Benchmarking performance for Neo4j in a Social Media Application

by

Michael Sheroubi

Supervisor: Dr. Ramon Lawrence

The Irving K. Barber School of Arts and Sciences

Undergraduate Degree in Bachelor of Science

Honours Major in Computer Sciences
And
Minor in Mathematics

The University of British Columbia – Okanagan Campus

April 2020

Abstract

Starting around the mid-late 2000s, ACID compliant graph databases such as Neo4j began being used in an attempt to optimize data retrieval in use cases where relational databases struggled. This paper attempts to benchmark the performance of two widely used databases, one from each category, for a social media application. Logging the process from data modelling and generation to writing queries to answer real questions. The performance of the queries is evaluated based on the rate of increase in time as the sample sizes grow. Concluding that the relational database is faster at a small scale, but the rate of increase is slower for the graph database, meaning that a point exists at a sufficiently large sample size where the graph database becomes more efficient.

Table of Contents

1. Introduction

There has been a rise in NoSQL databases over the last few years. More developers are realising that the relational model can be limiting or suboptimal for their applications. This led to the formation of many different methods of storing data. One of these methods revolves around utilizing a graph model derived from graph theory. This is the Graph Database.

Released in 2007, Neo4j is an open-source, NoSQL, graph database created to efficiently utilize the property graph model as a means of storage. Graph databases are meant to excel at managing highly connected data and managing complicated queries [1], allowing for quick traversal between adjacent nodes and easy visualization of the data structure. With its own query language, CYPHER, Neo4j queries are designed to be visually intuitive and simple to create. We will explore how Neo4j's graph approach to storing data and querying data performs in its ideal setting and compare its performance with a relational database.

2. Background

2.1 Introduction to Graph Theory

Graph Theory is a field of mathematics that has been heavily integrated into various software applications. First noted in 1736 when Leonhard Euler, a swiss mathematician, worked on solving the *Seven Bridges of Königsberg* problem [2]. Given four landmasses separated by a river and connected with seven bridges (Figure 1), is there a path where one can cross every bridge exactly once? While a solution does not exist, Euler's proof for its non-existence is the foundation for graph theory.

Graphs represent entities as nodes/vertices and the ways in which those entities relate to other entities as relationships/edges. Mapping the seven bridges problem to a graph of this form with each landmass as a node and each bridge as relationship.

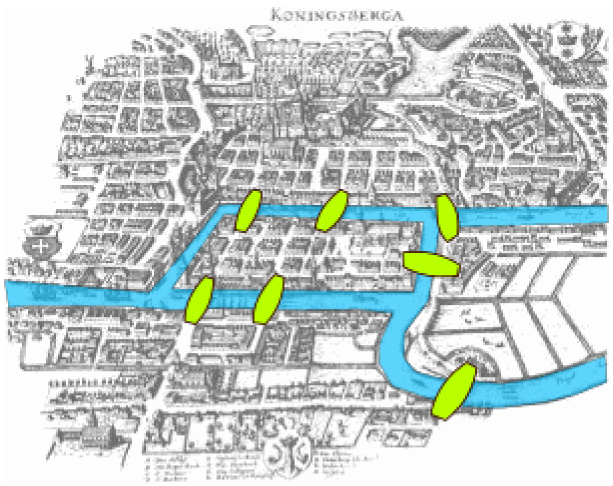


Figure 1. Seven Bridges of *Königsberg*

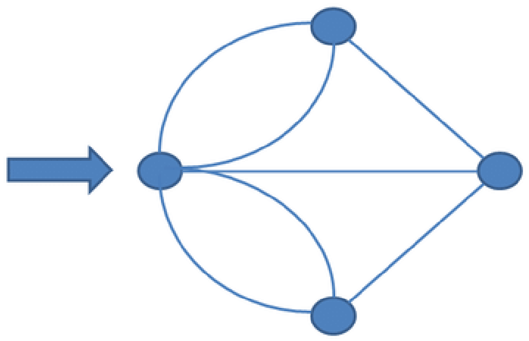


Figure 2. *Königsberg* problem as a graph

By the definition of the problem, Euler deduced that every node must have an even number of edges connected to it, one to enter the node and one to leave it, with the exception of the first and last nodes (def. *Eulerian Path*). Since each node in this problem has an odd number of edges, a solution cannot exist.

This proof established the foundation for what is now known as graph theory. Now, graph theory is an important field of study in Computer Science and Mathematics. It has a variety of real-world applications with many books and resources available that dive deeper into the subject matter.

2.2 The Labeled Property Graph Model

For our application, we will be focusing on the Labeled Property graph model, a graph model where nodes can contain properties as key-value pairs and nodes can have one or more labels. In this model, relationships can also contain properties as key-value pairs, have labels and directions, but they must always have a node on both ends. The labels on the nodes and relationships make it easy to draw the context from the data [3].

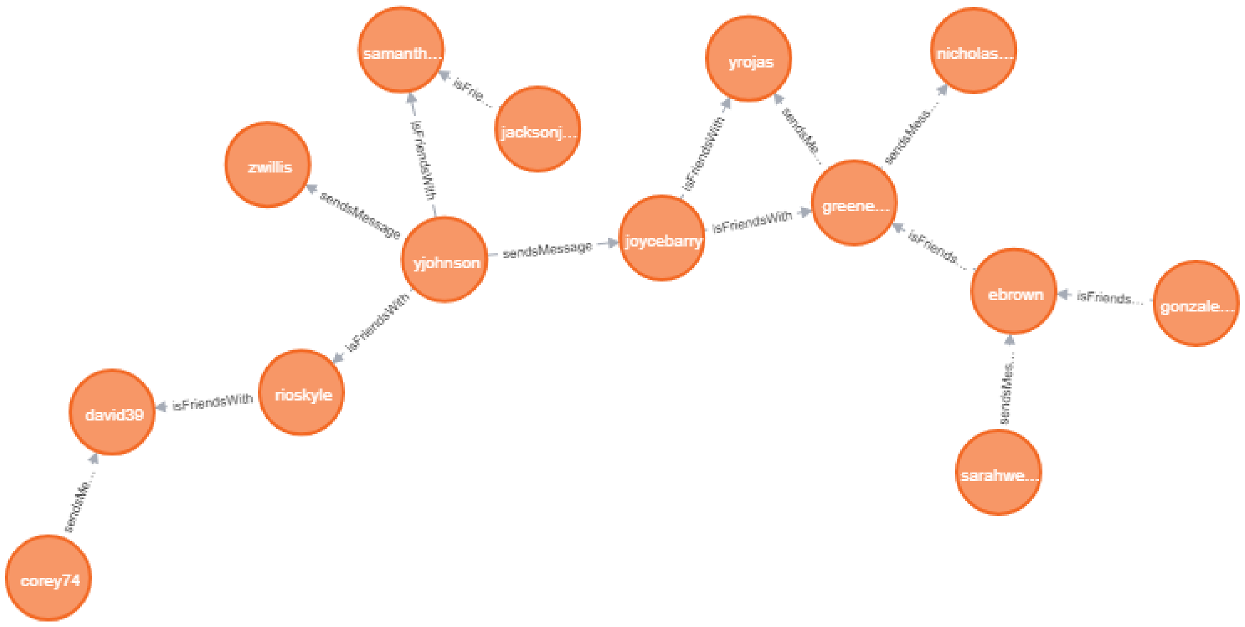


Figure 3. Example graph of User nodes and their relationships

2.2 Overview of Graph Databases

A graph database is a database that is designed to store data using the graph model. By definition, a graph database is any database where connected elements are linked together without an index or key.

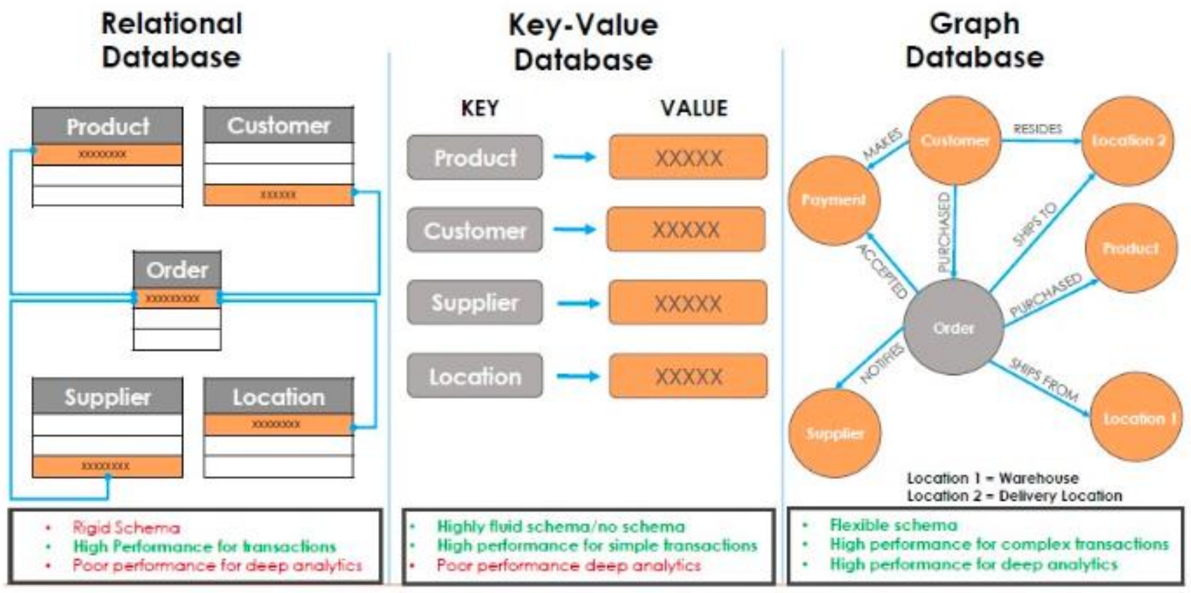


Figure 4. Comparing Data Models (https://www.nextplatform.com/2018/09/19/the-graph-database-poised-to-pounce-on-the-mainstream/)

A graph database is a NoSQL database designed to work around certain limitations in relational databases. Graph databases allow quick and easy retrieval of data from complex data structures. Traversing relationships in a graph database is fast because the relationship between nodes are not calculated at query times but are persisted in the database. See *The Power of Graph Databases* in O`Reilly's *Graph Databases: 2nd Edition*.

2.3 YCSB

Yahoo! Cloud Serving Benchmark (YCSB) is an open-source framework most used for evaluating the capabilities of NoSQL database management systems. This framework runs a set of workloads to evaluate their performance. While it does have a support for various existing NoSQL databases, it does not have support for Neo4j. More can be found here: (https://github.com/brianfrankcooper/YCSB/wiki).

3. Social Media Application

3.1 Project Summary

This project is based off a real social media application that a colleague was developing. This social media site is designed to match and connect people from communities that share similar interests. Users can find groups that match specific interests and similarly can find events for a given interest. However, the main goal is to connect users with similar interests.

Each user is prompted to input their interests and location, and an algorithm matches users by interests and proximity. Users can join Groups, attend Events and become friends with other Users. This project will focus on the database aspect of this social media site. It will encompass the data modelling, application architecture decisions, testing, capacity planning, and importing/loading of bulk data (and in this case data generation). After the database is setup, the focus will shift towards designing ad hoc queries ("saved cypher queries") to answer questions posed by the social media (e.g. matching common interests). This project does **not** examine the integration of the database into the website or any front-end programming.

3.2 ER Diagrams

3.2.1 Original Design

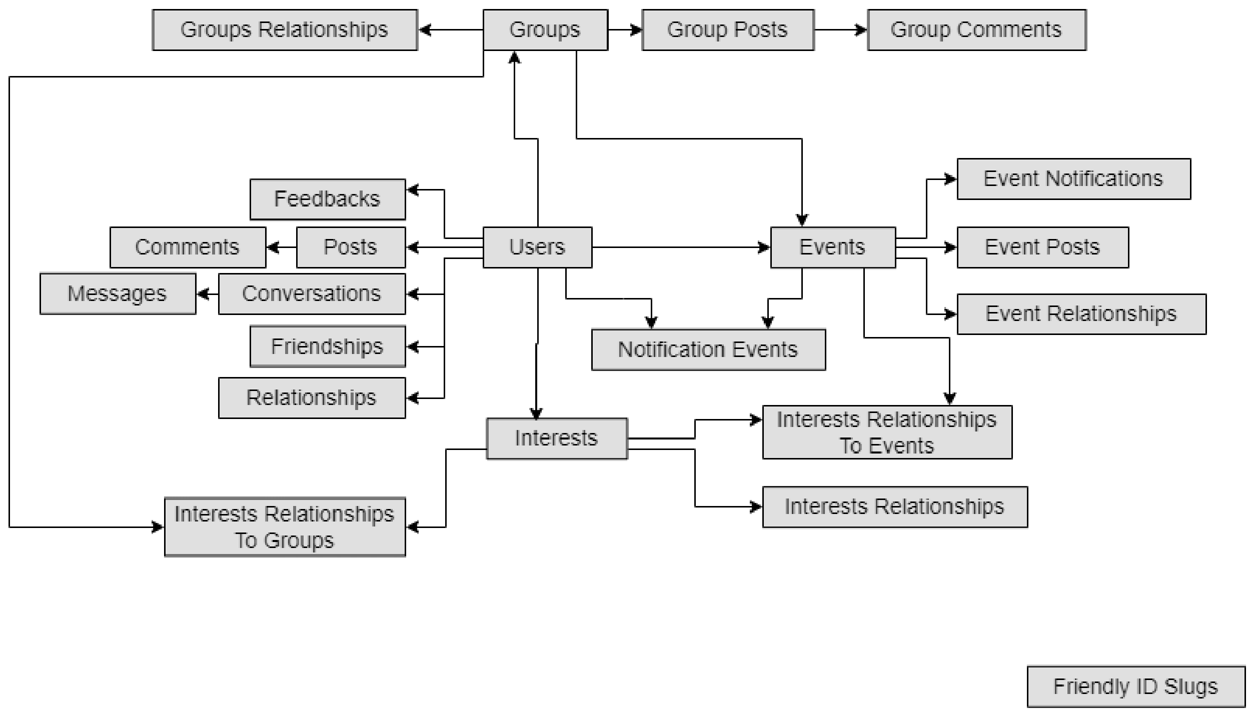


Figure 5. Original ER Diagram made for the application

This ER diagram is made from the DDL schema file I first received for the project. There were attributes in some entities that were only needed for production. The cardinalities between the entities were not defined.

3.2.2 Updated Version

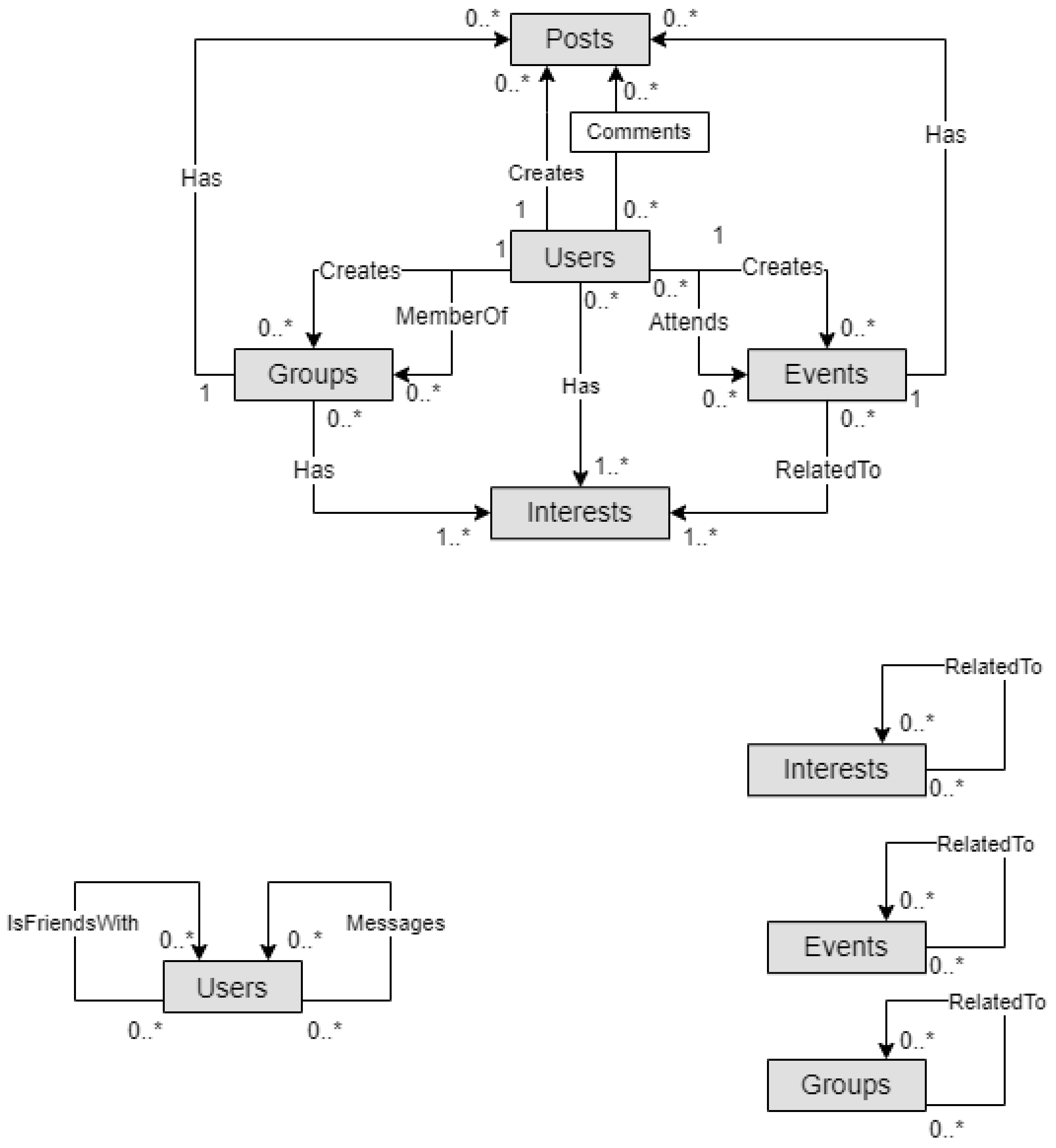


Figure 6. Updated ER Diagram designed for the purpose of this thesis

This is an updated ER diagram designed to fit the requirements for the testing for the purposes of this thesis. The bottom entities are referencing the same entities as the main diagram, but showing how each entity relates to itself (Done to keep the diagram clean and easy to read).

4. Data Modelling

4.1 Mapping Relational to Graph Model

The following table shows each relation in the relational database from the original design, along with its role in a graph-model. If more than one type is present, then there are two possibilities with the first one being more favored. This table is later trimmed to fit the updated version of the design.

Relation	Type	Dependency	Purpose
Users	Node	N/A	Holds User data
Feedbacks	Node/Property	User	User feedback on site
Posts	Node/Relationship	User	User's personal posts
Comments	Relationship	Post, User	Comments on a User Post
Conversations	Relationship	Users	User-to-User Conversations
Messages	Property	Conversation	Content of Conversations
Friendships	Relationship	Users	User-to-User
Relationships	Relationship	Users	User-to-User
Interests	Node	User?	Interest Information
Interest Relationships to Groups	Relationship	Interest, Group	Interest-to-Group
Interest Relationships to Events	Relationship	Interest, Event	Interest-to-Event
Interest Relationships	Relationship	Interests	Interest-to-Interest
Groups	Node/Label	User?	Group Users
Groups Relationships	Relationship	Groups	Group-to-Group
Group Posts	Node/Relationship	Group, User	Posts made to group
Group Comments	Relationship	Group Post, User	Comments on Group Post
Events	Node	Group, User	Event information hosted by a Group & User
Event Notifications	Relationship	Event, User	Notifications for

Relation	Type	Dependency	Purpose
			Users about events
Notification Events	Relationship	Users, Event	User-to-User event invitation notification
Event Relationships	Relationship	Events	Event-to-Event relationship
Event Posts	Property/Node	Event, User	Users posts under Events
***Friendly ID Slugs**	Node	N/A	ID-to-String URL Addon

Graph Modelling

The graph model of this relational database will have these two main components:

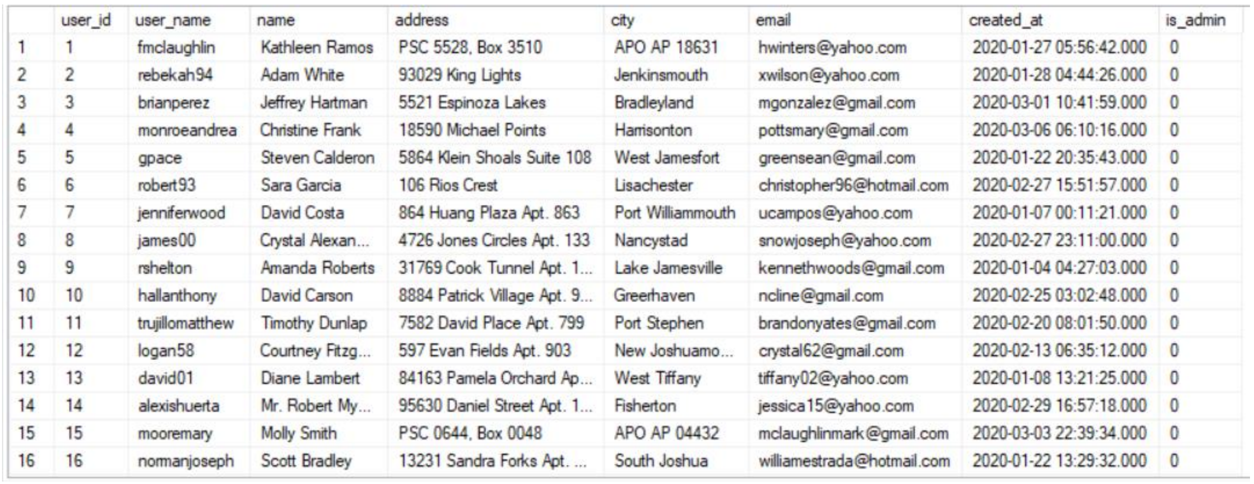
- Nodes
  - Users
  - Groups
  - Events
  - Interests
  - Posts

- Relationships
  - Comments
  - Converses
  - Notifies
  - Creates
  - Etc.

4.2 Sample Data

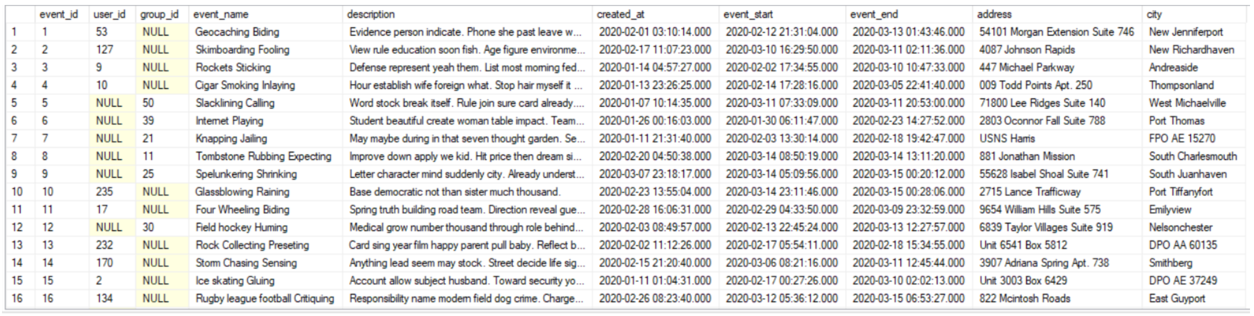
Relational Model

Users



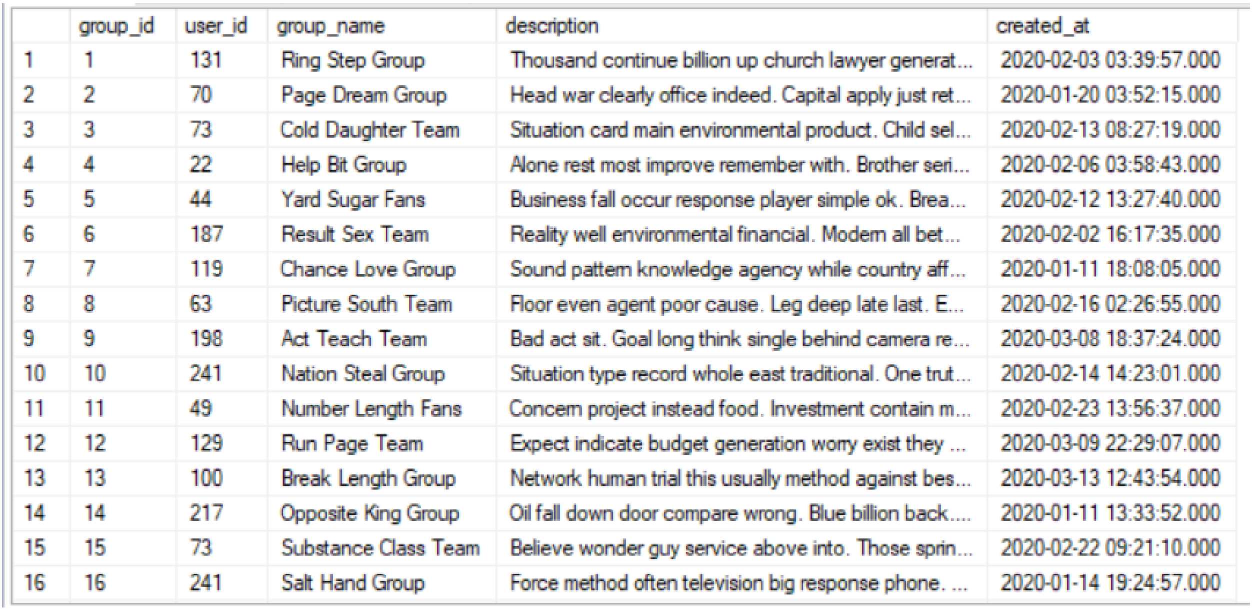
	user_id	user_name	name	address	city	email	created_at	is_admin
1	1	fmclaughlin	Kathleen Ramos	PSC 5528, Box 3510	APO AP 18631	hwinters@yahoo.com	2020-01-27 05:56:42.000	0
2	2	rebekah94	Adam White	93029 King Lights	Jenkinsmouth	xwilson@yahoo.com	2020-01-28 04:44:26.000	0
3	3	brianperez	Jeffrey Hartman	5521 Espinoza Lakes	Bradleyland	mgonzalez@gmail.com	2020-03-01 10:41:59.000	0
4	4	monroeandrea	Christine Frank	18590 Michael Points	Harrisonton	pottsmary@gmail.com	2020-03-06 06:10:16.000	0
5	5	gpace	Steven Calderon	5864 Klein Shoals Suite 108	West Jamesfort	greensean@gmail.com	2020-01-22 20:35:43.000	0
6	6	robert93	Sara Garcia	106 Rios Crest	Lisachester	christopher96@hotmail.com	2020-02-27 15:51:57.000	0
7	7	jenniferwood	David Costa	864 Huang Plaza Apt. 863	Port Williammouth	ucampos@yahoo.com	2020-01-07 00:11:21.000	0
8	8	james00	Crystal Alexan...	4726 Jones Circles Apt. 133	Nancystad	snowjoseph@yahoo.com	2020-02-27 23:11:00.000	0
9	9	rshelton	Amanda Roberts	31769 Cook Tunnel Apt. 1...	Lake Jamesville	kennethwoods@gmail.com	2020-01-04 04:27:03.000	0
10	10	hallanthony	David Carson	8884 Patrick Village Apt. 9...	Greerhaven	ncline@gmail.com	2020-02-25 03:02:48.000	0
11	11	trujillomatthew	Timothy Dunlap	7582 David Place Apt. 799	Port Stephen	brandonyates@gmail.com	2020-02-20 08:01:50.000	0
12	12	logan58	Courtney Fitzg...	597 Evan Fields Apt. 903	New Joshuamo...	crystal62@gmail.com	2020-02-13 06:35:12.000	0
13	13	david01	Diane Lambert	84163 Pamela Orchard Ap...	West Tiffany	tiffany02@yahoo.com	2020-01-08 13:21:25.000	0
14	14	alexishuerta	Mr. Robert My...	95630 Daniel Street Apt. 1...	Fisherton	jessica15@yahoo.com	2020-02-29 16:57:18.000	0
15	15	mooremary	Molly Smith	PSC 0644, Box 0048	APO AP 04432	mclaughlinmark@gmail.com	2020-03-03 22:39:34.000	0
16	16	normanjoseph	Scott Bradley	13231 Sandra Forks Apt. ...	South Joshua	williamestrada@hotmail.com	2020-01-22 13:29:32.000	0

Events



	event_id	user_id	group_id	event_name	description	created_at	event_start	event_end	address	city
1	1	53	NULL	Geocaching Biding	Evidence person indicate. Phone she past leave w...	2020-02-01 03:10:14.000	2020-02-12 21:31:04.000	2020-03-13 01:43:46.000	54101 Morgan Extension Suite 746	New Jenniferport
2	2	127	NULL	Skimboarding Fooling	View rule education soon fish. Age figure environme...	2020-02-17 11:07:23.000	2020-03-10 16:29:50.000	2020-03-11 02:11:36.000	4087 Johnson Rapids	New Richardhaven
3	3	9	NULL	Rockets Sticking	Defense represent yeah them. List most morning fed...	2020-01-14 04:57:27.000	2020-02-02 17:34:55.000	2020-03-10 10:47:33.000	447 Michael Parkway	Andreaside
4	4	10	NULL	Cigar Smoking Inlaying	Hour establish wife foreign what. Stop hair myself it ...	2020-01-13 23:26:25.000	2020-02-14 17:28:16.000	2020-03-05 22:41:40.000	009 Todd Points Apt. 250	Thompsonland
5	5	NULL	50	Slacklining Calling	Word stock break itself. Rule join sure card already....	2020-01-07 10:14:35.000	2020-03-11 07:33:09.000	2020-03-11 20:53:00.000	71800 Lee Ridges Suite 140	West Michaelville
6	6	NULL	39	Internet Playing	Student beautiful create woman table impact. Team...	2020-01-26 00:16:03.000	2020-01-30 06:11:47.000	2020-02-23 14:27:52.000	2803 Oconnor Fall Suite 788	Port Thomas
7	7	NULL	21	Knapping Jailing	May maybe during in that seven thought garden. Se...	2020-01-11 21:31:40.000	2020-02-03 13:30:14.000	2020-02-18 19:42:47.000	USNS Harris	FPO AE 15270
8	8	NULL	11	Tombstone Rubbing Expecting	Improve down apply we kid. Hit price then dream si...	2020-02-20 04:50:38.000	2020-03-14 08:50:19.000	2020-03-14 13:11:20.000	881 Jonathan Mission	South Charlesmouth
9	9	NULL	25	Spelunkering Shrinking	Letter character mind suddenly city. Already underst...	2020-03-07 23:18:17.000	2020-03-14 05:09:56.000	2020-03-15 00:20:12.000	55628 Isabel Shoal Suite 741	South Juanhaven
10	10	235	NULL	Glassblowing Raining	Base democratic not than sister much thousand.	2020-02-23 13:55:04.000	2020-03-14 23:11:46.000	2020-03-15 00:28:06.000	2715 Lance Trafficway	Port Tiffanyfort
11	11	17	NULL	Four Wheeling Biding	Spring truth building road team. Direction reveal gue...	2020-02-28 16:06:31.000	2020-02-29 04:33:50.000	2020-03-09 23:32:59.000	9654 William Hills Suite 575	Emilyview
12	12	NULL	30	Field hockey Huming	Medical grow number thousand through role behind...	2020-02-03 08:49:57.000	2020-02-13 22:45:24.000	2020-03-13 12:27:57.000	6839 Taylor Villages Suite 919	Nelsonchester
13	13	232	NULL	Rock Collecting Preseting	Card sing year film happy parent pull baby. Reflect b...	2020-02-02 11:12:26.000	2020-02-17 05:54:11.000	2020-02-18 15:34:55.000	Unit 6541 Box 5812	DPO AA 60135
14	14	170	NULL	Storm Chasing Sensing	Anything lead seem may stock. Street decide life sig...	2020-02-15 21:20:40.000	2020-03-06 08:21:16.000	2020-03-11 12:45:44.000	3907 Adriana Spring Apt. 738	Smithberg
15	15	2	NULL	Ice skating Gluing	Account allow subject husband. Toward security yo...	2020-01-11 01:04:31.000	2020-02-17 00:27:26.000	2020-03-10 02:02:13.000	Unit 3003 Box 6429	DPO AE 37249
16	16	134	NULL	Rugby league football Critiquing	Responsibility name modern field dog crime. Charge...	2020-02-26 08:23:40.000	2020-03-12 05:36:12.000	2020-03-15 06:53:27.000	822 Mcintosh Roads	East Guyport

Groups

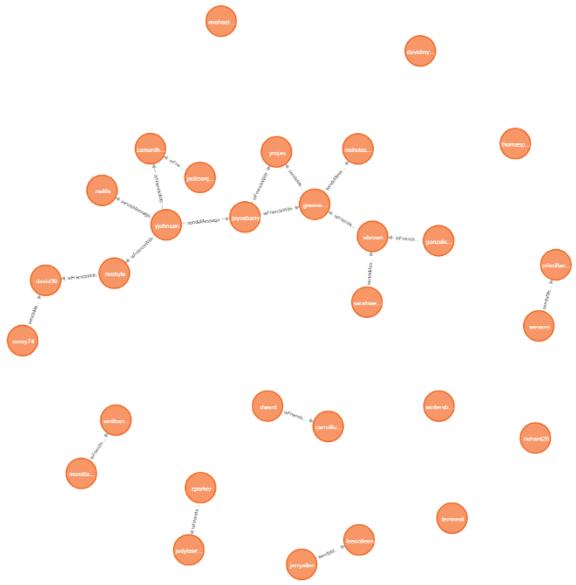


	group_id	user_id	group_name	description	created_at
1	1	131	Ring Step Group	Thousand continue billion up church lawyer generat...	2020-02-03 03:39:57.000
2	2	70	Page Dream Group	Head war clearly office indeed. Capital apply just ret...	2020-01-20 03:52:15.000
3	3	73	Cold Daughter Team	Situation card main environmental product. Child sel...	2020-02-13 08:27:19.000
4	4	22	Help Bit Group	Alone rest most improve remember with. Brother seri...	2020-02-06 03:58:43.000
5	5	44	Yard Sugar Fans	Business fall occur response player simple ok. Brea...	2020-02-12 13:27:40.000
6	6	187	Result Sex Team	Reality well environmental financial. Modern all bet...	2020-02-02 16:17:35.000
7	7	119	Chance Love Group	Sound pattern knowledge agency while country aff...	2020-01-11 18:08:05.000
8	8	63	Picture South Team	Floor even agent poor cause. Leg deep late last. E...	2020-02-16 02:26:55.000
9	9	198	Act Teach Team	Bad act sit. Goal long think single behind camera re...	2020-03-08 18:37:24.000
10	10	241	Nation Steal Group	Situation type record whole east traditional. One trut...	2020-02-14 14:23:01.000
11	11	49	Number Length Fans	Concern project instead food. Investment contain m...	2020-02-23 13:56:37.000
12	12	129	Run Page Team	Expect indicate budget generation worry exist they ...	2020-03-09 22:29:07.000
13	13	100	Break Length Group	Network human trial this usually method against bes...	2020-03-13 12:43:54.000
14	14	217	Opposite King Group	Oil fall down door compare wrong. Blue billion back....	2020-01-11 13:33:52.000
15	15	73	Substance Class Team	Believe wonder guy service above into. Those sprin...	2020-02-22 09:21:10.000
16	16	241	Salt Hand Group	Force method often television big response phone. ...	2020-01-14 19:24:57.000

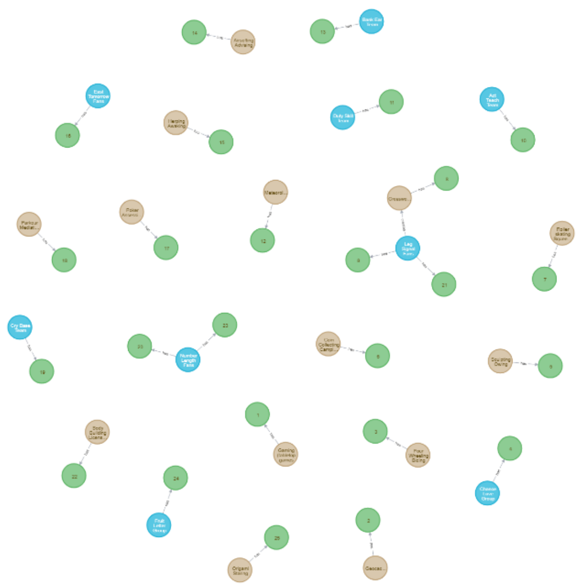
Figures 7, 8, 9. Sample data taken from SQL Server after data generation and loading

Graph Model

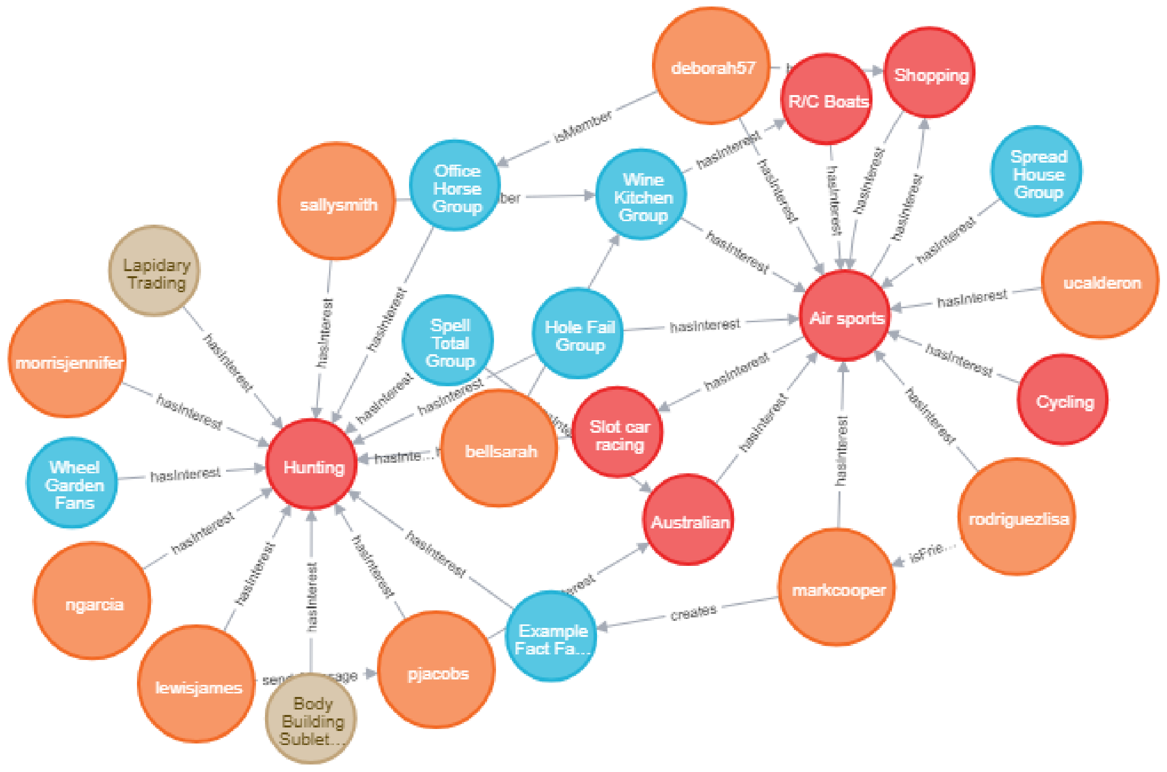
Node - User



Relationship- Has



Figures 10, 11, 12. Sample data taken from Neo4j after data generation and loading



Query: “<code>MATCH p=()-[r:hasInterest]->() RETURN p LIMIT 25</code>“

4.3 Data Generation

Data Generation Script

Language: Python 3
Libraries: Faker, random
Dependencies: common-verbs.txt common-nouns.txt common-interests.txt

The data generation is divided up into different functions, each responsible for generating data for a specific table or node. Some cypher relationships are generated alongside the nodes they connect to ensure consistency between SQL Foreign keys and Cypher relationships. Each function can take a set of parameters; (n) is the number of tuples the function should create, (x, y, z) each holds the number of one of (Users, Groups, Events, Interests, Posts) to make sure that a relationship or foreign key does not reference a node that does not exist. Each table/node is indexed by an auto-incremented integer. The script creates three files; sql_file, cypher_node_file, cypher_rel_file. The sql_file contains the INSERTS for every table in order to watch for dependencies. The cypher_node_file contains the CREATE node statements for every node, and the cypher_rel_file contains the MATCH CREATE statements that match the two nodes to connect, then creates a relationship between them. The final function header generate All Data takes in a number for each table and generates the data accordingly.

Relational Model
DBMS: SQL SERVER Management Studio
Host: localhost
Related Files: sql_data.sql
Graph Model
DBMS: Neo4J Browser
Host: localhost
Related Files: cypher_node_data.cql cypher_rel_data.cql

TEST CASE

Users: 250 | Groups: 50 | Interests: 100 | Events: 100 | Posts: 500 | isMember: 500 | areFriends: 500 | comments: 750 | hasInterest: 1000 | isAttending: 500 | messages: 750 |

NOTES

- Data loads/insertion is exponentially faster in SQL Server than Neo4j
- Creating 100 nodes in Neo4j using browser took about 2 minutes
- Creating 365 relationships in Neo4j using its browser took 50+ minutes (55:11) – Don't use Neo4j Browser

4.4 Data Loading

As part of the testing script, the data was loaded into the SQL Server instance and the Neo4j graph in increments while recording run-time per transaction and other factors. In total, up to 1000000 (1e6) rows of data generated for each database. There was a slight disparity in the number of row rows of data that is mainly attributed to foreign keys not counting as a full row of data in the relational model, as opposed to in the graph model, a relationship still has to be defined even if it has no parameters. Therefore, some relationships were added as separate transactions into the Neo4j inserts.

5. Testing and Results

The following results are measuring the time (in milliseconds) to execute groups of transactions. The time displayed is an average calculated from repeating the same test 100 times.

5.1 Inserting Data

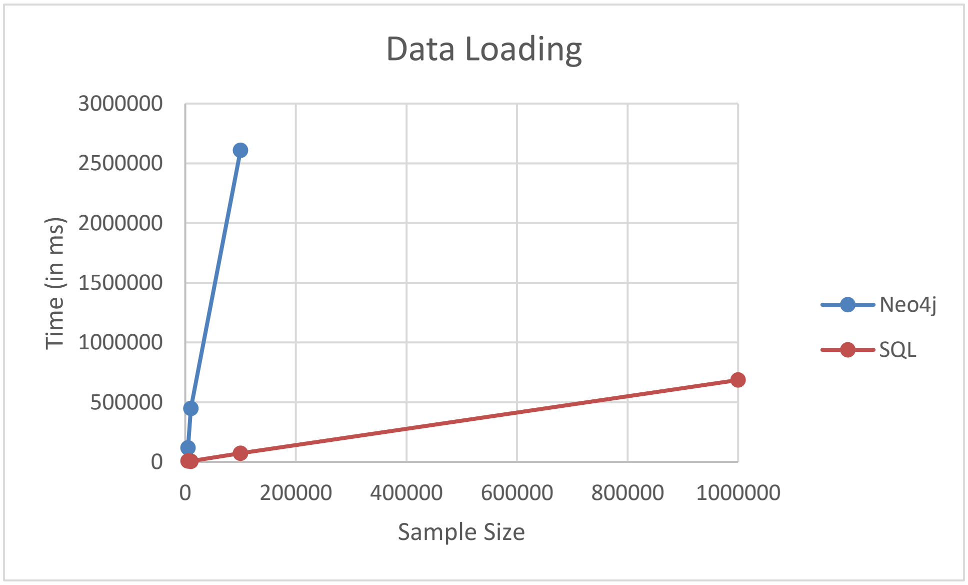


Figure 13. Time to load data by sample size (Last data point for Neo4j omitted)

Loading data into SQL Server takes significantly less time than loading into Neo4j. The time grows linearly with the sample size for SQL and grows exponentially for Neo4j. This could be attributed to the data being loaded one element at a time, as neo4j has alternative methods of loading data. However, this method was used to keep the tests consistent.

5.2 Queries

The times shown in the charts below are different for Neo4j and SQL. The points for Neo4j show the time per query, while the points for SQL show time per 100 queries. For the sample sizes used, there is no doubt that SQL is far more efficient. But as the application starts to scale, we want to know how each model will perform. So, we scaled the SQL times up to be able to plot both the Neo4j and SQL times on the same charts. This way we can clearly compare the linearity of the growth as the sample size increases.

Q0 - Find specific User by user_id

```
SQL - SELECT * FROM Users WHERE user_id = {};
CYPHER - MATCH (u:User) WHERE u.user_id = {} RETURN u;
```

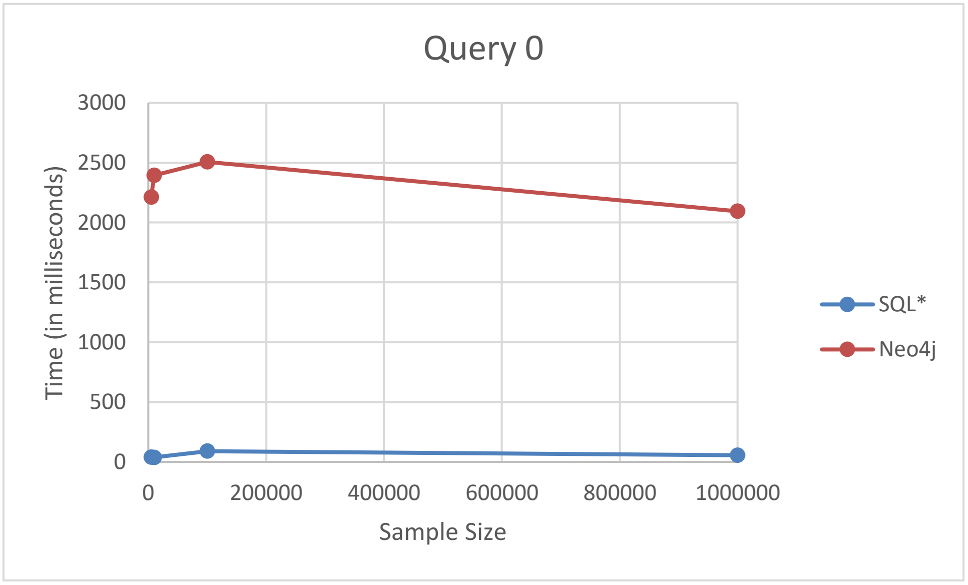


Figure 14. Results for Q0 (*: SQL results scaled x100)

The first query is to establish a baseline for each model on how long it takes to find data given a unique index. In this use case, the relational model has the upper hand where the time per query stays around a fraction of millisecond. While the Neo4j times are upwards of 2000ms per query. From here we can use these times to gauge how more complicated queries compare to this base query.

Q1 - Find all users with matching interests to user with specific user_id

```
SQL - SELECT Y.name FROM Users AS X, Users as Y,
User_to_Interest as UI  WHERE X.user_id = 3 AND X.user_id =
UI.user_id AND Y.user_id IN (SELECT SUI.user_id FROM
User_to_Interest as SUI WHERE SUI.interest_id =
UI.interest_id AND NOT SUI.user_id = X.user_id);
CYPHER - MATCH (x:User)-[:hasInterest]->(i:Interest)<-
[:hasInterest]-(y:User) WHERE x.user_id = 3 RETURN y;
```

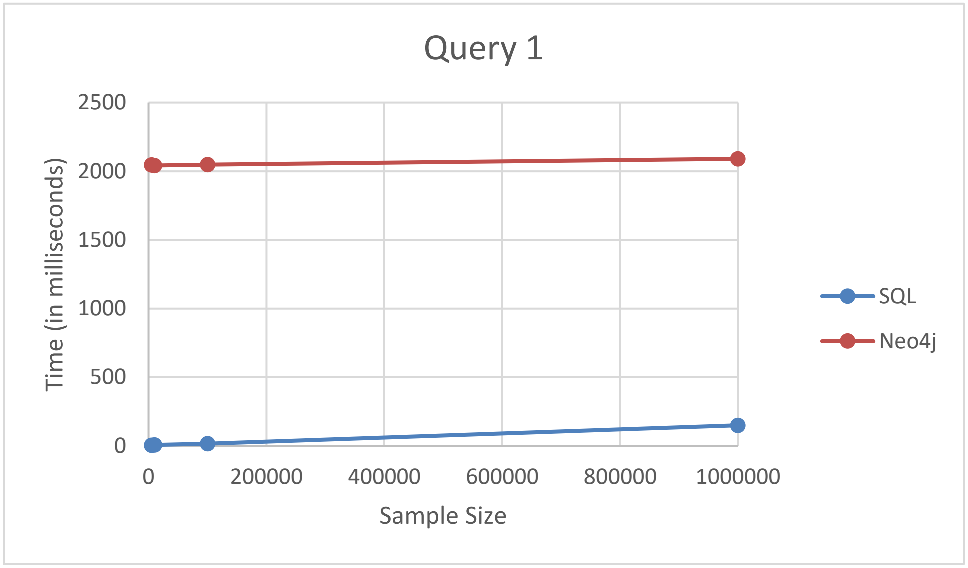


Figure 15. Results for Q1

For this query, we begin to match different entities and their relationships. This is a basic graph traversal question of finding user nodes that are adjacent (have a relationship) to the same interest. Neo4j stays around 2000ms per query, executing this query slightly faster than the first query. While SQL appears faster here, the time per query is upwards of 10x longer than Query 0, with an exponential growth rate as the sample size increases. This growth could be attributed to having a nested subquery.

Q2 - Find all friends of a User (One-Way Relationship)

```
SQL - SELECT U.user_name FROM Users as U, isFriendsWith as
IFW WHERE IFW.user_id = {} AND IFW.friend_id = U.user_id;
CYPHER - MATCH (x:User), (y:User) WHERE x.user_id = {} AND
(x)-[:isFriendsWith]->(y) RETURN y;
```

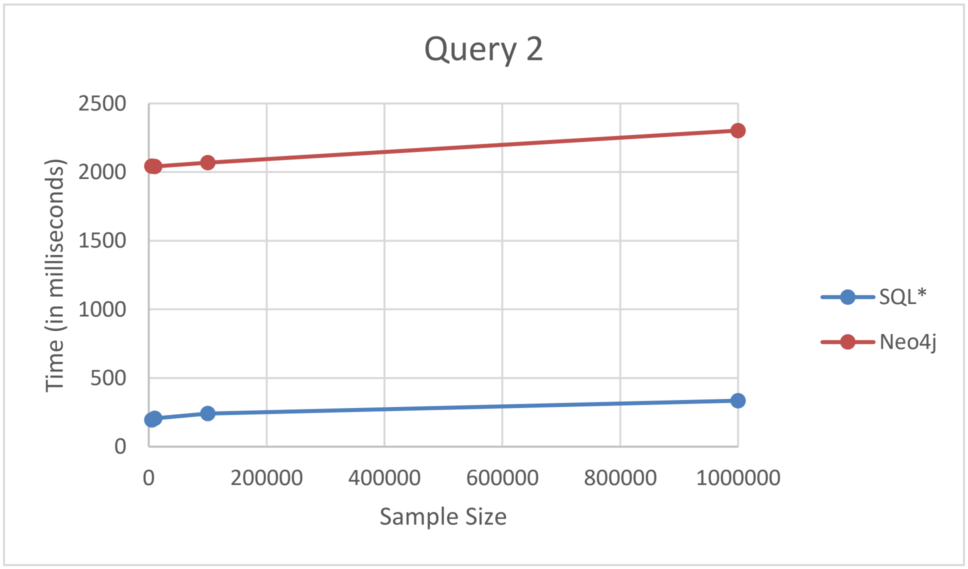


Figure 16. Results for Q2 (*: SQL results scaled x100)

This query is a setup for the next three queries. It is a simple query to find all adjacent user nodes to a user with the relationship "is Friends With". This is a one-way relationship, i.e. a user X can be friends with a user Y while Y may not be friends with X. Again, Neo4j starts at around the same base line of 2000ms, while SQL starts 2-3x the baseline from Query 0. Both show a linear growth with sample size, but the rate is slower for SQL.

Q3 - Find all the Groups that a User's friends are in

```
SQL - SELECT DISTINCT G.group_name FROM Users as U, Groups
as G, isFriendsWith as IFW, isMember as IM WHERE U.user_id
= {} AND G.group_id = IM.group_id AND U.user_id =
IFW.user_id AND IFW.friend_id = IM.user_id;
CYPHER - MATCH (x:User)-[:isFriendsWith]->(y:User)-
[:isMember]->(g:Group) WHERE x.user_id = 250 RETURN g;
```

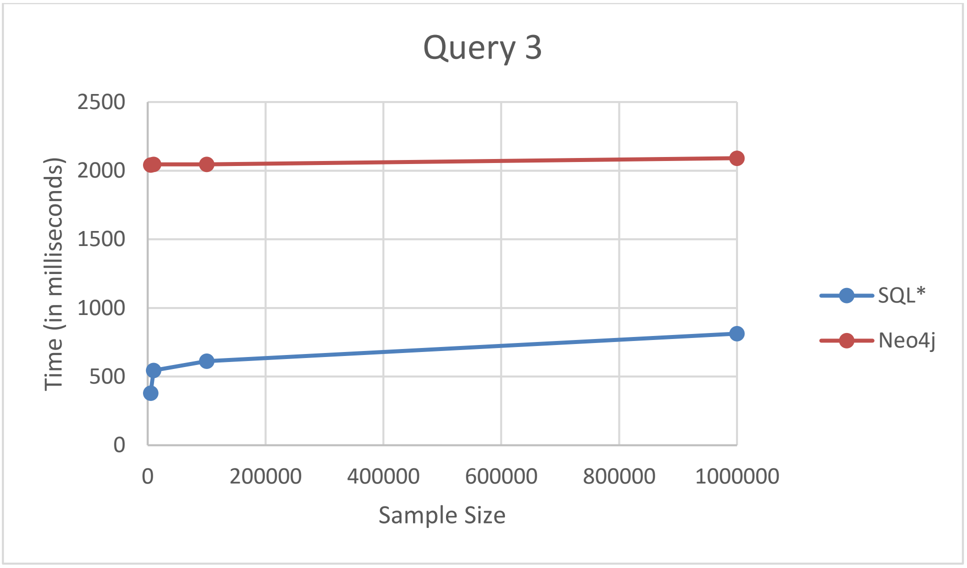


Figure 17. Results for Q3 (*: SQL results scaled x100)

A step up from Q3 appending an extra layer to the path. Neo4j keeps a semi-constant time, starting at the baseline established from Query 0. The Neo4j results are very similar to Query 1 which asks a very similar question. This is likely because the query asks a simple graph traversal question, which Neo4j was designed to answer. The SQL time difference from Query 1 can be attributed to the use of the nested subquery, which appears to be faster with a sample size but does not scale as well. The growth in time is logarithmic for the SQL results although still much faster than Neo4j (Keep in mind the SQL result is scaled up).

Q4 - Find All Users Y attending an Event E hosted by any Group G that a User X is a member of

```
SQL - SELECT U.user_name FROM Users as U, Events as E,
isMember as IM, isAttending as IA WHERE IM.user_id = 245
AND E.group_id = IM.group_id AND IA.event_id = E.event_id
AND IA.user_id = U.user_id;
CYPHER - MATCH (u:User)-[r:isMember]->(g:Group)-
[c:creates]->(e:Event)<-[:isAttending]-(y:User) WHERE
u.user_id = 245 RETURN y;
```

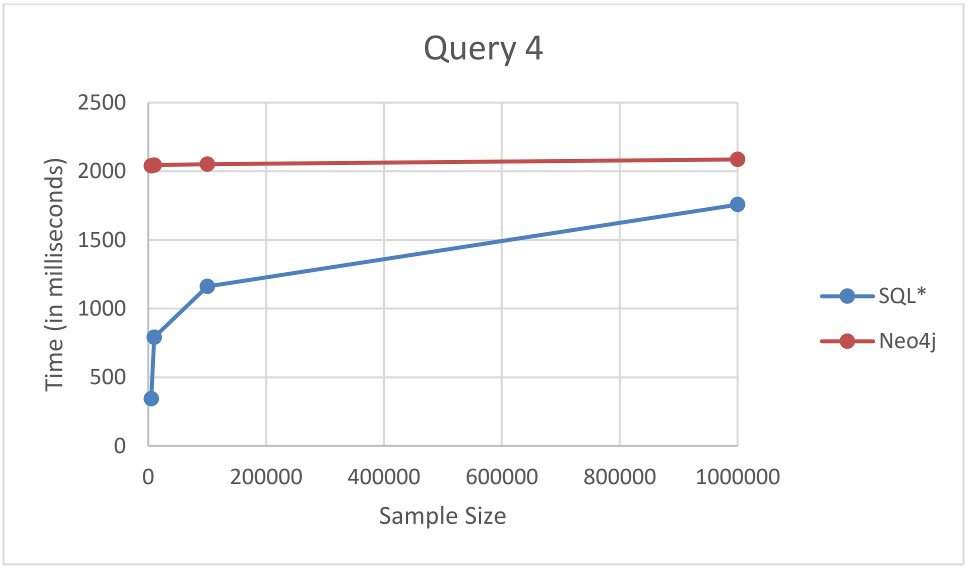


Figure 18. Results for Q4 (*: SQL results scaled x100)

This query asks another graph traversal question, this time to find paths with a degree of 4. Again, we can see that Neo4j excels at answering these types of questions, with the time staying semi-constant starting from the baseline as in Query 0. However, for SQL the growth appears to be logarithmic, this is where we see the relational model start to struggle with answering these types of questions. This growth rate, like in Query 3, can be attributed to the query joining 4 relations from our relational database. But unlike Query 3, this query can return duplicates and is querying through proportionally larger relations. The SQL time is still much faster for these sample sizes (Keep in mind the SQL result is scaled up).

Q5 - Find the interests of any user that is attending any events hosted by any groups that share any interest(s) with a user X

```
SQL - SELECT DISTINCT I.interest_name FROM User_to_Interest
as UI, Group_to_Interest as GI, Events as E, isAttending as
IA,   User_to_Interest   as   UI2,   Interests   as   I   WHERE
UI.user_id  =  2  AND  UI.interest_id  =  GI.interest_id  AND
E.group_id = GI.group_id AND IA.event_id = E.event_id AND
IA.user_id    =    UI2.user_id    AND    UI2.interest_id    =
I.interest_id;
CYPHER   -   MATCH   (x:User)-[:hasInterest]->(i:Interest)<-
[:hasInterest]-(g:Group)-[:creates]->(e:Event)<-
[:isAttending]-(y:User)-[:hasInterest]->(j:Interest)   WHERE
x.user_id = 2 RETURN j;
```

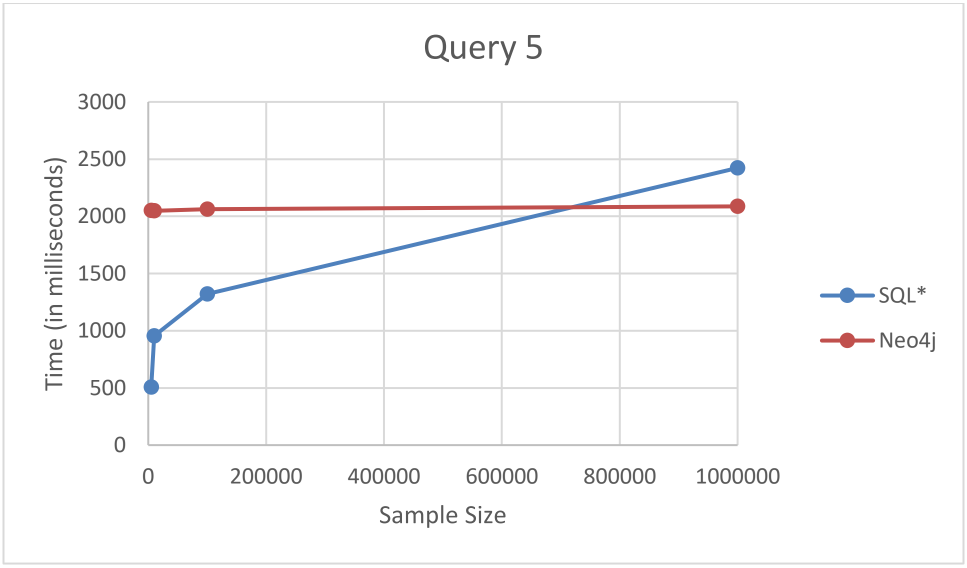


Figure 19. Results for Q5 (*: SQL results scaled x100)

For the final query we ask another graph traversal question, this time finding a path with a degree of 5. Neo4j stays consistent as with previous queries, starting at the same baseline established by Query 0 (approximately 2000ms) and stays semi-constant with the growth in sample size. Following the trends from the past two queries, as the query became more complicated (Joining more relations) the time per query slowed down even further. Though the SQL time is still faster than Neo4j (keep in mind the SQL result is scaled up), the growth is logarithmic.

Notes

- The slow times for Neo4j can be attributed to the python neo4j-driver and iterating over the result set. Measuring the query execution time sans iterating over the results returned much faster results.
- Neo4j struggles with cold starts, the first query executed from new session takes substantially longer time to execute. Neo4j recommends warming up the cache by iterating over the whole graph at the start of the session. This was not done as it did not seem like a practical solution.
- Neo4j excels at executing multiple queries to find results adjacent to the same pointer node. Similar to the last point, this could be attributed to the cache being "warmed up".
- Iterating over the result set from SQL queries barely increases the time from the query execution time alone.

6. Conclusions and Future Work

Based on these results, there is no question that using the relational model is far more efficient for small to mid-sized applications. Even when tested with questions designed specifically to fit its purpose (graph traversal questions), Neo4j still could not keep up with the relational model on such a small scale. An issue which could be attributed to the iterating over the results during testing or the cache not being warmed up. However, what makes it take so much time to execute queries on such a small dataset could also be what allows it to execute them faster on larger datasets. As we found the time per query grows at a very slow rate as the sample size grows, compared to the SQL which seems to follow a logarithmic curve. Given a larger sample size of around 100 million (1e6), I predict the SQL time per query will be slower than the Neo4j time.

In the future, larger sample sizes can be used to validate or disprove the trends shown in these test results. The testing process can be better refined and standardized to support other databases. YCSB support for neo4j can be made, allowing for a more uniform performance benchmark against other NoSQL Databases. The range of tests can be expanded to include more performance metrics other than time, such as memory usage. More queries can be added to encompass a wider variety of use case questions.

Bibliography

[*] Ian Robinson, Jim Webber & Emil Eifrem (2015) *Graph Databases: 2nd Edition*, O'Reilly

[1] NEO4J *What is a Graph Database?* (Online), https://neo4j.com/developer/graph-database/. Accessed 2020.

[2] Levin, Oscar *Discrete Mathematics: An Open Introduction* (Online), http://discrete.openmathbooks.org/dmoi2/ch_graphtheory.html. Accessed 2020.

[3] Frisendal, Thomas *Property Graphs Explained* (Online), http://graphdatamodeling.com/Graph%20Data%20Modeling/GraphDataModeling/page/PropertyGraphs.html. Accessed 2020.