An Approach of SQL to JSON Transformation For Handling Database Operations

K Anil Kumar, A Sureshbabu
Department of CSE, JNTUACEA, Anantapur, Andhra Pradesh, India

DOI: 10.23956/ijarcsse/V7I5/0125

Abstract— Nowadays NOSQL databases are becoming more popular. Companies like Google, Facebook, and Amazon has created their own NOSQL databases based on their requirements. Different types of querying approaches are followed by different NOSQL databases, whereas traditional databases like MySQL, ORACLE, etc. follows SQL for querying. Most of the companies are shifting from traditional databases to NOSQL databases depending upon their requirements. But the problem they encounter is the change of the querying approach where the majority of the code needs to get refactored. This paper proposes the initial step of research on how to create and manage databases with SQL as the querying approach. In a cloud computing environment, most of the research is being done for making REST as the querying approach for various data stores. This paper also proposes an SQL-to-NOSQL scheme transformation over MySQL and CouchDB.

Keywords- DBMS, JsonDB, NoSQL, SQL, JSON

I. INTRODUCTION

A database management system (DBMS) is a set of programs that manages the structure of the database and controls the access to the data stored in the database. The DBMS manages the interaction between the database and the end user. Relational databases are traditional databases which use SQL as query approach whereas Non-relational databases or NoSQL databases uses their own way of querying the database.

In 2000, Eric Brewer [3] had proposed CAP theorem, which is useful while migrating from one database to another. The CAP theorem states that any networked shared-data system can have at most two of three desirable properties i.e., Consistency (C), Availability (A) and Partition Tolerance (P). Based on CAP theorem most of the companies are shifting from traditional databases to NoSQL databases based on the requirements.

II. LITERATURE SURVEY

Migrating from SQL database to NoSQL database and providing efficient query became a challenging task [2]. Most of the researchers are focusing on developing a framework for bridging SQL and NoSQL [1]. Some of the researchers are also focusing on developing GUIs for various NoSQL databases. Due to ever-changing requirements, several NoSQL databases are coming into existence [8].

In a cloud computing environment, Rami Sellami et al. [4] proposed a generic resources model in which resources are managed by OPEN-PaaS-DataBase Application Programming Interface (ODBAPI). It is a streamlined and unified REST-based API [7]. It also enables to perform various operations on relational and NoSQL databases or through the REST-based API. During application migration, this simplifies the developer’s work by using unique API. It also alleviates the burden of interacting with other data stores at the same time.

Atzeni et al. [5] proposed a unified programming interface to seamlessly access to relational data stores and NoSQL referred to as Save Our Systems (SOS). SOS is a database access layer between an application and the different data stores that it uses. Authors defined a common interface to access different NoSQL data stores and a common data model to map application requests to the destination data store. They also suggested that SOS can be extended to integrate traditional data store; meanwhile, there is no related promise of the efficiency and the extensibility.

These related works do not support the SQL query approach which is a traditional approach for querying any relational database. Most of the databases can be queried using REST API [6]. Hence there is a requirement of creating a transformation schema over SQL and CouchDB as CouchDB uses REST API for querying its data store. The proposed work can also make use of unified programming interfaces [4] [6] which can serve various other data stores.

III. PROPOSED APPROACH

The transformation from SQL scheme to CouchDB can comprise of Query Transformation module and Representational State Transfer (REST) Application Programming Interface (API), whereas for managing own data store comprises of Query Transformation module and Storage module. Outline of the proposed approach is shown in Figure.1.
Query Transformation module involves two stages:

- **Query Selector**
- **Query Processor**

Figure 1. Architecture of the proposed work

**A. Query Selector**

Query selector takes input from the user in the form of SQL query. It processes the query by tokenizing the string and results in the query module to be processed. It results in the name of the query module if valid or an error message on failure.

**B. Query Processor**

The result of the query selector is taken as input for Query processor. It consists of various algorithms related to different operations of the query. Based on the result of query selector an appropriate algorithm is selected for transformation. This sub-module results in different ways depending on the query to be processed. Most of the resultant data could be in the form of JSON.

The result of the Query Transformation module can be:

- Taken as an input to JsonDB, one of the proposed approach which acts as a DBMS
- Make use of REST approach for managing CouchDB or other REST-based data stores

**C. JsonDB**

JsonDB is the proposed database storage and retrieval or management system to implement the proposed work. It takes SQL query as the querying approach. It is similar to accessing file system with a programming language, but the file is in JSON format. As JSON is a lightweight data-interchange format, it can be available in most of the languages to encode into it and vice versa.

**D. Accessing CouchDB using REST approach**

The proposed work can make use of REST way to handle CouchDB by treating document in CouchDB as a table in a relational environment. Creating and dropping operations in a database is similar to as in CouchDB. Creating a table involves in structuring the table with the required fields like the number of columns, column names, types, maximum length etc. in the form of JSON and creating it as a document in the data store. Insert and update operations can be treated in a similar way as editing the document. It involves

- Retrieving the data based on name
- Checking for column validations
- Creating a new data with given values and conditions in the form of a two-dimensional array (as proposed but can differ based on requirements)
- Update/Append it to the existing data
- Convert it into JSON and update the file.

**E. Accessing other data stores using REST approach**

Alongside CouchDB, proposed system can also make use of other data stores which uses REST as their querying approach. The transformations and scheme evaluation can be identified similar to the data operations as in CouchDB, discussed in the Accessing the CouchDB using REST section. Instead of using CouchDB as the data store, here a unified or common API [4] [6] can be used to use various other data stores for data operations.
IV. IMPLEMENTATION

A. **Query Selector can be achieved by the following algorithm:**

Declare variables query, len, i, temp, result
read query from user
len ← length of query
initialize i ← 0
while i<len
Read character query[i]
if query[i] = s
  temp ← substring of query indexed from i to i+3
  if temp = show
    result ← show
  elseif temp = sele and query[i+4] = c and query[i+5] = t
    result ← select
  else result ← error
else query[i] = d
  temp ← substring of query indexed from i to i+3
  if temp = drop
    result ← drop
  else result ← error
elseif query[i] = c
  temp ← substring of query indexed from i to i+5
  if temp = create
    result ← create
  elseif query[i] = i
    temp ← substring of query indexed from i to i+5
    if temp = insert
      result ← insert
    else result ← error
  else result ← error
else result ← error
Increment i by 1
end

B. **Query Processor involves various queries to be processed.**

1) **Operations related to database**
   - retrieve for the database name
   - if not found then perform only create operation, if found then perform either select or drop, otherwise an error
   - perform the required operation

2) **Operations related to the table also involve selecting a database.**

3) **Create table query**
   - retrieve the table name
   - check for the number of columns
   - retrieve each column name, type, length
   - assign all retrieved information into an associative array
   - execute createTbl

4) **Dropping a table can be in a similar way as dropping a database.**

5) **Insert query**
   - retrieve the table name
   - check if the table exists
   - retrieve column names if present from the query
   - retrieve values from the query
   - check if the number of values and number of column names if exists are same
   - validate present column names with existing column names
   - Initialize other columns with their default values
   - compare the order of existing column names and present column names
   - append column data accordingly to the existing data
   - execute editTbl
These implementations are tested with client Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36 and with server Apache/2.4.4 (Win32) and PHP-5.4.16. Implementations may slightly differ if we consider other specifications based on their environments.

V. CONCLUSION AND FUTURE WORK

The proposed system helps to create and manage new data stores. It proposes JsonDB, a new data store to make operations related to databases by using SQL as the querying approach. This system also discusses the procedure of query transformation schema among SQL and CouchDB, a NoSQL data store which uses REST API as its querying approach. It also proposed a sequence of steps to be followed to perform some of the operations related to data stores. It also helps those who require a data store which uses JSON format for storage.

The proposed work handles only basic operations on data stores. JsonDB can be further extended to make use of other advanced operations on it by modifying the proposed algorithm(s). Further, the optimization and performance of the query and implementation on other environments were also not discussed in the proposed system. After sufficient enhancements in the performance and query optimization, this system can serve a huge number of requirements.

REFERENCES


