CONTEXT BASED CASSANDRA QUERY LANGUAGE

Dissertation submitted to Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of

Master of Technology
In
Computer Science & Technology

Submitted
By
Shivendra Kumar Pandey

Under the Supervision
of
Prof. Parimala N.

School of Computer and Systems Sciences

Jawaharlal Nehru University

New Delhi -110067, India

July-2015

जवाहरलाल नेहरु विश्वविद्यालय

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
INDIA

Certificate

This is to certify that dissertation entitled **"Context Based Cassandra Query Language"** is being submitted by **Mr. Shivendra Kumar Pandey** to the **School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi-110067, India**, in the partial fulfillment of the requirements for the award of the degree of **"Master of Technology"** in **"Computer Science & Technology"**. This work is carried out by himself in the School of Computer & Systems Sciences under the supervision of **Prof. Parimala N.**. The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

Dean
Prof. C. P. Katti
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

Supervisor
Prof. Parimala N.
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

जवाहरलाल नेहरु विश्वविद्यालय
SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
INDIA

Declaration

I hereby declare that the dissertation work entitled **"Context Based Cassandra Query Language"** in partial fulfillment of the requirements for the award of degree of **"Master of Technology"** in **"Computer Science & Technology"** and submitted to the School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi-110067, India is the authentic record of my own work carried out during the time of Master of Technology under the supervision of **Prof. Parimala N.**. This dissertation comprises only my own work. This dissertation is less than 14,000 words in length, exclusive tables, figures and references. The matter personified in the dissertation has not been submitted for the award of any other degree or diploma.

Shivendra Kumar Pandey
M.Tech (2013-2015)
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi-110067

Dedicated to

My Loving Family, Friends and Teachers…

Acknowledgements

I would like to gratefully acknowledge the enthusiastic supervision of **Prof. Parimala N.** during this work. This work wouldn`t have been possible without her constant support, valuable suggestions and comments during my whole tenure of this dissertation work. I must surely say, she has given her best in providing me the infrastructure required, which led to the successful completion of my dissertation. I would take this opportunity to thank her once again for her esteemed support.

I would like to thank Council of Scientific and Industrial Research (CSIR) department for providing me research fellowship.

I wish to thank my colleague **Mr. Sudhakar** and my seniors **Mr. Ranjeet K. Ranjan, Mr. Hariom Sinha, Mrs. Rancha, Mr. Gaurav Kumar** for creating a homelike environment in our lab to keep the stress away and special thanks to **Mr. Amit Singh** for his constant support and suggestions. I would like to thank all my friends, for their valuable suggestions and support. They have a special place in my heart.

My parents deserve special mention for their inseparable support and prayers. My Father, **Mr. Vijay Kumar Pandey,** in the first place is the person who put the fundaments of my learning character, showing me the joy of intellectual pursuit ever since I was a child. My Mother, **Mrs. Rekha Devi,** is the one who sincerely raised me with her caring and gently love. I am also grateful to my uncle **Mr. Ashok Kumar Pandey,** for his love, affection and encouragement.

Finally, I would like to thank the whole faculty members of our department for clarifying my doubts throughout this work. Last but not least, thanks are due to the JNU administration for creating such a secular and healthy environment amongst the students.

Shivendra Kumar Pandey

Abstract

With the advancement of technology, the data generated by users are increasing exponentially. The unstructured nature of data with the speed it is generated, demands a system that has very high read/write throughput with failure tolerance property. This makes it difficult for traditional database to manage such data.

A new type of non-relational database has come into existence and is known as NoSQL database. 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 CQL so it has the same power as Cassandra, but provides additional functionality of querying on result of previous query. In this dissertation, CBCQL is discussed in detail.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

NoSQL	Not only SQL.
SQL	Structured Query Language
CQL	Cassandra Query Language
CBCQL	Context Based Cassandra Query Language
DBMS	Database Management System
RDBMS	Relational Database Management System
ACID	Atomicity, Consistency , Isolation, Durability
BASE	Basically Available, Soft State , Eventually Consistent
API	Application Programming Interface
CPU	Central Processing Unit
XML	Extensible Markup Language
UML	Unified Modeling Language
DDL	Data Definition Language
DML	Data Manipulation Language

Chapter 1

Introduction

The population of the world in 2014 was 7.2 Billion and out of them the internet users are 2.8 Billion, which is nearly 40 percent of the total population of the world and the number is increasing day by day (Mohamed, Altrafi & Ismail, 2014). With the development of technology and internet users, there is a need for a system that can manage data efficiently and provide high performance (Gajendran, 2012). Relational databases are facing many challenges, especially in scaling, concurrency and in providing write throughput (Zhang, 2013). 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 and unstructured data (Duggal & Paul 2013). NoSQL is an abbreviation of “Not only SQL” (Cattell, 2011; Moniruzzaman et al., 2013). They support more than SQL. These databases are very popular in companies for their cost, performance, and scalability.

NoSQL databases use many methods to store and retrieve data and, therefore, have as many types of databases as per methods used for data access. In our world, we see data as a large heterogeneous collection of structured and unstructured data (Agrawal et al., 2008). 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 (Nance et al., 2013). They support distributed data storage and distributed computing and therefore, do not have a single point of failure (Zaki et al., 2014).

In our work, among the NOSQL databases, we consider Cassandra.

1.1 Cassandra

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 BigTable's data model of Google for data storage and the data distribution concept of Amazon Dynamo (Wang & Tang, 2012). Cassandra Query Language is used to access Cassandra database.

1.2 Cassandra Query Language

Cassandra query language is the language for communicating with Cassandra database. We interact with Cassandra database with the help of CQL shell, known as cqlsh. cqlsh can be invoked from the command line of Windows or Linux. We can execute CQL command through cqlsh utility. Cassandra uses BigTable for data storage. The syntax of CQL and SQL are very similar, so understanding and working is easy for a developer with SQL background. The significant difference between CQL and SQL is that CQL does not support JOIN operations and sub queries.

1.3 Motivation

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. It will be very difficult for him to understand the table and query for the most suitable flat. Suppose he queries for a best suitable and economical flat, by putting conditions on one or many attributes that he can think of at the moment. Even now, what if he gets a few hundreds of rows with a few columns? It is definitely better than millions or billions of rows with thousands of columns as it was in database table, but still very difficult to get the information for the best flat. Even if the user is sure that the best suitable flat is in the result set, he cannot further query on it. Next time again, he has to query the entire table and there is no guarantee that he will get only a few rows in which he can decide easily.

To explain the above example we have taken a dataset from (Sacramento_Homes_for_Sale, 2014) and did some modifications as per our need. The dataset contains fifteen columns. The dataset is described in section 4.2 in detail.

Suppose a customer wants to buy a home. The sequence of his queries are:

Query1: Select price , baths , beds , city , area , parking_lot , placeid , rpayment , sq__ft , type ;

Query2: Select * WHERE type = ‘Residential ‘ ;

Query3: Select price , baths , beds , city , area , parking_lot , placeid , rpayment , sq__ft WHERE type = ‘Residential ‘ and beds = 3 ;

Query4: Select price , baths , beds , city , area , parking_lot , placeid , rpayment , sq__ft WHERE type = ‘Residential ‘ and beds = 3 and baths > 1 ;

Query5: Select city ,price , area , parking_lot , placeid , sq__ft WHERE type = ‘Residential ‘ and beds = 3 and baths > 1 and parking_lot='yes' ;

Query6: city ,price , area , parking_lot , placeid , sq__ft WHERE type = 'Residential ' and beds = 3 and baths > 1 and parking_lot='yes' and city='SACRAMENTO' ;

Query7: 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';

Query8: 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.

1.4 Our Contribution

To handle such problems we propose a query language, known as Context Based Cassandra Query Language "CBCQL". CBCQL is implemented over and above CQL. As a result, it supports CQL queries. In CBCQL a new concept, known as "Context" is introduced. Context is used to remember the result of the previous query and provides the facility of querying on the result of the previous query.

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. CBCQL is explained in chapter 3 in detail.

The contributions of this dissertation are:

- Definition of a Context.
- Definition of CBCQL for Context based querying.
- Facility to save and recall the context.
- A GUI facility for specifying the queries.

1.5 Dissertation Outline

The layout of the dissertation is as follows:

Chapter 2, Overview of methodologies: This chapter includes a survey on different methods and technology.

Chapter 3, Context Based Cassandra Query Language: In this chapter, the language CBCQL, proposed by us, is explained. A comparison with CQL is also given. It is compared to CQL and is discussed in detail.

Chapter 4, Experimental Setup and Results: Our experiment and its results are shown in this chapter.

Chapter 5, Conclusion. This chapter deals with a conclusion and future work.

Chapter 2

Overview of Methodologies

In this chapter an overview of the methodologies that are used in this dissertation is given. First, an overview of NoSQL databases is given. Cassandra, the NoSQL database used here is explained in more detail. The query language of Cassandra, CQL, which forms the basis of Context Based CQL is considered next.

2.1 NoSQL

Most of the companies were facing two major issues: low latency access to high volume data and continuous service availability in the unreliable environment (Duggal & Paul, 2013). So they developed a system that has very high read/write throughput and can work properly even some part of the system fails. This system is known as NoSQL database management system. NoSQL is not a database. This term is used to differentiate non-relational databases from relational databases. In NoSQL databases, data is stored in other than relational tables. Most of NoSQL databases have very high read/write throughput and have no single point of failure (Truong et al., 2009). NoSQL databases are compatible with structured, semi structured and even unstructured data (Moniruzzaman et al., 2013; Cattell, 2011).

The NoSQL databases have many advantages over a relational database. The figure below shows why people are going towards NoSQL database from relational databases:

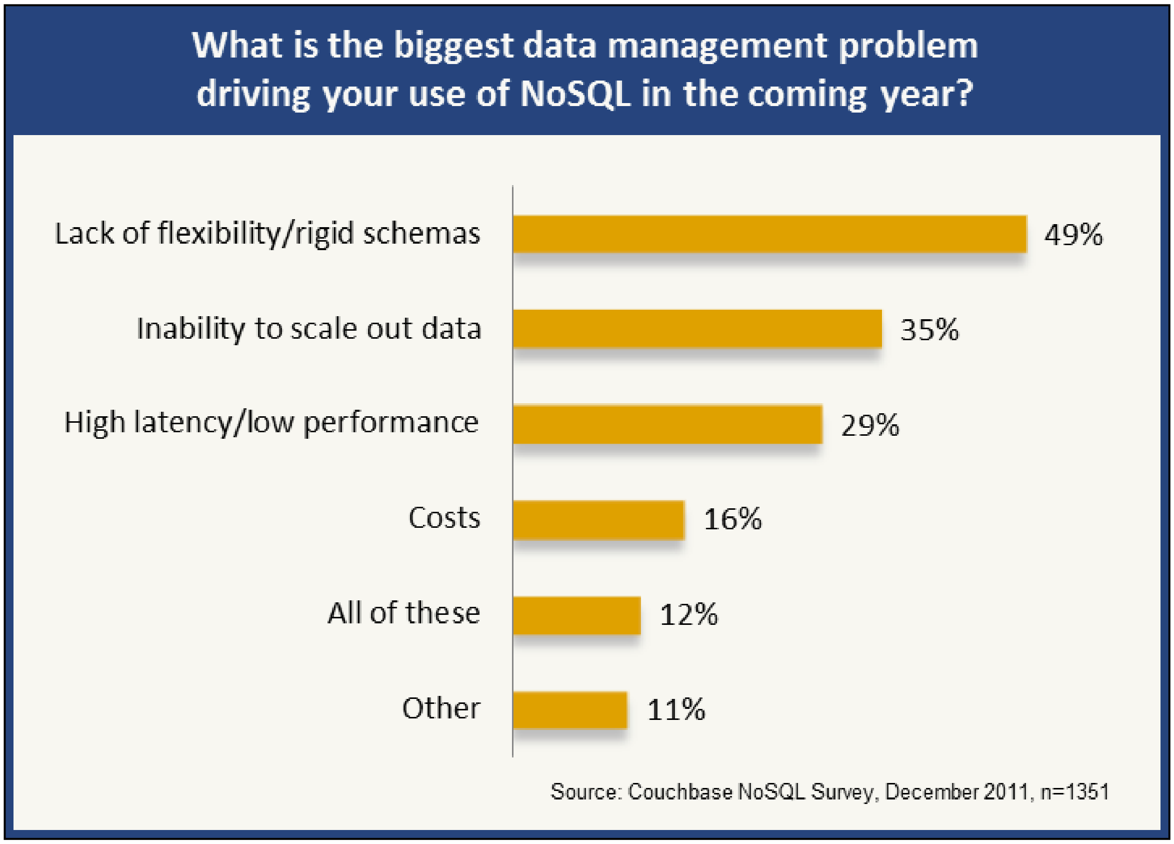


Figure 2.1: Key problems-driving to NoSQL databases (source: Moniruzzaman et al., 2013)

As we know, **RDBMS** follows **ACID** properties, **NoSQL** follows **BASE** properties (Nance, C. et al., 2013), and they are:

Basically Available: Uses replication to make more data available and uses sharding or partitioning of the data among many different servers, so failures become partial. As a result, we get a system that is always available even if some part of it fails (Strauch et al., 2011).

Soft state: Consistency is a hard requirement in relational databases. But NoSQL systems allow data to be inconsistent for some period of time. Soft state means the state of the system may change after some time even without any input. It happens because of eventual consistency.

Eventually consistent: NoSQL databases does not guarantee the consistency as in a relational database but they ensure that the system will be consistent at some future point in time.

There are various categories of NoSQL databases on the basis of data storage methodologies. NoSQL databases have been classified into three categories: Column-oriented database, Key-value store and Document oriented database [(Strauch et al., 2011). We explain NoSQL databases as per these three categories in brief next.

2.1.1 Column-oriented database

In column-oriented database, data is stored as sections of columns of data (Abadi et al., 2013), rather than as rows of data. A column is essentially a table with a single field. A column can store only one type of data. Data can be compressed in column-oriented store because of the similarities of adjacent records so in a column oriented stores data can be stored in less space (Abadi et al., 2006). A column family is the collection of these columns.

Fig 2.1 explains the difference between row-oriented store and column-oriented store of data.

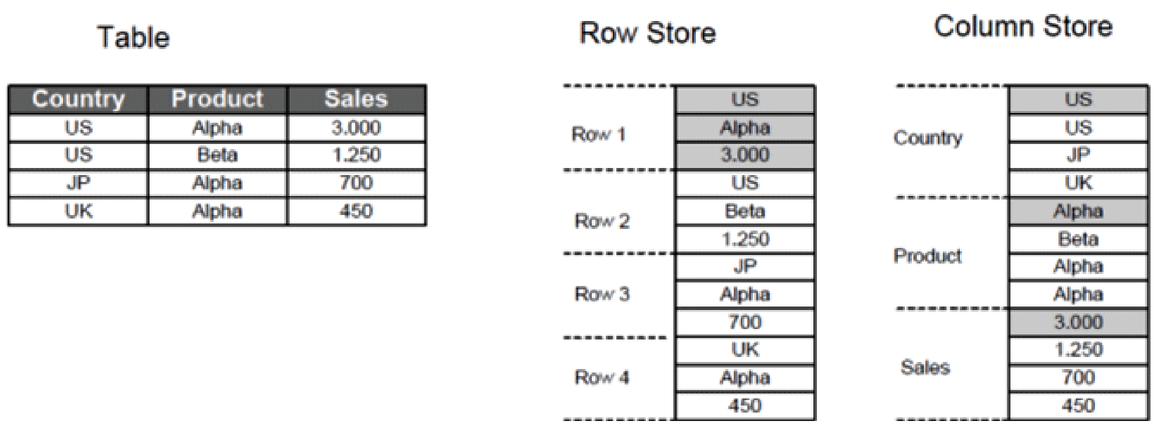


Figure 2.2: Row and Column-oriented store of table

Example of column-oriented databases are HBase, Cassandra, Accumulo, Amazon SimpleDB.

2.1.2 Key-value stores

These databases allow us to store key/value pairs in the database and subsequently read these values using the keys.

While storing or reading values by key, the system works extremely efficiently. So this method provides the efficient performance and low cost of implementation and scaling.

Disadvantages of the key-value store are that they do not ensure data integrity and there is no way to query from the content of value (McCreary et al., 2013). Applications control the data integrity in the key-value store.

The key-value store is illustrated in Fig 2.2.

Key	**Value**
1	placeid: 135110 price: 59222 beds: 3
2	placeid: 135110 price:68212 beds: 2
3	placeid:135104 price: 179580 beds:4

Figure 2.3: Key-value Store

Example of key-value stores are Amazon DynamoDB, Riak, Redis, LevelDB.

2.1.3 Document–oriented database

Document-oriented stores are designed to store, search, and manage document-oriented information.The document is similar to a row or a record in RDBMS but are more flexible than RDBMS. To store data in document-oriented store we encapsulate and encrypt them in several standard document formats like XML, JSON, BSON, PDF, etc. (Abramova & Bernardino, 2013).

Suppose we want to store the given information in a document:

Shivendra Pandey,
135 periyar hostel,

Jnu ,
New Delhi.

The document will be in pseudo XML format as

<contact>
<first_name> Shivendra </firstname>
<last_name> Pandey </last_name>
<room_no> 135 </room_no>
<hostel> periyar hostel </hostel>
<university> Jnu </university>
<city> New Delhi </city>
</contact>

Examples of document-oriented databases are MongoDB, Couchbase, CouchDB, RethinkDB.

In this dissertation, column oriented Cassandra database forms the basis. Cassandra database is explained next.

2.2 Cassandra Database

Cassandra is a distributed database management system with peer-to-peer architecture. Cassandra provides very high scalability and availability (Hewitt, 2010). There is no concept of master node in Cassandra so there is no single point of failure. To handle increasing I/O traffic, no ELT process or data movement is required. Cassandra automatically partitions and replicates the data when a node is added to the cluster. Because of data replication, Cassandra can work perfectly at some hardware failure. Cassandra provides tunable consistency. Cassandra comes under AP part of CAP theorem (According to CAP theorem, only two properties can be realized at a time from Consistency, Availability and Partition-tolerant). In Cassandra if we increase availability, the consistency will decrease and if we decrease availability, the consistency will increase (Lakshman & Malik, 2010).

2.2.1 Comparing the Cassandra Data Model to the Relational Database

The Cassandra data model is designed for distributed data on a very large scale. In a relational database, data is stored in tables, and the tables are typically related to each other. Data in a relational database, is usually normalized to reduce redundant entries, and tables are joined on common keys, satisfying a given query. But in Cassandra, tables (or column family) are independent (Hewitt, 2010). A table of Cassandra may be stored on one or more than one node and may have more than one copy. Cassandra is different from a relational database in many ways, but they have some similarities too. Some similarities and differences in terms of data storage are given below in tables.

Table 2.1: RDBMS vs. CASSANDRA

	RDBMS	**CASSANDRA**
1	Database	Keyspace
2	Table	Column Family
3	Primary Key	Row Key
4	Column Name	Column Name/Key
5	Column Value	Column Value

Data in RDBMS:

Id	Name	City	State
1	Ram	Allahabad	UP
2	Shyam	Basti	
3	Sohan	Muzaffarnagar	UP
4	Mohan		
5	Gita	Varanasi	
6	Sita		

Figure 2.4: Data in RDBMS table

Data in Cassandra Database:

Id:1	Name: Ram	City: Allahabad	State: UP
Id:2	Name: Shyam	City: Basti	
Id:3	Name: Sohan	City: Muzaffarnagar	State: UP
Id:4	Name: Mohan		
Id:5	Name: Gita	City: Varanasi	
Id:6	Name: Sita		

Figure 2.5: Data in Cassandra table

2.2.2 Column and Column Family in Cassandra Database

A Cassandra column contains three things: a name, a value and a timestamp (Hewitt, 2010). The value of a Cassandra column can be of a data type defined in Cassandra or may be sub column. The collection of columns is called column family in Cassandra and is very much similar to the Bigtable system (Lakshman & Malik, 2010). Column family is like a table of RDBMS. A row in Cassandra contains millions of columns, with their name, value and timestamp. A row key typically has automatically generated names (Universally Unique Identifier "UUID" or timestamp) (Hewitt, 2010). Figure 2.6 explains column and column family of Cassandra.

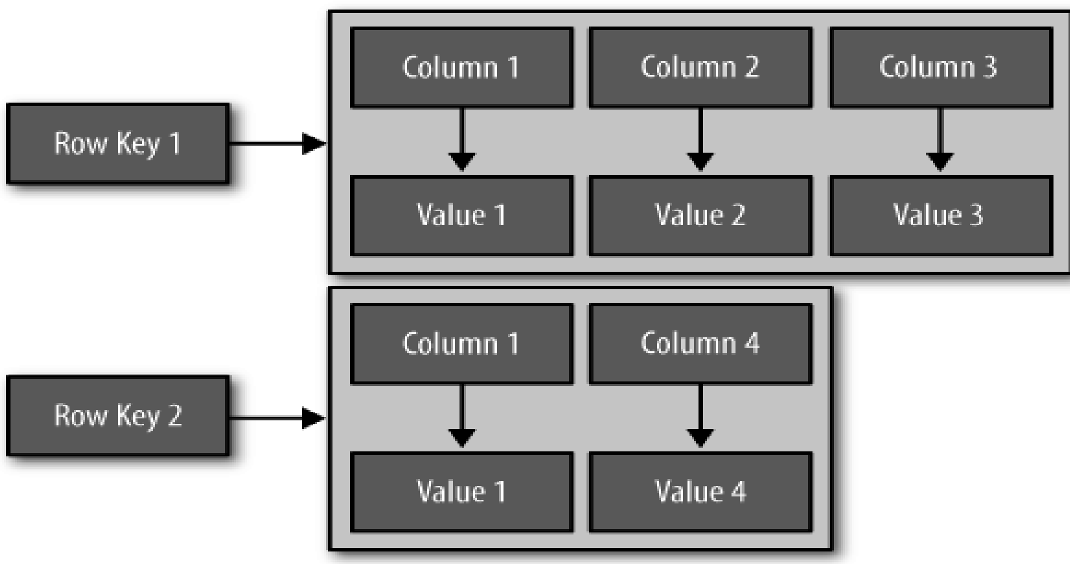


Figure 2.6 Column and column family (Hewitt, 2010)

2.2.3 Super column and super column family

If we store sub-columns instead of values in a column of Cassandra database, then such columns are called super column. We cannot store super columns, in a column of Cassandra database. The row key in a super column family is similar to the row key in the column family (Hewitt, 2010; Lakshman & Malik, 2009).

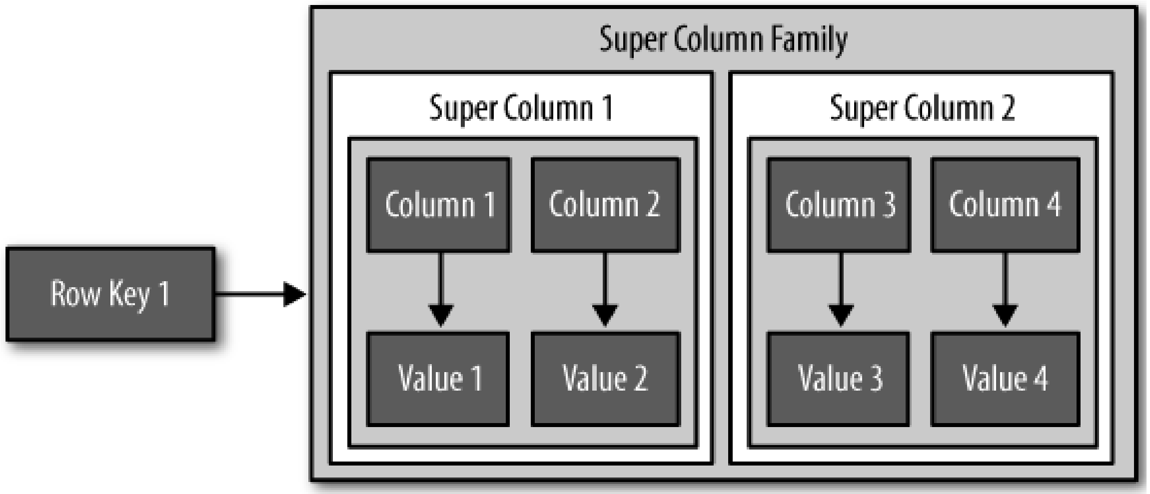


Figure 2.7: Super column and super column family (Hewitt, 2010)

2.3 Cassandra Query Language

Cassandra query language is the language we use for accessing Cassandra database. The syntax of the Cassandra query language is very much similar to the syntax of CQL, but some of them have different functionality. The version of CQL used in this work is v3.1.7 (as this was the latest version when we started our work, even though a later version of CQL, v3.2.0, has been subsequently released). We now, give an overview of CQL v3.1.7. CQL supports queries for all the three types of commands: DDL, DML and queries. The syntax of these types are given below.

DDL: CREATE KEYSPACE, USE, ALTER KEYSPACE, DROP KEYSPACE

CREATE TABLE, ALTER TABLE, DROP TABLE, TRUNCATE, CREATE INDEX, DROP INDEX.

DML: INSERT, UPDATE, DELETE, BATCH.

Queries: SELECT.

2.3.1 Data Types

CQL supports a rich set of data type to define data of columns in a column family. Data types of CQL can be categorized in three types:

- Native type.
- Collection type.
- String (used for custom types).

Some important native types are:

Ascii (ASCII character string), bigint (64 bit signed integer), blob (arbitrary type), Boolean (true or false), decimal (variable precision decimal), double (64 bit IEEE-754 floating point), float (32-bit IEEE-754 floating point), int (32-bit signed int), varchar (UTF8 encoded string), timestamp (used for conflict free timestamp).

Collection types are:

List (list<native-type>), Set (set<native-type>) and map (map<native-type, native-type>)

String: String is used for custom data types.

2.4 Summary

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 (Strauch et al., 2011). 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 (Tudorica, 2011)) main types of NoSQL databases (Strauch et al., 2011; Moniruzzaman et al., 2013).

Most of the NoSQL databases are non-relational, query-at-a-time and query-at-a-table. So 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. However, none of these uses Context within the framework of CQL. In (Parimala et al., 1989) context is defined for a network query language and in (Parimala, 2002) a component based query language includes the definition of a context. Stream based query language incorporated context in (Parimala & Bhawna, 2012).

Chapter 3

Context Based Cassandra Query Language

In this chapter, we explain Context Based Cassandra Query Language. Section 3.1 explains Context. Section 3.2 explains Querying in a Context. Section 3.3 explains CBCQL and its syntax. In section 3.4 the mapping of CBCQL to CQL is shown. Section 3.5 explains the Querying in the context with with an example on Sacramento_Homes_for_Sale dataset (Sacramento_Homes_for_Sale. (n.d.). Retrieved December 10, 2014). In the last section the architecture of CBCQL is discussed.

3.1 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.

3.2 Querying in a Context

We have proposed a query language CBCQL for querying in a context. Every query executes in the context. After the query is executed, the context is updated with the result of the query and this form the context for the subsequent query. This process goes on till the context is deleted, or the session is over.

It is possible that a user thinks that he may need the context in the future. In this case, he can save the context with a name and later recall it when he needs it.

3.3 Context Based Cassandra Query Language (CBCQL)

Context based Cassandra query language provides the facility of querying in a context. In CBCQL, a user writes a query in CBCQL language, in the textArea of GUI provided by CBCQL system and gets the result in the table of the GUI. The user also gets time of execution of the query and the number of rows in the result in the textarea of GUI. The state diagram of CBCQL is given in the figure 3.1. The CBCQL queries are mapped to CQL queries internally and run on Cassandra database. The result of the CBCQL query is mapped back to the table of CBCQL GUI. The architecture of CBCQL system is shown in figure 3.2.

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
- Add Table
- Select
- Save Context
- Recall Context
- Delete Context

In the syntax given below, the symbols “<” and “>” indicate that the user has to provide the information.

3.3.1 Create Context

This is the first query. A context with a given context name will be created. Initially the context is empty.

The syntax is:

```
Create Context<context name>;
```

3.3.2 Add Table

```
Add Table<table_name>;
```

Add table will add a table in the context and will print the table using the GUI. Now user is ready to query the table.

3.3.3 Select

There are three cases in select statement:

- ❖ Select without WHERE:

```
Select<column_name1>,<column_name2>,<column_name3>……….;
```

- ❖ Select with the single condition in WHERE:

```
Select<column_name1>,<column_name2>…..WHERE <condition>;
```

- ❖ Select with multiple conditions in WHERE:

```
Select<column_name1>,<column_name2>…..WHERE <condition1> and
<condition2>…….;
```

3.3.4 Save Context

In a context based query, if the user thinks that he may need the context in the future, he can save it with the command 'save context'. It may be recalled that the context is continuously updated. Thus, if the context in the intermediate sequence is deemed by the user as being useful later, then the user can save the context and use it later. Notice that, in the absence of this command, the sequence of queries which created this context have to be executed all over again.

The syntax of saving context is:

```
Save Context as<context_name>;
```

3.3.5 Recall Context

When a user wants to query on a result, stored by save context command, he can recall it by recall context command. The context will be updated by the context he recalled. The current context which existed before the command is executed is lost.

The syntax is:

```
Recall Context<context_name>;
```

3.3.6 Delete Context

When a client realizes that a context has no more use, he can delete it by delete context command.

The syntax is:

```
Delete context<context_name>;
```

It is important to know that after deleting the current context, a client can query further only after adding a new table or after recalling a saved context.

3.4 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.

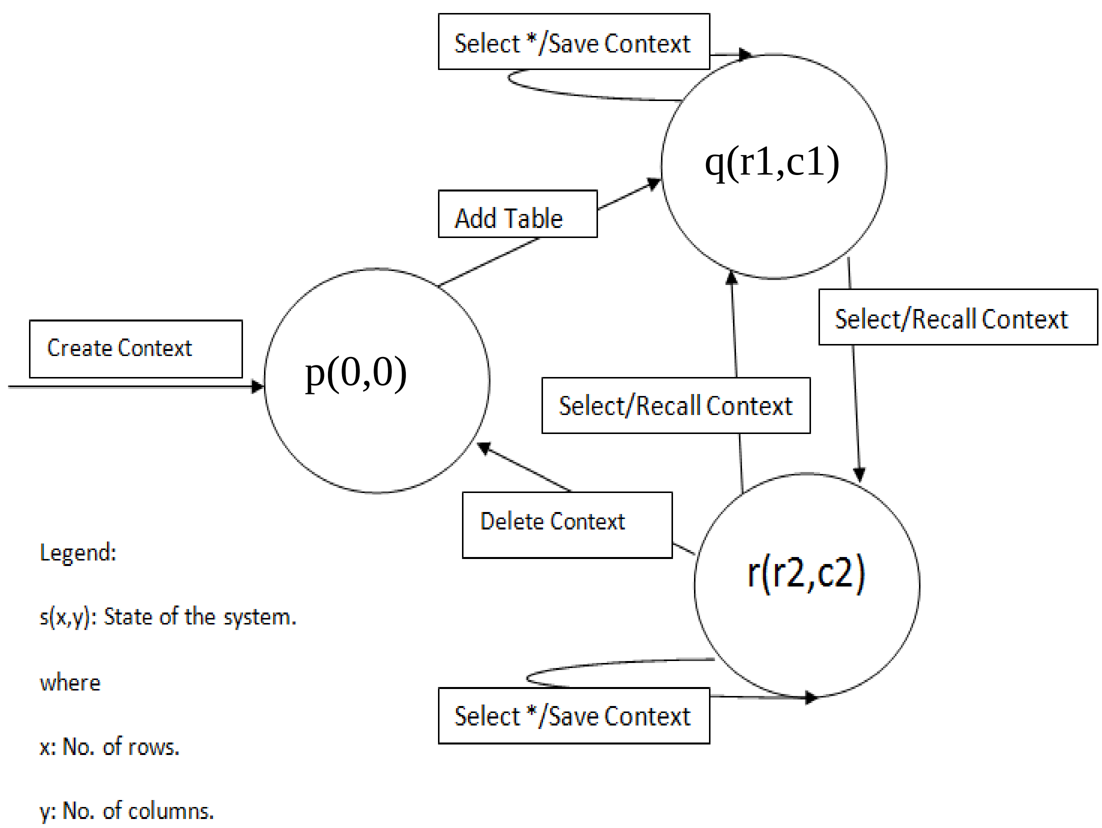


Figure 3.1: State diagram of CBCQL system

3.5 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.

Query1: Select placeid , price , baths , beds , city, WHERE type=’Residential’ ;

Query2: Select placeid , price , baths , beds WHERE city=’ SACRAMENTO’ ;

Query3: Select placeid , price , baths WHERE beds=’3’ ;

Query4: Select placeid , price WHERE baths>2 ;

Query5: select placeid WHERE piece<10000 ;

In the above example, in query1 customer selects details of a home for residential type.

After the execution of query1, the context updates its table with the data of the result of query1. In query2, we have no need to repeat the condition of query1 because our query will run in the context, and all the records of context are already satisfying the condition of query1. Same thing happens in query3, query4 and query5. We have no need to repeat the conditions of previous queries as we have to do while executing query without context in Cassandra database. It is clear from the example that in CBCQL, every time the query will run on a subset of whole table of database and it is easy to use and understand the system.

So for such queries using CBCQL is best.

3.6 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 below.

Table 3.1: 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>;

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

3.7 Architecture of CBCQL System

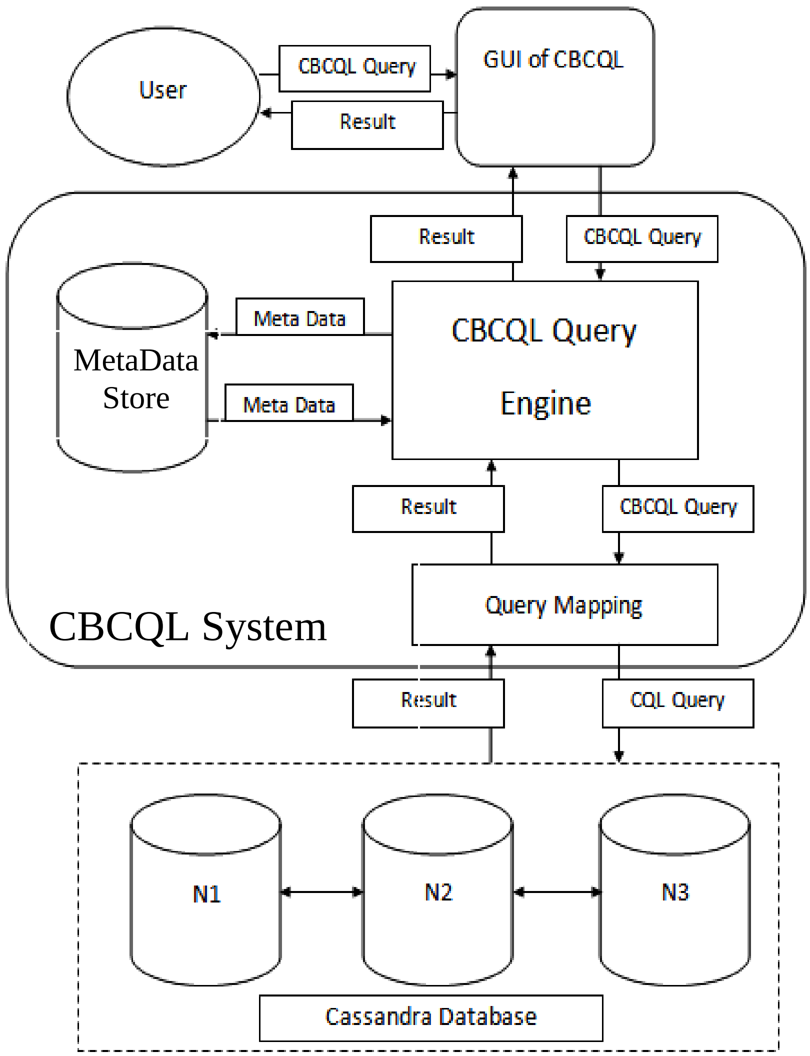


Figure 3.2: Architecture of CBCQL

The architecture of CBCQL system is shown in figure 3.2. 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.

Chapter 4

Experimental Setup and Results

In this section, we will describe the system on which we did the experiment. Section 4.1 describes the experimental setup and in section 4.2, the overview of the dataset is given. Section 4.3 contains queries and their results that we have executed and the last section 4.4 is a summary of our experimental setup and results.

4.1 Experimental Setup

Table 4.1: Experimental setup

S.No	Hardware/Software	Model/Version
1.	CPU	Intel(R) Xeon(R) 2.27GHz.
2.	RAM	16 GB DDR3.
3.	Operating System	Ubuntu 14.04 LTS 64 bit
4.	Cassandra	2.1.5
5.	cqlsh	5.0.1
6.	Eclipse	Luna 4.4.1
7.	Java	1.7.0_79

4.2 Data Set for Experiment

We have taken the dataset from (Sacramento_Homes_for_Sale. (n.d.). Retrieved December 10, 2014.) 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.

4.3 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.

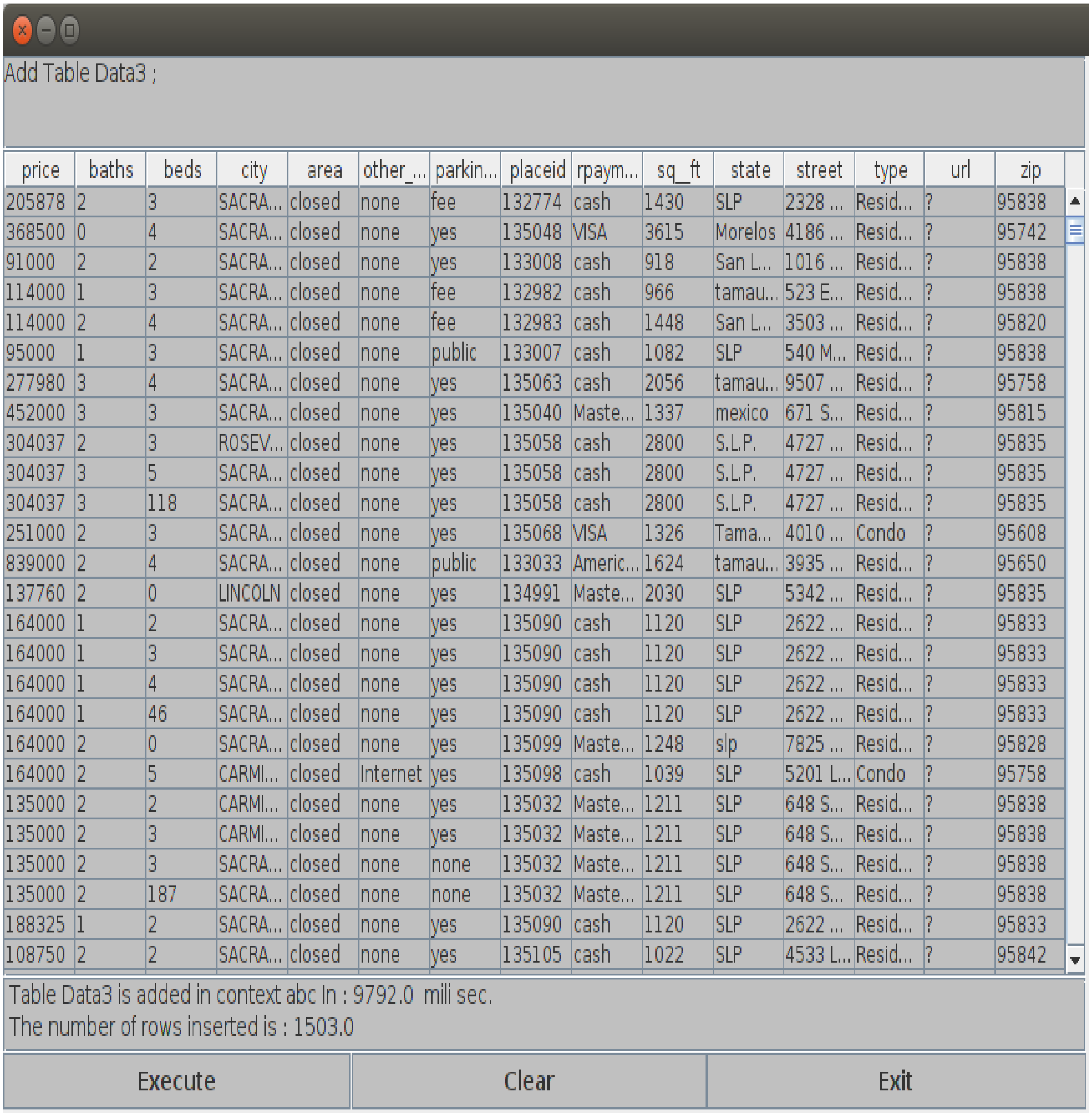
The example of chapter1 is executed in CBCQL using the implemented system. The queries and their results are shown below:

Query 1: Create Context abc ;

Figure 4.1: Output of "Create Context" query

This query creates the context "abc". Initially, the context is empty.

Query 2: Add Table Data3 ;


Add Table Data3 ;

price	baths	beds	city	area	other_...	parkin...	placeid	rpaym...	sq_ft	state	street	type	url	zip
205878	2	3	SACRA...	closed	none	fee	132774	cash	1430	SLP	2328 ...	Resid...	?	95838
368500	0	4	SACRA...	closed	none	yes	135048	VISA	3615	Morelos	4186 ...	Resid...	?	95742
91000	2	2	SACRA...	closed	none	yes	133008	cash	918	San L...	1016 ...	Resid...	?	95838
114000	1	3	SACRA...	closed	none	fee	132982	cash	966	tamau...	523 E...	Resid...	?	95838
114000	2	4	SACRA...	closed	none	fee	132983	cash	1448	San L...	3503 ...	Resid...	?	95820
95000	1	3	SACRA...	closed	none	public	133007	cash	1082	SLP	540 M...	Resid...	?	95838
277980	3	4	SACRA...	closed	none	yes	135063	cash	2056	tamau...	9507 ...	Resid...	?	95758
452000	3	3	SACRA...	closed	none	yes	135040	Maste...	1337	mexico	671 S...	Resid...	?	95815
304037	2	3	ROSEV...	closed	none	yes	135058	cash	2800	S.L.P.	4727 ...	Resid...	?	95835
304037	3	5	SACRA...	closed	none	yes	135058	cash	2800	S.L.P.	4727 ...	Resid...	?	95835
304037	3	118	SACRA...	closed	none	yes	135058	cash	2800	S.L.P.	4727 ...	Resid...	?	95835
251000	2	3	SACRA...	closed	none	yes	135068	VISA	1326	Tama...	4010 ...	Condo	?	95608
839000	2	4	SACRA...	closed	none	public	133033	Americ...	1624	tamau...	3935 ...	Resid...	?	95650
137760	2	0	LINCOLN	closed	none	yes	134991	Maste...	2030	SLP	5342 ...	Resid...	?	95835
164000	1	2	SACRA...	closed	none	yes	135090	cash	1120	SLP	2622 ...	Resid...	?	95833
164000	1	3	SACRA...	closed	none	yes	135090	cash	1120	SLP	2622 ...	Resid...	?	95833
164000	1	4	SACRA...	closed	none	yes	135090	cash	1120	SLP	2622 ...	Resid...	?	95833
164000	1	46	SACRA...	closed	none	yes	135090	cash	1120	SLP	2622 ...	Resid...	?	95833
164000	2	0	SACRA...	closed	none	yes	135099	Maste...	1248	slp	7825 ...	Resid...	?	95828
164000	2	5	CARMI...	closed	Internet	yes	135098	cash	1039	SLP	5201 L...	Condo	?	95758
135000	2	2	CARMI...	closed	none	yes	135032	Maste...	1211	SLP	648 S...	Resid...	?	95838
135000	2	3	CARMI...	closed	none	yes	135032	Maste...	1211	SLP	648 S...	Resid...	?	95838
135000	2	3	SACRA...	closed	none	none	135032	Maste...	1211	SLP	648 S...	Resid...	?	95838
135000	2	187	SACRA...	closed	none	none	135032	Maste...	1211	SLP	648 S...	Resid...	?	95838
188325	1	2	SACRA...	closed	none	yes	135090	cash	1120	SLP	2622 ...	Resid...	?	95833
108750	2	2	SACRA...	closed	none	yes	135105	cash	1022	SLP	4533 L...	Resid...	?	95842

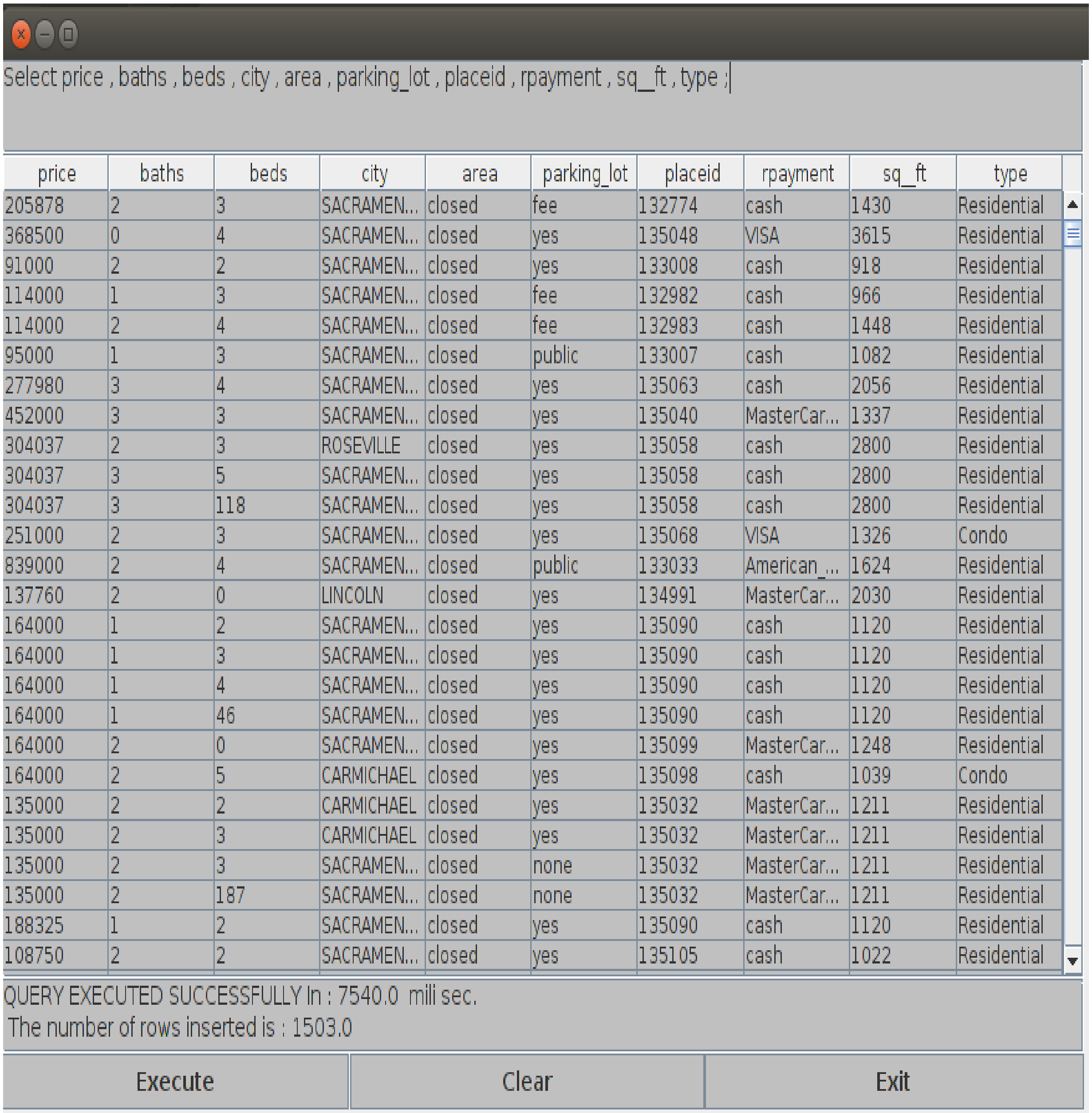
Table Data3 is added in context abc In : 9792.0 mili sec.
The number of rows inserted is : 1503.0

Figure 4.2: Output of “add table” query

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.

Query3: Select price, baths , beds , city , area , parking_lot , placeid , rpayment ,

sq__ft , type ;



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

price	baths	beds	city	area	parking_lot	placeid	rpayment	sq__ft	type
205878	2	3	SACRAMEN...	closed	fee	132774	cash	1430	Residential
368500	0	4	SACRAMEN...	closed	yes	135048	VISA	3615	Residential
91000	2	2	SACRAMEN...	closed	yes	133008	cash	918	Residential
114000	1	3	SACRAMEN...	closed	fee	132982	cash	966	Residential
114000	2	4	SACRAMEN...	closed	fee	132983	cash	1448	Residential
95000	1	3	SACRAMEN...	closed	public	133007	cash	1082	Residential
277980	3	4	SACRAMEN...	closed	yes	135063	cash	2056	Residential
452000	3	3	SACRAMEN...	closed	yes	135040	MasterCar...	1337	Residential
304037	2	3	ROSEVILLE	closed	yes	135058	cash	2800	Residential
304037	3	5	SACRAMEN...	closed	yes	135058	cash	2800	Residential
304037	3	118	SACRAMEN...	closed	yes	135058	cash	2800	Residential
251000	2	3	SACRAMEN...	closed	yes	135068	VISA	1326	Condo
839000	2	4	SACRAMEN...	closed	public	133033	American_...	1624	Residential
137760	2	0	LINCOLN	closed	yes	134991	MasterCar...	2030	Residential
164000	1	2	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	3	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	4	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	46	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	2	0	SACRAMEN...	closed	yes	135099	MasterCar...	1248	Residential
164000	2	5	CARMICHAEL	closed	yes	135098	cash	1039	Condo
135000	2	2	CARMICHAEL	closed	yes	135032	MasterCar...	1211	Residential
135000	2	3	CARMICHAEL	closed	yes	135032	MasterCar...	1211	Residential
135000	2	3	SACRAMEN...	closed	none	135032	MasterCar...	1211	Residential
135000	2	187	SACRAMEN...	closed	none	135032	MasterCar...	1211	Residential
188325	1	2	SACRAMEN...	closed	yes	135090	cash	1120	Residential
108750	2	2	SACRAMEN...	closed	yes	135105	cash	1022	Residential

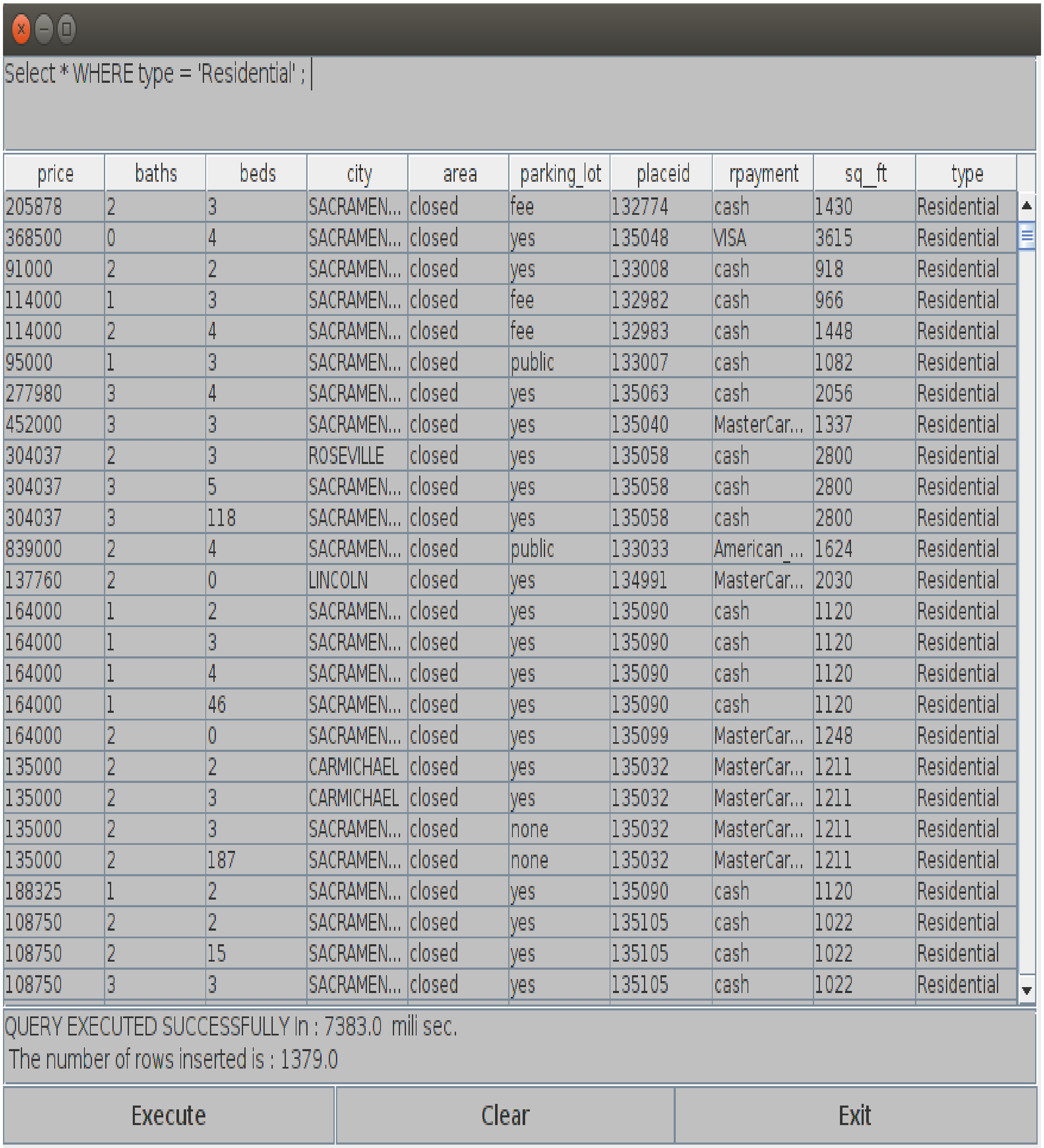
QUERY EXECUTED SUCCESSFULLY In : 7540.0 milli sec.
The number of rows inserted is : 1503.0

Execute | Clear | Exit

Figure 4.3: Output of query

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

Query4: Select * WHERE type='Residential' ;



Select * WHERE type = 'Residential' ;

price	baths	beds	city	area	parking_lot	placeid	rpayment	sq_ft	type
205878	2	3	SACRAMEN...	closed	fee	132774	cash	1430	Residential
368500	0	4	SACRAMEN...	closed	yes	135048	VISA	3615	Residential
91000	2	2	SACRAMEN...	closed	yes	133008	cash	918	Residential
114000	1	3	SACRAMEN...	closed	fee	132982	cash	966	Residential
114000	2	4	SACRAMEN...	closed	fee	132983	cash	1448	Residential
95000	1	3	SACRAMEN...	closed	public	133007	cash	1082	Residential
277980	3	4	SACRAMEN...	closed	yes	135063	cash	2056	Residential
452000	3	3	SACRAMEN...	closed	yes	135040	MasterCar...	1337	Residential
304037	2	3	ROSEVILLE	closed	yes	135058	cash	2800	Residential
304037	3	5	SACRAMEN...	closed	yes	135058	cash	2800	Residential
304037	3	118	SACRAMEN...	closed	yes	135058	cash	2800	Residential
839000	2	4	SACRAMEN...	closed	public	133033	American_...	1624	Residential
137760	2	0	LINCOLN	closed	yes	134991	MasterCar...	2030	Residential
164000	1	2	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	3	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	4	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	1	46	SACRAMEN...	closed	yes	135090	cash	1120	Residential
164000	2	0	SACRAMEN...	closed	yes	135099	MasterCar...	1248	Residential
135000	2	2	CARMICHAEL	closed	yes	135032	MasterCar...	1211	Residential
135000	2	3	CARMICHAEL	closed	yes	135032	MasterCar...	1211	Residential
135000	2	3	SACRAMEN...	closed	none	135032	MasterCar...	1211	Residential
135000	2	187	SACRAMEN...	closed	none	135032	MasterCar...	1211	Residential
188325	1	2	SACRAMEN...	closed	yes	135090	cash	1120	Residential
108750	2	2	SACRAMEN...	closed	yes	135105	cash	1022	Residential
108750	2	15	SACRAMEN...	closed	yes	135105	cash	1022	Residential
108750	3	3	SACRAMEN...	closed	yes	135105	cash	1022	Residential

QUERY EXECUTED SUCCESSFULLY In : 7383.0 mili sec.
The number of rows inserted is : 1379.0

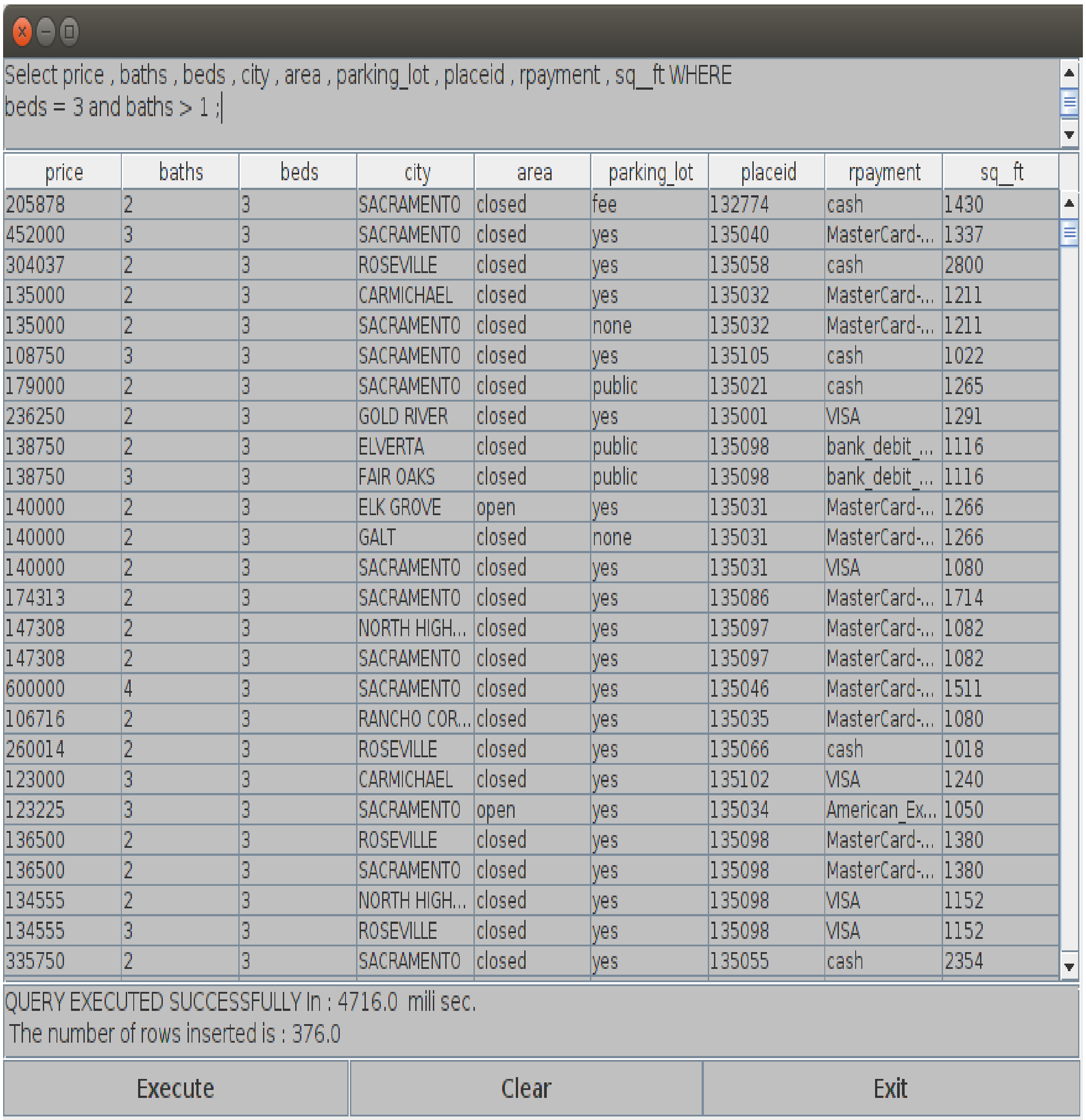
Execute | Clear | Exit

Figure 4.4: Output of query.

This query selects all records for home of type residential.

Query5: Select price , baths , beds , city , area , parking_lot , placeid , rpayment ,

sq__ft WHERE beds = 3 and baths > 1 ;

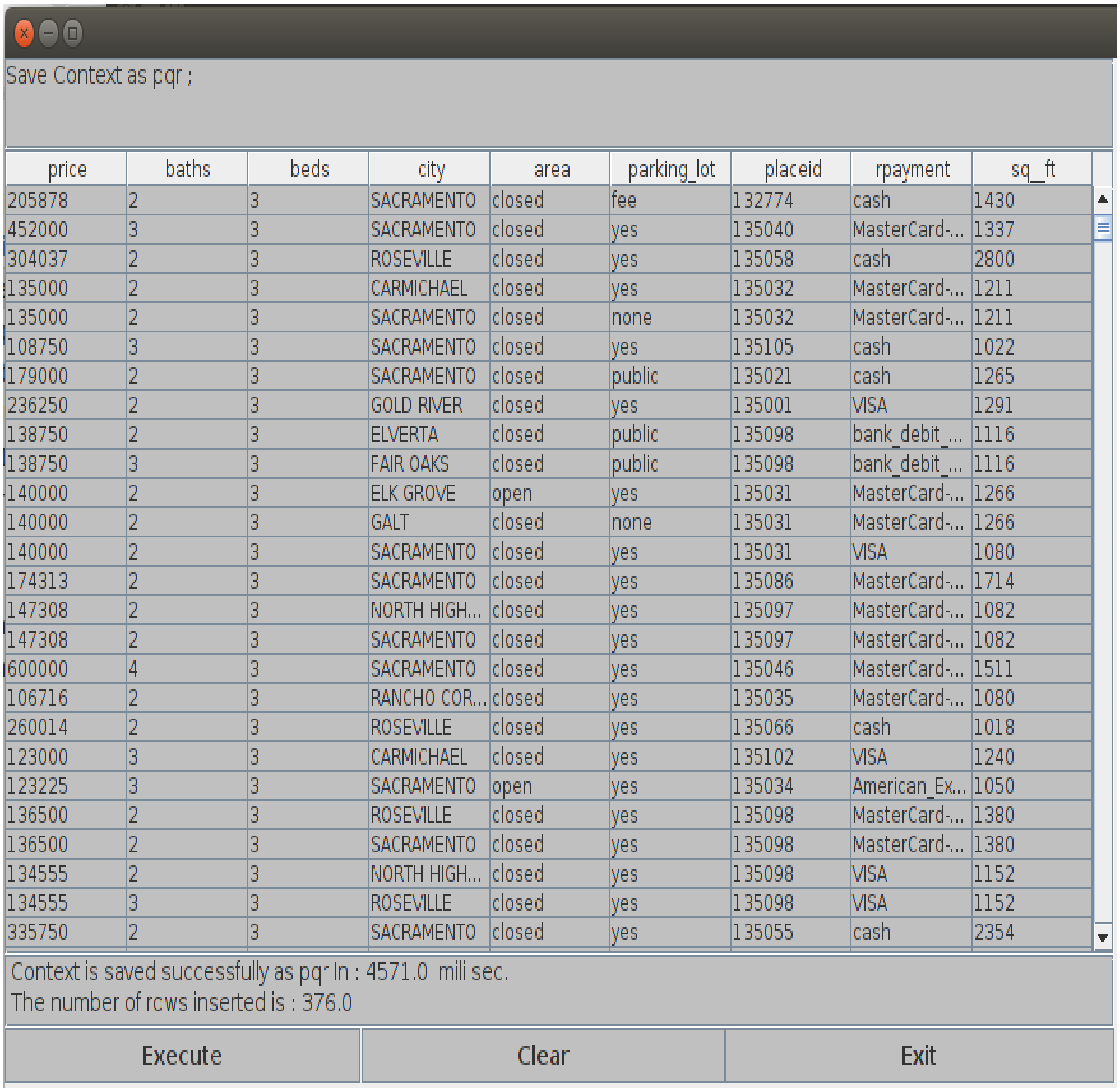


price	baths	beds	city	area	parking_lot	placeid	rpayment	sq_ft
205878	2	3	SACRAMENTO	closed	fee	132774	cash	1430
452000	3	3	SACRAMENTO	closed	yes	135040	MasterCard-...	1337
304037	2	3	ROSEVILLE	closed	yes	135058	cash	2800
135000	2	3	CARMICHAEL	closed	yes	135032	MasterCard-...	1211
135000	2	3	SACRAMENTO	closed	none	135032	MasterCard-...	1211
108750	3	3	SACRAMENTO	closed	yes	135105	cash	1022
179000	2	3	SACRAMENTO	closed	public	135021	cash	1265
236250	2	3	GOLD RIVER	closed	yes	135001	VISA	1291
138750	2	3	ELVERTA	closed	public	135098	bank_debit_...	1116
138750	3	3	FAIR OAKS	closed	public	135098	bank_debit_...	1116
140000	2	3	ELK GROVE	open	yes	135031	MasterCard-...	1266
140000	2	3	GALT	closed	none	135031	MasterCard-...	1266
140000	2	3	SACRAMENTO	closed	yes	135031	VISA	1080
174313	2	3	SACRAMENTO	closed	yes	135086	MasterCard-...	1714
147308	2	3	NORTH HIGH...	closed	yes	135097	MasterCard-...	1082
147308	2	3	SACRAMENTO	closed	yes	135097	MasterCard-...	1082
600000	4	3	SACRAMENTO	closed	yes	135046	MasterCard-...	1511
106716	2	3	RANCHO COR...	closed	yes	135035	MasterCard-...	1080
260014	2	3	ROSEVILLE	closed	yes	135066	cash	1018
123000	3	3	CARMICHAEL	closed	yes	135102	VISA	1240
123225	3	3	SACRAMENTO	open	yes	135034	American_Ex...	1050
136500	2	3	ROSEVILLE	closed	yes	135098	MasterCard-...	1380
136500	2	3	SACRAMENTO	closed	yes	135098	MasterCard-...	1380
134555	2	3	NORTH HIGH...	closed	yes	135098	VISA	1152
134555	3	3	ROSEVILLE	closed	yes	135098	VISA	1152
335750	2	3	SACRAMENTO	closed	yes	135055	cash	2354

Figure 4.5: Output of query

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

Query6: Save context as pqr ;



Save Context as pqr ;

price	baths	beds	city	area	parking_lot	placeid	rpayment	sq_ft
205878	2	3	SACRAMENTO	closed	fee	132774	cash	1430
452000	3	3	SACRAMENTO	closed	yes	135040	MasterCard-...	1337
304037	2	3	ROSEVILLE	closed	yes	135058	cash	2800
135000	2	3	CARMICHAEL	closed	yes	135032	MasterCard-...	1211
135000	2	3	SACRAMENTO	closed	none	135032	MasterCard-...	1211
108750	3	3	SACRAMENTO	closed	yes	135105	cash	1022
179000	2	3	SACRAMENTO	closed	public	135021	cash	1265
236250	2	3	GOLD RIVER	closed	yes	135001	VISA	1291
138750	2	3	ELVERTA	closed	public	135098	bank_debit_...	1116
138750	3	3	FAIR OAKS	closed	public	135098	bank_debit_...	1116
140000	2	3	ELK GROVE	open	yes	135031	MasterCard-...	1266
140000	2	3	GALT	closed	none	135031	MasterCard-...	1266
140000	2	3	SACRAMENTO	closed	yes	135031	VISA	1080
174313	2	3	SACRAMENTO	closed	yes	135086	MasterCard-...	1714
147308	2	3	NORTH HIGH...	closed	yes	135097	MasterCard-...	1082
147308	2	3	SACRAMENTO	closed	yes	135097	MasterCard-...	1082
600000	4	3	SACRAMENTO	closed	yes	135046	MasterCard-...	1511
106716	2	3	RANCHO COR...	closed	yes	135035	MasterCard-...	1080
260014	2	3	ROSEVILLE	closed	yes	135066	cash	1018
123000	3	3	CARMICHAEL	closed	yes	135102	VISA	1240
123225	3	3	SACRAMENTO	open	yes	135034	American_Ex...	1050
136500	2	3	ROSEVILLE	closed	yes	135098	MasterCard-...	1380
136500	2	3	SACRAMENTO	closed	yes	135098	MasterCard-...	1380
134555	2	3	NORTH HIGH...	closed	yes	135098	VISA	1152
134555	3	3	ROSEVILLE	closed	yes	135098	VISA	1152
335750	2	3	SACRAMENTO	closed	yes	135055	cash	2354

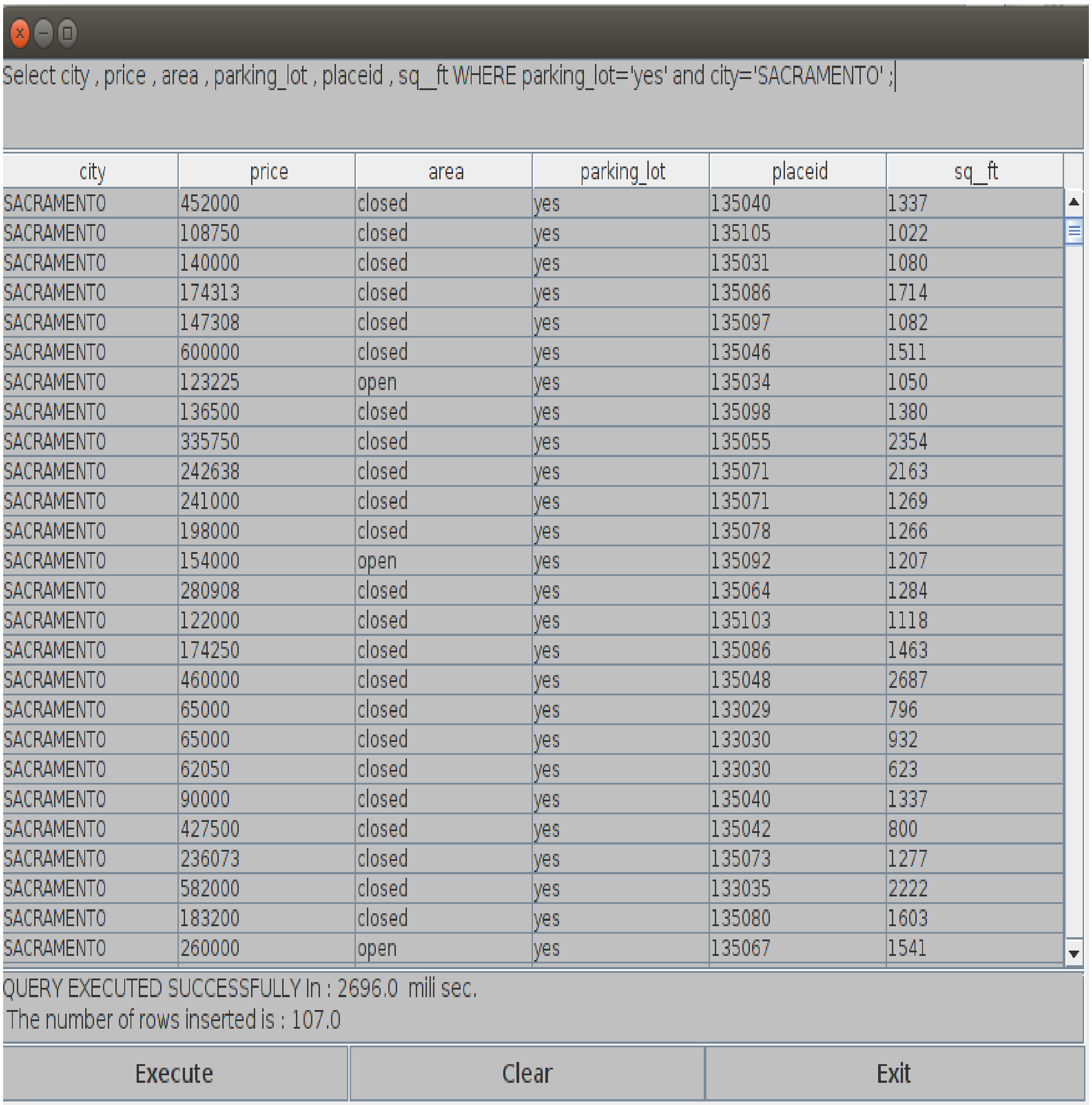
Context is saved successfully as pqr In : 4571.0 mili sec.
The number of rows inserted is : 376.0

Execute Clear Exit

Figure 4.6: Output of query

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.

Query7: Select city , price , area , parking_lot , placeid , sq__ft WHERE parking_lot='yes' and city='SACRAMENTO' ;



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

city	price	area	parking_lot	placeid	sq__ft
SACRAMENTO	452000	closed	yes	135040	1337
SACRAMENTO	108750	closed	yes	135105	1022
SACRAMENTO	140000	closed	yes	135031	1080
SACRAMENTO	174313	closed	yes	135086	1714
SACRAMENTO	147308	closed	yes	135097	1082
SACRAMENTO	600000	closed	yes	135046	1511
SACRAMENTO	123225	open	yes	135034	1050
SACRAMENTO	136500	closed	yes	135098	1380
SACRAMENTO	335750	closed	yes	135055	2354
SACRAMENTO	242638	closed	yes	135071	2163
SACRAMENTO	241000	closed	yes	135071	1269
SACRAMENTO	198000	closed	yes	135078	1266
SACRAMENTO	154000	open	yes	135092	1207
SACRAMENTO	280908	closed	yes	135064	1284
SACRAMENTO	122000	closed	yes	135103	1118
SACRAMENTO	174250	closed	yes	135086	1463
SACRAMENTO	460000	closed	yes	135048	2687
SACRAMENTO	65000	closed	yes	133029	796
SACRAMENTO	65000	closed	yes	133030	932
SACRAMENTO	62050	closed	yes	133030	623
SACRAMENTO	90000	closed	yes	135040	1337
SACRAMENTO	427500	closed	yes	135042	800
SACRAMENTO	236073	closed	yes	135073	1277
SACRAMENTO	582000	closed	yes	133035	2222
SACRAMENTO	183200	closed	yes	135080	1603
SACRAMENTO	260000	open	yes	135067	1541

QUERY EXECUTED SUCCESSFULLY In : 2696.0 mili sec.
The number of rows inserted is : 107.0

Figure 4.7: Output of query

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

Query8: Select area , price , placeid , sq__ft WHERE area='open' and price < 10000 ;

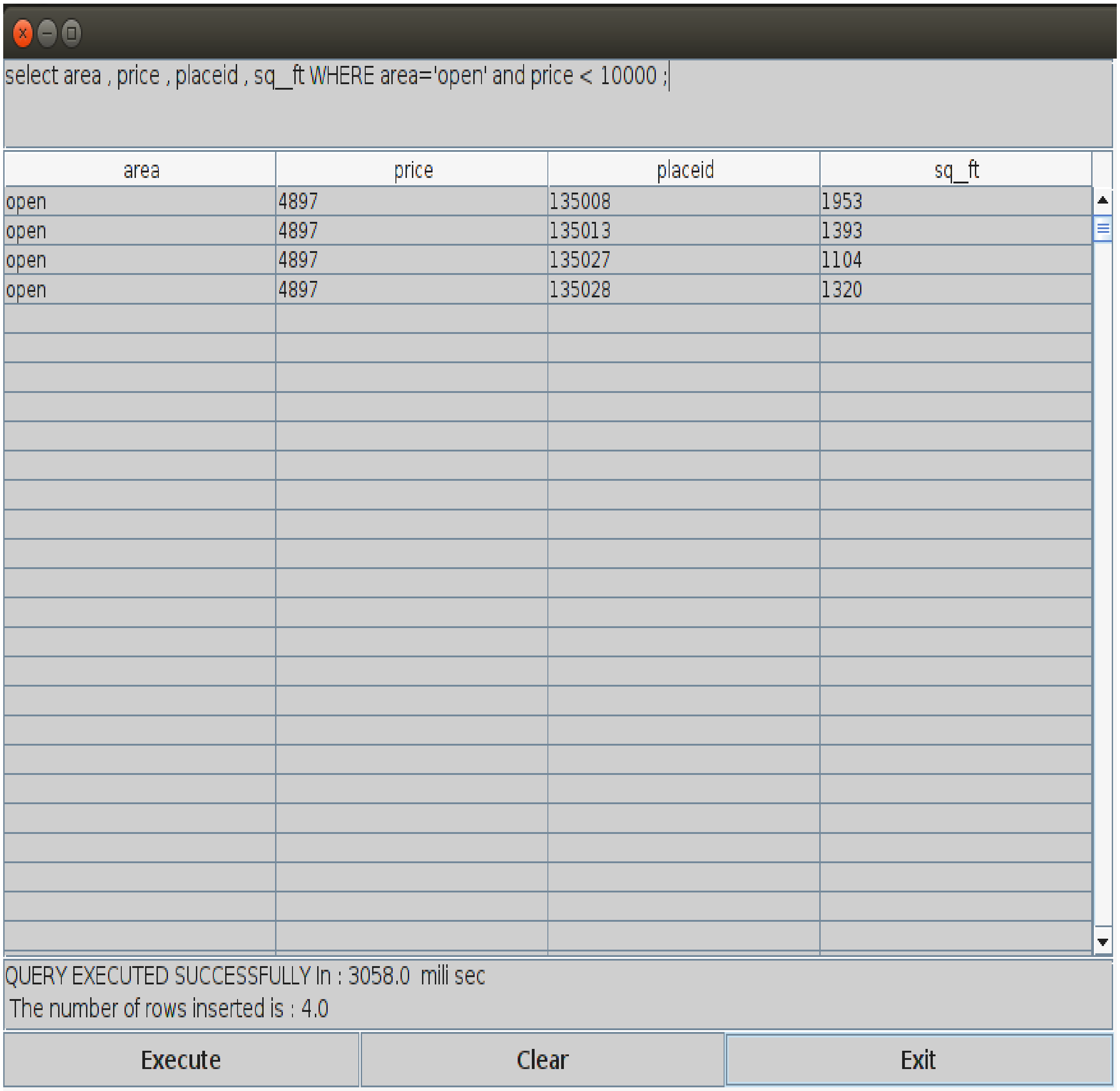
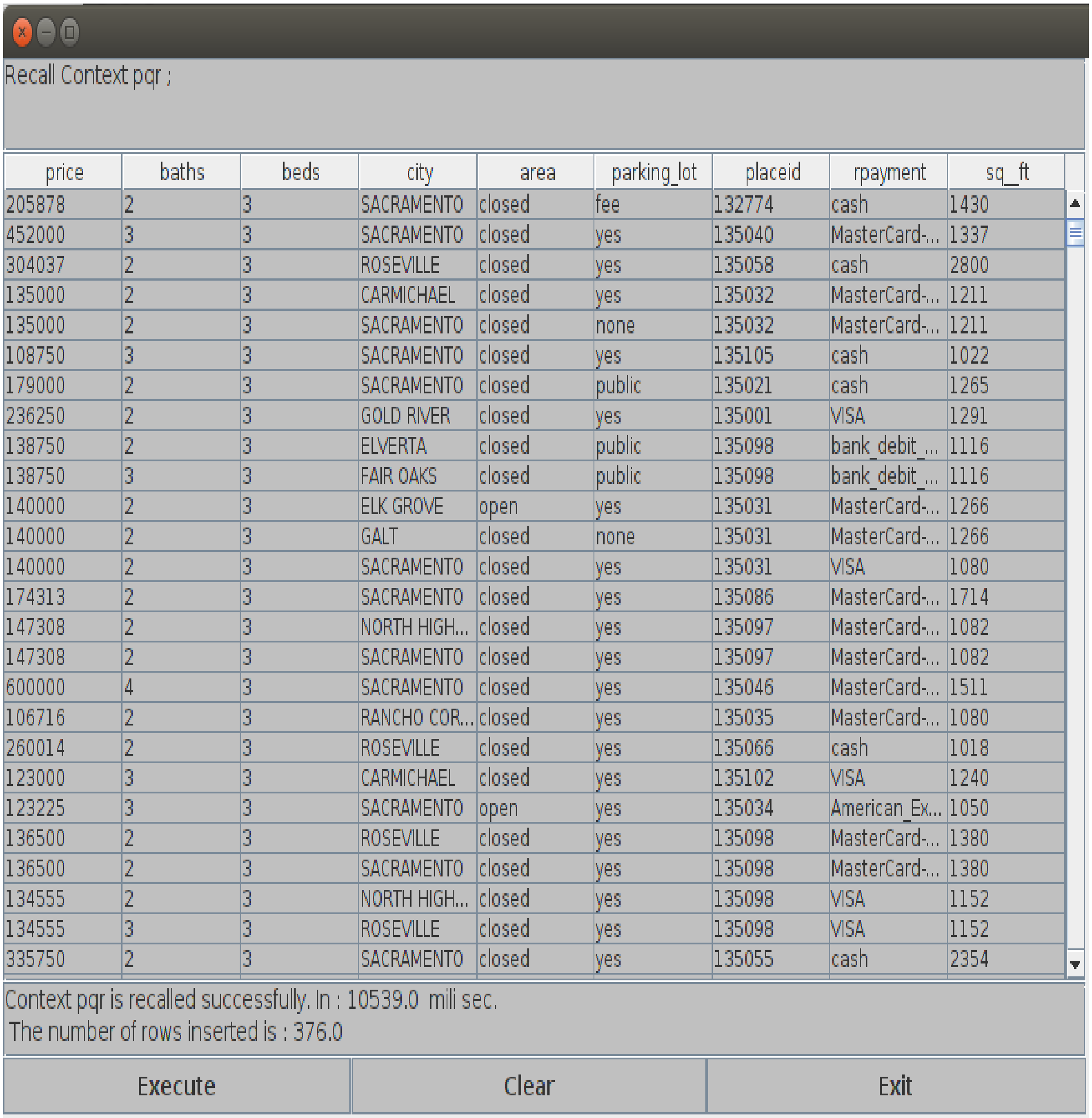


Figure 4.8: Output of query

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

Query9: Recall Context pqr ;



price	baths	beds	city	area	parking_lot	placeid	rpayment	sq_ft
205878	2	3	SACRAMENTO	closed	fee	132774	cash	1430
452000	3	3	SACRAMENTO	closed	yes	135040	MasterCard-...	1337
304037	2	3	ROSEVILLE	closed	yes	135058	cash	2800
135000	2	3	CARMICHAEL	closed	yes	135032	MasterCard-...	1211
135000	2	3	SACRAMENTO	closed	none	135032	MasterCard-...	1211
108750	3	3	SACRAMENTO	closed	yes	135105	cash	1022
179000	2	3	SACRAMENTO	closed	public	135021	cash	1265
236250	2	3	GOLD RIVER	closed	yes	135001	VISA	1291
138750	2	3	ELVERTA	closed	public	135098	bank_debit_...	1116
138750	3	3	FAIR OAKS	closed	public	135098	bank_debit_...	1116
140000	2	3	ELK GROVE	open	yes	135031	MasterCard-...	1266
140000	2	3	GALT	closed	none	135031	MasterCard-...	1266
140000	2	3	SACRAMENTO	closed	yes	135031	VISA	1080
174313	2	3	SACRAMENTO	closed	yes	135086	MasterCard-...	1714
147308	2	3	NORTH HIGH...	closed	yes	135097	MasterCard-...	1082
147308	2	3	SACRAMENTO	closed	yes	135097	MasterCard-...	1082
600000	4	3	SACRAMENTO	closed	yes	135046	MasterCard-...	1511
106716	2	3	RANCHO COR...	closed	yes	135035	MasterCard-...	1080
260014	2	3	ROSEVILLE	closed	yes	135066	cash	1018
123000	3	3	CARMICHAEL	closed	yes	135102	VISA	1240
123225	3	3	SACRAMENTO	open	yes	135034	American_Ex...	1050
136500	2	3	ROSEVILLE	closed	yes	135098	MasterCard-...	1380
136500	2	3	SACRAMENTO	closed	yes	135098	MasterCard-...	1380
134555	2	3	NORTH HIGH...	closed	yes	135098	VISA	1152
134555	3	3	ROSEVILLE	closed	yes	135098	VISA	1152
335750	2	3	SACRAMENTO	closed	yes	135055	cash	2354

Figure 4.9: Output of "Recall Context" query

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.

Query10: Delete Context pqr ;

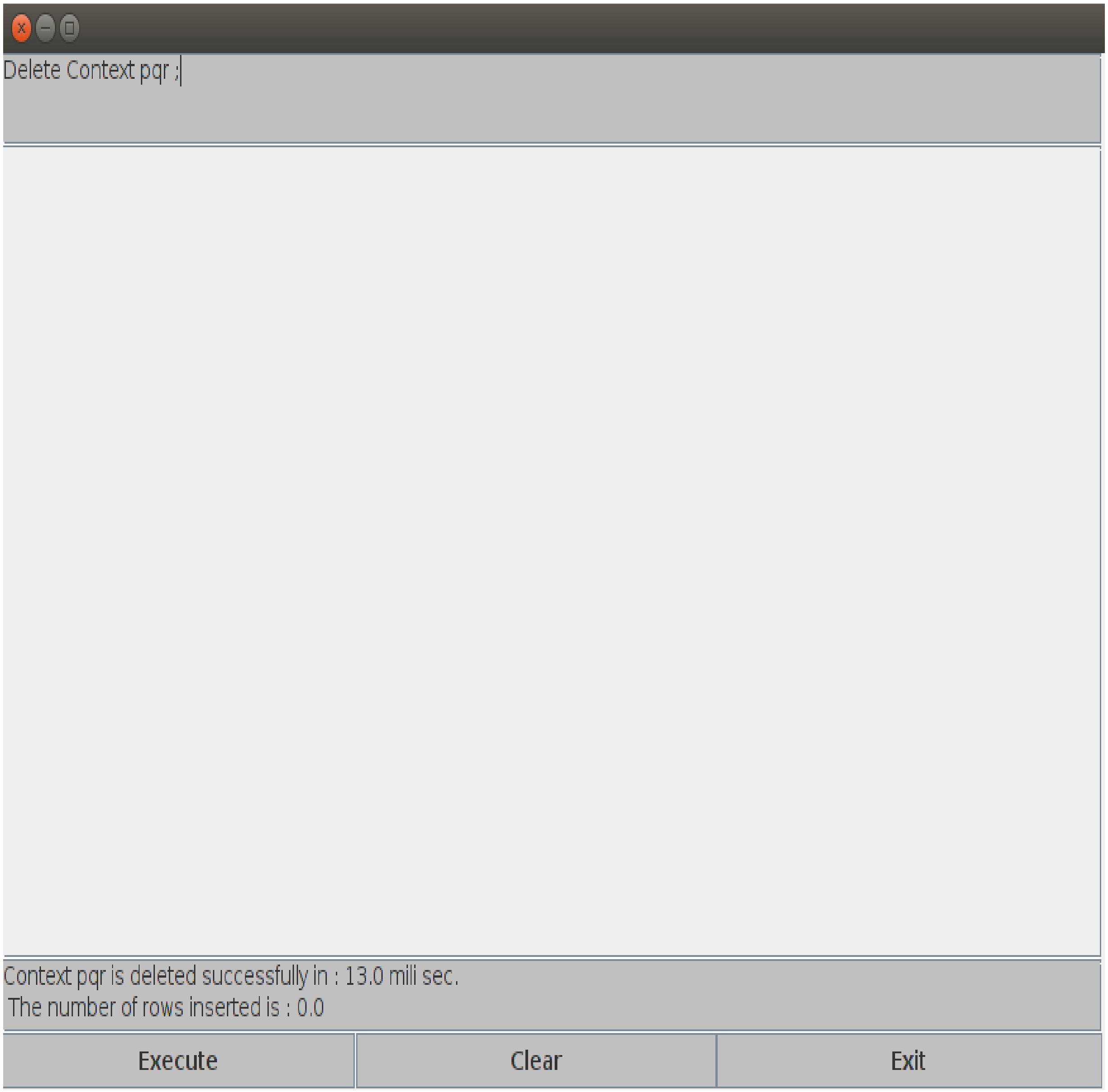


Figure 4.10: Output of “Delete Context” query

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.

4.4 Summary

In this chapter 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.

Chapter 5

Conclusion & Future Work

In this dissertation, 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 dissertation, 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.

References

Abadi, D., Boncz, P., Harizopoulos, S., Idreos, S., & Madden, S. (2013). *The design and implementation of modern column-oriented database systems*. Now.

Abramova, V., & Bernardino, J. (2013). NoSQL databases: MongoDB vs. Cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering. ACM*, 14-22.

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

Cattell, R. (2011). Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*, *39*(4), 12-27.

Duggal, P. S., & Paul, S. (2013). Big Data Analysis: Challenges and Solutions. In *International Conference on Cloud, Big Data and Trust*, 13-15.

Eckerstorfer, F. (2011). Performance of NoSQL Databases.

Gajendran, S. K. (2012). *A survey on nosql databases*. Technical report.

Hewitt, E. (2010). *Cassandra: the definitive guide*. " O'Reilly Media, Inc.".

Kumar, R., Parashar, B. B., Gupta, S., Sharma, Y., & Gupta, N. Apache Hadoop, NoSQL and NewSQL Solutions of Big Data. *International Journal of Advance Foundation and Research in Science & Engineering (IJAFRSE)*, *1*(6), 28-36.

Lakshman, A., & Malik, P. (2009). Cassandra: structured storage system on a p2p network. In *Proceedings of the 28th ACM symposium on Principles of distributed computing* ACM, 5-5.

Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35-40.

Leavitt, N. (2010). Will NoSQL databases live up to their promise?. *Computer*, *43*(2), 12-14.

McCreary, D., & Kelly, A. (2013). Making sense of NoSQL. *Greenwich, Conn.: Manning Publications*

Mohamed, M. A., Altrafi, O. G., & Ismail, M. O. (2014). Relational vs. NoSQL Databases: A Survey. *International Journal of Computer and Information Technology*.

Mohapatra, S., Rekha, K. S., & Mohanty, S. (2013). A Comparison of Four Popular Heuristics for Load Balancing of Virtual Machines in Cloud Computing. *International Journal of Computer Applications*, *68*(6), 33-38.

Moniruzzaman, A. B. M., & Hossain, S. A. (2013). Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. *arXiv preprint arXiv:1307.0191*.

Nance, C., Losser, T., Iype, R., & Harmon, G. (2013). Nosql vs rdbms-why there is room for both. In *Proceedings of the Southern Association for Information Systems Conference*, 111-116.

Parimala, N. (2002). Explicit operation specification for component databases. *The Computer Journal*, 45(2), 202-212.

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

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

Sacramento_Homes_for_Sale.(2014, December 10). Retrieved from Sacramento_Homes_for_Sale website: http:// samplecsvs.s3.amazonaws.com/Sacramento_Homes_for_Sale.csv

Strauch, C., Sites, U. L. S., & Kriha, W. (2011). NoSQL databases. *Lecture Notes, Stuttgart Media University*.

Truong, H. L., & Dustdar, S. (2009). A survey on context-aware web service systems. *International Journal of Web Information Systems*, *5*(1), 5-31.

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

Wang, G., & Tang, J. (2012). The nosql principles and basic application of cassandra model. In *Computer Science & Service System (CSSS), 2012 International Conference on* IEEE, 1332-1335.

Zaki, A. K. (2014). 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), 403-409.

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