# Database Technologies: Neo4j Practical Assignment 1

Create the following databases as graph models. Visualize the models after creation, Return properties of nodes, Return the nodes labels, Return the relationships with its properties.

**NB:** You may assume and add more labels , relationships, properties to the graphs

# Create a library database , as given below.

There are individual books, readers, and authors that are present in the library data model.. A minimal set of labels are as follows:

**Book**: This label includes all the books

**Person**: This label includes authors, translators, reviewers, Readers, Suppliers and so on

**Publisher**: This label includes the publishers of books in the database

A set of basic relationships are as follows:

**PublishedBy**: This relationship is used to specify that a book was published by a publisher **Votes**: This relationship describes the relation between a user and a book, for example, how a book was rated by a user.

**ReviewedBy** : This relationship is used to specify that a book was reviewed and remarked by a user.

**TranslatedBy**: This relationship is used to specify that a book was translated to a language by a user.

**IssuedBy**: This relationship is used to specify that a book was issued by a user. ReturnedBy: This relationship is used to specify that a book was returned by a user

Every book has the following properties:

**Title**: This is the title of the book in string format

**Tags**: This is an array of string tags useful for searching through the database based on topic, arguments, geographic regions, languages, and so on

**Status**: the book status , specifying whether its issued or in library.

**Condition**: book condition, new or old

# Cost : Cost of book

**Type**: book is a Novel, Journal, suspense thriller etc

# Consider a Song database, with labels as Artists, Song, Recording\_company, Recoding\_studio, song author etc.

Relationships can be as follows

Artist → [Performs] → Song →[Written by] → Song\_author.

Song → [Recorded in ] → Recording Studio →[managed by] → recording Company Recording Company → [Finances] → Song

You may add more labels and relationship and their properties, as per assumptions.

# Consider an Employee database, with a minimal set of labels as follows Employee: denotes a person as an employee of the organization Department: denotes the different departments, in which employees work. Skillset: A list of skills acquired by an employee

Projects: A list of projects in which an employee works.

A minimal set of relationships can be as follows: Works\_in : employee works in a department Has\_acquired: employee has acquired a skill Assigned\_to : employee assigned to a project Controlled\_by: A project is controlled by a department Project\_manager : Employee is a project\_manager of a Project

# Consider a movie database, with nodes as Actors, Movies, Roles, Producer, Financier, Director. Assume appropriate relationships between the nodes, include properties for nodes and relationships.

1. **Create a Social network database , with labels as Person, Affiliations, Groups, Story, Timeline etc. Some of the relationships can be as follows:**

Person →[friend of]→ Person→[affiliated to]→affiliations

Person →[belongs to]→ Groups, Person →[create]→Story→[refers to] →Person Person→[creates]→Timeline→[reference for]→ Story , Timeline→[contains]→Messages

# Database Technologies: Neo4j Practical Assignment 2 Simple Queries.

1. Library Database :
   1. List all people, who have issued a book “……”
   2. Count the number of people who have read “ ….”
   3. Add a property “Number of books issued “ for Mr. Joshi and set its value as the count
   4. List the names of publishers from pune city.
2. Song Database:
   1. List the names of songs written by “:…..”
   2. List the names of record companies who have financed for the song “….”
   3. List the names of artist performing the song “…..”
   4. Name the songs recorded by the studio “ …….”
3. Employee Database:
   1. List the names of employees in department “ ”
   2. List the projects along with their properties, controlled by department “……”

List the departments along with the count of employees in it

* 1. List the skillset for an employee “ ”

1. Movie Database:
   1. Find all actors who have acted in a movie “ ”
   2. Find all reviewer pairs, one following the other and both reviewing the same movie, and return entire subgraphs.
   3. Find all actors that acted in a movie together after 2010 and return the actor names and movie node
   4. Find all movies produced by “ ”
2. Social Network Database:
   1. Find all friends of “John”, along with the year, since when john knows them.
   2. List out the affiliations of John.
   3. Find all friends of john, who are born in the same year as John
   4. List out the messages posted by John in his timeline, during the year 2015.

# Database Technologies: Neo4j Assignment 3 Complex pattern Queries:

1. Library database
   1. List all readers who have recommended either book “…” or “……..” or “ ”
   2. List the readers who haven’t recommended any book
   3. List the authors who have written a book that has been read / issued by maximum number of readers.
   4. List the names of books recommended by “ ” And read by at least one reader
   5. List the names of books recommended by “ ” and read by maximum number of

readers.

* 1. List the names of publishers who haven’t published any books written by authors from Pune and Mumbai.
  2. List the names of voracious readers in our library

1. Song Database:
   1. List the names of artists who have sung only songs written by “ ”
   2. List the names of artists who have sung the maximum number of songs recorded by “……” studio
   3. List the names of songs financed by “…..”, and sung by “ ”
2. Employee Database:
   1. List the names of employees having the same skills as employee “ ”
   2. List the projects controlled by a department “ ” and have employees of the same

department working in it.

* 1. List the names of the projects belonging to departments managed by employee “ ”

1. Movie Database:
   1. List the names of actors that paired in multiple movies together.
   2. List all pairs of actor–movie subgraphs along with the roles played.
   3. List all reviewers and the ones they are following directly or via another a third Reviewer
   4. List the names of movies that have the most number of reviews.
2. Social Network Database:
   1. List out the people, who have created maximum timeline messages.
   2. List all friends of John’s friend, Tom
   3. List the people with maximum friends
   4. List the people who are part of more than 3 groups.

# Database Technologies: Neo4j Practical Assignment 1

Create the following databases as graph models. Visualize the models after creation, Return properties of nodes, Return the nodes labels, Return the relationships with its properties.

**NB:** You may assume and add more labels , relationships, properties to the graphs

# Create a Library database,

**There are individual books, readers, and authors that are present in the library data model.. A minimal set of labels are as follows:**

**Book** : This label includes all the books

**Person** : This label includes authors, translators, reviewers, Readers, Suppliers and so on

**Publisher** : This label includes the publishers of books in the database

A set of basic relationships are as follows:

**PublishedBy** : This relationship is used to specify that a book was published by a publisher **Votes**: This relationship describes the relation between a user and a book, for example, how a book was rated by a user.

**ReviewedBy** : This relationship is used to specify that a book was reviewed and remarked by a user.

**TranslatedBy** : This relationship is used to specify that a book was

translated to a language by a user.

**IssuedBy** : This relationship is used to specify that a book was issued by a user.

**ReturnedBy :** This relationship is used to specify that a book was returned by a user

Every book has the following properties:

**Title** : This is the title of the book in string format

**Tags** : This is an array of string tags useful for searching through the database based on topic, arguments, geographic regions, languages, and so on **Status** : the book status , specifying whether its issued or in library.

**Condition**: book condition, new or old

### Cost : Cost of book

**Type** : book is a Novel, Journal, suspense thriller etc

#View All nodes, labels, Relationships etc.

---> match (n) return n

#Delete All nodes, labels, Relationships etc.

---> MATCH (n) DETACH DELETE n

@CREATE (tinker:Book{title:'Bayari',tag: [ 'Social Issues','Maharashtra'], published:1988,cost:550, type:'Novel'})

#Delete All nodes, labels, Relationships where Book="Bayari"

--->

match (n:Book{title:'Bayari'}) Detach delete n

CREATE(pk:Publisher{name:'PK',city:'Pune'})

CREATE (john:Author{name:'John Le Carre', born:'19-10-1932'})

CREATE (graham:Author{name:'Graham Greene',born:'02-10-1904',died:'02-04- 1991'})

CREATE (tinker:Book{title:"Tinker Tailor Soldier Spy", tag:['English', 'Japanies'], status:'not Issued', condition:'New', published:1974, cost:350, type:'Novel'})

CREATE (our:Book{title:'Our Man in Havana', tag:['Ameriaca','English','Korian'], status:'Issued',condition:'new', published:1958, cost:250, type:'suspense thriller'})

CREATE (lan:Reader:Author{name:'Lan'}) CREATE (alan:Reader{name:'Alan'}) CREATE(clay:Reader{name:'Clay'}) CREATE(han:Reader{name:'Hanahha Baker'})

CREATE(Jassica:Auther{name:'Jassica'}) CREATE (our)-[:PUBLISHED\_BY]->(pk),

(Jassica)<-[:TRANSLATED\_BY]-(tinker),

(our)-[:ISSUED\_BY]->(clay),

(tinker)-[:REVIEWED\_BY]->(pk),

(han)-[:VOTES{stars:4}]->(our),

(tinker)-[:PUBLISHED\_BY]->(pk),

(john)-[:WROTE]->(tinker),

(alan)-[:RECOMMENDED{date:'05-07-2011'}]->(tinker),

(lan)-[:RECOMMENDED{date:'09-09-2011'}]->(tinker),

(lan)-[:RECOMMENDED{date:'03-02-2011'}]->(our),

(graham)-[:WROTE]->(our)

CREATE(b:Book{title:"Bayari",

tag:['English', 'Marathi'], status:'not Issued', condition:'New', published:1999, cost:333, type:'Novel'})

match (lan:Reader{name:'Lan'}) create (b)-[:RECOMMENDED]->(lan) return b,lan

match (:Reader{Name:"Lan"})-[r:RECOMMENDED]->(:Book{title="Tinker Tailor Soldier Spy"})

detach delete r

match (b:Book),(r:Reader)

where b.title="Our Man in Havana" and r.name="Hanahha Baker" create(b)-[i:ISSUED\_BY]->(r)

return b,r

match (b:Book),(r:Reader)

where b.title="Tinker Tailor Soldier Spy" and r.name:"Clay" create(b)-[i:ISSUED\_BY]->(r)

return b,r

CREATE (sane:Author{name:'sane guruji', born:'09-10-1942', city :"Satara"}) match(b:Book),(a:Author)

where b.title="Bayari" and a.name="sane guruji" create (a)-[:WROTE]->(b)

return a,b

match (p:Publisher)<-[r:PUBLISHED\_BY]-(b:Book)<-[rr:WROTE]-(a:Author) return r,rr

# Consider a Song database,

**with labels as Artists, Song, Recording\_company, Recoding\_studio, song author etc.**

Relationships can be as follows

Artist → [Performs] → Song →[Written by] → Song\_author.

Song → [Recorded in ] → Recording Studio →[managed by] → recording Company Recording Company → [Finances] → Song

You may add more labels and relationship and their properties, as per assumptions.

CREATE(pk:Artist:Song\_Author{Name:'PK', Age:20, followers:'50M'}) CREATE(bantai:Artist:Song\_Author{Name:'Emiway Bantai', Age:26, followers:'5M'})

CREATE(guru:Artist:Song\_Author{Name:'Guru', Age:27, followers:'12M'}) CREATE(raf:Artist:Song\_Author{Name:'Raftaar', Age:30, followers:'4M'}) CREATE(divine:Artist:Song\_Author{Name:'Divine', Age:31, followers:'14M'}) CREATE(neha:Artist:Song\_Author{Name:'Neha Kakkar', Age:29, followers:'3M'})

CREATE(hard:Song{Name:'Bohot Hard', likes:'40M'}) CREATE(gully:Song{Name:'Mere Gully Main', likes:'12M'}) CREATE(azadi:Song{Name:'Azadi', likes:'7M'}) CREATE(asli:Song{Name:'Asli',likes:'8M'}) CREATE(gabru:Song{Name:'High Rated Gabru', likes:'10M'}) CREATE(ladki:Song{Name:'Ladki Marwake Marke Maneggii', likes:'2.5M'}) CREATE(machayenge:Song{Name:'Machayenge', likes:'8M'}) CREATE(chull:Song{Name:'Kar Gayi Chull', likes:'5M'})

CREATE(arijit:Song\_Author{Name:'Arijit Singh',No\_songs:50}) CREATE(tony:Song\_Author{Name:'Tony Kakkar',No\_songs:112})

CREATE(coke:Recording\_company{Name:'Coke Studio'}) CREATE(zee:Recording\_company:Recoding\_studio{Name:'Zee Studio'})

CREATE (pk)-[:PERFORMS]->(hard),

(gabru)-[:WRITTEN\_BY]->(bantai),

(bantai)-[:PERFORMS]->(machayenge)-[:WRITTEN\_BY]->(bantai), (guru)-[:PERFORMS]->(gabru)-[:WRITTEN\_BY]->(arijit),

(raf)-[:PERFORMS]->(ladki)-[:WRITTEN\_BY]->(raf),

(divine)-[:PERFORMS]->(gully)-[:WRITTEN\_BY]->(divine),

(divine)-[:PERFORMS]->(azadi)-[:WRITTEN\_BY]->(divine), (neha)-[:PERFORMS]->(chull),

(asli)-[:RECORDED\_IN]->(zee)-[:MANAGED\_BY]->(zee)-[:Finances]->(hard),

(gabru)-[:RECORDED\_IN]->(coke)-[:MANAGED\_BY]->(zee),

(ladki)-[:RECORDED\_IN]->(zee)-[:MANAGED\_BY]->(coke)-[:Finances]-

>(chull),

(azadi)-[:RECORDED\_IN]->(coke)-[:MANAGED\_BY]->(coke),

(neha)-[:FOLLOWS]->(arijit)-[:FOLLOWS]->(guru)-[:FOLLOWS]->(raf)- [:FOLLOWS]->(tony)-[:FOLLOWS]->(pk)

# Consider an Employee database,

**with a minimal set of labels as follows Employee: denotes a person as an employee of the organization Department: denotes the different departments, in which employees work. Skillset: A list of skills acquired by an employee**

**Projects**: A list of projects in which an employee works. A minimal set of relationships can be as follows:

**Works\_in** : employee works in a department **Has\_acquired:** employee has acquired a skill **Assigned\_to** : employee assigned to a project **Controlled\_by**: A project is controlled by a department

**Project\_manager** : Employee is a project\_manager of a Project

-->

CREATE(harry:Employee {Name:'Harry', age:29, Qualification:['MCS','BCS'], Experience:8})

CREATE(pashya:Employee {Name:'Pashya', age:30, Qualification:['MCA','BCA','MSCIT'],Experience:8}) CREATE(bablu:Employee {Name:'Bablu', age:28, Qualification:['B.Tech','MSCIT'],Experience:5}) CREATE(monu:Employee {Name:'Monu', age:26, Qualification:['B.Tech','M.Tech'],Experience:3}) CREATE(babu:Employee {Name:'Babu', age:32,

Qualification:['M.Tech','B.Tech','MSCIT','MCS','BCS'],Experience:10}) CREATE(nandu:Employee {Name:'Nandu', age:34, Qualification:['B.Tech','BCS','MSCIT'],Experience:4})

CREATE(it:Department{Name:'IT',no\_of\_Emp:5}) CREATE(bpo:Department{Name:'BPO',no\_of\_Emp:5}) CREATE(cbo:Department{Name:'CBO',no\_of\_Emp:5}) CREATE(cmn:Department{Name:'TeleCommunication',no\_of\_Emp:5})

CREATE(vgd:Skillset{skills:['Fluent Communication','Leadership Qualities','Optimistic']})

CREATE(bet:Skillset{skills:['Good Communication','Java Devloper']}) CREATE(gd:Skillset{skills:['Leadership Qualities','Optimistic','Finnance Matser']})

CREATE(av:Skillset{skills:['Fluent Communication']})

CREATE(sg:Projects{Name:'SG Website Design', TimeSpan:'30day', clinet:'SG Architecture'})

CREATE(food:Projects{Name:'Food Deliver app', TimeSpan:'35day', clinet:'Hydrabaad Biryanni'}) CREATE(location:Projects{Name:'Location Finder', TimeSpan:'45day', clinet:'AI Location Developer'})

CREATE(ecom:Projects{Name:'Ecommerce Website Design', TimeSpan:'50day', clinet:'Eco-market'})

CREATE(tata:Projects{Name:'Tata Sky', TimeSpan:'20day', clinet:'SkyVoice India'})

CREATE(out:Projects{Name:'Out Bound Process', TimeSpan:'90day', clinet:'Ruby Max'})

CREATE (harry)-[:Works\_in]->(it),

(pashya)-[:Works\_in]->(bpo),

(bablu)-[:Works\_in]->(cbo),

(monu)-[:Works\_in]->(cmn),

(nandu)-[:Works\_in]->(it),

(harry)-[:Has\_acquired]->(vgd),

(pashya)-[:Has\_acquired]->(bet),

(bablu)-[:Has\_acquired]->(gd),

(monu)-[:Has\_acquired]->(vgd),

(nandu)-[:Has\_acquired]->(bet),

(harry)-[:Has\_acquired]->(av),

(harry)-[:Assigned\_to]->(sg),

(nandu)-[:Assigned\_to]->(food),

(bablu)-[:Assigned\_to]->(ecom),

(monu)-[:Assigned\_to]->(sg), (pashya)-[:Assigned\_to]->(location), (harry)-[:Assigned\_to]->(out),

(bablu)-[:Assigned\_to]->(out),

(out)-[:Controlled\_by]->(it), (location)-[:Controlled\_by]->(cbo), (sg)-[:Controlled\_by]->(it),

(ecom)-[:Controlled\_by]->(bpo),

(food)-[:Controlled\_by]->(cmn),

(nandu)-[:Project\_manager]->(sg), (bablu)-[:Project\_manager]->(food), (monu)-[:Project\_manager]->(ecom), (pashya)-[:Project\_manager]->(location), (babu)-[:Project\_manager]->(tata), (harry)-[:Project\_manager]->(out)

# Consider a movie database, with nodes as Actors, Movies, Roles, Producer, Financier, Director. Assume appropriate relationships between the nodes, include properties for nodes and relationships.

CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})

CREATE (Keanu:Person {name:'Keanu Reeves', born:1964}) CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})

CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961}) CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})

CREATE (LillyW:Person {name:'Lilly Wachowski', born:1967}) CREATE (LanaW:Director{name:'Lana Wachowski', born:1965}) CREATE (JoelS {name:'Joel Silver', born:1952})

CREATE

(Keanu)-[:ACTED\_IN {roles:['Neo']}]->(TheMatrix),

(Carrie)-[:ACTED\_IN {roles:['Trinity']}]->(TheMatrix),

(Laurence)-[:ACTED\_IN {roles:['Morpheus']}]->(TheMatrix),

(Hugo)-[:ACTED\_IN {roles:['Agent Smith']}]->(TheMatrix), (LillyW)-[:DIRECTED]->(TheMatrix),

(LanaW)-[:DIRECTED]->(TheMatrix), (JoelS)-[:PRODUCED]->(TheMatrix)

CREATE (Emil:Person {name:"Emil Eifrem", born:1978}) CREATE (Emil)-[:ACTED\_IN {roles:["Emil"]}]->(TheMatrix)

CREATE (TheMatrixReloaded:Movie {title:'The Matrix Reloaded', released:2003, tagline:'Free your mind'})

CREATE

(Keanu)-[:ACTED\_IN {roles:['Neo']}]->(TheMatrixReloaded), (Carrie)-[:ACTED\_IN {roles:['Trinity']}]->(TheMatrixReloaded), (Laurence)-[:ACTED\_IN {roles:['Morpheus']}]->(TheMatrixReloaded), (Hugo)-[:ACTED\_IN {roles:['Agent Smith']}]->(TheMatrixReloaded), (LillyW)-[:DIRECTED]->(TheMatrixReloaded),

(LanaW)-[:DIRECTED]->(TheMatrixReloaded), (JoelS)-[:PRODUCED]->(TheMatrixReloaded)

CREATE (AngelaScope:Person {name:'Angela Scope'})

CREATE (JessicaThompson:Person {name:'Jessica Thompson'})

CREATE

(JessicaThompson)-[:REVIEWED {summary:'An amazing journey', rating:95}]-

>(TheMatrixReloaded),

(JessicaThompson)-[:REVIEWED {summary:'Silly, but fun', rating:65}]-

>(TheMatrix),

(AngelaScope)-[:REVIEWED {summary:'Pretty funny at times', rating:62}]-

>(TheMatrixReloaded)

Match (n) detach delete n

# Create a Social network database , with labels as Person, Affiliations, Groups, Story, Timeline etc. Some of the relationships can be as follows:

Person →[friend of]→ Person→[affiliated to]→affiliations

Person →[belongs to]→ Groups, Person →[create]→Story→[refers to] →Person Person→[creates]→Timeline→[reference for]→ Story , Timeline→[contains]→Messages

CREATE(j:person{name:"john",yoj:"2004",birthd+ay:"1996"}),(a:person{name:"am an",birthday:"1996",yoj:"2005"}),(b:person{name:"bunny",birthday:"1995",yoj: "2006"}),(e:affiliation{name:"facebook"}), (g:group{name:"group"}),(t:timeline{name:"timeline"}),(s:story{name:"himacha

lstory"}),(m:message{name:"newmessage"}),(g1:group{name:"g1"}),(g2:group{nam e:"g2"}),(g3:group{name:"g3"})

match (b:person),(s:story) where b.name ="bunny" and s.name = "himachalstory" create(s)-[r3:refers\_to]-> (b)

match (m:message),(t:timeline) where m.name ="newmessage" and t.name = "timeline" create(t)-[r4:contains]-> (m)

match (s:story),(t:timeline) where s.name ="himachalstory" and t.name = "timeline" create(t)-[r4:reference\_for]-> (s)

match (j:person),(t:timeline) where j.name ="john" and t.name = "timeline" create(j)-[r4:creates]-> (t)

match (j:person),(s:story) where j.name ="john" and s.name = "himachalstory" create(s)-[r3:refers\_to]-> (j)

match (j:person),(s:story) where j.name ="john" and s.name = "himachalstory" create(j)-[r3:create]-> (s)

match (a:person),(e:affiliation) where a.name ="aman" and e.name = "facebook" create(a)-[r1:affiliated\_to]->(e)

match (j:person),(e:affiliation) where j.name ="john" and e.name = "facebook" create(j)-[r1:affiliated\_to]->(e)

match (b:person),(e:affiliation) where b.name ="bunny" and e.name = "facebook" create(b)-[r1:affiliated\_to]->(e)

match (j:person),(b:person) where j.name ="john" and b.name = "bunny" create(j)-[r1:friend\_of]-> (b)

match(j:person),(g1:group) create(j)-[r2:belongs\_to]->(g1)

# Database Technologies: Neo4j Practical Assignment 2 Simple Queries.

**# 1. Library Database :**

### List all people, who have issued a book “Our Man in Havana”.

-->MATCH (b:Book)-[r:ISSUED\_BY]->(rd:Reader) WHERE b.title='Our Man in Havana'

RETURN b,r,rd

### Count the number of people who have read “Tinker Tailor Soldier Spy” .

**-->**[Note : Count this query doesn't give output for both conditions Issued\_by and RECOMMENDED (i.e. both are readers)]

MATCH (a:Reader)-[r:RECOMMENDED]->(b:Book)

WHERE b.title="Tinker Tailor Soldier Spy" RETURN COUNT(a)

MATCH (a:Reader)-[r:ISSUED\_BY]->(b:Book) WHERE b.title="Our Man in Havana"

RETURN COUNT(a)

### Add a property “Number of books issued" for "Mr. Clay" and set its value as the count

-->[Note: These query is not running.]

MATCH (clay:Reader{name:'Clay'}) SET clay.No\_of\_Issued=4

RETURN clay

### List the names of publishers from pune city.

-->match(p:Publisher{city:'Pune'}) return p.name

# # 2.Song Database:

### List the names of songs written by “:Emiway Bantai”

-->Match(s:Song)-[r:WRITTEN\_BY]->(a:Song\_Author) where a.Name='Emiway Bantai'

return s.Name

### List the names of record companies who have financed for the song “Kar Gayi Chull” .

-->Match(rec:Recording\_company)-[r:Finances]->(s:Song) where s.Name='Kar Gayi Chull'

return rec,s,r

### List the names of artist performing the song “Bohot Hard” .

-->MATCH (a:Artist)-[r:PERFORMS]->(s:Song)

WHERE s.Name='Bohot Hard'

RETURN a, s,r

### Name the songs recorded by the studio “ …….”

-->MATCH (s:Song)-[r:RECORDED\_IN]->(rec:Recoding\_studio) WHERE rec.Name='Zee Studio'

RETURN s,rec,r

# # 3. Employee Database:

### List the name of employees in department" IT".

-->match (e:Employee)-[:Works\_in]->(:Department{Name:"IT"}) return e.Name

### List the projects along with their properties, controlled by department “IT”

-->match (d:Department{Name:'IT'})<-[:Controlled\_by]-(p:Projects) return d,p

### List the departments along with the count of employees in it

-->[Note : Not solved yet]

match (:Employee)-[:Works\_in]->(d:Department)--((e:Employee)) return d,COUNT(e)

### List the skillset for an employee “Harry”

-->MATCH (e:Employee{Name:"Harry"})-[:Has\_acquired]->(s:Skillset) return s,e

# # 4. Movie Database:

### Find all actors who have acted in a movie “The Matrix” .

-->MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

where m.title='The Matrix' return p.name

### Find all reviewer pairs, one following the other and

both reviewing the same movie, and return entire subgraphs.

-->match (p:Person)-[:REVIEWED]->(m:Movie), (:Movie)<-[:REVIEWED]-(p) return p.name,m.title

### Find all actors that acted in a movie together after 2000 and return the actor names and movie node .

-->MATCH (p:Person)-[:ACTED\_IN]->(m:Movie)

WHERE m.released>1998 RETURN p.name, m.title

### Find all movies produced by “ Joel Silver".

-->MATCH (p:Person)-[:PRODUCED]->(m:Movie)

where p.name='Joel Silver'

Return m.title

# # 5. Social Network Database:

### List out the affiliations of John.

-->match(j:person{name:"john"})-[r1:affiliated\_to]->(n) return n.name

### Find all friends of “John”, along with the year, since when john knows them.

-->match(j:person{name:"john"})-[r1:friend\_of]->(n) return n.name,n.yoj

### Find all friends of john, who are born in the same year as John

-->match(j:person{name:"john"})-[r1:friend\_of]->(n)

where j.birthday=n.birthday return n.name,n.birthday

### List out the messages posted by John in his timeline, during the year 2015.

**-->**

# Database Technologies: Neo4j Assignment 3 Complex pattern Queries:

**# 1. Library Database :**

### List all readers who have recommended either book “…” or “……..” or “… ”

MATCH (a:Reader)-[r:RECOMMENDED]->(b:Book)

WHERE b.title="Tinker Tailor Soldier Spy" or b.title="Our Man in Havana"

RETURN a

### List the readers who haven’t recommended any book

match (a:Reader) where not (a:Reader)-[:RECOMMENDED]->(:Book) return a

### List the authors who have written a book that has been read / issued by maximum number of readers.

MATCH (b:Book)-[r:ISSUED\_BY]->(a:Reader) RETURN b.title,COUNT(b)

MATCH (b:Book)-[r:ISSUED\_BY]->(a:Reader) where max(count(b))

RETURN b.title, COUNT(b)

### List the names of books recommended by “……….” And read by at least one reader

MATCH (a:Reader{name:"Lan"})-[r:RECOMMENDED]->(b:Book) WHERE count(r)>0

RETURN b

MATCH (a:Reader{name:"Lan" })-[r:RECOMMENDED]->(b:Book)-[rr:ISSUED\_BY]-

>(rd:Reader)

RETURN a,r,rd,rr,b

### List the names of books recommended by “………” and read by maximum number of readers.

1. **List the names of publishers who haven’t published any books written by authors from Pune and Mumbai.**

### List the names of voracious readers in our library [Voracious means the reader who haven't issued any book] MATCH (a:Reader)

WHERE NOT (:Book)-[:ISSUED\_BY]->(a:Reader)

RETURN a.name

# # 3. Employee Database:

### List the name of employees in department" IT".

-->match (e:Employee)-[:Works\_in]->(:Department{Name:"IT"}) return e.Name

### List the projects controlled by a department “IT.” and have employees of the same department working in it.

--->match (d:Department{Name:'IT'})<-[:Controlled\_by]-(p:Projects)<- [:Assigned\_to]-(e:Employee),

(e:Employee)-[:Works\_in]->(d:Department) return d,p,e

### List the names of the projects belonging to departments managed by employee “…….”

--->match (e:Employee{Name:'Harry'})-[:Project\_manager]->(p:Projects)- [:Controlled\_by]->(d:Department)

return e,p,d

### List the names of employees having the same skills as employee “… ”

--->match(e:Employee{Name:'Harry'})--(s:Skillset)--(ee:Employee) return ee,s

# # 5. Social Network Database:

1. **List out the people, who have created maximum timeline messages.**

-->match(n:person)-[r4:creates]->(t:timeline) return max(n)

1. **List all friends of John’s friend, Tom**

## -->match(n:person)<-[r4:friend\_of]-(j:person) return n

1. **List the people with maximum friends**

## -->match (n:person)-[r1:friend\_of]->(m:person) return max(n)

1. **List the people who are part of more than 3 groups.**

## -->match(m:person)-[r3:belongs\_to]->(g:group) with m, count(\*) as cnt where cnt>3 return m.name

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# 1. Create a library database , as given below.

There are individual books, readers, and authors that are present in the library data model.. A minimal set of labels are as follows:

**Book**: This label includes all the books

**Person**: This label includes authors, translators, reviewers, Readers, Suppliers and so on

**Publisher**: This label includes the publishers of books in the database

# Library Database Neo4j. NODE

CREATE (carre:author{name: "john le carre", DOB:1-10-1905,DOD:1-10-1995, city:"mumbai"}),(greene:author{name: "graham greene", DOB:1-10- 1905,DOD:1-10-1995, city:"pune"}),(kanitar:author{name: "yashwanth kanitkar", DOB:1-10-1905,DOD:1-10-1995, city:"delhi"})

CREATE (tinker:book{title:"tinker tailor soldier spy", DOP:1974}),(havana:book{title:"our man in havana", DOP:1958})

CREATE (alan:reader{name: "alan",email:["ala](mailto:alan@gmail.com)n[@gmail.com](mailto:alan@gmail.com)"}),(Ian:reader{name:"Ian",email:"[Ian@gmail.c](mailto:Ian@gmail.c) om"})

CREATE (BPB:publisher{name:"BPB",CITY:"Pune"}),(phi:publisher{name:"PHI", CITY:"Agra"}),(jaya:reader{name:"jaya"})

# RELATION

match (carre:author),(tinker:book) where carre.name="john le carre" and tinker.title= "tinker tailor soldier spy" create (carre)-[r1:WROTE]-> (tinker)

match (greene:author),(havana:book) where greene.name="graham greene" and havana.title= "our man in havana" create (greene)-[r1:WROTE]-> (havana)

match (alan:reader),(havana:book) where alan.name="alan" and havana.title= "our man in havana" create (alan)-[r2:ISSUED]-> (havana)

match (alan:reader),(havana:book) where alan.name="alan" and havana.title= "our man in havana" create (alan)-[r3:READ]-> (havana)

match (alan:reader),(tinker:book) where alan.name="alan" and tinker.title= "tinker tailor soldier spy" create (alan)-[r1:RECOMENDED]-> (tinker)

match (Ian:reader),(tinker:book) where Ian.name="Ian" and tinker.title= "tinker tailor soldier spy" create (Ian)-[r1:RECOMENDED]-> (tinker),(Ian)-[r2:ISSUED]-> (tinker),(Ian)-[r3:READ]-> (tinker)

match (Ian:reader),(havana:book) where Ian.name="Ian" and havana.title= "our man in havana" create (Ian)-[r1:RECOMENDED]-> (havana)

match (BPB:publisher),(tinker:book) where BPB.name="BPB" and tinker.title= "tinker tailor soldier spy" create (tinker)-[r1:PUB\_by]-> (BPB)

match (BPB:publisher),(havana:book) where BPB.name="BPB" and havana.title= "our man in havana" create (havana)-[r1:PUB\_by]-> (BPB)

match(kanitkar:author),(PHI:publisher) where kanitkar.name="yashwanth kanitkar" and PHI.name= "PHI" create(PHI)-[r2:likes]->(kanitkar)

# QUERRIES

1. **PEOPLE ISSUED A BOOK**

match(havana:book{title:"our man in havana"})<-[:ISSUED]-(n) return n.name

# COUNT NO OF PEOPLE WHO READ THE BOOK.

match(havana:book{title:"our man in havana"})<-[:ISSUED]-(n) return COUNT(n)

# ADD A PROPERTY NO OF BOOKS ISSUED FOR MR. Ian and set its value as count.

* + MATCH (alan:reader{name: "alan",email:["alan@gmail.co](mailto:alan@gmail.com)m"}) SET alan.no\_of\_booksissued=50 return alan
  + MATCH (Ian:reader{name: "Ian",email:"[Ian@gmail.com"](mailto:Ian@gmail.com)}) SET Ian.no\_of\_booksissued=25 return Ian

# LIST THE NAMES OF PUBLISHERS FROM PUNE CITY.

* + MATCH (n:publisher) WHERE n.CITY = "Pune" RETURN n

# LIST ALL READERS WHO HAVE RECOMMENDED EITHER BOOK "...." OR"..."OR ". "

* + MATCH (n:reader)-[r1:RECOMENDED]->(tinker:book) OR (havana:book) RETURN n.name

# `LIST THE AUTHOR WHO HAVEN'T RECOMMENDED ANY BOOK.

* + MATCH(n:reader) where not (n:reader)-[:RECOMENDED]->(:book) return n.name

# LIST THE AUTHORS WHO HAVE WRITTEN A BOOK THAT HAS BEEN READ/ISSUED BY MAXIMUM NUMBER OF READERS.

-

# LIST THE NAMES OF BOOKS RECOMMENDED BY"......." AND READ BY LEAST ONE READER.

-

# 9..LIST THE NAMES OF BOOKS RECOMMENDED BY"......." AND READ BY MAXIMUM NUMBER OF READERS.

-

# LIST THE NAMES OF PUBLISHERS WHO HAVEN'T PUBLISHED ANY BOOKS WRITTEN BY AUTHORS FROM PUNE AND MUMBAI

-

# LIST THE NAMES OF VORACIOUS READERS IN OUR LIBRARY.

- match(n:reader)-[:READ]->(:book) return n.name

# 5. Create a Social network database , with labels as Person, Affiliations, Groups, Story, Timeline etc. Some of the relationships can be as follows:

Person →[friend of]→ Person→[affiliated to]→affiliations

Person →[belongs to]→ Groups, Person →[create]→Story→[refers to] →Person Person→[creates]→Timeline→[reference for]→ Story , Timeline→[contains]→Messages

# SOCIAL NETWORK DATABASE Neo4j. NODE

CREATE(j:person{name:"john",yoj:"2004",birthd+ay:"1996"}),(a:person{name:

"aman",birthday:"1996",yoj:"2005"}),(b:person{name:"bunny",birthday:"1995

",yoj:"2006"}),(e:affiliation{name:"facebook"}),

(g:group{name:"group"}),(t:timeline{name:"timeline"}),(s:story{name:"hima

chalstory"}),(m:message{name:"newmessage"}),(g1:group{name:"g1"}),(g2:gro up{name:"g2"}),(g3:group{name:"g3"})

# RELATIONSHIP

match (b:person),(s:story) where b.name ="bunny" and s.name = "himachalstory" create(s)-[r3:refers\_to]-> (b)

match (m:message),(t:timeline) where m.name ="newmessage" and t.name = "timeline" create(t)-[r4:contains]-> (m)

match (s:story),(t:timeline) where s.name ="himachalstory" and t.name = "timeline" create(t)-[r4:reference\_for]-> (s)

match (j:person),(t:timeline) where j.name ="john" and t.name = "timeline" create(j)-[r4:creates]-> (t)

match (j:person),(s:story) where j.name ="john" and s.name = "himachalstory" create(s)-[r3:refers\_to]-> (j)

match (j:person),(s:story) where j.name ="john" and s.name = "himachalstory" create(j)-[r3:create]-> (s)

match (a:person),(e:affiliation) where a.name ="aman" and e.name = "facebook" create(a)-[r1:affiliated\_to]->(e)

match (j:person),(e:affiliation) where j.name ="john" and e.name = "facebook" create(j)-[r1:affiliated\_to]->(e)

match (b:person),(e:affiliation) where b.name ="bunny" and e.name = "facebook" create(b)-[r1:affiliated\_to]->(e)

match (j:person),(b:person) where j.name ="john" and b.name = "bunny" create(j)-[r1:friend\_of]-> (b)

match(j:person),(g1:group) create(j)-[r2:belongs\_to]->(g1)

# QUERRIES

1. **FIND ALL FRIENDS OF "JOHN" ALONG WITH THE YEAR OF JOINING.**

match(j:person{name:"john"})-[r1:friend\_of]->(n) return n.name,n.yoj

# LIST OUT THE AFFILIATIONS OF JOHN.

-match(j:person{name:"john"})-[r1:affiliated\_to]->(n) return n.name

# Find all friends of joh,,who are born in the same year as john.

- match(j:person{name:"john"})-[r1:friend\_of]->(n) where j.birthday=n.birthday return n.name,n.birthday

# LIST OUT THE MESSAGES POSTED BY JOHN IN HIS TIMELINE,DURING THE YEAR 2015.

-

# LIST OUT THE PEOPLE ,WHO HAVE CREATED MAXIMUM TIMELINE MESSAGES.

-match(n:person)-[r4:creates]->(t:timeline) return max(n)

# LIST OUT FRIENDS OF JOHN .

-match(n:person)<-[r4:friend\_of]-(j:person) return n

# LIST THE PEOPLE WITH MAXIMUM FRIENDS.

-match (n:person)-[r1:friend\_of]->(m:person) return max(n)

# List THE PEOPLE WHO ARE PART OF MORE THAN 3 GROUPS.

-match(m:person)-[r3:belongs\_to]->(g:group) with m, count(\*) as cnt where cnt>3 return m.name