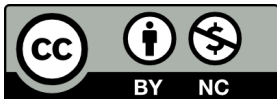


Business 4720 - Class 4

Querying Graph and Document Databases

Joerg Evermann

Faculty of Business Administration
Memorial University of Newfoundland
`jevermann@mun.ca`



Unless otherwise indicated, the copyright in this material is owned by Joerg Evermann. This material is licensed to you under the [Creative Commons by-attribution non-commercial license \(CC BY-NC 4.0\)](#)

This Class

What You Will Learn:

- ▶ Querying Property Graphs with Neo4J and Cypher

Use Cases

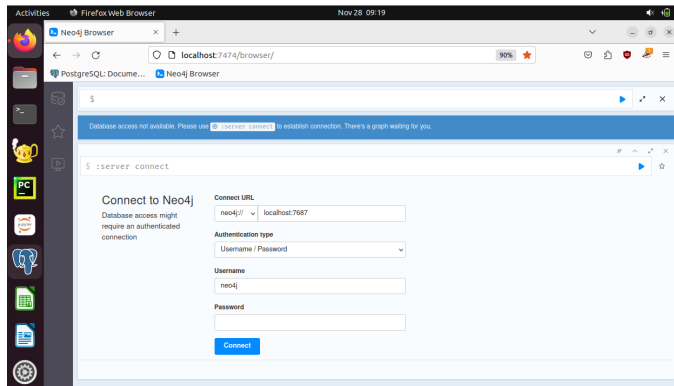
- ▶ Fraud detection
- ▶ IT infrastructure monitoring
- ▶ Recommender engines
- ▶ Master data management
- ▶ Social media and social network analytics
- ▶ Supply chain management
- ▶ Financial services
- ▶ Life sciences

Graph Query Languages

- ▶ SPARQL SPARQL Protocol and RDF Query Language (W3C, 2008, 2013)
- ▶ Gremlin (Apache Tinkerpop 2009, 2023)
- ▶ Cypher (Neo4J 2011, openCypher 2015)
- ▶ GraphQL (Facebook, 2015, 2021)
- ▶ GQL (ISO/IEC, forthcoming 2023)

Graph Analytics with Neo4J and Cypher

- ▶ Neo4J Community Edition installed in course virtual machine
- ▶ Browse to `http://localhost:7474`
- ▶ Username **neo4j** password **busi4720**



Neo4J Property Graphs

Nodes

- ▶ May be labelled with zero, one or more labels
- ▶ Labels group nodes into sets
- ▶ Can have key–value pairs ("properties")

Relationships

- ▶ Directed, named connection between two nodes
- ▶ Typed with one relationship type
- ▶ Can have key–value pairs ("properties")
- ▶ Can be navigated in any direction

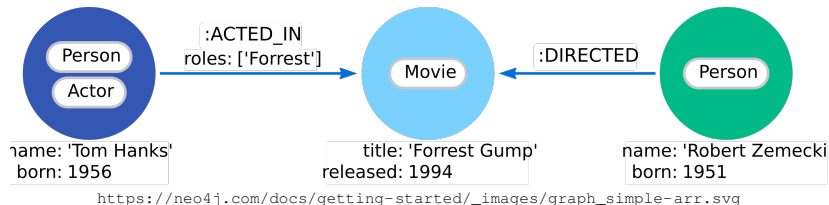
Path

- ▶ Sequence of alternating nodes and relationships
- ▶ Starts and ends at a node

Basic Ideas

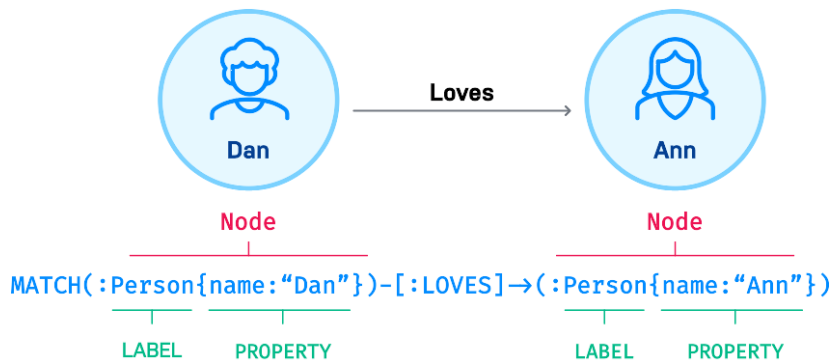
- ▶ Declarative (styled after SQL)
- ▶ Pattern matching (styled after SPARQL)
- ▶ Cypher query has multiple clauses ("query pipelines")
- ▶ Read and write in a single Cypher statement
- ▶ Queries must return data

Cyper and Graph Concepts



```
CREATE (:Person:Actor)
  -[:ACTED_IN {roles: ['Forrest']}]>
  (:Movie {title: 'Forrest Gump', released: 1994})
  <-[:DIRECTED]-
  (:Person {name: 'Robert Zemeckis', born: 1951})
```


Cypher and Graph Concepts



https://neo4j.com/docs/getting-started/_images/sample-cypher.svg

Graph Nodes

`(variable : Label)`

- Optional variable name, optional label

Relationships

`() - [variable : Label] - ()`

`() - [variable : Label] -> ()`

`() <- [variable : Label] - ()`

`() - - ()`

`() - -> ()`

`() <- - ()`

- Optional variable name, optional label
- Directionality matters for querying and must match that of the relationship as created

Node Properties

```
(v:L { propertyName:  propertyValue } )
```

Relationship Properties

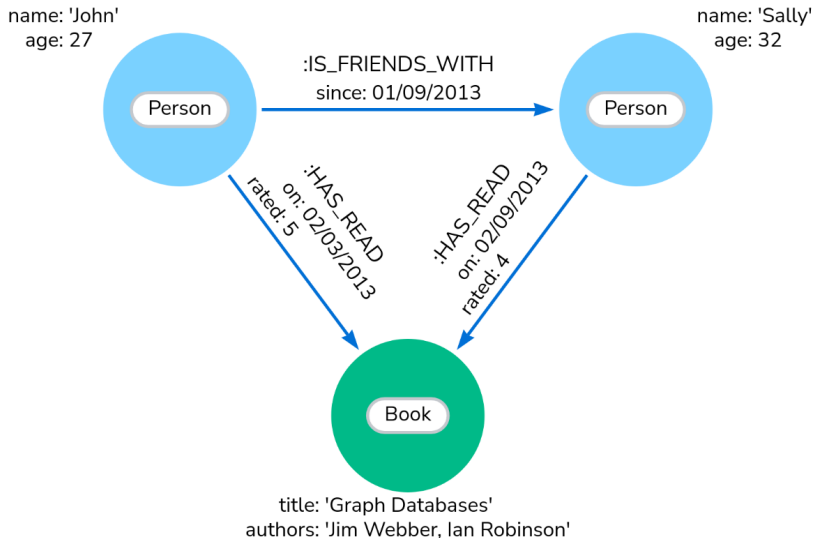
```
[r:L { propertyName:  propertyValue } ]
```

Pattern

```
(n1:L1 {p1:v1})-[r:L2 {p2:v2}]->(n2:L2 {p3:v3 } )
```

- ▶ Can be complex or simple
- ▶ Must be used with a keyword like MATCH for querying or like CREATE or MERGE for data definition

Defining Graphs in Cypher



https://neo4j.com/docs/getting-started/_images/modeling_johnsally_properties-arr

Defining Graphs in Cypher

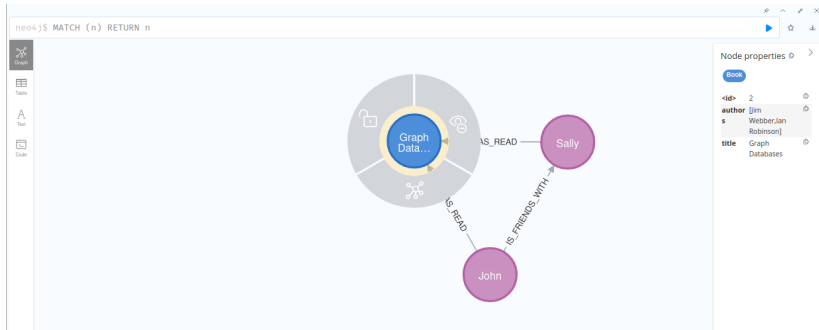
```
MERGE (j:Person {name: 'John'})
  ON CREATE SET j.age = 27
MERGE (s:Person {name: 'Sally'})
  ON CREATE SET s.age = 32
MERGE (b:Book {title: 'Graph Databases'})
  ON CREATE SET b.authors = ['Jim Webber', 'Ian Robinson']
MERGE (j)-[rel1:IS_FRIENDS_WITH]->(s)
  ON CREATE SET rel1.since = '01/09/2013'
MERGE (j)-[rel2:HAS_READ]->(b)
  ON CREATE SET rel2.on = '02/03/2013', rel2.rated = 5
MERGE (s)-[rel3:HAS_READ]->(b)
  ON CREATE SET rel3.on = '02/09/2013', rel3.rated = 4
```

MERGE ensures a node or relationship exists in the graph, creating it if necessary; CREATE creates a node or relationship

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

MATCH searches the graph for a pattern

Defining Graphs in Cypher



Define a graph in Cypher that represents the following statement:

You are completing the course BUSI 4720 in this semester with a final grade of 100. BUSI 4720 is part of the BCom program where it is offered in the 4th year. BUSI 4720 carries 3 credit hours of academic credit. It is a course on the topic of Business Analytics.

- 1 Identify nodes, relationships, and properties of nodes and relationships
- 2 Use CREATE or MERGE statements to create nodes first, then relationships
- 3 Use MATCH to verify your graph is correct.

To remove Persons and Books and relationships between them:

```
MATCH (:Person|Book)-[r]-(:Person|Book) DELETE r;  
MATCH (n:Person|Book) DELETE n;
```

Similar for other types of relationships or labels.

To remove **all** relationships and nodes use:

```
MATCH ()-[relationship]-() DELETE relationship;  
MATCH (node) DELETE node;
```


Property or Relationship?

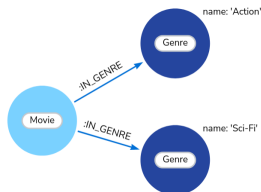


genre: 'Action', 'Sci-Fi'

```
//find the genres for  
// a particular movie  
MATCH (m:Movie {title:"The Matrix"})  
RETURN m.genre;  
  
//find which movies share genres  
MATCH (m1:Movie), (m2:Movie)  
WHERE any(x IN m1.genre  
           WHERE x IN m2.genre)  
AND m1 <> m2  
RETURN m1, m2;
```

https://neo4j.com/docs/getting-started/_images/modeling_genre_property-arr.svg

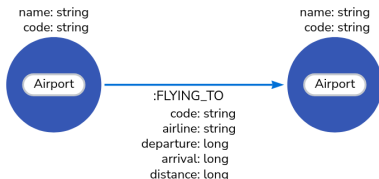
Property or Relationship?



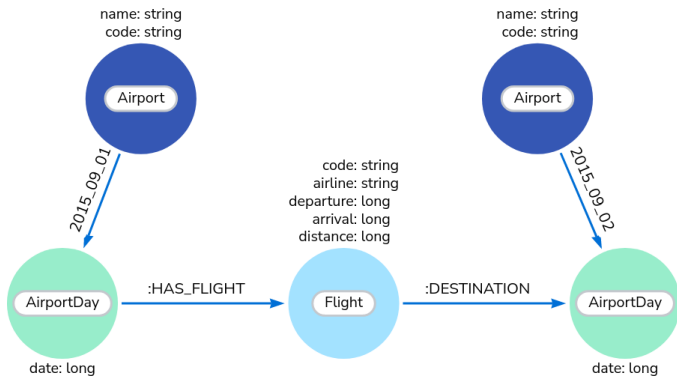
```
//find the genres for a  
//particular movie  
MATCH (m:Movie {title:"The Matrix"}),  
        (m)-[:IN_GENRE]->(g:Genre)  
RETURN g.name;  
  
//find which movies share genres  
MATCH (m1:Movie)-[:IN_GENRE]->(g:Genre),  
        (m2:Movie)-[:IN_GENRE]->(g)  
RETURN m1, m2, g
```

https://neo4j.com/docs/getting-started/_images/modeling_genre_node-arr.svg

Flexible Data Modeling

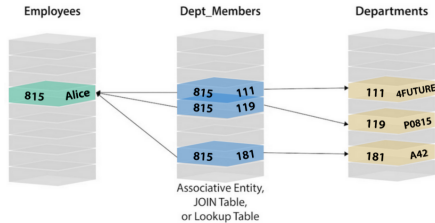


https://neo4j.com/docs/getting-started/_images/modeling_airport_flights-arr.svg

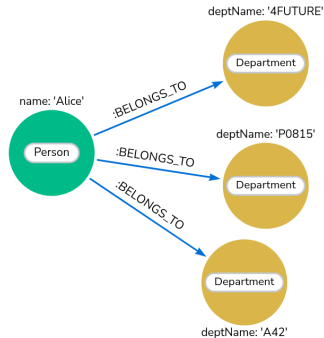


https://neo4j.com/docs/getting-started/_images/modeling_airport_flight_dates-arr.svg

Graph Data versus Relational Data



https://neo4j.com/docs/getting-started/_images/relational_model.svg



https://neo4j.com/docs/getting-started/_images/relational_graph_model-arr.svg

Conversion

- ▶ Tables to Node Labels
- ▶ Rows to Nodes
- ▶ Columns to Node Properties
- ▶ Foreign keys to Relationships
- ▶ Join tables to Relationships
- ▶ Remove NULL and default values

The Pagila Database for Neo4J

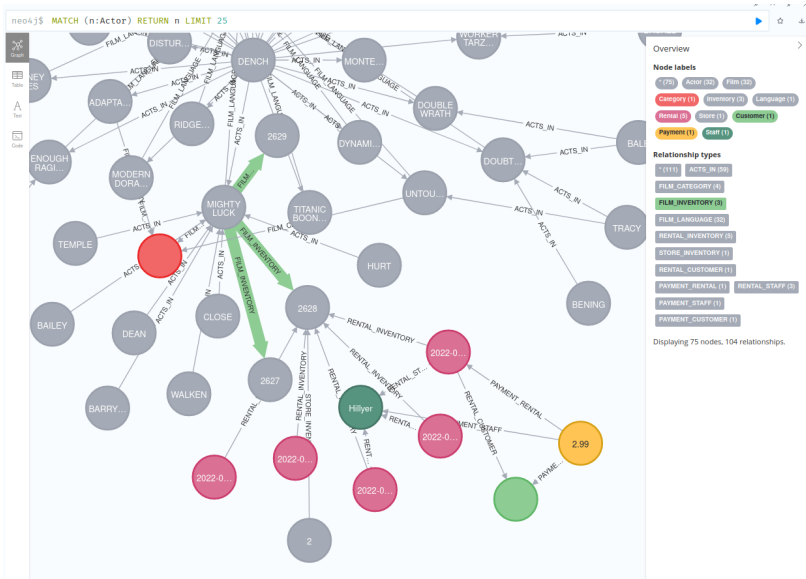
Import the Pagila Database (This may take ten or more minutes; already done in the course Virtual Machine):

```
CALL apoc.cypher.runFile(  
  'https://evermann.ca/busi4720/neo4j/import_pagila_from_csv.cypher')
```

Verify some data

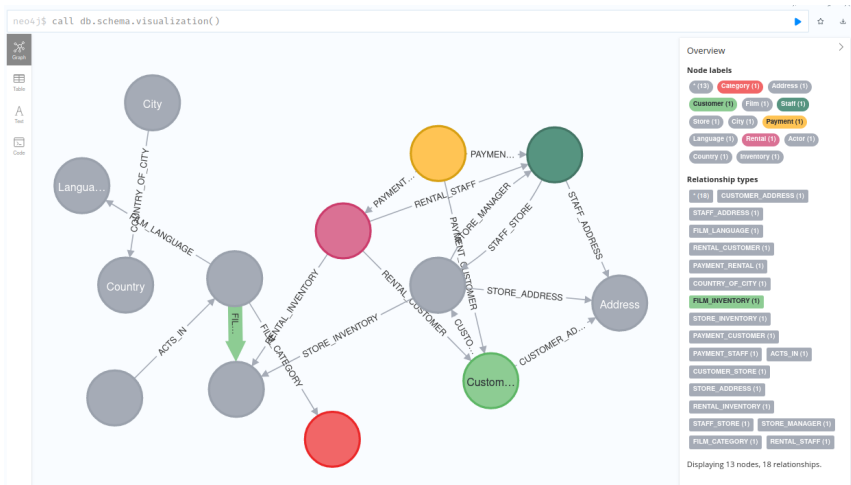
```
MATCH (n:Actor) RETURN n LIMIT 25
```

Explore the Pagila Graph



Explore the Pagila Schema

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



Cypher Query Examples

Find actors by last name, limit to 10

```
MATCH (a:Actor)
RETURN a.firstName, a.lastName
ORDER BY a.lastName DESC
LIMIT 10;
```

Find films whose title starts with a 'T' and that have a rental rate less than 3, sort by film title, limit to 10

```
MATCH (f:Film {rating: 'PG'})
WHERE (f.title STARTS WITH 'T') AND (f.rentalRate < 3)
RETURN f.title, f.rating, f.rentalRate
ORDER BY f.title ASC LIMIT 10;
```

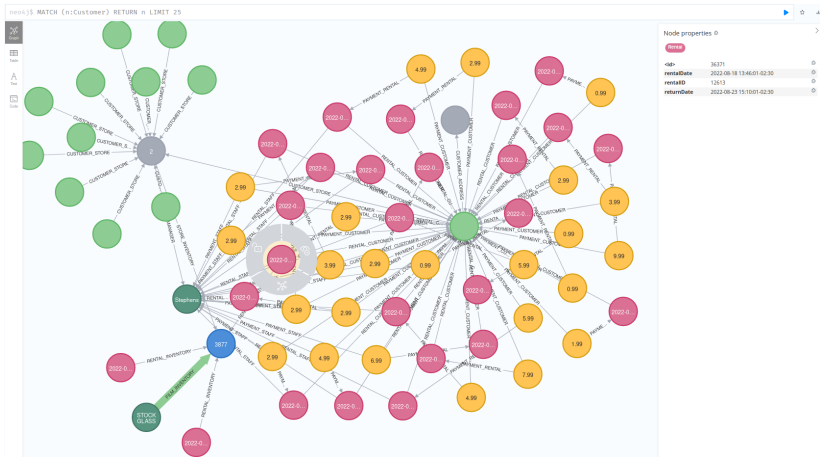
Find rental customers that live in India

```
MATCH (r:Rental)
  - [:RENTAL_CUSTOMER] -> (c)
  - [:CUSTOMER_ADDRESS] -> ()
  - [:ADDRESS_CITY] -> ()
  - [:COUNTRY_OF_CITY] -> (ct {country: 'India'})
RETURN c.firstName, c.lastName, r.rentalDate LIMIT 5
```

Find all customers that have rented a film with rating "PG"

- 1 Explore the graph visually in Neo4J browser, note the relationship types
- 2 Consider the path from customer to film via rental and inventory
- 3 Design a pattern that starts with a customer node and ends with a film node
- 4 Define an appropriate WHERE clause of property restrictions in node patterns

Hands-On Exercise



Aggregation: Find the mean and standard deviation of rental payments by country

```
MATCH (p:Payment)
  - [:PAYMENT_RENTAL] -> (r:Rental)
  - [:RENTAL_CUSTOMER] -> (c)
  - [:CUSTOMER_ADDRESS] -> ()
  - [:ADDRESS_CITY] -> ()
  - [:COUNTRY_OF_CITY] -> (ct)
WITH ct,
  avg(p.amount) AS amountMean,
  stDev(p.amount) AS amountSD
RETURN ct.country, amountMean, amountSD
ORDER BY amountMean DESC LIMIT 5
```

https:

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

Cypher Query Examples [cont'd]

Collection: Find the sets of last names of the movie cast, and the total number of actors

```
MATCH (a:Actor) - [:ACTS_IN] -> (f:Film)
RETURN f.title,
        collect(a.lastName) AS cast,
        count(*) AS numActors;
```

Collection: Find the set of film title by rental customer and the number of rentals

```
MATCH (f:Film) - [:FILM_INVENTORY]
        - () - [:RENTAL_INVENTORY]
        - (r:Rental) - [:RENTAL_CUSTOMER]
        -> (c:Customer)
RETURN c.lastName,
        collect(f.title) AS filmRentals,
        count(*) AS numRentals;
```

Collection: Find the set of rental customers for each film and the rental count

```
MATCH (f:Film) - [:FILM_INVENTORY]
      - () - [:RENTAL_INVENTORY]
      - (r:Rental) - [:RENTAL_CUSTOMER]
      -> (c:Customer)
RETURN DISTINCT f.title,
      collect (c.lastName+' '+left(c.firstName,1)+'.')
      AS custNames,
      count(*) as rentalCount
```

Sub-Query: Find the customers who rent films that are in inventory at multiple stores

```
MATCH (c:Customer) <-[:RENTAL_CUSTOMER]
      - (r:Rental) -[:RENTAL_INVENTORY]
      - () -[:FILM_INVENTORY]
      - (f:Film)
WITH c, count{
  MATCH (f) -[:FILM_INVENTORY]
        - () -[:STORE_INVENTORY]
        - (s:Store)
  RETURN DISTINCT s.storeID
} AS storeNum
where storeNum > 1
RETURN DISTINCT
  c.lastName
  + ' '
  + left(c.firstName,1)
  + '.' AS custName,
  storeNum
```


Christian Akroyd's co-actors

```
MATCH (a:Actor {firstName: 'CHRISTIAN',  
                lastName: 'AKROYD'})  
  - [:ACTS_IN]  
  - (f:Film)  
<- [:ACTS_IN]  
  - (coActors)  
RETURN coActors.firstName + ' ' +  
        coActors.lastName AS Name;
```

Quantified Relationships: Movies and actors up to 2 "hops" away from Christian Akroyd

```
MATCH (a:Actor {firstName: 'CHRISTIAN',  
                lastName: 'AKROYD'})  
  - [:ACTS_IN*1..2]  
  - (others)  
RETURN distinct others;
```

Built-In Function: The shortest path of an acts-in relationship between Christian Akroyd and Charlize Dench

```
MATCH path=shortestPath(  
  (a1: Actor {firstName: 'CHRISTIAN',  
                lastName: 'AKROYD'})  
  - [:ACTS_IN*]  
  - (a2: Actor {firstName: 'CHARLIZE',  
                lastName: 'DENCH'})  
RETURN path;
```

Pattern in WHERE clause, multiple MATCH patterns Find actors that Christian Akroyd hasn't yet worked with, but his co-actors have. Extend Christian Akroyd's co-actors, to find co-co-actors who haven't worked with him.

```
MATCH (a1:Actor {firstName:'CHRISTIAN',
                    lastName:'AKROYD'})
    - [:ACTS_IN] -> (m) <-[:ACTS_IN] -(coActors),
    (coActors)-[:ACTS_IN]->(m2)<-[:ACTS_IN]-(cocoActors)
WHERE NOT (a1)-[:ACTS_IN]->()<-[:ACTS_IN]-(cocoActors)
    AND a1 <> cocoActors
RETURN cocoActors.firstName+' '+
    cocoActors.lastName AS Recommended,
    count(*) AS Strength
ORDER BY Strength DESC
```

Find someone who can introduce Christian Akroyd to Susan Davis

```
MATCH (a1:Actor {firstName:'CHRISTIAN',  
                lastName:'AKROYD'})  
    -[:ACTS_IN]->(m) <-[:ACTS_IN]-(coActors),  
    (coActors)-[:ACTS_IN]->(m2)  
    <-[:ACTS_IN]-(a2:Actor {firstName:'SUSAN',  
                            lastName:'DAVIS'})  
RETURN a1, m, coActors, m2, a2
```

Further Information

Getting Started	https://neo4j.com/docs/getting-started/
Cypher Manual	https://neo4j.com/docs/cypher-manual
Graph Data Science	https://neo4j.com/docs/graph-data-science
APOC Library	https://neo4j.com/docs/apoc/current/
Use Cases	https://neo4j.com/use-cases/
Resources	https://neo4j.com/resources/

Hands-On Exercises

- 1 Are there two customers that have the same address?
- 2 Which customers have rented the same set of films?
- 3 Find all films with a single actor
- 4 Calculate the rental revenue per customer. Who are the top 5? Bottom 5?
- 5 Calculate the rental counts for each country of customer. Are there countries with no rentals?
- 6 Create a graph that represents a product hierarchy.