International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 12, December - 2013

DualFetchQL System: A Platform for Integrating Relational and NoSQL Databases

ThankGod S. Adeyi, Saleh E. Abdullahi, Sahalu.B Junaidu

Department of Mathematics, Ahmadu Bello University

Zaria, Kaduna State, Nigeria.

ABSTRACT: *Relational Databases or RDBMS has forty years of production experience as it has been the dominant model for database management since it was developed by Edgar Codd in 1970.*

A new database model called NoSQL is gaining significant attention in the enterprise, recently. NoSQL databases are non-relational data stores that have been employed in massively scaled web site scenarios, where traditional relational database features matter less, and the improved performance of retrieving relatively simple data sets matters most.

A big challenge in the research community is to conflate the benefits of a simple NoSQL storage engine (scalability, fault tolerance) with the benefits of relational databases (transactions, usability, consistency guarantees), where possible. Problems arise when a single software product requires data storage where a part of the data is ideally stored in a NoSQL database, whereas the rest of the data is perfectly relational and thus well-suited for a traditional SQL database.

In this paper, we present DualFetchQL system that provides a platform for accessing and integrating data from MySQL, representing the Relational databases and MongoDB for the NoSQL databases. We introduced a new query syntax called Aggregate Query which is used when combined data from these two separate worlds is required in an application. The Aggregate Query syntax was tested on our DualFetchQL system.

1. INTRODUCTION

NoSQL databases have had an enormous growth with the massive usage of social networks, such as facebook and twitter. This does not, however, imply that relational databases have been outdated. In order to understand the actual differences between these ways of storing and retrieving data one has to take a closer look at each of them. In doing so, we might find that they are not that incompatible, and that some benefits can be taken from a mix of both.

On one hand there is the NoSQL approach, which offers higher scalability, meaning that it can run faster and supports bigger loads. On the other hand, a Relational Database Management System (RDBMS) offers more consistency as well as much more powerful query capabilities and a lot of knowledge and expertise gained over the years [Stonebraker 2010].

In this paper, we attempt to join these two worlds by creating a platform to enable one enjoy the best of both worlds. We take a step towards bridging the worlds of relational and NoSQL data, with our proposed DualFetchQL System and Aggregate Query syntax. Based on these two proposals, the system was implemented and used for querying over relational and NoSQL stores seamlessly. The prototype system works well and we believe it can be adapted in practical environments.

The rest of the paper is structured as follows. Section 2 highlights related work and Section 3 presents the proposed DualFetchQL system. We evaluate performance of the system in Section 4 and conclude in Section 5. References are given in Section 6.

2. Related Work

In this section we give an overview of current research on the topics related to our intended goal to bring relational and non-relational data closer together.

A detailed survey of Non Relational Databases was presented in [Varley 2009]. His research gives an overview of non-relational data models and how they differ from the relational model. Throughout the research, Varley tries to determine a winner between the two model paradigms from a data modeling perspective by considering various strengths and weaknesses and comparing them side by side. His research provides a good background for our work, but the issue of data retrieval from the two database models was not addressed at all.

A thorough introduction to NoSQL was provided in [Strauch 2011]. The paper describes the rationales behind the NoSQL movement, some common techniques and algorithms for solving issues concerning, e.g., consistency and distributed data processing, and also presents a number of concrete systems with implementation details and data models. His focus was on proving that NoSQL is not a replacement for SQL but was not concerned about how data from both databases could be used in a single application.

[Ferreira 2012] presented the first generic extensible framework for coordinated querying across SQL and NoSQL stores. His work attests the feasibility of the general approach by providing a prototype that enables the execution of ANSI SQL queries on top of Cassandra. His approach allows doing migration of data from both datastores without losing the transactional guarantees given by a traditional relational system. He did not address a situation where combined data is required from the two databases.

The closest related effort in this area that we are aware of is the work of [Roijackers 2012]. His thesis attempts to bridge the gap between SQL and NoSQL by transforming the NoSQL data to a triple format and incorporating these triples in the SQL database as a virtual relation. His implementation accepts a single query language; ordinary SQL queries extended with NoSQL query patterns. Via a series of self joins, the original NoSQL data can be reconstructed from this triple relation. Obviously, this approach of bridging the gap between SQL and NoSQL requires much work to be done in converting data from one form to another as it involves many expensive join operations to get the original data which in the process could lead to data loss and time delays in data access.

3. DualFetchQL System

3.1 System Architecture

The architecture of the system developed is made up of four major phases, as shown in Figure 1.1.

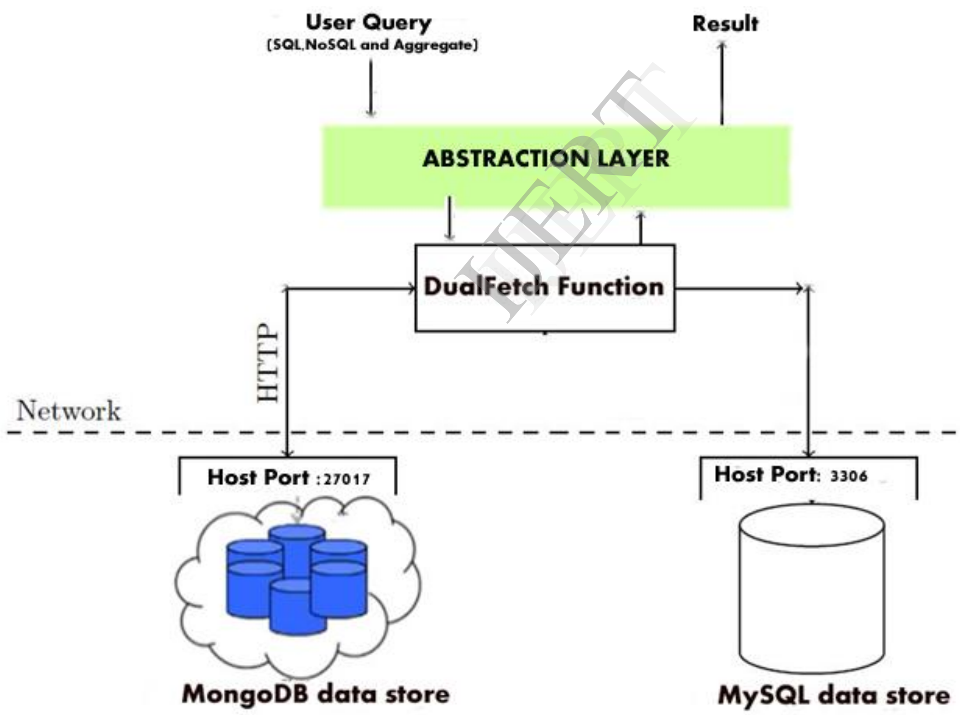

Figure 1.1: The DualFetchQL Architecture

The architecture consists of a DualFetch Function, an Abstraction Layer, a MongoDB data store and a MySQL data store.

The **DualFetch Function** phase forms the center of the architecture and oversees the relationship between other components of the system as well as determining 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 appropriate databases and, finally, sends back result of the query to the Abstraction Layer for presentation.

The **Abstraction layer's** phase is the domain of the user. It contains basically two parts; the part where users can enter queries and the other part where results of queries are displayed. A user query is passed on to the DualFetch Function's phase when the *execute* action is initiated by the user. The DualFetch Function's phase processes the query and sends back the result to the Abstraction Layer.

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

The **MySQL data store's** Phase hosts the MySQL Database's server. All SQL related queries are handed over to the MySQL data store by the DualFetch Function for execution. The query is executed with the result returned back to the DualFetch Function.

3.2 Implementing the DualFetchQL System

To implement the DualFetchQL System, we needed to choose one RDBMS and one NoSQL implementation. The chosen systems were the MySQL server as the SQL database manager and the MongoDB as the NoSQL implementation. These choices were made mainly because of the popularity of the two systems and also the fact that they both have Java interfaces that make them easier to communicate.

DualFetchQL System presents a software layer for querying both SQL and NoSQL databases. We created a new query syntax called aggregate query that is used when data from both SQL and NoSQL databases are both required in a single view.

The DualFetchQL System parses user queries to determine in which of the underlying databases the required data is located and return the result of the query on the result panel of the Layer.

3.3 The Derived Aggregate Query Syntax

We present new query syntax, called Aggregate Query to bridge the gap between the relational database (MySQL) and the NoSQL database (MongoDB). It supports reading data from both data sources. The Aggregate Query syntax is made up of two major components separated by an "**and**" key word. One component is the SQL query that recognizes the conventional SQL query while the other component is the NoSQL query that recognizes the basic MongoDB (CRUD operations) query.

The syntax of the Aggregate Query is shown below;

NoSQL<mongoDB query> **and SQL**<sql query>

OR

SQL<sql query> **and NoSQL**<mongoDB query>

This syntax contains a set of reserved words whose semantics is easy to understand by a programmer. For instance, the “**NoSQL**<MongoDB Query>” component enables users to define a valid MongoDB query statement while the “**SQL**<SQL Query>” component enables users to define a valid SQL query statement.

The “**NoSQL**” keyword signals the coming of a “nosql query” statement bracketed between the closest pair of opening and closing angular brackets, “<” and “>” . An SQL query is signaled in a similar manner with the "SQL" keyword.

The “**and**” keyword indicates that the user is interested in getting a combined result from two underlying data stores as specified in the query.

4. Result Evaluations

Screenshot of a query that involves the use of Aggregate query to retrieve combined data from the two data stores is shown in Figure 1.2.

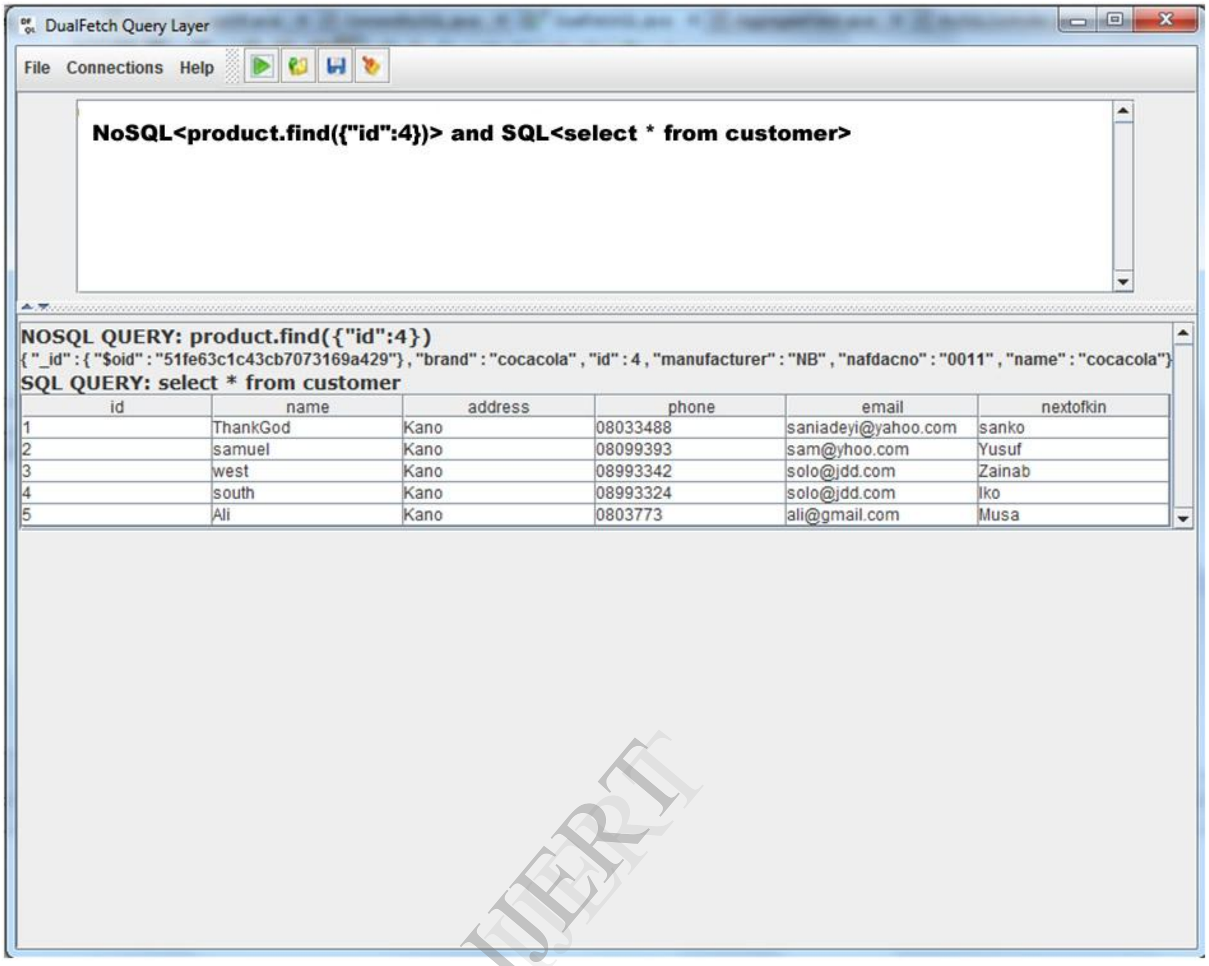

Figure 1.2*:* The screenshot of the use of Aggregate query to retrieve combine result.

Screenshot of a query that involves the use of Aggregate query to retrieve combined data from the two data stores with error in one part of the query is shown in Figure 1.3.

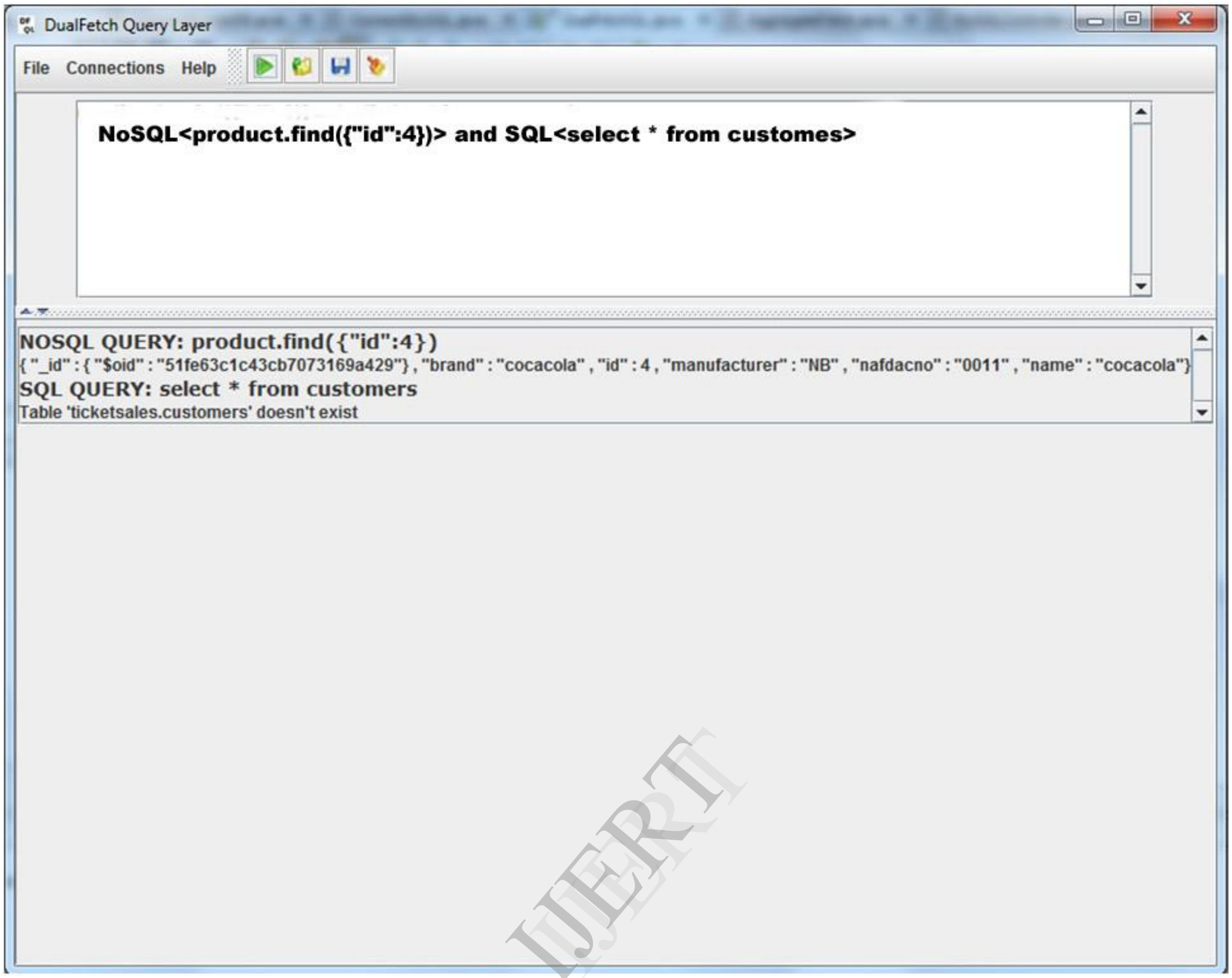

Figure 1.3: The screenshot of the use of Aggregate query to retrieve combine result with error in the SQL table name.

5. CONCLUSION

We designed and implemented a DualFetchQL System that acts as a software layer for querying both SQL and NoSQL databases. We created a new query syntax called aggregate query that is used when data from both SQL and NoSQL databases may be required in a single view.

The DualFetchQL System parses user query, determines in which of the underlying databases the required data is located and returns a result.

We tested the Aggregate query syntax against MySQL and MongoDB databases and showed how the DualFetchQL system generates reports for the given queries.

- ✓ The research presented in this paper may be enhanced in the following ways: This system is built specifically for the case of MySQL and MongoDB. An area to be explored would be building such a system for other NoSQL databases and if possible build a generic transactional system for most, if not for all, NoSQL database families.
- ✓ Improving the user interface for administrators where application can be tuned to categorize users based on the type of queries they can perform.
- ✓ Reviewing the Aggregate query syntax by merging the two components into just one to avoid the users the burden of knowing two query languages

6. REFERENCES

1. Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. Semantics preserving SPARQL-to-SQL translation. Data & Knowledge Engineering, 68(10):973–1000, 2009.
2. Codd, E.F. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377–387, 1970.
3. International Conference on Advanced Information Networking and Applications, 927-932, 2013.
4. Luís Ferreira. Bridging the Gap Between SQL and NoSQL:SQL and ACID over a VLSD. Master's thesis, Universidade do Minho,October 2012.
5. 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
6. PETTER NÄSHOLM. Extracting Data from NoSQL Databases:A Step towards Interactive Visual Analysis of NoSQL Data. Master's thesis, University of Gothenburg,Sweden, January 2012.
7. Roijackers John. Bridging SQL and NoSQL. Master's thesis, Eindhoven University of Technology, May 2012.
8. Shuxin, Y. and Indrakshi, R.. Relational database operations modeling with UML, Proceedings of the 19th, 2005
9. Stonebraker, M. SQL databases v. NoSQL databases. Communications of the ACM, 53(4):10–11, April 2010.
10. Strauch, C. NoSQL Databases. Accessed January 25, 2012. Feb. 2011.
11. Varley, I.T. No Relation: The Mixed Blessings of Non-Relational Databases". MA thesis. The University of Texas at Austin, 2009.

ThankGod Sani Adeyi is a masters student of computer science in the Department of Mathematics Ahmadu Bello University, Zaria-Nigeria. His current research interests include NoSQL Databases, e-payment system strategies and adoptions in Nigeria and cloud computing. Adeyi earned a B.Sc. degree in computer science from Benue State University Makurdi, in 2007. He is currently a staff of Pyrich Global Services Limited, Abuja.

Dr. Abdullahi, E. Saleh is a visiting Senior Lecturer at Mathematics Department of Ahmadu Bello University, Zaria. He earned his MSc. degree in Computer Science at University of Lagos – Nigeria in 1990 and PhD Computer Science at Queen Marry College, University of London. He is currently the Acting Managing Director/Chief Executive Officer of Nigerian Mobile Telecommunication Limited. His areas of research include Operating Systems, Memory Management, Programming Languages and Simulation.

SB. Junaidu is professor of Computer Science at Mathematics Department of Ahmadu Bello University, Zaria. He earned his MSc. degree in Computer Science at Queen Marry & Westfield College, University of London in 1992 and PhD Computer Science at St. Andrews University Scotland, UK in 1998. He is currently the Director of Iya Abubakar Computer Center, Ahmadu Bello University, Zaria. His areas of research include Parallel Computing, Web Applications Development, Programming Languages, Computer Science Education, E-Learning.