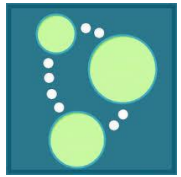
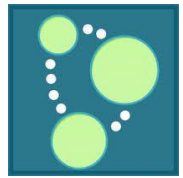
Neo4J

- *Priyanka Goyal*



Introduction

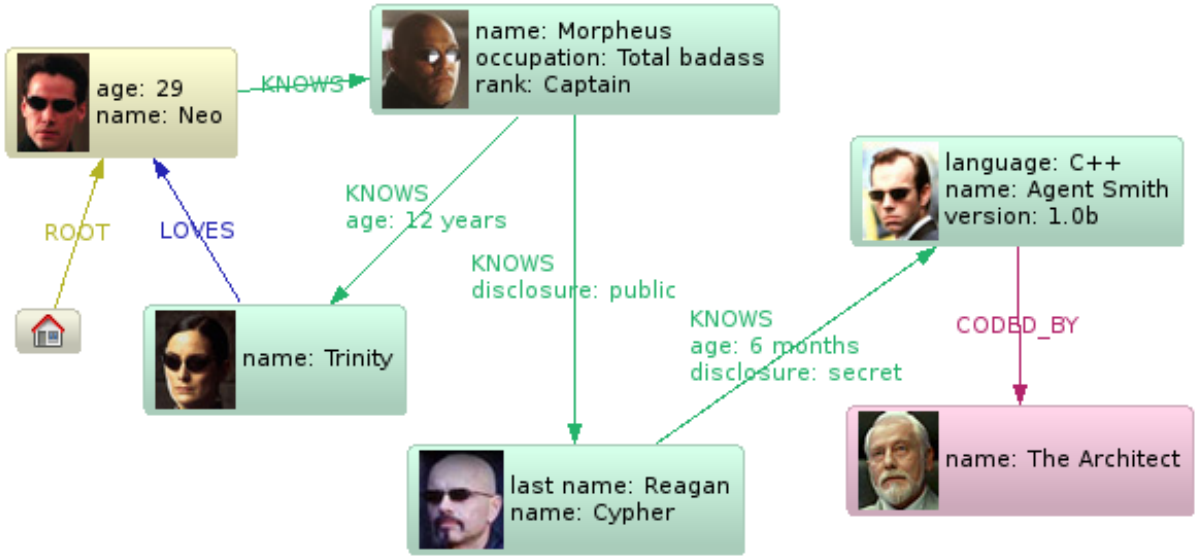
- NoSQL Graph Database
- Whiteboard Friendly
- Relationship focused
- Java-Based



Graph Database

- Linked nodes
- Stores data as nodes and relationships

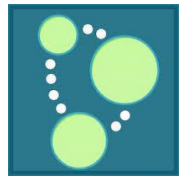
- Nodes = entities
- Relationships = edges (directional)



http://stackoverflow.com/questions/13533779/how-to-develope-neo4j-database-web-ui

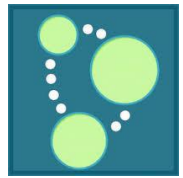
Graph Database (Cont.)

- Optimal for searching social network data
- Faster for associative data sets
- Data connections are stored
- Allows to attach Java objects as properties



Graph Database (Cont.)

- Cheap to traverse (query) along the relationships
- more likely to run on single server than clusters
- ACID properties need to cover nodes and edges to achieve consistency



Features

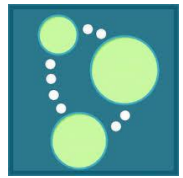
Example:

```
Node martin = graphDB.CreateNode();
martin.setProperty(“name”, “Martin”)

Node pramod = graphDB.CreateNode();
pramod.setProperty(“name”, “Pramod”)

martin.createRelationshipTo(pramod, FRIEND)
pramod.createRelationshipTo(martin, FRIEND)
```

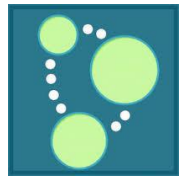
- Relationships are first class citizens
- Does not have start/end but have properties
- Based on the properties, add more content (when did they become friends, distance between nodes)
- Changing existing nodes and their relationships is similar to DATA MIGRATION



Consistency and Transactions

1. **Consistency** is achieved through transactions. Neo4J works on single server - to have no waiting time between master and slaves while writing.

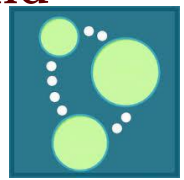
2. **Transactions** - Neo4J is ACID compliant. Before changing nodes/r'ships we have to start transaction else it will throw NotInTransactionException. Need to finish transaction by transaction.success() and transaction.finish().



Availability, Query Features

3. **Availability** - Achieves by providing replicated read-write slaves. Write is first committed to the master first, then slaves eventually. Uses Apache ZooKeeper to track transactions on each slave node and the current master node.

4. **Query Features** - Supported by Gremlin (domain specific language for traversing graphs). Properties of nodes and edges can be indexed using IndexManager. Neo4J also uses Lucene for full text search. It also searches for relationship status, the depth of the graph, # of paths and shortest path available. Relationships(Direction.INCOMING), findAllPaths(pramod, martin)



Scaling

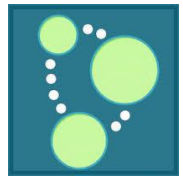
5. Scaling - Sharding is difficult as they are not aggregate oriented.

Three ways to achieve Scaling:

1. Add more RAM to server to held set of nodes and edges in memory.

2. Improve read scaling by adding more slaves with read-only access data, write access to the master.

3. Perform Application level sharding.



Wrap up!

1. Typeless, schemaless, no constraints on data, fast lookup with Lucene.
2. Cannot shard subgraphs, Neo4J is partition tolerant but use only for read requirements, best for social network.
3. Accepts ad-hoc data relations, supports HTTP/REST, written in Java, ACID durability, No mapreduce, Ad-hoc querying done by graph walking.
4. Master-slave replication, no sharding, write lock consistency, secondary indexes via Lucene.
5. ACID transactions, transaction event handlers, no multitenancy
6. Flexible graph, no security provided.