**Create a Person with the name Alice that was born in 1976**

create (p:Person {name : 'Alice' , born : 1976}) return p

**Find Keanu Reeves and return the name and birthdate properties**

match (p:Person {name :'Keanu Reeves'}) return p.name as name , p.birthdate as dob

**Find People and return just 5 persons**

match(p:Person) return p limit 5

**Find Person who were born in the 1980s and return the name and born properties in descending order by the born property**

match(p:Person{ born:1980}) return p.name as name , p.birthdate as dob order by dob desc

**Find the director of the movie The Matrix, and return the name property of the directors**

match(m:Movie { title:’The Matrix’})<-[:DIRECTED]-(d:Person) return d.name as Director

**what movies the Tom Hanks was acted in or directed?**

match(:Person { name : 'Tom Hanks'})-[r]->(m:Movie) return type(r) as type , m.title as movie

**Find all relationship to the Tom Hanks expect ACTED\_IN**

match(:Person { name: 'Tom Hanks'})-[r:**!**ACTED\_IN]-(m) return type(r) as type , m.title as movie

**Find the shortest path between the Tom Hanks and Keanu Reeves**

match p = **shortestPath**((:Person {name:'Keanu Reeves'})-[**\***]-(:Person{name:'Tom Hanks'})) return p

**Recommend co-actors for Keanu Reeves, who has yet to work with but who his who co-actors have worked with. Then order the result by how frequently a matched co-co-actor has collaborated with one of Keanu Reeves’ co-actors.**

match(keanu:Person{name:'Keanu Reeves'})-[:ACTED\_IN]->(:Movie)<-[:ACTED\_IN]-(coActors:Person) **,** (coActors:Person)-[:ACTED\_IN]->(:Movie)<-[:ACTED\_IN]-(cocoActors:Person) where not (keanu)-[:ACTED\_IN]->()<-[:ACTED\_IN]-(cocoActors) and keanu **<>** cocoActors return cocoActors.name as recommended , count(cocoActors) as strength order by strength desc limit 7

**Consider there are two movies that Tom Hanks and Keanu Reeves have never worked together. Recommend two co-actors who have worked with both of them in separate movies.**

match(tom:Person{name:'Keanu Reeves'})-[:ACTED\_IN]->(:Movie)<-[:ACTED\_IN]-(coActor:Person), (coActor)-[:ACTED\_IN]->(:Movie)<-[:ACTED\_IN]-(:Person{name:'Tom Hanks'}) return distinct coActor.name as coActor

**Find the People who have DIRECTED and ACTED\_IN relationship**

match(p:Person)-[r:DIRECTED|ACTED\_IN]-(:Movie) return p

CREATE

(alice:Person {name:'Alice', age: 38, eyes: 'brown'}),

(bob:Person {name: 'Bob', age: 25, eyes: 'blue'}),

(charlie:Person {name: 'Charlie', age: 53, eyes: 'green'}),

(daniel:Person {name: 'Daniel', eyes: 'brown'}),

(eskil:Person {name: 'Eskil', age: 41, eyes: 'blue'}),

(alice)-[:KNOWS]->(bob),

(alice)-[:KNOWS]->(charlie),

(bob)-[:KNOWS]->(daniel),

(charlie)-[:KNOWS]->(daniel),

(bob)-[:MARRIED]->(eskil)

**Find People and return 1 for blue eyes ,2 for brown eyes,3 for others also eyes property**

match(p:Person) where p.eyes <> 'null' return distinct case p.eyes when 'blue' then 1 when 'brown' then 2 else 3 end as result , p.eyes

**Find People who has friend ( IS\_FRIENDS\_WITH ) with Jennifer and return name property**

match(:Person {name :'Jennifer'})-[:IS\_FRIENDS\_WITH]->(friend:Person) return friend.name

//or

match(jennifer:Person {name:'Jennifer'}) match(jennifer)-[:IS\_FRIENDS\_WITH]->(friend) return friend.name as name

**Create a new person and make a relationship as friendship with an existing person**

match(ali:Person { name :'Ali Montazeri'}) create (ali)-[:IS\_FRIENDS\_WITH]->(fatemeh:Person {name:'Fatemeh Montazeri'}) return ali , Fatemeh

Make friendship between two existing persons

match(arezo:Person {name:'Arezo Montazeri'}) match (ali:Person{name : 'Ali Montazeri'}) create (arezo)-[:IS\_FRIENDS\_WITH]->(ali) return arezo , ali

**Note**: Every time you run the above query, It creates relationship again

**Do this in a query**

**Create a new Person named Jay-jay and return the count and return the count of people whose name starts with J**

create(:Person {name:'Jay-jay'}) return count(\*) as count **union** match(p:Person) where p.name starts with 'J' return count(\*) as count

**Find all friendship of Jennifer and add them to the friend’s property of the node John**

match(jennifer:Person {name:'Jennifer'}) set jennifer.friends = [] with john match (jennifer)-[:IS\_FRIENDS\_WITH]->(friend:Person) call { with jennifer, friend set jennifer.friends = jennifer.friends + friend.name } return Jennifer

**Find all people or Movie**

match(n:People | Movie) return n.name , n.title

**Find all Tom Hanks’ Movies with outgoing relationships**

match(p:Person{name:'Tom Hanks'})**-->**(movie) return movie.title

**Find all Tom Hanks’ Movies with unidirectional relationship**

match(p:Person{name:'Tom Hanks'})***--***(movie) return movie.title

**Match on multiple relationship types**

**Find Tom Hanks’ movies he was acted or directed**

match(p:Person{name:'Tom Hanks'})**-**[:ACTED\_IN **|** DIRECTED]**->**(movie) return movie.title

match(p:Person{name:'Tom Hanks'})**-**[:ACTED\_IN**|**DIRECTED]**-**(movie) return movie.title

**Relationship types with uncommon characters – (using** backticks (`)**)**

**Make a relationship as OLD FRINDS between Martin Sheen and Rob Reiner**

match(p1:Person {name :'Martin Sheen'}),match(p2:Person{ name:'Rob Reiner'}) create (p1)-[:**`OLD FRINDS`**]->(p2) return p1,p2

**Multiple relationships**

**Find Charlie Sheen’s movies title and their director’s name**

match(actor:Person { name:'Charlie Sheen'})-[:ACTED\_IN]->(movie:Movie)<-[:DIRECTED]-(director:Person) return movie.title as movie , director.name as director

**Optional match**

**What movies directed by Martin Sheen?** (use optional and optional match)

match(p:Person { name :'Martin Sheen'}) optional match(p)-[r:DIRECTED]->() return p.name , r

**Optional relationships**

**Find all relationships of the movie Wall Street** (use match and optional match)

match(m:Movie { title:'Wall Street'}) **optional** match (m)-->(x) return x

match(m:Movie { title:'Wall Street'}) match (m)-->(x) return x

**Find the movie Man of Tai Chi and return title property and list of other relationships( with movies and persons) in a column**

match(m:Movie { title:'**Man of Tai Chi'**}) return m.title , [p=(m)**--**(x) | p ] as result

**Unique result**

**Find all actors that acted at least one movie**

match(p:Person)-[:ACTED\_IN]->(x) return **distinct** p.name

**Return the name of person connected to Daniel whose name starts with a C, regardless of capitalization.**

match(george:Person { name:'Daniel'})<--(person) with toupper(person.name) as name where name =~'C.\*' return name

**Using the wildcard to carry over variables**

**Find all people and return all properties regardless of type of their relationships**

match(person:Person)-[r]->(otherPerson:Person) with **\*** , type(r) as type return person.name ,otherPerson.name, type

**List the names of people in reverse order, limited to 3, is returned in a list.**

match(p:Person) with p **order by** p.name desc **limit** 3 return **collect**( p.name )

**Starting at Jennifer, find all matching nodes, order by name descending and get the top result, then** **find all the nodes connected to that top result, and return their names.**

match(person:Person{ name:'Jennifer'})--() **with** person order by person.name desc **limit** 1 match(person)--(other) return other.name

**Make a list from two list [1,2] and [3,4] and members in individual rows**

with [1,2] as a , [3,4] as b unwind (a+b) as c return c

**UNWIND [] reduces the number of rows to zero**

UNWIND [] AS empty

RETURN 'literal\_that\_is\_not\_returned'