GRAPH DATABASE AND SQL DATABASE: A Comparative Study

http://jurnal.universitaskebangsaan.ac.id/index.php/ensains
Email: ensains@universitaskebangsaan.ac.id / ensainsjournal@gmail.com

ENSAINS: Vol. 2 Nomor. 1 Januari 2019

Lukman Hadijanto
Department of Electrical Engineering - UK

Abstract: Today's reality is that everything being more and more interconnected. And as reality being developing into that situation, the information realm has to cope with this new situation - more interconnected, more interrelated objects. One important component in managing information is database. The current established database is relational database, in which information of objects are saved in tables, and their relationship with other objects in foreign key constraints. This relational database is proven to be reliable and robust, but there is a new database development where the interconnectivity between objects being the focus. This new kind of database is called graph database.

As a new breed of database, we want to compare graph database with the established relational database. Having some works been done on this database comparison, here we focus on the subjective and objective review of usage easiness and the response time of graph database compared to relational database

As the result, querying graph database is more natural than relational database. The cypher query language is easy to understand and intuitive. Adding depth of query in cypher language is just like adding a link of a chain. On the other hand, the query response time of relational database is much better than graph database. Current implementation of graph database is still less impressive than relational database in term of time performance. Relational database is still the promising choice for time-critical application.

Keyword: Database, Comparative

INTRODUCTION

Today's reality is that many things being interconnected. Development of technology brings us to a new world where things are interconnected. People, places, web pages, buyers, sellers, products, services, etc. are interconnected through either physical connections or conceptual connections. People are interconnected physically when they make phone calls or send messages through their phones. Sellers, buyers and products are interconnected conceptually when transactions occur.

Interconnection is not a new thing, but new technologies make it much more escalated. This new paradigm is a challenge for IT professionals to cope. To model and to efficiently handle information of highly and deeply interconnected objects, we need a new tool, something that is specificaly designed to handle this particular situation.

One important component or tool in managing information is database. Up untill now, we have a quite mature database technology: the relational database, in which information of objects are saved in tables, and their relationship with other objects in foreign key constraints. Relational database has been proven to be a robust and reliable technology in handling big ammount of data efficiently and confidently. We certainly can use relational database to manage information of highly connected objects. But we also realize that there are some characteristics of highly connected objects that relational database is not very good at. For examples, in relational database we must first define the database schema before it can be used. This database schema is rather a bit rigid, any changes or additions of relations between objects result in modifying the schema.

The query language of relational database (the structured query language or SQL) is also designed with table-oriented view. This is good for querying not too many tables in one query. But for queries which involves many tables at once could make the query statements become complex and difficult to understand (with many WHERE clauses). Other requirements in managing information of highly connected objects could be some specific functions, like a function to find the shortest path of connection between two objects, or a function to check if two objects are connected indirectly or not.

The above mentioned challenges and specific requirements have result in developing a new kind of database, which called graph database. Here, we would like to investigate if this new graph database can handle interconnected objects better than relational database. We particularly focus on the easiness of querying the database and the time response while querying against it.

GRAPH DATABASE

The interconnected objects in real world can be model with graphs. A graph consists of nodes (represent objects) and binary relations depicted using arcs (represent connections between objects). Figure 1 shows an example of a graph.

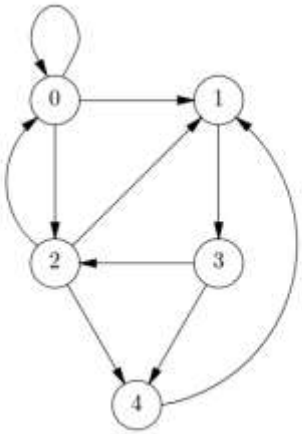

Figure 1. Graph Example

A graph database is a database which is developed with main purpose of storing graph-oriented data structures. In a graph database, data are represented by nodes, edges and properties. Data objects are represented as nodes and edges manifest the relationship between objects. Both nodes and edges can have properties that describes their particular characteristics (Patil, 2014). One implementation of graph database is Neo4j. Neo4j is an open-source NoSQL native graph database which provides an ACID-compliant transactional backend for applications (Garima, 2013). It is the most mature open-source graph database for current development.

Neo4j uses Cypher Query Language (CQL) as the query language, equivalent with SQL for relational database. Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store.

Cypher is designed to be a humane query language. Its constructs are based on English prose and neat iconography which helps to make queries more self-explanatory. Being a declarative language, Cypher focuses on the clarity of expressing what to retrieve from a graph, not on how to retrieve it.

DESIGNING THE COMPARATIVE TEST

The graph database used in this test is Neo4j version 1.6. Neo4j is the most actively developed graph database and the most mature graph database implementation. Neo4j is also developed with open-source scheme and has a lot support from open source community. For the relational database, we use MySql version 5.5.16. MySql has been developed since 1995 and been widely used in many applications. It is proven to be a stable and reliable database, has a large support from community and from the leading developer company.

To illustrate a highly interconnected environment, we simulate phone call connections between phone users. In reality, this call connections are recorded by a phone company in a database called Call Data Record (CDR). CDR records caller numbers, destination numbers, date & time, call duration, connection types, and other phone identification (like IMSI and EMSI for mobile phones).

We simulate this phone call connections by creating some phone numbers randomly, and some phone connections randomly too.

To investigate wether or not size of sample data has impact to test result, we create several combinations of number of phones and number of call connections. The combinations are as follow:

Table 1. Test Sample Combination

Test Sample	Number of Phones	Number of Calls
1	100	100
2	100	1.000
3	1.000	1.000
4	1.000	5.000
5	1.000	10.000

We use Neo4j-Shell as the interface to interact with Neo4j database. To create nodes as representation of phones, we use command: mknode, for example: mknode Phone --np "{'number': '8326309'}". And to create relations between nodes as representation of phone calls, we use command: mkrel, for example: mkrel -d

OUTGOING -t Calls 66 (note: we need to "move" to the initiating node first, by using command: cd -a ([node_number]))
As for MySQL database, we simply create two tables: one table for recording phone numbers and the other one for recording phone calls (the transaction table).
Then, we run query against these two types of database. We want to investigate query response time of different query depth. What we mean with "depth" here is the number of call connection between two phone numbers. For example: if A calls B, depth = 1; if A calls B and B calls C, depth = 2; if A calls B and B calls C and C calls D, depth =3; and so on. We limit the query depth to five.
For querying Neo4j, we use Cypher Query Language (CQL). The command to execute the query of depth=1 is:

```
start a = node ([node_number])
a-[:Calls]->b
return a, b
```

And the command for query of depth=2 is:

```
start a = node([node_number])
a-[:Calls]->b-[:Calls]->c
return a, b, c
```

Query statements for depth = 3, 4, or 5 can be constructed with the same pattern.
The SQL query of depth=1 for our relational database in MySQL is:

```
SELECT call_from, call_dest
FROM call_txn
WHERE call_from = [phone_number]
```

And SQL query of depth=2 is:

```
SELECT a.call_from, b.call_from, b.call_dest
FROM call_txn a, call_txn b
WHERE a.call_from = [phone_number]
AND a.call_dest = b.call_from
```

SQL query statements for depth = 3, 4, or 5 can be constructed with the same pattern.
TEST RESULT AND ANALYSIS
As we insert simulation data into graph database, the interconnected data can be visualized as graph as the following Figure 2:

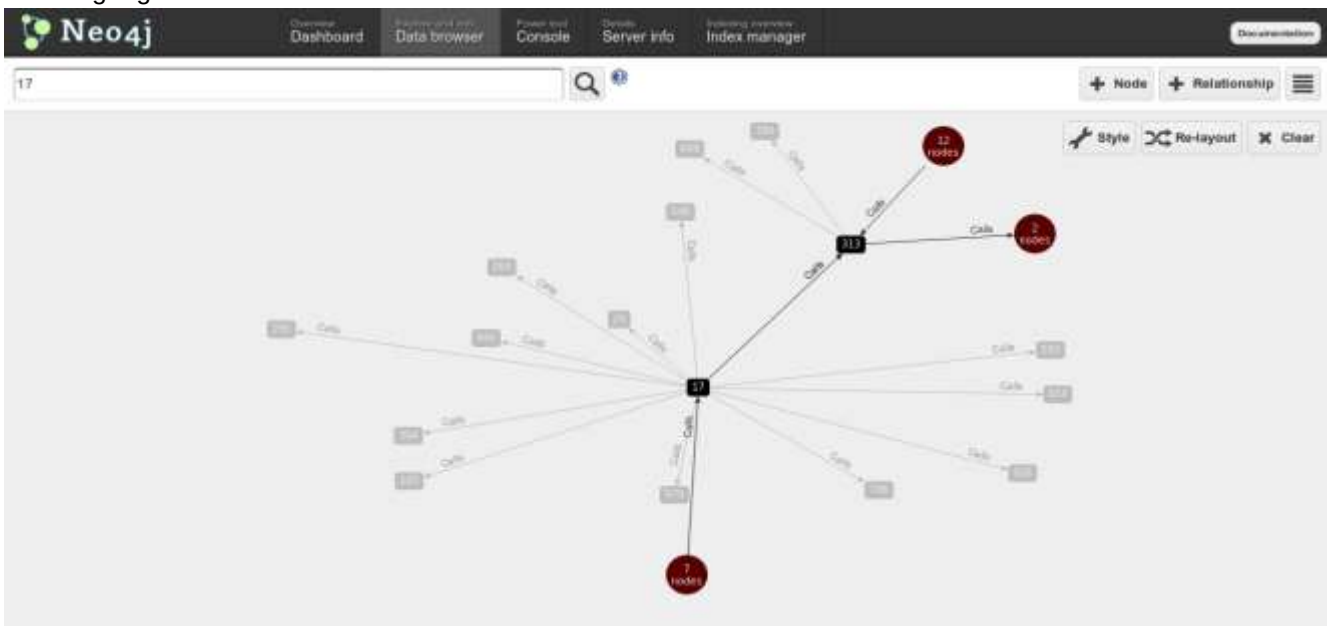

Figure 2. Data Visualization

1.1 Query Statements

Test results show that both databases give the expected result sets as queries are executed against them. As described in section 3, the test is designed to investigate how the database handle connections between phones. SQL is used to query relational database, while CQL is used to query graph database. By comparing the SQL query statements with CQL query statements, we can see that CQL is more natural for querying heavily connected objects in the database. The CQL query statement uses notations that can intuitively be interpreted as the nodes and relations of a graph. And as the depth of query are increased, we can easily add a new "node" to the query statement. It is just like adding a link to a chain.

1.2 Query Time Response

The following table shows the query time results for each test sample (note: the header (100, 100) means the sample of 100 phone numbers and 100 phone calls, and the same for the other headers):

Table 2. Query Time Test Result

(query time, in milli-seconds)

	(100,100)		(100,1000)		(1000,1000)		(1000,5000)		(1000,10000)	
Depth	Rel DB	Graph DB	Rel DB	Graph DB	Rel DB	Graph DB	Rel DB	Graph DB	Rel DB	Graph DB
1	0.4	1.0	0.5	8.0	0.2	1.0	0.5	3.0	0.6	1.0
2	0.6	14.0	1.1	44.0	0.6	2.0	0.9	20.0	1.1	10.0
3	0.7	4.0	1.0	97.0	0.8	4.0	1.3	88.0	1.2	93.0
4	0.7	4.0	1.1	584.0	0.9	8.0	2.0	173.0	1.1	768.0
5	0.8	4.0	1.2	5,678.0	0.9	7.0	1.3	325.0	2.5	10,396.0

The test results show the query time of graph database is slower than relational database. For each query being executed, the Neo4j-Shell - used as the interface to Neo4j - prints out all results on screen while the php My Admin only shows the first 20 records, with a link to see the rest. This could make query time of Neo4j become slower a little bit. But even if we take that into account, the query time of MySQL database is still significantly faster.

If we compare the query time of Neo4j of one test sample to another, we also can see that the query time of Neo4j of more connected objects is slower than of the less connected objects. For example, the query time of Neo4j for sample (100, 1000) is slower than for sample (100, 100); and the query time for sample (1000,10000) is slower than for sample (1000,1000).

The reason of this slower query time of Neo4j could be the fundamental design of graph database itself. In response to a query, graph database will traverse from one node to another linked node and check the node if it satisfied the query constraints. By contrast, relational database uses the WHERE clause to narrow down the size of data to be checked. Relational database also heavily uses indexes, this mechanism can surely reduce the time needed in responding to a query.

One other interesting result is that query time for Neo4j also depends on which node being selected as starting node. We can see this when comparing the query time of Neo4j for sample (100, 1000) with sample (1000, 1000). The test result shows that query time for sample (100.1000) is slower than for sample (1000, 1000), even though there are more objects in the second sample. The test of these two samples are using different starting nodes. If the starting node has many connections then the query time becomes slower, and vice versa. Again, this characteristic is the result of how graph database is designed.

Another factor that we should consider in comparing graph database to relational database is their development maturity. Relational database is very much more mature comparing to graph database, resulting in better implementation, more efficient, faster software. Graph database is still in its growth phase, so at current implementation, it needs some improvement of its performance.

CONCLUSIONS

A comparative test between relational database and graph database has been conducted. The test focus on subjective and objective review. The subjective review is on the easiness of making and understanding the query statements, while the objective review is on the query time of both databases.

Querying graph database is more natural than relational database. The cypher query language is easy to understand and more intuitive. Adding depth of query in cypher language is just like adding a link of a chain.

The query response time of relational database is much better than graph database. Current implementation of graph database is still less impressive than relational database in term of time performance. Relational database is still the promising choice for time-critical application

REFERENCES

"What is a Graph Database", available at https://neo4j.com/developer/graph-database

Garima Jaiswal, Arun Prakash Agrawal, "Comparative Analysis of Relational and Graph Databases", IOSR Journal of Engineering Vol.3 Issue 8, 2013

Patil, Vaswani, Bhatia, "Graph Databases - An Overview", International Journal of Computer Science and Information Technologies, 2014