Querying Financial Data using Neo4j and MongoDB

Federico Giacardi[1]

[1]University of Turin

MAADB Final Assignment

Table of Contents

Table of Contents

Introduction

- Goal: testing NoSQL databases Neo4j and MongoDB
- Need a lot of data with complex relations and nested structure to test strengths: LDBC FinBench Generator
- Local with different scale factors for performance evaluation, cloud copy with scale factor 0.1 for web app development
- Data in TSV format: conversion to CSV with csv_converter.py
- Schema described in LDBC Technical Report must be replicated

Neo4J Model

- Simple mapping to graph-based format: classes become nodes, relations become edges
- Attributes become nodes and edges properties
- Data types: 64-bit-wide integers and float for numerical values and dates, represented as timestamps
- Model defined using Aura visual editor, used also offline.
- Data imported from CSV using dedicared Neo tool

MongoDB Model

- Must decide between main concepts with autonomous collections and side ones, embedded
- Idea: embed documents that contain data on relations exploited by queries, so that no join is needed
- Relevant for query: documents modelling account ownership, loan stipulation, and investments embedded into the company collection
- Must merge CSV before importing them: merger_mongo.py, result as JSON
- Data imported from CSV using import_mongo.py

Table of Contents

Similarity computation

- Three simpler queries to compute the needed data
- Jaccard Similarity: $\frac{|C1 \cap C2|}{|C1 \cup C2|}$
- List of persons that invest in at least one company: user will select parameter from here
- List of companies both customers invest in: distinct to avoid same match for different pairs
- List of companies at least one customer invests in: distinct to avoid counting common companies twice

Summary sheet

- Multistage query evaluated using aggregate: new, improved version of mapreduce
- First stage: lists of companies returned by union query
- Second stage: adds fields that compose summary
- Third stage: maps company to summary fields
- No join needed: data are inside main relation, faster evaluation, thanks to document aggregation
- Polyglot persistence: different data models benefit different queries. Graph for relation traversal, document-based for aggregation

Lookup query

- Suspect money laundering: transfer between three accounts where at least one is blocked
- Transaction starts in given time frame
- Exploits the ability to rapidly traverse complex, multi-layered, relations
- No need for join: faster, easier to formalize

Table of Contents

Structure

- Made in React and Typescript for flexible structure and fast development
- Backend in JavaScript: access to uge ecosystem
- Connects to cloud databases generated with scale factor 0.1
- Backend: 53-bit-wide integers used, may loose precision on ids for bigger scale factors
- Frontend: Home module for presentation, Lookup and Analytic modules for queries. Graph result visualization

Table of Contents

Results

Query	SF 0.1	SF 0.3	SF 1	SF 3
investors	62	133	152	215
intersection	12	23	48	56
union	26	40	57	71
money laundering	215	317	543	1993
indexed money laundering	60	196	369	1316
summary sheets	19	55	293	409

Discussion

- Results in milliseconds. Execution on local copies on a Intel i7-9750H, 12 cores machine
- Range indexes on Person.id and Company.id
- First similarity query slower then the others: needs to retrieve matching companies for each person
- Money laundering query very slow: faster then using join, but still has to traverse long, complex relations structure
- Range index on Transfer.createTime improves performance (penultimate row)
- Summing up: significant performance degradation as data size grows. Processing big data requires computational power and storage