Context Based Cassandra Query Language

Shivendra Kumar Pandey
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi, India
shivaiert@gmail.com

Sudhakar
Indian Computer Emergency Response Team
Ministry of Electronics & Information Technology
New Delhi, India
sudhak82_scs@jnu.ac.in

***Abstract*— NoSQL databases are distributed, non-relational databases designed for large-scale data storage and for parallel data processing across a large number of commodity servers. Cassandra is a NoSQL database that stores data in non-related tabular forms. Cassandra works on "query at a time" and "query at a table" concept. In our daily life the queries of a user are related to each other. If queries by a user are related, that is, the current query is related to the previous query, then there is no support in Cassandra to state this. Cassandra runs each query on the entire table because Cassandra has neither memory to remember the result of a previous query, nor supports VIEW or JOIN on tables (or keyspace). Cassandra uses Big Table (of Google) for data storage which has thousands of columns and millions of rows, so it is not efficient to run the query on such a huge table, after knowing that the result of the previous query is sufficient to answer our current query. To solve such problems of Cassandra database, we implemented a new query language named as "Context Based Cassandra Query Language". CBCQL is internally mapped to Cassandra Query Language (CQL) so it has the same power as Cassandra, but provides additional functionality of querying on result of previous query. In this paper, CBCQL is explained with examples.**

Keywords—Cassandra; database; NoSQL; CQL; big data; query language; VIEW; JOIN; CBCQL.

I. INTRODUCTION

With the development of technology and internet users, there is a need for a system that can manage data efficiently and provide high performance [1]. Relational databases are facing many challenges, especially in scaling, concurrency and in providing write throughput [2]. To solve these problems, a new type of non-relational database management system was developed. This system is known as NoSQL. NoSQL databases are highly scalable, non-relational databases and provide high read/write throughput. They support Big Data and can run on a cheap commodity server. Big Data is a heterogeneous mixture of structured, semi structured and unstructured data [3]. NoSQL is an abbreviation of "Not only SQL" [4] [5]. These databases are very popular in companies for their cost, performance, and scalability.

In our world, we see data as a large heterogeneous collection of structured and unstructured data [6]. The main idea behind NoSQL databases is that they can store and retrieve structured (any relational database that has some schema), semi structured (XML or CSV file) and unstructured data (pdf, doc, email) efficiently [7]. They support distributed data storage and distributed computing and therefore, do not have a single point of failure [8].

In this work we have used Cassandra database. Cassandra is a distributed, column oriented, NoSQL database with high scalability, high availability and provides high performance with no single point of failure. Cassandra is the best choice for the companies that need reliability, high availability and very fast performance. Cassandra has very write throughput and good read throughput with flexible schema . Cassandra uses Big Table's data model of Google for data storage and the data distribution concept of Amazon Dynamo [9].

A. Motivation

CQL is used to access Cassandra database. CQL is essentially 'query-at-a-time' language. That is, each query is executed, the result is given to the user and has no bearing on the next query. We believe, that users tend to ask a series of related queries which is dictated by a thought process. Consider an example. Suppose a user wants to buy a flat or home and he wants to select the best suitable one. He/she will execute a CQL query for finding the flats. A typical table in Cassandra has thousands of columns and millions of rows. As a consequence, the result will also contain millions of rows. In such a situation, it will be very difficult for the customer to select a flat or a home as per his need in such a huge table. Since Cassandra does not have memory to remember the result of the previous query, so every time user has to query the whole table. Consider the example given below:

1) Select price, baths, beds, city, area, parking_lot, placeid, rpayment, sq__ft , type ;

2) Select price, baths, beds, city, area, parking_lot, placeid, rpayment, sq__ft , type WHERE type = 'Residential ';

3) Select price, baths, beds, city, area, parking_lot, placeid, rpayment, sq__ft WHERE type = 'Residential' and beds = 3;

4) Select price, baths, beds, city, area, parking_lot, placeid, rpayment, sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 ;

5) Select city , price, area, parking_lot, placeid, sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 and parking_lot='yes';

6) Select city ,price, area, parking_lot, placeid, sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 and parking_lot='yes' and city='SACRAMENTO';

7) Select area, price, placeid, sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 and parking_lot='yes' and city='SACRAMENTO' and area= 'open';

8) Select area, price, placeid, sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 and parking_lot='yes' and city='SACRAMENTO' and area= 'open' and price < 10000;

The above sequence of queries reflects the thought process of the user. In the first query user selects the columns that are important to him. In subsequent queries, he specifies some conditions to get the best suitable deal. It is clear from the queries that there is no need to search the entire database every time. We have to just reduce the number of rows by putting conditions on columns. But Cassandra does not support this functionality. We have tried to overcome this limitation by proposing CBCQL. Our proposed prototype has the capability of creating, saving, recalling and deleting the context.

II. RELATED WORK

A lot of work has been done in the field of non-relational database. The term NoSQL was first used in 1998 for a relational database that omitted the use of SQL [10]. But now a day, the term NoSQL is used to differentiate non-relational databases from relational databases. NoSQL databases are providing high performance, but they have a lot of security issues. Till now we have three (or four by Tudorica [11]) main types of NoSQL databases [4] [10]. Most of the NoSQL databases are non-relational, query-at-a-time and query-at-a-table. Hence the concept of context can be used to improve performance, and make them user friendly. The notion of context as defined here has been proposed earlier in [12], [13] and [14]. In [12], context is defined for a network query language and in [13], a component based query language includes the definition of a context. Stream based query language incorporated context in [14]. However, none of these uses Context within the framework of CQL. We have used the concept of context in our work to facilitate analysis of data where the result of previous query is sufficient to answer the present query of the user. The context in our proposed work is designed in such a way that it fetches result from the context and updates the context with the result. We are providing the facility of saving a context and recalling it, so backtracking will be also easy for the user while querying. To reduce the typing mistakes, we made the CBCQL case insensitive.

III. CONTEXT BASED CASSANDRA QUERY LANGUAGE

In CBCQL, a query is similar to a CQL query, but the FROM clause is not present. The data to be picked up is available in the context and thus, the FROM clause is done away with. Every query is executed in the current context. It, in turn, updates the context which forms the context for the subsequent query. In addition, constructs to save and restore context are also defined. This additional functionality allows the user to go back in the sequence of queries and follow a different path for querying.

A. Context

A context consists of the table of interest and the data corresponding to it. Since Cassandra does not support JOIN operations on tables, it is not desirable for us to have more than one table in a context. Initially, the context is null. It contains no table or data. After creating a context, the first command of the user, is to add a table. The table and the data in the table, now define the context for the first query.

There are six types of queries in CBCQL. To reduce the possibility of errors, we made the syntaxes case insensitive (except the syntax "WHERE" used in select query). They are as follows:

- Create Context: (Create Context<context name>;)
- Add Table: (Add Table<table_name>;)
- Select: (Select<column_name1>,<column_name2>,<column_name3>;)
- Save Context: (Save Context as<context_name>;)
- Recall Context: (Recall Context<context_name>;)
- Delete Context: (Delete context<context_name>;)

B. State Diagram of CBCQL System

The state diagram of CBCQL system is given below. When we create a context, it will go to state p(0,0). Initially context is empty. When we add a table, it will go to state q(r1,c1), where r1 is the number of rows in the table and c1 is the number of columns at state q. When we use Select statement or Recall statement, it will go to state r(r2,c2). This is because, in both the cases, the context is modified. If we delete our current context, the context will go to state p(0,0) which has no records. Again, we have to add a table if we want to query further in a context. The state diagram of CBCQL system is shown below.

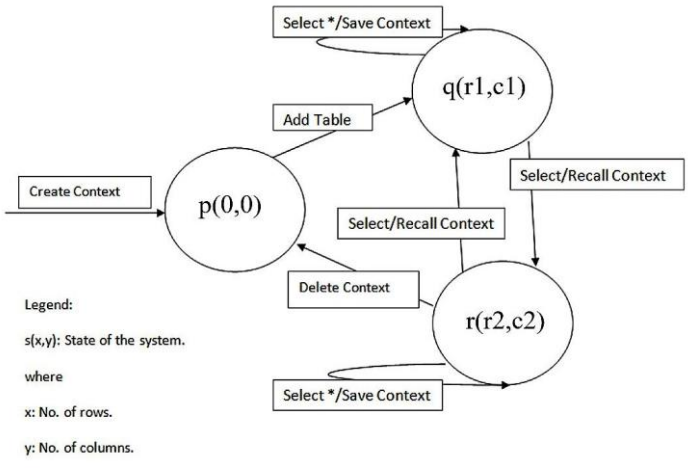

Fig. 1. State diagram of CBCQL system.

C. An Example of Querying in CBCQL

Let's see a sequence of queries, on a database for buying a home online in a context based environment.

1) Select placeid , price , baths , beds , city, WHERE type='Residential';

2) Select placeid , price , baths , beds WHERE city=' SACRAMENTO';

3) Select placeid , price , baths WHERE beds='3';

4) Select placeid , price WHERE baths>2;

5) Select placeid WHERE piece<10000;

D. Mapping CBCQL to CQL

When we execute a CBCQL query, it map to CQL query internally and then run on Cassandra database. The result of query map back to the GUI of CBCQL. The mapping from CBCQL to CQL is shown in the Table 1 below:

TABLE I. MAPPING OF CBCQL TO CQL

S.No	CBCQL Query	CQL Query
1.	Create Context<ccontext_name>;	No mapping.
2.	Add Table<table_name>;	Select*from< Keyspace_name.ccontext_name> ;
3.	Select<column_name1>,<column_name2>,<column_name3>…….<;	Select<column_name1>,<column_name2>……from <Keyspace_name.ccontext_name>;
4.	Select<column_name1>,<column_name2>…..WHERE <condition>;	Select<column_name1>,<column_name2>…….from <Keyspace_name.ccontext_name> WHERE <condition>;
5.	Select<column_name1>,<column_name2>…..WHERE <condition1> and <condition2>…….;	Select<column_name1>,<column_name2>……from <Keyspace_name.ccontext_name >WHERE<condition1>and <condition2>…… ;
6.	Save context as<scontext_name>;	No direct mapping[1]
7.	Recall Context<scontext_name>;	Select* from<keyspsce_name.scontext_name>;
8.	Delete context <dcontext_name>;	Drop table<keyspace_name.dcontext_name>;

E. Architecture of CBCQL System

The architecture of CBCQL system is shown in Fig. 2. The front end provides CBCQL GUI for interaction. A user's query is expressed using the GUI. This query is passed to the CBCQL system. Within the system, it is actually received by the CBCQL query engine. CBCQL query engine stores some information on metadata store and accesses information from metadata store and passes the CBCQL query with this information to query mapper. Query mapper maps the CBCQL query to CQL query and passes to Cassandra database. There may be one, more than one, or no CQL query for a single CBCQL query. The mapped CQL query runs on Cassandra database. The result of the query is passed to the CBCQL engine through query mapper. The CBCQL query engine stores some information from result to metadata store and sends the result to the GUI. Now the user will see the result from GUI and query on it. The system is designed in such a way that the query of the user will run only in its Context. All these processes are hidden from the user. The user will only query through the GUI and will see his result in the table of the GUI.

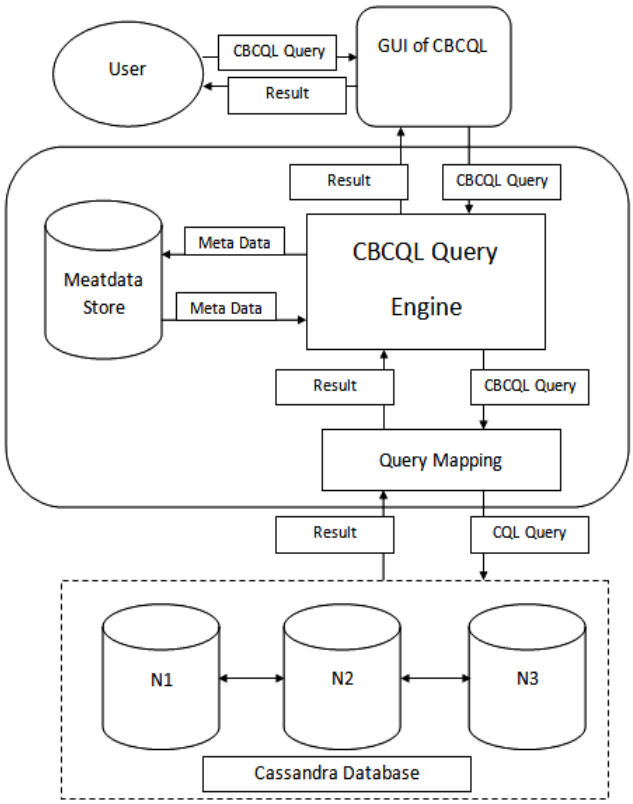

Fig.2. Architecture of CBCQL.

F. Data Set For Experiment

We have taken the dataset from [15] and did some modifications as per our need. In our dataset, there are fifteen attributes, and one thousand five hundred three rows, for describing the homes for sale. These attributes are price, baths, beds, city, area, other_services, parking_lot, placeID, Rpayment, sq__ft, state, street, type, url, and zip. A user can find a desired home by putting conditions on these attributes.

[1] Two queries are invoked for Save Context query. The first is Create table and second is Insert into table.

G. Query Execution And Results

We created a GUI by using Java Swing. In the first part of the GUI, there is a text area, for writing query followed by a dynamic table that is created for showing the results of the queries. Next, in the third part we have shown the query execution time and the messages. The number of rows, we get from the execution of a query is printed in the textArea field of our GUI. In the last part, there are three buttons. Execute button is for executing the query. The Clear button clears the text area that is used for the query. The Exit button closes the GUI. Queries are as follows:

1) Create Context abc ;

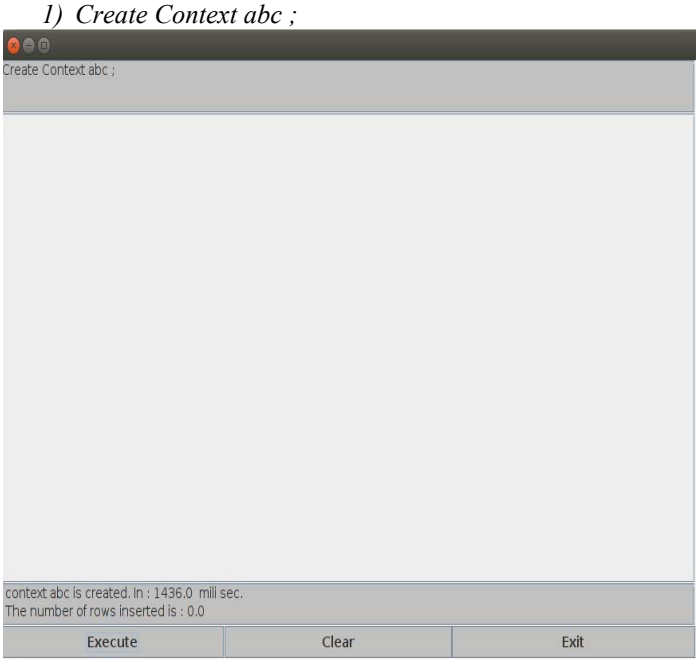

Fig.3. Output of query 1.

2) Add Table Data3 ;

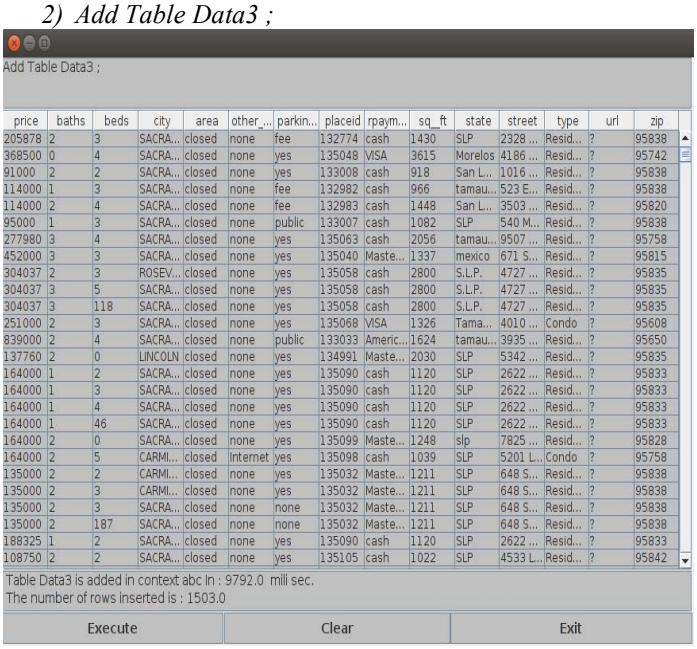

Fig. 4. Output of query 2.

This query has added the table in the context. Table Data3 is our data set. In our dataset, we have taken fifteen attributes to describe a home. The client will put conditions on these attributes to get the most suitable deal for him.

3) Select price, baths , beds , city , area , parking_lot , placeid , rpayment , sq__ft , type ;

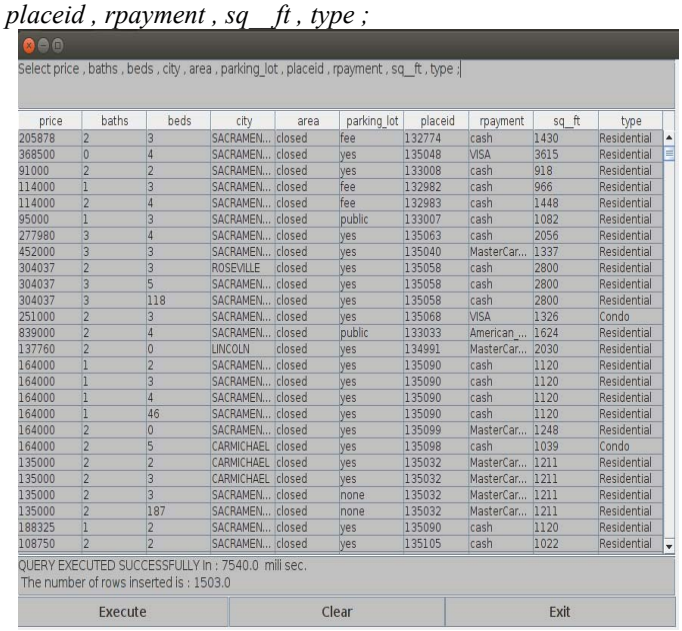

Fig.5. Output of query 3.

In this query, the client selects some relevant attributes to him and leaves the remaining.

*4) Select * WHERE type='Residential' ;*

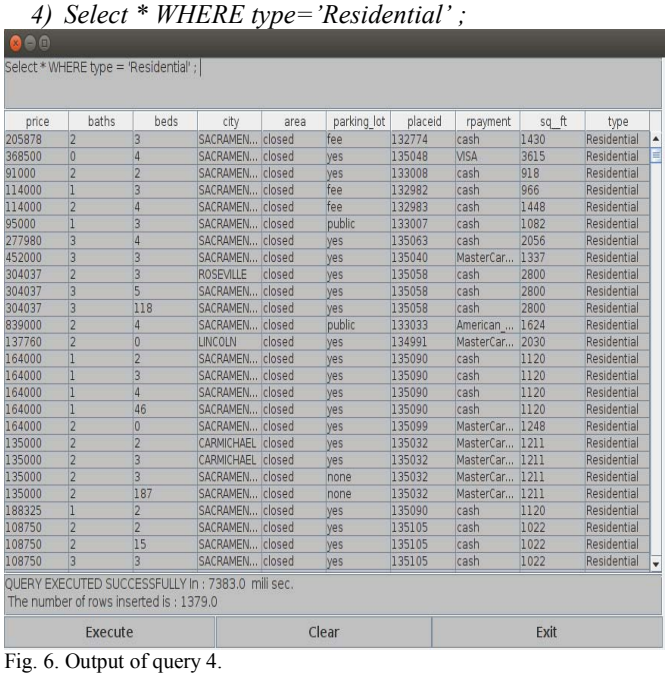

Fig. 6. Output of query 4.

This query selects all records for home of type residential. The result of this query has same fields as of query 3 as shown in Fig. 6.

5) Select price , baths , beds , city , area , parking_lot , placeid , rpayment , sq__ft WHERE beds = 3 and baths > 1 ;

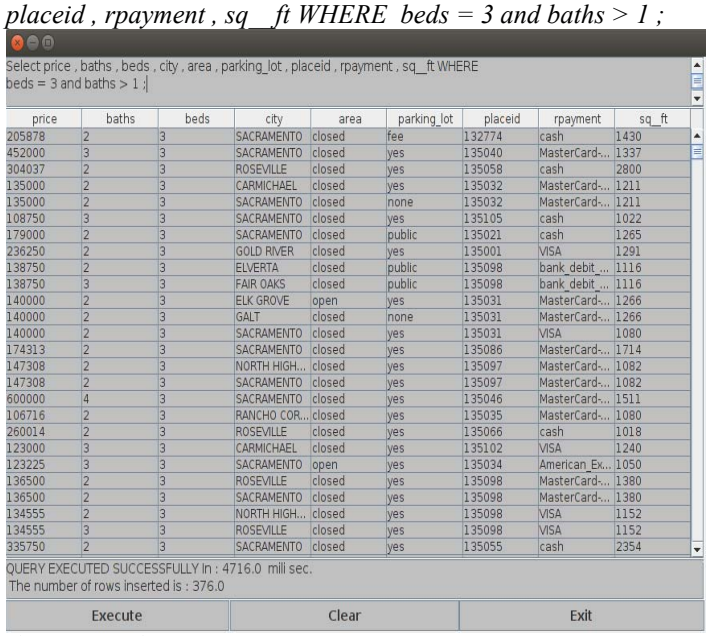

Fig.7. Output of query 5.

This query selects the record for a home with three bedrooms and at least two bathrooms.

6) Save context as pqr ;

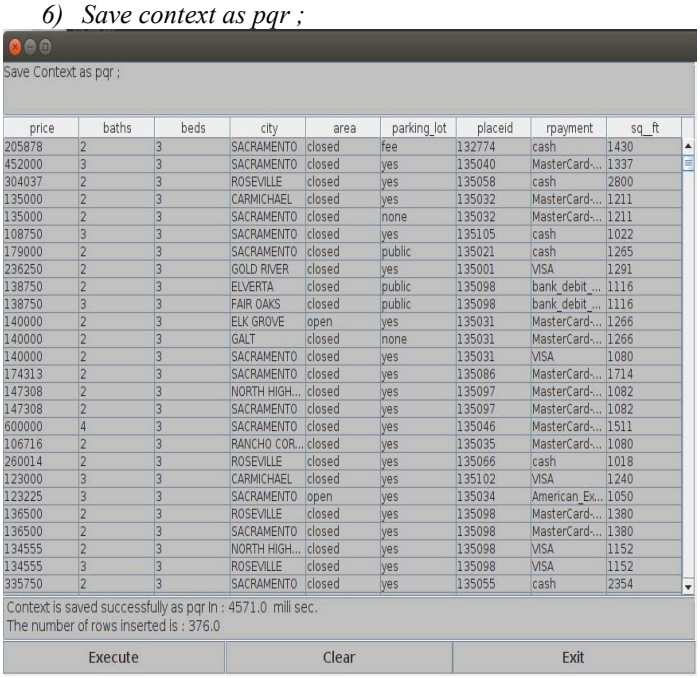

Fig. 8. Output of query 6.

Context is being saved in this query. The essential requirement of the client was a home for residential purpose with three bedrooms and at least two bathrooms. The result of this query is fulfilling all these conditions. Since, in context based querying, we cannot backtrack so it's better to save context and when we need, recall the context.

7)Select city , price , area , parking_lot , placeid , sq__ft WHERE parking_lot='yes' and city='SACRAMENTO' ;

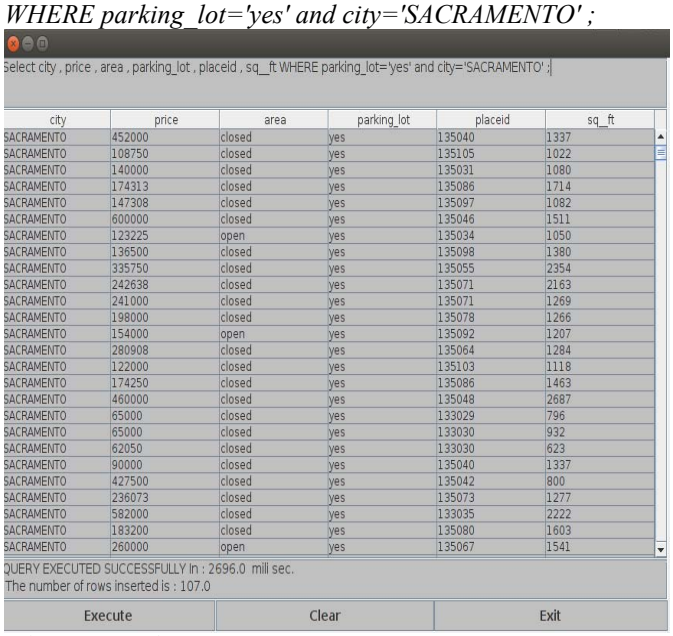

Fig. 9. Output of query 7.

The result of this query retrieves the detail of home with three bedrooms and at least two bathrooms for a resident in the Sacramento city.

8) Select area , price , placeid , sq__ft WHERE area='open' and price < 10000 ;

Fig. 10. Output of query 8.

This query displays the detail of all homes, fulfilling all the above conditions and in less than 10000, with an open area.

9) Recall Context pqr ;

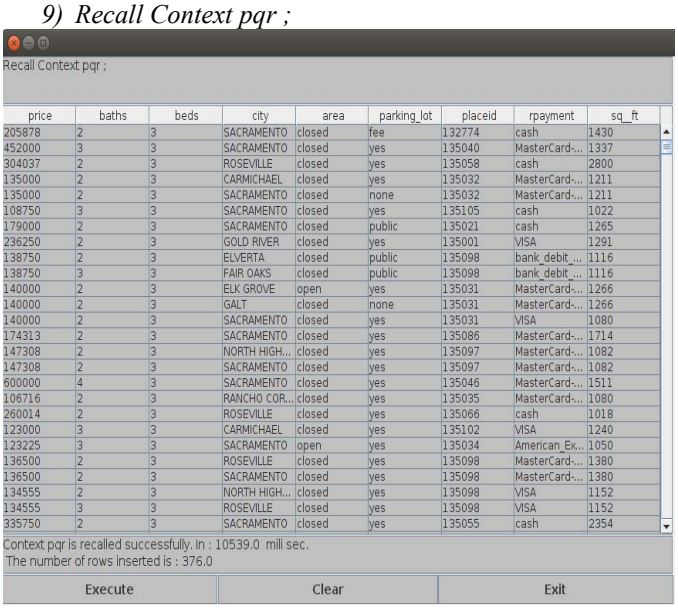

Fig. 11. Output of query 9.

If the client does not find a better deal, he can recall the context and search with some other conditions as in another city or different price. Now the subsequent query will run on the recalled context.

10) Delete Context pqr ;

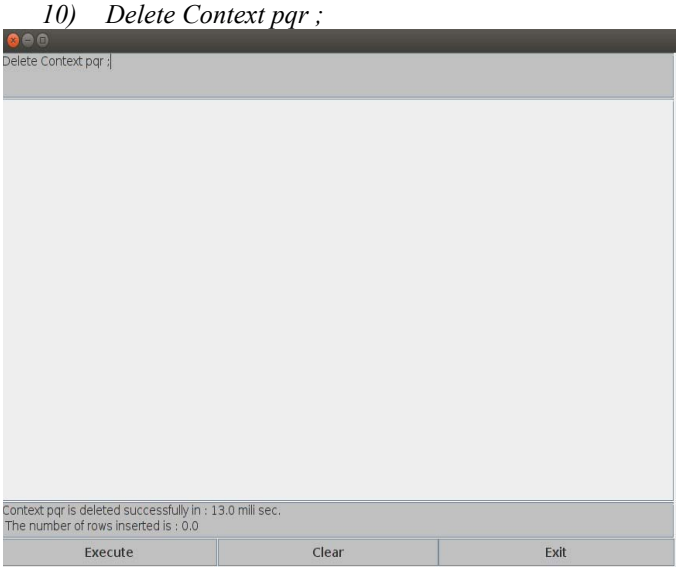

Fig. 12. Output of query 10.

This query deleted the context pqr. If pqr was our current context, then we cannot query further in the context before adding a table in the context, or recalling a context.

Here an example was demonstrated to show the working of our CBCQL system. We have shown the queries and screenshot of the results. The sequence of queries reflect the thought process of the user. In some queries there was only one condition to narrow down the result and in some other cases more than one. For example, we combined multiple conditions at some places because they are logically related and are considered together in our daily life, as number of bedrooms and bathrooms.

IV. CONCLUSION & FUTURE WORK

In this paper, we have proposed a new query language named as Context Based Cassandra Query Language. The purpose of this language is to provide a mechanism by which a user can ask a sequence of related queries. As a result, an easy way of querying with simpler queries and dictated by the thought process was provided. The user has to specify only SELECT and WHERE clause in the context. The context is designed in such a way that it fetches result from the context and updates the context with the result. Once a condition was expressed in a query within a context, there was no need to repeat the condition in the subsequent queries. We provided the facility of saving a context and recalling it, so backtracking is also easy for the user while querying. CBCQL has the same power as Cassandra with additional functionality because it is built over and above Cassandra. For using CBCQL we have provided a GUI which is very easy to use and simple to understand. CBCQL has a very simple and case insensitive syntax, so the possibility of errors is reduced. Cassandra is a new database and there is a major difference in terms of power and functionality in every new version of Cassandra. Even with low support of Java for Cassandra database the system was fully implemented with the desired results. In this paper, we implemented CBCQL for native data types of CQL. In future CBCQL can be implemented for collection data types and string data types (custom data types) of CQL.

ACKNOWLEDGMENT

The authors would like to thank CSIR for providing financial assistance. The authors would like express their sincere gratitude to all the staffs of SC & SS, JNU for their support behind this work.

REFERENCES

[1] S.K.Gajendran, “A survey on nosql databases ”, Technical report, University of Illinois, 2012.

[2] D. Zhang, “Inconsistencies in big data”, In Cognitive Informatics & Cognitive Computing (ICCI* CC) 12th IEEE International Conference on IEEE, pp. 61-67,2013.

[3] P.S. Duggal & S.Paul, “Big Data Analysis: Challenges and Solutions”, In International Conference on Cloud, Big Data and Trust, pp. 13-15,2013.

[4] A.B.M. Moniruzzaman & S.A. Hossain , “Nosql database: New era of databases for big data analytics-classification, characteristics and comparison”, arXiv preprint arXiv:1307.0191,2013.

[5] R.Cattell, “Scalable SQL and NoSQL data stores”, ACM Sigmod Record, 39(4), pp.12-27,2011.

[6] R. Agrawal, A. Ailamaki,P. A. Bernstein, E.A. Brewer, M.J. Carey, S. Chaudhuri, & G. Weikum, “The Claremont report on database research”. ACM Sigmod Record, 37(3), pp. 9-19,2008.

[7] C. Nance, T.Losser, R. Iype, & G.Harmon, “Nosql vs rdbms-why there is room for both”, In Proceedings of the Southern Association for Information Systems Conference, pp.111-116,2013.

[8] A.K. Zaki, “NoSQL Databases: New Millennium Database for Big Data, Big Users, Cloud Computing and Its Security Challenges”, International Journal of Research in Engineering and Technology (IJRET), 3(15), pp. 403-409,2014.

[9] G. Wang, & J. Tang (2012), "The nosql principles and basic application of cassandra model", In proceedings of Computer Science & Service System (CSSS) on IEEE, pp.1332-1335,2012.

[10] C. Strauch, U.L.S. Sites & W.Kriha, "NoSQL databases". Lecture Notes, Stuttgart Media University,2011.

[11] B.G. Tudorica, & C.Bucur, "A comparison between several NoSQL databases with comments and notes", In Roedunet International Conference (RoEduNet), 10th IEEE, pp.1-5,2011.

[12] N. Parimala, N. Prakash, B.L.N. Rao, & N. Bolloju, " A Query Facility to a network DBMS", The Computer Journal, 32(1), 55-62,1989.

[13] N. Parimala, "Explicit operation specification for component databases" , The Computer Journal, 45(2), pp. 202-212, 2002.

[14] N. Parimala & S. Bhawna, "Continuous multiple olap queries for data streams", International Journal of Cooperative Information Systems, 21(02), pp.141-164,2012.

[15] Sacramento_Homes_for_Sale,"Retrieved from Sacramento_Homes_for_Sale"December**,**2014,[Online]. Available:http:// samplecsvs.s3.amazonaws.com/Sacramento_Homes_for_Sale.csv.