Class 7 Neo4j

Elements of Databases

Oct 29, 2021

Instapolls

- Check your GCP credit balance
- Check your Neo4j setup

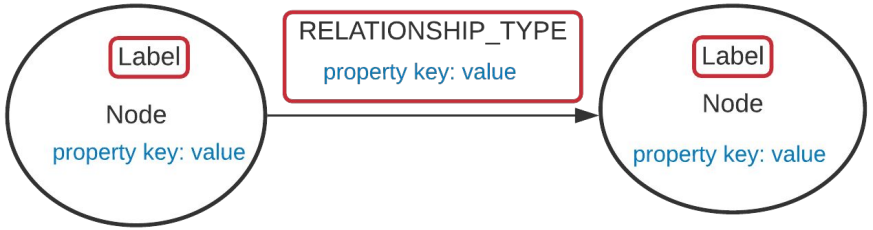
Announcements

- Review session next Friday at 4pm
- Exam 2 on Nov. 12th at 4pm

Exam rules:

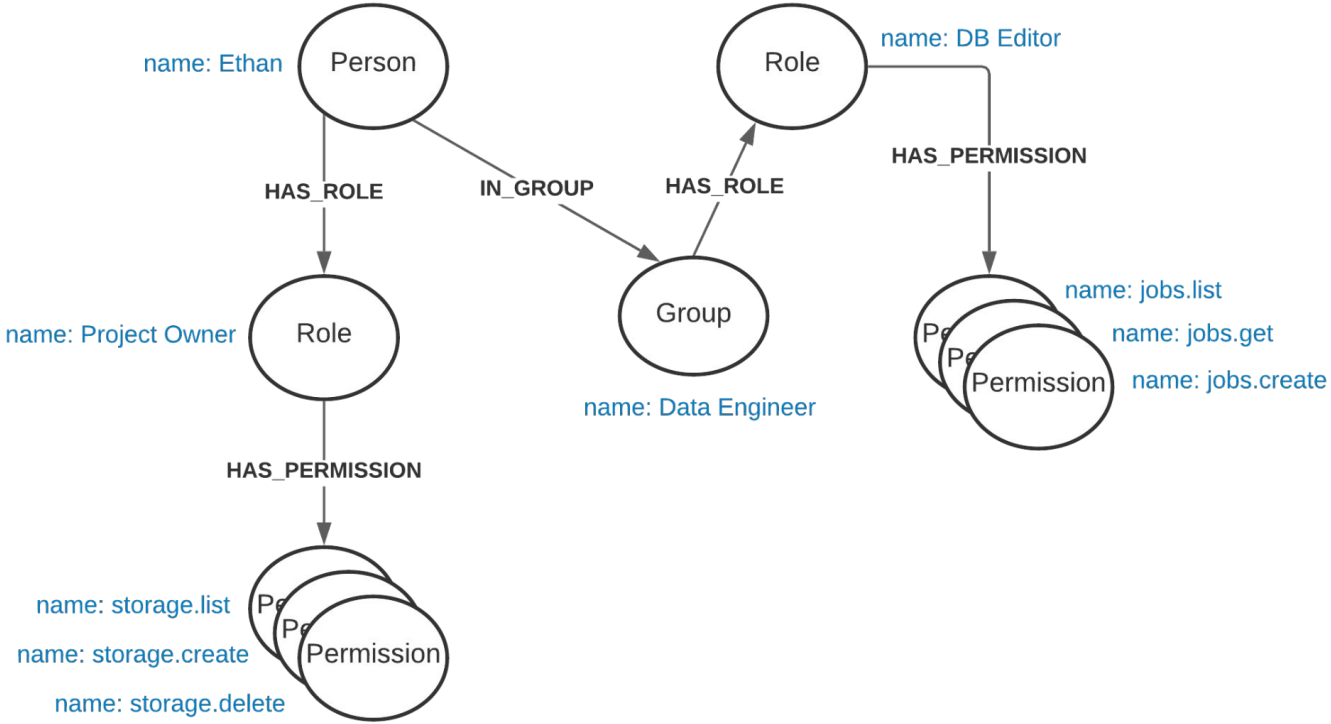
- 60-minute exam
- Open-note and open-book
- Piazza will be disabled during exam
- May **not** consult with any human in any form

Why Neo4j?



+ Labeled property graph data model
+ Flexible schema
+ Highly connected data
+ Declarative, SQL-inspired query language (Cypher)
+ Open-source, sponsored by Neo4j Inc.
+ Rich plugin and extension language (similar to Postgres)
+ ACID-compliant transactions
+ Distributed architecture for scaling reads
+ Visualization tools (Neo4j Browser, Bloom)
+ Optimized for graph traversals
+ Available as a cloud offering (Aura)
- Limited scalability for writes (no sharding)

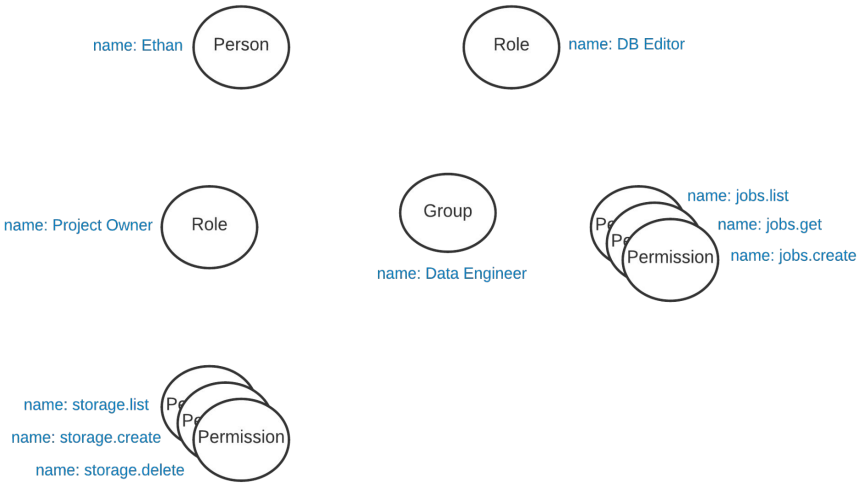
Neo4j’s Data Model Illustrated



“Hello World” example in Cypher

```
CREATE ();
CREATE (:Person);
CREATE (:Place);
CREATE (:Person {name: "Ethan"})-[:LIVES_IN]->(:Place {city: "Austin"});

MATCH(n) RETURN n;

MATCH ()-[r]->()
RETURN type(r), COUNT(r);

MATCH (p)-[r:LIVES_IN]->(c)
WHERE p.name = "Ethan"
AND c.city = "Austin"
RETURN p, r, c;
```

Creating the Nodes

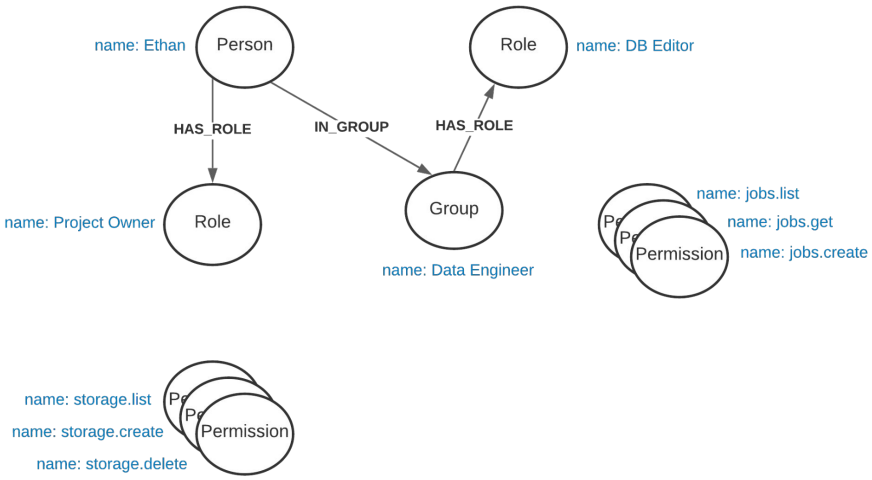



```
CREATE (:Permission {name: "jobs.list"});
CREATE (:Permission {name: "jobs.get"});
CREATE (:Permission {name: "jobs.create"});
```

```
CREATE (:Permission {name: "storage.list"});
CREATE (:Permission {name: "storage.create"});
CREATE (:Permission {name: "storage.delete"});
```

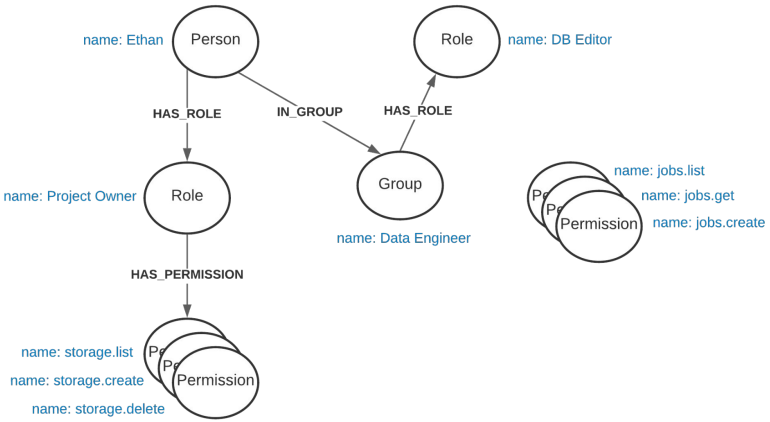
```
CREATE (:Person {name: "Ethan", email: "ethan@utexas.edu"});
CREATE (:Group {name: "Data Engineer", owner: "Alex"});
CREATE (:Role {name: "Project Owner", type: "GCP"});
CREATE (:Role {name: "DB Editor", type: "MySQL"});
```

Creating the Relationships



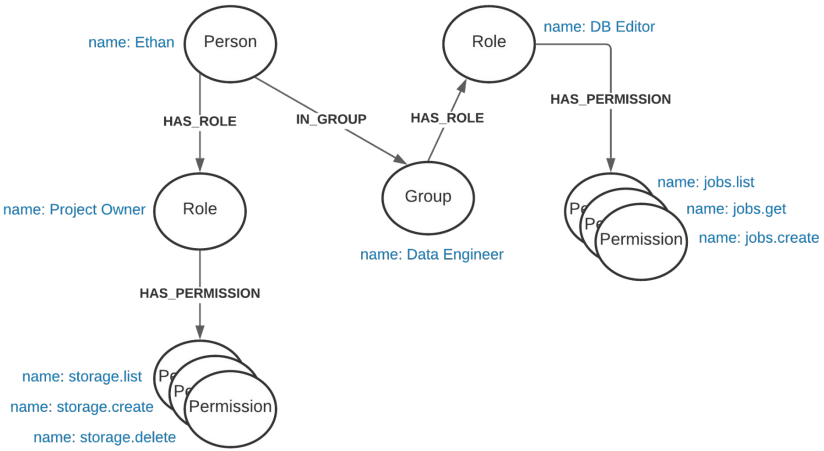

```
MATCH (p:Person {name: "Ethan"})
MATCH (r:Role {name: "Project Owner"})
CREATE (p)-[:HAS_ROLE]->(r);

MATCH (p:Person {name: "Ethan"})
MATCH (g:Group {name: "Data Engineer"})
CREATE (p)-[:IN_GROUP]->(g);

MATCH (g:Group {name: "Data Engineer"})
MATCH (r:Role {name: "DB Editor"})
CREATE (g)-[:HAS_ROLE]->(r);

MATCH (p)-[h]->(r) RETURN p, h, r;

MATCH (p:Person)-[h]->(r:Role)
WHERE r.name = "Project Owner"
RETURN p, h, r;
```

Creating the Relationships (cont.)



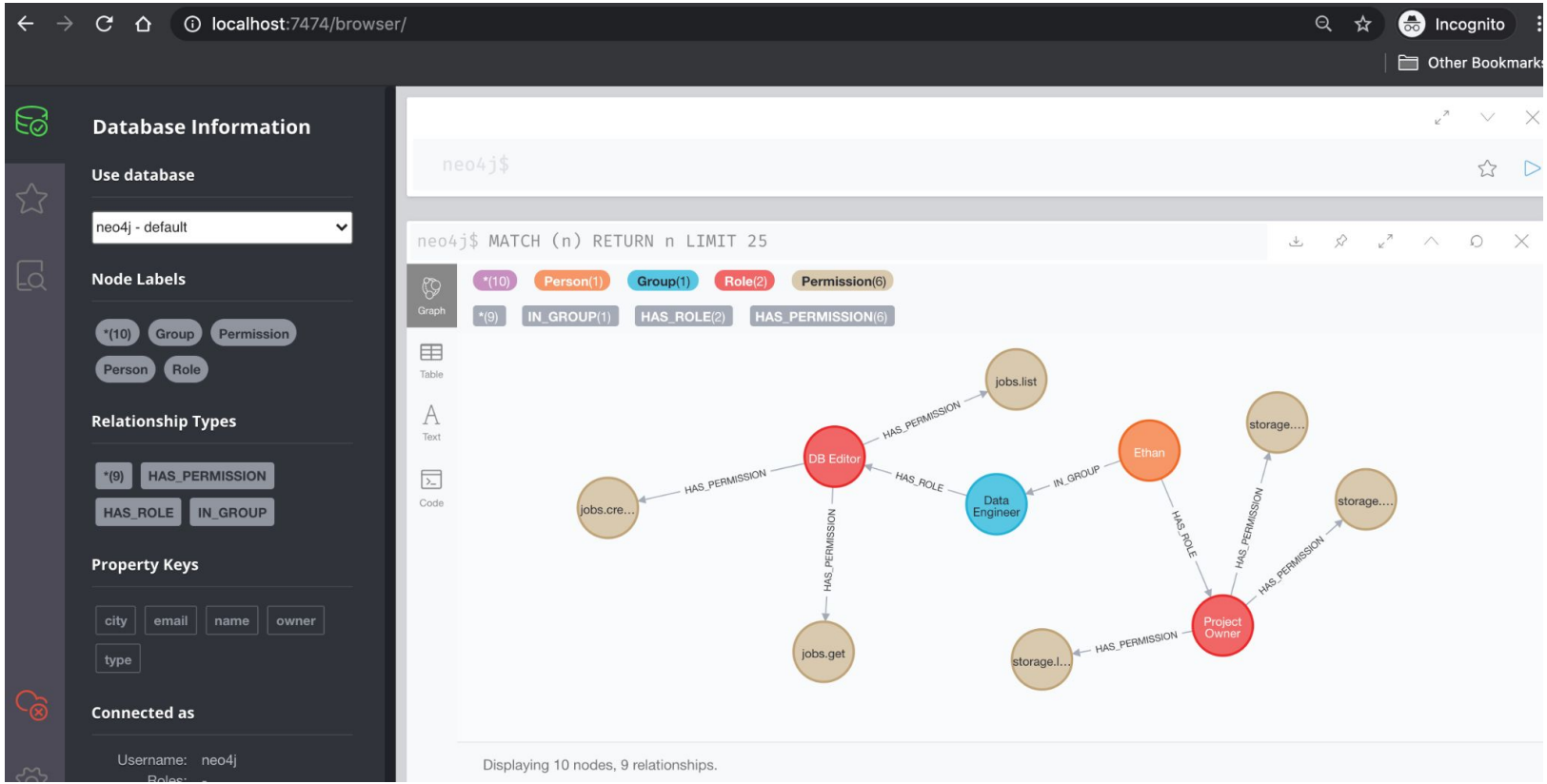

```
MATCH (r:Role {name: "Project Owner"})
MATCH (p:Permission {name: "storage.list"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role {name: "Project Owner"})
MATCH (p:Permission {name: "storage.create"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role {name: "Project Owner"})
MATCH (p:Permission {name: "storage.delete"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role)-[h]->(p)
WHERE r.name = "Project Owner"
RETURN r, h, p;
```

Creating the Relationships (cont.)




```
MATCH (r:Role {name: "DB Editor"})
MATCH (p:Permission {name: "jobs.list"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role {name: "DB Editor"})
MATCH (p:Permission {name: "jobs.get"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role {name: "DB Editor"})
MATCH (p:Permission {name: "jobs.create"})
CREATE (r)-[:HAS_PERMISSION]->(p);

MATCH (r:Role)-[h]->(p)
WHERE r.name = "DB Editor"
RETURN r, h, p;
```

Visualizing the Graph



Counting Nodes and Relationships

```
MATCH (n:Person)
RETURN count(n);
```

```
MATCH (n)
RETURN distinct labels(n), count(n);
```

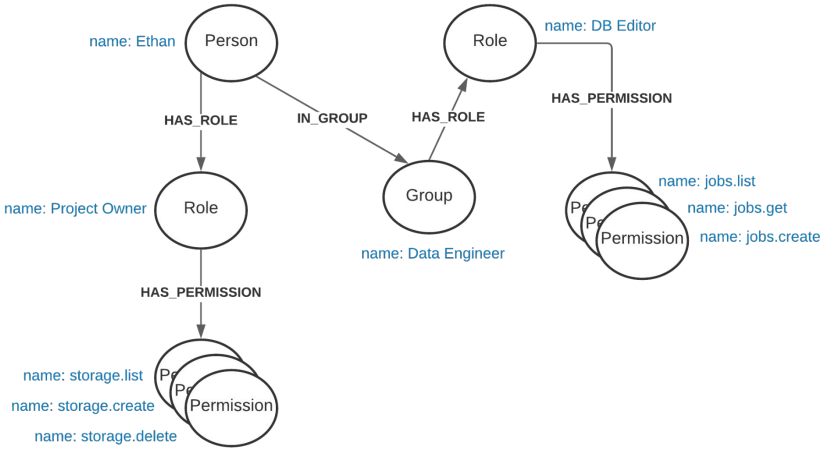
```
+--------------------------+
| labels(n)      | count(n) |
+--------------------------+
| ["Person"]     | 1        |
| ["Group"]      | 1        |
| ["Role"]       | 2        |
| ["Permission"] | 6        |
+--------------------------+
```

```
MATCH ()-[r:HAS_ROLE]->()
RETURN count(r);
```

```
MATCH ()-[r]->()
RETURN type(r), count(r);
```

```
+----------------------------+
| type(r)           | count(r) |
+----------------------------+
| "IN_GROUP"        | 1        |
| "HAS_ROLE"        | 2        |
| "HAS_PERMISSION"  | 6        |
+----------------------------+
```

Querying the Graph




```
MATCH (p:Person)-[r*]->(m:Permission)
WHERE p.name = "Ethan"
RETURN r, m.name
ORDER BY m;
```

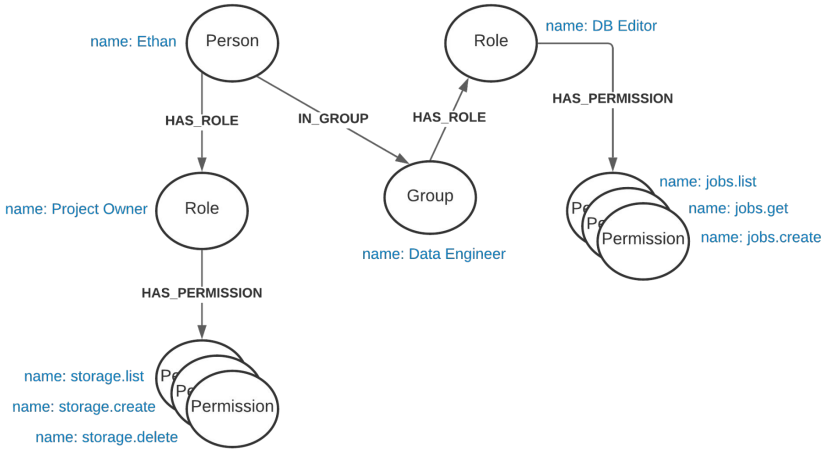
```
+-----------------------------------------------------------------+
| r                                                  | m.name     |
+-----------------------------------------------------------------+
| [[:IN_GROUP], [:HAS_ROLE], [:HAS_PERMISSION]] | "jobs.list"     |
| [[:IN_GROUP], [:HAS_ROLE], [:HAS_PERMISSION]] | "jobs.get"      |
| [[:IN_GROUP], [:HAS_ROLE], [:HAS_PERMISSION]] | "jobs.create"   |
| [[:HAS_ROLE], [:HAS_PERMISSION]]              | "storage.list"  |
| [[:HAS_ROLE], [:HAS_PERMISSION]]              | "storage.create"|
| [[:HAS_ROLE], [:HAS_PERMISSION]]              | "storage.delete"|
+-----------------------------------------------------------------+
```

If Ethan had *many more* permissions, we would add a `LIMIT` clause to the end of the query.
If Ethan had *duplicate* permissions, we would use `DISTINCT m` in the `RETURN` clause.

Querying the Graph

```
MATCH (p:Person)-[r*1]->(m:Permission)
WHERE p.name = "Ethan"
RETURN r, m.name
ORDER BY m;
```

```
+------------+
| r | m.name |
+------------+
+------------+
```




```
MATCH (p:Person)-[r*1..2]->(m:Permission)
WHERE p.name = "Ethan"
RETURN r, m.name
ORDER BY m;
```

```
+-----------------------------------------------------+
| r                                   | m.name        |
+-----------------------------------------------------+
| [[:HAS_ROLE], [:HAS_PERMISSION]] | "storage.list"   |
| [[:HAS_ROLE], [:HAS_PERMISSION]] | "storage.create" |
| [[:HAS_ROLE], [:HAS_PERMISSION]] | "storage.delete" |
+-----------------------------------------------------+
```

Updating Nodes

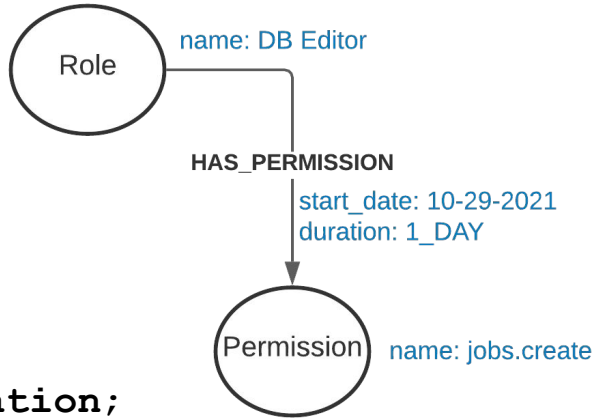
```
MATCH (n:Person {name: "Ethan"})
SET n.dob = "10-19-2000",
    n.occupation = "Student"
RETURN n.name, n.dob, n.occupation;
```

```
+-------------------------------------+
| n.name  | n.dob        | n.occupation |
+-------------------------------------+
| "Ethan" | "10-19-2000" | "Student"    |
+-------------------------------------+
```

```
MATCH (n {name: "Ethan"})
SET n:Principal
RETURN n.name, labels(n) AS labels;
```

```
+-----------------------------------+
| n.name  | labels                  |
+-----------------------------------+
| "Ethan" | ["Person", "Principal"] |
+-----------------------------------+
```

Updating Relationships




```
MATCH (n:Role {name: "DB Editor"})
MATCH (p:Permission {name: "jobs.create"})
MERGE (n)-[r:HAS_PERMISSION]->(p)
ON MATCH SET r.start_date = "10-29-2021",
r.duration = "1_DAY"
RETURN n.name, type(r), r.start_date, r.duration;
```

```
+-----------------------------------------------------------+
| n.name      | type(r)          | r.start_date | r.duration |
+-----------------------------------------------------------+
| "DB Editor" | "HAS_PERMISSION" | "10-29-2021" | "1_DAY"    |
+-----------------------------------------------------------+
```

Deleting Nodes and Relationships

```
MATCH (p:Person)-[r]->()
DELETE r;

MATCH (p:Person)
DELETE p;


MATCH (n)
DETACH DELETE n;
```

```
neo4j@neo4j> MATCH (n)
             DETACH DELETE n;
0 rows available after 7 ms, consumed after another 0 ms
Deleted 10 nodes, Deleted 9 relationships
neo4j@neo4j>
```

Neo4j Code Lab

- Clone snippets repo
- Open neo4j notebook
- Create movie and actor graph
- Run some queries over graph

Practice Problem

Translate the following scenario into a Cypher query:

Which persons acted in their own movie?

Return the person's name, movie title, and role they played in the movie which they directed.

Order the results by person's name.

Project 7

http://www.cs.utexas.edu/~scohen/projects/Project7.pdf