1. IN,AND,OR operation
2. Where
3. Set,remove
4. Merge ,on create, on match
5. Aggregate
6. Skip,limit,order by
7. Union,union all
8. String expression
9. Incoming, outgoing
10. Variable length, shortest path, allpair shortest path,

Graph Data :

```
CREATE
  (e1:Employee {name: "Ram", age: 35, role: "Manager"}),
  (e2:Employee {name: "Shyam", age: 28, role: "Developer"}),
  (e3:Employee {name: "Amit", age: 30, role: "Developer"}),
  (e4:Employee {name: "Sita", age: 32, role: "Analyst"}),

  (d1:Department {name: "IT"}),
  (d2:Department {name: "HR"}),
  (d3:Department {name: "Finance"}),

  (e1)-[:Manages]->(d1),
  (e1)-[:Manages]->(d3),
  (e2)-[:Works_In]->(d1),
  (e3)-[:Works_In]->(d1),
  (e4)-[:Works_In]->(d2);
```

Lets take scenario: (Task)

Three Students(Utkarsh,Anmol, Prashant)
Two dept(CS,MATH)
Utkarsh study on CS
Anmol study on MATH
Prashant take Lab classes both CS and MATH dept
Relationship are study and take classes

Scenario:

Chrish works for scope and jeff also, make this in one query

Utkarsh and anmol works for MATH and Utkarsh ,Prashant take lab classes to CS-create a full path

Order,skip,limit:
match(e:Employee) return e.name as name order by name skip 1 limit 3

union and union all:

- **`UNION`** → Removes duplicates (returns distinct results).
- **`UNION ALL`** → Includes duplicates (keeps all rows).

match(e:Employee) return e.name union match(e:Employee) return e.name
match(e:Employee) return e.name union all match(e:Employee) return e.name

Aggregation :

match(e:Employee) return min(e.salary),max(e.salary)
match(n:Employee) return count(*)

Merge:
`MERGE` ensures that the relationship is **created only once** and avoids duplicate edges between the same nodes. If the relationship already exists, `MERGE` prevents adding a duplicate

- **Creating it if it does not exist** (`ON CREATE`).
- **Matching an existing node/relationship** (`ON MATCH`).

```
MERGE (p:Person {name: "Tom"})
ON CREATE SET p.age = 30
RETURN p;
```

Effect:

- If `Tom` does **not exist**, it **creates** `(:Person {name: "Tom", age: 30})`.
- If `Tom` **exists**, it does **nothing**

MERGE (p:Person {name: "Tom"})

ON MATCH SET p.age = p.age + 1

RETURN p;

Effect:

- If `Tom` exists, it **increments his age**.
- If `Tom` does not exist, it **creates** him but does not modify age.

MERGE (p:Person {name: "Tom"})

ON CREATE SET p.age = 30

ON MATCH SET p.lastUpdated = timestamp()

RETURN p;

Effects:

- If `Tom` is **created**, he gets `age = 30`.
- If `Tom` **already exists**, `lastUpdated` is set.

Merge relationship:

MATCH (a:Person {name: "Tom"}), (b:Person {name: "Ram"})

MERGE (a)-[r:FOLLOWS]->(b)

ON CREATE SET r.since = date()

RETURN r;

Effect:

- If `(:Tom)-[:FOLLOWS]->(:Ram)` exists, it is **matched**.
- If it does not exist, it is **created with a timestamp**.

```
// Creating Nodes & Relationships in One Step
MERGE (chris:Person {name: "Chris"})
```

```
MERGE (jeff:Person {name: "Jeff"})
MERGE (satish:Person {name: "Satish"})
MERGE (mohan:Person {name: "Mohan"})
MERGE (sam:Person {name: "Sam"})
MERGE (david:Person {name: "David"})
MERGE (anil:Person {name: "Anil"})
MERGE (tom:Person {name: "Tom"})
MERGE (venkat:Person {name: "Venkat"})
MERGE (ram:Person {name: "Ram"})

// Ensuring All 11 Relationships Exist
MERGE (chris)-[:FOLLOWS]->(jeff)
MERGE (jeff)-[:FOLLOWS]->(satish)
MERGE (satish)-[:FOLLOWS]->(anil)
MERGE (satish)-[:Follows]->(mohan)
MERGE (satish)-[:Follows]->(david)
MERGE (mohan)-[:FOLLOWS]->(sam)
MERGE (mohan)-[:FOLLOWS]->(venkat)
MERGE (sam)<-[:FOLLOWS]-(david)
MERGE (tom)-[:FOLLOWS]->(chris)
MERGE (ram)-[:FOLLOWS]->(tom)
MERGE (sam)-[:FOLLOWS]->(ram)
```

merge (c:Clerk) return c
merge(c:Clerkd) on create set c.name="oncreate" on match set c.name="onmatch"
return c

String and Case Expression:
uppercase
MATCH (p:Person)
RETURN p.name, toUpper(p.name) AS UpperCaseName;

First 3 letter of each name
MATCH (p:Person)
RETURN p.name, substring(p.name, 0, 3) AS ShortName;

Replace :
MATCH (p:Person)
RETURN p.name, replace(p.name, "a", "X") AS ModifiedName;
Contains:
MATCH (p:Person)

```
WHERE p.name CONTAINS "a"
RETURN p.name AS NameWithA;
```

Startswith:

```
MATCH (p:Person)
WHERE p.name STARTS WITH "S"
RETURN p.name AS StartsWithS;
```

Endswith:

```
MATCH (p:Person)
WHERE p.name ENDS WITH "n"
RETURN p.name AS EndsWithN;
```

```
MATCH (p:Person)
WHERE p.name STARTS WITH "S" AND p.name CONTAINS "a"
RETURN p.name AS MatchedNames;
```

Case expression:

```
MATCH (p:Person)
RETURN p.name,
  CASE
    WHEN length(p.name) <= 4 THEN "Short Name"
    WHEN length(p.name) <= 6 THEN "Medium Name"
    ELSE "Long Name"
  END AS NameCategory;
```

```
MATCH (p:Person)
RETURN p.name,
  CASE
    WHEN p.name STARTS WITH "S" THEN "Starts with S"
    WHEN p.name STARTS WITH "J" THEN "Starts with J"
    ELSE "Other"
  END AS Category;
```

Aggregation :

```
MATCH (p:Person)
RETURN COUNT(p) AS TotalPeople;
```

```
MATCH ()-[r:FOLLOWS]->()
```

```
RETURN COUNT(r) AS TotalFollows;

MATCH (p:Person)
RETURN MIN(LENGTH(p.name)) AS ShortestName, MAX(LENGTH(p.name)) AS
LongestName;

MATCH (p:Person)
RETURN AVG(LENGTH(p.name)) AS AvgNameLength;

MATCH (p:Person)
RETURN COLLECT(p.name) AS NameList;

CREATE (a:Person {name: "Tom", age: 30}),
    (b:Person {name: "Ram", age: 25}),
    (c:Person {name: "Venkat", age: 35}),
    (d:Person {name: "Satish", age: 40}),
    (e:Person {name: "Mohan", age: 30}),
    (f:Person {name: "Tom", age: 30})  // Duplicate node

MATCH (p:Person)
WHERE p.age > 30
RETURN p.name AS Name, p.age AS Age
UNION
MATCH (p:Person)
WHERE p.name = "Tom"
RETURN p.name AS Name, p.age AS Age;

MATCH (p:Person)
WHERE p.age > 30
RETURN p.name AS Name, p.age AS Age
UNION ALL
MATCH (p:Person)
WHERE p.name = "Tom"
RETURN p.name AS Name, p.age AS Age;
```

Indegree, outdegree:

```
MATCH (p:Person)
RETURN p.name, SIZE((p)<--()) AS Indegree;

MATCH (p:Person)
```

```
RETURN p.name, SIZE((p)-->()) AS Outdegree;

MATCH (p:Person)
RETURN p.name, SIZE((p)<--()) AS Indegree, SIZE((p)-->()) AS Outdegree;
```

Using `COUNT()` with `MATCH`

An alternative approach using `COUNT()` instead of `SIZE()`.

```
MATCH (p:Person)<-[r]-()

RETURN p.name, COUNT(r) AS Indegree;


MATCH (p:Person)-[r]->()

RETURN p.name, COUNT(r) AS Outdegree;


MATCH (p:Person)

OPTIONAL MATCH (p)<-[r1]-()

OPTIONAL MATCH (p)-[r2]->()

RETURN p.name, COUNT(r1) AS Indegree, COUNT(r2) AS Outdegree;
```

Why use `OPTIONAL MATCH`?

- Ensures nodes without relationships still appear in the results instead of being ignored.

```
Variable length relationship :
 Graph Data :
```

```
// Creating Nodes & Relationships in One Step
MERGE (chris:Person {name: "Chris"})
MERGE (jeff:Person {name: "Jeff"})
MERGE (satish:Person {name: "Satish"})
MERGE (mohan:Person {name: "Mohan"})
MERGE (sam:Person {name: "Sam"})
MERGE (david:Person {name: "David"})
MERGE (anil:Person {name: "Anil"})
MERGE (tom:Person {name: "Tom"})
MERGE (venkat:Person {name: "Venkat"})
MERGE (ram:Person {name: "Ram"})

// Ensuring All 11 Relationships Exist
MERGE (chris)-[:FOLLOWS]->(jeff)
MERGE (jeff)-[:FOLLOWS]->(satish)
MERGE (satish)-[:FOLLOWS]->(anil)
MERGE (satish)-[:FOLLOWS]->(mohan)
MERGE (satish)-[:FOLLOWS]->(david)
MERGE (mohan)-[:FOLLOWS]->(sam)
MERGE (mohan)-[:FOLLOWS]->(venkat)
MERGE (sam)<-[:FOLLOWS]-(david)
MERGE (tom)-[:FOLLOWS]->(chris)
MERGE (ram)-[:FOLLOWS]->(tom)
MERGE (sam)-[:FOLLOWS]->(ram)
```

Question: who are the first label connectens of user Satish?

```
match(p:Person{name:"Satish"})-[*1]-(p1:Person) return p,p1
```

Who are the 2nd label connections of Satish ?

```
match(p:Person{name:"Satish"})-[*2]-(p1:Person) return p,p1
```

Who are 3rd label connection of Satish ?

```
match(p:Person{name:"Satish"})-[*3]-(p1:Person) return p,p1
```

Who are all the first and 2nd label connections of Satish ?

```
match(p:Person{name:"Satish"})-[*1..2]-(p1:Person) return p,p1
```

What is the shortest path between Satish and sam ?

```
match(p1:Person{name:"Satish"}),(p2:Person{name:"Sam"}), p=shortestpath((p1)-
[:FOLLOWS*1..2]->(p2)) return p
```

What are the shortest paths between Satish and sam ?

```
match(p1:Person{name:"Satish"}),(p2:Person{name:"Sam"}), p=allshortestpaths((p1)-[:FOLLOWS*1..2]->(p2)) return p
```