Neo4j Intro

(graphs)-[:ARE]->(everywhere)

by Mark Kreyman

LinkedIn / GitHub / Email

Database Comparison

Relational databases

- Can’t Handle Relationships Well
- Cannot model or store data and relationships without complexity
- Performance degrades with number & levels of relationships, and database size
- Query complexity grows with need for JOINs, aka JOIN hell
- Adding new types of data and relationships requires schema redesign

NoSQL

- Don’t handle relationships
- No data structures to model or store relationships
- No query constructs to support relationships
- Relating data requires "JOIN logic" in the application

Neo4j

- WYSIWYG -- Model and store your data as a graph
- "pre-built" relationships
- Query relationships in real-time
- Query time in microseconds
- Seamless evolution as your understanding of the customer domain grows

Performance

Finding extended friends in a Relational Database vs. in Neo4j

Depth	RDBMS exec time(s)	Neo4j exec time(s)	Records returned
2	0.016	0.01	~2,500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

** 1 mln records/nodes*

Terminology

Nodes for Things, Relationships for Structure

Nodes

- The objects in the graph
- Can have properties
- Can be labeled

Relationships

- Relate nodes by type and direction
- Can have properties

Modeling your domain

- O'Reilly's Graph Databases

Intro to Cypher

Cypher vs SQL

- Declarative, pattern matching, easy to understand
- Requires 10x to 100x less code than SQL

Example 1: Create a report

- Find all direct reports and how many people they manage, up to 3 levels down

```
MATCH (boss)-[:MANAGES*0..3]->(sub), (sub)-[:MANAGES*1..3]->(report)
WHERE boss.name = 'John Doe'
RETURN sub.name AS Subordinate, count(report) as Total
```

Example 2: Recommendation engine

- Top 25 Movies that I haven't seen with the same genres as Toy Story given high ratings by people who liked Toy Story

```
MATCH (watched:Movie {title:"Toy Story”})<-[r1:RATED]-()-[r2:RATED]->(
WHERE r1.rating > 7 AND r2.rating > 7
AND watched.genres = unseen.genres
AND NOT( (:Person {username: "mkreyman"})-[:RATED | WATCHED]->(unseen)
RETURN unseen.title, COUNT(*)
ORDER BY COUNT(*) DESC
LIMIT 25
```

Example 3: Recommendation engine (continued...)

- What are the Top 25 Movies that Zoltan Varju has not seen using the average rating by my top 3 neighbors

```
MATCH (m:Movie)<-[r:RATED]-(b:Person)-[s:SIMILARITY]-(p:Person {name:
WHERE NOT((p)-[:RATED|WATCHED]->(m))
WITH m, s.similarity AS similarity, r.rating AS rating
ORDER BY m.name, similarity DESC
WITH m.name AS movie, COLLECT(rating)[0..3] AS ratings
WITH movie, REDUCE(s = 0, i IN ratings | s + i)*1.0 / LENGTH(ratings)
ORDER BY recommendation DESC
RETURN movie, recommendation
LIMIT 25
```

Demos

Demo 1: Dependencies graph (`mix graph_deps`)

- What packages depend on *absinthe*?

```
MATCH path = (p:Package {name: "absinthe"})<--(d)
RETURN path
```

- What packages does *absinthe* depend on?

```
MATCH path = (p:Package {name: "absinthe"})-->(d)
RETURN path
```

Demo 2: ...with maintainers (`mix graph_deps_more`)

- Packages maintained by *José Valim*

```
MATCH path = (:Maintainer {name: "José Valim"})<--(:Package)
RETURN path
```

Demo 3: `:play movie graph`

- Actors who played in some movie

```
MATCH (m:Movie {title: 'Sleepless in Seattle'})<-[:ACTED_IN]-(a)
RETURN m, a
```

- Find the actors with 5+ movies, and the movies in which they acted

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WITH a, collect(m.title) AS movies
WHERE length(movies) >= 5
RETURN a, movies
```

- Find movies released in the 1990s

```
MATCH (nineties:Movie) WHERE nineties.released >= 1990 AND nineties.re
RETURN nineties.title
```

More examples

- Modeling comics at Marvel
  - https://vimeo.com/79399404
- An offer engine
  - https://maxdemarzi.com/2018/05/17/offers-with-neo4j/
- A recommendation engine in 2 mins
  - https://www.youtube.com/watch?v=qbZ_Q-YnHYo

Common Use Cases

Stats

- *76% of Fortune 100* have adopted or piloted Neo4j
- *Finance*: 20 of top 25
- *Software*: 7 of top 10
- *Logistics*: 3 of top 5
- *Retail*: 7 of top 10
- *Airlines*: 3 of top 5
- *Telco*: 4 of top 5
- *Hospitality*: 3 of top 5

Common Use Cases

- Internal Applications
- Master Data Management
- Network and IT Operations
- Fraud Detection
- Real-time Recommendations
- Graph-based Search
- Identity and Access Management
- Investigative journalism (i.e. "Panama papers")

Importing existing data

Import CSV, XML, API/JSON, databases

- LOAD CSV: up to 10 mln nodes and relationships
- Command-line Bulk Loader: For loads 10B+ records @ 1M records per second

Demo

- Generate dependencies with `Mix.Tasks.Xref.calls()` and export to CSV

```
defp fields(%{
       callee: {callee_module, func, arity},
       caller_module: caller_module,
       file: caller_file,
       line: line
     }) do
  [callee_module, func, arity, caller_module, caller_file, line]
end
```

- Create constraints in neo4j

```
CREATE CONSTRAINT ON (p:Module) ASSERT p.name IS UNIQUE;
CREATE CONSTRAINT ON (m:Function) ASSERT m.name IS UNIQUE;
```

Demo continued...

- Check first few raw lines

```
LOAD CSV FROM "file:///tmp/deps.csv" AS row
WITH row
RETURN row
LIMIT 5;
```

- Import all lines

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM "file:///tmp/deps.csv" AS row
WITH row
MERGE (callee_module:Module {name: row[0]})
MERGE (function:Function {name: (row[1] + '/' + row[2])})
MERGE (caller_module:Module {name: row[3]})
MERGE (function)-[d:DEFINED_IN]->(callee_module)
MERGE (caller_module)-[c:CALLS {file: row[4], line: row[5]}]->(funct
```

GraphQL integration

GraphQL native support

Neo4j server, with a plugin, can act as a GraphQL server

- Drop plugin into `/var/lib/neo4j/plugins`
- Configure in `neo4j.conf`

```
...
dbms.unmanaged_extension_classes=org.neo4j.graphql=/graphql
```

- Or configure in `docker-compose.yml`

```
...
environment:
  - NEO4J_dbms_unmanaged__extension__classes=org.neo4j.graphql=/grap
```

GraphQL schema definition

- Define schema

```
CALL graphql.idl(
  'type Maintainer {
    id: ID
    name: String
    packages: [Package] @relation(name:"DEVELOPED_BY",direction:IN)
  }
  type Package {
    id: ID
    name: String
    maintainers: [Maintainer] @relation(name:"DEVELOPED_BY")
  }')
```

- What's my schema, again?

```
CALL graphql.schema()
```

Run queries `http://localhost:7474/graphql/`

- Who maintains `absinthe` package?

```
{ Package (name:"absinthe") {
    name
    maintainers {
      name
    }
  }
}
```

- What packages are being maintained by José Valim?

```
{ Maintainer(name:"José Valim") {
    name
    packages {
      name
    }
  }
}
```

Nested queries

- What are other maintainers for packages maintained by José Valim?

```
{ Maintainer(name:"José Valim") {
    name
    packages {
      name
      maintainers {
        name
      }
    }
  }
}
```

Local development

Docker

- Docker images I use

  ```
  docker pull neo4j
  docker pull shufo/phoenix
  ```

- Sample docker-compose.yml

Elixir Drivers

- Bolt: florinpatrascu/bolt_sips
- Sips: florinpatrascu/neo4j_sips
- No "official" adapter for Ecto
- Sample application: https://github.com/StabbyMcDuck/elixir_ravelry

(graphs)-[:ARE]->(everywhere)