Graph Databases 📈

Lecture plan

1. Many to many relationships
2. Graph databases
3. Neo4j

1️⃣ Many to many relationships

So far most of the data we have dealt with has had **one to one** or **one to many** relationships

Some data does not fit into these models and instead have **many to many** relationships

❓ Can you **think of any examples**?

🎯 Lets try modeling a many to many relationships in SQL

It will be a user and friends relationship here most users will have many friends on the platform which are other users!

Lets create a table `users` and a table `friends`

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) NOT NULL UNIQUE
);
CREATE TABLE friends (
  user_id INTEGER REFERENCES users(id),
  friend_id INTEGER REFERENCES users(id),
  PRIMARY KEY (user_id, friend_id)
);
```

Lets populate `users` with 10000 users

```
DO $$
DECLARE
  counter INTEGER := 0;
BEGIN
  WHILE counter < 10000 LOOP
    INSERT INTO users (id, name, email)
    VALUES (
      counter,
      'User ' || counter,
      'user' || counter || '@example.com'
    )
    ON CONFLICT (id) DO NOTHING;
    counter := counter + 1;
  END LOOP;
END $$;
```

Lets populate `friends` with 20000 random friendships!

```
DO $$
DECLARE
  counter INTEGER := 0;
  user1 INTEGER;
  user2 INTEGER;
  max_id INTEGER;
BEGIN
  SELECT INTO max_id MAX(id) FROM users;
  WHILE counter < 20000 LOOP
    user1 := FLOOR(RANDOM() * max_id) + 1;
    user2 := FLOOR(RANDOM() * max_id) + 1;
    IF user1 <> user2 THEN
      BEGIN
        INSERT INTO friends (user_id, friend_id)
        VALUES (LEAST(user1, user2), GREATEST(user1, user2))
        ON CONFLICT DO NOTHING;
        counter := counter + 1;
      EXCEPTION WHEN unique_violation THEN
      END;
    END IF;
  END LOOP;
END $$;
```

❓ How would you query these tables to find the friends of a user and friends of friends and friends of friends of friends etc ...

```sql
WITH RECURSIVE friends_cte(id, jumps) AS (
    SELECT friend_id, 1
    FROM friends
    WHERE user_id = 1
    UNION ALL
    SELECT f.friend_id, fc.jumps + 1
    FROM friends f
    INNER JOIN friends_cte fc ON f.user_id = fc.id
    WHERE fc.jumps < 3
)
SELECT id, jumps
FROM friends_cte;
```

This is pretty unintuitive 😅 lets try a different solution!

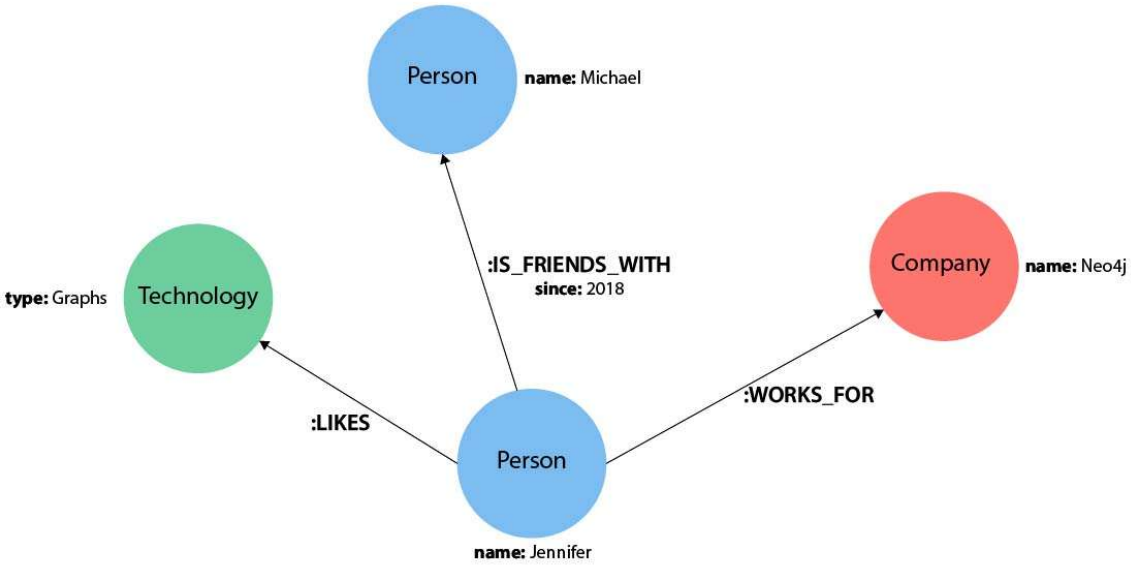
2️⃣ Graph Databases

e.g. Neo4j, Dgraph

The easiest way to know when to use graph databases is when the relationships are as important as the data itself!

Graph model components

- **Vertices** (also known as nodes or entities
- **Edges** (also known as relationships or arcs)
- **Properties** (extra information attached to the main two components)

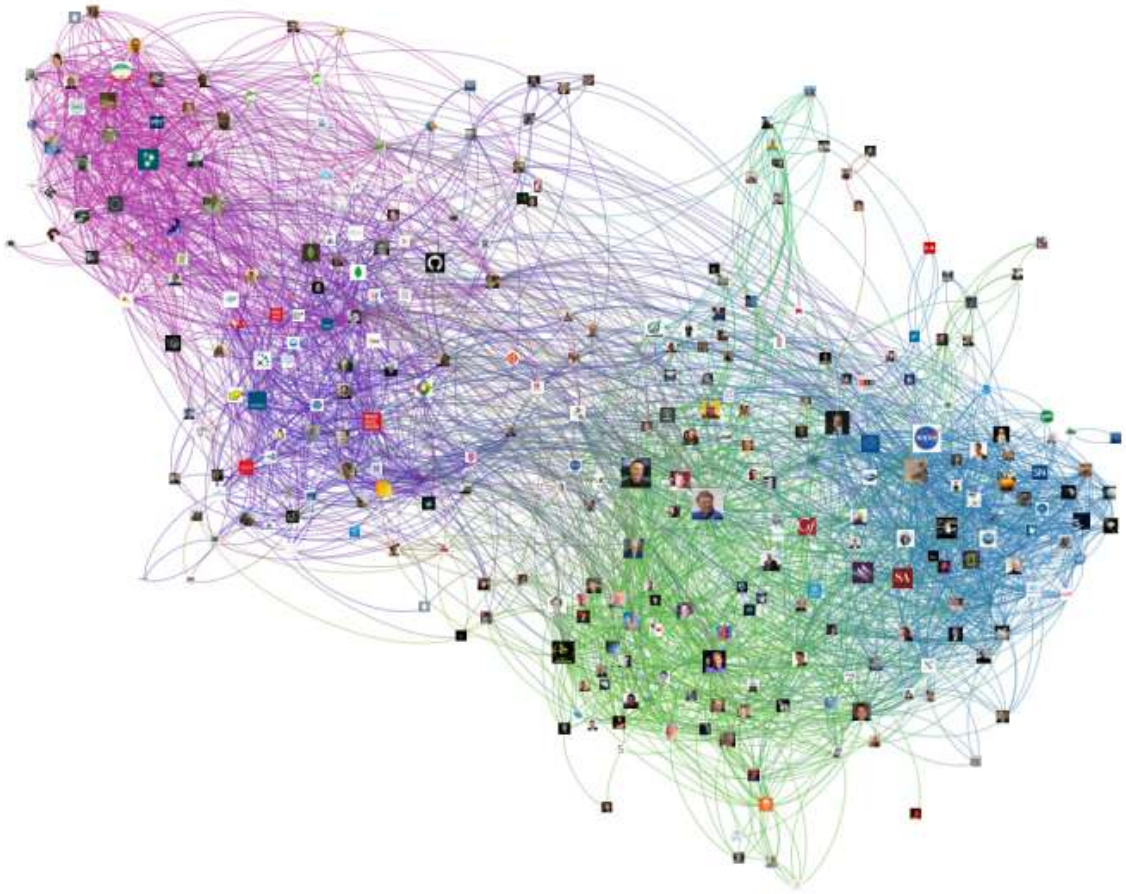


❓ For the following data what component do you think is appropriate?

- Users
- Date of Birth
- Which other users they follow
- Posts
- Likes
- Which users like the posts

A few main usecases

Social media and Social network graphs



Fraud Detection

Graph representations make it very clear **a pattern** is out of place

Lots of fraudulent transactions involve **many relationships** for example tracing money via many bank accounts!

More recently **graph machine learning** has improved fraud detection even more!

360 Customer view

A newer use case which is gaining a lot of popularity

The idea is **centralizing a customers data** along with bringing the important information as relationships can lead to much richer understanding of a customer and their needs!

Supply chain

Tracing goods through a supply chain is the perfect time to use a graph as it is a series of important relationships!

Become very popular with the rise of responsibility for your entire supply chain **legally and morally**

✋ Downsides of graph databases

- Scales poorly (jumping across servers really inefficient)
- Expensive
- Write operations slow
- Different language (easy enough to learn but restricts lots of people from your company)!

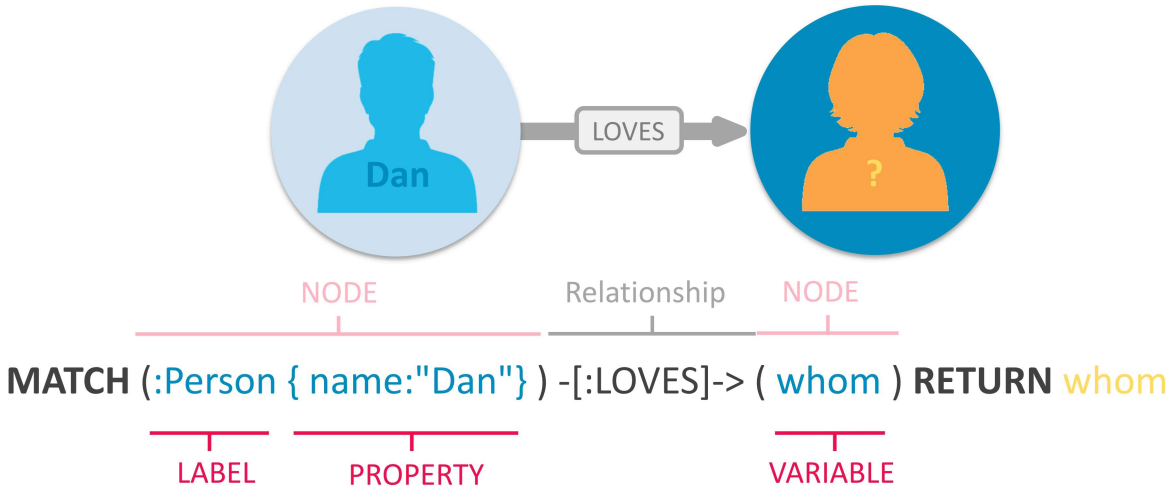
3️⃣ Neo4j

Neo4j is the **most popular graph database**

It uses its own query language **cypher**

It is **graph native** so efficient operations on graph problems!

Cypher query language



Testing neo4j with Docker 🐬

```
docker run \
    --name testneo4j \
    -p7474:7474 -p7687:7687 \
    -d \
    -v $HOME/neo4j/data:/data \
    -v $HOME/neo4j/logs:/logs \
    -v $HOME/neo4j/import:/var/lib/neo4j/import \
    -v $HOME/neo4j/plugins:/plugins \
    --env NEO4J_AUTH=neo4j/password \
    neo4j:latest
```

🎯 Lets redo the task of modelling friends using neo4j

Creating the users and friends

```
UNWIND range(0, 9999) AS counter
CREATE (:User {id: counter, name: 'User ' + counter, email: 'user' + counter +
'@example.com'});
UNWIND range(0, 19999) AS counter
WITH toInteger(rand() * 10000) AS user1, toInteger(rand() * 10000) AS user2
WHERE user1 <> user2
MERGE (u1:User {id: user1})
MERGE (u2:User {id: user2})
MERGE (u1)-[:FRIENDS]->(u2);
```

Same query as before to find friends **within two jumps of a user**

```
MATCH (user:User {id: 1})
MATCH (user)-[:FRIENDS*1..2]-(friend)
RETURN DISTINCT user, friend;
```

🤯 **Shortest path** between two different users

```
MATCH (start:User {id: 1})
MATCH (end:User {id: 2})
MATCH path = shortestPath((start)-[:FRIENDS*]-(end))
RETURN path
```

Info from python

```
from neo4j import GraphDatabase

class UserShortestPath:

    def __init__(self, uri, user, password):
        self.driver = GraphDatabase.driver(uri, auth=(user, password))

    def close(self):
        self.driver.close()

    def find_shortest_path(self, id1, id2):
        with self.driver.session() as session:
            path = session.execute_write(self._create_and_return_greeting, id1,
id2)
            path = path[0]["path"]
            for item in path:
                if item == "FRIENDS":
                    print(" "*5 + "|\n" + " "*2 + "FRIENDS\n" + " "*5 + "|")
                else:
                    print(item)



@staticmethod
    def _create_and_return_greeting(tx, id1, id2):
        result = tx.run("MATCH (start:User {id: $id1})"
                        "MATCH (end:User {id: $id2})"
                        "MATCH path = shortestPath((start)-[:FRIENDS*]-(end))"
                        "RETURN path", id1=id1, id2=id2)
        return result.data()


if __name__ == "__main__":
    greeter = UserShortestPath("bolt://0.0.0.0:7687", "neo4j", "password")
    greeter.find_shortest_path(1, 2)
    greeter.close()
```

📖 Appendix

- Great lecture (https://www.youtube.com/watch?v=yOYodfN84N4)
- Case studies (https://neo4j.com/use-cases/)
- JanusGraph optimized for huge databases (https://janusgraph.org/)