A research on Architectural and Performance Comparison of Relational Database, NoSQL and Graph database (Neo4j)

Research · August 2018

DOI: 10.13140/RG.2.2.33279.25762/1

CITATIONS
0

READS
132

4 authors:

Santosh Pandey
Rosebay
4 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Erika Joshi
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Manish Maharjan
Rosebay Consulting
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Nissan Karki
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Comparing performance and data modelling in RDMS (MySQL), NoSQL and Graph database (Neo4j) for Transaction data

Relational databases have been the power-horse of software applications since the 80s, and continue so to this day but cannot cope with big data and connectedness. That brought the movement of NoSQL which solved the problems with big data but could not address the connectedness of data properly. So new solution was required to fill the void which influenced the birth of graph databases. Neo4j, a popular graph database, came with a promise of handling large and connected data with ease which puts relationship as a first priority.

You can guess that the graph database would be best for data having a network structure. But can it outperform other databases for transactional data? This blog tries to analyze different types of databases: Relational database(MySQL), NoSQL and graph database (Neo4j) on the foundation of query performance based on data size, query complexity, query number etc.

We performed an experiment which was implemented in Relational Database(MySQL), Graph Database (Neo4j) and Document Database (MongoDB). All three of these solutions will represent the same data but will do it in their own way. This allows us to quickly see the commonalities and the differences between the three solutions. We used the dataset containing details on vehicle transactions between a company and their customers through different company's branches. The initial header(attributes) of the unorganized dataset is given below which was restructured into respective schema according to the database's requirement.

	A	B	C	D	E	F	G	H	I	J	K	
1	transaction_id	counter_id	branch_id	date_of_transaction	hour	high_level_territory	staff_id	product_id	payment_amount	BeaAdmin	cust_id	name

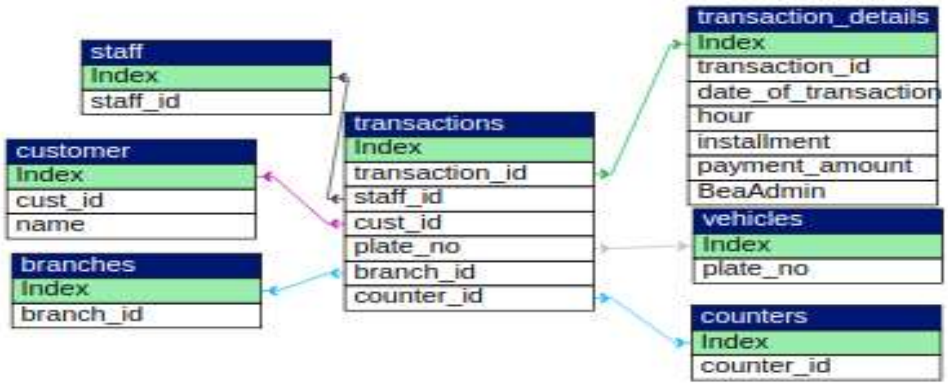
Fig: Dataset

Basic database operations were performed with the same types of queries on all three databases. These queries are run for all tested databases, irrespective of the data model they are using internally.

Relational Database

In a Relational Database, the above dataset was modelled using 7 tables.: staff, customer, branches, transactions, transaction_details, vehicles, counters.

We ended up with 7 different tables, with 6 foreign key relationships between them. Six of these tables contain actual data while one contains nothing more than links between entities of this system.



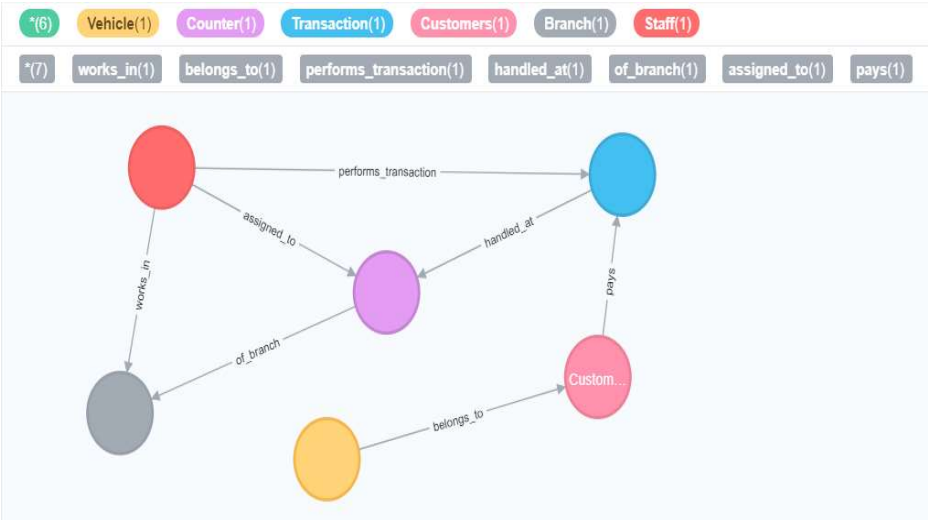
Document Store(NoSQL)

In a document store database, there are a number of different ways that this can be modelled depending on the goal. Relationships between entities of different types are difficult to achieve, either being modelled as nested documents or as a manually enforced foreign key. We used the mixture of all attributes in a single document.


```
{
  "_id" : "212342",
  "transaction_detail" : {
     "transaction_id" : "800919",
     "date_of_transaction" : "20170309",
     "hour" : "95720",
     "installment" : "5",
     "payment_amount" : "23000",
     "BeaAdmin" : "2000"
  },
  "staff" : {
     "staff_id" : "123456"
  },
  "customer" : {
     "customer_id" : "1234567890",
     "name" : "John Parsons"
  },
  "vehicle" : {
     "plate_no" : "A23BKS"
  },
  "branches" : {
     "branch_id" : "23231"
  },
  "counters" : {
     "counter_id" : "2111"
  }
}
```


Graph Database

In the graph database, we modelled the entities as Nodes and Relationships as Edges. The power of the Relational Model excludes the links between Entities which was present in the relational database model. The relationships make the traversing between the nodes easy.



Comparison Analysis

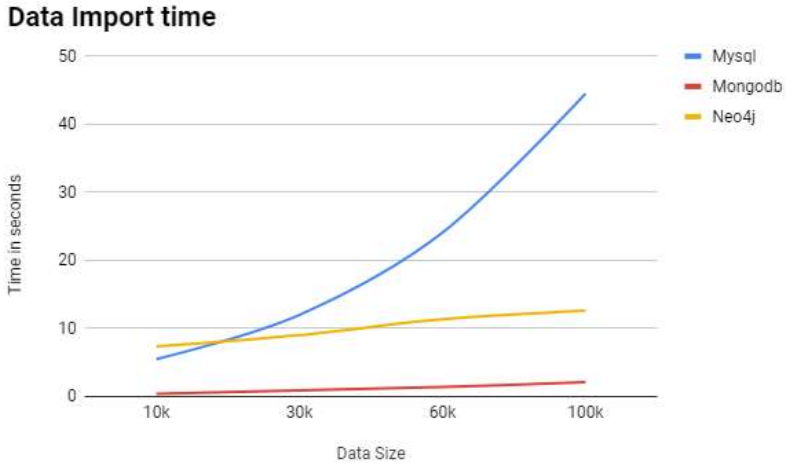


Fig : Graph for Persist time in SQL, NoSQL and GraphDB

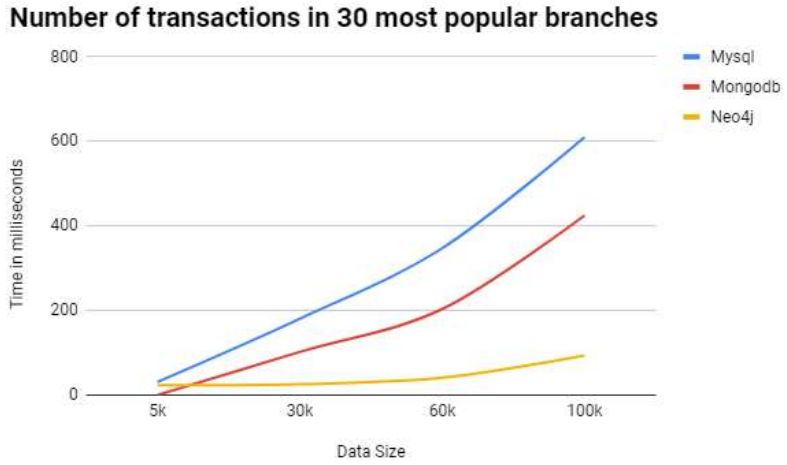


Fig: Graph for Query : No. of transaction in 30 most popular branches

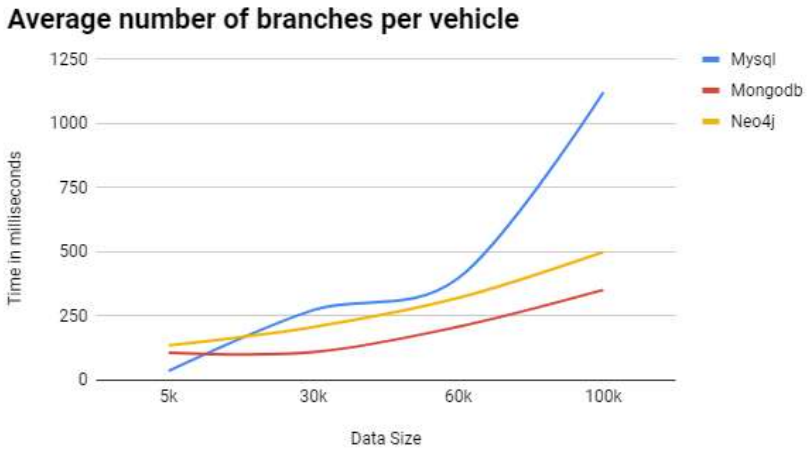


Fig: Graph for Query : Average number of branches per vehicle

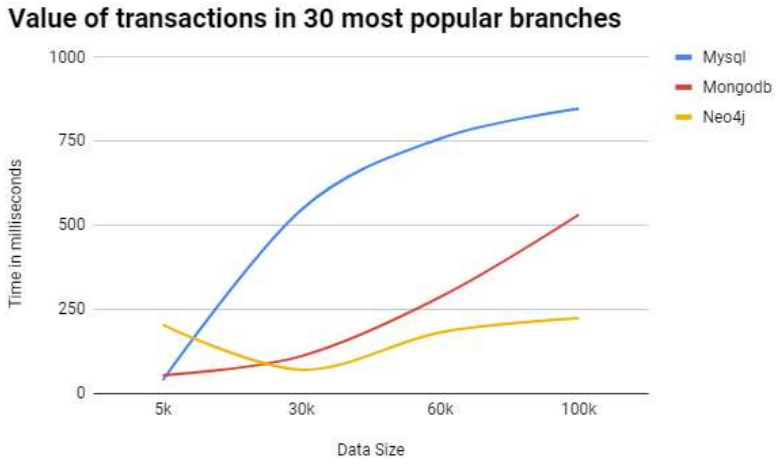


Fig: Graph for Query : Value of transactions in 30 most popular branches

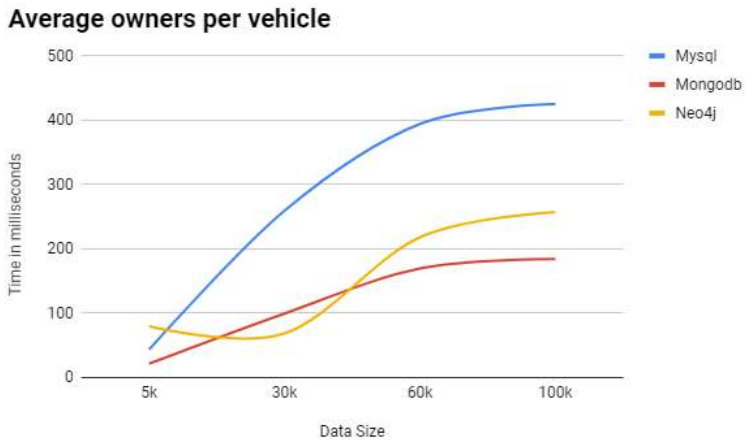


Fig: Graph for Query : Average owners per vehicle

As a reference to graph plotted above, we made a comparative analysis of persist time between SQL, NoSQL and Graph DB for different dataset size i.e 10k, 30k, 60k and 100k. The persist time for MongoDB was least and almost constant with different data sizes. SQL took the least time for 10k data size but Neo4j surpasses it after 30k data sizes. As the number of data increased, the SQL's performance decreased while MongoDB and Neo4j's processing time remained constant with the rise in the number of data.

Similarly, queries were performed in all three databases on every change of data size. SQL performance was close to MongoDB and Neo4j for 5K data size. By increasing the number of data, SQL couldn't process as fast as the rest of the databases. While querying the queries, Neo4j query run time did not increased linearly like other databases. MongoDB query performance decreased as the size of data increased. Although for some queries(i.e. Avg branches per vehicle), neo4j takes more time than others, it shows the best performance for other queries above.

From the benchmarking test performed, we found that graph databases are able to handle relationship more efficiently. NoSQL provides fast performance while working with a large number of datasets. It is superior when the data are document based without having any forms of relationships between them. For the example dataset (that we used for the test), Graph database could handle better than the relational database and NoSQL. As the graph database has index-free adjacency, it's processing time for any number of data is relatively constant. Query performance is dependent upon the no. of relationship it uses. The queries used in this experiment processes all the records in the database. But if we were to search for or process on specific data, then graph database would outperform its competitor. Neo4j can work with a specific portion of data due to its index-free adjacency while other databases need to scan the whole database.

When starting a new project, there is often a tendency to use technologies that are well known, or else that are new and well discussed. Consideration should be taken to see if these are really the best fit for what you need though, or if something else might work better for you. A Relational Database is often considered the safe option, and there is a myriad of NoSQL database solutions that get a lot of discussions online these days that may be tempting to use as well. For better performance, flexibility, the speed of iteration which is common in the NoSQL database, combined with the relational modelling power from a Relational Database then Graph Database must be considered.

But a graph database may not always be the best fit for all types of data. A graph database is good if the data are highly relational or if the data contains a number of many-to-many relationships, One-To-One or One-To-Many relationships. The flexibility of the graph database allows storing data with much more fluidity.