From Relational Model to Rich Document Data Models - Best Practices Using MongoDB

Vinu Sherimon[1], Sherimon P.C.[2]

***Abstract*— Open Source Software steps up the development of today's diverse applications. NoSQL databases gain popularity in the recent years as they address the limitations of relational databases at a lower cost. MongoDB is a cross-platform and rich document NoSQL model database designed for modern applications. It works on the concept of collection and document offering high performance, scalability and flexibility. This paper describes data modeling using MongoDB. The schema design and the representation of data in MongoDB are explained. The modeling of relationships with embedding and referencing is also described. The query model in MongoDB is also outlined in the paper. The paper concludes with the comparison of document based model with relational model.**

***Index Terms*— BSON, data modeling, document model, MongoDB**

I. INTRODUCTION

Open source coding is gaining momentum nowadays and continues its growth and development, while proprietary software is slowly losing ground. A variety of open source software is available to cater the needs of every organization. IT industry is steadily turning towards open source software. IT Techies' are looking for easy and better way of coding problems. They gather their ideas and knowledge in open forums, work together and generates new software. The code is open to anyone to contribute more, thus open source software offers complete freedom of solutions. There are several reasons that an organization is looking for a change in proprietary software. The main reason is the need to handle unstructured or semi-structured data which comes in huge volumes which cannot be handled by existing systems. Other reasons are to adapt to the exiting market requirements quickly and to identify less expensive options as compared to expensive proprietary software. Now most of the online courses available focus on teaching using open source software. World's best IT competitions are looking for guys who can contribute to open source community. Relational databases were popular before the era of big data. Structured data was safe in the hands of relational databases. Now with the big data, the performance of the relational databases comes down drastically and NoSQL databases catch the market [1].

The rest of the paper is organized as follows: - Section II explains the background. The concepts of data modeling in MongoDB are described in Section III. It includes the schema design, creation of databases, collections and documents, validation constraints etc. The modeling of one-to-one, one-to-many and many-to-many relationships is explained in Section IV. Section V outlines the query modeling in DB. The comparison of document based model with relational model is explained in Section VI. Conclusion is presented in Section VII followed by References.

[1]Department of IT, Higher College of Technology, Muscat, Sultanate of Oman (e-mail: vinusheri@hct.edu.om)

[2]Faculty of Computer Studies, Arab Open University, Muscat, Sultanate of Oman (e-mail: sherimon@aou.edu.om)

II. BACKGROUND - MANAGING DIVERSE DATA

Even though the hardware, applications and data have undergone a massive change during the past years, the underlying data management techniques remained the same. Relational model was dominating the world. Today's businesses have a wealth of data. These data gets accumulated in this world daily from mobile devices, emails, audios, videos, web logs, social media networks, sensors and other sources. The data type of much of the data comes under unstructured, semi-structured or polymorphic categories and they are held up in different data stores, in different locations. These unstructured data can help companies gain a better understanding of their customers, products, services and business in general [2]. For example, by gathering and analyzing the data from different social media networks, it is easy to understand about the interests of customers towards a particular product, thus marketing the specific product with more focus. This solving of unstructured data is a real challenge.

Frequently cited statistics say that 80-90% of data in an organization is not structured data. The amount of this unstructured data is growing significantly. Relational model is not sufficient to handle this data [3] in terms of performance. To handle this potentially huge volume of diverse data, a variety of open source tools are in use. Hadoop is the basic platform based on big data to handle unstructured data. NoSQL (Not only SQL) databases are becoming increasingly popular nowadays [3]. Most of the open source tools demands a technology upgrade, using new analytical tools and shifting from the traditional structured data modeling to the unstructured data modeling. What is important is to understand about the type of information or data insights, the businesses are looking to figure out from the data. The future of analytics lies in open source technology.

NoSQL databases are either based on document, key-value pairs, graphs or wide-column stores. Several studies are being done to compare the relational model with NoSQL databases. In [1], the authors have compared the performance of data storage and retrieval mechanisms of the relational model with graph databases and column-store databases. Another study compared the relational, document and graph databases in the context of a web application [3]. The comparison of MySQL and MongoDB databases was done in a study which analyzed the CRUD (Create, Read, Update, and Delete) operations on different data sets [4]. A comprehensive study was published in 2013on the good and the bad aspects of NoSQL systems [5]. In this paper, the author condemns about the lack of a common language in NoSQL systems, no sophisticated query optimizer and the migration between different NoSQL systems.

III. DATA MODELING CONCEPTS IN MONGODB

Before the advent of online and mobile applications, databases processes structured data. Simple data models were described as a set of entities, attributes and relationships among the entities. Today's unstructured data is not organized into rows and columns as done in structured data. To address unstructured data, NoSQL databases were developed. MongoDB, the document based database was built specifically to handle unstructured data. It is the leading modern database platform and serves as a catalyst for the Big Data movement.

A. Schema Design

The fundamental change in transferring from a relational database structure to NoSQL database systems is the change in the modeling of data. For instance, a relational database model follows a two-dimensional structure in which data is kept in rows and columns, whereas in MongoDB, there is no rigid structure and it follows a document based model which contains data which can be embedded sub-documents, or arrays. In SQL databases, it is necessary to define the database schema before inserting any data. But MongoDB documents do not have a pre-defined schema. It follows a flexible and dynamic data structure. Figure 1 shows the comparison of basic terminologies in RDBMS and MongoDB.

RDBMS	**MongoDB**
Database	Database
Table	Collection
Tuple/ Row	Document
Column	Field
Primary Key	Default _id field in a document
Index	Index

Fig. 1 Comparison of basic terminologies

In MongoDB, a collection is created without defining the schema first (document fields and their data types). At any point new fields can be inserted and existing fields can be updated or removed, thus changing the schema of a collection. This is very beneficial for developers as they can change the database design as they develop the application as per any new requirements. While designing the schema, special care must be taken to understand about the ratio of read/write operations, type of query operations to be performed, etc.

B. Databases, Collections and Documents

The *use* command is used to create a database in MongoDB. For example, *use DB1* will create a new database "DB1". The default database of MongodB is '*test'*. The *db.createCollection()* method is used to create a collection in MongoDB [6]. The following example illustrates the creation of "products" collection.

```
db.createCollection("products")
```

Documents are inserted into a collection using *db.collection.insert()* method [6]. For example, suppose we want to insert documents into the *products* collection. This is done as follows:-

```
db.products.insert( { pname: "pencil", qty: 24, price:2 } )
```

The *find()* method is used to query a MongoDB document [6]. It displays a document in an unstructured format. The *pretty()* method is used to display a document in a structured format [6]. For instance, *db.products.find().pretty()* displays the result as follows:-

```
{
  "_id": ObjectId("644de2f80 mn8670f89r789a8"),
  "pname" : "Pencil",
  "qty" : 24,
   "price":2
}
```

An _id field is automatically created and a unique ObjectId is assigned to it as shown below. Each object has a unique ID to identity it within the collection. The_id field in each document is either generated automatically or created by the user.

C. BSON format

Nowadays data is often represented as JSON (Javascript Object Notation) documents rather than using tables since they follow complex structures. MongoDB uses BSON (Binary JSON) to represent the data. BSON extends JSON and includes additional data types such as *int*, *long* and *floating point*. BSON format allows embedded data to contain in the collections. JSON documents also align with the structure of objects in modern programming languages [7]. This makes it easy for developers to map the data used in the application to its associated document in the database [7].

The maximum allowed size of a MongoDB document (BSON) is 16 MB. In reality, many documents are a few kilobytes. Some applications require more size for a document. MongoDB supports many data types such as *String, Integer, Boolean, Array, Object*, etc.For instance, consider "products" and "reviews". For a product, there may be millions of reviews. It is not good to embed all the reviews in a product document. Instead, the reviews can be kept as a separate collection with a reference to the product document. Also few top reviews can be embedded in the product document itself for immediate access.

D. Storage Engines

WiredTiger is the default storage engine for MongoDB [8]. The latest versions of MongoDB offers an encrypted storage engine based on WiredTiger. So in this case a separate file encryption is not required. There is also an In-Memory storage engine for extremely powerful applications that require real time analytics. MongoDB uniquely allows users to mix and match multiple storage engines within a single MongoDB cluster [9].

E. Validation Constraints

NoSQL way of structuring data get rid of the constraints imposed by the relational model [4]. In all MongoDB collections, an _id field is automatically inserted in every document. This field is reserved for primary key. If the user didn't fill this field, then it will be automatically filled with MongoDB ObjectID, which is a unique value. Document based model embeds the sub-documents inside the parent document. So JOIN is mostly avoided. So the use of foreign key constraint is also reduced. Also validation rules can be defined for a collection. For example, rules can be enforced to check data types, domain, other components of the documents, etc.

Document Validation is now supported in the latest versions of MongoDB. A document is validated during any insertions or updates. The *validator* option is used to specify the validation rules in a document. Validation rules can be added when creating a collection or to an existing collection. The example below shows how to add a validator to a *student* collection when creating the collection.

```
db.createCollection (“student”, {
  validator: {age: {$lte: 40, $gte: 17}}
})
```

Another validation is to ensure the existence of a common field among all documents in a collection. For example, suppose we want to create a validation that that there must be an “email” field in an existing “student” collection. This is done through the following validator:

```
db.runCommand( {
   collMod: "student",  validator: {email: {$exists: true}}
})
```

validationLevel and *validationAction* are two other options available in MongoDB. The option *validationLevel* determines how strictly MongoDB apply the validation rule during a document update. The default value of *validationLevel* is ‘strict’, means MongoDB applies validation rules to all inserts and updates. But if the value is set as ‘moderate’, then MongoDB will not apply rules to such documents that do not satisfy the rules. MongoDB uses the option *validationAction* to handle documents that violate the rules. By default the value of *validationAction* is ‘error’, means MongoDB rejects all the documents that violates the rules. But if the value is ‘warn’, then MongoDB proceeds with the insert/update operation and issues a warning message.

Example: Consider the following two documents in “student” collection.

```
{
_id:23,
name:”Samar”,
age:16,
phone:99339021
}

{
_id:45,
name:”Ammar”,
age:22,
email:”amr@hct.edu.om”,
phone:24444904
}
```

In the above example, suppose we want to apply a validator as shown below.

```
db.runCommand( {
   collMod: "student",  validator: {email: {$exists: true}},
validationLevel:”moderate”,
validationAction:”warn”
})
```

Here if any attempt is made to update the document with _id 45, MongoDB applies the validation rules since it satisfies the criteria. But MongoDB will not apply the rules for any updates made to the document with _id 23 since it doesn't match the criteria in the rule. Also since the *validationAction* is warn, the following student document is inserted into the collection, but a warning message is generated.

```
db.student.insert({_id:78,name:”Amna”,age:23})
```

IV. Modeling Relationships in MONGODB

MongoDB allows embedding sub-documents or arrays inside the main document or creating references between documents in different collections [10].

A. One-to-One relationships

If the data follows a one-to-one relationship, then related document is usually embedded inside the main document, but it is not mandatory. For example, consider “customers” and “addresses”. Each customer has one address and each address is associated with one customer. It is modeled as follows:-

```
{
“_id”:”99”,
“custName”:”Khalid”,
“address”: [
{  “way_no”:”788”,”flat_no”:”F22”,”city”:” Muscat”   }
       ]
}
```

Here the address field is a sub-document represented as an array of fields.

B. One-to-N relationships

For modeling One-to-N relationship, there are different options as per the cardinality. The basic modeling is to embed an array of sub-documents into the parent document. But it is not the only available option. It depends on what exactly is the cardinality of “N”. The different options for “N” are few, many or millions.

- One-to-few - An example of this type of relationship is to model “Parent” and “Child”. The details of children can be embedded inside each Parent document.

```
{
parentName:”Khalid”
children: [
{  _id:”1”,”cname”:”Sara”,”age”: 14  },
{  _id:”2”,”cname”:”Amal”,”age”: 11},
{  _id:”3”,”cname”:”Amna”,”age”: 7}
      ]
   }
```

In the above example, the _id field is provided by the user.

- One-to-many - An example of this type of relationship is to model “Student” and “Course”. Here also the details of courses can be embedded inside each student document as the one-to-few relationship.

```
{
StudentID: “101”,
FirstName: “Salim”,
LastName: “Said”,
City: “Muscat”,
Courses:  [
   {  CourseCode: “DB1102”,
                    CourseTitle: “Introduction to Database”
                    Credits: “3”
   },
   {
      CourseCode: “SE1101”,
                    CourseTitle: “Introduction to
          Programming”
                    Credits: “4”
   }
]
   }
```

- One-to-millions - An example is to model “People” and “Cities”. Here we have a large number of people living in each city. So it is not efficient to embed the details of people in city document since the size of the document becomes very large. The size of a document in MongoDB is limited to 16 MB. Here the only option is to apply normalization as in the case of any RDBMS.

C. *N-to-N relationships*

To model many-to-many relations, two collections are used. Within each document in a collection, the related links to other collection’s documents are stored.

For instance, consider the entities “Student” and “Course”. Each student study many courses and each course is studied by many students.

The documents in “Student” collection can be:

```
{
  "_id" : ObjectId("623de 2f30 mn8sd0f89r205z1"),
  "stname" : "Salim Said",
  "courses" : [
     ObjectId("623de12a6mn8sd0f89r205z2"),
     ObjectId("623de 34a0 mn8sd0f89r205z3"),
       ]
}

{
  "_id" : ObjectId("623de 2f30 mn8sd0f89r205z5"),
  "stname" : "Malek Moosa",
  "courses" : [
     ObjectId("623de 2f30 mn8sd0f89r205z6"),
     ObjectId("623de12a6mn8sd0f89r205z2")
  ]
}
```

The documents in “Course” collection are:

```
{
  "_id" : ObjectId("623de12a6mn8sd0f89r205z2"),
  "cname" : "Introduction to Database",
  "students" : [
     ObjectId("623de 2f30 mn8sd0f89r205z1"),
     ObjectId("623de 2f30 mn8sd0f89r205z5")
  ]
}

{
  "_id" : ObjectId("623de 34a0 mn8sd0f89r205z3"),
  "cname" : "Introduction to Programming",
  "students" : [
     ObjectId("623de 2f30 mn8sd0f89r205z1")
  ]
}
{
  "_id" : ObjectId("623de 2f30 mn8sd0f89r205z6"),
  "cname" : "Computer Hardware",
  "students" : [
     ObjectId("623de 2f30 mn8sd0f89r205z5")
  ]
             }
```

The _id’s are used to link the student documents with course documents and vice versa. Now the fetching of related data must be handled in the corresponding application.

In one-to-one, one-to-few and one-to-many the data is pre-joined by combining the related fields in a single data structure. This avoids the need of normalization which is applied in traditional RDBMs systems. Even though the de-normalization of document model approach increases the storage requirements, it is insignificant when we compare it with the efficiency of reading/writing an entire record with a single operation.

V. QUERY MODELING USING MONGODB

MongoDB, provides a rich set of indexing options to query a collection. It includes text indexes, geospatial indexes, compound indexes, sparse indexes, time to live (TTL) indexes, unique indexes, and others [11]. MongoDB provides native drivers for all popular programming languages and frameworks such as Java, .NET, Ruby, PHP, JavaScript, Python, Perl, Scala, in addition to 30+ community-developed drivers [12]. MongoDB doesn’t use SQL as in the case of RDBMS. It uses methods or functions from the concerned API to communicate with database.

MongoDB offers a set of tools to interact with database. The mongo shell is a rich, interactive JavaScript shell that is included with all MongoDB distributions [12]. The shell is used to search, insert, update, and delete data. Similarly MongoDB Compass provides a graphical user interface that allows users to visualize their schema and perform ad-hoc queries against the database [12].

VI. COMPARISON OF DOCUMENT MODEL WITH RELATIONAL MODEL

MongoDB offers a flexible data model thus letting the easy storage of structured/semi-structured/un-structured data. No schema is required. MongoDB systems are scalable compared to relational model systems. In document model, a complete document can be accessed using a single read operation since each MongoDB document is physically stored as a single object. The ease of accessing data from an application is efficient in document based model compared to RDBMS. But in relational model we require multiple read operations from multiple locations since we need to use JOINS to access the data from multiple tables. In some cases, normalization is beneficial particularly if we need to blend the data from multiple sources for further analysis. Based on the application requirements, frequently accessed data can be stored together in a single document. When storage efficiency is considered, RDBMS systems are better than document model. MongoDB uses a technique called sharding to horizontally scale out the databases [11]. The data is distributed across multiple physical partitions (preferably in cloud), called shards. This technique is used mainly to avoid the failure of data processing due to less RAM or any other hardware limitations. Sharding causes operational problems in the case of RDBMS systems.

VII. CONCLUSION

With the advent of Big Data applications, open source software has a remarkable place in the software development industry. MongoDB, the fastest growing open source database is being used by most of the world’s giant companies like Metlife, Expedia, Facebook, Google, etc [13]. It is considered as the next-generation database. This paper focused on the data modeling using MongoDB, the document model.

REFERENCES

[1] Upeksha, Dimuthu, Chamila Wijayarathna, Maduranga Siriwardena, Lahiru Lasandun, Chinthana Wimalasuriya, N. H. N. D. de Silva, and Gihan Dias. "Comparison between performance of various database systems for implementing a language corpus." In International Conference: Beyond Databases, Architectures and Structures, pp. 82-91. Springer International Publishing, 2015.

[2] http://www.cio.com/article/2941015/big-data/solving-the-unstructured-data-challenge.html

[3] Plechawska-Wójcik, Małgorzata, and Damian Rykowski. "Comparison of Relational, Document and Graph Databases in the Context of the Web Application Development." In Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology–ISAT 2015–Part II, pp. 3-13. Springer International Publishing, 2016.

[4] Aghi, Rajat, Sumeet Mehta, Rahul Chauhan, Siddhant Chaudhary, and Navdeep Bohra. "A comprehensive comparison of SQL and MongoDB databases." International Journal of Scientific and Research Publications 5, no. 2 (2015).

[5] Mohan, C. "History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla." In Proceedings of the 16th International Conference on Extending Database Technology, pp. 11-16. ACM, 2013.

[6] https://www.tutorialspoint.com/mongodb/

[7] https://www.mongodb.com/blog/post/thinking-documents-part-1?jmp=docs&_ga=1.208290130.1389537427.1477403936

[8] Performance Best Practices for MongoDB, MongoDB 3.2 June 2016, White Paper

[9] MongoDB Operations Best Practices, September 2016, White Paper

[10] RDBMS to MongoDB Migration Guide, Considerations and Best Practices, June 2016, White Paper

[11] Top 5 Considerations When Evaluating NoSQL Databases, June 2016

[12] MongoDB Architecture Guide, June 2016, White Paper

[13] https://simplified-analytics.blogspot.com/2016_07_01_archive.html