Date: M T W T F S S

MATCH

```
MATCH (P:Person)
RETURN p.
```

RETURN

```
WITH 10 as x
RETURN x.
```

OPTIONAL MATCH

e.g.

~~MATCH~~

```
OPTIONAL MATCH (p:Person {name:'TomHanks'}) - [:DIRECTED] → (m:Movie).
RETURN m.title
```

→ returns Movie Tom Hanks directed (or null).

WHERE

```
MATCH (m:Movie)
WHERE m.release_year = 2000
RETURN m.title.
```

STRINGS

Q) Find actors whose names contain 'John' (case sensitive).

~~MATCH (m:Movie) WHERE m.title~~

- → STARTS WITH
- → CONTAINS
- → tolower(...)
- → toUpper(...).

```
MATCH (p:Person) WHERE tolower(p.name) CONTAINS "john"
RETURN *
```

DISTINCT

ORDER BY

```
MATCH (m:Movie)
RETURN m.title, m.release_year
ORDER BY m.release_year DESC
```

LIMIT/SKIP.

```
MATCH (m:movie)
RETURN m.title
SKIP 10 LIMIT 5
```

* ~~skips first 10 movies and~~ returns next 5 (pagination)

COLLECT

```
MATCH (m:Movie) <-[:ACTED_IN]- (a:Person)
~~RETURN m.title~~, collect(a.name) AS Actors.
```

* aggregates result into list.

COUNT, MIN, MAX

```
MATCH (m:Movie)        —①
RETURN count(m).
```

```
MATCH (M:Movie)
RETURN $MIN(m.release_year), MAX(m.release_year)
```

* returns oldest and newest movie release year.

WITH

```
MATCH (m:Movie)
WITH m ORDER BY m.release_year DESC LIMIT 10
RETURN m.title.
```

* First sorts movies by release_year, then select latest 10.

Date:
M T W T F S S

CREATE

→ nodes.

CREATE (n:Person {name: 'Steve'})

~~→ nodes and~~ relationships

CREATE (n:Person {name: John}) -[:KNOWS]→ (n:Person {name: Ali}).

MERGE

MERGE (n:Person {name: John})
MERGE (n:Person {name: John}) -[:KNOWS]→ (n:Person {name: Ali})

UPDATE

→ SET

① properties.

MATCH (p:Person {name: John})
~~SET p.age = 30.~~

② array.

MATCH (p:Person {name: John})
~~SET person.interests = person.interests +~~ "Cooking"

→ Update label.

MATCH (p:Person {ID: 123})
SET p: Employee.

→ REMOVE

① properties:

MATCH (p:Person {name: John})
REMOVE p.age.

② labels.

MATCH (p: Person {ID: 123})
REMOVE p:Person
RETURN p.

DELETE

① relationship.

MATCH (p:Person) -[r:KNOWS]→ (x:Person)
DELETE r.

② node only.

MATCH (p: Person)
WHERE p.age > 60
DELETE p.

③ deleting node with relationship.

MATCH (p: person {name: John})
DETACH DELETE p.

PROFILE MATCH

*see performance of keyword.

PROFILE MATCH (p:Person)
RETURN *

DEGREE

Q) Total no of relationship connected to each node.

```
MATCH(n)
OPTIONAL MATCH (n)-[r]-()
RETURN n, COUNT(r).
```

IN-DEGREE

```
MATCH (n)
OPTIONAL MATCH (n)<-[r]-()
RETURN n, COUNT(n).
```

OUT-DEGREE

```
MATCH(n)
OPTIONAL MATCH (n)-[r]->()
RETURN n, count(r).
```

Date:_____

DENSITY OF GRAPH

```
MATCH (m)
OPTIONAL MATCH (m)-[r]-()
WITH count(distinct m) as V, count(distinct r) as E
RETURN V, E, toFloat(E)/((V*(V-1))/2).
```

DIAMETER

```
MATCH (start)
MATCH (end)
WHERE start <> end
MATCH p=shortestPath((start)-[*]-(end))
RETURN max(length(p)) AS diameter.
```

AVERAGE PATH LENGTH

```
MATCH (start)
MATCH (end)
WHERE start <> end
MATCH p= ShortestPath((start)-[*]-(end))
RETURN avg(length(P)) as AVG_PATH_LEN.
```

CLUSTERING COEFFICIENT

1) Count no of unique neighbours.

```
MATCH (m)
OPTIONAL MATCH (m)-[]-(n)
RETURN m, COUNT(distinct n) as K.
```

2) Calculate no of edges among friends.

```
MATCH(n)
OPTIONAL MATCH (n)-[]-(m).
WITH n, count(distinct m) as k.
OPTIONAL MATCH (n)-[]-(m)-[r]-(p)-[]-(n).
RETURN n, k, count(distinct r) as L
```

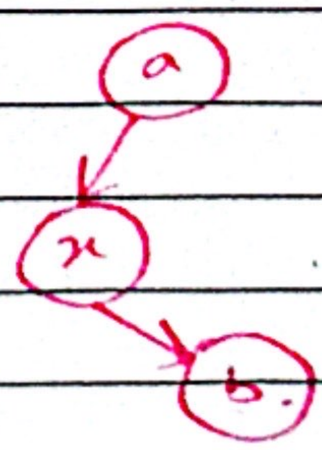
Q) Apply formula.

```
MATCH (n)
OPTIONAL MATCH (n)-[]-(m)
WITH n, count(distinct m) as K
OPTIONAL MATCH (n)-[]-(m)-[r]-(p)-[]-(n).
WITH n, k, count(distinct r) as L.
RETURN n, K, L
    CASE WHEN k<2 THEN 0
    ELSE (2*toFloat(L))/(k*(k-1))
    END as CC.
```

RECOMMENDATION

Q) Recommend friends for node A.

```
MATCH(a {name: "A"})-[:FOLLOWS]-(x)-[:FOLLOWS]-(b)
WHERE NOT (a)-[:FOLLOWS]-(b)
RETURN a,b.
```



FRIENDS OF FRIENDS

Q) list names of my friends friends.

```
MATCH (m)
MATCH (n)
MATCH (n)--(m)--(x)
WHERE n<>m and m<>x
RETURN n, collect(distint x.Name).
```

FINDING TRIANGLES

```
MATCH (a)-[:CONNECTED]->(b)-[:CONNECTED]->(c)-
      [:CONNECTED]->(a)
WHERE a<>b AND b<>c a<>c.
RETURN a,b,c.
```

Date:_____

AVERAGE NO OF FRIENDS OF MY FRIENDS

```
MATCH(m)
OPTIONAL MATCH (m)--(f)
WITH m,f
OPTIONAL MATCH (f)--(ff)
WITH m, count(distinct f) as F , count (ff) AS FF.
return m.Name , F,
    CASE WHEN F>0 THEN round(toFloat(FF)/F,2)
    ELSE 0
    END AS AVG.
```

NODE OUT-DEGREE

Q) using apoc.agg.statistics

```
MATCH (u:User)
WITH u , size((u)-[:FOLLOWS]->())
    as outdegree
RETURN apoc.agg.statistics(outdegree)
```

Q) using size() operator.

```
MATCH (u:User)
RETURN u.username as user
    size((u)-[:FOLLOWS]->())
    as outDegree
ORDERBY outdegree DESC
LIMIT 5.
```

DEGREE DISTRIBUTION OF GRAPH

Q) write query for counting how many nodes have each degree.

```
MATCH(m)
OPTIONAL MATCH (m)-[r]-()
WITH m, count(r) as degree.
RETURN degree, count(degree)
ORDER BY degree.
```