**Class Participation for Week 10**

**Tuesday**

**Learning objectives:**

1. Gain more understanding about graph DBMS and graph databases.
2. Be able to create nodes and edges with properties and values assigned in Neo4j graph database.
3. Compare and contrast Neo4j property graph data model with ER model, and relational data model.
4. Learn that the same data can be modeled using different data models.
5. Practice Extract-Transform-Load of data into a Neo4j database.
6. Practice Cypher queries.
7. Design Neo4j database from relational schemas.

To earn the full class participation credits, provide answers **for all the questions**. For Question 4, submit answers for Q4.a-d. There are no partial credits. Put the answers of all the questions below in a pdf file named <netid>WK10.pdf where <netid> is replaced by your IASTATE netid. The pdf file needs to include the screenshot from Question 1, ER diagram from Question 2, and relational schemas from Question 3.

**Instruction:**

This exercise asks you to create a social graph database that uses nodes to store data about people and their interests and edges to store friendships and other relationships among them. Note that the friendship is modeled by a directed edge. An edge from node A to node B means A thinks of B as his/her friend, but it says nothing about whether node B thinks of node A as a friend or not.

* Step 1: First, run the Cypher statements in CreateNodes.cypher to create different types of nodes. The instruction for running them is in the file itself. The files with the extension “cypher” is a text file that you can open with any text editor.
* Step 2: Run the Cypher statements in CreateEdges.cypher to create different types of edges. The instruction for running them is in the file itself.

1. Write one or more Cypher statement(s) to add a node to represent you into the social graph database and add an edge with the label FRIENDSHIP from the node with the name property value of “James” to the new node that represents you. **Take the screenshot** that shows the social graph with the new node and the new edge you add.

Chart, diagram

Description automatically generated

1. **Draw an ER diagram** that captures the data in the social graph database. Name an entity set using the corresponding node label. Name a relationship set using the corresponding edge label. Use the system generate unique id of nodes as the primary key of corresponding entity sets. Use the properties of nodes as attributes of corresponding entity sets. Use the properties of edges as attributes of corresponding relationship sets. See the ER diagram in Fig. 1 that is mapped from PERSON node labels and FRIENDSHIP edge labels.

Diagram

Description automatically generated

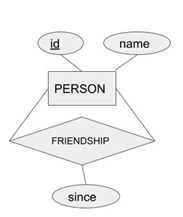


Fig. 1: ER diagram mapped from Person nodes and FRIENDSHIP edges in the social graph database.

1. Design **relational schemas** from the ER diagram in your answer in Question 2 for the relationship sets that represent POSTED and LIVES and the entity sets that represent STATE. Omit the data types in the schemas.

Example of relational schemas for the diagram in Fig. 1.

person(id, name, primary key(id))

friendship(fromperson1, topperson2, since, primary key(fromperson1, topperson2), foreign key(fromperson1) references person(id), foreign key(fromperson1) references person(id))

**Lives(pid, lid, primary key(pid, lid), foreign key(pid) references person(id), foreign key(lid) references location(id))**

**Location(id, state, primary key(id))**

**Posted(pid, message, primary key(pid), foreign key(pid) references person(id))**

**Thursday**

**Instruction**

Like MySQL, Neo4j allows data import from a specific folder for each database. In this activity, you will import data from three csv files: parts.csv, suppliers.csv, supplies.csv. To know which folder to put these files in, follow the video recording of the lecture. Put the parts.csv, suppliers.csv, and supplies.csv in the import folder. The data were exported from a relational database that was converted from the ER diagram in Fig. 2.

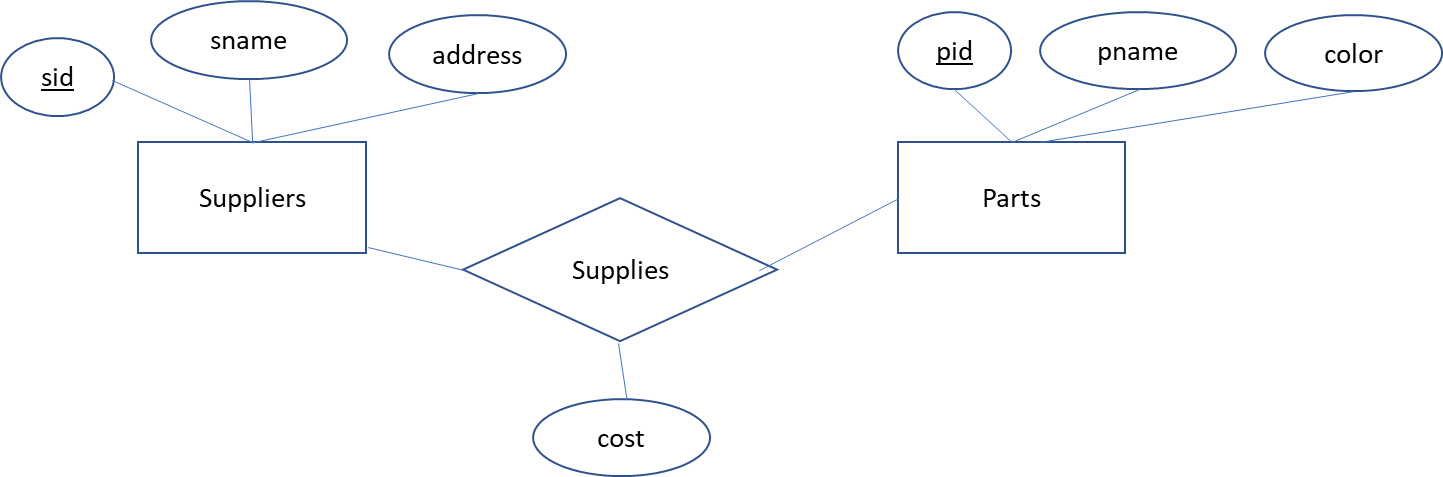


Fig. 2 ER diagram of a database of suppliers, parts, and suppliers supply parts.

Q4. Execute the Cypher statements in Lines 1-6 from LoadData.cypher into Neo4j query window and execute the statements. Repeat the same process for Lines 8-12 and then Lines 16-19. Write Cypher statements to answer the following queries.

1. Return the number of suppliers (number of nodes with the label “Suppliers”).

**MATCH (s:Suppliers) RETURN COUNT(s)**

1. Return the number of edges of the label “SUPPLIES”.

**MATCH (:Suppliers) – [e:SUPPLIES]🡪(:Parts) RETURN COUNT(e)**

1. Add one new supplier with your name as the value of sname. Find the *snames* of suppliers who do not supply any part. Show the result in ascending order of *snames*. This query should return only the sname that you just add.

**CREATE(:Suppliers{sname: “Chimzim”}) Return s.pid = null**

1. Find unique *sids* and *snames* of suppliers who **supply a black part and a blue part**. Show the result in ascending order of *sids*.

**MATCH (:Parts{Color: ‘Blue’})🡨(s:Suppliers)🡪(:Parts{Color: ‘Black’}) RETURN distinct s.sid, s.sname order by s.sid**

1. Find suppliers who supply exactly 40 parts. List sid and sname values.
2. Find the snames of the suppliers who supply **every green part**. Show the result in ascending order of *snames*.