Modeling and Querying Data in MongoDB

Rupali Arora, Rinkle Rani Aggarwal

Abstract— With the uninterrupted growth of data volumes, the storage of information, support and maintenance have become the biggest challenge. Relational database products fall behind to scaling the applications according to the incoming traffic. Due to huge data storage and scaling demands, growing number of developers and users have begun turning to NoSQL databases. This paper describes data modeling and query execution in MongoDB Document database. This paper shows how data is retrieved from MongoDB Document database without using JOIN.

Index Terms— *Column-oriented, Document-based, Graph database, Key-value, MongoDB, NoSQL, Relational database*

1 INTRODUCTION

RELATIONAL databases have been used for decades for general data storage in web and business applications with millions of reads but few writes requirements. With the advent of web 2.0 applications with millions of users reads and writes, a more scalable solution is required. The data stores for these applications needs to provide good horizontal scalability. Horizontal scalability means the ability to distribute both the data and simple read/write operations over many servers. The relational databases have little capabilities to horizontal scale over many servers. NoSQL databases were developed to deal with such large scale data needs. The term "NoSQL" was first used by Carlo Strozzi [1] in 1998 for his RDBMS, Stozzi NoSQL. Recently, the term NoSQL (Not Only SQL) has been used for databases which don't use SQL (Structured Query Language) as its query language and which don't require fixed table schema.

A key feature of NoSQL systems is "Share nothing" horizontal scaling-replicating and partitioning data over many servers [2]. Due to this feature, NoSQL systems can support a large number of simple read/write operations per second. NoSQL systems don't provide ACID (Atomicity, Consistency, Isolation, Durability) guarantees but follow BASE. BASE is acronym for Basically Available, Soft state and Eventually consistent.Basically available means that most data is available most of the time [8]. Soft-state means data is not consistent all the time but will be in eventually consistent state. MongoDb [3] by 10gen, Neo4j by Neo Technologies, Cassandra [4] by Facebook, HBase and Google's Big Table [5] are examples of NoSQL databases. NoSQL solutions are divided into four classes.

1.1 Key-Value databases

In key value data stores, data is stored in the form of keys and values [6]. Each key is unique and keys are used to retrieve the values. The query speed of these databases is higher than the relational databases. Amazon's Dynamo and Riak are famous key value data stores. These are used in applications where schema is continuously evolving.

1.2 Column-oriented databases

In column oriented data stores data is stored by columns and columns of related data is stored in same file which are called column families. These data stores are mostly used in read intensive applications. Hbase and Cassandra are famous column oriented data stores.

1.3 Document databases

Document data stores are similar to key value data stores but the value is stored in JSON or XML format. It is used for applications in which data is changed occasionally like Customer Relationship Management System

1.4 Graph databases

Graph Database uses graph structure with nodes, edges and properties of the edges to store the data. They are suited for the applications in which there are more interconnections between the data like social networks. OrientDB and neo4j are popular open source graph databases.

The objective of this paper is to show how schema is modeled and data is queried in MongoDB. The paper is organized as follows: In section 2 document\oriented databases are described. Section 3 shows class diagram and JSON representation of MongoDB schema. In section 4 describes how queries are executed in MongoDB.

2 DOCUMENT DATABASES

Document database stores data in the form of documents rather than as normalized relational table in relational databases. Data format of these documents can be JSON, BSON or XML [7]. Documents are stored into collections. The relational equivalent of document and collection are record (tuple) and relation (table).But like relation collection does not enforce fixed schema. It can store documents with completely different set of attributes. Documents can be mapped directly to the class structure of programming language but it is difficult to map RDBMS entity relationship data model. This makes easier to do programming with document databases. There is no need of JOINS in document databases as in RDBMS due to

- *Rupali Arora is currently pursuing masters degree program in computer science and engineering inThapar University, Patiala, India, PH-9463166166. E-mail: arora.mona2@mail.com*
- *Rinkle Rani Aggarwal is Assistant Professor in computer science and engineering inThapar University, Patiala, India, PH-9915554748. E-mail: raggarwal@thapar.edu*

embedded document and arrays. That is why today a growing number of developers are moving to document databases. CouchDB by Apache Software Foundation and MongoDB by 10gen are open source databases built for scalability and ease of use.

MongoDB is an open source NoSQL document database, initiated by 10gen Company. It was designed to handle growing data storage needs. It is written in c++ and its query language is JavaScript. MongoDB stores data in the form of collections. Each collection contains documents. MongoDB documents are stored in binary form of JSON called BSON format. BSON supports Boolean, float, string, integer, date and binary types. Due to document structure, MongoDB is schema less. It is easy to add new fields to a document or to change the existing structure of a model. MongoDB offers a technique named **Sharding** to distribute collections over multiple nodes. When nodes contains different amount of data, MongoDB automatically redistribute the data so that load is equally distributed across the nodes. MongoDB also support **Master-slave replication**. The slave nodes are copies of Master nodes and used for reads or backups.

2 Data Modeling In MongoDB

The example database which has been used for querying MongoDB contains three collections- User, Tag and Post. Post contains an embedded document named comments. A user can have any number of posts. A post contains tags. User can comment under any post. Class diagram and JSON format is used for modeling schema of the database.

2.1 Class Diagram

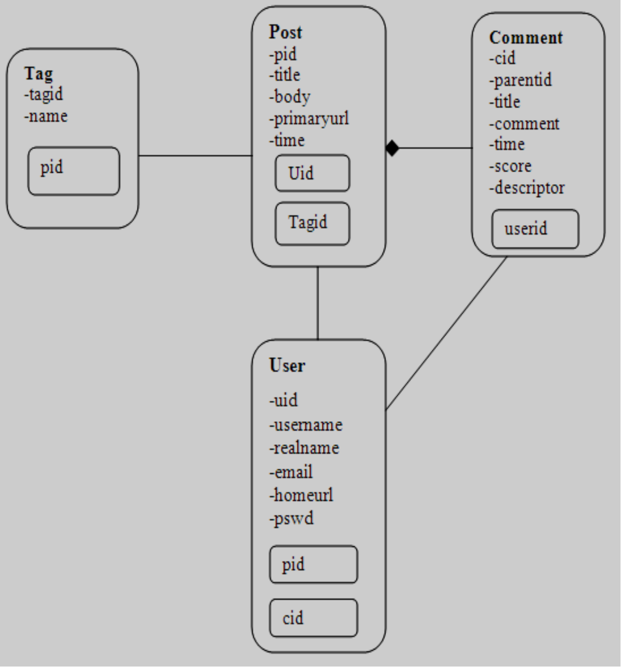

Fig. 1. Class diagram representation of case study

MongoDB does not use JOINS to relate documents like Relational database [3]. In MongoDB data is stored in the denormalized form in which related data in stored in single document. This is known as Embedding. In the fig 1 Post and Comment are embedded documents. But sometimes we need to store the data in different documents like in fig 1. Post, User and Tag are different documents. So, to relate these documents _id field of one document is saved in another document as reference. E.g. in fig 1 uid is used as reference field of User document in Post document to indicate a relationship between documents.

2.2 JSON Format Representation

In MongoDB data is stored in the form of BSON documents. BSON is binary representation of JSON. In MongoDB documents data is represented in the form of field and value pairs. A field-value pair is comprised of a “field name” in double quotes, followed by colon “:” and then “value” in double quotes. The values can be another documents, arrays and array of documents. Each pair is separated by comma. Documents are held within curly (“{ }”) brackets and arrays are held within square (“[ ]”) brackets. Example database that have been used for querying MongoDB has the document structure shown below. Post, User and Tag are collections. “_id” field is unique- id field and may contain value of any BSON data type other than array. In post collection tagid is array of strings and comments contain array of sub documents.

```
“Post”:
{
        “_id”:”<string val>”,
        “uid”:”<integer val>”,
        “title”:”<string val>”,
        “body”:”<string val>”,
        “primary url”:”<string val>”,
        “time”:”<datetime val>”,
        ”tagid”:["<string val>”],
         “comments”:[
                {
                                “cid”:”<string val>”,
                                 “parented”:”<string val>”,
                                “userid”:”<integer val>”,
                                “title”:”<string val>”,
                                “comment”:”<string val>”,
                                 “time”:”<datetime val>”,
                                  “score”:”<number val>”,
                                  “descriptor”:”<string val>”
                }
        ]
}
  “User”:
        {
       “_id” :”< integer val>”,
       “username”: “<string val>”,
       “realname”: “<string val>”,
       “email”:”<string val>”,
       “homeurl”:”<string val>”,
        “pswd”:”<string val>”,
      ”pid”:[“<string val>”],
      “cid”: [“<string val>”]
       }

   “Tag”:
       {
     “_id”: “<string val>”,
      “name”:”<string val>”,
      ”pid”:[“<string val>”]
        }
```

7 Querying MongoDB

As data modeling is important, it is also important to know how queries are executed in MongoDB. This section describes how queries are written in MongoDB. Seven Queries are designed to describe syntax in MongoDB and to show how same queries are written in MySQL.

Query1: Find the tag names which have been used in the post under which a particular user has commented. (user=’a1’)

Query2: Find the tag(s) which have been used in the posts of each user.

Query3: Given a cid of a comment find its related post, parent comment and tags associated with the post.

Query4: Find the time of the post under which some user has commented. (user=’a1’)

Query5: List the Posts of each user.

Query6: Find the total number of posts by a particular user. (uid =102)

Query7: Find out the username, uid, tags associated with posts of a comment and score associated with comments.

Query1.

<table>
<tr><td><b>MySQL</b></td><td>mysql&gt; select t.name from comment c,post p,user u,tag t where p.pid=c.postid an d c.userid=u.uid and p.pid=t.pid and u.username='a1';
output:
+--------+
| name |
+--------+
| space |
| apple |
| google |
| apple |
+--------+</td></tr>
<tr><td><b>MongoDB</b></td><td>&gt; var u=db.user.findOne({username:"a1"})
>var tag1=db.post.find({"comments.userid":u._id})
> db.tag.find({_id:tag1.tagid},{_id:0,name:1})
Output:
{ "name" : "space" }
{ "name" : "google" }
 "name" : "apple" }</td></tr>
</table>

The find () method selects documents from a collection that meets the <query> argument. <Projection> argument can also be passed to select the fields to be included in result set. The find () method returns a cursor to the results. This cursor can be assigned to variable.

db.collection.find(<query>,<projection>)

The findOne() method is similar to find() method but it selects only one document from a collection.

In this query first the document with username ‘a1’ is extracted from User collection and stored in variable u. Then the comments corresponding to that user are found by matching u. id field with userid field of subdocument comments (comments.userid) in Post Collection and returned cursor is stored in the variable tag1. Then _id is matched in Tag collection. To exclude _id field {_id:0} is written in projection field and the fields which needs to be included like here name is to be included is written as {name: 1} in projection argument.

Query2.

MySQL	mysql>select u.uid,t.tagid from user u,post p,tag t where u.uid=p.uid and p.pid =t.pid;
MongoDB	db.post.find({},{_id:0,uid:1,tagid:1})

Query3.

MySQL	mysql> select c.cid, c.postid, c.parentid, t.tagid from comment c,post p ,tag t where c.postid=p.pid and p.pid=t.pid and c.cid='c2';
MongoDB	db.post.find({"comments.cid":"c2"},{"comments. cid":1, "comments.parentid":1,tagid:1});

Query4.

MySQL	mysql> select p.time from comment c,post p,user u where p.pid=c.postid and c. erid=u.uid and u.username='a1';
MongoDB	> var u=db.user.findOne({username:"a1"}) > db.post.find({uid:u._id},{time:1,_id:0})

Query5.

MySQL	mysql> select u.uid,p.pid from user u,post p where u.uid=p.uid;
MongoDB	db.post.find({},{_id:1,uid:1})

Query6.

MySQL	mysql> select count(p.pid) from post p,user u where u.uid=p.uid and u.uid=102;
MongoDB	> db.post.count({uid:102})

Query7.

MySQL	mysql> select c.cid, u.username,c.userid,c.score, t.tagid from comment c,user u,tag t where c.userid=u.uid and c.postid=t.pid;
MongoDB	> db.post.find({},{tagid:1,_id:0,"comments.score":1, "comments.userid":1,"comments.cid”:1})

4 Conclusion

In this paper data modeling in MongoDB has been shown by using Class diagram and JSON format. It also has been shown that how queries are written in MongoDB. MongoDB does not use JOINs to relate documents like Relational Databases. In this all the data is stored in Single document or if needs to store in different documents then documents are related by using reference fields.

References

[1] Strozzi, Carlo: NoSQL-A relational database management system.2007-2010-http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql /Home%2520 page

[2] R. Cattell. Scalable sql and nosql data stores. SIGMOD Rec., 39(4):12–27, May 2011.

[3] MongoDB. Available. http://www.mongodb.org/

[4] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer. Apache hadoop goes realtime at facebook. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th symposium on Operating systems design and implementation, OSDI '06, pages 205–218, Berkeley, CA, USA, 2006. USENIX Association.

[6] InfiniteGraph: The Distributed Graph Database, Objectivity, Inc., Sunnyvale, CA,2012

[7] Johannes Zollmann. (2012, Aug 20). NoSQL Databases [Online].Available:http://sewiki.iai.uni-bonn.de/_media/teaching/labs/xp/ 2012b/seminar/10-nosql.pdf

[8] Olli Sutinen,” NoSQL – Factors Supporting the Adoption of Non-Relational Databases” M.Sc thesis, Dept. Comput. Sci.,tampere Univ., Finland, 2010