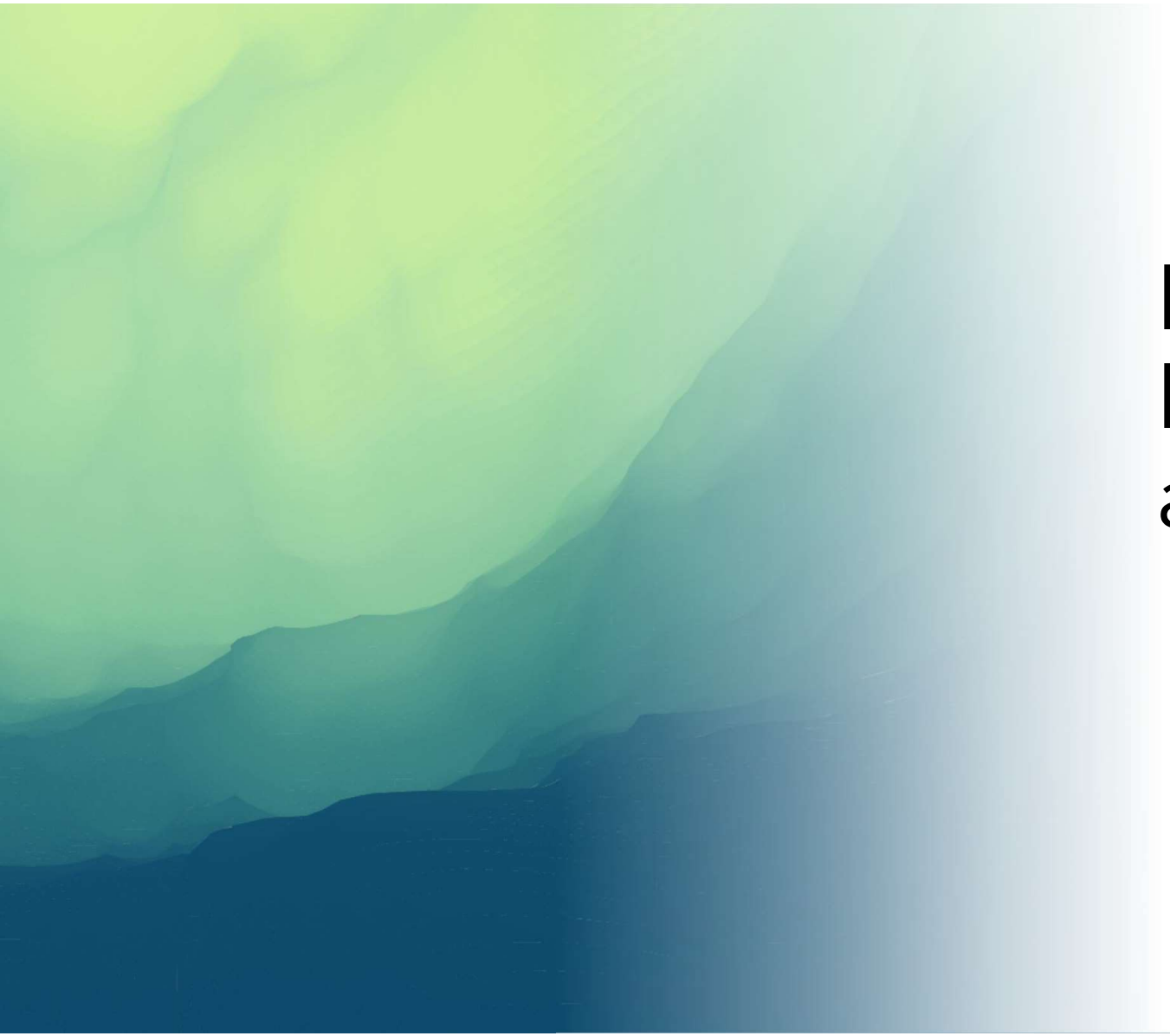


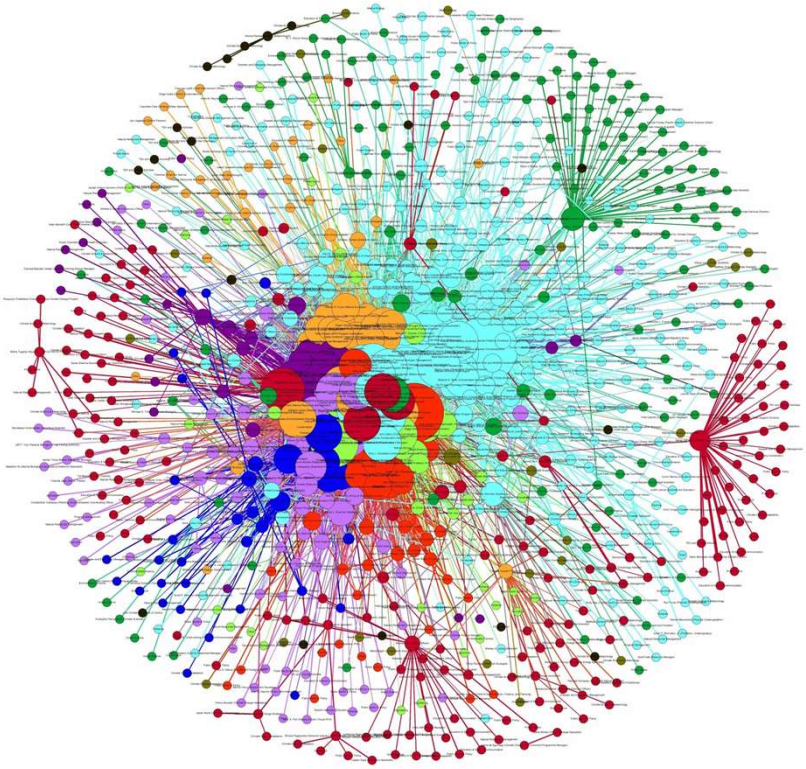
No-SQL Databases and DBMS

No-SQL (Not Only SQL) Databases

- Graph database
  - Neo4j
- Document database
  - MongoDB
- Key-value stores
  - E.g., Riak, Redis
- Wide-column stores
  - E.g., Hbase, Cassendra

Real-world data seen as graphs

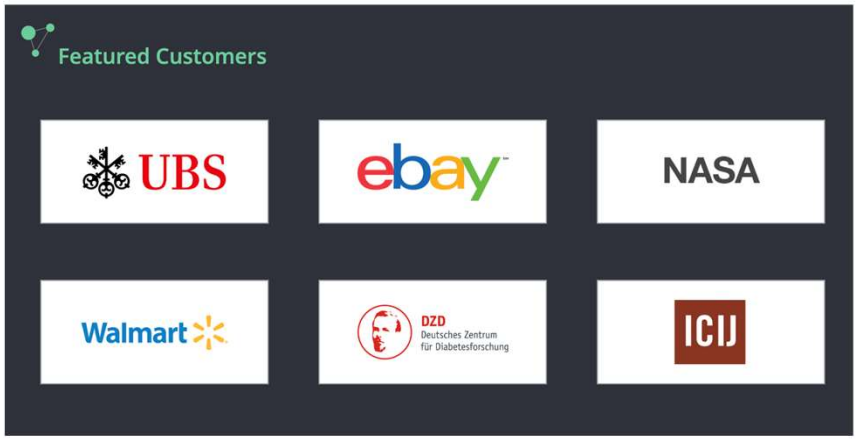
- Social networks
  - Facebook, LinkedIn, PayPal



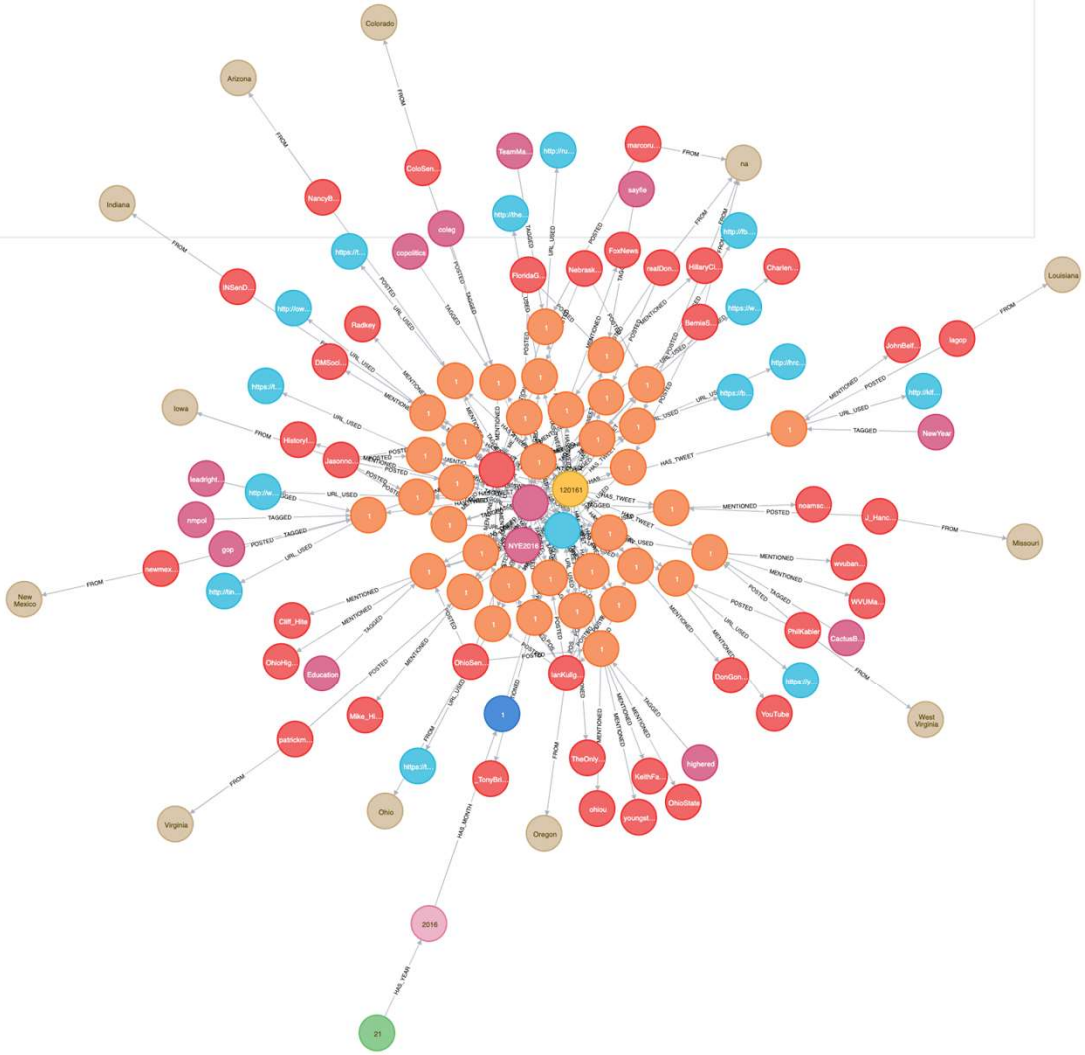
When to use Graph DBMS

- Queries focusing more on relationships
- Frequent queries involve
  - starting from specific node(s) and traverse graphs
  - ending at specific node(s) and traverse back
- Visualization of relationships provides more information

neo4j

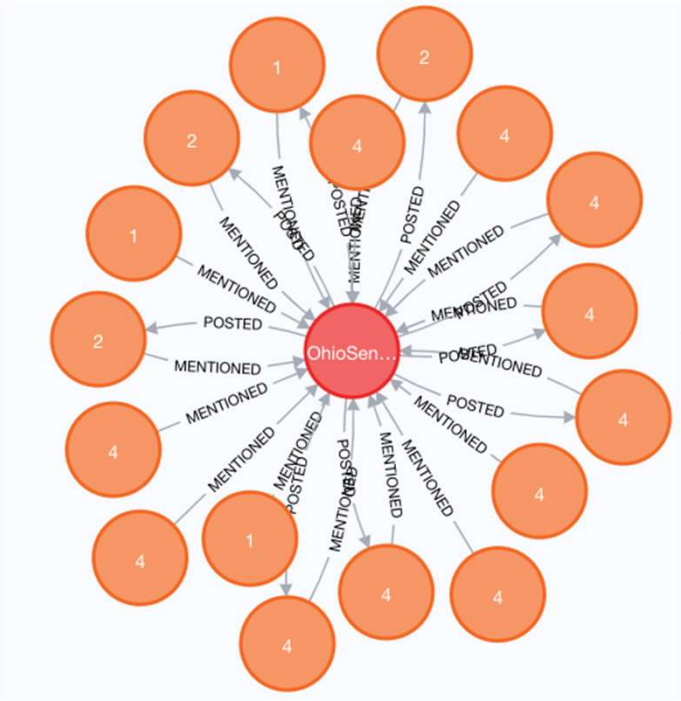


customers



Graph Model in Neo4j

- Collection of nodes and edges with properties
  - Properties are key-value pairs
- A node represents one instance
- An edge represents a relationship between two nodes
- Query language: Cypher



- Label: Nodes/ edges with almost the same properties grouped together

User <id>: 1 **category:** senate_group **followers:** 7271 **following:** 397 **location:** Ohio **name:** OhioSenateGOP

Tweet <id>: 0 **created_at:** 1.451606453e+12 **day:** 1 **id:** 682713066287566848 **month:** 1 **retweet_count:** 1

POSTED <id>: 0 **MENTIONED** <id>: 403561

Neo4j advantage

- Flexible data model
- Connected and semi structures data
  - Easily represent connected and semi-structured data
- Neo4j cypher query language commands
  - Human readable format and easy to learn
- No Joins
  - Retrieve its adjacent node or relationship details by traversing graph

Cypher Query Language (CQL)

- Type: boolean, byte, short, long, int, char, string, float, double
- You do not need to specify type when writing your Cypher
- Assign nodes/edges different label to group them together

Cypher: Create a node

- CREATE (node) // node is variable to represent this node
- CREATE (node1),(node2) // node1 and node2 are variables to represent the two empty nodes
- CREATE (node:Person) // create a node with label Person
- CREATE (node:Person:Player) // create a node with two labels Person and Player

Cypher: Create a node with Properties

- CREATE (p1: Person {name:'Alice', YOB:1999}) // create a node of label Person with two attributes name and YOB
- CREATE (p2: Person {name:'Bob', YOB: 1998})
- p1 and p2 are variables to specify the node instance
- Name and YOB are attributes with corresponding values
- Node variables are defined inside the parentheses ()

Cypher: Create a relationship

- CREATE (node1)-[:RelationshipType]->(node2)
- CREATE (p1)-[k:KNOWS]->(p2)
- p1 and p2 are nodes
- KNOWS is the relationship label
- Arrow specify the direction
- k here is the edge variable, edge variable is defined inside the brackets []

Cypher: Create a relationship between existing nodes

- Use MATCH statements to find out existing nodes
- Use CREATE statement to create a relationship
- MATCH (a:Person), (b:Person)
  WHERE a.name = "Alice" AND b.name = "Bob"
  CREATE (a) – [:Friend] – (b) // only create the relationship between "Alice" and "Bob"

Cypher: Match clause

- match (n) return n // retrieve all nodes
- match (n:Person) return n // retrieve all nodes label Person
- match (n) where n.name is null return n // find nodes with undefined name attribute
- match (n) where id(n)=1 return n // find nodes where the system created id of the node is 1

Cypher: Deletion

- match (n) where n.name is null delete n // delete nodes with undefined name
- match (n) detach delete n // delete all nodes
- match (n) optional match (n)-[r]-() delete n, r // delete all nodes and edges
- match (n) remove n:Person // remove Person label
- Delete the entire database: go to the directory and delete the folder

Cypher: Order by/ limit / Skip

- match (n) return n order by n.name DESC// order nodes by name attribute in descending order
- match (n) return n order by n.name, n.YOB // order nodes first by name attribute then by YOB
- match (n) return n order by n.name skip 1 limit 1 // skip the first result and return one record

Cypher: With

- Chain query different parts together
  - One part computes some data and the next query part can use that computed data
- match (a:Person{name:"Alice"}) – [:Friend] –(b:Person)

  with distinct b

  order by b.name limit 10

  match (b)-[:From]->(c:Country)

  return c.name

Cypher: Count / Max / Min / Avg / Sum

- match (n:Person) -->(x) return n, count(x) // return a node label Person and the number of nodes connected to it
- match (n:Person) return max(n.age) // return the oldest person
- match (n:Person) return min(n.age) // return the youngest person
- match (n:Person) return avg(n.age) // return the average age of all
- match (n:Person) return sum(n.salary) // return the total salary

Cypher: Collection

- Aggregates data by aggregating multiple records or values into a single list
- Any null values are ignored and will not be added to the list
- match (n:Person)

  where toInteger(n.age) > 20

  return collect(n.name) // returned result: [John, Alice, David, Bob]

Cypher: Not

- Find orphan nodes without incoming/outgoing edges of any relationship
- match (n)

  where NOT (n)-->()

  return n

Cypher: Query nodes N hops away

- Find friends of Alice and friends of friends of Alice

```
// return friends of Alice and friends of friends of Alice
match (n: Person {name: "Alice"})-[r: Friends*1..2]->(x) return n, x
```

- Find friends of friends of Alice or find 2-hop friends of Alice

```
// return friends of friends of Alice
match (n: Person {name: "Alice"})-[*2]->(x) return n, x
```

Cypher: Single shortest path

```
// return the shortest path between Alice and Jack
// maximum 10 hops
match (p1: Person {name: "Alice"}), (p2 : Person {name: "Jack"}),
p = shortestPath((p1)-[*..10]-(p2)) return p
```

Cypher: All shortest paths

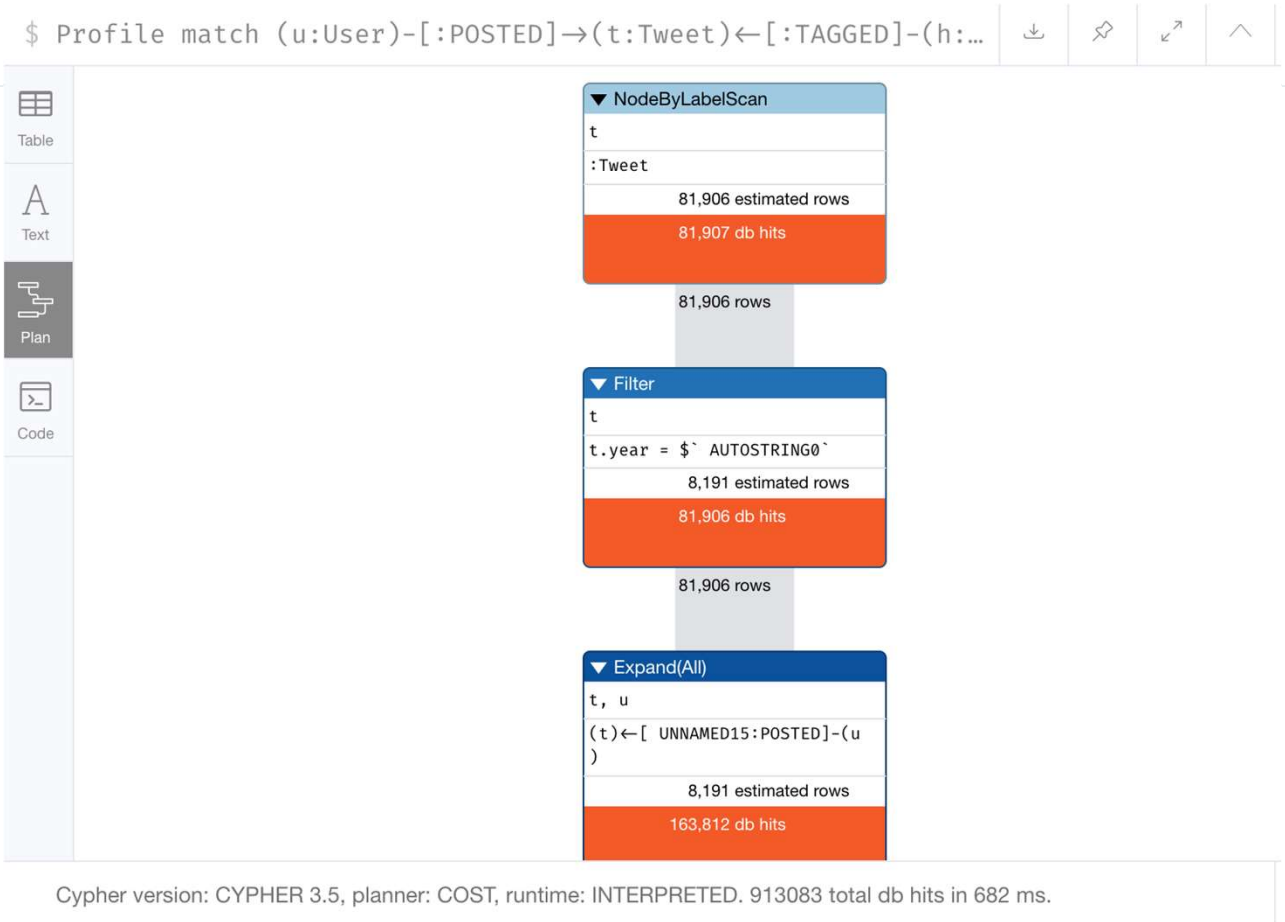
```
// return all shortest paths between Alice and Jack
// without hop number limitation
match (p1: Person {name: "Alice"}), (p2 : Person {name: "Jack"}),
p = allShortestPaths((p1)-[*]-(p2)) return p
```

Cypher: Explain / Profile

- View execution plan with keyword Profile or Explain

```
Profile match (u:User)-[:POSTED]->(t:Tweet)<-[:TAGGED]-(h:Hashtag)
where t.year='2016' and h.name <>''
return toUpper(h.name), count(distinct u.screen_name) as numOfUser,
collect(distinct u.screen_name) order by numOfUser desc limit 5
```

Cypher: Explain / Profile



Comparison

RDBMS	Graph Database
Tables	Graphs
Rows	Nodes
Columns and Data	Properties and its values
Constraints	Relationships
Joins	Traversal

References

- https://www.tutorialspoint.com/neo4j/index.htm
- Slides from Fall2019 ComS 363