Evolution of Databases in Big Data

Jubair Hassan
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
hassa357@morris.umn.edu

ABSTRACT

The evolution of data and its increasing interconnections in today's world has every big company that is reliant on big data looking for solutions that will sustain them for a good period of time in the dynamic world of technology. It would be bold to claim that no such solution exists since companies are clearly learning how to adapt to the fast changing technology around them. This paper compares relational databases and graph databases. The results suggest that relational databases, as useful as they may be for simple data, are not a good choice for handling big data and provides an alternative to move on from it.

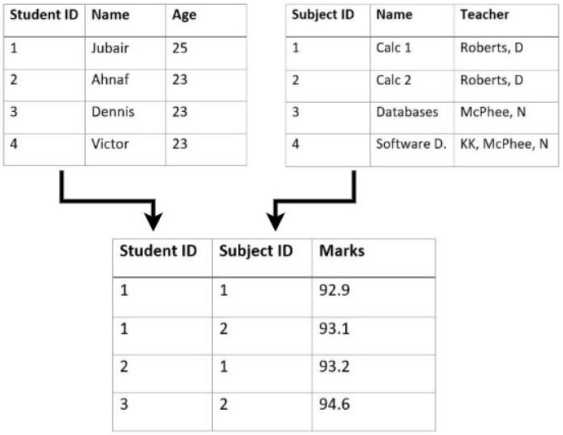
1. INTRODUCTION

Data is growing at an exponential rate right now. Internet Data Center (IDC) predicts that the collective worldwide data - both from cloud and from data centers, will grow to approximately 175 zettabytes from the current 33 zettabytes. That is a compounded annual growth rate of sixty-one percent [3]. This increase comes from a shift to big data.

Big data refers to the large, diverse sets of information that grow at ever-increasing rates. It encompasses the volume of information, the velocity or speed at which it is created and collected, and the variety or scope of the data points being covered. Big data often comes from multiple sources and arrives in multiple formats. Traditionally, there are three "V"s that characterize big data: the volume (amount) of data, the velocity (speed) at which it is collected, and the variety of the information. As we keep going forward, big data is becoming more and more unstructured, which is any data that does not have a particular structure. It could be anything from texts to music, images, videos etc.

In order to store it efficiently, the databases that are used currently need to be evaluated. Relational databases have been the staple for data storage for the longest time. This paper intends to look into why that is a challenge for handling big data. To follow up, multiple database systems will be evaluated using studies that were done earlier. It will then conclude by showing how graph databases are better at handling big data.

UMM CSci Senior Seminar Conference, November 2019 Morris, MN.



Student ID	Name	Age
1	Jubair	25
2	Ahnaf	23
3	Dennis	23
4	Victor	23

Subject ID	Name	Teacher
1	Calc 1	Roberts, D
2	Calc 2	Roberts, D
3	Databases	McPhee, N
4	Software D.	KK, McPhee, N

Student ID	Subject ID	Marks
1	1	92.9
1	2	93.1
2	1	93.2
3	2	94.6

Figure 1: An Example of a Relational Database

2. BACKGROUND

2.1 Relational Databases

A relational database is a collection of formally defined tables that can be used to reassemble or access data in various ways without needing to reorganize the tables in the database. Relational databases focus more on a straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called a *key*. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points [8]. The tables can have *primary keys*. Primary keys are a unique attribute in a table that identifies a certain row. In Figure 1, in the top left table, *Name* and *Age* cannot be a primary key since there can be multiple students with the same name and age. Therefore, *Student ID* is the primary key of the table. Multiple tables are linked by *foreign keys*. A foreign key references the primary key of another table and thereby it is used as a cross-reference between tables. This is shown in Figure 1, where in the table at the bottom, *Student ID* and *Subject ID* are the foreign keys.

Another key terminology of relational databases is *join* statements. These are primarily used to combine data from

```
SELECT
Orders.OrderID,
Customers.CustomerName,
Orders.OrderDate
FROM Orders
INNER JOIN Customers ON
Orders.CustomerID=Customers.CustomerID;
```

Figure 2: A Code Snippet of "Join" Statements [4]

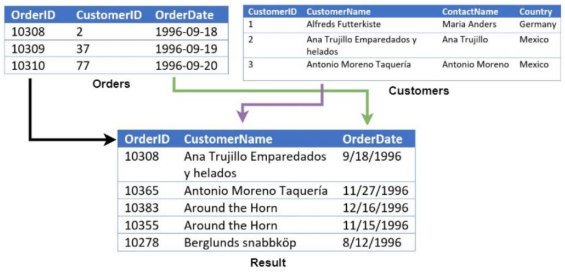


Figure 3: The Result of the Code Snippet [4]

two tables. Figure 2 is an example of what join statements look like. What it essentially is asking is to print out all the instances of *OrderID* and *OrderDate* from the *Orders* table along with the *CustomerName* from the *Customers* table where the *CustomerID* in each table matches. Figure 3 is the result of the join statement from Figure 2.

There are two types of *scaling* options in relational databases: *vertical* and *horizontal*. When a database is scaled horizontally, more machines are added and data is distributed among those machines. The pros of this process is that it is cheap, and since data is distributed, the load is less on a single machine and the performance is better. However, this data distribution is what makes joins harder, because it might involve communication across servers. Vertical scaling is quite basic: the machine is upgraded to a more powerful one. The advantage of this option is that it is a simple process that yields better performance. One disadvantage is that it is more difficult to perform multiple queries. There is also the cost efficiency factor to consider: since upgrading to higher performance machines always expensive since most relational databases rely on proprietary hardware [1].

Elasticity is a key concept in database technology which essentially means the ability to "undistribute" data after the distributions and the allocation of additional space have already taken place.

3. NOSQL DATABASES

NoSQL - not only SQL - is a blanket term for projects that have been in development in recent years after investigation into storage alternatives for relational databases. The key difference between a NoSQL database and a relational one is that a NoSQL database does not require a predefined schema. If a student's data, such as name, address, parents, grades, etc., was going to be stored in a relational database, the schema needs to be configured beforehand. NoSQL databases can insert any sort of data dynamically without any pre-defined schema.

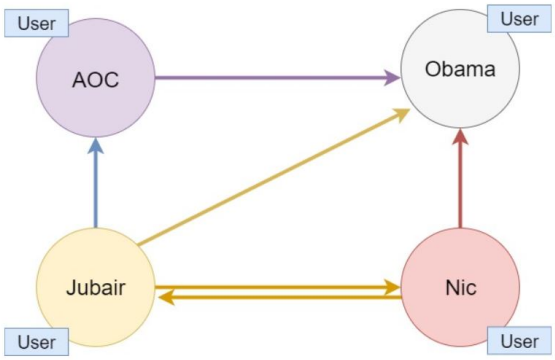


Figure 4: A Standard Graph

There are different types of NoSQL databases. Some examples are: Cassandra, BigTable, CouchDB and Project Voldemort. All of these have one thing in common, they ardently reject the relational and object-relational models [12]. When trying to decide whether the NoSQL model is a better fit for the current set of data at hand, one needs to look at the following criteria: tables with a large number of columns, each of them being used by only a few rows; attribute tables; a large volume of many-to-many relationships; tree-like characteristics; and the constant need of schema updates. If the set of data meets several of these criteria, a NoSQL model should be explored [10].

3.1 Graph Databases

Graphs are used to represent numerous types of data in today's world. Common examples are data from social networks and information networks. The type of graphs that are most commonly used are directed graphs that are enhanced by their properties. To put it simply, these graphs have a number of nodes, which are combined together with binary relations known as edges [9]. Figure 4 is a sample twitter graph map. Here, the Nodes are the Users: AOC, Obama, Jubair and Nic. The vertices (relationships) are the arrows which denotes who follows whom.

Any system that uses graphs to represent and store data is called a graph database. As mentioned earlier, there are multiple types of NoSQL databases. Graph databases are a type of NoSQL database because of some particular properties and features that they have.

This paper focuses on a particular graph database - Neo4j; and it is essential that two key properties of this database are explored.

3.1.1 Native Graph Storage

When discussing native graph storage, it refers to the underlying architecture of a database to store graph data. Any graph database that has native graph storage is designed explicitly to store and manage graph data. In Neo4j, graph data is stored in files that each contains the data for a particular part of the graph, such as nodes, relations, labels, and properties. The graphs are now traversed at a higher efficiency rate given that data storage is divided [5].

Non-native graph storage, unlike the native storage, does not use a system explicitly designed to store and manage graph data. It uses some kind of general purpose data storage systems like relational databases to store graph data. Since the nodes and relations are not stored efficiently and there is a disconnect between them, it would imply that connected data have to be retrieved and reassembled for every new query. For any sort of ceaseless operation that is going to result in a big number of performance issues [5].

3.1.2 Native Graph Processing

A graph database that uses *index-free adjacency* has native graph processing. What that means is that each node acts as an micro-index for all its nearby nodes, referring to the ones adjacent to it. This is very efficient, as the query time does not increase as the total size of the data grows rather, it is proportional to the amount of the graph that is searched [5].

An important aspect of graph database is that it always prioritizes relationships between two nodes. Therefore, if the reliance on indexes is minimized, traversing relationships are much more efficient. For example, in Figure 4, it is a very simple task to find out who is connected to whom with index-free adjacency, but this would not be the case with non-native graph processing where multiple indexes would be used to connect the nodes, making it more expensive.

4. DATABASE SYSTEMS IN BIG DATA

4.1 Relational Databases in Big Data

Relational databases are great for simple data types; they represent data well on a small scale. When it comes to big data, however, it is a completely different story. The number of interconnections in big data is constantly growing. As a result, when we try to query data in a relational database, it takes multiple join operations and a longer time than usual. In today's world, companies handling big data would move away from a system without hesitation if it keeps taking up more than a reasonable amount of time to retrieve data. It would not be a bad idea to vertically scale the database. But when we are scaling vertically, it is often limited to the capacity of a single machine. Scaling beyond that capacity often involves downtime and comes with an upper limit. Then, there is the other option: to scale it horizontally. The problem with this is relational databases are not designed for scaling of any type [7].

Allen [7] talks about how achieving scalability and elasticity is a big challenge for relational databases. Relational databases were designed in a period when data could be kept small, neat, and orderly. That is just not the case anymore. Yes, to stay in business, all database vendors say they scale big; but, if the functions are properly examined, the fundamental flaws start to become more evident. Relational databases are typically designed to run on a single server to preserve table mapping integrity and to prevent distributed computing problems. This design requires users to buy bigger, more complex, and more costly proprietary hardware with more processing power, memory, and space if a system needs to be scaled. Improvements are also a concern because the company must endure a lengthy acquisition process and because the system must often be taken offline to make the change. All of this is taking place while the number of users continues to grow and the potential for risk in the undersupplied resources continues to grow [7].

To address these issues, relational database suppliers have made a number of improvements. Nowadays, the evolution of relational databases permits them to use more complex architectures using a "master-slave" model where the "slaves" are additional servers which can handle parallel processing, replicated information, or *sharded* data to reduce workloads on the master server. Sharded data is data that has been divided and spread out among multiple servers - something that happens when you scale a database horizontally. Numerous enhancements to relational databases, such as shared storage, in-memory processing, improved replica use, distributed caching, and other recent and 'innovative' architectures certainly have had a positive impact on relational database scalability. Nonetheless, if they were examined properly, it would not be difficult to find a single point-of-failure. For instance, Oracle RAC, a "clustered" relational database with a cluster-aware file system, still has a shared disk subsystem below. The high costs of these systems is quite restrictive oftentimes, as the set-up of a single data warehouse can easily go north of a million dollars. The upgrade of relational databases also comes with major concessions. For example, to maintain performance, the horizontal scaling of data across a relational database is based on predefined queries. Additionally, relational databases are not designed to scale back down. It is almost impossible to "undistribute" this information after the distribution and the allocation of additional space [7].

4.2 Graph Databases in Big Data

When relational databases are mentioned, it should be visualized as numerous structured tables. But when graph databases are discussed, the base starts from something like in Figure 4. Then it evolves, as the interconnections grow between the data and the volume of data keeps increasing, and it ends up looking like Figure 6.

4.2.1 A MySQL-Neo4j Comparative Analysis

Vicknair et al [12] did a comparative analysis where they ran queries in two different type of databases namely MySQL and Neo4j. How this study worked was that it had twelve databases of each type set up, and each database stored a DAG that consisted of some number of nodes and edges. A directed acyclic graph (DAG), as shown in Figure 8, is what provenance is stored as usually. Provenance refers to lineage. If the provenance of a data item was to be traversed,

Figure 5: A Graph Database [2]

Database	#Nodes	Data Type	MySQL Size	Neo4j Size
1000int	1000	Int	0.232M	0.428M
5000int	5000	Int	0.828M	1.7M
10000int	10000	Int	1.6M	3.2M
100000int	100000	Int	15M	31M
1000char8k	1000	8K Char	18M	33M
5000char8k	5000	8K Char	87M	146M
10000char8k	10000	8K Char	173M	292M
100000char8k	100000	8K Char	1700M	2900M
1000char32k	1000	32K Char	70M	85M
5000char32k	5000	32K Char	504M	406M
10000char32k	10000	32K Char	778M	810M
100000char32k	100000	32K Char	6200M	7900M

Figure 6: Generated Database with Sizes [12]

it would mean to look at the description of the data as well as how was it made.

The databases had just enough information about the structure of the DAG as well as any data that could be connected to the node objects, otherwise known as payloads of the DAG. Since the payloads comprised of random integers, 8K strings and 32K strings, the databases (Figure 7) that were generated (twelve MySQL and twelve Neo4j) were from all random graphs.

Queries were developed to simulate some of the sort of queries used in the provenance systems. There were two types of queries: *structural* and *data* queries. Structural queries refer to the DAG structure only and the data query refers to the payload data.

The queries were run ten times each on each database, collecting the runtime in milliseconds (ms). The longest and shortest times were dropped to ensure that the timings were not affected by system activity or caching. The rest of the eight results were averaged.

Neo4j performed significantly better in the three structural queries that were performed as can be seen in Figure 9. The three structural queries were:

- S0: To return all orphan nodes.
- S4: Determine the number of nodes that can be reached by traversing the graph to a depth of 4.
- S128: Determine the number of nodes that can be reached by traversing the graph to a depth of 128.

The data queries showed the superiority of the relational databases in handling such queries in integer payload databases

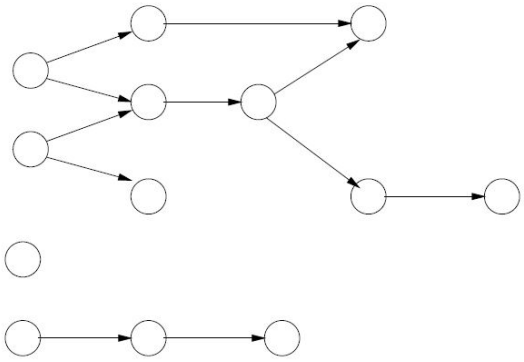


Figure 7: A Sample DAG [12]

as Neo4j treated all data as text when querying, conversions had to made before any sort of comparisons.

Four databases of 8K and 32K were used to do full-text searches with character payloads. The results were quite intriguing. The researchers used random data with just letter at the beginning and then used data with spaces that matched data from the real world. The first time around MySQL outperformed Neo4j by a good margin. When the data reflecting real world scenarios were queried, even though, at a smaller scale, both performed almost equally, Neo4j performed dramatically better as the size of the data continued to grow.

4.2.2 *Migration from a Relational Database to a Graph Database*

Unal et al [11] looked into the relative usefulness of migrating data from MySQL, a relational database, to Neo4j, a graph database.

The reason they chose to look at MySQL was because it is one of the most popular open-source relational database, meaning it had reliable and scalable (to an extent) relational database applications. The MySQL Community Server was used as a relational database management system for storing the application data. It should not come as a surprise that they chose Neo4j, since it is the most popular graph database management system, as well as one of the most popular NoSQL database systems.

The data model that was chosen to migrate from a relational database to a graph database was a legal document system. The system had eighteen data entity types with three levels of hierarchy for each type. The figure shows the structure of the legal document system data model. There are cross-relationships between different documents and data types besides the more simple parent-child relationships.

This particular data model was designed as a relational model. Each table had a large volume of data, and the self-referencing tables, along with the tree-hierarchy meant that traversing through the data caused a decreased performance. Self-referencing tables are tables whose primary key is also

Database	MySQL S0	Neo4j S0	MySQL S4	Neo4j S4	MySQL S128	Neo4j S128
1000int	1.5	9.6	38.9	2.8	80.4	15.5
5000int	7.4	10.6	14.3	1.4	97.3	30.5
10000int	14.8	23.5	10.5	0.5	75.5	12.5
100000int	187.1	161.8	6.8	2.4	69.8	18.0
1000char8K	1.1	1.1	1.1	0.1	21.4	1.3
5000 char8K	7.6	7.5	1.0	0.1	34.8	1.9
10000 char8K	14.9	14.6	1.1	0.6	37.4	4.3
100000 char8K	187.1	146.8	1.1	6.5	40.9	13.5
1000char32K	1.3	1.0	1.0	0.1	12.5	0.5
5000 char32K	7.6	7.5	2.1	0.5	29.0	1.6
10000 char32K	15.1	15.5	1.1	0.8	38.1	2.5
100000 char32K	183.4	170.0	6.8	4.4	39.8	8.1

Figure 8: Query Results [12]

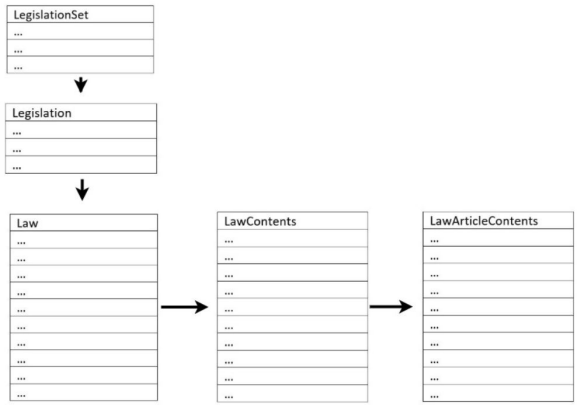


Figure 9: The Relational Model [11]

its foreign key, and that creating a loop which can be quite difficult to traverse.

The same model was taken and converted to a graph model. There were some key transformations:

- Foreign Key references became edges between nodes
- Each table became a node label
- Each data item in the table became a node instance

Figure 10 shows the relational model and Figure 11 shows the graph model that results from the transformation process. Since this was conducted in Turkey, the words were foreign and have been translated for clarity.

The data migration was a two step process: the first step was to extract the metadata and table data from MySQL using Schema Crawler (a tool to comprehend and output the schema of a database) and Java SQL Library, and the second step was to import it to Neo4j, using its own API. Figure 12 shows what changed through the transformation process from the relational side to the graph side.

To compare the performance of the two databases, all laws related to the "Tax Legislation Set" (translated) were queried. It took two join statements in MySQL to perform the query. As discussed earlier, executing join statements takes more time, therefore data was retrieved ten times

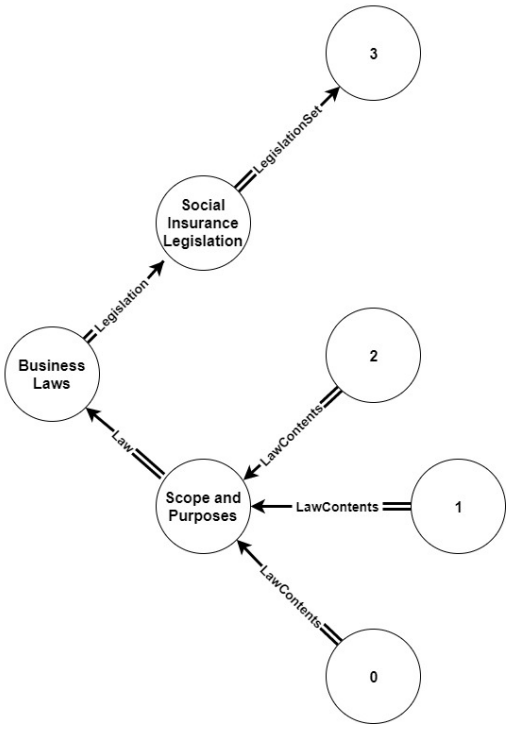


Figure 10: The Graph Model after the Transformation [11]

Relational Model	Graph Model
Entity Table	Label on the Nodes
Row in the Entity Table	Node
Columns in the Entity Table	Node Properties
Join Tables	Relationships
Columns in the Join Tables	Relationship Properties

Figure 11: Relational Side vs Graph Side [11]

faster on the graph database.

In general, in the graph model, data was accessed:

- SIX TIMES faster when there were a thousand records.
- THIRTY TIMES faster when there were ten thousand records.

The researchers concluded that it was ideal to evaluate the type of data before choosing a database using some basic criteria. Graph databases are highly recommended in the case of data having numerous interconnections and a system which would go through constant updates as relational databases must have a predefined schema and it would be inconvenient to update the schema every time.

4.3 The Hybrid Approach

An interesting approach is proposed by Vyawahare et al [13] where the idea is to use both a relational and graph database. This is so that each database can fill in any of the weaknesses that the other might have. The process would be something like this: first data would be loaded, then it will go through a classifier to determine which kind of database it would be stored in the relational database if structured and graph database if unstructured. Since the query language being used is that for graph databases, there is an additional layer for the relational side which is a query translator that helps load the data into the relational database if it is structured. After that, the hybrid interface is updated. It enables the user to then View, Delete, Update and Exit.

This is the proposal and even though it is interesting, it is still yet to be tested.

5. CONCLUSION

Big data is becoming more common for big companies. Customers are becoming more used to getting information filtered and presented to them as they wish in a matter or milliseconds. Companies with the foresight and will to expand understand one thing: pulling the necessary information out of big data is the next step. Relational databases have long been the staple for anyone and everyone trying to store data. It still is, for structured data. But with data becoming more and more unstructured for big companies, it is not logical to adhere to the same statement. Eighty percent of the data in the world is projected to be unstructured by 2025 [6].

Graph databases have proven to be quite the replacement for that matter. After comparing the performances of both relational and graph databases, most results indicate that graph databases seem to be doing better at handling big data than relational databases. Therefore, any company hoping to expand in the future are highly recommended to incorporate graph databases from the start.

6. ACKNOWLEDGEMENTS

The author would like to thank Prof. Hussam Ghunaim and KK Lamberty for the advice and feedback on the topic. The author would also like to thank Clara Martinez for her feedback and advice on the writing and language.

7. REFERENCES

[1] Database scaling. https://hackernoon.com/database-scaling-horizontal-and-vertical-scaling-85edd2fd9944. Accessed = 2019-11-12.

[2] Graph database picture. https://www.cbronline.com/enterprise-it/software/graph-technology-data-standby-every-fortune-500-company/. Accessed = 2019-11-12.

[3] Idc data prediction. https://www.networkworld.com/article/3325397/idc-expect-175-zettabytes-of-data-worldwide-by-2025.html. Accessed = 2019-11-18.

[4] Join statements. https://www.w3schools.com/sql/sql$_j oin.asp.Accessed = 2019 - 11 - 12$.

[5] Neo4j properties. https://neo4j.com/blog/native-vs-non-native-graph-technology/. Accessed = 2019-10-22.

[6] Prediction on unstructured data. https://solutionsreview.com/data-management/80-percent-of-your-data-will-be-unstructured-in-five-years/. Accessed = 2019-11-22.

[7] Relational databases are not designed for scale. https://www.marklogic.com/blog/relational-databases-scale/. Accessed: 2019-10-27.

[8] What is a relational database? http://https://www.oracle.com/database/what-is-a-relational-database/. Accessed: 2019-10-27.

[9] J. Pokorný. Conceptual and database modelling of graph databases. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, IDEAS '16, pages 370–377, New York, NY, USA, 2016. ACM.

[10] S. Tamane. Non-relational databases in big data. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*, ICTCS '16, pages 134:1–134:4, New York, NY, USA, 2016. ACM.

[11] Y. Unal and H. Oguztuzun. Migration of data from relational database to graph database. In *Proceedings of the 8th International Conference on Information Systems and Technologies*, ICIST '18, pages 6:1–6:5, New York, NY, USA, 2018. ACM.

[12] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: A data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference*, ACM SE '10, pages 42:1–42:6, New York, NY, USA, 2010. ACM.

[13] H. R. Vyawahare, P. P. Karde, and V. M. Thakare. A hybrid database approach using graph and relational database. In *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, pages 1–4, Aug 2018.