NoSQL Databases

Polytech Nice-Sophia, SI4

Pierre Monnin

pierre.monnin@inria.fr

Special thanks to **Dr. Sabeur Aridhi**, **Dr. Jörn Franke** and **Dr. Nicolas Travers** for providing their materials

Chapter 5: Graph Databases

Overview

- Key idea: Use Graph data structures for storing and processing information
- Useful for data with a lot of relations, e.g. social networks, travel planning
  - Example query: how are customer A and B related

- Use a lot of popular algorithms you know already from your CS classes
  - Example: Shortest path, Dijkstra

Use Cases (1/2)

- Social networks
  - InfluencerAnalysis: Which customer influences the most other customers (++/--)
  - Finding expertise across projects and departments within your organization
- Facebook Graph Search: search through your social networks based on various criteria
- Google Knowledge Graph: improved search through semantic processing
- Data Vault Modeling: particularly in context of Master Data Management with data from a lot of different sources

Use Cases (2/2)

- Recommendations
  - Based on the products you buy find related products

- Real-World-Modeling/Internet of Things: Define how are things interconnected
- Find different flight routes to a destination
- Navigation: find shortest path on map

Schema

- **Key elements**

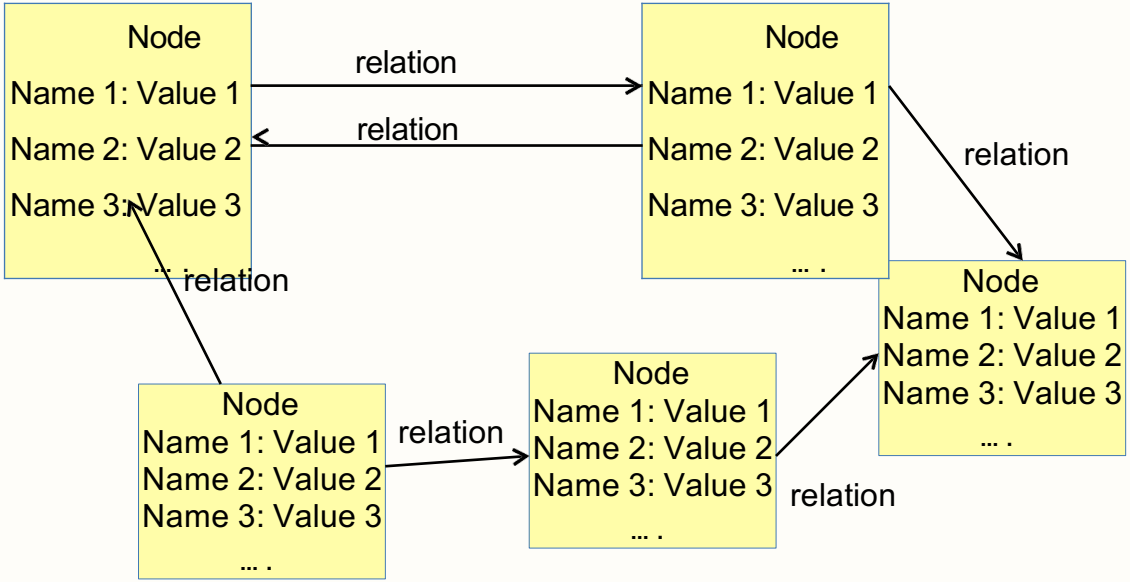
–Vertices

- *Represent objects/data*

–Edges

- *Describe relations between vertices*
  - +/- associated with objects/data
- *Have a direction*
- *There can be multiple edges between two vertices*

Illustration



Type of Queries

- No SQL-Style Queries
- Many different graph query languages

Sharing similar concepts
Examples: Cypher, SPARKQL, Gremlin

- You can query, insert, update and delete elements of a graph

Distributed Persistence

- Out of the box usually only replication for high availability is supported (Neo4J)

- Sharding (Partioning): Different strategies
  - By Edges
  - By Vertices
  - Combination

- Sharding is a non-trivial concept in the Graph world
  - Generalized systems: GraphX
  - Highly Specialized system: PowerGraph
  - You can use any graph system if you add your own sharding/parallel data processing logic on top

Transactional Guarantees

- Usually support full ACID
  - We do not want edges without vertices or vertices hanging „around“
  - Example: Neo4J
- Remember: The other NoSQL technologies did not support relations very well
  - Hence: Other NoSQL technologies could relax transactional guarantees
- Slower to respond – transactions cause overhead
- Alternatively you can program your own transaction model

Technologies

- Neo4J

–Standalone Embedded in Application

–Client/Server-Version

–Can be integrated with Apache Hadoop Big Data Processing Platform

- GraphX

–Integrated with Apache Spark Big Data Processing Platform

Evaluation

- Graph Databases are very suitable for any graph specific problem
  –A lot of use cases
- Contrary to many other NoSQL technologies they are very suitable for data with a lot of relations to other data
- They are not very suitable for updating a lot of nodes in a graph
- Parallel Processing of Graph Operations is more challenging
- Are less known than aforementioned NoSQL technologies

- Graph Database
- Neo4J Exercises

Overview (1/2)

- Neo4J was initially released in 2007
- Developed in Java
- Dual-License: GPLv3 and AGPLv3 / commercial
- Usage

–Embedded directly into Java-application – without server

–Client accessing a server

Overview (2/2)

- Typical graph storage

- Cypher Query Language

- Not very suitable for parallel data processing across several nodes (sharding)

–Most of the applications won't need it

–You can manually split the graph into different ones → manual sharding

Difference from SQL/relational databases

- Different data structures: Vertexes and edges
- Better suitable for a lot of relations
- Different Query Languages
- Same: ACID properties

Schema

- Database – Usually one Neo4J instance is responsible for one graph
- Nodes
  - Label (optional): Group nodes
- Relationships: between Nodes
- Both can contain properties (optional)
- Very flexible Schema
  - No data types need to be specified, but implicitly uses Java data types
- Path: A subset of a graph (nodes and relationships)

Queries: Cypher

- Cypher is a declarative query language

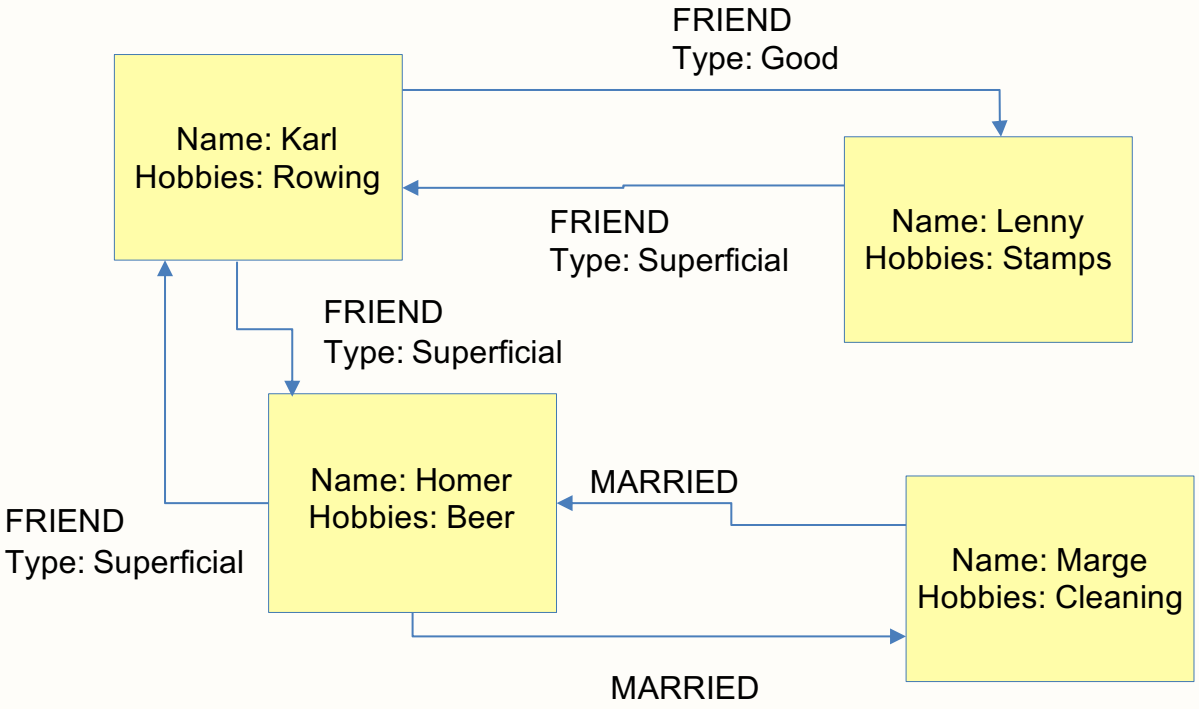
–Version: 2.1

- You can use it to query, create, update, delete nodes and relationships

- Query: Match nodes/relationships having a certain pattern

- Traversal: Describe how one traverse through a graph – main benefit in comparison with SQL/relational databases

Queries: Example Graph



Queries: Create a node

- CREATE (variable: label {properties})

- Example:

```
CREATE (n: Person {Name: 'Homer', Hobbies: 'Beer'});
```

Queries: Match Nodes

- Similar to SQL SELECT

```
MATCH (variable:label)
WHERE (variable.property = '')
RETURN variable;
```

- Example: Match nodes with label person and name 'Homer'

```
MATCH (a:Person)
WHERE a.Name = 'Homer'
RETURN a;
```

Queries: Create a Relationship

- Relationships have a direction
- Condition: Nodes between the relationship exist already

```
MATCH (a:Person),(b:Person)
WHERE a.name = 'name1' AND b.name = 'name2'
CREATE (a)-[r:RELTYPE]->(b)
RETURN r
```

- Example: Homer is FRIEND of Karl

```
MATCH (a:Person),(b:Person)
WHERE a.Name = 'Homer' AND b.Name = 'Karl'
CREATE (a)-[r:FRIEND{type: 'Superficial'}]->(b)
RETURN r;
```

Queries: Match Relationships

- Similar to SQL select
- Match relationships between nodes

```
MATCH (n:Person)-[r:FRIEND]-m
RETURN n,r,m
```

- Example:

```
MATCH (n:Person)-[r:FRIEND{ type: 'Superficial' }]-m
RETURN n,r,m;
```

Returns incoming and outgoing relations!

Queries: Graph Algorithm /Shortest Path

- Remember: Graph structures enable many different very useful algorithms

–Shortest Path (Maps like Google Maps, Robotics, Game Development)

–Analysis and Simulation

– …

- Other Algorithms can be implemented using the Traversal API
- Example Query:

```
MATCH (a:Person { Name:'Homer' }),(b:Person { Name:'Karl' }), p = shortestPath((a)-[*..15]-(b))
RETURN p;
```

Index

- **Nodes**
  - You can create fulltext search indexes (using Lucence) on properties

- **Relationships**
  - Similar as node index
  - Can be constrained to certain start and end nodes

Transactions

- Fully ACID compliant transactions
- Any query you issue is a transaction
- You can put multiple queries into a transaction
- You can do rollback

Distributed Persistence

- Neo4J is mostly about replication

–Higher Query performance

–Fail-over

- Sharding has to be done manually

Interfaces

- JDBC
- REST
- neo4j-shell: command line interface

Who uses it

- Cisco – Master Data Management

–Create a lot of relations between your key assets represented in master data objects

–Perform analytics on these relations

- Neo Technologies – Fraud Visualisation & Analytics

–Relations between family, friends, business partners

–Relations to objects, e.g. restaurant bill

- Telenor – Advanced Resource Authorization

– Resource access defined as a graph – easy to understand and query

– Once getting access to resources Solr is used for searching in documents one has access to

– Before: 24 hours old data

– Now: Real-time access

MegaSale

- Mega Sales is now participating in the movie business
- It wants to give its customer the possibility to flexible search for movies according various criteria
- It want to give recommendations based on what friends think about movies

Excursus: Loading data into Neo4J

- Copy database from other Neo4J instances
- Neo4J CSV Batch Import
- ETL tools

–Pentaho Kettle

- Develop your own import routine and use cypher to create the elements

Other types of graphs!

- Neo4J relies on Property Graphs
- Other types of Graphs: Semantic Web (knowledge) graphs

→ *Your Semantic Web / Linked Open Data / Knowledge Engineerin classes (SI5)*

- Gaps and similarity between the two worlds?

Query languages, graph model, theoretical properties

- Bridges between these two worlds?

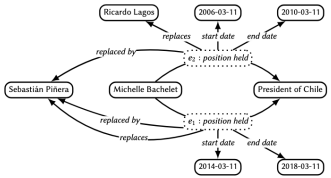


Figure 5: Multilayer graph for Figure 1

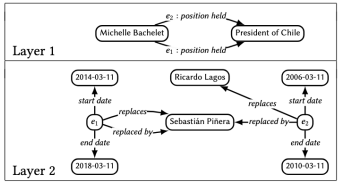


Figure 6: Two layers of Figure 5

Michelle Bachelet [Q320]

position held [P39] President of Chile [Q466956]
start date [P580] 2014-03-11
end date [P582] 2018-03-11
replaces [P155] Sebastián Piñera [Q306]
replaced by [P156] Sebastián Piñera [Q306]

position held [P39] President of Chile [Q466956]
start date [P580] 2006-03-11
end date [P582] 2010-03-11
replaces [P155] Ricardo Lagos [Q331]
replaced by [P156] Sebastián Piñera [Q306]

Figure 1: Wikidata statement group for Michelle Bachelet

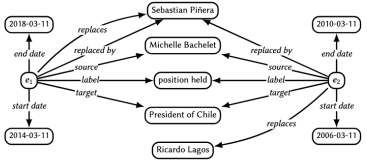


Figure 2: Directed labeled graph reifying Figure 1

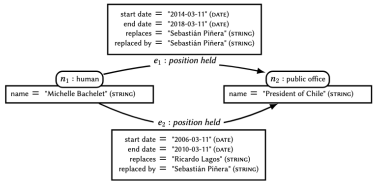


Figure 3: Property graph for Figure 1, with extra node labels

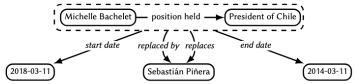


Figure 4: RDF* graph for the first statement of Figure 1

Multilayer graphs: A unified data model for graph databases

Renzo Angles
DCC, Universidad de Talca & IMFD
Chile
rangles@utalca.cl

Aidan Hogan
DCC, University of Chile & IMFD
Chile
ahogan@dcc.uchile.cl

Ora Lassila
Amazon Web Services
USA
ora@amazon.com

Carlos Rojas
IMFD
Chile
cirojas6@uc.cl

Daniel Schwabe
Visiting Researcher USC Information Sciences Institute
USA
dschwabe@gmail.com

Pedro Szekely
USC Information Sciences Institute
USA
dvrgoc@ing.puc.cl

Domagoj Vrgoč
PUC Chile & IMFD
Chile
dvrgoc@ing.puc.cl

ABSTRACT

In this short position paper, we argue that there is a need for a unifying data model that can support popular graph formats such as RDF, RDF* and property graphs, while at the same time being powerful enough to naturally store information from complex knowledge graphs, such as Wikidata, without the need for a complex reification scheme. Our proposal, called the *multilayer graph model*, presents a simple and flexible data model for graphs that can naturally support all of the above, and more. We also observe that the idea of multilayer graphs has appeared in existing graph systems from different vendors and research groups, illustrating its versatility.

edge. This forms the basis of concrete data models, such as the Resource Description Framework (RDF) [7]. Such an abstraction is also used by the Machine Learning community for topics such as knowledge graph embeddings [25] and in the Networks community for topics such as community detection [17]. It is also popular in the Database community, particularly in the theoretical literature, where such graphs are often simply called *graph databases* [27].

However, in practice, directed labeled graphs are sometimes considered *too simple*. What if, for example, we want to add data that describe edges themselves, or graphs themselves? While more complex data can be modeled in directed labeled graphs using various forms of reification [14], the result can often be verbose and

Table 1: Features supported without reserved terms
(NG = Named Graphs, PG = Property Graphs, MG = Multilayer Graphs)

	RDF	RDF*	NG	PG	MG
Edge type/label	✓	✓	✓	✓	✓
Node label				✓	
Edge annotation		✓	✓	✓	✓
Node annotation	✓	✓	✓	✓	✓
External annotation				✓	
Edge as node		✓	✓		✓
Edge as nodes			✓		✓
Arbitrary layers		✓	✓		✓
Graph as node			✓		
Quotation		✓			

https://dl.acm.org/doi/pdf/10.1145/3534540.3534696