International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 10 – No.5, February 2016 – www.ijais.org

TripleFetchQL: A Platform for Integrating Relational and NoSQL Databases

Oluwafemi E. Ooju
Department of Mathematics,
Ahmadu Bello University,
Nigeria

Sahalu B. Junaidu
Department of Mathematics,
Ahmadu Bello University,
Nigeria

S.E. Abdullahi
Department of Mathematics,
Ahmadu Bello University,
Nigeria

ABSTRACT
There has been an unprecedented growth and increase in the domain unstructured/semi-structured data is increasing over the years. While relational database systems remain popular and relevant, they are incapable of handling the growth level of unstructured data in the area of web applications. Conflating the benefits of a simple NoSQL storage engine with the relational databases and the unique ability of presenting query results from the duo to users at minimal costs have been a critical challenge in the research community. So far so good there has been a tremendous works such as using SQL with an extension of NQP on transformed NoSQL data which is store as SQL virtual relation, querying Apache Cassandra with SQL after altering the data structure, querying both world of relational and NoSQL and produced two results instead of single output. The TripleFetchQL system developed enables users to query relational and NoSQL databases and presents query results as if they were querying the familiar relational database alone. Furthermore, TripleFetchQL provide applications the ability of leveraging the benefits of relational and NoSQL databases at the small cost of learning the simple syntax of the TripleFetchQL system.

Keywords
ACID, CAP, NOSQL, TFQL

1. INTRODUCTION
Database was introduced in 1960 as a simple layer that would serve as fundamental principle behind Information systems. New architecture of separating application from data was introduced [Codd, 1970]. In 1970s Edgar Codd proposed Relational model for storing Data. Relational model uses SQL to let applications find data within tables [Smith, 2013].

In recent years, applications began to produce wide range of data through complex systems. These large amounts of data gave rise to concerns like database structure, scalability, and availability of data which emerged the term ***NoSQL*** [Stonebraker, et al., 2007].

Different databases are designed to solve different problems. Using a single database engine for all of the requirements usually leads to non-performant solutions. RDBMS solutions are good at enforcing that relationships exist. [Pramod & Martin, 2012].

A relational database is no more an ideal database system that is fit for the massive growth and the unstructured data of certain modern web applications in terms of the immense amount of data, yet some data are still relational bounded.

The now increasingly popular NoSQL alternatives enable applications to sacrifice data consistency which is part of SQL ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability) properties in benefit of the factors more important to modern web applications which are: availability, scalability and performance [Brewer, 2012].

Large websites must now be able to serve billions of pages every day and web users expect that their data are ubiquitously accessible at the speed of light no matter the time of day.

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are increasingly used in big data and real-time web applications.

These databases intend to be almost schema-less and not as strict as their relational counterparts on what concerns the data model, in order to achieve higher scalability.

In this paper, there is an attempt to combine both relational and NoSQL world on a single platform to enable users and developers exploit the benefits of both on a click in the proposed TripleFetchQL System and Aggregate Query Syntax with the introduction of ***transformation agent***.

Considering the proposal, the system was implemented and used to query MySQL, MongoDB and Apache Cassandra databases respectively. This solution eliminates any manual alterations by users and unified results from the three involving databases into one tabular SQL-Like format. The prototype system was tested and worked well and it's believed it can be adapted into practical environment.

The rest of the paper is organized as follows. Section 2 highlights related work, Section 3 presents the proposed TripleFetchQL System, in Section 4, System evaluation and performance was done over others. Section 5 handles the conclusion while references are given in Section 6.

2. RELATED WORKS
Overview of the current and up-to-date research on the works related to the intended goal of this paper was discussed in this section.

In 2009, an overview of non-relational data models and how they are differing from relational model was placed on scholar palm by [Varley 2009]. He tries to determine which is best among the two model paradigms; using data modeling by considering their strengths and weaknesses. His research provides a good background to this work, but the issue of combination and data retrieval was not addressed.

[Strauch 2011] gave an in-depth introduction to NoSQL. He describes the rationales behind the NoSQL, techniques and algorithms for solving some issues of concerns. He concluded that NoSQL is not a replacement for SQL which gives rooms for rational thinking of bringing the two together in a single application which he didn't mentioned.

The very first extensible framework for coordinating queries across SQL and NoSQL using ANSI SQL queries on top of

Cassandra was presented by [Ferreira 2012]. His tactic allows migration of data from both data stores without losing the transactional guarantees given by the traditional relational system. His approach did not look into the situation where there will be need for data combination from both databases.

[Roijackers 2012] attempts to bridge the gap between the SQL and NoSQL with the idea of transforming the NoSQL data into triple format and incorporate the triples into SQL database as a virtual relation. His implementation accepts a single query language which is SQL queries which is extended with NoSQL query patters. The original NoSQL is reconstructed via series of joins from the relations. This approach requires a lot of transformation of data from one form to another which in process could lead to data loss and also delay in data access.

The most recent closest related work in this area is the work of [Adeyi *et al* 2013/2014]. They create a system called *DualFetchQL System* which integrates Relational and NoSQL databases and also claimed to present a unified output on using the derived aggregate query syntax. This obviously did not happened in there finally output which leads to manual alterations by users before full knowledge of data pulls from the databases.

3. TRIPLEFETCHQL SYSTEM

3.1 Theoretical Framework

The architecture of the system is made up of five major phases namely, Application, Abstraction Layer, TripleFetchQL Function, Transformation Agent and the databases as show in Figure 2.

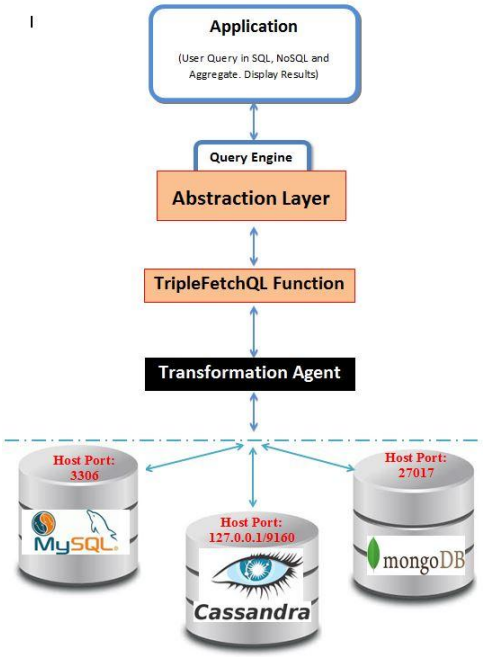


Figure 2: The TripleFetchQL System Architecture

Application phase of the design enables users to input queries in either SQL or NoSQL in addition to the aggregate query as well as the unique capability of displaying the result of the processes**.**

Abstraction Layer is the domain of the user. It contains basically two parts; the part where users can enter query and the other part where result of query is being displayed. User query is passed on to the TripleFetchQL Function. When the Execute action is initiated by the user, the TripleFetchQL Function processes the query and sends back the result to the Abstraction Layer.

TripleFetchQL Function oversees the relationship between other components of the system. It also determines how the system functions. It extracts the query entered by a client from the Abstraction Layer, determines the type of query and interacts with the databases and finally sends back result of the query to the Abstraction Layer for display.

Transformation Agent is responsible for transforming the output of the aggregate query across all the databases with the result being displayed in a tabular form. This helps users to eliminate manual edit needed in combining the data.

Apache Cassandra is the environment where the Apache Cassandra server resides. All the Cassandra related queries which were handed over by TripleFetchQL Function for execution are handled here. The result of the query will be sent back to the TripleFetchQL Function which will in turn transfer it to Abstraction Layer.

MongoDB is the environment where the MongoDB's server resides. All MongoDB related queries are handed over to the MongoDB by the TripleFetchQL Function for execution. The query is executed with the result returned back to the TripleFetchQL Function.

MySQL, this is the MySQL database's server side. All SQL related queries are handed over to the MySQL by the TripleFetchQL Function for execution. The query is executed with the result returned back to the TripleFetchQL Function.

3.2 TripleFetchQL Implementation

In other to successfully implement TFQL System, the three involving databases which are MySQL server as the SQL database manager, MongoDB as Document-Store family of NoSQL and Apache Cassandra as a Column-Store family of NoSQL was choosing carefully based on the fact that no other researcher has worked with three databases across the two world, acceptability of the databases, fault tolerant, ease of use and maintenance, free to use/open source, among the top 10 ranking databases, among many other reasons. They are also chosen because of their Java API ability which aids easy communication.

TFQL System uses derived aggregate query syntax to query data across all the databases and present to the user a unified result with the help of the transformation agent. The TFQL only transform the data pulled from the databases to SQL like tabular form without any alteration to the data in the databases.

3.3 Aggregate Query Syntax

New query syntax was developed to query all the three databases. The purpose of this query syntax is to bridge the communication gaps between the relational and NoSQL databases. It is made up of two major component which is separated with key word "and". The NoSQL component of the query was later partitioned into two components using the same keyword "and". There are actually three major keywords: SQL, NoSQL and AND. The SQL keyword enables the system to recognize query that's pertaining to SQL while the NoSQL enable the system to recognize query pertaining to NoSQL. The keyword AND helps in separating the query accordingly to their various databases servers for execution and also tell the system that the users is communicating with more than one database. Find below the syntax of the aggregate query:

SQL[SQL Query Syntax] **and NoSQL**[MongoDB Query Syntax and Cassandra Query Syntax]

3.4 System Workflow

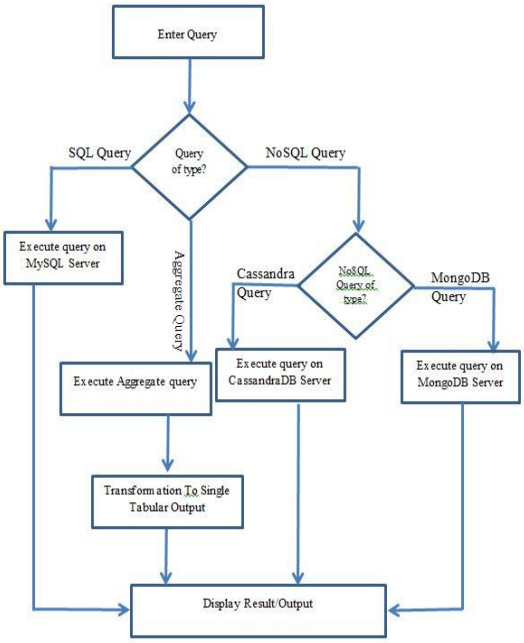


Figure 3: The System Workflow of TFQL

The workflow illustrates how the system executes the different user queries in line with the "of type" ability of the system.

4. SYSTEM EVALUATION AND COMPARISON

In order to prove the capability of the TFQL System, some screenshot of the system were made when it's performing aggregate query functionality with and without the transformation agent. This is to prove to the world that there are lot benefits when the relational world and NoSQL world were brought together.

Figure 4a show to us data pulled from the three databases according to the aggregate query statement issued by the user before the operation of transformation agent.

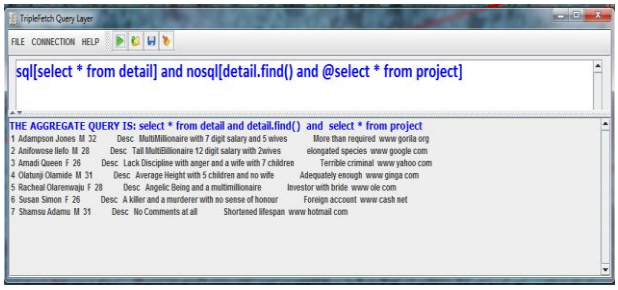
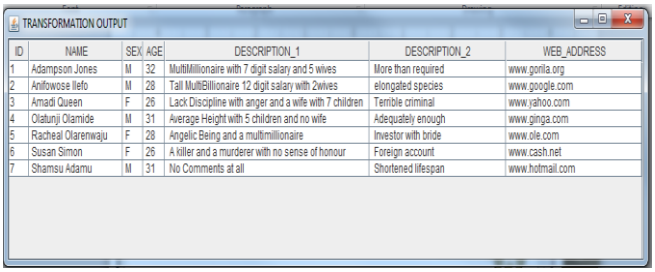


Figure 4a: Screenshot of Aggregate Query without Transformation

Figure 4b show to us the effect of transformation agent on the result of the aggregate query displayed in figure 4.1 and how it helps unified the result without the need of any manual alterations for the users, no identification of which database hold which data and finally, the result is SQL like.



ID	NAME	SEX	AGE	DESCRIPTION_1	DESCRIPTION_2	WEB_ADDRESS
1	Adampson Jones	M	32	MultiMillionaire with 7 digit salary and 5 wives	More than required	www.gorila.org
2	Anifowose Ilefo	M	28	Tall MultiBillionaire 12 digit salary with 2wives	elongated species	www.google.com
3	Amadi Queen	F	26	Lack Discipline with anger and a wife with 7 children	Terrible criminal	www.yahoo.com
4	Olatunji Olamide	M	31	Average Height with 5 children and no wife	Adequately enough	www.ginga.com
5	Racheal Olarenwaju	F	28	Angelic Being and a multimillionaire	Investor with bride	www.ole.com
6	Susan Simon	F	26	A killer and a murderer with no sense of honour	Foreign account	www.cash.net
7	Shamsu Adamu	M	31	No Comments at all	Shortened lifespan	www.hotmail.com

Figure 4b: Screenshot of Aggregate Query with Transformation Agent

Table 1 shows both similarities and differences between TFQL and that of (Roijackers, 2012) and (Adeyi, 2014) in terms of platform acceptability for more databases, query support, aggregate functionality and unified output of aggregate query.

5. CONCLUSION

TFQL System was designed to acts as a software layer for querying both relational and NoSQL databases with the help of the derived aggregate query syntax when there is need for unified data across the three sampled databases.

TFQL was tested with the aggregate query syntax against MySQL, MongoDB and Apache Cassandra and showed how it handled the issued query with no hesitation and produced the required result in a unified form.

To further enhance this research, the following areas can be looked into:

- Looking into the time taken to transform the data pulled from the databases by the users query into the tabular output by the transformation agent.
- Reviewing the Aggregate Query Syntax by merging the three components into just one so as to avoid the users the burden of knowing more than two query languages.

This system is built basically on MongoDB and Apache Cassandra as a sample of NoSQL family, research can exploit other members apart from the column store and data store.

Table 1: Result Evaluation and Comparison with other Systems

	Roijackers 2012	Adeyi 2013/2014	TripleFetchQL 2015
SQL			
Select	Yes	Yes	Yes
Create	Yes	Yes	Yes
Insert	Yes	Yes	Yes
Update	Yes	Yes	Yes
Delete	No	Yes	Yes
Alter	No	Yes	Yes
Grant	No	Yes	Yes
Revoke			
MONGO DB			
Find()	No	Yes	Yes
Remove()	No	Yes	Yes
Drop()	No	Yes	Yes
Update()	No	Yes	Yes
Insert()	No	Yes	Yes

Create()	No	Yes	Yes
CASSANDRA DB			
Select	No	No	Yes
Insert	No	No	Yes
Create	No	No	Yes
Create Keyspace	No	No	Yes
Delete	No	No	Yes
Update	No	No	Yes
Alter	No	No	Yes
Drop	No	No	Yes
Grant	No	No	Yes
Revoke	No	No	Yes
AGGREGATE FUNCTIONALITY			
Aggregate Query	No	Yes	Yes
Single Result/Output	No	No	Yes

6. REFERENCES

[1] Adeyi, T. S., Abdullahi, S. E., & Junaidu, S. B. (2013). DualFetchQL System: A Platform for Integrating Relational and NoSQL Databases. *IJERT Vol. 2, Issue 12, December*, 1973 - 1981.

[2] Brewer, E. A. (2012). CAP twelve years later: how the 'rules' have changed. *Computer, Vol. 45 No. 2*, 22 - 29.

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

[4] Desire, A. (2014). Can SQL and NoSQL databases live happily together? *An Article Retrieved: http://www.techradar.com/news/internet/web/can-sql-and-nosql-databases-live-happily-together--1278322/1*.

[5] Ferreira, L. (2012). Bridging the Gap between SQL and NoSQL: SQL and ACID over a VLSD. *Master Thesis, Universidade do Minho.*

[6] Nance et al. NOSQL VS RDBMS - WHY THERE IS ROOM FOR BOTH. Proceedings of the Southern Association for Information Systems Conference, Savannah, GA, USA, March 8th–9th, 2013

[7] Pramod, J. S., & Martin, F. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. *Retrieved: http://www.thoughtworks.com/insights/blog/nosql-databases-overview*.

[8] Roijackers, J. (2012). *Bridging SQL and NoSQL. Master Thesis.* Eindhoven University of Technology.

[9] Roijackers, J., & Fletcher, G. H. (2013). On bridging relational and document-centric data stores. *BNCOD' 13 Proceeding of the 29th British National Conference on Big Data.*, 135 - 148.

[10] Smith, S. (2013, October 13). Why NoSQL database is used by Facebook, Google and LinkedIn Application? *Retrieved from http://blog.outsourcing-partners.com/2012/10/why-nosql-database-is-used-by-facebook-google-and-linkedin-applications/*.

[11] Stonebraker, M., Madden, S., Abadi, D. J., Harizopoul, S., Hachem, N., & Helland, P. (2007). The end of an architectural era: (it's time for a complete rewrite)," in Proceedings of the 33rd international conference on Very large data bases.

[12] Strauch, C. (2011). NoSQL Databases - NoSQL Introduction and Overview. *Paper retrieved: http://highscalability.com/blog/2011/4/13/paper-nosql-databases-nosql-introduction-and-overview.html*.

[13] Varley, I. T. (2009). No Relation: The Mixed Blessings of Non-Relational Databases. Master Thesis in The University of Texasat Austin. Retrieved: http://ianvarley.com/UT/MR/Varley_MastersReport_Full_2009-08-07.pdf.