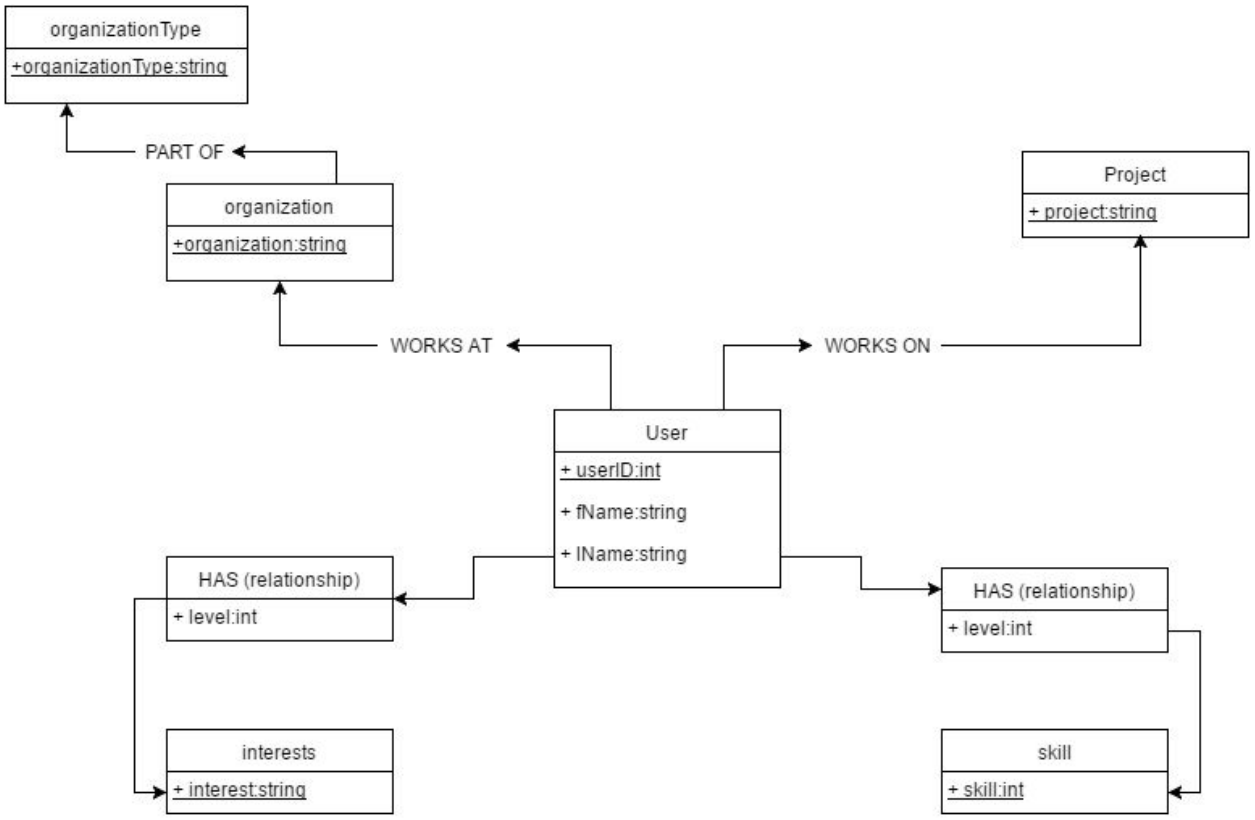
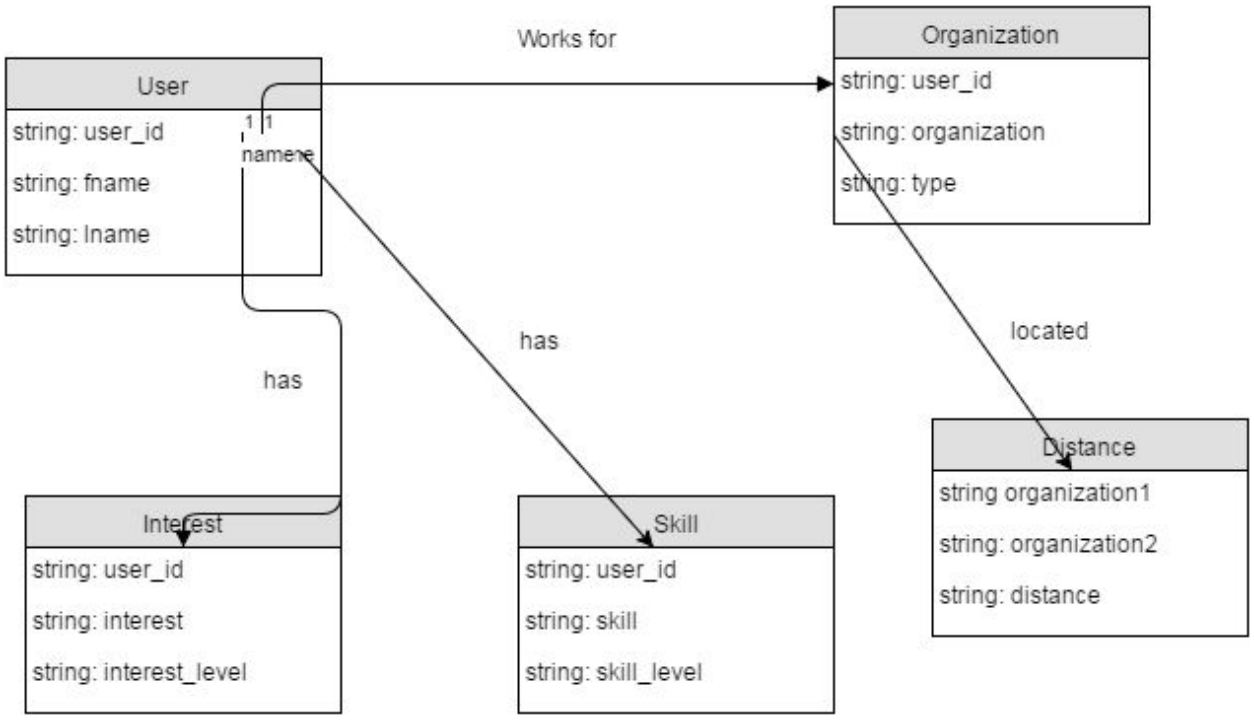
Design Diagram

Neo4J:



MongoDB:



Description of Data Used

There were six (6) files used in order to test the queries. The files are as followed:

names.csv	Column 1: UserID Column 2: First Name Column 3: Last Name
organizations.csv	Column 1: UserID Column 2: Organization Name Column 3: Organization Type
distance.csv	Column 1: Organization 1 Column 2: Organization 2 Column 3: Distance between Organizations
interests.csv	Column 1: UserID Column 2: Interest Column 3: Significance of Interest
skills.csv	Column 1: UserID Column 2: Skill Column 3: Level of Skill
projects.csv	Column 1: UserID Column 2: Project Name

- UserIDs, Organization Names, Interests, Skill, and Projects were unique
- A user can have zero or more skills and interests. Each skill or interest must have a level.
- A user can only work on one project and work at one company
- A company can only belong to one organization type
- A user must have a level/significance for a skill/interest
- Organization1 and Organization2 pair values are unique

Queries Used

Neo4J:

Use: Query database if user exists via userID

MATCH (n:user) WHERE n.userID = %d RETURN n

Use: Query database if organization exists via orgName

MATCH (n {organization: %s}) RETURN n

Use: Query database if organizationType exists via orgTypeName

MATCH (n {organizationType:%s}) RETURN n

Use: Query database if relationship between organization and organizationType exists

MATCH (a{organization:%s})--(b{organizationType:%s}) RETURN a

Use: Query database if user is already added to organization

MATCH (a{userID:%d})--(b{organization:%s}) RETURN a

Use: Query database if organization exists via projectName

MATCH (n {project: %s}) RETURN n

Use: Query database if organization exists via skillName

MATCH (n {skill: %s}) RETURN n

Use: Query database if organization exists via interestName

MATCH (n {interest: %s}) RETURN n

Use: Query database if distance relationship between two organizations exist

MATCH (a{organization:"%s"})--(b{organization:"%s"}) RETURN a

Use: Queries database by userID selected and returns trusted colleagues of colleagues that share common interests, their organizations, their projects, and shared interests with user

1. *MATCH (a:user)--(b:organization) WHERE a.userID = %d*
   WITH b.organization as orgName
2. *MATCH (c:user)--(d:organization) WHERE d.organization = orgName and c.userID <> %d*
   WITH c.userID as userIDs
3. *MATCH (e:user)--(f:project) WHERE e.userID = userIDs*
   WITH f.project as project,userIDs as users
4. *MATCH (g:user)--(h:project) WHERE h.project = project and g.userID <> users*
   WITH g as trusted
5. *MATCH (a:user)--(b:interest) WHERE a.userID = %d*
   WITH b as interests,trusted
6. *MATCH (c:user)--(d:interest) WHERE d=interests and c.userID <> %d and c=trusted*
   WITH DISTINCT c as trustedUsers ,collect(d.interest) as interests
7. *MATCH (a:user)--(b:organization) WHERE a = trustedUsers*
   WITH trustedUsers,b.organization as trustedOrgs,interests
8. *MATCH (a:user)--(b:project) WHERE a = trustedUsers*
9. *RETURN a as trustedUsers,trustedOrgs,b.project as projects,interests*

Explanation:

1. The organization of the user is retrieved
2. Using the organization, retrieve all members of that organization (userIDs), excluding user
3. For every colleague of user, retrieve their projects
4. If their projects have other members, retrieve their names (trusted)
5. Find interests of the user
6. Find all trusted colleagues who share the same interests as user and list interests (interests)
7. Retrieve all trusted colleagues' organizations
8. Retrieve all trusted colleagues' projects
9. Returns a list of all trustedUsers, their respective organizations, their project, and interests shared with user

Use: Queries database by userID and returns all users within a 10-mile radius that share common interests with the user and return the users, the sum of their interest levels, their interests, and their organizations in descending order

1. *match (a:user)-[`WORKS AT`]-(m:organization)--(n:organizationType) where a.userID = %d*
   WITH m.organization as Org, n.organizationType as orgType
2. *match (n:organization)-[a:DISTANCE]-(m:organization)--(t:organizationType) where a.distance <= 10 and n.organization = Org and t.organizationType = orgTypeWITH m.organization as closeOrgs, Org*
3. *MATCH (a:organization)-[`WORKS AT`]-(c:user) WHERE a.organization = closeOrgs or a.organization = Org*
   WITH DISTINCT c.userID as closeUsers, a.organization as org
4. *MATCH (a:user)--(b:interest)-[r:HAS]-(c:user) WHERE a.userID = %d and c.userID = closeUsers*
5. *RETURN c as users, sum(r.level) as userSum, org, collect(b.interest) as interests*
   ORDER BY userSum DESC

Explanation:

1. Retrieve the organization where the user works at and its organization type
2. Retrieve all organizations within a 10-mile radius of the user's organization and type, including the user's organization (closeOrgs, Org)
3. Retrieve all users who work at nearby organizations (closeUsers)
4. Retrieve all closeUsers who have a common interest with user
5. Return closeUsers, the sum of their interest levels, their organization, and a list of their interests and order by descending skill levels

Use: Queries database by userID and returns all users within a 10-mile radius that share common skills with the user and return the users, the sum of their skill levels, their skills, and their organizations in descending order

1. *match (a:user)-[`WORKS AT`]-(m:organization)--(n:organizationType) where a.userID = %d*
   WITH m.organization as Org, n.organizationType as orgType
2. *match (n:organization)-[a:DISTANCE]-(m:organization)--(t:organizationType) where a.distance <= 10 and n.organization = Org and t.organizationType = orgType*
   WITH m.organization as closeOrgs, Org
3. *MATCH (a:organization)-[`WORKS AT`]-(c:user) WHERE a.organization = closeOrgs or a.organization = Org*
   WITH DISTINCT c.userID as closeUsers, a.organization as org
4. *MATCH (a:user)--(b:skill)-[r:HAS]-(c:user) WHERE a.userID = %d and c.userID = closeUsers*
5. *RETURN c as users, sum(r.level) as userSum, org, collect(b.skill) as skills*
   ORDER BY userSum DESC

Explanation:

1. Retrieve the organization where the user works at and its organization type
2. Retrieve all organizations within a 10-mile radius of the user's organization and type, including the user's organization (closeOrgs, Org)
3. Retrieve all users who work at nearby organizations (closeUsers)
4. Retrieve all closeUsers who have a common skill with user

5. Return closeUsers, the sum of their skill levels, their organization, and a list of their skills and order by descending skill levels

MongoDB:

Query Looks through a collection based on userId because it is unique for all users.

- db.collection.find({userId : value})

Creates a rule for a collection for the attribute userID to be unique and ascending value

- db.collection.create_index("userID", unique=True)

The query looks up the key value pair and returns all matches in that collection. For example if we are looking for a user in the Skills collection we would use db.Skills.find{userId: 1}), it would then return the dictionary containing all the attributes for the user with id 1.

CreateIndex makes sure that given a key it keeps its uniqueness in the database else it would throw an error in our case we want to make sure there are no duplicate entries in collections like users this way we keep the collection clean and prevent redundancy

Potential Improvements

Neo4J Database:

- Performance of inserting could've been increased by removing some queries which checked for duplicates
- Queries could include multiple relationship searches instead of searching relationships one by one
- **Uncertain of Performance Increase:** Instead of creating distance relationships between each company, companies could possibly be returned via a smaller query, ran in MongoDB, distances then returned into Neo4J query.

MongoDB:

- Find query returns all matching results therefore there is an issue of duplicate entries since we are assuming each user has a unique id.
- Time complexity will be high given a large amount of data for collections such as skills and interest, our code contains a check for multiple skills for each person if such a condition is true then we create a list to store the values for this attribute, this requires a double for loop one to iterate through the file and another to check if the next entry read is a duplicate user id in the collection