A single access platform for different structural NoSQL and SQL databases

MSc Research Project
Cloud Computing

Abhilash Roy
x17109141

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet – 2017/2018
School of Computing

Student Name:	Abhilash Roy
Student ID:	x17109141
Programme:	Cloud Computing
Year:	2017
Module:	MSc Research Project
Lecturer:	Manuel Tova-Izquierdo
Submission Due Date:	13/08/2018
Project Title:	A single access platform for different structural NoSQL and SQL databases
Word Count:	4286

I hereby certify that the information contained in this (A single access platform for differentstructural NoSQL and SQL databases) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	16th September 2018

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A single access platform for different structural NoSQL and SQL databases

Abhilash Roy
x17109141
MSc Research Project in Cloud Computing

16th September 2018

Abstract

Today, many applications access both SQL and NoSQL databases as per data size, storage, structure and access requirements. Partial data sets reside on both SQL and NoSQL databases and becomes a tedious task to access databases at different times and also problem lies in concatenation. It is problematic for both end user and developers as query syntax and access interfaces are different for different databases. This paper aims at an approach to uniformly access data from both SQL and NoSQL systems. This approach called as Common Access Platform (CAP) allows users to query and interact with the database of their choice by abstracting depth details of various structural databases. The major goal of the platform is to help users and developers to store and access data in different SQL and NoSQL databases at ease and saving time by accessing data simultaneously from them on a click.

1 Introduction

Mostly relation databases like MySQL, Oracle etc are used as the back-end for web applications. Still, as SQL databases don't scale evenly and perform inadequately in distributed condition (Bansel et al.; 2016), application designers are thinking about other database alternatives. With the growth of application size and user, applications demand scalability. NoSQL databases like MongoDb, Cassandra, Neo4j etc provides better scalability and performance along with storing a voluminous amount of storage with worrying about structure. They prove to be helpful to store large data generated by social platforms, census, user information, stock markets etc.

Big data is booming along with cloud computing giving importance to storage provided by nonrelational systems. Considering the size of big data, NoSQL databases are of primary choice. RDBMS is powerful, easy and secure but lags when it comes in terms of handling huge amount of NoSQL data. As recent day applications consist of enormous unstructured data but relation systems become trivial in this case. Data without structure is useless if there is an absence of any rules or framework (Yafooz et al.; 2013). In order to deal with similar data, NoSQL systems were born. Nonrelation systems are indifferent to RDBMS in terms of data models, structure, access methods, drivers and interfaces. Even all types of SQL databases are not similar and vary as per their service provider. Considering the advantages and disadvantages of both SQL and NoSQL databases, both are essential at times to take care of fully structured, partial structured or structureless data.

So, Cloud vendors are supplying different Cloud databases as a Service in terms of both SQL and NoSQL databases to handle all sorts of requirements. In many scenarios, data is required to access simultaneously from SQL and NoSQL where both relational and no relational properties are required (For example, querying a user record where user basic info is stored in MySQL and work details is stored in MongoDB) In such condition, the output from each source database must be collected as one document or collection though they are stored on multiple heterogeneous databases.

The Initial idea of accessing different system by SQL was given by (Su and Swart; 2012) where SQL queries where used and processed to perform map-reduce operations.(Shirazi et al.; 2012) further illustrates the flexibility between graph and columnar types of databases. This formed the basis of idea that nuances between relational and non relational systems can be reduced by reducing the variant factors among them.

The motivation from many such scenarios urges to have a environment where relational and non relation databases can be evenly queried by any platform or system. Mostly, previous work focuses on the transformation of one model into another while CAP keeps the core back-end data models intact with quick and effective solution. CAP along with MySQL experiments with MongoDb (document-based), Cassandra (Columnar-based) and Neo4j (Graph-based) databases. These databases are loosely coupled in CAP and can be replaced by any other database or also any new database can be added. This application does not have to bother regarding the storage type, location, schema or memory demands.

The Research Problem: Can there be a uniform interface to access one or more different structural SQL and NoSQL databases?

2 Related Work

There have been many astounding advancements in the ever emerging field of database technologies, but no approach, that is helpful enough for running complex queries over composite data stores, has been put forth. One major reason for this is the non-existence of a standard data model for nosql databases. Due to this constraint, the developers have to migrate from one database to other and rework on the source coding and implementations. One more challenge that is faced while translating databases is that it is difficult and lengthy process. Though few attempts of research was already done in this area, no major breakthrough was profound enough to handle numerous databases and shed some light into data division techniques.

Knowledge about the structure of databases plays a vital role in evaluating various approaches in this field. Structure of four important type of nosql databases can be discussed as,

Key Value: It has a very plain structure of key-value pair.

Document type: This type also consists of data in form of key value pairs but stored in JSON/XML format. One more distinguishing factor from key value databases is that document data stores consist of a secondary index -(Han et al.; 2011).

Columnar data stores: They have a tabular structure but do not practice relational table

interrelations. Data is persisted in individual columns distinctly.
Graph data store: Data model is graph consisting of nodes as entities and association between nodes is represented by edges.
Relational data store: These databases are commonly known as SQL databases with the structure of rows and columns.

The main objectives of accessing the data from multiple variant databases are:
(i) To capture identical elements in source databases
(ii) To build a consistent database access functionality that conveniently handles the diversity of source systems.
Each data store will exhibit its own query language, APIs and access methods. More the diversity more the complexity for developers to achieve a uniform access platform. Each database access api/interfaces needs to be taken care in a disparate manner. These differences leads to wastage of resources and time. In order to simplify things in this field, many pieces of research have been done and they can majorly classified in following sections.

2.1 SQL interface above non relational databases

Mostly work done for this approach states use of sql to interact with one of the relational databases. An SQL engine was integrated on top of HBase by -(Vilaça et al.; 2013), which successfully processed various transactions that involved joins as HBase doesnt provide advanced query capabilities. This SQL engine depended on Apache Derby , which provides embedded drivers of JDBC as well, thereby facilitating indexes and joins as they are not supported in HBase. This technique works by transforming relational data models and secondary indexes to HBase data model. One primary advantage of using this approach is that all forms of congestion or load that occurs between the HBase and the query engine are lessened by indexing and filtering. It further enhances the result by dealing with complications related to latency and scalability by adding more number of nodes. Still, the drawback of this approach is that it is inadequate with respect to general solution of having SQL Engine for multiple variant nosql databases.

Work in (Tatemura et al.; 2012) states a framework called Partiqle, where a Key value data store has SQL interface above it. Partiqle deals with entity-group related transactions above key-value stores and it basically comprises of two major parts : (i)A component that reads input queries and generates key-value store respective query, (ii) A component that deals with caching all the key-value pairs. One more approach to consolidate the key-values and sql was undertaken by (Tahara et al.; 2014) through Sinew . Sinew is basically a SQL platform which stores documents encompassing key-value pairs into columns, both physically and virtually. By using binary serialization techniques, this methodology tries to query on partially structured data like JSON. It is efficient enough in random field access. If additional attributes are added to a query, then fetching virtual columns become a lengthy process.

2.2 Migration from sql to nosql

Research performed in this areas focusses on migrating SQL databases into one or more NoSQL databases. A new framework was implemented in (Rocha et al.; 2015) which comprises two components: the First one automatically transcribes the relational structure into a non-relational structure; The other module performs data mapping model ensuring effortless compatibility between MySQL to MonogoDB database. The main objective of this work is to persist the data in the NoSQL database and to build queries in SQL. This is accomplished by fetching the relational (MySQL) metadata using Java Database MetaData API. A new data model is generated using metadata and mapping the tables with respect to documents. MySQL Proxy, acting as a middleware software along with the Mapping class, generates indexes in the documents. But, this work restricts itself with only one NoSQL database i.e Mongodb. Further, the translation procedure banks completely on the theoretical model of the source database and may be noncompliant with some applications.

(Schreiner et al.; 2015) proposes a framework to migrate the relational schema to any of the key oriented NoSQL databases. This architecture transforms a relational model to a canonical model which acts as a middleware. This middleware model also have data in key-value form and it is mapped to document, key-value and columnar models using REST api's. But, the drawback is that it is unable to process all SQL transactions and also fails to deal with graph systems. This architecture has constraints of exibility and scalability for new databases.

The approach in (Lee and Zheng; 2015) develops a framework to translate relation data model HBase data model. For data model denormalization, DDI standards are considered where identical sql tables are categorized into single huge nosql table. Also, every row is recognized distinctively using a row key. Later, using SQL schema a linked list is generated which comprises of primary key and foreign keys. It has a major role in helping to analyze and relationships of tables in context of attributes. This analysis helps in development of nosql datasets. One short-come of this approach is that duplicates are not discarded from the table . Also, another constraint is flexibility for multiple nosql databases.

2.3 Single interface to multiple nosql databases

Framework in (Curé et al.; 2011) facilitates developers to query both SQL and NoSQL systems using SQL. The methodology has two components: translating SQL to BQL (Bridge Query Language), an intermediate stage. The other component helps to convert this BQL to respective NoSQL queries. A better approach was showcased by (Atzeni et al.; 2014) where a uniform interface is developed for Key value, columnar and document systems. A general data structure is created and then mapped to specific nonrelational data structures. (Atzeni et al.; 2012) deals with common access to HBase, Redis and MongoDb. The platform here works by using meta layer and various nosql database drivers. But, the author does not specify the project configurations and also merging of data collected from variant databases.

2.4 Uniform Access to both relational and non relation databases

As relational systems have schemas and non-relational system are schema-less, to fix this gap, (Liu et al.; 2016), proposed a json model of Oracle by adding extra features. This json is in a binary format making it lightweight and cater the performance difference between

SQL and NoSQL databases. Moreover, metadata is extracted at run time which helps to achieve this. However, the methodology transforms the primitive value types to one array for compatibility. This creates problems for apps which have already imported previous primitive values. (Liao et al.; 2016) by its work proposed a data adapter to query between SQL and NoSQL systems. Architecture depicts three variations in querying which differs in terms of filters and intermediate values. Along with query modes, their framework does following: 1) An SQL access interface for HBase and SQL supporting databases. 2)Maintaining synchronization, migrating MySQL system to HBase.

Work	**Approach**	**Database supported**	**Advantages**	**Limitations**
Scalable sql engine for nosql databases	SQL on top of NoSQL	HBase	SQL to access HBase	Only HBase
Partiqle	SQL Engine over Key Value database	Generic SQL interface	Supports complex SQL Query	Only Key value Database
Sinew	Storing key-value in relational columns	Key Value datbases	Abstraction layer	Only key value, Reliability on external systems
A framework for migrating relational datasets to nosql	Migration from MySQL to MongoDB	MongoDb	Efficient Migration	No other Database
Sqltokeynosql	Transforming relational to key value store	Document, Key Value and Columnar	DDL and DML instructions supported	No support to Graph and Relational
Sql-to-nosql schema denormalization and Migration	Migration of SQL model to HBase	Relatonal and HBase	Good for semistructured database	All databases not supported
Integration over nosql stores using access path based mappings	Query translation to variant NoSQL	Document, Key-Value, Columnar	Common access abstraction layer	Graph and Relational not supported
Uniform access to nosql systems	Common interface for NoSQL and SQL	Simple REST api to query	Document, Key-Value	No support to columnar and graph
The SOS platform	SQL interface and Meta layer to access	Document, Key-Value, Columnar	Common interface to access	No Join between results
Closing gap between sql and nosql	JSON schema approach	Relational, Document and Key value	Dynamic schema; so fast	Only Json schema database
Data adapter between sql and nosql database	SQL interface to NoSQL database	Relational and HBase	Common interface and data translation	Other NoSQL databases not supported
CAP Framework	Relational and Non Relational	Relational and Non Relational	Common access	complex query

Figure 1: Summarizing Approaches

3 Methodology

As previously discussed, the motive is to access and perform operations on different SQL and NoSQL systems without knowing in prior about them by single application. We will detail about the characteristics of such interface in this proposal. The major goal lies in how data is modelled and accessed and not on scalability or throughput performance of NoSQL systems. The architecture of CAP framework can be seen in Figure 2.

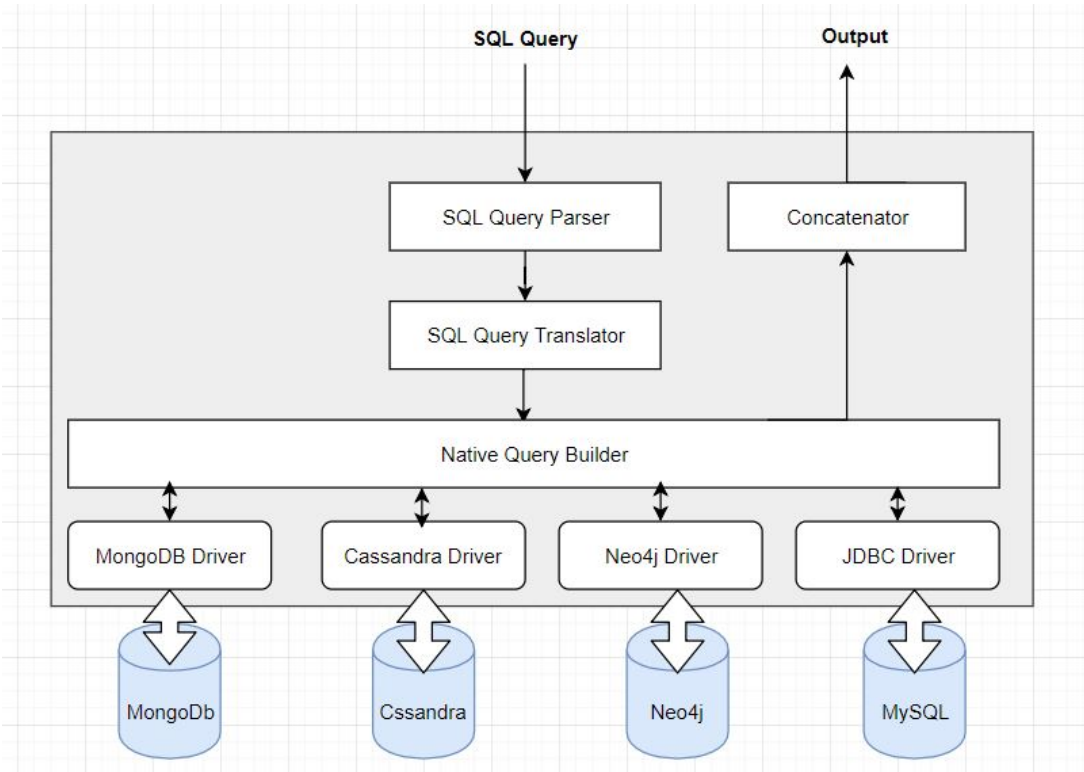

Figure 2: System Architecture

MySQL, Mongodb, Cassandra and Neo4j is chosen for this proposal. Each The framework designed would have input from user through GUI in form of SQL Query and Database option to work on. Then SQL query has to be mentioned in a way where it determines specific information about the shards on which the user wants to work on, which parameter the user wants to target and perform the operation. It is expected that user would have basic knowledge of sql query. The database selection option determines on which particular database operation would be taking place. The different subparts of architecture can be explained as following sections

3.1 Project Components

1)**SQL Query Parser:**
The SQL Query Parser is developed with the help of General Query Parser (GQP) which internally uses JavaCC . JavaCC written in java is an open source jar which helps in parsing specially SQL queries. The parser also verifies weather the query is syntactically and semantically proper. The parser breaks down SQL query into following important

parts:

- DDL/DML Clauses
- Table names
- Column names
- Parameters
- Attributes

Also, the parser generates Abstract Syntax Tree (AST) which helps to understand the relationship between attributes and parameters. A sample of generated AST from SQL query can be seen as:

SQL	AST
select u.name, u.age from user as u;	<?xml version="1.0" encoding="UTF-8"?> <sqlscript dbvendor="dbvoracle" stmt_count="1" <statement type="sstselect"> <select_statement> <query_expression is_parenthesis="false"> <query_specification> <select_list <result_column> <expression expr_type="simple_object_name_t"> <column_referenced_expr> <objectName object_type="column"> <full_name>u.name</full_name> <object_name>u</object_name> <part_name>name</part_name> </objectName> </column_referenced_expr> </expression> </result_column> <result_column> <expression expr_type="simple_object_name_t"> <column_referenced_expr> <objectName object_type="column"> <full_name>u.age</full_name> <object_name>u</object_name> <part_name>age</part_name> </objectName> </column_referenced_expr> </expression> </result_column> </select_list> <from_clause> <table_reference type="objectname"> <named_table_reference> <table_name object_type="table"> <full_name>user</full_name> <object_name>user</object_name> </table_name> <alias_clause with_as="true"> <object_name object_type="table_alias"> <full_name>u</full_name> <object_name>u</object_name> </object_name> </alias_clause> </named_table_reference> </table_reference> </from_clause> </query_specification> </query_expression> </select_statement> </statement> </sqlscript>

Figure 3: SQL to AST

2)**Query Translator**

The Query translator accepts input in the form of String and syntax tree. It generates an intermediate output String which consists of:

- CRUD operation type
- Affected Objects
- Generated Database specific relationships

  One of the important task of Query Translator is to find relationship between the attributes and values. Also, it prepares all set of parameters for native query builders to write database specific queries, as seen in Figure 4.

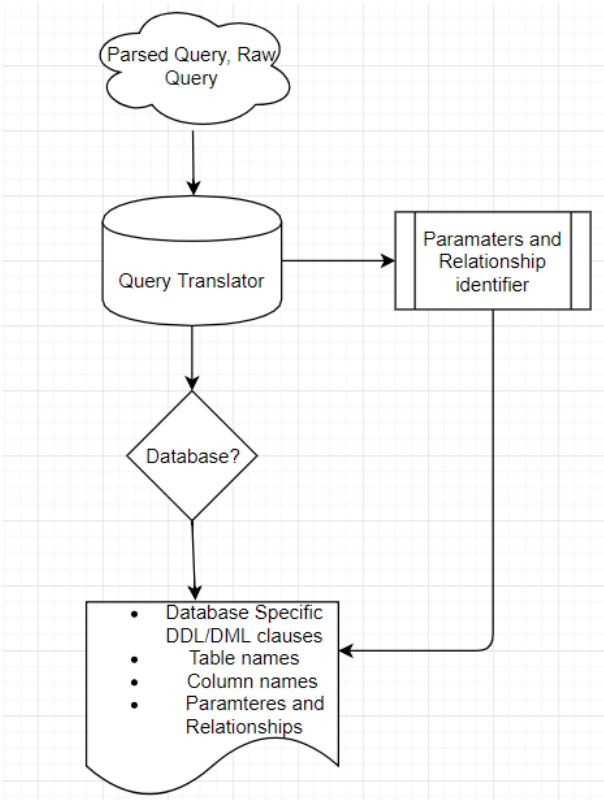

Figure 4: Query Translator

Query Translator helps Native Query Builders to build database specific queries.

3)**Native Query Builder**

The task of Native Query Builders are to develop native database specific queries and interact with respective database drivers to execute them. Metadata are obtained from different database drivers which helps Native Query Builders to have information about database/collection/column family names, read/write formats, attributes present etc. This information along with all query information together helps in formation of database specific queries. Example of clauses can be seen which is Native Query builders form: Clause describes the DDL statement. Currently clauses that are considered are

Clauses	MongoDb	Cassandra	Neo4j
select	db.find()	SELECT keyspace.tablename	MATCH nodes
insert	db.CollectionName. insertOne()	INSERT into keyspace.tablename	CREATE node
update	db.CollectionName. updateMany({ }, { $set: { } })	ALTER columnName FROM keyspace.tablename	MATCH node and SET node
delete	db.people.drop()	DELETE [column_name (term)] FROM keyspace_name.table_name	MATCH node and DELETE node

of following types: Select, Insert, Delete and Update. Clause is important in order to understand which operation to run. The clauses of all databases are different and can be presented as tabular format.

4)**Concatenate Result**

The interface helps in fetching information from multiple databases simultaneously. Structure of connected databases are different and it is important to present the result in a uniform form. This is achieved by Concatenate Result class.

3.2 Project Details

1)Algorithm:
The proposed CAP framework works with the help of a simple algorithm. The Input is of Query and Database choice and output is database results. The algorithm can be seen as,

Algorithm 1 Common Access Platform

Result: Database output
Input: String Query, String Databases[]
`SqlParser(`*query*`)` parsedString **return**
forall *Databases name* **do**
  if $name = MongoDb$ *or* $name = Cassandra$ *or* $name = Neo4j$ **then**
    `QueryTranslator(`*clause, tableName, columnName,parameters,relations*`)`
    `NativeQueryBuilder(`*operationType, databaseName, QueryDetails*`)`
  else
    `callSqlDriver(`*query*`)`
  end
end
`Concatenator(`*MongodbOutput,CassOutput,Neo4jOutput,MySQLOutput*`)`
unfiedOutput
return

2)Application Flow

The application flow starts with SQL query. The next step is check of database. If only work on SQL requested, execute the query and show result. If not, then translate SQL to AST, then AST to JSON and with help of metadata collected, build the native queries. Execute these queries one by one, collect the result, concatenate them and show output. The flow diagram can be seen in Figure 5.

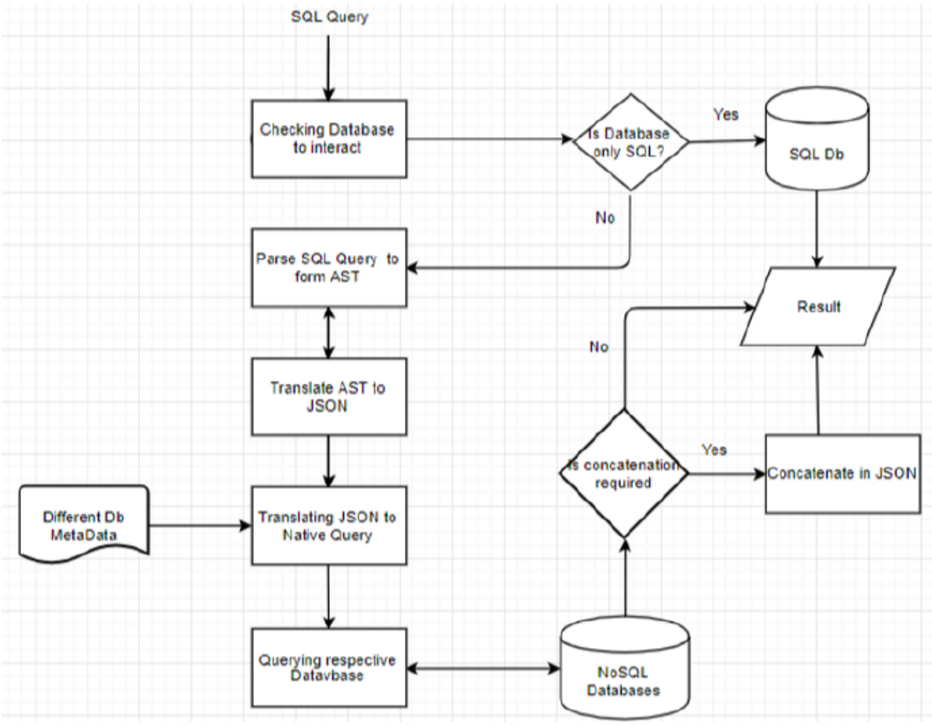

Figure 5: System Flow Chart

4 Implementation

In order to demonstrate the CAP Framework, we have developed a web application. The Web application is developed in Java (JDK 1.8) using Maven build. Database drivers used are, Neo4j-java-driver 1.4.4, cassandra-driver-core 2.6, gsp, mongodb-driver-3.0.1 and mongodb-driver-core-3.0.1. The major components of web application are:

1. Application GUI:

Front end of application consists of 2 jsp pages. The first jsp page consists of a form with two sections, Query part and database selection part. In Query part user has to enter SQL query and in database selection part, user has to select one or more database (MySQL, MongoDb, Cassandra and Neo4j). This can be seen in Figure 6.

The page consists of a user form to enter SQL query and list of databases available. The user can select one or more databases. Both the input are very crucial for the system. The inputs are received by the framework, the operation is performed and returned to front end to another jsp page which is result page.

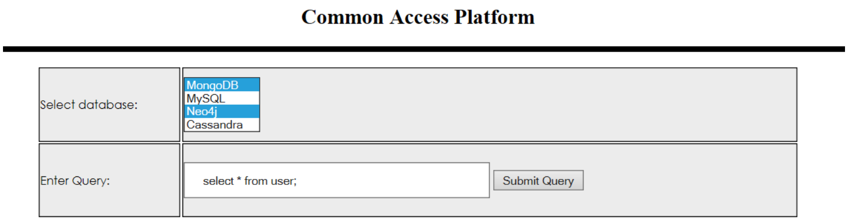

Figure 6: Application Home page

2. Back-end system:
The back-end system consists of request servlets, parsers, translators and general parser jar. It can be broadly classified as:

SQL Query Parser:
The SQL parser is implemented using GSP parser. The GSP functionality can be understood by Figure 7.

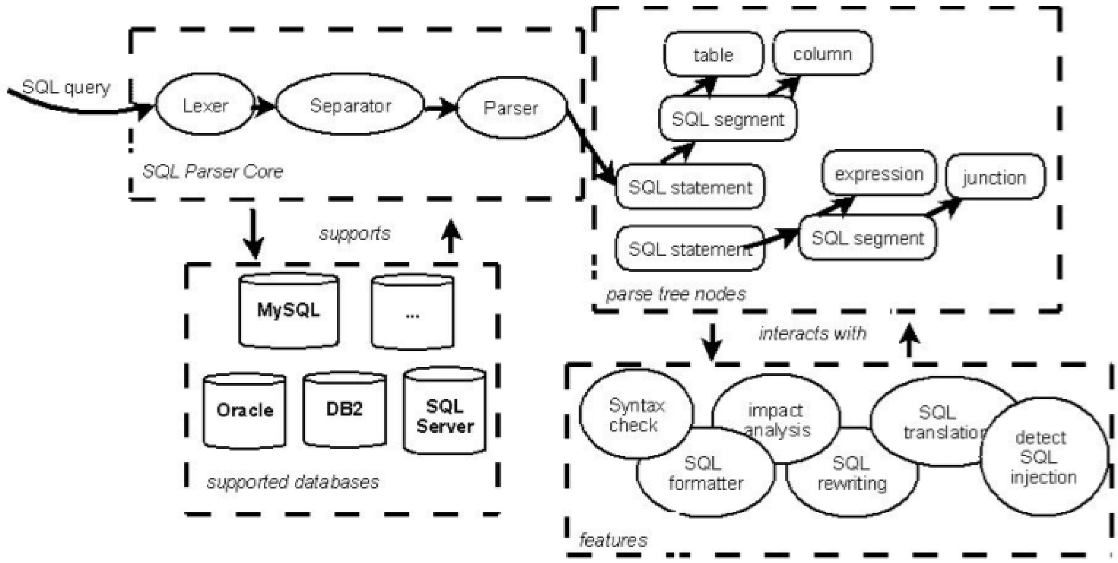

Figure 7: SQL Parser

The Data flow of parser can be seen in Figure 8.

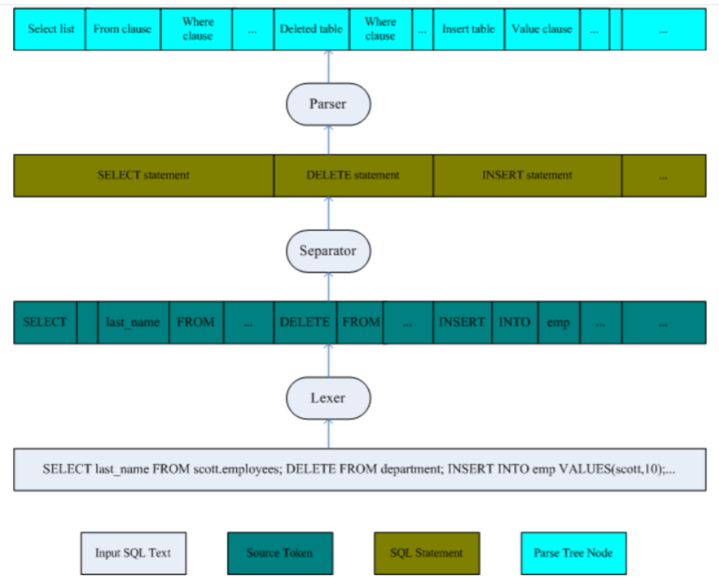

Figure 8: SQL Parser Data Flow

Query Translator:

The Query translator accepts input parameter as AST and has output depending on which database to connect. Input and output both are String. For example,

SQL Model	**MongoDb Model**	**Cassandra Model**	**Neo4j Model**
Database	Database	Keyspace	- Cluster
Table	Collection	Column Family	Node Labels
Row	Document	Super Column	Nodes
Column	Field	Column	Node property
Primary Key	_id	partition key	Node Id

Native Query Builder:

Native Query Builder has two inputs: One from Parsed query and one metadata from databases. Database specific drivers as discussed, helps them to perform their task. Also, meta data fetched from databases verifies weather the objects are appropriate or not. Using mapping standards defined in the project, Native Query Builder builds queries which are executed using drivers. Functionality of Native Query Builder can be understood by Figure 9.

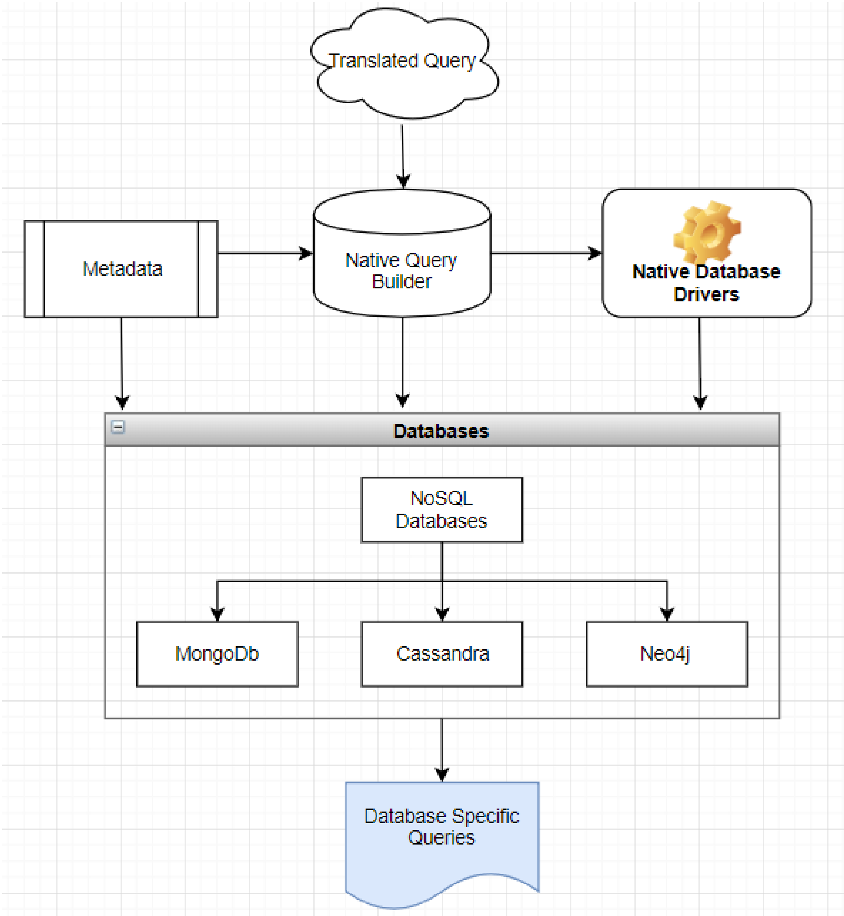

Figure 9: Native Query Builder

Concatenate Result Class:
This class concatenates the result from different data stores selected by user. Concatenation of output proves to be important when multiple databases are queried. Each database outputs result in different format and structure. So in order to present a uniform and clear output, the results obtained from individual databases are collected and provided as an input to this class. The input and output format for this class is String. An internal String parser performs the business here which filters the unwanted information and binds together the relevant data.

5 Evaluation

This section details about experiments executed on developed framework in order to verify appropriate results. The project is tested by multiple scenarios by altering queries and database choices.

5.1 Access Single Database

This scenario tests with single Database. MongoDb is selected as database and SQL query mentioned is an select statement for attribute name as 'Abhilash'. Data is inserted in MongoDb database and the from front end SQL is given as input and output obtained consists of the same data inserted in MongoDb database. The application page can be seen of user selection and also the result returned.

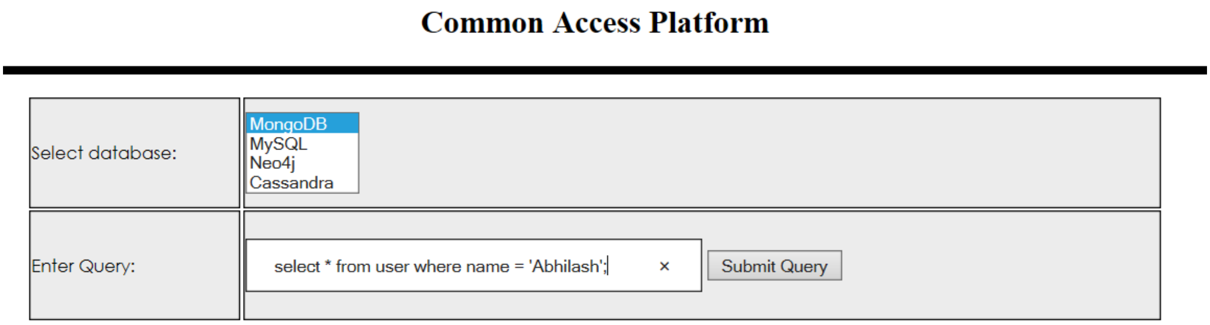

Figure 10: First Page

Databases	Output
MongoDb	{ "_id" : { "$oid" : "5b6a2dd14b2a312cbe22e187"} , "id" : 15.0 , "name" : "Abhilash Roy" , "home" : "Dublin"}

Figure 11: Output

5.2 Access Multiple Database

For each of these databases, data is inserted and can be see in Figure 12. User selecting

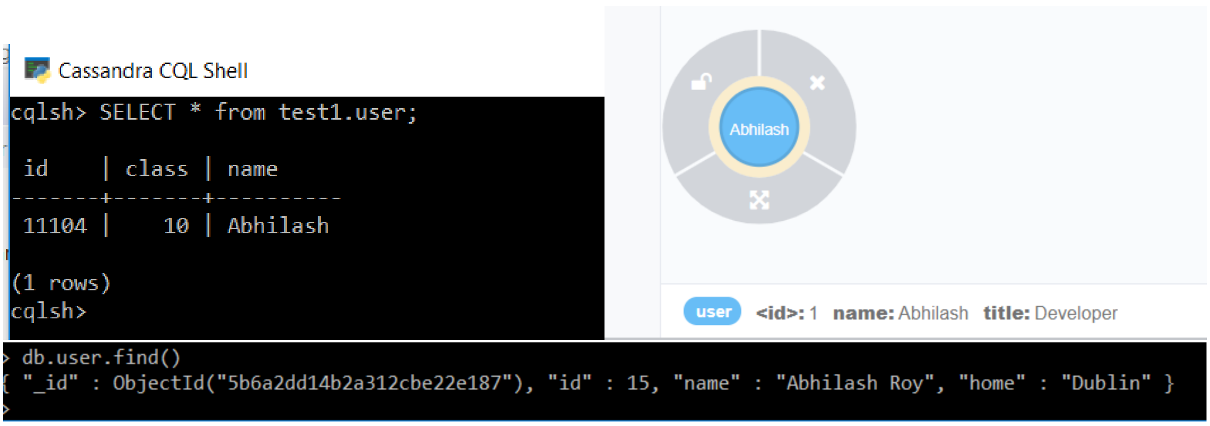

Figure 12: Database Records of Cassandra, MongoDb and Neo4j

multiple database along with entering query can be seen in the Figure 13.

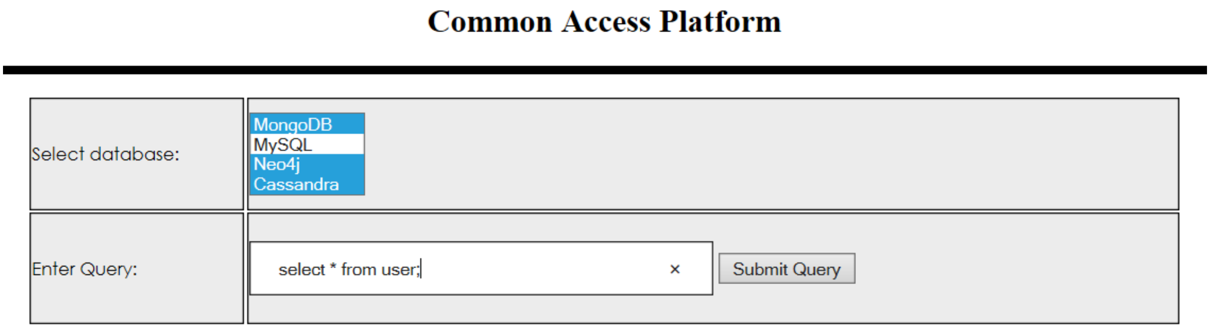

Figure 13: Selecting multiple Database

Result obtained for this scenario can be seen in Figure 14.

Databases	Output
MongoDb	{ "_id" : { "$oid" : "5b6a2dd14b2a312cbe22e187"} , "id" : 15.0 , "name" : "Abhilash Roy" , "home" : "Dublin"}
Cassandra	Row[11104, 10, Abhilash]
Neo4j	{Id: 1, name: Abhilash, title: Developer}

Figure 14: Result From multiple Database

5.3 Discussion

The two cases selected for the experiment verify that from a single interface multiple different structural databases can be accessed. First case represents a simple case how SQL can be used to interact with a NoSQL database (in this case MongoDb).Data inserted in MongoDb database matches with the output returned and the search criterion attribute was name of user. The query entered by user goes through lot of processing and is mapped with a structure which is completely different from of relation one. The output justifies that any attribute can be used to fetch data from document based database. In second test, data inserted is a simple data of a user consisting of three attributes but slightly varying in few attributes (eg, title, age) This test checks the flexibility of system with more than one NoSQL database. For this scenario data is inserted in Mongodb, Cassandra and Neo4j and is tested. All the database return appropriate result. Also, time taken is less as there is no migration and translation.

6 Conclusion and Future Work

In this work, we proposed a uniform interface for heterogeneous NoSQL databases. We developed a flexible mapping approach with use of meta data, sql parser and data mappers for accessing data from more than one SQL and NoSQL databases simultaneously. The approach used is successfully evaluated with the help of simple operations which ensures clarity for users and developers. It is observed that, this proposal is unique and indifferent in many terms like, it is not developing an intermediate query language, there is no migration of database and more databases can be added. Our approach is not based on conceptual model and also does not conceals by adding an extra layer. In fact, generalization is achieved by analyzing common features in different data models.

However, in future to add more databases, prior knowledge of database model or structure is important for application developer as end mapping of query is done with respective to destination database. The proposal handles basic CRUD operations and hence, more complex SQL queries can be considered in future. Though, document database having key value pairs has been tested using the application, key value database can also be used and tested with the application.The framework proposed promises flexibility to add or remove databases and this also can be tested in near future.

References

Atzeni, P., Bugiotti, F. and Rossi, L. (2012). Uniform access to non-relational database systems: The sos platform, *International Conference on Advanced Information Systems Engineering*, Springer, pp. 160–174, Core Rank A.

Atzeni, P., Bugiotti, F. and Rossi, L. (2014). Uniform access to nosql systems, *Information Systems* **43**: 117–133, Core Rank A*.

Bansel, A., González-Vélez, H. and Chis, A. E. (2016). Cloud-based nosql data migration, *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*, IEEE, pp. 224–231, Heraklion, Crete, Greece, Core Rank C.

Curé, O., Hecht, R., Le Duc, C. and Lamolle, M. (2011). Data integration over nosql stores using access path based mappings, *International Conference on Database and Expert Systems Applications*, Springer, pp. 481–495, Berlin, Heidelberg, Core Rank B.

Han, J., Haihong, E., Le, G. and Du, J. (2011). Survey on nosql database, *Pervasive computing and applications (ICPCA), 2011 6th international conference on*, IEEE, pp. 363–366, Port Elizabeth, South Africa, Core Rank C.

Lee, C.-H. and Zheng, Y.-L. (2015). Sql-to-nosql schema denormalization and migration: a study on content management systems, *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*, IEEE, pp. 2022–2026, Kowloon, China, Core Rank B.

Liao, Y.-T., Zhou, J., Lu, C.-H., Chen, S.-C., Hsu, C.-H., Chen, W., Jiang, M.-F. and Chung, Y.-C. (2016). Data adapter for querying and transformation between sql and nosql database, *Future Generation Computer Systems* **65**: 111–121, Core Rank A.

Liu, Z. H., Hammerschmidt, B., McMahon, D., Liu, Y. and Chang, H. J. (2016). Closing the functional and performance gap between sql and nosql, *Proceedings of the 2016 International Conference on Management of Data*, ACM, pp. 227–238, San Francisco, California, USA, Core Rank B.

Rocha, L., Vale, F., Cirilo, E., Barbosa, D. and Mourão, F. (2015). A framework for migrating relational datasets to nosql1, *Procedia Computer Science* **51**: 2593–2602, Scottsdale, Arizona, USA, Core Rank A.

Schreiner, G. A., Duarte, D. and dos Santos Mello, R. (2015). Sqltokeynosql: a layer for relational to key-based nosql database mapping, *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, ACM, pp. 74, Brussels, Belgium, Core Rank C.

Shirazi, M. N., Kuan, H. C. and Dolatabadi, H. (2012). Design patterns to enable data portability between clouds' databases, *Computational Science and Its Applications (ICCSA), 2012 12th International Conference on*, IEEE, pp. 117–120, Salvador, Brazil, Core Rank C.

Su, X. and Swart, G. (2012). Oracle in-database hadoop: when mapreduce meets rdbms, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 779–790, Scottsdale, Arizona, USA, Core Rank A*.

Tahara, D., Diamond, T. and Abadi, D. J. (2014). Sinew: a sql system for multi-structured data, *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, ACM, pp. 815–826, Snowbird, Utah, USA, Core Rank A*.

Tatemura, J., Po, O., Hsiung, W.-P. and Hacigümüş, H. (2012). Partiqle: An elastic sql engine over key-value stores, *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 629–632, Scottsdale, Arizona, USA, Core Rank A*.

Vilaça, R., Cruz, F., Pereira, J. and Oliveira, R. (2013). An effective scalable sql engine for nosql databases, *IFIP International Conference on Distributed Applications and Interoperable Systems*, Springer, pp. 155–168, Berlin, Heidelberg, Core Rank B.

Yafooz, W. M., Abidin, S. Z., Omar, N. and Idrus, Z. (2013). Managing unstructured data in relational databases, *Systems, Process & Control (ICSPC), 2013 IEEE Conference on*, IEEE, pp. 198–203.