



```
╒════════════════════════╤════════════╤══════════════════════════════╕
│"p.name"                │"p.born"    │"labels(p)"                   │
╞════════════════════════╪════════════╪══════════════════════════════╡
│"Liev Schreiber"        │"1967-10-04"│["Actor","Director","Person"]│
├────────────────────────┼────────────┼──────────────────────────────┤
│"Jodie Foster"          │"1962-11-19"│["Actor","Director","Person"]│
├────────────────────────┼────────────┼──────────────────────────────┤
│"Keith Gordon"          │"1961-02-03"│["Actor","Director","Person"]│
└────────────────────────┴────────────┴──────────────────────────────┘
```

Discovering relationship types

- use the `type()` function to return the type of the relationship, r.
- Check the uniqueness, use `WITH DISTINCT` to constrain.

4. Testing Strings

- Use `START WITH`, `END WITH`, `CONTAIN` with `WHERE` clause to filter the characteristics in the string
- **Case sensitive**: If string values could be mis-interpreted if the letters do not match in case, your queries may miss data in the graph. Then we use `toUpper`,`toLower` to unify the property while matching.

```
MATCH (p:Person)
WHERE toLower(p.name) ENDS WITH 'demille'
RETURN p.name
```

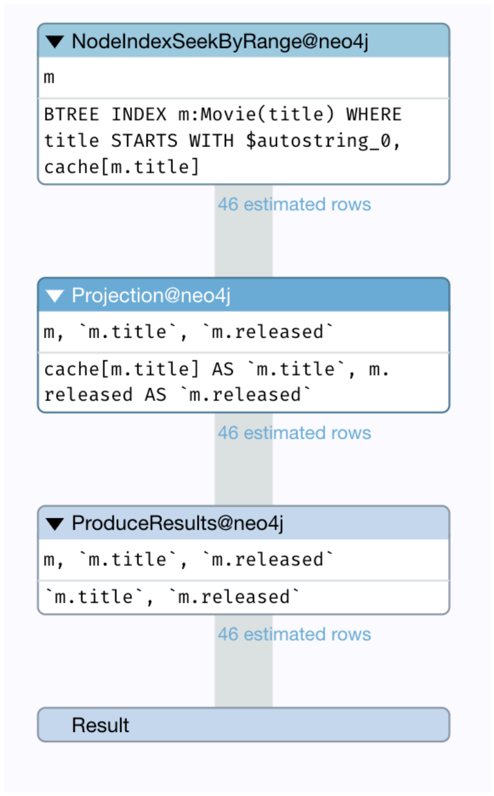
We do not know whether the data is 'DeMille', 'Demille', or 'deMlle' , to ensure matching all Person nodes that could be one of these, we transform the property value to *lower-case*.

```
MATCH (p:Person)
WHERE toUpper(p.name) CONTAINS ' DE '
RETURN p.name
```

To ensure matching all Person nodes that could be one of these, we transform the property value to *upper-case.*

- **Indexes for queries**: With any query,check if an index will be used by prefixing the query with `EXPLAIN`.

  ```
  EXPLAIN MATCH (m:Movie)
  WHERE  m.title STARTS WITH 'Toy Story'
  RETURN m.title, m.released
  ```

  ```
  NodeIndexSeekByRange@neo4j
  m
  BTREE INDEX m:Movie(title) WHERE
  title STARTS WITH $autostring_0,
  cache[m.title]
  46 estimated rows

  Projection@neo4j
  m, `m.title`, `m.released`
  cache[m.title] AS `m.title`, m.
  released AS `m.released`
  46 estimated rows

  ProduceResults@neo4j
  m, `m.title`, `m.released`
  `m.title`, `m.released`
  46 estimated rows

  Result
  ```

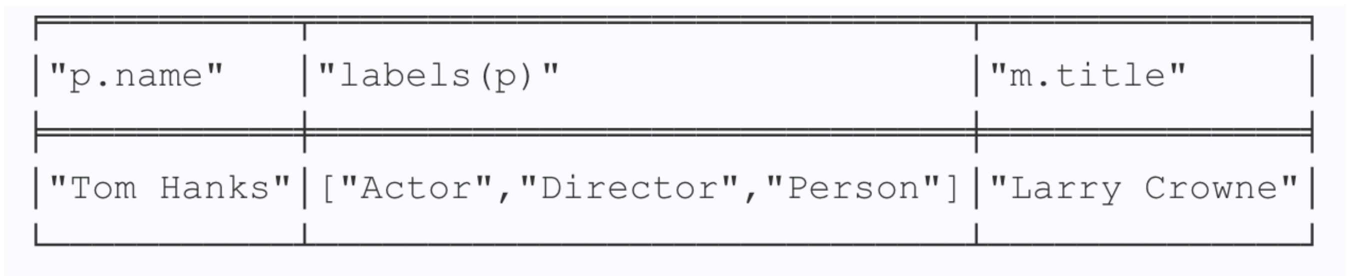
  This query produces the execution plan where the first step is NodeIndexSeekByRange.

5. Query Patterns and Performance

- A pattern is a combination of *nodes and relationships* that is used to traverse the graph at runtime

Write queries that test whether a pattern exists in the graph, using `exists { }` test:

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE  p.name = 'Tom Hanks'
AND exists {(p)-[:DIRECTED]->(m)}
RETURN p.name, labels(p), m.title
```

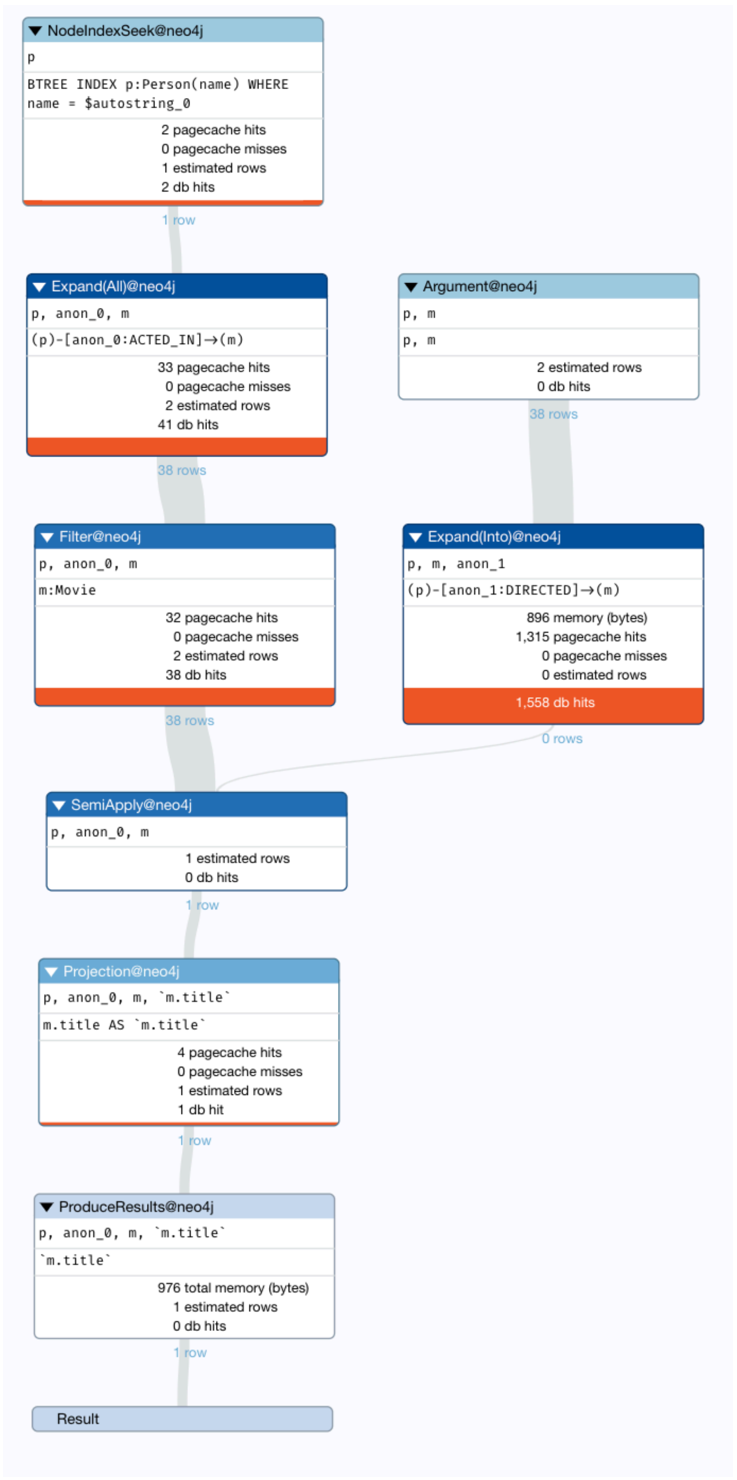


```
╒═════════════╤══════════════════════════════╤══════════════╕
│"p.name"     │"labels(p)"                   │"m.title"     │
╞═════════════╪══════════════════════════════╪══════════════╡
│"Tom Hanks"│["Actor","Director","Person"]│"Larry Crowne"│
└─────────────┴──────────────────────────────┴──────────────┘
```

Profiling queries

- Use the PROFILE keyword to show the *total number of rows* retrieved from the graph in the query.

```
PROFILE MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE  p.name = 'Tom Hanks'
AND exists {(p)-[:DIRECTED]->(m)}
RETURN m.title
```



```
NodeIndexSeek@neo4j
p
BTREE INDEX p:Person(name) WHERE
name = $autostring_0
2 pagecache hits
0 pagecache misses
1 estimated rows
2 db hits
```
1 row

```
Expand(All)@neo4j
p, anon_0, m
(p)-[anon_0:ACTED_IN]->(m)
33 pagecache hits
0 pagecache misses
2 estimated rows
41 db hits
```
38 rows

```
Argument@neo4j
p, m
p, m
2 estimated rows
0 db hits
```
38 rows

```
Filter@neo4j
p, anon_0, m
m:Movie
32 pagecache hits
0 pagecache misses
2 estimated rows
38 db hits
```
38 rows

```
Expand(Into)@neo4j
p, m, anon_1
(p)-[anon_1:DIRECTED]->(m)
896 memory (bytes)
1,315 pagecache hits
0 pagecache misses
0 estimated rows
1,558 db hits
```
0 rows

```
SemiApply@neo4j
p, anon_0, m
1 estimated rows
0 db hits
```
1 row

```
Projection@neo4j
p, anon_0, m, `m.title`
m.title AS `m.title`
4 pagecache hits
0 pagecache misses
1 estimated rows
1 db hit
```
1 row

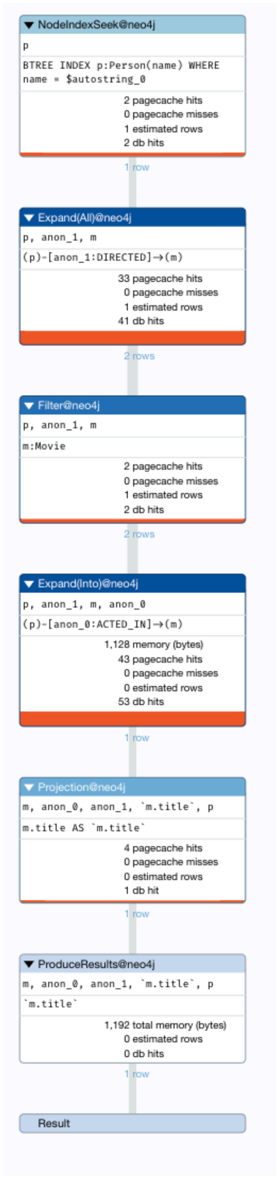
```
ProduceResults@neo4j
p, anon_0, m, `m.title`
`m.title`
976 total memory (bytes)
1 estimated rows
0 db hits
```
1 row

Result

The graph shows initial row is retrieved, but then 38 rows are retrieved for each Movie that Tom Hanks acted in.

There is a better way to do the same query:

```
PROFILE MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(p)
WHERE  p.name = 'Tom Hanks'
RETURN  m.title
```



```
▼ NodeIndexSeek@neo4j
p
BTREE INDEX p:Person(name) WHERE
name = $autostring_0
            2 pagecache hits
            0 pagecache misses
            1 estimated rows
            2 db hits
                1 row
▼ Expand(All)@neo4j
p, anon_1, m
(p)-[anon_1:DIRECTED]→(m)
           33 pagecache hits
            0 pagecache misses
            1 estimated rows
           41 db hits
                2 rows
▼ Filter@neo4j
p, anon_1, m
m:Movie
            2 pagecache hits
            0 pagecache misses
            1 estimated rows
            2 db hits
                2 rows
▼ Expand(Into)@neo4j
p, anon_1, m, anon_0
(p)-[anon_0:ACTED_IN]→(m)
        1,128 memory (bytes)
           43 pagecache hits
            0 pagecache misses
            0 estimated rows
           53 db hits
                1 row
▼ Projection@neo4j
m, anon_0, anon_1, `m.title`, p
m.title AS `m.title`
            4 pagecache hits
            0 pagecache misses
            0 estimated rows
            1 db hit
                1 row
▼ ProduceResults@neo4j
m, anon_0, anon_1, `m.title`, p
`m.title`
        1,192 total memory (bytes)
            0 estimated rows
            0 db hits
                1 row
Result
```

This traversal is very efficient because the graph engine can **take the [internal] relationship cardinalities into account.** It retrieves one row then two rows; much less data than the first query

- Differences between `EXPLAIN` and `PROFILE`:
    - `EXPLAIN` provides estimates of the query steps
    - `PROFILE` provides the exact steps and number of rows retrieved for the query.
        - When query tuning,execute the query at least twice. The first `PROFILE` of a query will always be more *expensive* than subsequent queries.
            - First execution: generation of the execution plan
            - Second execution: cached.

Finding non-patterns

- Use `NOT exists { }` to exclude patterns

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE  p.name = 'Tom Hanks'
AND NOT exists {(p)-[:DIRECTED]->(m)}
RETURN  m.title
```

Exclude the `:DIRECTED` relationships to movies for Tom Hank.

6. Multiple MATCH Clauses

Using multiple patterns in the `MATCH` clause

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie),
      (m)<-[:DIRECTED]-(d:Person)
WHERE m.year > 2000
RETURN a.name, m.title, d.name
```

- In general, using a single `MATCH` clause will **perform better** than multiple `MATCH` clauses. Because *relationship uniquness* is enforced so there are fewer relationships traversed.

Using a single pattern

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)<-[:DIRECTED]-(d:Person)
WHERE m.year > 2000
RETURN a.name, m.title, d.name
```

Optionally matching rows

```
MATCH (m:Movie) WHERE m.title = "Kiss Me Deadly"
MATCH (m)-[:IN_GENRE]->(g:Genre)<-[:IN_GENRE]-(rec:Movie)
OPTIONAL MATCH (m)<-[:ACTED_IN]-(a:Actor)-[:ACTED_IN]->(rec)
RETURN rec.title, a.name
```

- Use **nulls** for *missing parts* of the pattern.
- Could be considered the Cypher equivalent of the **outer join** in SQL.

7. Ordering Returned Results

- `ORDER BY` property `DESC`, the default ordering is ascending
- Specify `ORDER BY` in the `RETURN` clause
- Eliminating null values returned: use `IS NOT NULL`, `IS NULL` in `WHERE` clause to declare.
- No limit to the number of properties you can order by.
- Ordering multiple results separate them by comma. The precedures will first follow the first stated rule, then the others in queues.

8. Limiting or Counting Results Returned

- Use `LIMIT` at the end of queries
- Add a `SKIP` and `LIMIT` keyword to control what page of results are returned.

```
MATCH (p:Person)
WHERE p.born.year = 1980
RETURN  p.name as name,
p.born AS birthDate
ORDER BY p.born SKIP 40 LIMIT 10
```

In this query, we return 10 rows representing page 5, where each page contains 10 rows.

- Use `DISTINCT` eliminates duplicates for rows, property, nodes in `RETURN` clause. If we do not declare specificly, by default, it eliminates duplicated nodes.

9. Map Projections to Return Data

Map Projections

```
MATCH (p:Person)
WHERE p.name CONTAINS "Thomas"
RETURN p { .* } AS person
ORDER BY p.name ASC
```

- Return data is without the internal node information(table, text,diagram...), that is, only property values with a JSON-style object for a node.

```
╒══════════════════════════════════════════════════════════════════════════════════════════════════╕
│"person"                                                                                          │
╞══════════════════════════════════════════════════════════════════════════════════════════════════╡
│{"tmdbId":"1198886","imdbId":"0359735","name":" Thomas Haneke","url":"https://themoviedb.org/person/1198886"} │
├──────────────────────────────────────────────────────────────────────────────────────────────────┤
│{"tmdbId":"16789","imdbId":"2049833","name":"Aaran Thomas","poster":"https://image.tmdb.org/t/p/w440_and_h660_fa│
│ce/sSsQKswXWHZXrpnN3hkq1ptXZ7f.jpg","url":"https://themoviedb.org/person/16789"}                  │
├──────────────────────────────────────────────────────────────────────────────────────────────────┤
```

- Customize what properties you return in the objects by specifing the properties name in { }.

```
RETURN p { .name, .born } AS person
```

- Adding information to the objects returned that are not part of the data in the graph.

```
MATCH (m:Movie)<-[:DIRECTED]-(d:Director)
WHERE d.name = 'Woody Allen'
RETURN m {.*, favorite: true} AS movie
```

```
╒══════════════════════════════════════════════════════════════════════════════════════════════════╕
│"movie"                                                                                           │
╞══════════════════════════════════════════════════════════════════════════════════════════════════╡
│{"languages":["English"],"year":1986,"imdbId":"0091167","runtime":103,"imdbRating":8.0,"movieId":"6993","countri│
│es":["USA"],"imdbVotes":51032,"title":"Hannah and Her Sisters","url":"https://themoviedb.org/movie/5143","revenu│
│e":40084041,"tmdbId":"5143","plot":"Between two Thanksgivings two years apart, Hannah's husband falls in love wi│
│th her sister Lee, while her hypochondriac ex-husband rekindles his relationship with her sister Holly.","favori│
│te":true,"poster":"https://image.tmdb.org/t/p/w440_and_h660_face/gARgIRb2QFRFVrsziwWE389u1pK.jpg","released":"19│
│86-03-14","budget":6400000}                                                                       │
├──────────────────────────────────────────────────────────────────────────────────────────────────┤
│{"languages":["English"],"year":2001,"imdbId":"0256524","runtime":103,"imdbRating":6.8,"movieId":"4733","countri│
│es":["USA"," Germany"],"imdbVotes":30941,"title":"Curse of the Jade Scorpion, The","url":"https://themoviedb.org│
│/movie/2779","tmdbId":"2779","plot":"An insurance investigator and an efficency expert who hate each other are b│
│oth hypnotized by a crooked hypnotist with a jade scorpion into stealing jewels.","favorite":true,"poster":"http│
│s://image.tmdb.org/t/p/w440_and_h660_face/6u2x0GaSOMGOJ69ygnSRpPt54Qe.jpg","released":"2001-08-24"} │
├──────────────────────────────────────────────────────────────────────────────────────────────────┤
```

10. Changing Result Returned

Changing data returned

```
MATCH (m:Movie)<-[:ACTED_IN]-(p:Person)
WHERE m.title CONTAINS 'Toy Story' AND
p.died IS NULL
RETURN 'Movie: ' + m.title AS movie,
p.name AS actor,
p.born AS dob,
date().year - p.born.year AS ageThisYear
```

```
╒═══════════════════════════════╤═════════════════╤════════════╤═══════════════╕
│"movie"                        │"actor"          │"dob"       │"ageThisYear"  │
╞═══════════════════════════════╪═════════════════╪════════════╪═══════════════╡
│"Movie: Toy Story"             │"Tim Allen"      │"1953-06-13"│69             │
├───────────────────────────────┼─────────────────┼────────────┼───────────────┤
│"Movie: Toy Story"             │"Tom Hanks"      │"1956-07-09"│66             │
├───────────────────────────────┼─────────────────┼────────────┼───────────────┤
│"Movie: Toy Story 2"           │"Kelsey Grammer" │"1955-02-21"│67             │
├───────────────────────────────┼─────────────────┼────────────┼───────────────┤
```

Performing string or numeric operations:

- Add data to each line by calculating the actor's age.
- Concatenate string data returned.

Conditionally changing data returned

Use `CASE WHEN ELSE END` clause that specify to compute the data returned which may be different from what is in the graph.

```
MATCH (m:Movie)<-[:ACTED_IN]-(p:Person)
WHERE p.name = 'Henry Fonda'
RETURN m.title AS movie,
CASE
WHEN m.year < 1940 THEN 'oldies'
WHEN 1940 <= m.year < 1950 THEN 'forties'
WHEN 1950 <= m.year < 1960 THEN 'fifties'
WHEN 1960 <= m.year < 1970 THEN 'sixties'
WHEN 1970 <= m.year < 1980 THEN 'seventies'
WHEN 1980 <= m.year < 1990 THEN 'eighties'
WHEN 1990 <= m.year < 2000 THEN 'nineties'
ELSE  'two-thousands'
END
AS timeFrame
```

```
╒══════════════════════════════════════════════════════════╤════════════╕
│"movie"                                                   │"timeFrame" │
╞══════════════════════════════════════════════════════════╪════════════╡
│"Tin Star, The"                                           │"fifties"   │
├──────────────────────────────────────────────────────────┼────────────┤
│"Wrong Man, The"                                          │"fifties"   │
├──────────────────────────────────────────────────────────┼────────────┤
│"Grapes of Wrath, The"                                    │"forties"   │
├──────────────────────────────────────────────────────────┼────────────┤
│"Spencer's Mountain"                                      │"sixties"   │
├──────────────────────────────────────────────────────────┼────────────┤
│"Mister Roberts"                                          │"fifties"   │
├──────────────────────────────────────────────────────────┼────────────┤
```

11. Aggregating Data

Counts

Use `count()` function to perform a count of nodes, relationships, paths, rows during query processing.

- Eager aggregation: In Cypher, need not specify a grouping key. All non-aggregated result columns become grouping keys.
- If count(*) include null values.
- If count(n) return n non-null values.

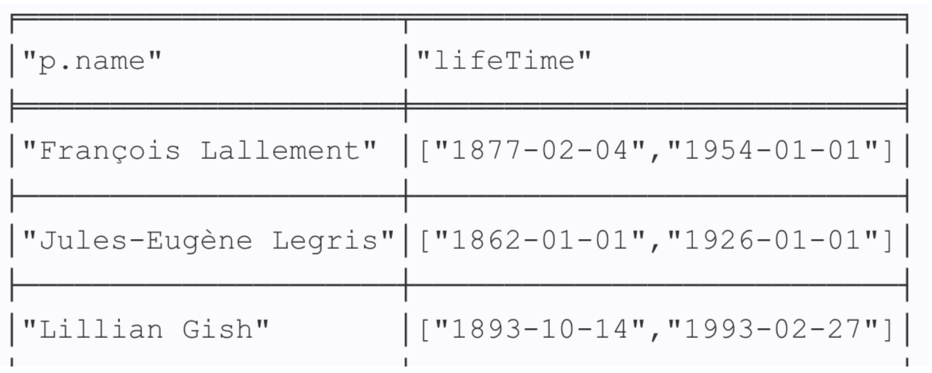
```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name = 'Tom Hanks'
RETURN a.name AS actorName,
count(*) AS numMovies
```

```
╒═══════════╤═══════════╕
│"actorName"│"numMovies"│
╞═══════════╪═══════════╡
│"Tom Hanks"│38         │
└───────────┴───────────┘
```

Lists

Return a list by specifying the square brackets

```
MATCH (p:Person)
RETURN p.name, [p.born, p.died] AS lifeTime
LIMIT 10
```

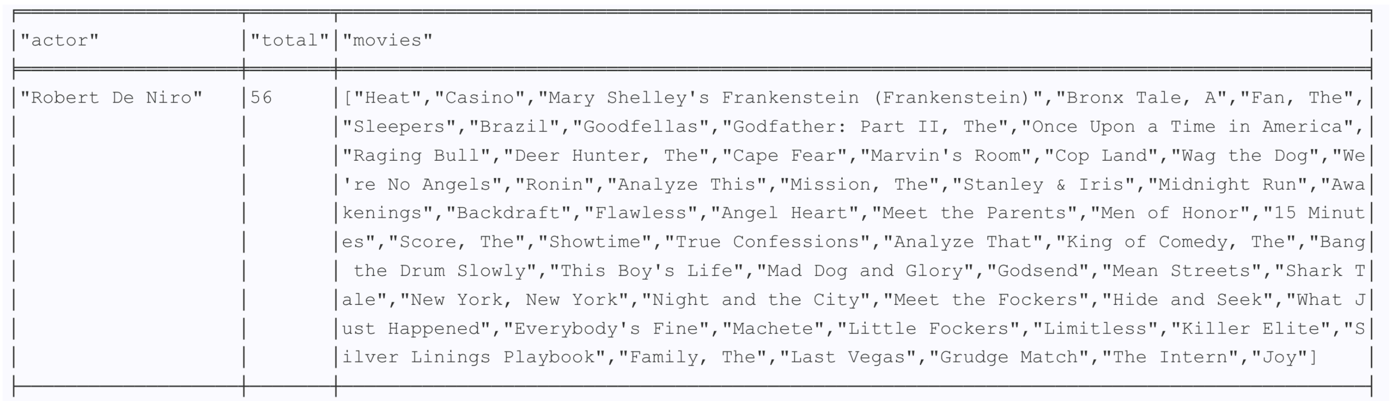


```
╒═════════════════════╤═════════════════════════════╕
│"p.name"             │"lifeTime"                   │
╞═════════════════════╪═════════════════════════════╡
│"François Lallement" │["1877-02-04","1954-01-01"]  │
├─────────────────────┼─────────────────────────────┤
│"Jules-Eugène Legris"│["1862-01-01","1926-01-01"]  │
├─────────────────────┼─────────────────────────────┤
│"Lillian Gish"       │["1893-10-14","1993-02-27"]  │
│                     │                             │
```

Using `collect()` to create a list

`collect()` aggregate values into a list. The value can be any expression.

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
RETURN a.name AS actor,
count(*) AS total,
collect(m.title) AS movies
ORDER BY total DESC LIMIT 10
```



```
╒═══════════════════╤═══════╤═══════════════════════════════════════════════════════════════════════════════════╕
│"actor"            │"total"│"movies"                                                                           │
╞═══════════════════╪═══════╪═══════════════════════════════════════════════════════════════════════════════════╡
│"Robert De Niro"   │56     │["Heat","Casino","Mary Shelley's Frankenstein (Frankenstein)","Bronx Tale, A","Fan, The",│
│                   │       │"Sleepers","Brazil","Goodfellas","Godfather: Part II, The","Once Upon a Time in America",│
│                   │       │"Raging Bull","Deer Hunter, The","Cape Fear","Marvin's Room","Cop Land","Wag the Dog","We│
│                   │       │'re No Angels","Ronin","Analyze This","Mission, The","Stanley & Iris","Midnight Run","Awa│
│                   │       │kenings","Backdraft","Flawless","Angel Heart","Meet the Parents","Men of Honor","15 Minut│
│                   │       │es","Score, The","Showtime","True Confessions","Analyze That","King of Comedy, The","Bang│
│                   │       │ the Drum Slowly","This Boy's Life","Mad Dog and Glory","Godsend","Mean Streets","Shark T│
│                   │       │ale","New York, New York","Night and the City","Meet the Fockers","Hide and Seek","What J│
│                   │       │ust Happened","Everybody's Fine","Machete","Little Fockers","Limitless","Killer Elite","S│
│                   │       │ilver Linings Playbook","Family, The","Last Vegas","Grudge Match","The Intern","Joy"]    │
├───────────────────┼───────┼───────────────────────────────────────────────────────────────────────────────────┤
```

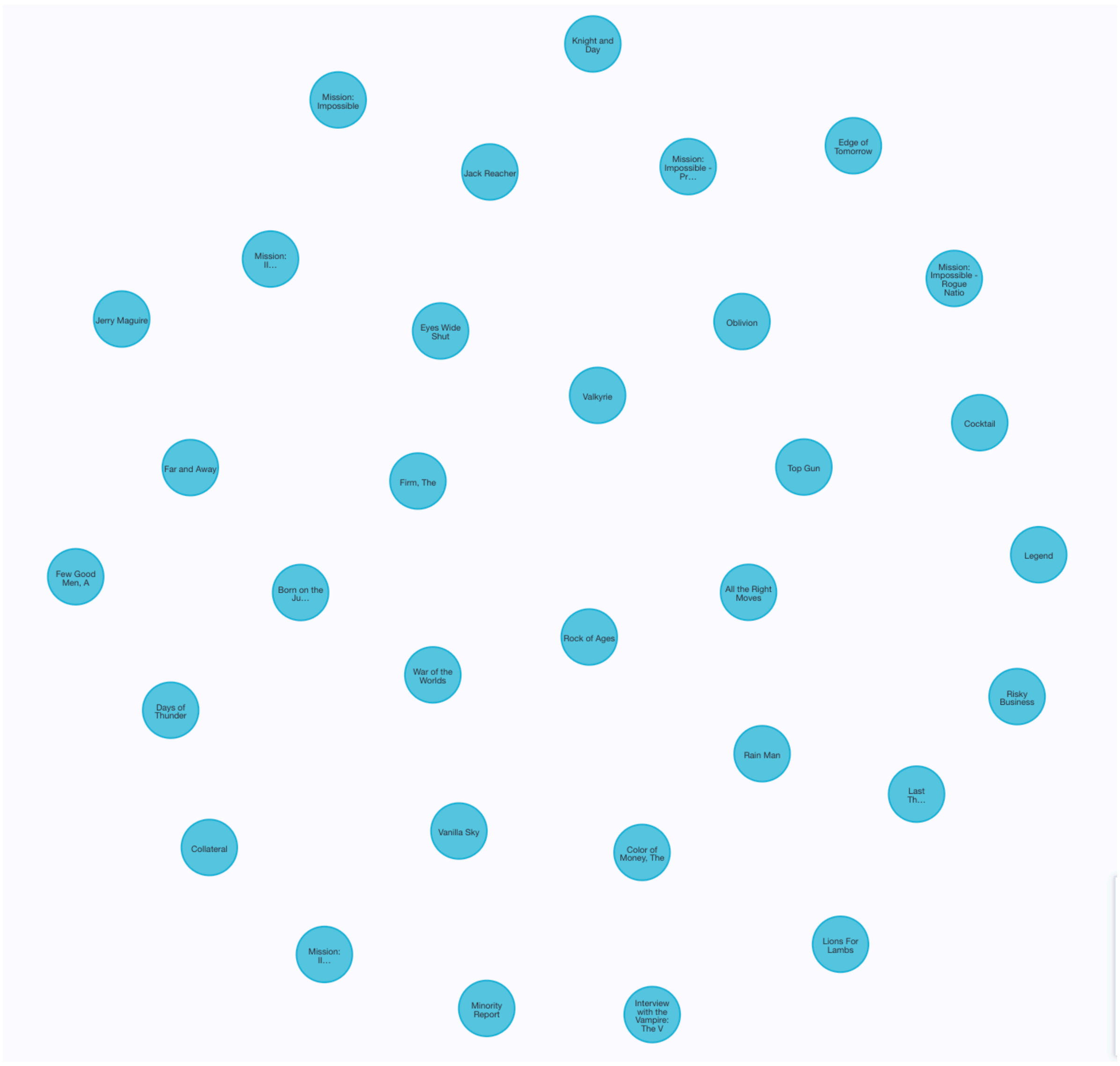
Eliminating duplication in lists

Use `DISTINCT`

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE m.year = 1920
RETURN  collect( DISTINCT m.title) AS movies,
collect( a.name) AS actors
```

Collecting nodes

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)
WHERE p.name ='Tom Cruise'
RETURN collect(m) AS tomCruiseMovies
```



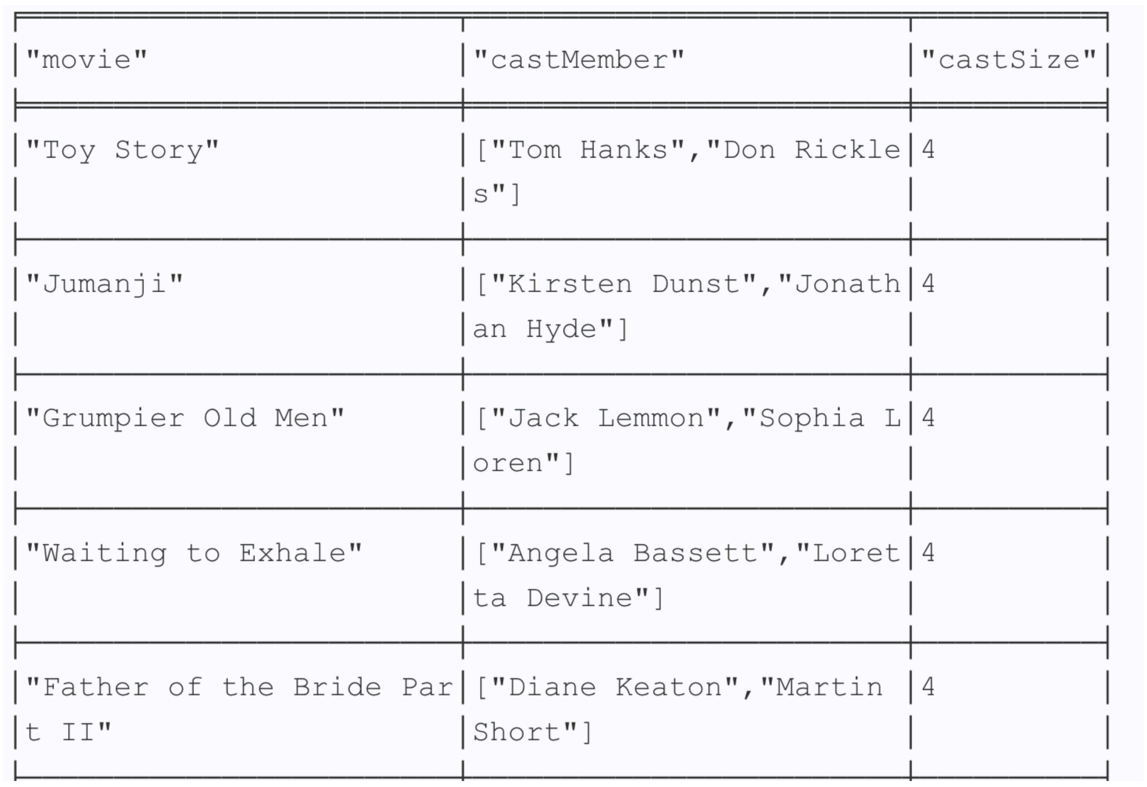
This query returns a list of all Movie nodes for Tom Cruise.

Accessing elements of a list

Access particular elements of the list using the `[index-value]` notation where a list *begins with index 0*. Return a slice of a collection

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
RETURN m.title AS movie,
collect(a.name)[2..] AS castMember,
size(collect(a.name)) as castSize
```

This query returns **the second to the end** of the list names of actors.



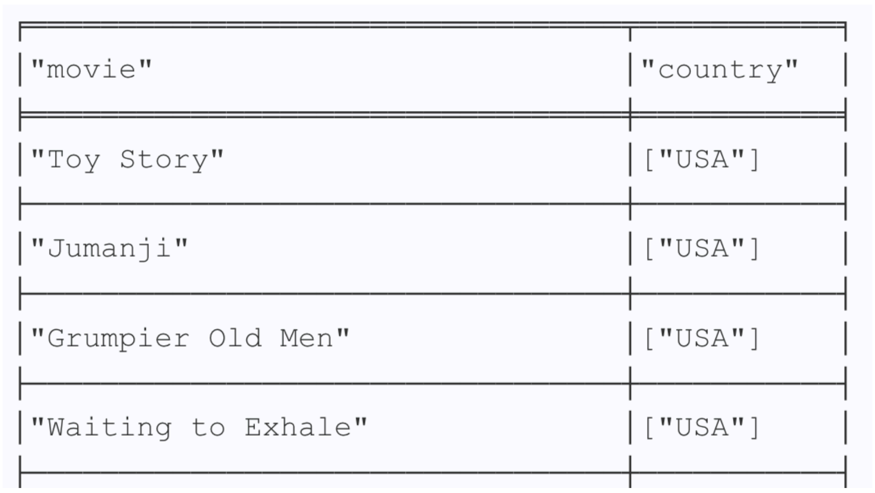
```
╒═════════════════════════╤═════════════════════════╤══════════╕
│"movie"                  │"castMember"             │"castSize"│
╞═════════════════════════╪═════════════════════════╪══════════╡
│"Toy Story"              │["Tom Hanks","Don Rickle │4         │
│                         │s"]                      │          │
├─────────────────────────┼─────────────────────────┼──────────┤
│"Jumanji"                │["Kirsten Dunst","Jonath │4         │
│                         │an Hyde"]                │          │
├─────────────────────────┼─────────────────────────┼──────────┤
│"Grumpier Old Men"       │["Jack Lemmon","Sophia L │4         │
│                         │oren"]                   │          │
├─────────────────────────┼─────────────────────────┼──────────┤
│"Waiting to Exhale"      │["Angela Bassett","Loret │4         │
│                         │ta Devine"]              │          │
├─────────────────────────┼─────────────────────────┼──────────┤
│"Father of the Bride Par │["Diane Keaton","Martin  │4         │
│t II"                    │Short"]                  │          │
└─────────────────────────┴─────────────────────────┴──────────┘
```

- Differences between `COUNT`() & `SIZE`()
  - `size()` function returns the number of **elements in a list**.
  - use `count()` to count the number of rows, more efficient than `size()`
  - Use `profile()` to check

List comprehension

```
MATCH (m:Movie)
RETURN m.title as movie,
[x IN m.countries WHERE x = 'USA' OR x = 'Germany']
AS country LIMIT 500
```

Create a list by evaluating an expression that tests for list inclusion.

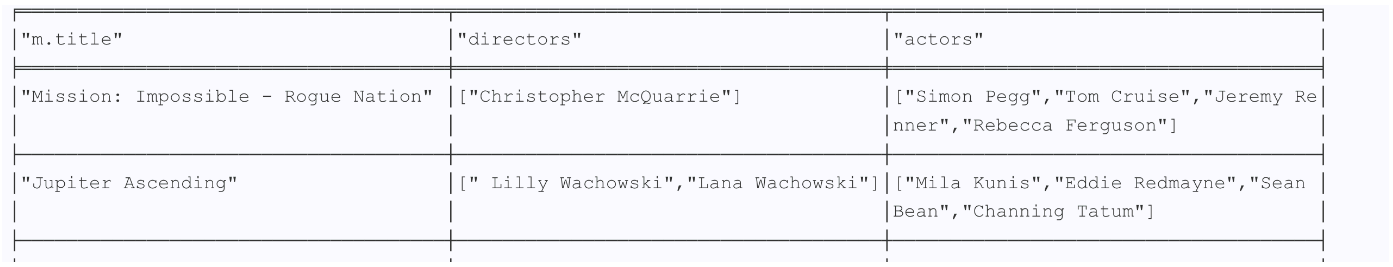


```
╒══════════════════════════════════╤════════════╕
│"movie"                           │"country"   │
╞══════════════════════════════════╪════════════╡
│"Toy Story"                       │["USA"]     │
├──────────────────────────────────┼────────────┤
│"Jumanji"                         │["USA"]     │
├──────────────────────────────────┼────────────┤
│"Grumpier Old Men"                │["USA"]     │
├──────────────────────────────────┼────────────┤
│"Waiting to Exhale"               │["USA"]     │
├──────────────────────────────────┼────────────┤
```

Pattern comprehension

- Create lists without changing the cardinality of the query.
- `[<pattern> | value]`
    - specify the list with the square braces to include the pattern followed by the pipe character to
    - specify what value will be placed in the list from the pattern.

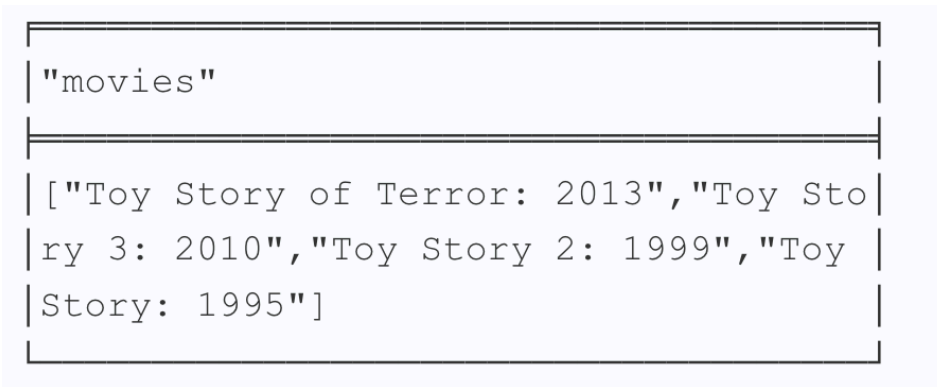
```
MATCH (m:Movie)
WHERE m.year = 2015
RETURN m.title,
[(dir:Person)-[:DIRECTED]->(m) | dir.name] AS directors,
[(actor:Person)-[:ACTED_IN]->(m) | actor.name] AS actors
```



```
╒════════════════════════════════════╤════════════════════════════════════╤════════════════════════════════════╕
│"m.title"                           │"directors"                         │"actors"                            │
╞════════════════════════════════════╪════════════════════════════════════╪════════════════════════════════════╡
│"Mission: Impossible - Rogue Nation"│["Christopher McQuarrie"]           │["Simon Pegg","Tom Cruise","Jeremy Re│
│                                    │                                    │nner","Rebecca Ferguson"]           │
├────────────────────────────────────┼────────────────────────────────────┼────────────────────────────────────┤
│"Jupiter Ascending"                 │[" Lilly Wachowski","Lana Wachowski"]│["Mila Kunis","Eddie Redmayne","Sean │
│                                    │                                    │Bean","Channing Tatum"]             │
├────────────────────────────────────┼────────────────────────────────────┼────────────────────────────────────┤
```

Create a list *specify a filter* for the pattern:

```
MATCH (a:Person {name: 'Tom Hanks'})
RETURN [(a)-->(b:Movie)
WHERE b.title CONTAINS "Toy" | b.title + ": " + b.year]
AS movies
```



```
╒═══════════════════════════════════════╕
│"movies"                               │
╞═══════════════════════════════════════╡
│["Toy Story of Terror: 2013","Toy Sto│
│ry 3: 2010","Toy Story 2: 1999","Toy │
│Story: 1995"]                          │
└───────────────────────────────────────┘
```

Working with maps

- Maps: list of key/value pairs. A node or relationship can have a property that is a map.
- Return the value for one of its elements:
  - `RETURN {maps}['key'] AS variableName`
  - `RETURN {maps}.key AS variableName`
- Return a list of keys of a map :
  - `RETURN {maps} AS variableName`
- A node in the graph returned in Neo4j Browser is a map.

12. Working with Dates and Times

```
RETURN date(), datetime(), time()
```

Create some date/time properties:

```
MERGE (x:Test {id: 1})
SET x.date = date(),
    x.datetime = datetime(),
    x.time = time()
RETURN x
```

```
╒══════════════════════════════════════╕
│"x"                                   │
╞══════════════════════════════════════╡
│{"date":"2022-10-30","datetime":"2022│
│-10-30T12:26:38.154000000Z","id":1,"t│
│ime":"12:26:38.154000000Z"}           │
└──────────────────────────────────────┘
```

Show the types of the properties:

```
CALL apoc.meta.nodeTypeProperties()
```

```
╒═══════════════════╤═══════════════════╤════════════════╤═════════════════╤═══════════╤═══════════════════╤═══════════════════╕
│"nodeType"         │"nodeLabels"       │"propertyName"  │"propertyTypes"  │"mandatory"│"propertyObservati│"totalObservations│
│                   │                   │                │                 │           │ons"               │"                  │
╞═══════════════════╪═══════════════════╪════════════════╪═════════════════╪═══════════╪═══════════════════╪═══════════════════╡
│":`Movie`"         │["Movie"]          │"languages"     │["StringArray"]│false      │1006               │1013               │
├───────────────────┼───────────────────┼────────────────┼─────────────────┼───────────┼───────────────────┼───────────────────┤
│":`Movie`"         │["Movie"]          │"year"          │["Long"]         │false      │1010               │1013               │
├───────────────────┼───────────────────┼────────────────┼─────────────────┼───────────┼───────────────────┼───────────────────┤
│":`Movie`"         │["Movie"]          │"imdbId"        │["String"]       │false      │1013               │1013               │
└───────────────────┴───────────────────┴────────────────┴─────────────────┴───────────┴───────────────────┴───────────────────┘
```

Extracting components of a date or datetime

```
MATCH (x:Test {id: 1})
RETURN x.date.day, x.date.year,
x.datetime.year, x.datetime.hour,
x.datetime.minute
```

```
╒════════════╤═════════════╤═════════════════╤═════════════════╤══════════════════╕
│"x.date.day"│"x.date.year"│"x.datetime.year"│"x.datetime.hour"│"x.datetime.minute│
│            │             │                 │                 │"                 │
╞════════════╪═════════════╪═════════════════╪═════════════════╪══════════════════╡
│30          │2022         │2022             │12               │26                │
└────────────┴─────────────┴─────────────────┴─────────────────┴──────────────────┘
```

Setting date values

Use a string to set a value for a date

```
SET x.date1 = date('2022-01-01'),
    x.date2 = date('2022-01-15')
RETURN x
```

```
╒════════════════════════════════════════════╕
│"x"                                         │
╞════════════════════════════════════════════╡
│{"date":"2022-10-30","datetime":"2022-10-30T12:2│
│6:38.154000000Z","date2":"2022-01-15","id":1,"ti│
│me":"12:26:38.154000000Z","date1":"2022-01-01"} │
└────────────────────────────────────────────┘
```

Setting datetime values

```
MATCH (x:Test {id: 1})
SET x.datetime1 = datetime('2022-01-04T10:05:20'),
    x.datetime2 = datetime('2022-04-09T18:33:05')
RETURN x
```

```
╒══════════════════════════════════════════════════════════════════════════════╕
│"x"                                                                           │
╞══════════════════════════════════════════════════════════════════════════════╡
│{"date":"2022-10-30","datetime":"2022-10-30T12:26:38.154000000Z","datetim│
│e2":"2022-04-09T18:33:05Z","datetime1":"2022-01-04T10:05:20Z","date2":"20│
│22-01-15","id":1,"time":"12:26:38.154000000Z","date1":"2022-01-01"}      │
└──────────────────────────────────────────────────────────────────────────────┘
```

Working with durations

Determine the difference between two date/datetime values or to add or subtract a duration to a value.

```
MATCH (x:Test {id: 1})
RETURN duration.between(x.date1,x.date2)
```

```
MATCH (x:Test {id: 1})
RETURN duration.inDays(x.datetime1,x.datetime2).days
```

```
MATCH (x:Test {id: 1})
RETURN x.date1 + duration({months: 6})
```