

# CS 327E Class 7

Oct 16, 2020

# Announcements

- Review session for Test 2
- Test 2 details

## Exam rules:

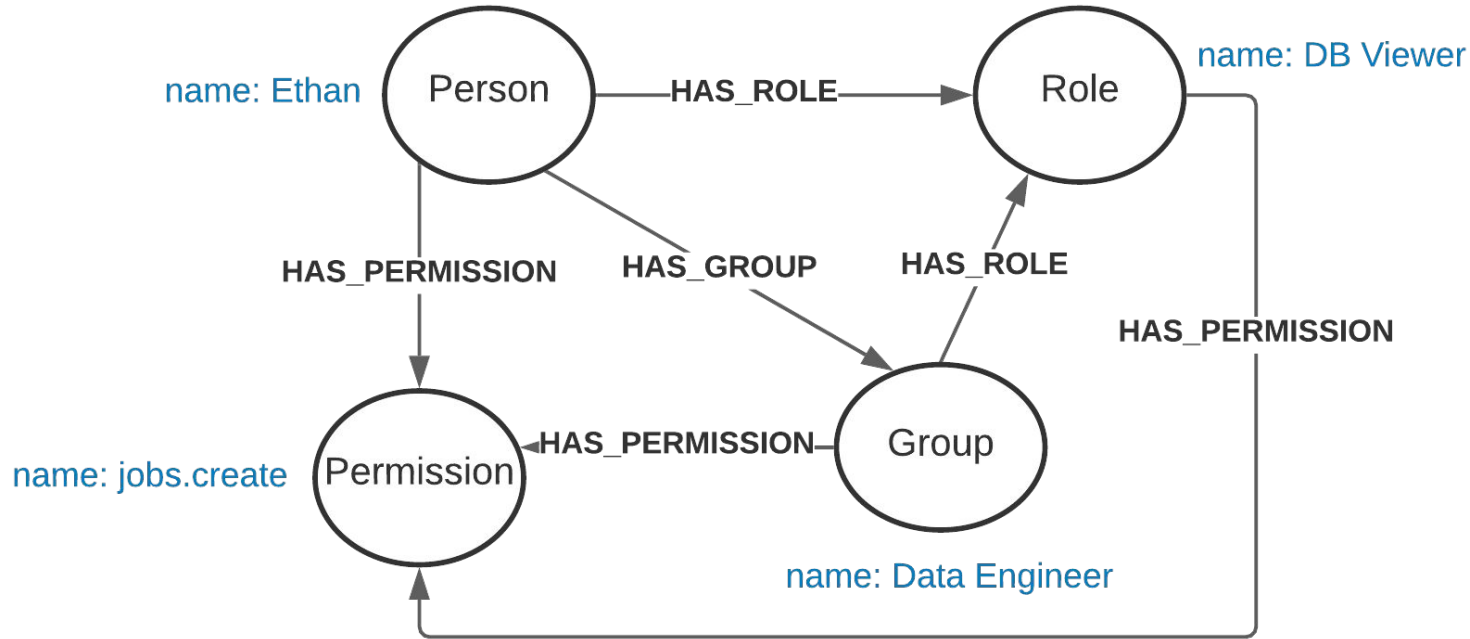
- Open-note and open-book
- Piazza will be disabled during exam
- May **not** consult with any human in any form

# Why Neo4j?

- Designed for storing and querying graphs
- Labeled property graph data model
- Optional schema
- Declarative, SQL-inspired query language (Cypher)
- Rich plugin and extension language (similar to Postgres)
- Open-source, sponsored by Neo4j Inc.
- ACID-compliant transactions
- Clustering option for scaling reads
- Visualization tools (Neo4j Browser, Bloom)
- Optimized for graph traversals



# Labeled Property Graph Model



# Creating Nodes

```
CREATE ();
```

```
CREATE (:Person);
```

```
CREATE (:Person {name: "Ethan", email: "ethan@utexas.edu"});
```

```
CREATE (:Role {name: "DB Viewer"});
```

```
CREATE (:Role {name: "DB Editor"});
```

```
CREATE (:Group {name: "Data Engineer"});
```

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

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

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

# Creating Nodes and Relationships

```
CREATE (:Person)-[:HAS_ROLE]->(:Role);
```

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

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

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

# Creating Relationships

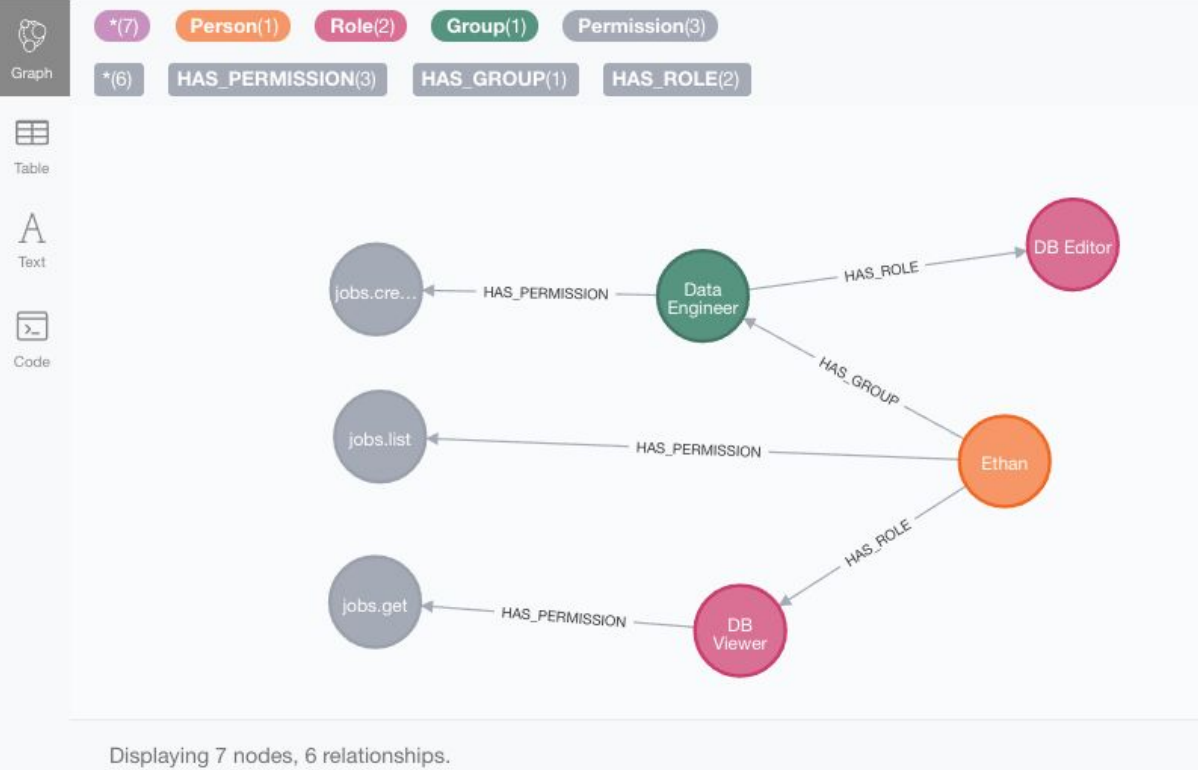
```
MATCH (p:Person {name: "Ethan"})
MATCH (m:Permission {name: "jobs.list"})
CREATE (p)-[:HAS_PERMISSION]->(m);
```

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

```
MATCH (g:Group {name: "Data Engineer"})
MATCH (m:Permission {name: "jobs.create"})
CREATE (g)-[:HAS_PERMISSION]->(m);
```

# Neo4j Browser

```
neo4j$ MATCH (n) RETURN n LIMIT 25
```





# Querying the Graph

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

| type(r)          | COUNT(r) |
|------------------|----------|
| "HAS_ROLE"       | 2        |
| "HAS_GROUP"      | 1        |
| "HAS_PERMISSION" | 3        |

```
MATCH (m:Permission)  
RETURN COUNT(m);
```

| COUNT(m) |
|----------|
| 3        |

```
MATCH ()-[r:HAS_PERMISSION]->()  
RETURN COUNT(r);
```

| COUNT(r) |
|----------|
| 3        |

# Querying the Graph

```
MATCH (p:Person {name: "Ethan"})-[r]->(m:Permission)
RETURN p, r, m;
```

```
+-----+
| p                               | r               | m               |
+-----+
| (:Person {name: "Ethan", email: "ethan@utexas.edu"}) | [:HAS_PERMISSION] | (:Permission {name: "jobs.list"}) |
+-----+
```

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

```
+-----+
| p                               | r               | m               |
+-----+
| (:Person {name: "Ethan", email: "ethan@utexas.edu"}) | [:HAS_PERMISSION] | (:Permission {name: "jobs.list"}) |
+-----+
```

# Querying the Graph

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

| p  | r                               | m                                   |
|--|---------------------------------|-------------------------------------|
| (:Person {name: "Ethan", email: "ethan@utexas.edu"}) | [:HAS_GROUP], [:HAS_PERMISSION] | (:Permission {name: "jobs.create"}) |
| (:Person {name: "Ethan", email: "ethan@utexas.edu"}) | [:HAS_ROLE], [:HAS_PERMISSION]  | (:Permission {name: "jobs.get"})    |
| (:Person {name: "Ethan", email: "ethan@utexas.edu"}) | [:HAS_PERMISSION]               | (:Permission {name: "jobs.list"})   |

3 rows available after 53 ms, consumed after another 1 ms

# Updating the Graph

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

```
MATCH (r:Role {name: "DB Editor"})
MATCH (p:Permission {name: "jobs.create"})
MERGE (r)-[rel:HAS_PERMISSION]->(p)
ON MATCH SET rel.name = "10-16-2020"
RETURN type(rel), rel.name;
```

```
+-----+
| type(rel)      | rel.name      |
+-----+
| "HAS_PERMISSION" | "10-16-2020" |
+-----+
```

```
1 row available after 1 ms, consumed after another 2 ms
Set 1 properties
```

# Querying the Graph

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

```
+-----+
| m      |
+-----+
| (:Permission {name: "jobs.create"}) |
| (:Permission {name: "jobs.create"}) |
| (:Permission {name: "jobs.get"})    |
| (:Permission {name: "jobs.list"})   |
+-----+
```

```
MATCH (p:Person {name: "Ethan"})-[r*]->(m:Permission)
RETURN DISTINCT m ORDER BY m.name;
```

```
+-----+
| m      |
+-----+
| (:Permission {name: "jobs.create"}) |
| (:Permission {name: "jobs.get"})    |
| (:Permission {name: "jobs.list"})   |
+-----+
```

# Deleting the Graph

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

```
MATCH (p:Person)  
DELETE p;
```

```
MATCH ()-[r]->(m:Permission)  
DELETE r;
```

```
MATCH (m:Permission)  
DELETE m;
```

```
MATCH (n)  
DETACH DELETE n;
```

# Set up Neo4j

<https://github.com/cs327e-fall2020/snippets/wiki/Neo4j-Setup-Guide>

# Practice Problem

Translate the following scenario into a Cypher query:

**Which persons directed a movie in which they also acted?**

**Return the person's name, movie title, and role they played in their own movie.**

**Order the results by person name.**



# Project 6

<http://www.cs.utexas.edu/~scohen/projects/Project6.pdf>