Data Modeling Using MongoDB

[1]Vaibhav V. Mhetre, [2]Nikhil R. Ranade
Dept. of MCA, FAMT[1,2]
Ratnagiri, India[1,2]
University of Mumbai[1,2]
[1]vaibhavmhetre27@gmail.com
[2]nikhilranade3189@gmail.com

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.

Key Terms- NoSQL, Shading, BSON

I. INTRODUCTION

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 Shading to distribute collections over multiple nodes. When nodes contains different amount of data, MongoDB automatically redistribute the data so that load is equally distribute across the nodes. MongoDB also support Master-slave replication. The slave nodes are copies of Master nodes and used for reads or backups.

MongoDB is a database management system designed for web applications and internet infrastructure. The data model and persistence strategies are built for high read and write throughput and the ability to scale easily with automatic failover. Whether an application requires just one database node or dozens of them, MongoDB can provide surprisingly good performance. If you've experienced difficulties scaling relational databases, this may be great news. But not everyone needs to operate at scale. Maybe all you've ever needed is a single database server.

II. WHY SHOULD WE USE MONGODB?

It turns out that MongoDB is immediately attractive, not because of its scaling strategy, but rather because of its intuitive data model. Given that a document-based data model can represent rich, hierarchical data structures, it's often possible to do without the complicated multi-table joins imposed by relational databases.

For example, suppose you're modeling products for an e-commerce site. With a fully normalized relational data model, the information for any one product might be divided among Dozens of tables. If you want to get a product representation from the database shell, we'll need to write a complicated SQL query full of joins. As a consequence, most developers will need to rely on a secondary piece of software to assemble the data into something meaningful.

With a document model, by contrast, most of a product's information can be represented within a single document. When you open the MongoDB JavaScript shell, you can easily get a comprehensible representation of your product with all its information hierarchically organized in a JSON-like structure.1 You can also query for it and manipulate it. MongoDB's query capabilities are designed specifically for manipulating structured documents, so users switching from relational databases experience a similar level of query power. In addition, most developers now work with object oriented languages, and they want a data store that better maps to objects. With MongoDB, the object defined in the programming language can be persisted "as is," removing some of the complexity of object mappers.

A. Data Modeling :

1. 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. 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 embedded document and arrays. That is why today a growing number of developers are moving to document databases. 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. It provides Sharing 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. 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.

```
{
_id:ObjectID ('4bd9e8e17cefd644108961bb'),
Title: 'Student Result Generation',
url: 'http://smyschool.com/studDB.txt',
     author: 'nikhil',vote_count: 20,
tags: ['databases', 'mongodb', 'indexing'],
image: {
          url:
          'http://smyschool.com/prof_logo.jpg',
          caption: '',
          type: 'jpg',
          size: 75381,
          data: "Binary"
          },
          Personal_info:{
          Userid,
          Address,
          }
comments: [ {
          user: 'Manoj', percentage:78.00
          Address:[   {
                              Street:'R.S.Road',
                              BloackNo:2,
                              City:'Ratnagiri'
                        }
                    ]
                    },
                    {
          user: 'Rahul',percentage:88.00
          Address:[   {
                              Street:'S..S.Road',
                              BloackNo:4,
                              City:'Pune'
                         }
                    ]
                    } ] }
```

Above listing shows a sample document representing student data as you can see, a document is essentially a set of property names and their values. The values can be simple data types, such as strings, numbers, and dates. But these values can also be arrays and even other documents. These latter constructs permit documents to represent a variety of rich data structures. You'll see that our sample document has a property, tags which store the tags in an array. But even more interesting is the comments property which references an array of comment documents.

Let's take a moment to contrast this with a standard relational database representation of the same data. Figure shows a likely relational analogue. Since tables are essentially flat, representing the various one-to-many relationships in your relational database are going to require multiple tables. You start with a student table containing the core information for each student like name, address. Then you create three other tables, each of which includes a class, subject, and marks referencing the original table. The technique of separating an object's data into multiple tables' likes this is known as *normalization*. A normalized data set among other things, ensures that each unit of data is represented in one place only .But strict normalization isn't without its costs. Notably, some assembly is required.

To display the post we just referenced, you'll need to perform a join between the student info and tags tables. You'll also need to query separately for the comments or possibly include them in a join as well. Ultimately, the question of whether strict normalization is required depends on the kind of data you're modeling, and I'll have much more to say about the topic in chapter 4. What's important to note here is that a document oriented data model naturally represents data in an aggregate form, allowing you to work with an object holistically: all the data representing a post, from comments to tags, can be fitted into a single database object.

3. MongoDB's key features

`In addition to providing a richness of structure, documents need not conform to a pre-specified schema. With a relational database, you store rows in a table. Each table has a strictly defined schema specifying which column and types are permitted. If any row in a table needs an extra field, you have to alter the table explicitly. MongoDB groups documents into collections, containers that don't impose any sort of schema. In theory, each document in a collection can have a completely different structure; in practice, a collection's documents will be relatively uniform. For instance, every document in the posts collection will have fields
For the title, tags, comments, and so forth.
But this lack of imposed schema confers some advantages.

First, your application code, and not the database, enforces the data's structure. This can speed up

initial application development when the schema is changing frequently.
Second, and more significantly, a schema less model allows you to represent data with truly variable properties.

Not all databases support dynamic queries.

For instance, key-value stores are query able on one axis only: the value's key. Like many other systems, key-value stores sacrifice rich query power in exchange for a simple scalability model. One of MongoDB's design goals is to preserve most of the query power that's been so fundamental to the relational database world.

To see how MongoDB's query language works, let's take a simple example involving posts and comments. Suppose you want to find all posts tagged with the term *politics* having greater than 10 votes. A SQL query would look like this:

```
SELECT * FROM student_info
INNER JOIN result ON student_info.id = result.stud_id
INNER JOIN class ON result.id == class.id
WHERE class.name='FYMCA" AND result.percentage> 50;
```

The equivalent query in MongoDB is specified using a document as a matcher. The special $gt key indicates the greater-than condition.

```
db.student_info.find({'class': ' FYMCA ', 'percentage': {'$gt': 10}});
```

Note that the two queries assume a different data model. The SQL query relies on a strict normalized model, where student and classes are stored in distinct tables, where as the MongoDB query assumes that tags are stored within each post document. But both queries demonstrate an ability to query on arbitrary combinations of attributes, which is the essence of ad hoc query ability.

III. CONCLUSION

In this paper it has been shown that how the data can be effectively modeled using MongoDB instead of writing complex queries. All related records stored as a single document. 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] MongoDB in action by Kyle Banker
[2] IEEE research paper on Modeling and Querying Data in MongoDB by Rupali Arora, Rinkle Rani Agarwal