Introduction to NoSQL Databases and Neo4j

Introduction

- NoSQL is an approach to databases that represents a shift away from traditional relational database management systems (RDBMS).
- NoSQL can mean “not SQL” or “not only SQL.”
- Relational databases rely on tables, columns, rows, or schemas to organize and retrieve data. In contrast, NoSQL databases do not rely on these structures and use more flexible data models.
- **Benefits**
  - Scalability
  - Performance:
  - High Availability
  - Global Availability
  - Flexible Data Modeling

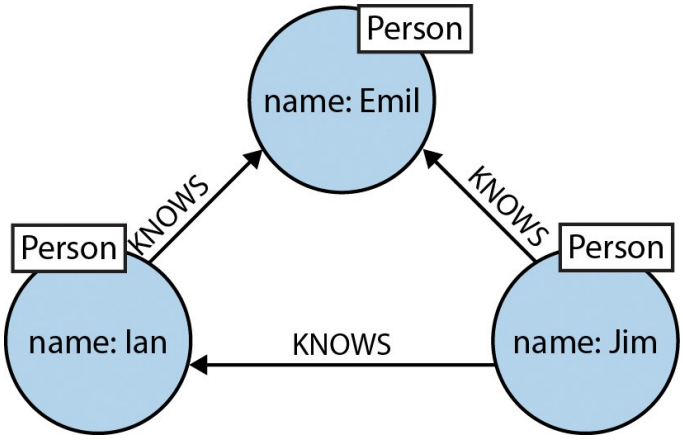
Types of NoSQL Databases

- **Key-value data stores:** data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection (Eg: ArangoDB, InfinityDB, Oracle NoSQL Database, Redis, and dbm)
- **Document Store:** They are similar to key-value stores, but in this case, a value is a single document that stores all data related to a specific key (Eg: XML, JSON, BSON)
- **Wide-column stores:** group columns of related data together. A query can retrieve related data in a single operation because only the columns associated with the query are retrieved (Eg: Accumulo, Cassandra, Druid, HBase, Vertica, SAP HANA)
- **Graph Database:** relations are well represented as a graph consisting of elements interconnected with a finite number of relations between them (eg: AllegroGraph, ArangoDB, InfiniteGraph, Apache Giraph, MarkLogic, Neo4J,)

Graph Databases

- There are no isolated pieces of information, but rich, connected domains all around us.
- Only a database that embraces relationships as a core aspect of its data model is able to store, process, and query connections efficiently.

- In **relational databases**, references to other rows and tables are indicated by referring to their (primary-)key attributes via foreign-key columns. This is enforceable with constraints, but only when the reference is never optional. Joins are computed at query time by matching primary- and foreign-keys of the many (potentially indexed) rows of the to-be-joined tables. These operations are compute- and memory-intensive and have an exponential cost.
- In **graph databases**, relationships take first priority. This means your application doesn't have to infer data connections using things like foreign keys or out-of-band processing, such as MapReduce. The data model for a graph database is also significantly simpler and more expressive than those of relational or other NoSQL databases.
- Graph databases are based on graph theory, and employ nodes, edges, and properties.
    - ***Nodes*** represent entities such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of the *record*, *relation*, or *row* in a relational database, or the *document* in a document database.
    - ***Edges,*** also termed *graphs* or *relationships*, are the lines that connect nodes to other nodes; they represent the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties, and edges. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in other systems.
    - ***Properties*** are germane information that relate to nodes. For example, if *Wikipedia* were one of the nodes, it might be tied to properties such as *website*, *reference material*, or *word that starts with the letter w*, depending on which aspects of *Wikipedia* are germane to a given database.

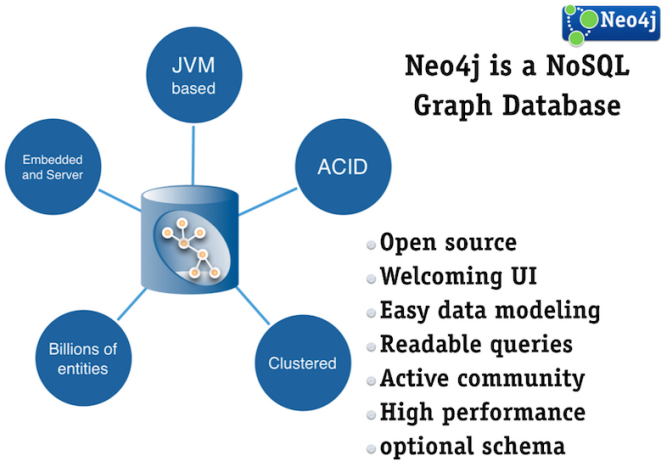


Comparison

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

Neo4j

- Sponsored by Neo Technology, Neo4j is an open-source NoSQL graph database implemented in Java and Scala.
- Neo4j provides full database characteristics including ACID transaction compliance, cluster support, and runtime failover, making it suitable to use graph data in production scenarios.
- Use cases include matchmaking, network management, software analytics, scientific research, routing, organizational and project management, recommendations, social networks, and more.
-



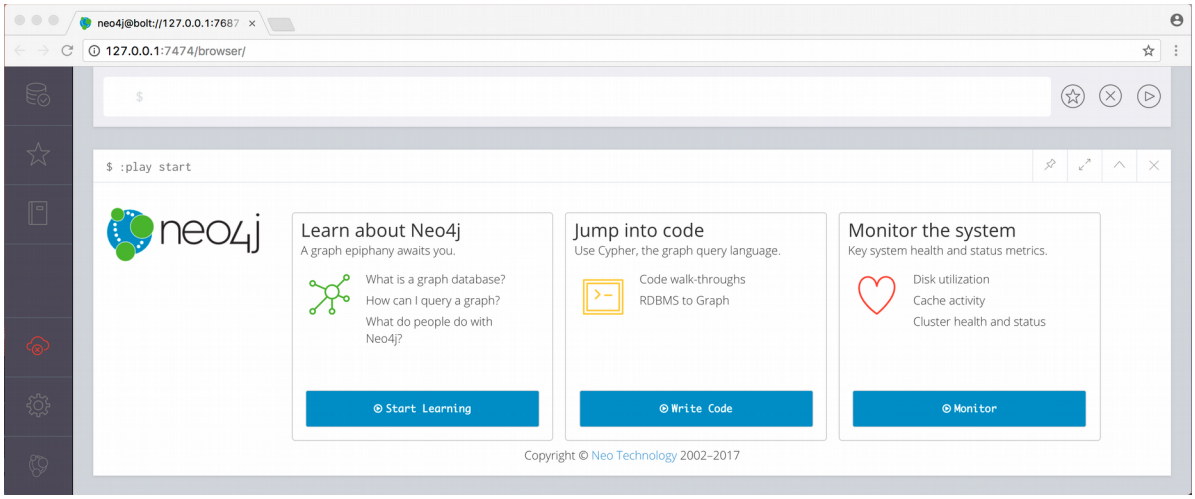
-

Neo4j Customers:

Neo4j - Environment Setup

For installing neo4j, follow the steps as in : https://medium.com/@Jessicawlm/installing-neo4j-on-ubuntu-14-04-step-by-step-guide-ed943ec16c56 . A summary is given here:

1. Make sure that Java 8 (or above) is installed. Otherwise install Java.
2. Open up your terminal/shell.
3. Add neo4j repository to software index: `wget -O - https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add -`
4. `echo 'deb http://debian.neo4j.org/repo stable/' >/tmp/neo4j.list`
5. `sudo mv /tmp/neo4j.list /etc/apt/sources.list.d`
6. `sudo apt-get update`
7. `To install community edition: sudo apt-get install neo4j=3.1.4`
8. `sudo service neo4j restart`
9. Visit http://localhost:7474 in your web browser.
10. Connect using the username 'neo4j' with default password 'neo4j'. You'll then be prompted to change the password. [Please set ‘bdbl’ as new password]



Neo4j - Building Blocks

- **Property is a key-value pair to describe Graph Nodes and Relationships.**
- **Node:** Node is a fundamental unit of a Graph. It contains properties with key-value pairs.
- **Relationships:** Relationships are another major building block of a Graph Database. It connects two nodes
- **Label** associates a common name to a set of nodes or relationships. A node or relationship can contain one or more labels.

Cypher Query Language (CQL)

CQL is a query language for Neo4j Graph Database.

- Is a declarative pattern-matching language.
- Follows SQL like syntax.
- Syntax is very simple and in human readable format.

CQL Read Cluases

CQL Write Cluases

Read Clauses	Usage
MATCH	This clause is used to search the data with a specified pattern.
OPTIONAL MATCH	This is the same as match, the only difference being it can use nulls in case of missing parts of the pattern.
WHERE	This clause id is used to add contents to the CQL queries.
START	This clause is used to find the starting points through the legacy indexes.
LOAD CSV	This clause is used to import data from CSV files.

Create Node

Write Clause	Usage
CREATE	This clause is used to create nodes, relationships, and properties.
MERGE	This clause verifies whether the specified pattern exists in the graph. If not, it creates the pattern.
SET	This clause is used to update labels on nodes, properties on nodes and relationships.
DELETE	This clause is used to delete nodes and relationships or paths etc. from the graph.
REMOVE	This clause is used to remove properties and elements from nodes and relationships.
FOREACH	This class is used to update the data within a list.
CREATE UNIQUE	Using the clauses CREATE and MATCH, you can get a unique pattern by matching the existing pattern and creating the missing one.
Importing CSV files with Cypher	Using Load CSV you can import data from .csv files.

```
CREATE (node:label)
```

Create Node with Properties

```
CREATE (node:label { key1: value, key2: value, . . . . . }) 
```

Create Relationship

```
CREATE (node1)-[:RelationshipType]->(node2)
```

Modify

```
merge (n:Node {name: 'John'})
set n = {name: 'John', age: 34, coat: 'Yellow', hair: 'Brown'}
return n
```

Delete a particular node:

```
MATCH (p:Person) where ID(p)=1
DELETE p
```

Delete a relation:

```
MATCH (n { name: 'Andres' })-[r:KNOWS]->()
DELETE r
```

(or)

match () -[r: knows]-> () delete r

Delete All

```
MATCH (n)
OPTIONAL MATCH (n)-[r]-()
DELETE n,r
```

Example

Relational Data Base

Employee

Eid	Ename	Dept id	Place	YoB	Salary
1000	John	100	Thrissur	1988	25000
1001	Jacob	100	Bangalore	1985	20000
1003	Harsh	101	Calicut	1989	20000
1004	Ann	101	Chennai	1989	24000

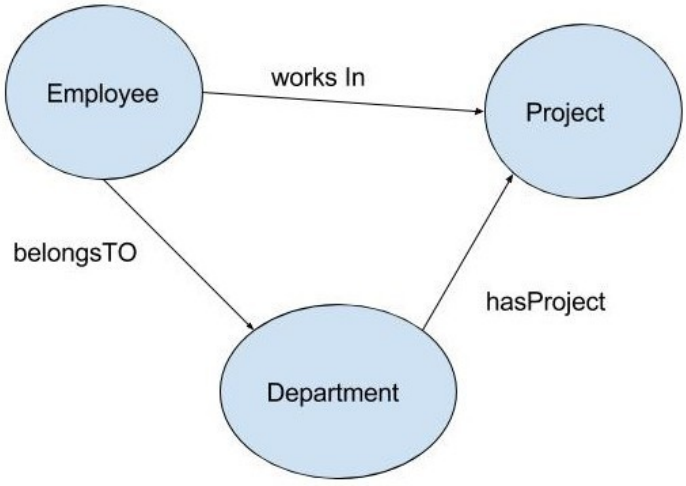
Department

Dept Id	Dept Name	Location
100	Research	Chennai
101	Administration	Trivandram
103	Development	Bangalore

Project

Project ID	Dept ID	Emp ID
P10	100	1000
P10	101	1003
P11	100	1001
P11	101	1003
P11	101	1004

Graph Model:



Create Nodes

create (john:employee {eid:1000,name:'John',place:'Thrissur',yob:1988,salary:25000})

```
CREATE CONSTRAINT ON (emp:employee) ASSERT emp.eid IS UNIQUE
create (jacob:employee {eid:1001,name:'Jacob',place:'Bangalore',yob:1987,salary:20000})
create (harsh:employee {eid:1003,name:'Harsh',place:'Calicut',yob:1989,salary:20000})


create (dept:department{did:100,dname:'Research',dloc:'Chennai'})
CREATE CONSTRAINT ON (dep:department) ASSERT dep.did IS UNIQUE
create (dept2:department{did:101,dname:'Admin',dloc:'Trivandrum'})

create (pjt:project{pid:'p10',name:'P10'})
create (pjt2:project{pid:'p11',name:'P11'})
```

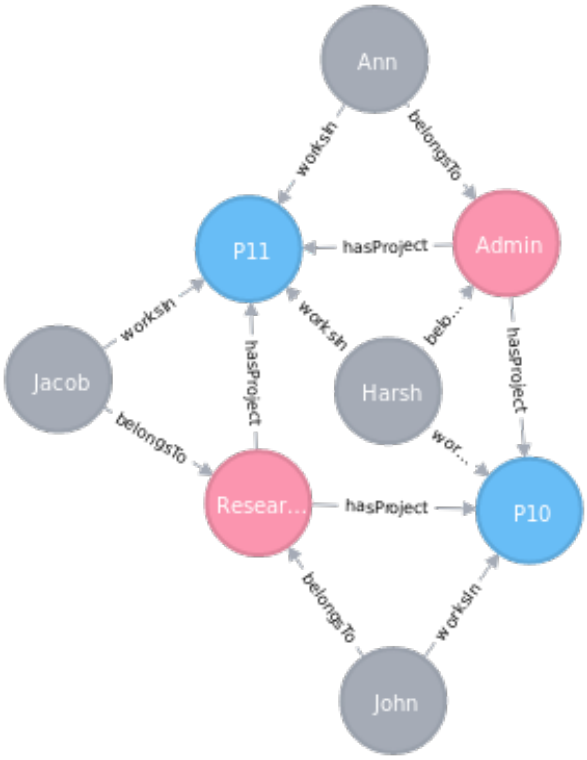
Add Relations

```
match (a:employee),(b:department) where a.eid=1000 and b.did=100 create (a)-[r:belongsTo] ->(b)return a,b
match (a:employee),(b:department) where a.eid=1001 and b.did=100 create (a)-[r:belongsTo] ->(b)return a,b
match (a:employee),(b:department) where a.eid=1003 and b.did=101 create (a)-[r:belongsTo] ->(b)return a,b
match (a:employee),(b:department) where a.eid=1004 and b.did=101 create (a)-[r:belongsTo] ->(b)return a,b

match (d:department),(p:project) where p.pid='p10' and d.did=100 create (d)-[r2:hasProject] ->(p)return d,p
match (d:department),(p:project) where p.pid='p10' and d.did=101 create (d)-[r2:hasProject] ->(p)return d,p
match (d:department),(p:project) where p.pid='p11' and d.did=100 create (d)-[r2:hasProject] ->(p)return d,p
match (d:department),(p:project) where p.pid='p11' and d.did=101 create (d)-[r2:hasProject] ->(p)return d,p

match (e:employee),(p:project) where p.pid='p10' and e.eid=1000 create (e)-[r3:worksIn] ->(p)return e,p
```

match (e:employee),(p:project) where p.pid='p10' and e.eid=1003 create (e)-[r3:worksIn] ->(p)return e,p

match (e:employee),(p:project) where p.pid='p11' and e.eid=1001 create (e)-[r3:worksIn] ->(p)return e,p

match (e:employee),(p:project) where p.pid='p11' and e.eid=1003 create (e)-[r3:worksIn] ->(p)return e,p



Query

1. List all employee names

   match (e:employee) return e.name

2. List employees works in project P10

   match (e:employee)-[:worksIn]->(p:project{pid:'p10'}) return e.name

Explore More....

1. **Developers' Manual,** https://neo4j.com/developer/

2**. Neo4j Tutorial:** https://www.tutorialspoint.com/neo4j/

3. Ian Robinson, **Graph Databases,** O'Reilly Media, Inc., 2013 (available at this link)

4. Ryan Boyd, **Intro to Graph Databases Series** [Video Lecture], (available at this link)