Global Scientific JOURNALS

GSJ: Volume 10, Issue 3, March 2022, Online: ISSN 2320-9186

www.globalscientificjournal.com

# Improved GraphQl Model Query Processing of Distributed Databases

Obasi E.C.M
Department of Computer Science and Informatics
Federal University Otuoke
Bayelsa State, Nigeria
anchinos@yahoo.co.uk

Eke B
Department of Computer Science
University of Port Harcourt
Choba, Rivers State Nigeria
bathoyol@gmail.com

Egbono F
Department of Computer Science
University of Port Harcourt
Choba, Rivers State Nigeria
fubaraegbono@gmail.com

**Abstract**: There are numerous means to access multiple databases in a distributed environment. The relational query language SQL usually acts as a language to access data. The performance of a relational databases drops as the datasets increase. Problems arise in relational databases when it is required to create multiple relationships between data saved in computer storage. The large datasets involves multiples tables and multiple joins and querying such database becomes a complex operation. Graph databases on the other hand are better in managing datasets that contain many links. Application programming interfaces (API's) simplifies the job of the web developers when developing applications for they enable interaction of softwares among themselves. This research work is based on designing a GraphQL Model to query a graph database. Querying graph database with GraphQl improves query result. GraphQL presents a profound transformation in how API providers enable access to their data, and can bring a lot of useful advantage to organizations. **Keywords**: GraphQl, REST, Relational Database, Graph Database, Query Processing, Application Programme Interface

## 1. INTRODUCTION

The emergence of computer network technology and database system technology has given birth to the advancement of distributed database. Distributed databases are defined as databases located at different machines at the same or different locations that look like one centralized database to the end user. A set of machines share an entire load instead of one centralized database. Distributed database (DDB) is a collection of multiple, logically interrelated databases distributed over a computer network (Idowu and Maitanmi, 2014). Retrieval of data from different sites in a DDB is known as distributed query processing. . Srinath (2016) looked at Query Processing Issues in Data Warehouses. According to him; "major query processing issues ranges from poor query plans, inadequate data extraction processes and inefficient data transformation concept". Query processing is an important concern in the field of distributed databases. Notwithstanding the numerous benefits of relational databases, however, they don't produce good results with the continuous increase in the volume connected data. To handle a growing volume of connected data, Neo4j database can be used. Neo4j is a non-relational graph database that's optimized for managing relationships. Data structures today are more complex, they are better thought of as graphs (multi-dimensional) instead of tables (two-dimensional).Querying those graphs requires a standard query language, and dynamic result structures which reflect what was actually queried for. The Neo4j database helps in building high performance and scalable application that uses large volumes of connected data.

## 2. BACKGROUND

### 2.1. Related Works

Naresh (2017) proposed a Graph-based User Interface architecture which was integrated into a Bluemix system that was already on use and the major execution advancement of the real-time dashboards was explored. This boosted the performance of the existing web applications by returning faster only those data that was explicitly defined by the client. The client specified the server about the precise data to return. This prevented fetching more than sufficient data that was needed or less.

Njoku Donatus el al (2016) proposed a solution for query execution and responses time and optimization in distributed database systems done by means of modified and enhanced Interactive Dichotomizer 3 (ID3) algorithm. A modified ID3 algorithm was developed to generate the decision tree, which created simple and efficient tree with the smallest depth. The learning agents ID3 must search through the hypothesis space and locates the best hypothesis when given the test sets. Result showed the ID3 query response time was minimal compared to that of the search engine.

GRAPHiQL (Jindal and Madden, 2014) has been introduced as another SQL-like general purpose graph processing language. GRAPHiQL provides its user with the ability to reason about graphs in terms of the intuitive abstraction of vertices and edges. It also provided optimized graph querying constructs such as recursion, looping, neighborhood access. The GRAPHiQL execution engine compiled the user query into SQL query that was executed by a standard relational engine and relied on query optimization techniques to tune the performance of these queries.

Jing et al (2017) looked at GraphCache: A Caching System for Graph Queries. The authors put forth GraphCache (GC), the first full-fledged caching system for general sub-graph/super-graph queries. They contributed the overall system architecture and implementation of GC. They also studied a number of novel graph cache replacement policies and show that different policies win over different graph datasets and/or queries; In addition, the authors contributed a novel hybrid graph replacement policy that was always the best or near-best performer.
M.B.Thuraisingham (2010) proposed Secure Query Processing In Intelligent Database Management Systems. He looked at methods for safe query processing in a different intelligent database system. Specifically, these systems were looked into. (i) relational systems (ii) systems focussed on distributed architecture (iii fuzzy system (iv) object oriented /semantic systems .His future plan was to develop a setup in such a way that he could perform an experiment and assess the techniques he had designed. Initially he would be carrying out an augmented relational database system for secure query processing .He planned to use a commercial relational database system which will be interfaced with a knowledge base and an inference engine. The knowledge base would use rules to represent the constraints and information on the reply that was made. Queries would be requested in logic. The query would be amended by the inference engine. A translator would change the amended query in logic by the relational database system. Evaluations can be done on converted query by the relational database system

Bhavani and Ammiel (2010) proposed Secure Query Processing In Distributed Database Management Systems - Design And Performance Studies. Their work concerned with query processing in a trusted distributed database management systems. .It wss based on two focal points. The main concern in the design of most TDBMS was the characteristics of hiding information from individuals that were not authorized. Another important area was the characteristics of performance. In many computer programs in command and control surrounding, it is necessary for a database system to work in a protected mode and in addition to satisfy real-time or at a minimum near-real-time execution. For the purpose of carrying out execution improvement to a TDDBMS, a baseline study was essential in the first place. Their paper was focused on early implementation of query processing in a TDDBMS and that was the first approach in their study. It provided a plain description of a query processing plan of action and the implementation of the design was discussed. Verifying the security principle and resolving the execution of query processing algorithm was the objective of the implementation. Their paper described the issues involved in secure distributed query processing. Next they described a simple design of a TDDBMS and have discussed the implementation of the design for query processing. The TDDBMS that they implemented consisted of two nodes interconnected via a communication channel. Each node had a local TDBMS and a distributed query processor. The local TDBMS was implemented by augmenting a commercial relational database system with a front-end component. Their objectives in this implementation were (i) to validate the security policy, and (ii) to analyze the performance of secure query processing algorithms. From their simple analysis, they were able to show that poly instantiation as well as transfer of tuples between nodes has a significant impact on performance. The following are the major limitations of the TDDBMS that we have implemented (1) the nodes were not connected over a communication network; (2) the local TDBMS was implemented on top of an untrusted commercial DBMS; (3) the system consisted of only two nodes. In order to develop a more realistic system, the distributed system should consist of more than two nodes whichweare interconnected via a communication network. Furthermore, a secure commercial TDBMS should be used as the local TDBMS-. Their future work will include such an implementation.

Saurabh G. et al (2015) proposed a Survey on Query Processing and Optimization in Relational Database Management System. In their paper, they have made an attempt to present a detailed review of query optimization techniques. The main idea behind this research was to review various techniques to implement query processing and optimization in an effective manner. Though the traditional, single-pass, optimize-then-execute strategy for query execution has served the database community quite well since the 1970. As queries have become more complex and widespread, however, they have started to run into limitations. Query optimization techniques and approaches primarily focus centralized and distributed databases. The paper also highlighted merits of these techniques by critically analyzing them with respect to their utility and efficacy. Schemes for robust optimization, parametric optimization, and inter-query adaptivity alleviate some of these difficulties by reducing sensitivity to errors. A significant virtue of these methods is that they impose little runtime overhead on query execution perhaps even more importantly; they serve as a simple upgrade path for conventional single pass query processor. Selection ordering was a much simpler problem than optimization complex multi-way join queries, and this simplicity not only enabled design of efficient algorithms for solving this problem. .They have discussed the existing techniques and their implementation for the sake of optimizing query. They have identified some of the proposed techniques that led towards achieving the key benefits of an optimized query as compare to an un-optimized query in terms of its throughput and response time.

## 2.2. Graph Databases/Relational Databases

Graph databases store data in vertices and edges or nodes and relationships as shown in figure.1 versus tables, as found in relational databases. Graph databases are the most efficient way of looking for relationships between data items, patterns of relationships or interactions between multiple data items. You can think of a relational database as made up of several tables, rectangular grids of information, each one looking much like a spreadsheet. Graph databases are well-suited for analyzing interconnection, which is why there has been a lot of interest in using graph databases to mine data from social media. Graph databases are also useful for working with data in business disciplines that involve complex relationships and dynamic schema such as supply chain management.

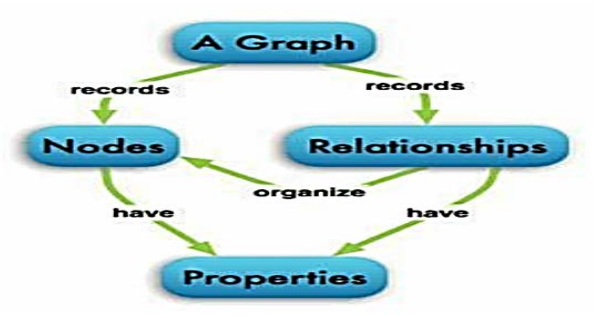


**Figure 1: Basic Structure of a Graph Database (Source: Smita and Patel, 2016)**

With the traditional relational database systems, many series of composite joins are required. It is quite complicated constructing these queries and the high cost of executing these queries constitutes to the short falls. Trying to scale these large queries in order for them to return real-time data is almost not possible in majority of the cases due to low expectations because as the query increases, execution drops due to more parameters are included. Graph databases offer great advantage in overcoming these difficulties as using the relationships to traverse the graph is an optimal solution. Real-time queries can be made at the point an odd transaction occurs giving the financial institution a unique ability to utilize various fraud patterns in real- time which can apply different individuals or groups. Financial institutions can benefit from this in detecting fraudulent behavior before serious crimes take place thereby making them to be alert.

## 2.3. Application Programming Interface (API)

An API is a group of protocols and definitions used to integrate different components of a system. It defines a contract between a “provider” and a “consumer”, and all the parameters involved in the communication between them:

1.The requests to be made

2.How to make them

3.The different responses

4.Data types

It is a cost effective means that allows product communications without having to know how they were implemented. With an API we can reuse existing services from other components of our app, with an abstraction layer over the details of how those services are implemented. REST(Representational State Transfer), an API specification is a group of architecture principles followed to create web services that work using the HTTP protocol. , REST is any interface between systems using transport protocols like HTTP to obtain service and generate operations on it in all possible formats, such as extensible Markup Language (XML) and Java Script Object Notation (JSON).

Graphql, another specification of API is the query protocol of an application layer developed by Facebook. It is developed to give more responsibility for clients to enhance flexibility and efficiency. GraphQl was created out of the needs of todays, especially the needs of mobile clients, Low powered devices transferred data with not ideal networks. Thus, Graph does minimize the amount of data transferred and enables new features to be added without removing the old ones (Buna, 2016).All in all, GraphQL forces queries and removes many design questions. With GraphQL, clients can write their own queries and use the language of their choice instead of forcing the developers make custom endpoints, representations or APIs to solve issues with different clients and their needs(Sturgeon,2017). GraphQl is often confused with being a database technology (Stubailo, 2016). This is a misunderstanding. GraphQL is a query language for APIs and not even for databases. GraphQL is the modern way to build declarative, efficient and performant APIs that developers love to work with. In GraphQL, the client determines precisely the data it wants from the server. With this, clients can make more efficient queries to a server by trimming down the response to meet their needs. This speeds up rendering time by reducing the payload size. GraphQL Server can be integrated with a graph database as shown in figure 2.

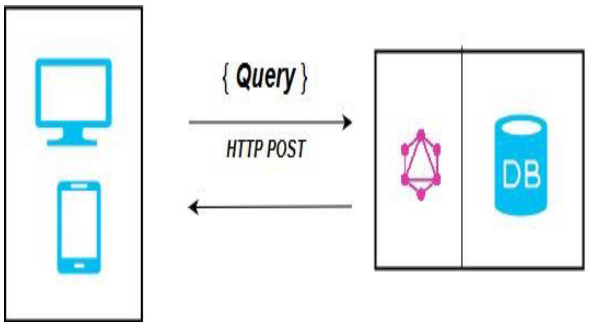


**Figure 2: GraphQL Server with Connected Database (Graphcool, 2017)**

Integrating GraphQl server with graph database makes it easier for the server to pull the data from the database and displays it for the client. GraphQl , a data querying language that is available for everyone is used for developing Application Program Interface for web and mobile computer programs. It is a very big alternative to Representational State Transfer and other web service structure and design. It allows the client side of the app to get the data in any structure. But GraphQl is just a query language. To carry out GraphQL instruction, we need a platform that can provide a lot of assistance for us and example of such platform is Apollo. The Apollo platform is an implementation of GraphQL that can transfer data between the cloud (server) to the user interface of your application. Actually, Apollo develops its environment in a flexible way that allows GraphQL to be used on both clients and server side of the computer program. Apollo platform contains things like the Apollo server, IDE plugins, the Apollo CLI (command line interface). A Javascript GraphQL server is a core part of Apollo platform. It specifies the structure and the set of resolvers that executes every part of the specified structure. The server has capability to extend in such a way that it can link to any request directed to it through the commercial plugins. When a query is sent to a GraphQL server, the query will be parsed into an Abstract Syntax tree which is validated against the schema defined by the GraphQL server. Validation checks for correct query syntax and existence of the fields. If the query passes the validation, the query is executed. The runtime walks through the AST, starting from the root of the tree, invokes resolver, collects up results, and emits JSON

## 3. ARCHITECTURE DESIGN

### 3.1. Research Methodology

The methodology employed for this research work is Object-Oriented Analysis and Design (OOAD) Methodology. Object-oriented programming (OOP) is a programming paradigm that uses “objects” and their interactions to design applications and computer programs. It has been touted as the great advance in software engineering. It promises to reduce development time, reduce the time and resources required to maintain existing applications, increase code reuse, and provide a competitive advantage to organizations that use it.

### 3.2. Analysis of Existing System

Graph query processing is essential for graph analytics, but can be very time-consuming as it entails. Eeda (2017) proposed a GraphQL-based UI Architecture which was integrated into the existing Bluemix system to boost an application’s performance. The key performance improvement of the real-time dashboards was explored. The performance of the existing web applications was boosted by fetching only the client-specific data faster. To avoid fetching more or less of the needed data, the client strictly dictates to the server the kind and amount of data to fetch. The architecture was created by developing a GraphQL layer and placing it above already existing data sources at the backend the system was using. These data sources could be a relational database, NOSQL database, REST endpoint, SOAP endpoint, message bus or a combination of any or all of these data sources. The architecture incorporates GraphQl to brings in all the advantages of GraphQL which includes returning the result of a query through a single endpoint, making fast and simple queries. Making a simple query improves the stability of a query process. When a client issues request to a GraphQL server, the request is splitted into multiple forms on reaching the GraphQL layer as proposed in the architecture. They designed and stored the states of all the components that were to be rendered on the UI using Redux and the different parts of React were binded with its value as shown in figure 3. A caching mechanism and subscription channel were also incorporated in their design.

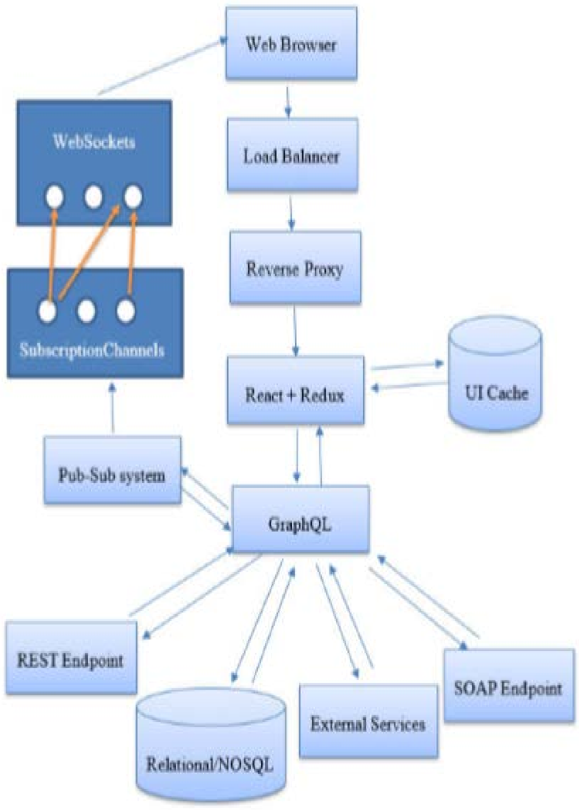


**Figure 3: Existing Architecture of Rendering real–time dashboards using a GraphQL–based UI**
**(Source: Eeda, Naresh, 2017)**

Contrary to the old method of loading all the data at once, data was gradually loaded onto the user interface without waiting for overall completion of the execution of other queries. They binded parts of the User Interface to the exact queries and loaded them immediately response was gotten. This allowed them to asynchronously load the data onto the UI. The fact that early component skeleton was kept in the redux, helped them in loading the user interface elements without the real data to be rendered as soon as the request was received. This improved the user experience by displaying the web page almost instantaneously.

#### *3.2.1. Disadvantages of the Existing System*

The following disadvantages of the Existing System are:
**ii) Inadequate modeling of highly interconnected data:** The efficient modeling of data was lacking in the Existing System.
**ii) Complexities involved in writing graph queries:** This is also a major issue associated with the existing system which arises from how efficiently we can find an unknown graph using distance or shortest path queries between its vertices

### 3.3 Analysis of the Proposed System

The GrapQL Model is deployed in a political party database and a graph database is built for a party administration system. In the proposed system, records of party members are collected and stored in a database as shown in figure 4. The various local governments regularly generate data of membership and various financial and non-financial activities which are stored in local databases in their offices. The states also have party offices which equally keep record of party officials and party activities at the state level. The parties also have a National Secretariat which equally has databases of members, officials, contestants and party officials. There are equally financial records at each level of the data store at the local government and states. The databases of the party members are kept separate from each other in many states. This leads to data redundancy as the same data are repeated in the relations. Accessing the databases requires generation of multiple queries. In other states where a relational databases are used, challenges are encountered as the records get larger.

Querying a large dataset is tedious and time consuming. The fact is that it involves multiple joins of the table leading to complex query formation. This has negatively affected the administration of political party system as query processing seems to be a very difficult task. Expanding a relational model is difficult as a result of strict schema maintained by the relational model. One of the relational model's design motives was to achieve a fast row-by-row access (Codd, 1970).Although relational model manages related records, many joins operations involve complex queries as many attributes of different tables are considered. Resolving relationships in relational model involves creating multiple joins with the tables. In working with relational models, foreign key constraints should be considered when retrieving relationships and this constitutes to the challenges faced by relational model. The cost and time of the query operation is on the increase too.
In the proposed system, the different records from the wards, local governments and states are collected and relationships among them are created. These records are stored in the graph database for easy access and query operation. Graph database is preferred because of its ability to handle big volume of data.

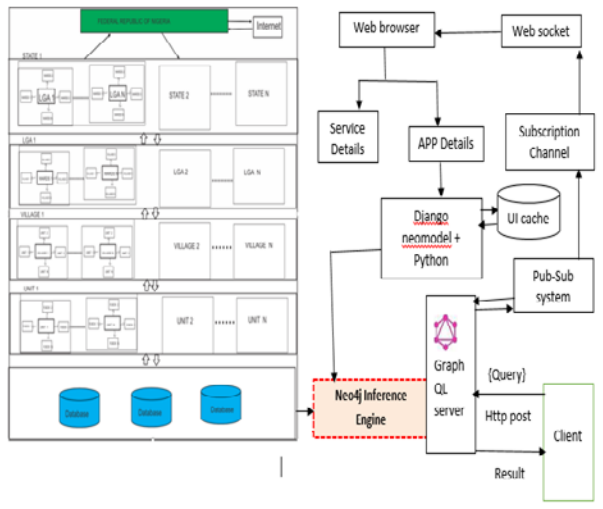


**Figure 4: Proposed System Architecture of an improved Graph Query Language Model**

Graph database overcomes some of the limitations of related databases. When it comes to a dataset that associates, graph database is faster. Graph database connects straight to the form of object-oriented programs. Normally, they are well suited to datasets that are very big as they do not not make use of computations involving joins. Join operations are very expensive. As they depend more on flexible schema, they are promoted as convenient to handle adhoc and developing schema.
A database is an important aspect of a software application. Apart from the fact that it stores information, it positively affect the general accomplishment of a software. In spite of the numerous benefits of relational database, however, they don't perform well with continuous increase in the volume of related data. So, selecting a database suitable for political party system is crucial. Using a graph database for a political party administration will make way for easy and fast query of records of party members.

The proposed architecture uses GraphQL to query the database of a political party system. The database has to be structured in form of graph for fast query. Graph databases perform well when the databases gets larger compared to relational databases. As the complexity in data and value in relationships increases, the ability of relational databases to address the data requirements decreases and use of graph database increases, which leads to the adoption of improved GraphQL system for a graph database in the proposed system. This will enable an easy and most private way to communicate with unit members in the whole state and also most promising channel of communication within party executives.

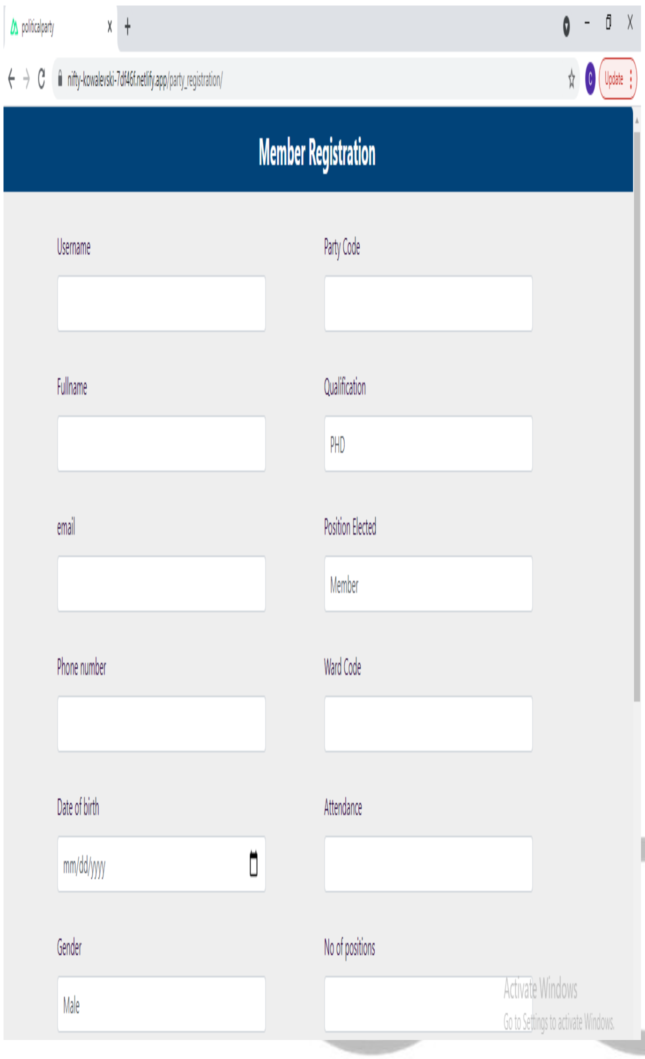
### 3.3.2 Advantages of the Proposed System

The following advantages of the proposed system are:

i) **Easy Query:** GraphQl query is easy to manage because of its flexibility. API does not require too much maintenance or modification.
ii) **The ability to make fast query:** The client can make only one request to retrieve only information they need. A backend server only needs to fetch and prepare what is being asked for, so the entire response can be delivered in a single network delivery.
iii) **Flexible Graph-Query Platform in the Neo4j Environment:** Neo4j uses property graphs to extract added value of data of any company with great performance and in an agile, flexible and scalable way. In terms of performance Graph databases such as Neo4j perform better than relational (SQL) and non-relational (NoSQL) databases.

## 4. IMPLEMENTATION AND SAMPLE RESULTS

To implement the improved GraphQl model for political party distributed database administration, detailed analysis were extensively done on the existing system to identify the major drawbacks faced in the storage method of political party information and to compare and contrast it with the new system. A graph database (Neo4j) is used in the new system instead of relational database system. Graph database helped us to model, query and expand data in a faster and more intuitive way when compared with a traditional SQL approach. For easy accessibility, the Neo4j database was hosted on the cloud. As the party member registers as shown in Figure 5, the information is store in the cloud.


politicalparty
nifty-kowalevski-7df46f.netlify.app/party_registration/
Update
Activate Windows
Go to Settings to activate Windows.

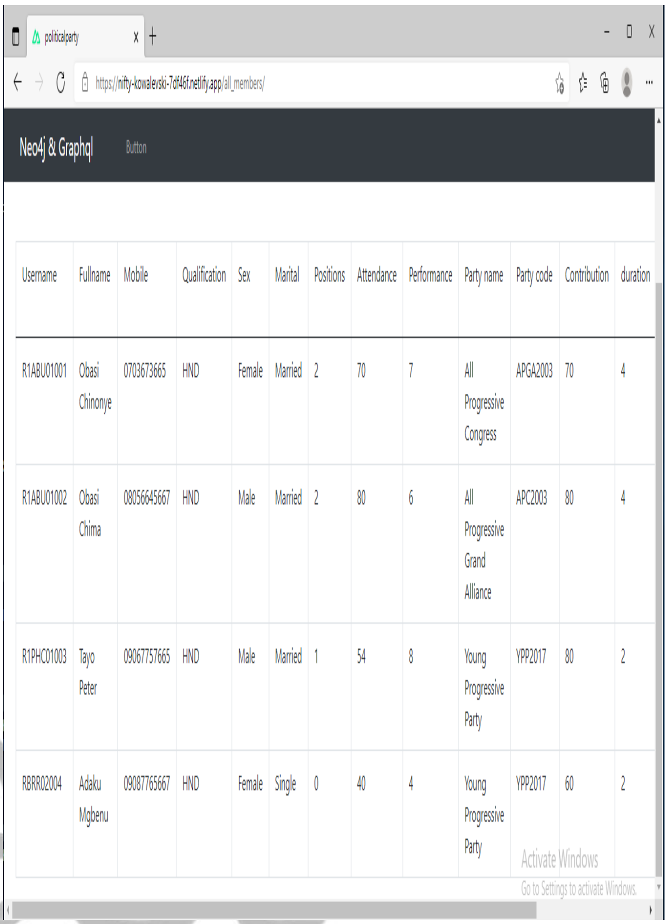
Member Registration

| | |
|---|---|
| Username | Party Code |
| Fullname | Qualification: PHD |
| email | Position Elected: Member |
| Phone number | Ward Code |
| Date of birth: mm/dd/yyyy | Attendance |
| Gender: Male | No of positions |

Figure 5: Registration  Form of the Party members

Neo4j is a non- relational graph database that is optimized for managing relationships. Neo4J database can help in building high performance and scalable applications that use large volumes of connected data. To add neo4j-based functionality to the proposed system, Django-neomodel plugin was used. Django neomodel is an Object Graph Mapper (OGM) for the neo4j graph database built on the awesome neo4j_driver. This module allowed us to use the neo4j graph database with Django using neomodel. With this, data displayed will be obtained from an API.

To handle the relationship between the database, the python models and other aspects of web development, an object mapper, Django was incorporated. A Django module allowed us to use the Neo4j database with Django using neomodel. Django made it possible to create the entire site and database backend, along with the  plugins used. Django is based on the idea of models. A model is a class of objects directly linked to objects in a database. To connect to the Neo4j database, Bolt protocol URL was used. Bolt connected only to the server with the IP specified. It will not route anywhere else. All queries over this protocol would go only to this machine, whether they're read or write queries. GraphQL was used as an abstraction layer to hide the database internals. Request sent by Client to GraphQL server would always be validated and executed by collecting the information from the Neo4j database. The data should be requested from the API. To create some level of abstraction, the results are displayed in tabular form. Figure 6 shows sample results of the query.


politicalparty
https://nifty-kowalevski-7df46f.netlify.app/all_members/
Neo4j & Graphql
Button
Activate Windows
Go to Settings to activate Windows.

| Username | Fullname | Mobile | Qualification | Sex | Marital | Positions | Attendance | Performance | Party name | Party code | Contribution | duration |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R1ABU01001 | Obasi Chinonye | 0703673665 | HND | Female | Married | 2 | 70 | 7 | All Progressive Congress | APGA2003 | 70 | 4 |
| R1ABU01002 | Obasi Chima | 08056645667 | HND | Male | Married | 2 | 80 | 6 | All Progressive Grand Alliance | APC2003 | 80 | 4 |
| R1PHC01003 | Tayo Peter | 09067757665 | HND | Male | Married | 1 | 54 | 8 | Young Progressive Party | YPP2017 | 80 | 2 |
| RBRR02004 | Adaku Mgbenu | 09087765667 | HND | Female | Single | 0 | 40 | 4 | Young Progressive Party | YPP2017 | 60 | 2 |

**Figure 6: Sample of GraphQL Query Results of all Party Members**

## 5. CONCLUSION

Relational Database is a mature way of storing records, the building block is table. Graph database is also another storage mechanism. The building block is graph. There is gradual increase in dataset as the day goes by. Hence the need for a database system that can accommodate the massive increase in data. As the amount of related data increases, the ability of a relational database to handle such data drops. Relational database is not efficient in handling highly interconnected data. Graph database on the hand displays great performance in handling highly interconnected data. GraphQL, which provides some level of abstraction is very efficient in querying graph database. Although it is not a graph database query language but an Application Program Language. It delivers query result within the shortest possible time. Querying Graph database with GraphQL helps in generating faster and easy query. The improved GraphQL model for a political party distributed database administration uses of a Neo4j database as a graph database . Based on the findings of the research work, the new system generates better results. Querying the database is faster and easy as GraphQL returns the result of a query through a single endpoint unlike its alternative REST that returns the result of a query through multiple endpoints. There is no under fetching and over fetching of data in graphql query.

## 6. ACKNOWLEDGMENTS

Our thanks go to the Almighty God for His grace and sustenance. We are also grateful to the lecturers in Computer Science Department, University of Port Harcourt, Choba, Rivers State, Nigeria for their inputs and support during the course of this research work.

## 7. REFERENCES

[1] Bhavani T. and Ammiel K. (2010). Secure Query Processing in Distributed Database Management

[2] Buna, S. (2017). REST APIs are REST-in-Peace APIs. Long Live GraphQL URL: https://medium.freecodecamp .org/rest-apis-are-rest-in-peace-apis-long-live-graphql

[3] Codd E.F. (1970). A relational model of data for large shared data banks. Communications of the ACM. 13(6). 377-387

[4] Eeda, Naresh (2017) "Rendering real-time dashboards using a GraphQL-based UI Architecture".Electronic Thesis and Dissertation Repository. 5136.

[5] Idowu, S.A, Maitanmi S.O. (2014) "Transactions-Distributed Database Systems: Issues and Challenges" International Journal of Advances in Computer Science and Communication Engineering (IJACSCE) 2(I), ISSN 2347-6788,24-26, Ilisan Remo, Ogun State, Nigeria.

[6] Jindal A, Madden S (2014) .GRAPHiQL: A graph intuitive query language for relational databases. In:2014 IEEE international conference on big data, big data, Washington, DC, USA, October 27–30,441–450

[7] Jing W., N. Nikos, T. Peter (2017), GraphCache: A Caching System for Graph Queries, Open Proceedings, 10.5441/002/edbt.2017.03, 13 – 24

[8] Njoku Donatus O., Nwokorie Chioma E., Madu Fortunatus U(2016). An Enhanced Query Processing Algorithmfor Distributed Database Systems,International Journal of Scientific & Engineering Research, 7(10)

[9] Smita A. and A. Patel (2016), A Study on Graph Storage Database of NoSQL, *International* Journal of Soft Computing, AI and Applications (IJSCAI), 5(1), 33 – 39

[10] Srinath S. (2016), Query Processing Issues in Data Ware Houses,Research gate Publications, https://www.researchgate.net/publication/2618638,1–13

[11] Stubailo, S. 2017. GraphQL vs. REST. URL: https://dev-blog.apollodata.com/graphql-vs□rest-5d425123e34b Accessed 22 November 2017

[12] Sturgeon, P. 2017. GraphQL vs REST: Overview. URL: https://philstur□geon.uk/api/2017/01/24/graphql-vs-rest-overview/ Accessed 20 November 2017

[13] Thuraisingham M. (2010). Secure Query Processing in Intelligent Database Management Systems.The MITRE Corporation, Burlington Road, Bedford.