Graph Databases

Antonio Maccioni

Big Data Course

24 April 2014

Rome

maccioni@dia.uniroma3.it

locatedIn

heldBy

email

topic

affiliated

lectureOf

offeredBy

ROMA TRE

UNIVERSITÀ DEGLI STUDI
“Graph Databases are an odd fish in the NoSQL pond”
- P.J. Sadalage, M. Fowler - NoSQL Distilled

“Over 25 percent of enterprises will use graph databases by 2017”
- Enterprise DBMS, Q1 2014. Forrester Research
**Why Graph Databases?**

Let us suppose Facebook data stored on relational databases.

> 1.2 billions active users

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>DATE OF BIRTH</th>
<th>RELATIONSHIP</th>
<th>WORKS IN</th>
<th>PICTURES</th>
<th>...</th>
</tr>
</thead>
</table>
Why Graph Databases?

Every time we visit a profile we have to find the common friends

~ 1.2 billions tuples
* average number of friends per person divided by two
Why Graph Databases?

Every time we visit a profile we have to find the common friends:

<table>
<thead>
<tr>
<th>FRIEND 1</th>
<th>FRIEND 2</th>
<th>FROM_DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

~ 1.2 billions tuples
* average number of friends per person divided by two
Why Graph Databases?

We would like to manage connections explicitly...

...keeping the cost of a basic traversal $O(1)$
How to define a Graph Database?

Three main properties:

1. **Property Graph** (as a data model)
2. **Index-free Adjacency** (for physical level organization)
3. **Path-traversal** (as query language)

A **Graph Database Management System (GDBMS)** is the system which manages graph databases.
A property graph is a directed multigraph $g = (N, E)$ where every node $n \in N$ and every edge $e \in E$ is associated with a set of pairs $<key, value>$, called properties.

It's a schema-less data model.
Property Graph Data Model

Index-free Adjacency

We say that a (graph) database \( g \) satisfies the index-free adjacency if the existence of an edge between two nodes \( n_1 \) and \( n_2 \) in \( g \) can be tested on those nodes and does not require to access an external, global, index.
Existing GDBMSs

... there are many visualization and exploration tools for GDBMSs:
Build a Neo4J graph database

Server Mode

1. Go to http://www.neo4j.org/, download Neo4J Server and unzip it

2. Run the command ./bin/neo4j start (use bin\Neo4j.bat on Windows)

3. Find a graphical dashboard at http://localhost:7474/

4. You can also use it with REST API: http://localhost:7474/db/data/
Build a Neo4J graph database

Embedded mode

1. Import in your java project the library `neo4j-kernel-*-**.jar` and its classes:

```java
import org.neo4j.graphdb.*;
import org.neo4j.graphdb.factory.GraphDatabaseFactory;
```

2. Create the database:

```java
GraphDatabaseService gdb = new GraphDatabaseFactory().
    newEmbeddedDatabase("/home/...");
```

3. Create nodes and edges:

```java
Node n1 = gdb.createNode();
Node n2 = gdb.createNode();
Relationship e12 = n1.createRelationshipTo(n2, EdgeType.TYPE);
n1.setProperty("name", "Rome");
n2.setProperty("name", "Lazio");
e12.setProperty("type", "locatedIn");
```

4. Set the properties:

```java
n1.setProperty("name", "Rome");
n2.setProperty("name", "Lazio");
e12.setProperty("type", "locatedIn");
```
Tinkerpop Stack

http://www.tinkerpop.com/

**BLUEPRINTS**

Blueprints is a property graph model interface with provided implementations.

**GREMLIN**

Gremlin is a domain specific language for traversing property graphs.

**FRAMES**

Frames exposes the elements of a Blueprints graph as Java objects: software is written in terms of domain objects and their relationships to each other.

**FURNACE**

Furnace is a property graph algorithms package.

**REXTER**

**PIPES**
1. Import in your java project the libraries
   `blueprints-core-*.jar` and `blueprints-neo4j-graph-*.jar` with their classes:

   ```java
   import com.tinkerpop.blueprints.*;
   import com.tinkerpop.blueprints.impls.neo4j.Neo4jGraph;
   ```

2. Create the database:

   ```java
   Graph gdb = new Neo4jGraph("/home/...");
   ```

3. Create nodes and edges:

   ```java
   Vertex n1 = gdb.addVertex(null);
   Vertex n2  = gdb.addVertex(null);
   Edge e12 = gdb.addEdge(null, n1, n2, "locatedIn");
   ```

4. Set the properties:

   ```java
   n1.setProperty("name", "Rome");
   n2.setProperty("name", "Lazio");
   e12.setProperty("type", "locatedIn");
   ```
Gremlin: a path traversal QL

```java
gremlin> g = new Neo4jGraph("/home/...");

gremlin> g.V.outE.filter{it.edgeid == 'e1'}
===>e[2][1-EDGE->4]
===>e[1][1-EDGE->3]
===>e[0][1-EDGE->2]
===>e[7][6-EDGE->2]
```
Gremlin: a path traversal QL

gremlin> g.V.outE.filter{it.edgeid == 'e1'}.inV.outE.
        filter{it.edgeid == 'e2'}.inV.nodeid

==>F
==>E
==>E
Cypher: a pattern matching QL

...with starting point (why?)

START: starting point in the graph
MATCH: the pattern to match, bound to the starting point
WHERE: filtering criteria
RETURN: what to return
Cypher: a pattern matching QL

START n = node(*)
MATCH n-[r1:EDGE]->x-[r2:EDGE]->m
WHERE (r1.edgeid = 'e1') and (r2.edgeid = 'e2')
RETURN m.nodeid

==>"F"
==>"E"
==>"E"
Cypher demo hands-on

http://www.neo4j.org/misc/beer

http://console.neo4j.org/?id=beer

Graph Setup:
CREATE {1 {name: "Beer"}}, {2 {name: "Trappist Beer"}}, {3 {name: "Orval"}}, {4 {name: "Westmalle"}}, {5 {name: "La Trappe"}}, {6 {name: "Chimay"}}, {7 {name: "Westvleteren"}}, {8 {name: "Local Beer"}}, {9 {name: "De Koninck"}}, {10 {name: "Seef"}}, {11 {name: "Specialty Beer"}}, {12 {name: "Malheur"}}, {13 {name: "Commodity Beer"}}, {14 {name: "Leffe"}}, {15 {name: "Duvel"}}, {16 {name: "Rodenbach"}}, {2 {isa} -> 1}, {3 {isa} -> 2}, {4 {isa} -> 2}, {5 {isa} -> 2}, {6 {isa} -> 2}, {7 {isa} -> 2}, {8 {isa} -> 1}, {9 {isa} -> 8}, {10 {isa} -> 8}, {11 {isa} -> 1}, {12 {isa} -> 11}, {13 {isa} -> 1}, {14 {isa} -> 13}, {15 {isa} -> 13}, {16 {isa} -> 13}

Query:
START beers=node=node_auto_index(name='Beer')
MATCH beers<-[:isa]-type<-[:isa]-beer
RETURN type, name AS type, count(*) AS count, co
Other Features

- **Secondary Indexes**: defined on properties

- **Transactions**: graph databases usually support ACID properties. In Neo4J all operations have to be performed in a transaction:

  ```java
  try ( Transaction tx = gdb.beginTx() )
  {
      tx.success();
  }
  ```

- **Language Drivers**: other than Java
Existing Issues with Graph DBs

- How to model a graph database?
- How to migrate data, queries, indexes, etc. from existing databases?
- How to shard a graph database?
R2G: from relations to graphs

**R2G: unifiability of data values**

- **Joinable** tuples $t_1 \in R_1$ and $t_2 \in R_2$:
  - there is a foreign key constraint between $R_1.A$ and $R_2.B$ and $t_1[A] = t_2[B]$.

- **Unifiability** of data values $t_1[A]$ and $t_2[B]$: 
  - (i) $t_1 = t_2$ and both A and B do not belong to a multi-attribute key;
  - (ii) $t_1$ and $t_2$ are joinable and A belongs to a multi-attribute key;
  - (iii) $t_1$ and $t_2$ are joinable, A and B do not belong to a multi-attribute key and there is no other tuple $t_3$ that is joinable with $t_2$. 

R2G: schema graph

Full Schema Paths:

\[ FR.fuser \rightarrow US.uid \rightarrow US.uname \]
\[ FR.fuser \rightarrow FR.fblog \rightarrow BG.bid \rightarrow BG.bname \]
\[ FR.fuser \rightarrow FR.fblog \rightarrow BG.bid \rightarrow BG.admin \rightarrow US.uid \rightarrow US.uname \]

R2G: data and query migration

SELECT US.uname
FROM User US, Blog BG
WHERE (BG.admin = US.uid) AND
(BG.bname = 'Inf. Systems')

Modeling Graph Databases

Different modeling strategies, from a “philosophical” point of view:

- **Compact**: fewer nodes by greedily aggregating data on the same node.
  - ✅ reduces the number of data accesses
  - ❌ needs to pay attention to the property graph constraints

- **Sparse**: more nodes to reduce the average degree of nodes
  - ❌ access and updates can be inefficient

- **Dense**: more edges between nodes
  - ✅ reduce number of joins at run time
  - ❌ needs a semantic enrichment

What about using the ER **conceptual modeling** for graph databases?

R. De Virgilio, A. Maccioni, R. Torlone – Model-driven design of graph databases (submitted paper)
Modeling Graph Databases

Oriented-ER

R. De Virgilio, A. Maccioni, R. Torlone – Model-driven design of graph databases (submitted paper)
Modeling Graph Databases

Partitioning

Graph Database Template

R. De Virgilio, A. Maccioni, R. Torlone - Model-driven design of graph databases (submitted paper)
Thesis 1 (Cypher in R2G)

Extend existing code to convert SQL queries into Cypher language

**SQL**

```
select *
from T
where T.A1 = v1
```

**Cypher**

```
Start ...
Match ...
Where ...
Return ...
```
Thesis 2 (Conceptual Model in R2G)

Implementation of a tool for the model-driven approach
Thesis 3 (Queries into R2G)

Implementation of an algorithm for the query-driven modeling

\[
\text{select } * \\
\text{from } T \\
\text{where } T.A1 = v1
\]

\[
\text{insert } \\
\text{where } T.A1 = v1
\]

\[
\text{select } * \\
\text{from } T2 \\
\text{where } T.A5 = v6
\]
Thesis 4 (Object Mapping)

Persist java objects into graph databases:
- Practice with existing tools for Object Mapping
- Exploiting the ideas of R2G

Data Nucleus
JO4Neo
Spring Data Neo4J
Spring Data Graph
Neomodel
Bulbflow
Tinkerpop Frames
Thesis 5 (Reachability)

Study existing reachability indexes:

- Apply those methods to graph databases keeping the index-free adjacency
- Exploit the reachability in Furnace's algorithms (Bron-Kerbosch, Strong Connected Components, Dijkstra, Search)