Representing Dimensional Model in MongoDB

by

Neha Gupta

Under the Supervision of Dr. Naveen Prakash

Indraprastha Institute of Information Technology Delhi

May, 2020

Representing Dimensional Model in MongoDB

by
Neha Gupta

Submitted
in partial fulfilment of the requirements for the degree of
Master of Technology

to

Indraprastha Institute of Information Technology Delhi
May, 2020

Certificate

This is to certify that the thesis titled "Representing Dimensional Model in MongoDB" being submitted by (Neha Gupta) to the Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

May, 2020

Dr. Naveen Prakash
Department of CSE
Indraprastha Institute of Information Technology Delhi
New Delhi 110020

Acknowledgements

With immense pleasure, I am presenting this report on thesis on “Representing Dimensional Model in MongoDB” as a part of the curriculum of MTech. Computer Science and Engineering at Indraprastha Institute of Information Technology Delhi. It gives me privilege to complete this report work under the valuable guidance of Dr. Naveen Prakash.

I would also like to thank all the Staff Members of Computer Science and Engineering Department, Management, friends and my family members, who have directly or indirectly guided and helped me for the preparation of this Report and gave me an unending support right from the stage the idea was conceived.

Name: Neha Gupta
Roll No.: MT18113

Abstract

Conventional data warehouse systems are implemented either on a multi-dimensional database or a relational database. While the earlier supports MOLAP operations, the other supports ROLAP operations. In this work, we used another option to implement a data warehouse on a NoSQL database. NoSQL (Not Only SQL) systems are becoming popular due to several advantages such as elasticity and horizontal scalability. We propose and implement the rules that convert the dimensional data model to the logical MongoDB model. We show the disadvantages of ROLAP and MOLAP and advantages of MongoDB, which is a document-oriented NoSQL database.

List of Figures

List of Tables

Contents

Chapter 1
Introduction

A Data Warehousing (DW) is process for collecting and maintaining data from various sources to provide meaningful business insights. A Data warehouse is typically used to connect and analyze business data from different sources [18].

Data warehouse are implemented either by using a multidimensional database that uses MOLAP operations or a relational database that uses ROLAP operations [21]. Relational OLAP (ROLAP) servers are placed between the relational back-end server and client front-end tools. To store and manage the warehouse data, it uses relational or extended-relational DBMS [19]. Multidimensional OLAP (MOLAP) uses array-based multidimensional warehouse for multidimensional data views. Among multidimensional data stores, the storage utilization may be less in the case of the sparse dataset [20].

Since, MOLAP operations are not proficient of containing detailed data and ROLAP operations uses relational databases which have several disadvantages, another approach is to use NoSQL databases for implementing a data warehouse.

A NoSQL database (which refers to non-SQL or non-relational database before) provides data retrieval and storage mechanism that uses models (like documents and graphs), other than tables which is used in relational databases [22]. Based on the data structures used, there are mainly four types of NoSQL Database, Key-Value Store, Document-based Store, Column-based Store, and Graph-based databases.

The motivation to use NoSQL databases is to overcome the disadvantages of the relational databases. Here, we used MongoDB to represent dimensional model which is a document-oriented* NoSQL database.

The disadvantages of relational databases are [21][23]:

- Sometimes, the piece of data is not present in underlying data sources at the extraction time (ETL). In a relational database, this can be handled by using a NULL ‘value’. It creates difficulties particularly in the use NULL as a foreign key in fact tables and as a dimension value. Instead of NULL values, designers of star schema use values like -1, 0, or ‘n/a’ in dimensions. The star schemas designers have described several problems associated with these practical solutions to the problem of NULL values. For example, misinterpretation of query results. These disadvantages can be handled in NoSQL database since NoSQL database like MongoDB does not required to store NULL values.

- Now, there is a need to save and process large data volumes which the relational databases may find challenging to handle. Besides, relational databases have challenges in operating in a distributed environment. Hence, the NoSQL database is used because it does not use the concept of JOINs which leads to high system performance in the case of a large volume of data.

- Since, relational database does not handle unstructured data, audio or video files. Hence, it a disadvantage when DW is implemented by using relational database. This disadvantage is overcome by

*Explained in Section 2.6 (Other Terminologies)

the use of NoSQL databases.

- ETL for a relational implementation of a DW is a time-consuming process, because the data from diverse sources need to be converted into one conventional structured format of dimension tables and the fact. Since, structured data is not necessary in the case of NoSQL databases, ETL time will be less.

- ROLAP uses relational database, hence, supports complex joins which can be hard to understand and take too much time to execute. Therefore, performance of DW systems can be increased if implemented in a NoSQL database.

- The BASE* acronym is used to describe the properties of NoSQL databases, whereas the ACID acronym is used to describe the features of relational databases. Since data warehouse mostly provides a read-only, the analytic environment with changes restricted to ETL time, the requirement of ACID is inappropriate, and the flexibility of BASE is acceptable and, indeed, may lead to a better data warehouse performance.

In this thesis, our focus is on comparing, at the schema level, a ROLAP schema with a MongoDB schema for a given multidimensional model. Our starting point is implementing a tool for instantiating the GOM4DW model to yield a GOM4DW schema. This tool is instantiated with the case of a vegetable vendor.

Thereafter, we built a tool for converting a GOM4DW schema into ROLAP and another tool for converting the GOM4DW schema to a MongoDB representation. For the former, we used rules given in [29], and for the latter, we used the proposals of [17].

For purposes of comparison, we took the vegetable trader GOM4DW schema as input for our two tools and produced its ROLAP and MongoDB representations, respectively. The comparison was done thereafter.

The layout of the thesis is as follows. Chapter 2 describes the concepts of MongoDB, whereas Chapter 3 focuses on Background and related work. Chapter 4 describes the description of the dimensional model and implementation with the use of conversion rules. Further, Chapter 5 shows the case study and validation of the implemented tools and conversion rules. Chapter 6 shows the ROLAP schema and its comparison with MongoDB. Lastly, Chapter 7 concludes the work and describes the future work to be done.

***Explained in Section 2.6 (Other Terminologies)**

Chapter 2
Understanding MongoDB

2.1 MongoDB - Introduction

MongoDB development was started by10gen software company in 2007 as a component of a planned PaaS (platform-as-a-service) product. In 2009, the company moved to an open-source development model, with the company offering commercial support and other services. In 2013, 10gen changed its name to MongoDB Inc. On October 20, 2017, MongoDB became a publicly-traded company. On October 30, 2019, MongoDB teams up with Alibaba (NYSE: BABA) Cloud, who will offer its customers a MongoDB-as-a-service solution. Customers can use the managed offering from BABA's global data centers [3].

MongoDB is a distributed*,document-oriented*, NoSQL database which is used to handle a large amount of data in the form of documents. MongoDB is written in C++ and an open-source database. It provides high performance*, high availability*, and easy scalability*. It works on the concept of collection* and document (described in section 2.2.1) [23].

2.2 Definitions

2.2.1 Document

- Documents are semi-structured entities, mainly in a standard format such as Extensible Markup Language (XML*) or JavaScript Object Notation (JSON) [4].
- Documents are stored in a collection*, which will build up a database in MongoDB.
- Fields in a document can contain arrays and or sub-documents (sometimes called nested or embedded documents). [5]

Representation of Document: The Structure of JSON Objects

JSON objects are constructed using several simple syntax rules:

- Like key-value databases, data organized in key-value pairs.
- Documents comprise of name-value pairs separated by commas.
- Documents begin with a *{*and end with a*}*.
- Names are strings, like "user" and "products".
- Values can be numbers, strings, Boolean (true or false), arrays, objects, or the NULL value.
- The values of arrays are listed within square brackets, i.e., [and].
- The values of objects are listed as key-value pairs within curly brackets, i.e., *{* and *}*.

JSON is just one option for representing documents in a document database like MongoDB [4].

Figure 2.1 shows the representation of a document of an order.

*Explained in Section 2.6 (Other Terminologies)

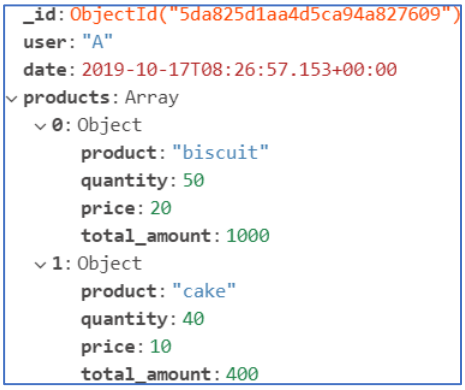


Figure 2.1: Representation of a Document in MongoDB

Document in MongoDB supports many data types. Some of them are:

- **ObjectId:** It is used to store the document's ID. Further explanation is in section 2.2.2.
- **String:** It is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer:** It is used to save a numerical value. It can be 32 bit or 64 bit depending upon the server.
- **Date:** It is used to store the current date or time in UNIX time format. We can specify our own date time by creating an object of date and passing day, month, a year into it.
- **Boolean:** It is used to store a boolean (true/ false) value.
- **Double:** It is used to store floating-point values.
- **Arrays:** It is used to store either arrays or list or multiple values into one key.
- **Timestamp:** It can be handy for recording when a document has either modified or added.
- **Object:** It is used for embedded documents.
- **Null:** It is used to store a Null value.

[6]

2.2.2 ObjectId in MongoDB

It is a 12-byte BSON type with the following structure:

- The first 4 bytes describing the seconds since the Unix epoch
- The next 3 bytes are the device identifier
- The next 2 bytes comprises of process id
- The last 3 bytes are counter value which is random

---

*Explained in Section 2.6 (Other Terminologies)

2.2.3 Array of Embedded Documents

An array of Embedded Documents is used in a one-to-many relationship where one entity is the primary document, and the many entities are described as an array of embedded documents. The primary document has fields about the one entity, and the embedded documents have fields about the many entities [4].

It can be used where all the documents are not related to each other. Example: One product's attributes (rate, quantity, etc.) are not dependent on other's product and can be retrieved independently.

In figure 2.2, "products" of an order is an array of embedded documents.

2.2.4 Embedded/Nested Documents

An embedded/nested document allows document database users to store related data in a single document. This allows the document database to avoid a process called joining in which data from one table, called the foreign key, is used to lookup data in another table [4].

It can be used where all the documents are related to each other. Example: All the subsections of a book can be dependent on each other and retrieved together as a section of a book.

In figure 2.3, the body of an email is an embedded document.

2.3 Querying

Querying in MongoDB can be done by either using find() or aggregate() function in MongoDB.

```
{
        "_id" : ObjectId("5da825eeaa4d5ca94a82760a"),
        "user" : "B",
        "date" : ISODate("2019-10-17T08:27:26.511Z"),
        "products" : [
                {
                        "product" : "candy",
                        "quantity" : 10,
                        "price" : 10,
                        "total_amount" : 100
                },
                {
                        "product" : "chips",
                        "quantity" : 10,
                        "price" : 20,
                        "total_amount" : 200
                }
        ]
}
```

Figure 2.2: "products" is an array of embedded documents

*Explained in Section 2.6 (Other Terminologies)

```
{
        "_id" : ObjectId("5d2f7be5876e05ae5d54503a"),
        "To" : "neha@gmail.com",
        "From" : "akash@gmail.com",
        "Subject" : "First Image",
        "Body" : {
                "text" : "Dear Sir, This is the image. PFA. Thanks",
                "image" : ObjectId("5d1a69f97481e73767849fee")
        }
}
```

Figure 2.3: ”body” is an embedded document

2.3.1 find()

It selects documents in a collection or view and returns a cursor to the selected documents.

Syntax: db.collection.find(query, projection)

Here, the query specifies a selection filter using query operators. To return all the documents in a collection, omit the parameter or pass an empty document (). And, the projection specifies the fields to return in the documents that match the query filter. To return all the fields in the matching documents, omit the parameter [7].

Note: Both query and projection are optional parameters in find().

Example of find() is shown in figure 2.4.

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.find({'products.product': 'biscuit' },{'_id':0}).pretty();
{
        "user" : "A",
        "date" : ISODate("2019-10-17T08:26:57.153Z"),
        "products" : [
                {
                        "product" : "biscuit",
                        "quantity" : 50,
                        "price" : 20,
                        "total_amount" : 1000
                },
                {
                        "product" : "cake",
                        "quantity" : 40,
                        "price" : 10,
                        "total_amount" : 400
                }
        ]
}
```

Figure 2.4: Result of a find query

2.3.2 *aggregate()*

It processes data records and returns computed results. It group values from multiple documents together also can perform a variety of operations on the grouped data to return a single result.

Syntax: db.collectionname.aggregate(aggregate operation)

Aggregate operation includes:

***Explained in Section 2.6 (Other Terminologies)**

- **$sum:** Adds the defined value from all documents in the collection.
- **$avg:** Computes the average of all given values.
- **$min:** Gets the minimum of the similar values from all documents in the collection.
- **$max:** Gets the maximum of the similar values from all documents in the collection.
- **$push:** It inserts the value to an array in the resulting document. Following are the possible stages in the aggregation framework in MongoDB:
- **$project:** Used to select some particular fields from a collection.
- **$match:** It is a filtering procedure; hence this can reduce the amount of documents that are given as input to the next stage.
- **$group:** It does the actual aggregation.
- **$unwind:** It is used to unwind document that are using arrays. When an array is used, the data is pre-joined, and this operation will be undone to have individual documents again. Therefore, with this stage, we will increase the number of documents for the next stage.

[8]

Example of aggregate() is shown in figure 2.5.

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.aggregate([{"$match": {"products.product": "biscuit"}}
,{"$unwind": "$products"},{"$match": {"products.product": "biscuit"}},{"$project": {"_id": 0}}]).pretty();
{
        "user" : "A",
        "date" : ISODate("2019-10-17T08:26:57.153Z"),
        "products" : {
                "product" : "biscuit",
                "quantity" : 50,
                "price" : 20,
                "total_amount" : 1000
        }
}
```

Figure 2.5: Result of aggregate query

Without $unwind and $match, the result of the aggregate is the same as find function in MongoDB in case of an array of embedded documents. An example is shown in figure 2.6.

*Explained in Section 2.6 (Other Terminologies)

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.aggregate([{"$match": {"products.product": "biscuit"}}
,{"$project": {"_id": 0}}]).pretty();
{
        "user" : "A",
        "date" : ISODate("2019-10-17T08:26:57.153Z"),
        "products" : [
                {
                        "product" : "biscuit",
                        "quantity" : 50,
                        "price" : 20,
                        "total_amount" : 1000
                },
                {
                        "product" : "cake",
                        "quantity" : 40,
                        "price" : 10,
                        "total_amount" : 400
                }
        ]
}
```

Figure 2.6: Result of an aggregate query without $unwind, and $match

2.4 Indexing

- **createIndex()** method is used to build or create an index in MongoDB. 1 is for creating the index in ascending order and -1 is for descending order [9].

  Syntax: db.COLLECTION NAME.createIndex({KEY:1})

  Example: db.order.createIndex({"products.product":1})

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.createIndex({"products.product" : 1})
{
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 2,
        "note" : "all indexes already exist",
        "ok" : 1,
        "operationTime" : Timestamp(1574344860, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1574344860, 1),
                "signature" : {
                        "hash" : BinData(0,"A9Xcjg2KUhz342oamo46c1m4xaI="),
                        "keyId" : NumberLong("6701605907681247234")
                }
        }
}
```

Figure 2.7: Create Index in MongoDB

To create an index on multiple fields, pass multiple fields in createIndex() method.

- **getIndexes()** method is used to find all the indexes created on a collection.

  Syntax: db.COLLECTION NAME.getIndexes()

  Example: db.order.getIndexes()

- **dropIndex()** method is used to drop the index in a collection.

  Syntax: db.collection name.dropIndex({index name: 1})

  Example: db.order.dropIndex({"products.product":1})

*Explained in Section 2.6 (Other Terminologies)

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "Orders.order"
        },
        {
                "v" : 2,
                "key" : {
                        "products.product" : 1
                },
                "name" : "products.product_1",
                "ns" : "Orders.order"
        }
]
```

Figure 2.8: Get Index in MongoDB

```
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.dropIndex({"products.product":1});
{
        "nIndexesWas" : 2,
        "ok" : 1,
        "operationTime" : Timestamp(1574346860, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1574346860, 2),
                "signature" : {
                        "hash" : BinData(0,"bCMaxOrlj4eul4jIFVD9sFwE8Ac="),
                        "keyId" : NumberLong("6701605907681247234")
                }
        }
}
MongoDB Enterprise MongoDB-Cluster1-shard-0:PRIMARY> db.order.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_",
                "ns" : "Orders.order"
        }
]
```

Figure 2.9: Drop Index in MongoDB

- Document databases improve on query capacities of key - value databases with indexing and the capability to filter documents based on attributes in the document [4].

2.5 Master-Slave Replication

- It is a process that enables data from one database server (the master) to be copied automatically to one or more database servers (the slaves).
- It is usually used to increase read access on various servers for scalability, although it can also be used for additional purposes like for fail-over, or analyzing data on the slave to avoid overloading the master.
- As it is a one-way replication (from master to slave), the master database is used for the write operations, while multiple slave databases are used for the read operations.
- If replication is used as the scale-out solution, you need to have at least two data sources defined, one for the write operations

---

***Explained in Section 2.6 (Other Terminologies)**

and the second for the read operations [10].

Replication in MongoDB

MongoDB uses Master-Slave Replication (Figure 2.10). MongoDB achieves replication by the use of a replica set. It is a group of MongoDB instances that host the same data set. In a replica, one node is a primary node that receives all write operations. All other nodes, such as secondaries, apply methods from the primary so that they have the same data set. A replica set must have only one primary node [1].

- All the data replicates from the primary to the secondary node.
- At the time of maintenance or automatic fail-over, election establishes for primary, and a new primary node is elected.
- After the recovery of the failed node, it again joins the replica set and works as a secondary node.

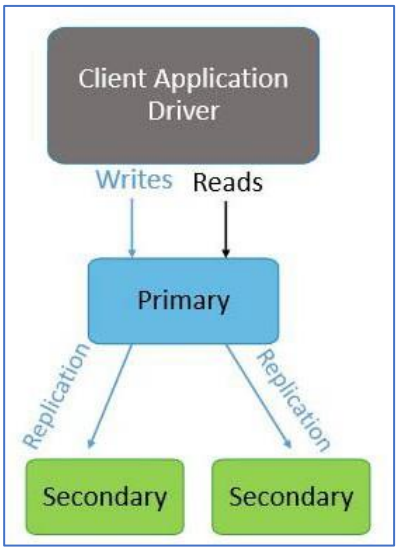


Figure 2.10: Master-Slave Replication in MongoDB (Source:[1])

2.6 Other Terminologies

2.6.1 BLOB

A BLOB is one of the datatypes used in a relational database like MySQL. It is a binary large object that can hold a variable amount of data. BLOB is the family of column type intended as high-capacity binary storage [11].

Limitations of BLOB

Following are the limitations of BLOB data type used in relational databases like MySQL:

- BLOB data columns cannot be part of an index as in figure 2.11 [12].

*Explained in Section 2.6 (Other Terminologies)

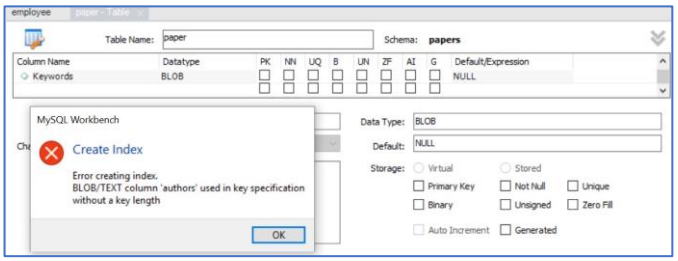


Figure 2.11: No index can be created on BLOB

- BLOB data columns cannot be part of a primary key as in figure 2.12 (here ”idproduct” is a primary key).

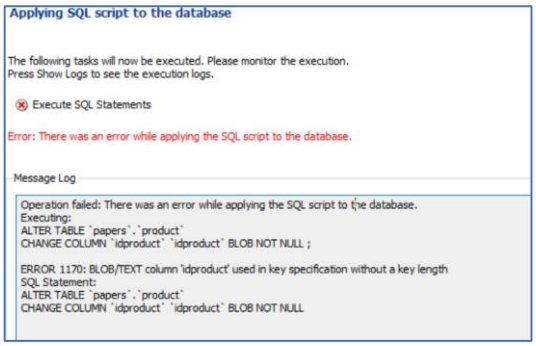


Figure 2.12: No primary key contains BLOB data type

Because of the limitations related to indexing in a BLOB, search query takes a lot of time in case of extensive data used in applications like commercial applications or social networking websites. It can overcome by using BSON type documents in MongoDB.

2.6.2 Distributed Database

Multiple NoSQL databases (like MongoDB) can be executed in a distributed manner. Hence, MongoDB is a distributed database. It offers auto-scaling and fail-over capabilities.

2.6.3 Document-Oriented Database

It retrieves and stores data as a key-value pair, but the value part is saved as a document. The document is stored or saved in JSON or XML* formats [2].

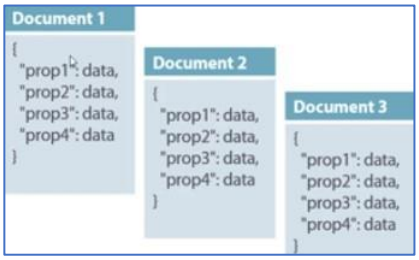


Figure 2.13: Document (Source:[2])

*Explained in Section 2.6 (Other Terminologies)

2.6.4 High Performance

It is based on two categories:

- **Increase Query Performance:** It uses BSON instead of JSON for storing data. BSON is a binary JSON. Being Binary it improves the processing efficiency of the MongoDB. It uses B Tree indexes, so they are sorted.
- **Increase Scale Performance:** Replica sets enable data availability at all times by having copies of data at multiple replicas. Sharding* as well helps to store large amounts of data.

2.6.5 High Availability

It indicates a system designed for redundancy, durability, and automatic fail-over so that the applications supported by the system can work continuously and without downtime for a long period. MongoDB replica sets support high availability when deployed according to the best practices in MongoDB.

- Replica sets use elections to support high availability.
- Elections of replica set occur when the primary becomes unavailable, and the replica set members autonomously select a new primary.

2.6.6 Easy Scalability

You can scale out your deployments quite efficiently, i.e., additional nodes can easily be added to the deployment to share or distribute data between them so that all data need not be saved in one node. It can be done by Sharding in MongoDB.

2.6.7 Collection

- In MongoDB, databases contain collections of documents.
- Collections are similar to the tables in relational databases.
- MongoDB creates the collection when data is stored for that collection for the first time if a collection does not exist. [13]

Figure 2.14 shows a MongoDB collection.

*Explained in Section 2.6 (Other Terminologies)

Figure 2.14: Collection in MongoDB

2.6.8 XML vs MongoDB Document

The central idea of a document-oriented database is the notion of a document.

Document stores allow different types of documents in a single store, let the fields within them to be optional. For example, a document encoded in JSON:

{
”FirstName”: ”Neha”,
”LastName”: ”Gupta”,
”Address”: ”IIIT Delhi, New Delhi”,
”Hobby”: ”research”
}

A second document might be encoded in XML as:

⟨contact⟩
⟨firstname⟩Neha ⟨/firstname ⟩
⟨lastname⟩Gupta⟨/lastname⟩
⟨address⟩
⟨type⟩College ⟨/type⟩
⟨addressline1⟩IIIT Delhi ⟨/addressline1⟩
⟨city⟩New Delhi ⟨/city⟩
⟨/address⟩
⟨/contact⟩

The above documents share some structural elements with one another, but each also has unique elements. Both XML and MongoDB documents follow the hierarchical structure.

***Explained in Section 2.6 (Other Terminologies)**

2.6.9 BASE Property

- **Basically available** means that the system does guarantee availability.
- **Soft state** means that the systems state may change over time, also without input.
- **Eventual consistency** symbolizes that the system will become consistent over time, given that the system doesn't receive input during that time [2].
- Example: Shopping Carts like Amazon, Flipkart, etc. follows the BASE property.

2.6.10 Sharding

- Another term for horizontal partitioning of data is sharding. It is the process of storing data records across multiple machines. Example: A Customer table has 100 rows. To shard it across four servers, we use 'key' based sharding in which customers will be distributed as follows: SHARD-1(1-25), SHARD-2(26-50), SHARD-3(51-75), SHARD-4(76-100)Sharding splits the data set and shares them over multiple databases, or shards. Sharding can be done in 2 ways: [5]
    1. **Hash-based:** MongoDB determines the hash of a fields value first and then creates chunks using those hashes.
    2. **Key-based:** Choose a key from a collection and divide or split the data using the keys value to deploy sharding in Mongo-DB. This key is known as shard key that determines how to distribute the documents of a collection among the different shards in a cluster. [23]

***Explained in Section 2.6 (Other Terminologies)**

Chapter 3
Background and Related Work

We have studied about the various approaches used for the data warehouse implementation on the NoSQL platform. Similar to the name ROLAP and MOLAP, some researchers used as the concept of NOSOLAP [21]. The work done till date is either on column-oriented databases and document-oriented databases. These are as follows:

- Chevalier and other researchers [24] converted the schema to both column-oriented (HBase) and document-oriented(MongoDB) NoSQL model. They used star schema for mapping rules which consist of only facts and dimensions. In the former model, each fact is mapped with measures as the columns to a column family. And, each dimension and dimension attributes are mapped to separate column families and columns in the respective column families. These families make a table which represents a single star schema. Whereas in the latter, each fact and dimension is a compound attribute consists of the measures and dimension attributes as simple attributes. A fact is considered as a document, and the measures are within this document. Chevalier and others [26] extended their work to only document-oriented NoSQL model. They mapped the rules directly from the multi-dimensional data model to document-oriented NoSQL model (instead of the multi-dimensional data model to the relational model to document-oriented NoSQL model) using three approaches and compare these approaches concerning performance.
- Dehdouh1 and others [25] uses a column-oriented NoSQL model to implement a data warehouse. They also used star schema as in [24] but used three approaches which differ in terms of structure and its attribute types.
- R. Yangui and others [27] has done a comparative study of the transformation of data warehouse schema to NoSQL database. Like [24], they also worked on column-oriented as well as document-oriented data models. Besides, they proposed two transformations, i.e., simple and hierarchical transformations. They used facts and dimensions in simple transformation, whereas the other transformation uses hierarchies with facts and dimensions.
- According to D. Prakash [21], the model with only facts and dimensions have two limitations related to aggregate functions modelling and recording the history in a star schema. Hence, the Information model is proposed where information can either be detailed, aggregate or historical. This model focuses on details of information instead of just facts and dimensions. Here, the researcher worked on Cassandra model, which is a column-oriented NoSQL database.

To overcome the limitations of the above models according to [21] for a document-oriented logical model, we used the rules to convert the dimensional model (as described in section 4.1) to the document-oriented logical model. In this work, we used MongoDB as the document-oriented model.

The implementation is done in two phases:

- Save the GOM4DW model in the database (MSSQL server).
- Retrieving the GOM4DW model from the database and then converting to MongoDB model using the rules described in section 5.1.

Chapter 4
Generic Object Model For DW

In this chapter, we describe the GOM4DW model and then show the implementation of our tool to instantiate it.

4.1 Components of GOM4DW

Components of the model are described as shown in the figure 4.1.

1. **Data Object:** It is also referred to as fact in dimensional modelling. A fact table is a primary table in a dimensional model [14]. If the data object is contained in another data object, then it may be treated as a dimension.
2. **Attribute:** Non-key columns are commonly referred to as attributes. The dimension table comprises of attributes associated with the dimension entry. These are rich and business-oriented textual details, such as customer name or product name [15].
3. **Category:** A category or dimension is a structure that categorizes data in order to enable users to answer business questions. Data objects can be categorized by categories. Example: Orders dimension can be categorized by product, location, etc.
4. **Category Attributes:** Category also comprises of category attributes like attributes of data objects. Category attributes have change types: Type 1 (Correction of an error in source systems / Overwriting the old value), Type 2 (A change to preserve old data / Creating a new additional record) and Type 3 (Soft tentative change, ‘what if’ change / Adding a new column).
5. **History:** History of data objects can be maintained on the basis of period/duration and frequency.
6. **Aggregates:** These are used in dimensional models of data warehouse to reduce the time it takes to query large sets of data. An aggregate is a summary table that can be derived by performing a Group by SQL query [16]. These are a powerful tool for improving query processing speed in dimensional data marts [15].

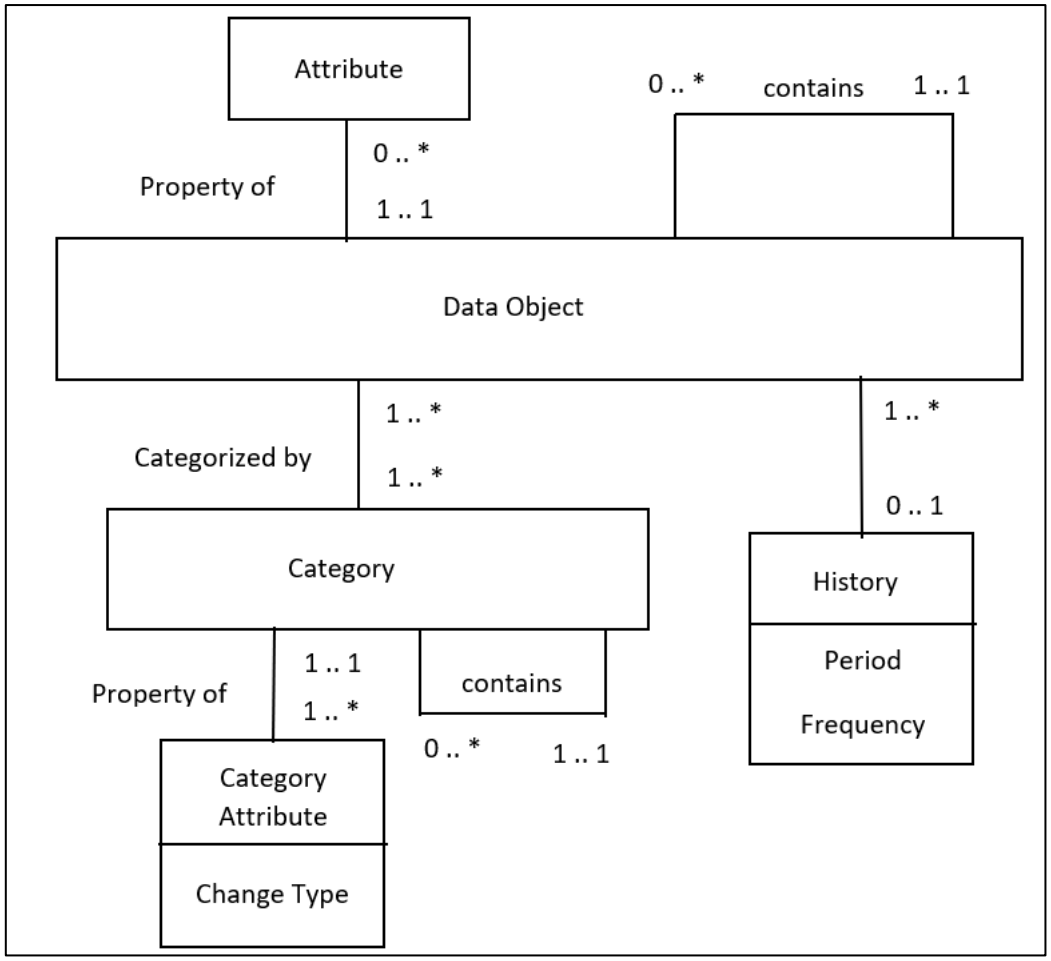


Figure 4.1: GOM4DW Model

Data object can be simple or aggregate and categorized by category as shown in the figure 4.2.

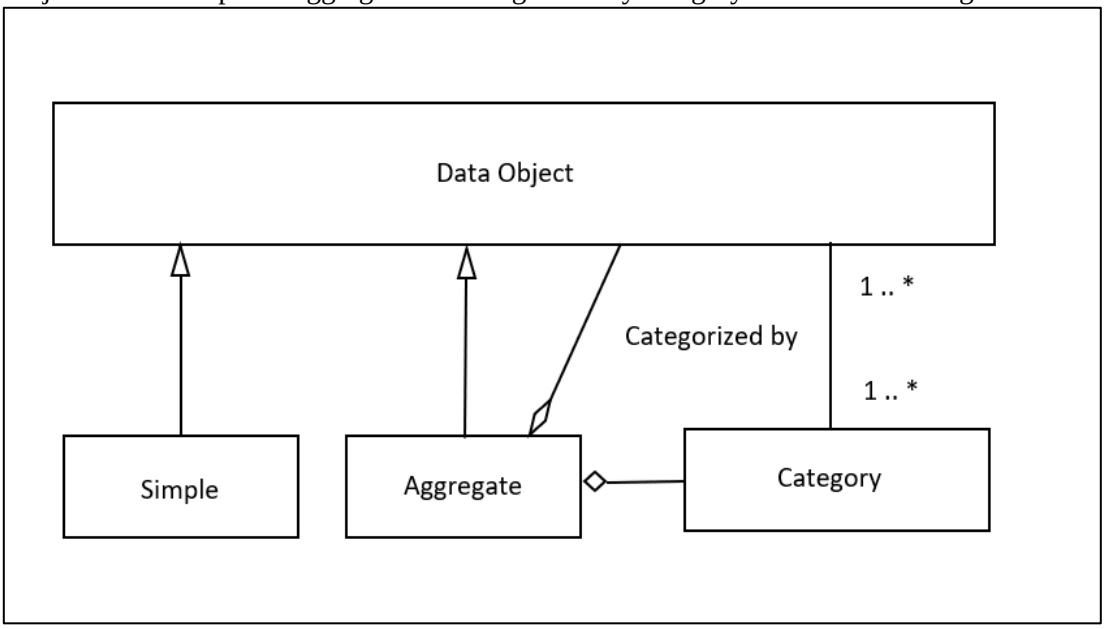


Figure 4.2: Data Object can be simple or aggregate

4.2 Implementation

The following screenshots provides implementation of the tool and user interaction.

1. The below screenshot shows the first screen of the tool. A new project can be registered using the 'New User' button where as an existing one can be logged in using login button.

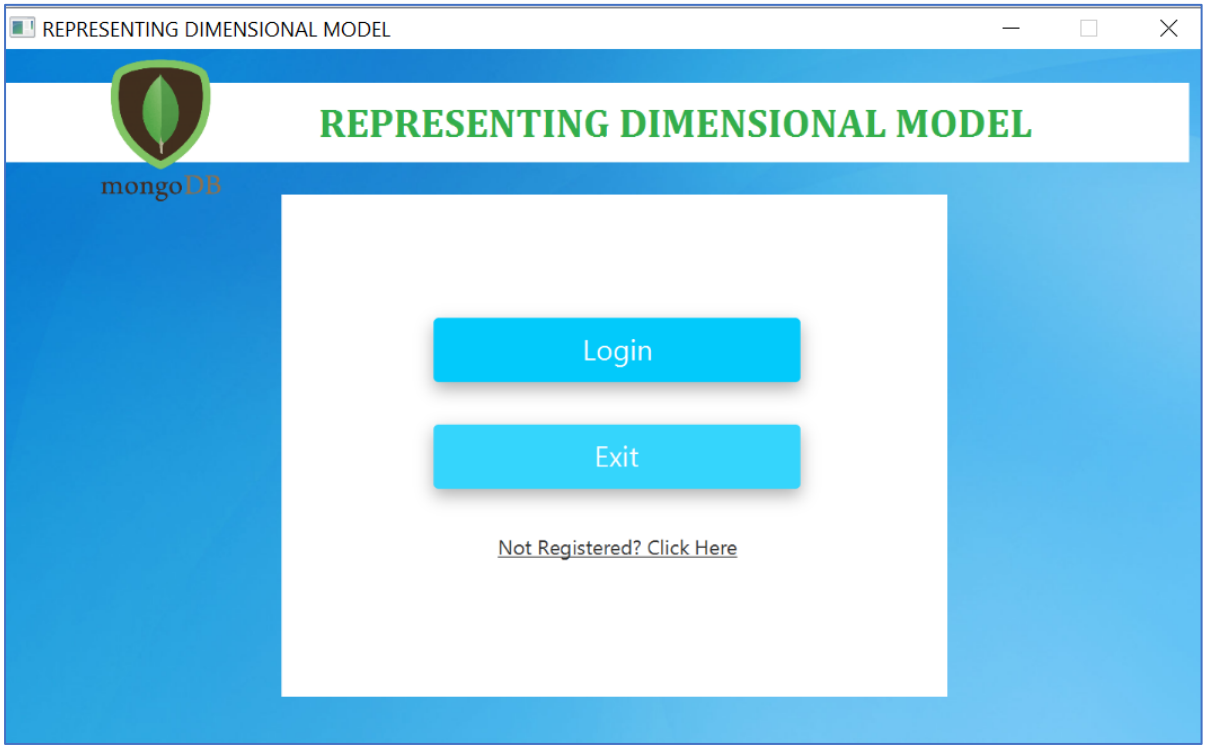


Figure 4.3: Login/Sign Up Screen

2. On Signup, the tool creates a unique project id which will be used to login in future sessions to work on that project. On pressing next button, the options menu will be displayed.

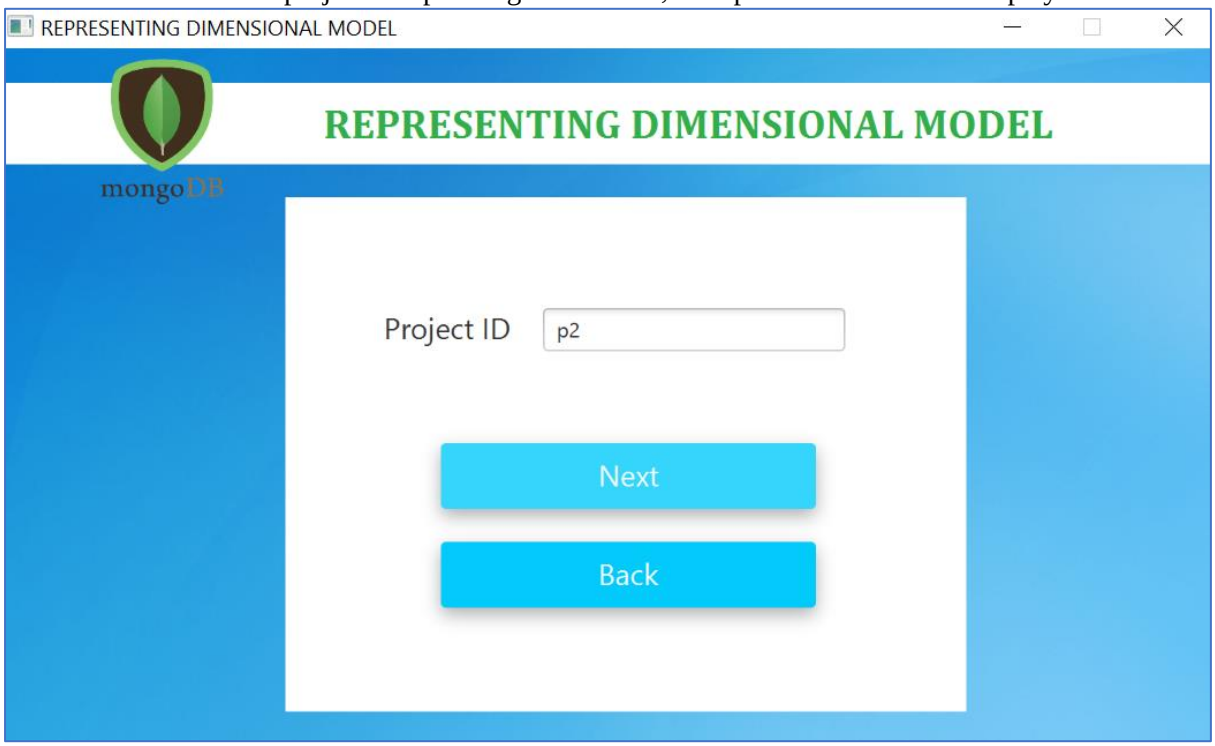


Figure 4.4: Unique Project Id generated on sign up

3. By clicking the login button in figure 4.2, the user will be taken to login screen as shown in figure 4.5. For logging in, the project id generated at time of registration is to be used.

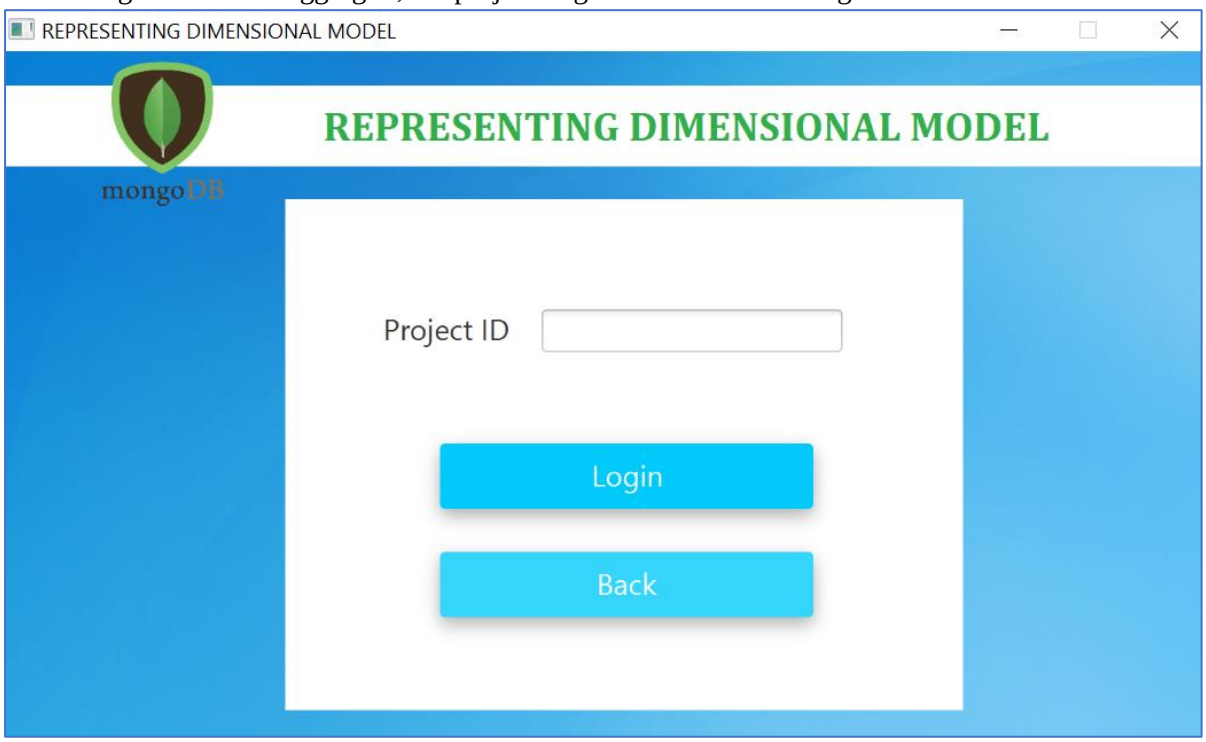


Figure 4.5: Login Screen

4. Figure 4.6 shows the option menu for the user. The project name gets loaded automatically from sign up/login details. The options menu consists of basic functionalities i.e. create, modify and view a data object. The user can choose which data object to view by selecting from the drop down placed alongside the button 'View Information'. Further the user can start conversion process by clicking 'Proceed to make MongoDB Model'.

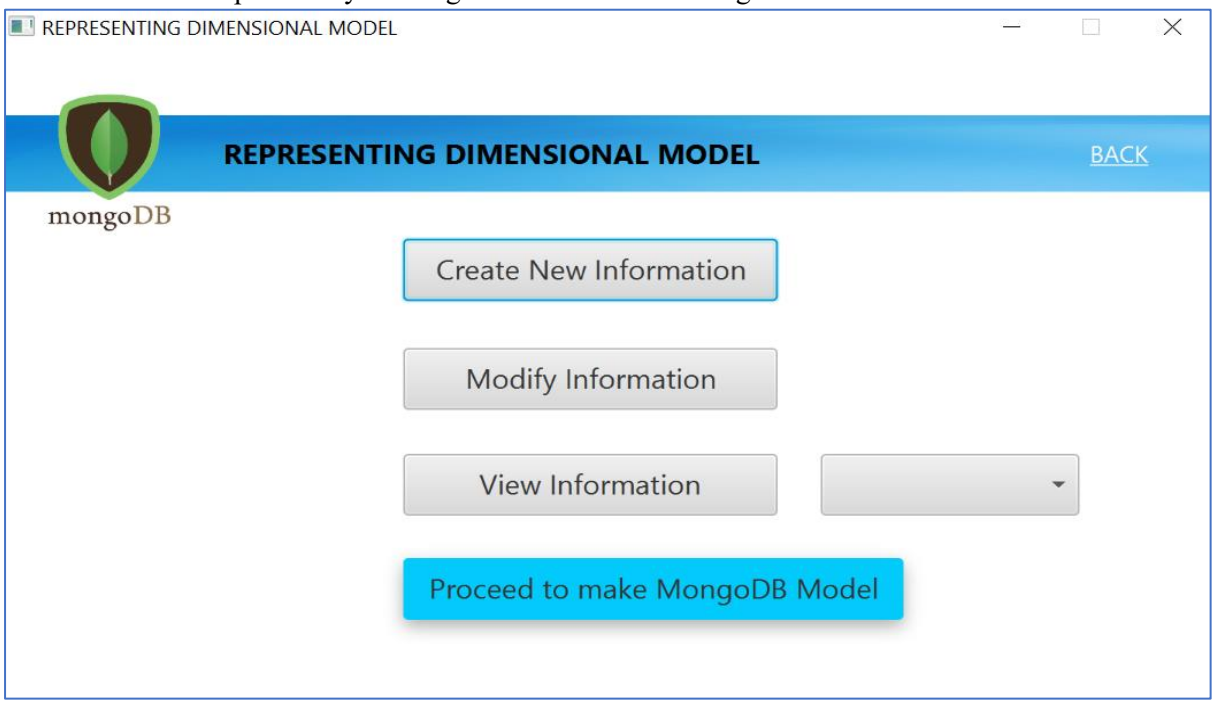


Figure 4.6: Options for User

5. To instantiate the dimensional model, user will click on ‘create new Information’ button and can enter the required details. Figure 4.7 shows the example: Orders.

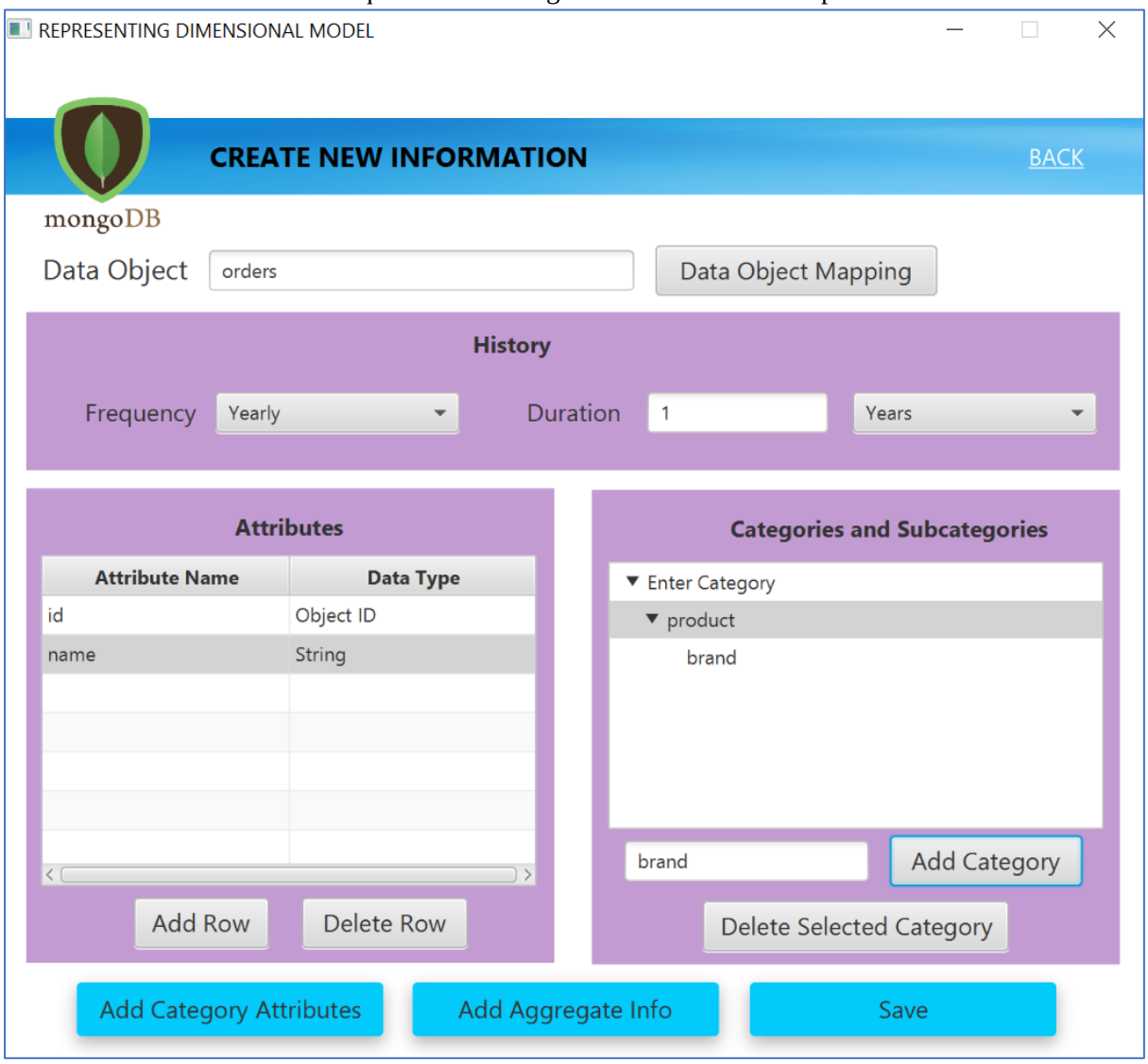


Figure 4.7: Instantiation of Data Object

6. Figure 4.8 shows the category attributes that user can enter by clicking on 'Add Category Attributes' button.

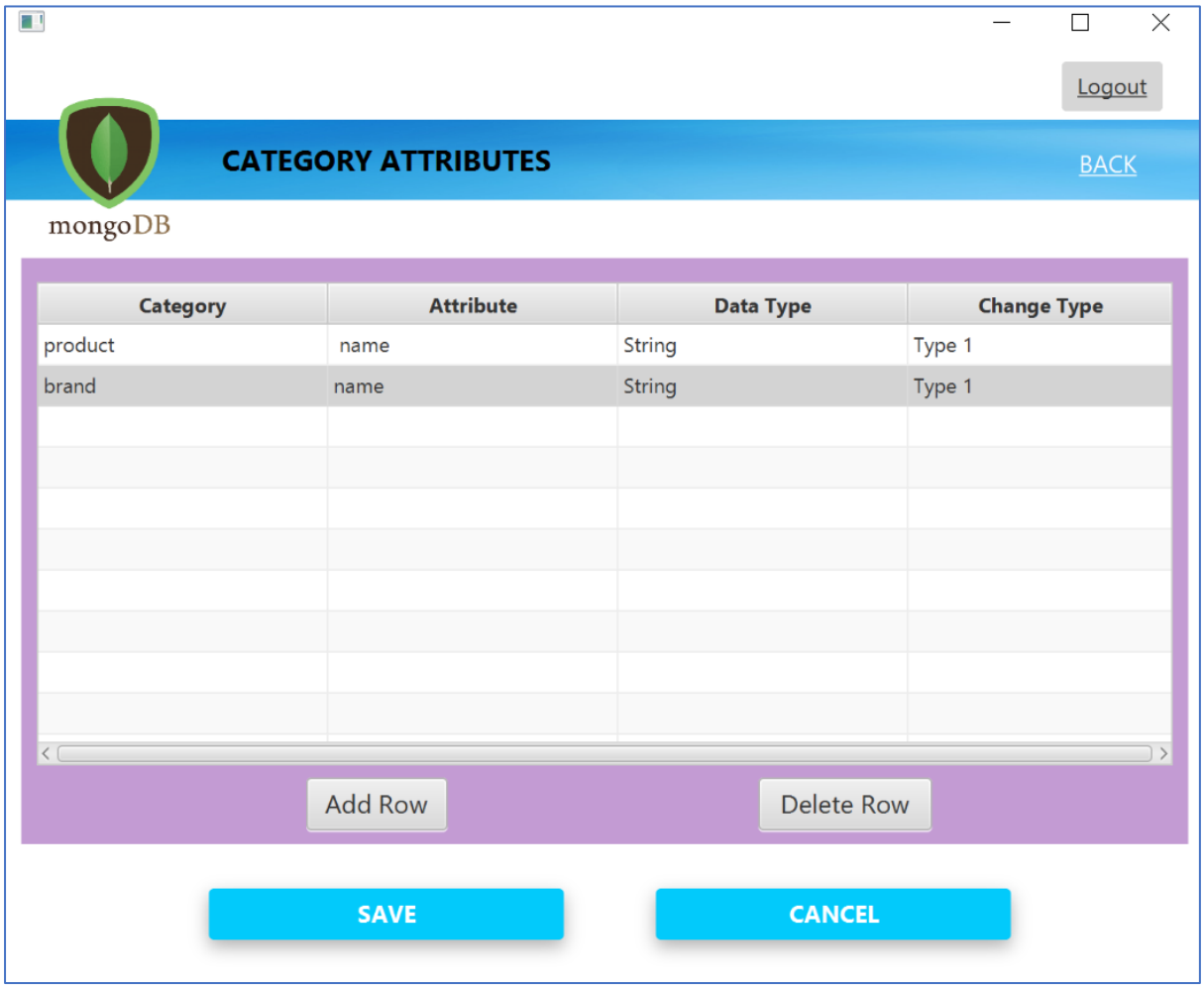


Figure 4.8: Category Attributes

7. Figure 4. 9 shows the mapping of data objects that user can enter by clicking on ‘Data Object Mapping’ button.

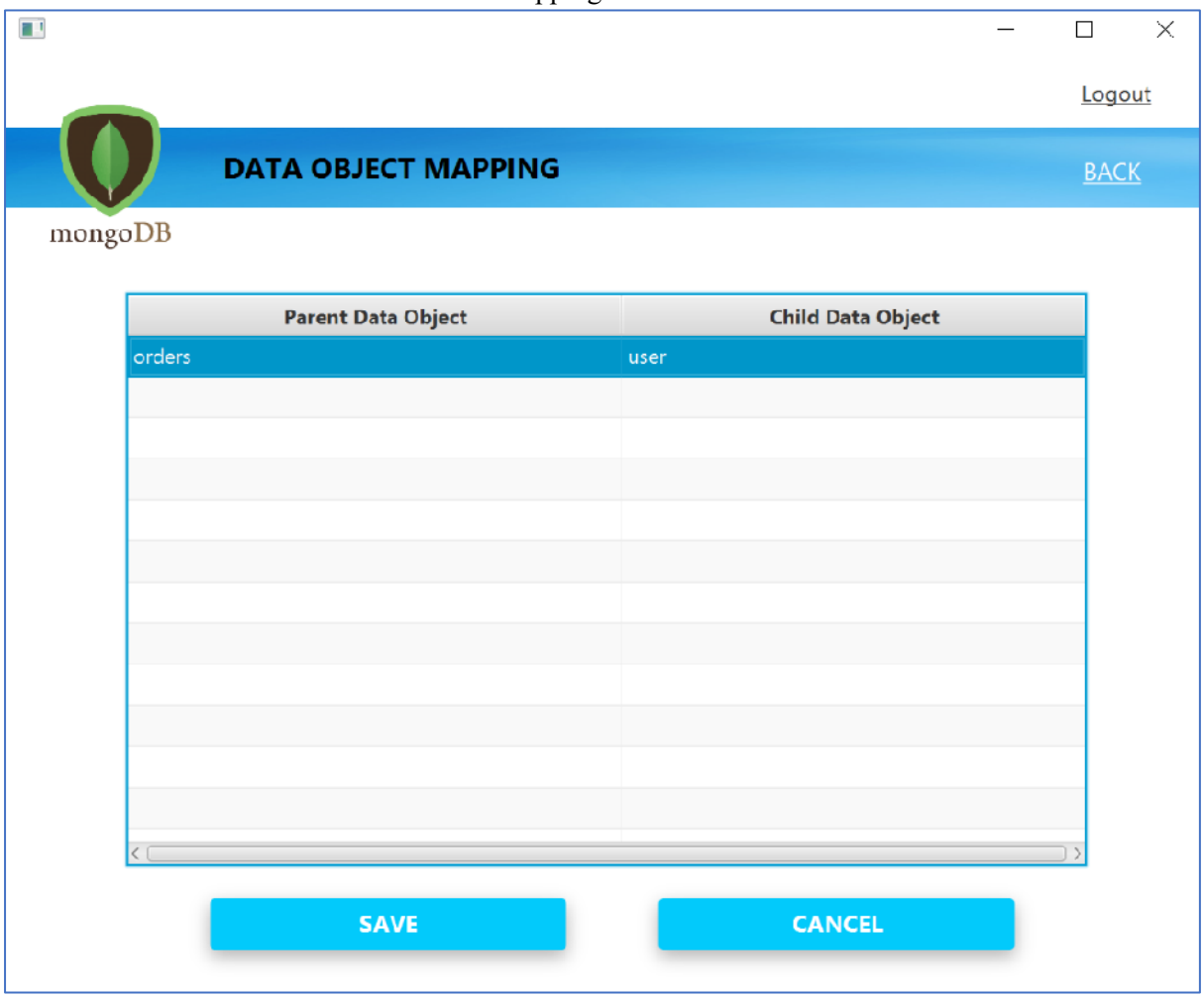


Figure 4.9: Data Object Mapping

8. Figure 4.10 shows the aggregate information user enters by clicking on 'Add Aggregate Info' button. This functionality is optional for the user.

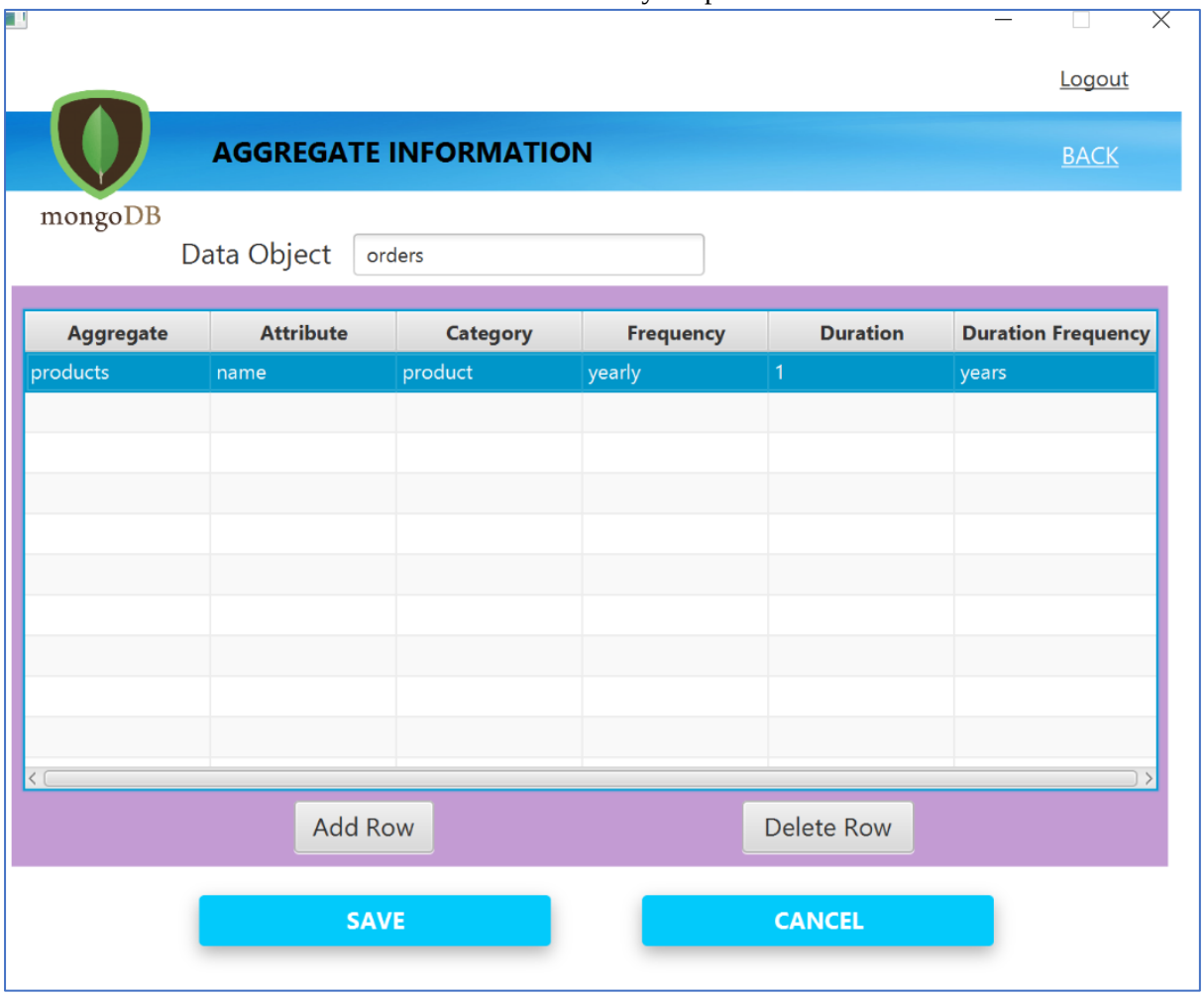


Figure 4.10: Aggregate Information

9. After pressing save button, on 'create new information' window, the user will be asked for confirmation for the details entered before saving the same to the database in figure 4.11. To save the user inputs, MS SQL Server is used.

Figure 4.11: Confirmation Screen

10. The user can modify the existing information in the project by clicking 'Modify Information' button from the options Menu. Figure 4.12 shows example of modification screen where duration is changed from 1 to 2 years. The user can choose from drop down which data object he / she wishes to modify.

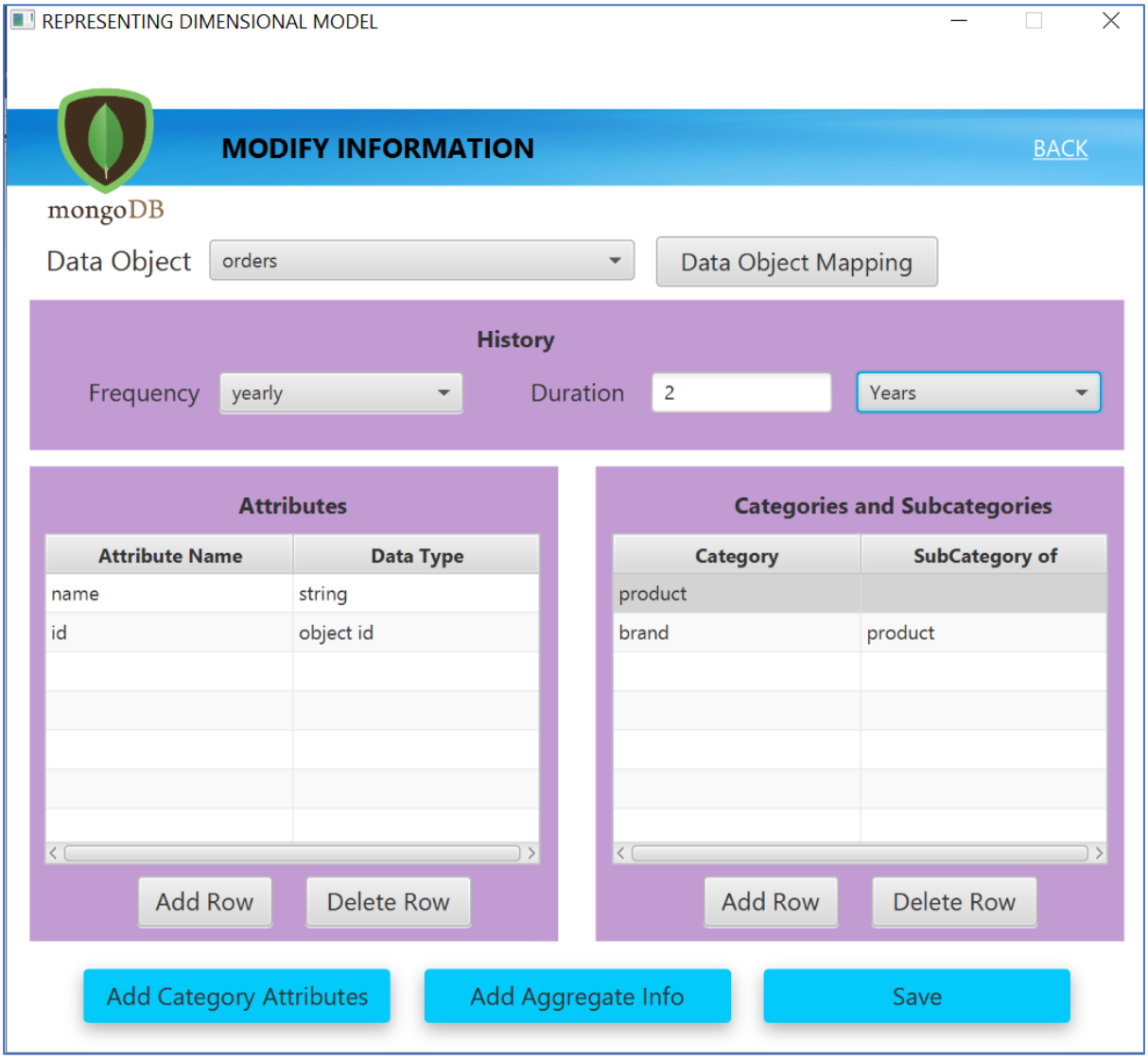


Figure 4.12: Modify Information

11. The user can view the existing information by selecting the data object from drop down clicking on 'View Information' button from the options Menu. Figure 4.13 shows example of view screen.

Figure 4.13: View Information

12. Figure 4.14 shows to model the data object using the conversion rules as discussed in the section 3.1.3. On clicking "Save" button, a data object is saved according to the rules, and, on clicking "Model Data Object" button, the rules will be saved in the JavaScript (.js) file.

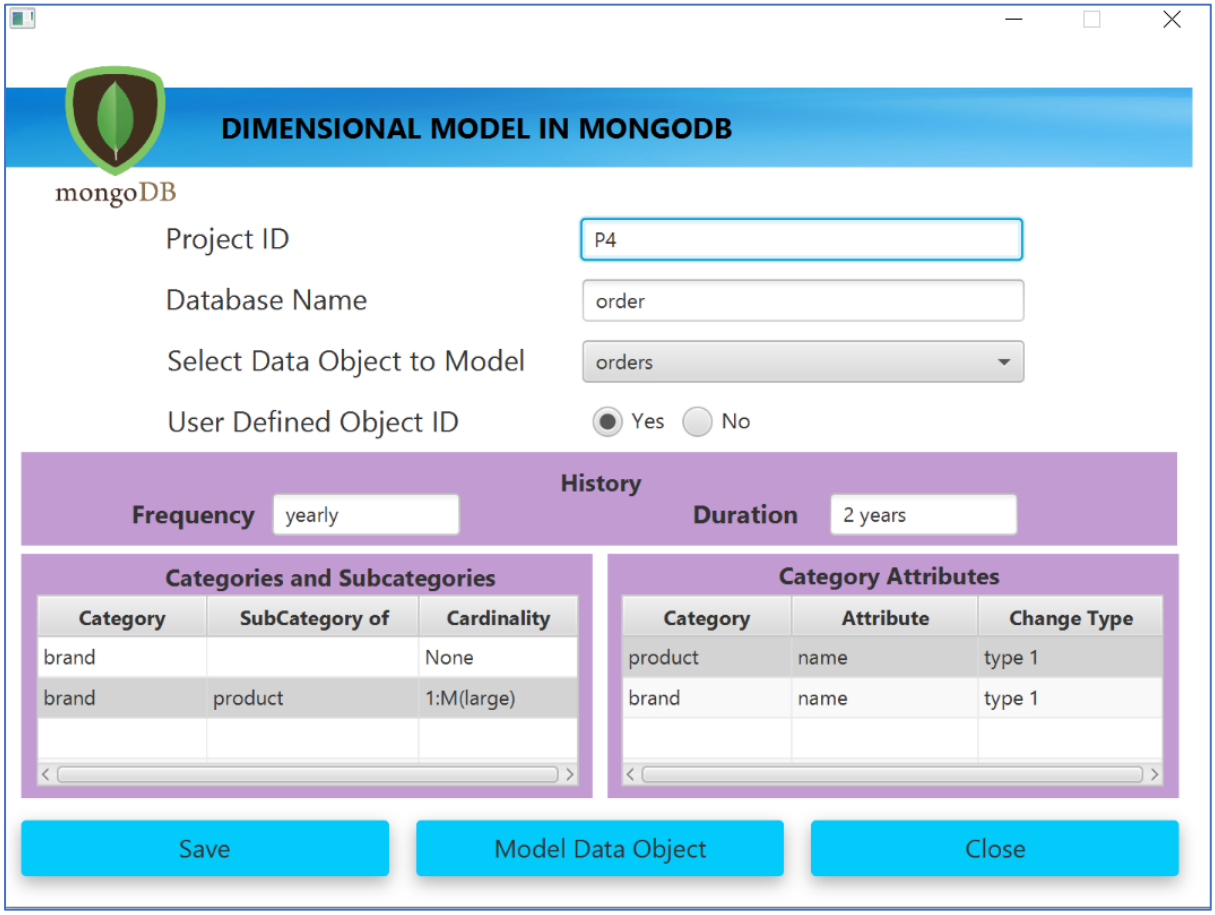


Figure 4.14: Model Data Object Using Conversion Rules (Section 3.1.3)

Note: MongoDB commands are saved in JavaScript(.js) file, since, all the queries will run sequentially at once using JavaScript file using the following command after starting mongo shell.

mongo localhost:27017 "dbScripts/P5_withvalues.js"

Chapter 5

In this chapter, we show the conversion rules for converting from GOM4DW schema to a MongoDB representation. The implementation produces a JavaScript file containing the script that is processed to yield the MongoDB database for the given GOM4DW schema.

5.1 Conversion Rules to Represent GOM4DW Schema in MongoDB

The following rules are described below to convert a GOM4DW schema in MongoDB [17].

Rule 1: Create a Database for the GOM4DW schema at hand.

```
use vegetable_trader;
or db = db.getSiblingDB('vegetable_trader');
```

Rule 2: For every data object, O that is not contained in another data object, create a MongoDB Collection, C.

```
db.createCollection("vendor_revenue")
```

Rule 3: Every attribute of O is a field of C.

```
db.vendor_revenue.insertOne({
cost:
,quantity:
```

Rule 4: For every data object, o that is contained in O, create an embedded document in C. The attributes of o will become the fields of the embedded document.

Rule 5: For every category object, CO, create an embedded document in C. The attributes of CO will become elements of the embedded document.

```
,Product:{ SKU: "P11", name: "Potato"}
,Vendor{ID:"V11", Name: "ABC"}
,DateP{ datePurchase:  new Date("18-04-2020 11:40")}
,Quality {number: 5}
```

Rule 6: For every category, co, contained in CO, two cases arise:

Case 1: *If the cardinality is 1:1*: Create an embedded document in C with the attributes of co as the elements of the document.

Case 2: *If the cardinality is 1:M where M is a few (<200):* Create an array of embedded documents in C with each document representing an instance of co. The attributes of co will become elements of the embedded document

Case 3: *If the cardinality is 1:M where M is large:* Create a separate document for instance of co with the attributes as elements of the document. Create an array of the ids as link/references to C

Case 4: *If the cardinality is 1:M where M is humungous: To be omitted*

```
,Product:{SKU: "P11", name: "Potato" , pdtType: [{typeid: 1, name:
"Root"}, { typeid: 99, name: "all round year"}]}
,Vendor:{ID:"V11", Name: "ABC"}
,DateP{ datePurchase: new Date("18-04-2020 11:40")}
, Quality:{number: 5}
```

Rule 7: If change_type = update then do nothing
If change_type = no_update then

If Case 3 of Rule 6 is selected: then

1. denormalize the attribute with change_type = no_update into the array
2. add an element CreationTime to C and set the value to current system timestamp

else

add an element CreationTime to C and set the value to current system timestamp

,CreationTime: new Date()

Rule 8: To capture Period, add a new field to the C, Duration. Create a TTL index on the field Duration. Two possibilities exist

a. Set the value of Duration to current timestamp. Convert the value of the attribute, period, to seconds. Set the the *expireAfterSeconds* of TTL index to this value

OR

b. Set Duration = Period + current time. Example 5 years will be 14/4/2025 21:00:00. Set the value of Duration to this value. Use *expireAfterSeconds* = 0.

,Duration: new Date()}
)
db.vendor_revenue.createIndex({Duration:1}, { expireAfterSeconds :3600})

Rule 9: Define a surrogate attribute for every category object, C, and add it to the respective embedded document.

,Product:{PdtKey: 11,SKU: "P11", name: "Potato" , pdtType: [{typeid: 1, name: "Root"}, { typeid: 99, name: "all round year"}]}
,Vendor: {VendorKey:1, ID:"V11", Name: "ABC"}
,DateP:{ DateKey: 1 datePurchase: new Date("18-04-2020 11:40")}
, Quality:{GradeKey:5, number: 5}

Rule 10: Define separate _id for each document C.

,_id : 47827703

Points to be considered:

- **Normalized Data Models:** Include a links/references in another document. Example, User has its _id. Include this _id as a field in say contact document.
- **De-normalized/Embedded Data Models:** Effective if cardinality is few (< 200) not many (>200). Or else document may become too large. Then better to normalize this part. If cardinality is very large (>few thousands) don't use array of object ids.
- **Disadvantages of embedded document:** Cant access the embedded document as a stand-alone.
- **Capped collections:** They are good if mainly reads required. Also, indexes can be added.

Note: The converted file (.js file) is only used for the testing purpose since it does not contain the data to be stored. The script has to be suitably edited at ETL time to include the data so it can produce the desired MongoDB multidimensional form.

Chapter 6
Case Study

6.1 Schema Used

The GOM4DW schema of a vegetable trader is represented in MongoDB. The schema is shown in Table 6.1.

Table 6.1: Vegetable Trader Schema

No.	Category Objects	Simple Data Objects	Aggregate Objects	Categories over which aggregated	History
1	Vyapari(name, address, telephone), Commission Rate(rate percent), Date	Required Re-Stock(Quality required, Quantity required, Price, SGST, CGST, Transport cost, Storage cost)			
2	Location(address), Date	Storage Capacity(Used, Vacant)			
3	Market(Name), Day, Quality(quality rating)	Market Price(Quantity in Market, Quantity sold in Market, Price in Market)			Daily, 2 years
4	Vyapari, Date	Agreement to sell (Quality agreed, Quantity agreed, Price agreed, Commission Rate agreed)			Each agreement, 5 years
5	Vyapari, date	Business Ease(easy quality change, easy new quantity, easy new price)			
6	Vyapari, Agreement(id), delivery date	Performance(quality agreed, quality supplied, quantity agreed, quantity Supplied, punctual supply, delivery lead time)			Each performance, 5 years
7	Vyapari, date	Delivery time agreement (Quality agreed, Quantity agreed, Delivered Quality, Original Price agreed, Delivery time price agreed, Commission Rate agreed)			Each delivery, 5 years
8	Vyapari, Day →	Daily Agreement (Quality, Quantity, Price, Commission Rate)			Daily, 5 years
9	Customer(Name, address),	Sale(Quantity, Amount)			

	date, Quality, Mode(cash, credit)				
10	Market, day	Market Price(Quality, Price)			
11	Vyapari, date	Discarded Stock(Quantity thrown)			Daily, 2 years
12	Vyapari, Customer, date, Quality	Order(Quantity, Amount)	Net Amount	Quality, Quantity	
13	Vyapari, day, Quality	Stock in hand(Total Quantity in hand)			
14	Date	Daily Earning(Quantity, Selling Rate, Commission Rate)			
15	Vyapari, day	Agreed commission(Quality agreed, Quantity agreed, Price agreed, agreed commission rate)			
16	Quality, date	Daily Sales(Quantity, Price, Discount)			
17	Customer, day	Commission due(quantity, price, commission amount)			

6.2 Validation of Conversion Rules

Applying the rules followed to the part the schema used in the section 6.1, we can create the database named "market" using the tool based on Rule 1 as shown in Table 6.2.

Table 6.2: Database created after applying Rule 1

```
db = db.getSiblingDB('market');
```

As shown in Table 6.3, Rule 2, 3, 5, 7, 9, 10 are applied for data object "performance" as there is no data object containment and sub-category. Here, quantity supplied, delivery lead time, quality supplied, quality agreed, punctual supply, quantity agreed are the data object attributes; vyapari(name), agreement(id), delivery date(date) are the category (and category attributes). According to Rule 8, index is created.

Table 6.3: Collection created and MongoDB queries for insertion of data for "performance" data object

```
db.createCollection("performance");
db.performance.insertOne({
_id:,
quantity_supplied:,delivery_lead_time:,quality_supplied:,quality_agreed:,
punctual_supply:,quantity_agreed:,vyapari:{vyapari_key:,name:},agreement:{agre
ement_key:,id:},delivery_date:{delivery_date_key:,date:},CreationTime: new
Date(),
Duration: new Date()});
db.performance.createIndex({Duration:1}, { expireAfterSeconds :157680000});
```

As shown in Table 6.4, Same rules are applied except Rule 8 for "order" data object as in Table 6.3. In addition, aggregate information "Net amount" is added with amount as attribute and quality as category.

Table 6.4: Collection created and MongoDB queries for insertion of data for "*order" data object*

```
db.createCollection("order");
db.order.insertOne({
_id:,
amount:,quantity:,date:{date_key:,date:},vyapari:{vyapari_key:,name:},customer:{custome
r_key:,name:},quality:{quality_key:,number:},Net_Amount:{amount:,quality:{quality_key:,
number:}},CreationTime: new Date(),
Duration: new Date()});
```

Note: The output file and the output file with values are described in Appendix A and Appendix B respectively.

Chapter 7
GOM4DW to RDBMS

We have seen the conversion rules [17] for converting from GOM4DW schema to MongoDB in the last chapter. Now, we present the rules used for converting the GOM4DW schema to a ROLAP schema [29]. These rules were used to build a conversion tool.

7.1 Algorithm

The following algorithm is used to convert GOM4DW schema to RDMS [29].

Algorithm: Conversion to multi-dimensional schema

Input: Instantiation of GOM4DW
Output: Snowflake schema for I

```
1.  for Each data object, I not contained in another data object do
2.      F:= createfact(I)
3.      for Each Attribute, A, of I,
4.          addAttribute(A, F);
5.      end for
6.      for Each data object, O, that contains other data objects do
7.          D:= createDimension(O);
8.          for Each attribute, A, of O do
9.                  addAttribute(A, D);
10.         end for
11.      Link D to F;
12.     end for
13.     for Each category, C do
14.         D:= createDimension(C);
15.         if D does not exist then
16.             for Each attribute, A, linked to C do
17.                 addAttribute(A, D);
18.             end for
19.             if change type=no_update & timestamp does not exit then
20.                     Add timestamp field to D
21.             end if
22.             Link D to F;
23.         end if
24.         else Link already existing D to F and discard current D
25.         for Each category, cc, contained in C do
26.             SD = createSubDimension(cc)
27.             if SD does not exist then
28.                 for Each attribute, A, linked to cc do
29.                     Add A to SD as a Dimensional attribute
30.                 end for
31.                 if change type=no_update & timestamp does not exit then
32.                     Add timestamp field to D
33.                 end if
34.                 Link SD to D
35.             end if
36.             else Link already existing SD to D and discard current SD
37.         end for
38.     end for
```

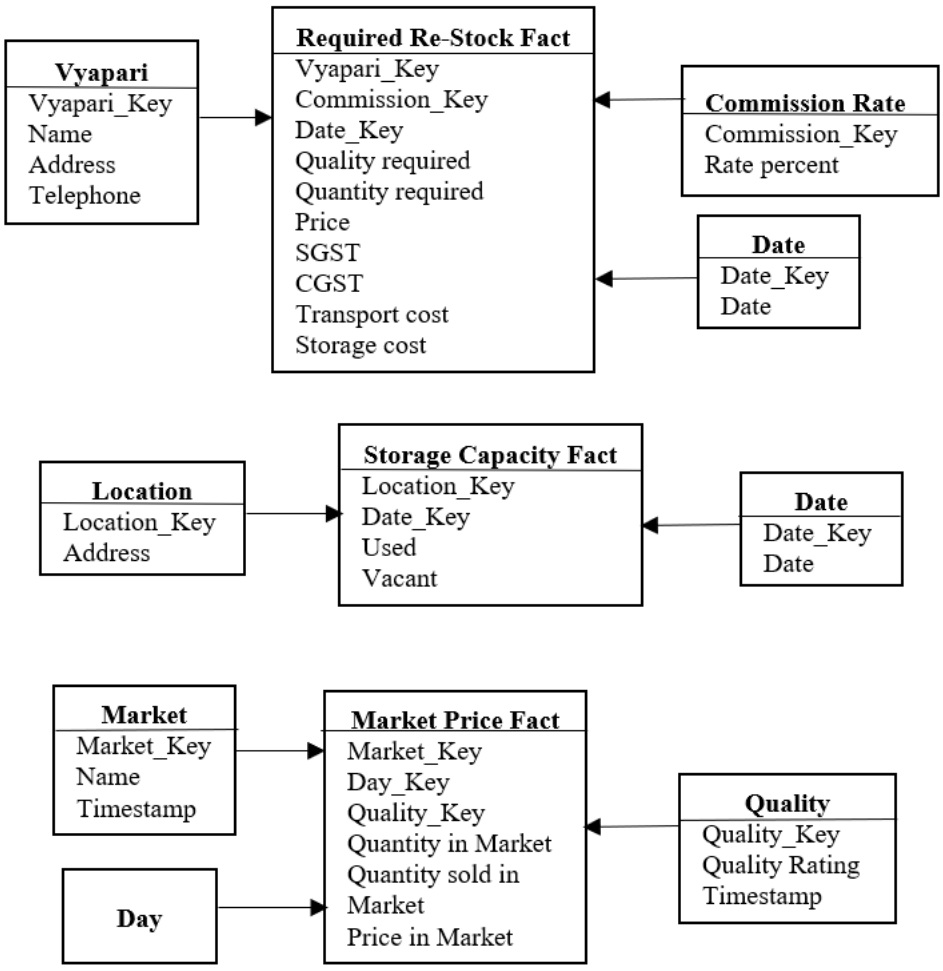
If Duration of history = *n units of time* is to be recorded then there is no provision for this in the above schema and the task is performed at ETL time. For this:

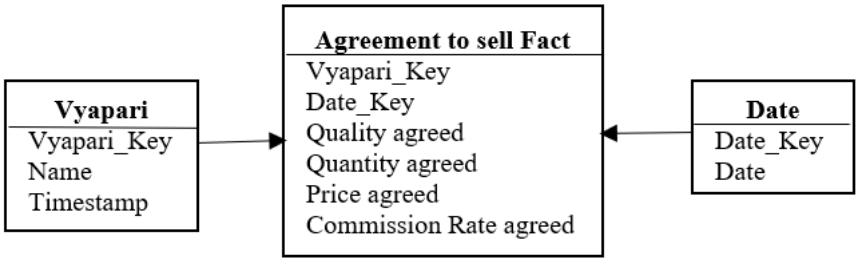
1. Define starting calendar year, month and day of n. For example: if n=5 years, then start date of recording data can be 1/1/2010.
2. After n *units of time* are completed, then expired data is removed from the DW as an additional step in ETL. So, in our example, on 31/12/2014 midnight, data of year 2010 will be removed from the DW. Notice, this additional step has to be performed only once a year in our example.

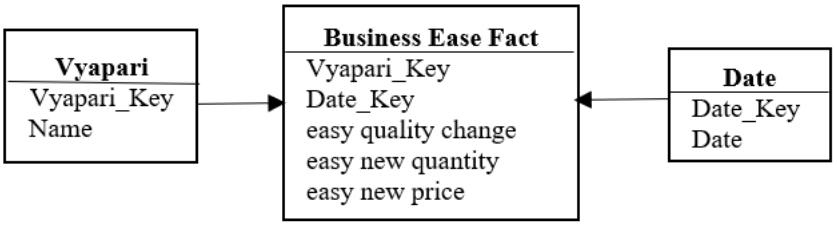
For Period = n, perform ETL every n units of time.

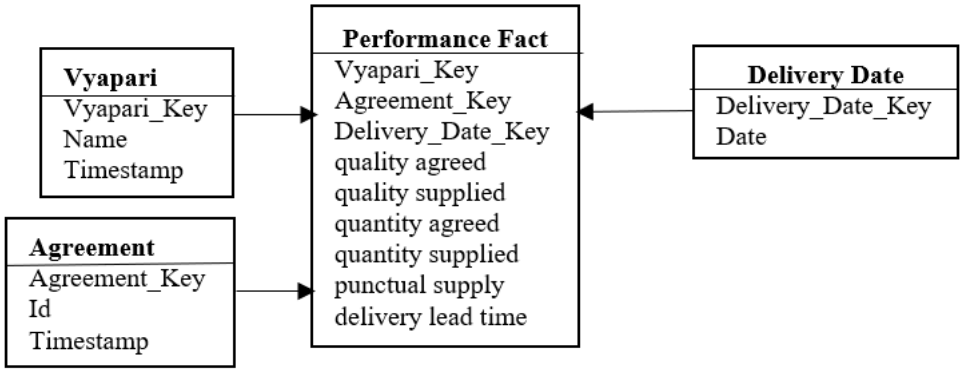
7.2 Snowflake Schema

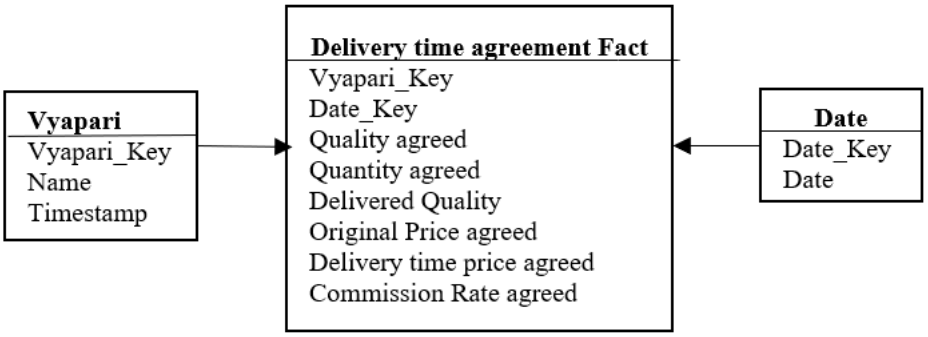
The snowflake schema generated using the algorithm in section 7.1 of the input schema given in section 6.1.

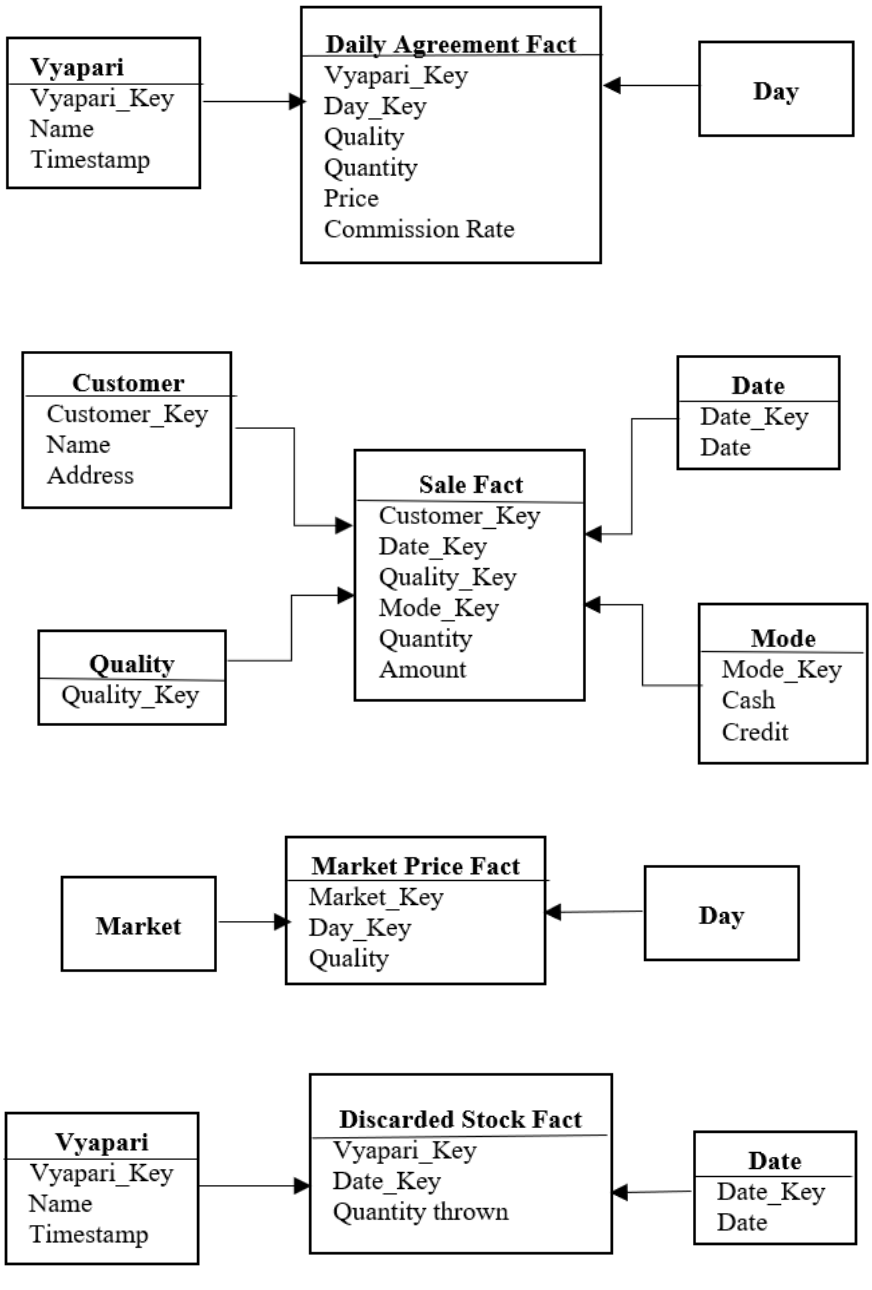


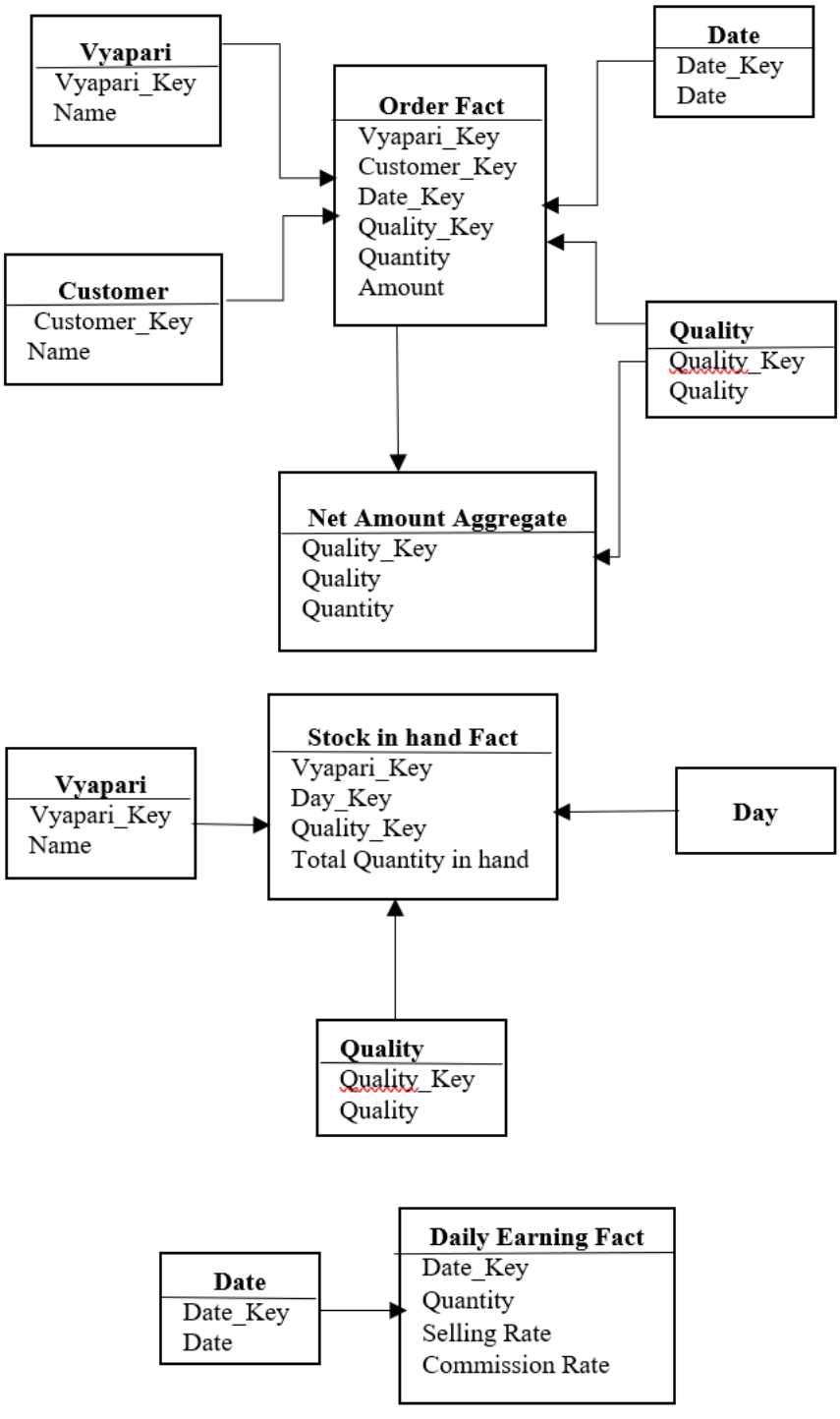

Agreement to sell Fact
Vyapari_Key
Date_Key
Quality agreed
Quantity agreed
Price agreed
Commission Rate agreed
Vyapari
Vyapari_Key
Name
Timestamp
Date
Date_Key
Date

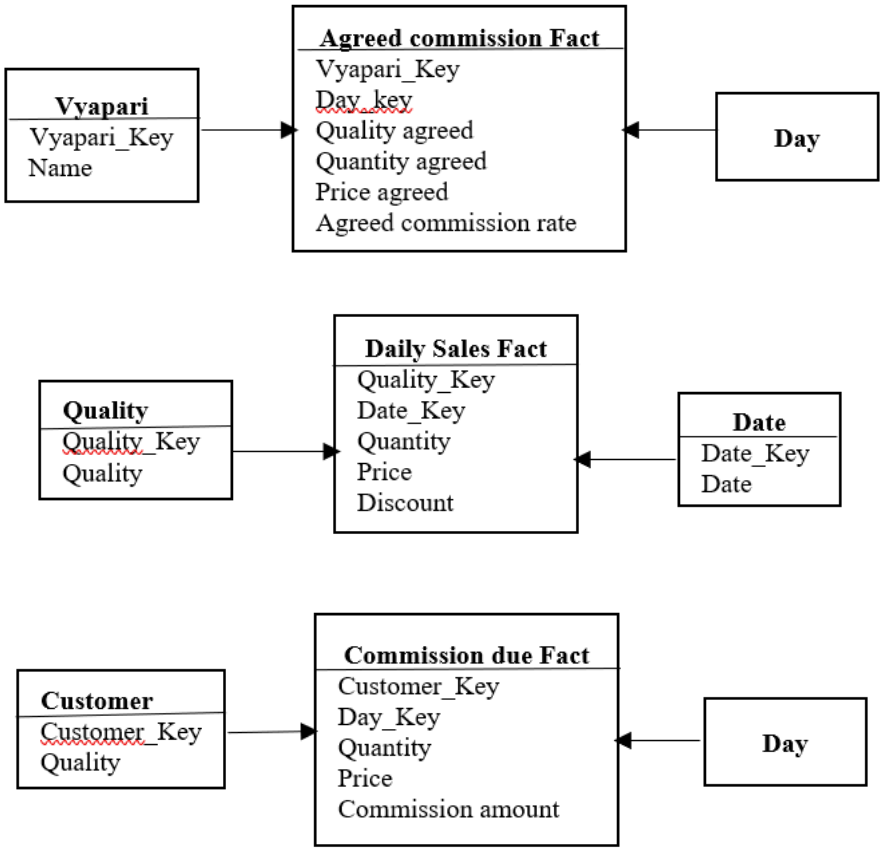

Business Ease Fact
Vyapari_Key
Date_Key
easy quality change
easy new quantity
easy new price
Vyapari
Vyapari_Key
Name
Date
Date_Key
Date


Performance Fact
Vyapari_Key
Agreement_Key
Delivery_Date_Key
quality agreed
quality supplied
quantity agreed
quantity supplied
punctual supply
delivery lead time
Vyapari
Vyapari_Key
Name
Timestamp
Agreement
Agreement_Key
Id
Timestamp
Delivery Date
Delivery_Date_Key
Date


Delivery time agreement Fact
Vyapari_Key
Date_Key
Quality agreed
Quantity agreed
Delivered Quality
Original Price agreed
Delivery time price agreed
Commission Rate agreed
Vyapari
Vyapari_Key
Name
Timestamp
Date
Date_Key
Date


Vyapari
Vyapari_Key
Name
Timestamp
Daily Agreement Fact
Vyapari_Key
Day_Key
Quality
Quantity
Price
Commission Rate
Day
Customer
Customer_Key
Name
Address
Date
Date_Key
Date
Sale Fact
Customer_Key
Date_Key
Quality_Key
Mode_Key
Quantity
Amount
Quality
Quality_Key
Mode
Mode_Key
Cash
Credit
Market
Market Price Fact
Market_Key
Day_Key
Quality
Day
Vyapari
Vyapari_Key
Name
Timestamp
Discarded Stock Fact
Vyapari_Key
Date_Key
Quantity thrown
Date
Date_Key
Date


Vyapari
Vyapari_Key
Name
Date
Date_Key
Date
Order Fact
Vyapari_Key
Customer_Key
Date_Key
Quality_Key
Quantity
Amount
Customer
Customer_Key
Name
Quality
Quality_Key
Quality
Net Amount Aggregate
Quality_Key
Quality
Quantity
Vyapari
Vyapari_Key
Name
Stock in hand Fact
Vyapari_Key
Day_Key
Quality_Key
Total Quantity in hand
Day
Quality
Quality_Key
Quality
Date
Date_Key
Date
Daily Earning Fact
Date_Key
Quantity
Selling Rate
Commission Rate

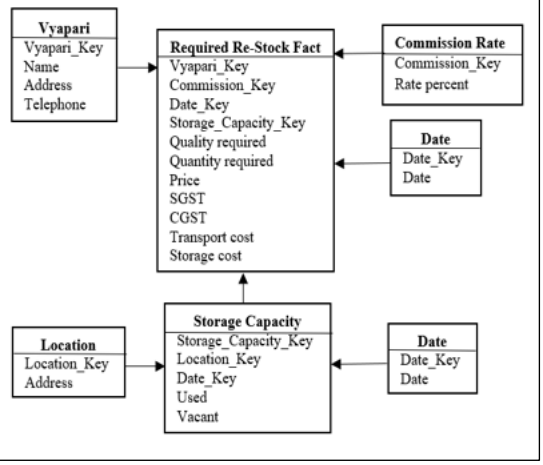


7.3 Comparison Between Relational Schema and MongoDB

Table 7.1 shows the comparison between relational schema and MongoDB. Data object containment and sub-category key are not described in the schema used in section 6.1*.

Suppose “Storage Capacity” data object is contained in “Required Re-Stock” data object in section 6.1. Then, it can be represented in MongoDB and relational schema as follows:

```
db.createCollection("required_re_stock");
db.required_re_stock.insertOne({
_id:,
storage_capacity: {
_id:,
vacant:,used:,date:{date_key:,date:},location:{location_key:,address:}},
sgst:,quantity_required:,transport_cost:,price:,cgst:,storage_cost:,quality_required
:,date:{date_key:,date:},commission_rate:{commission_rate_key:,rate:},vyapari:
{vyapari_key:,address:,name:,telephone:},CreationTime: new Date(),
Duration: new Date()}});
```



Suppose "Commission Rate" category is a sub-category of "Vyapari" category of "Required Re-Stock" data object in section 6.1. Then, it can be represented in MongoDB and relational schema as follows:

```
db.createCollection("required_re_stock");
db.required_re_stock.insertOne({
_id:,
sgst:,quantity_required:,transport_cost:,price:,cgst:,storage_cost:,quality_required
:,date:{date_key:,date:},vyapari:{vyapari_key:,address:,name:,telephone:,
commission_rate:{commission_rate_key:,rate:}},CreationTime: new Date(),
Duration: new Date()});
```

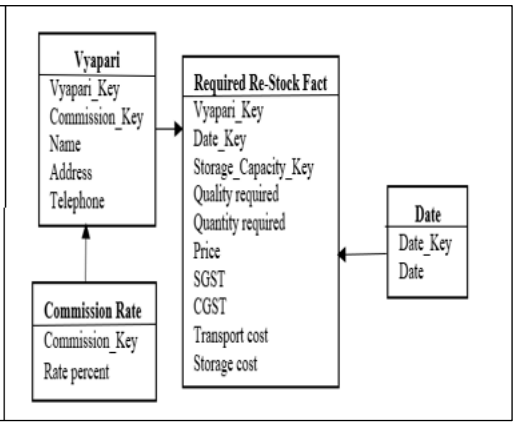


Table 7.1: Relational Schema and MongoDB Comparison

Key	Relational Schema	MongoDB
NULL Values	Required Re-Stock Fact has Foreign Key (Vyapari_Key and Commission_Key). If any of the Key value is not present in the Vyapari or Commission dimension, then there will be the problem of NULL.	Required Re-Stock Data Object has categories (Vyapari and Commission). In the case of NULL values, MongoDB will handle it by removing the NULL attributes.
Schema Representation	There are 17 snowflakes schema generated where each data object and category is represented by a fact and dimension respectively, which is connected using the concept of a foreign key.	There are 17 documents generated where each data object and category is represented by a document and nested document, respectively.
Data Object Containment*	For every data object that is contained in a parent data object, a dimension is created, which is connected to the fact (parent data object)using the concept of JOINs.	For every data object that is contained in a parent data object (collection), an embedded document is created.
Sub Category*	For every sub-category of a category, a dimension is created. A sub-category is connected to a dimension (category) using the concept of JOINs.	For every sub-category of a category , a nested document is created using the concept of hierarchical data storage in MongoDB.
Principle	ACID (Atomicity, Consistency, Isolation and Durability) acronym is used to describe the features of relational databases.	BASE (Basically Available, Soft State, Eventual Consistency) acronym is used to describe the properties of MongoDB. Here, data inconsistency is possible, but data is always available.

Facts and Dimension Representation	Fact is represented as a table, and each dimension is represented as separate tables. Fact tables and dimension tables generally reference each other. A fact table contains typically several primary keys that later form their entries in the dimension tables. It is shown in section 7.2.	Fact is represented as a collection and dimensions are represented as nested documents or embedded documents in the collection. It is shown in section 6.2.

OLAP Operations:
The OLAP operations are [32] [33] :

- **Roll-Up:** It performs aggregation on a data cube in either of the following ways:
  - By stepping up a concept hierarchy for a dimension
  - By reducing a dimension
- **Drill-down:** It is the reverse operation of roll-up. It is implemented in either of the following ways:
  - By rising down a concept hierarchy for a dimension
  - By introducing a new dimension
- **Slice:** It selects one particular dimension from a given cube and provides a new sub-cube.
- **Dice:** It selects two or more dimensions from a given cube and provides a new sub-cube.

"Storage Capacity" data object from Vegetable Trader Schema in section 6.1 is taken to illustrate the example in table 7.2.

Table 7.2: OLAP Operations in Relational Schema and MongoDB

OLAP Operations	Query	Relational Schema	MongoDB
Roll-up	Return the total used storage across all used storage at increasing aggre-gation levels of location: from state to country to region for different Quarters.	SELECT Date, Location ,sum (Used) AS Total_used_Storage FROM StorageCapacity GROUP BY ROLLUP (Date, Location);	db.StorageCapacity.group({ "key":{"ROLLUP(Date, Location)": true}, "initial":{ "sumUsedASTotal_used_Storage ": 0}, "reduce": function(obj , prev){ prev.sumUsedASTotal_used_Storage= prev.sumUsedASTotal_used_Storage + obj.UsedASTotal_used_Storage - 0; } });
Drill-down	Return the total used storage across all used storage at decreasing aggre-gation levels of location: from region to country to state for different Quarters.	SELECT Date, Location ,sum (Used) AS Total_used_Storage FROM StorageCapacity GROUP BY ROLLDOWN (Date, Location);	db.StorageCapacity.group({ "key":{"ROLLDOWN(Date, Location)": true}, "initial": { "sumUsedASTotal_used_Storage ": 0}, "reduce": function(obj ,prev){ prev.sumUsedASTotal_used_Storage= prev.sumUsedASTotal_used_Storage + obj.UsedASTotal_used_Storage - 0; } });

Slice	Return the total number of used storage capacity on Date “05-05-2020” sold across all of the Vegetable Trader locations	Select Date, sum (used) from StorageCapacity wh ere Date = '2020-05-05' GROUP BY Date;	db.StorageCapacity.group({
			"key":{ " Date": true},
			"initial": { "sumused ": 0},
			"reduce": function(obj , prev)
			{ prev.sumused = prev.sumused + obj. used - 0;},
			"cond": {
			"Date " : '2020-05-05'} });

Dice	Return the total number of used storage capacity on Date “05-05-2020” in the particular Vegetable Trader location of Europe	Select Date, sum (used) from StorageCapacity wh ere Date = '2020-05-05' and Location = ’Europe’ group by Date;	db.StorageCapacity.group({
			"key":{ " Date": true},
			"initial": { "sumused ": 0},
			"reduce": function(obj , prev)
			{ prev.sumused = prev.sumused + obj.used - 0; },
			"cond": {
			"$and": [{
			"Date " : '2020-05-05'
			},{ "$where": "this. Location == this. ’Europe’ "
			}]
			} });

Normalization:

- Fact table is normalized, but the dimension tables are not normalized in the case of a star schema. A snowflake schema is an expansion of a star schema, and it adds new dimensions. The dimension tables are also normalized, which splits data into other tables [30]. It is shown in section 7.2.
- In MongoDB, in case of normalization, you are dividing your data into multiple collections with references between those collections. Each piece of data will be in a collection, but various documents will reference it [31]. In MongoDB, the fact and dimension tables are de-normalized, since dimension tables are represented as nested or embedded documents. It is shown in section 6.2.

Query capacity: The basic relational algebra operations are shown in table 7.3. “Storage Capacity” data object from Vegetable Trader Schema in section 6.1 is taken to illustrate the example.

Table 7.3: Relational Algebra Operations in Relational Schema and MongoDB

Relational Algebra Operations	Query	Relational Schema	MongoDB

UNION	Returns all the dates and select all the locations.	SELECT Date
		FROM Date
		UNION
		SELECT Address
		FROM Location;
			db. Date.aggregate([{
			$lookup:{ from:"Location ",
			pipeline: [], as: " Location" } },
			{$addFields: { Location: {
			$map:{input: "Location", as: "loc",
			in: {"type": " Location ",
			"address": "$$loc. address ", }
			} } } }, {
			$group: { _id: null, Date: {
			$push:{ type: "Date",
			Date: "$Date" } },
			Location: { $first: "$Location" } } }, {
			$project: { items: {
			$setUnion:["$Date","$Location"]
			} } }, { $unwind: "$items" }, {
			$replaceRoot:{newRoot: $items" } }
			]);

CROSS PRODUCT	Returns cross product if all the dates and locations.	SELECT Date, Address FROM Date, Location;	NOT POSSIBLE in MongoDB. *Note: Only LEFT JOIN is possible from MongoDB 3.2 using $lookup.*
SELECTION	Returns all the details of location.	SELECT * FROM Location;	db.Location.find({});

DIFFERENCE	Returns all the location key from Location which are not present in storage capacity.	SELECT Location_Key
		FROM Location
		MINUS
		SELECT Location_Key
		FROM StorageCapacity
		;
			locKey = db.StorageCapacity. distinct ("Location_Key ");
			
			db.Location.distinct("Location_Key", { "Location_Key": { $nin: locKey }});

PROJECTIONS	Returns the location key in storage capacity where used storage capacity is greater than 100.	SELECT Location_Key FROM StorageCapacity WHERE Used>100;	db.StorageCapacity.find({ "Used":{ "$gt" : 100 } }, { "Location_Key": 1 });

Note: A language is relationally complete if the basic relational algebra operation can be performed: UNION, CROSS PRODUCT, SELECTION, DIFFERENCE, PROJECTIONS [34]. Since the "CROSS PRODUCT" operation cannot be implemented, MongoDB is relationally incomplete.

Chapter 8
Conclusion

There are mainly two ways in which a Data warehouse is implemented. One approach is to use the data cube in a multi-dimensional database directly. These systems give high performance but are not proficient of containing detailed data. The other method to implement a data warehouse is to use a relational database. Here, facts and dimensions are implemented as relational tables. Relational databases have several disadvantages like relational database allows NULLs, not supporting all the different types of data that is to be saved, involving join operations which leads to system performance issues. Hence, we use a document-oriented NoSQL database, MongoDB. After all the information needs of the dimensional model of data warehouse has been recognized by the implementation phase of data objects, we introduce mapping rules using the concept of the faithfulness of schema* to convert the information obtained to the logical model of MongoDB. After the mapping of rules, the commands can be used to perform OLAP operations. We also illustrate a relational schema for the same case study and lastly compare between MongoDB and relational schema.

Future work includes:

1. Developing mapping rules for a graph database like Neo4j.
2. Developing mapping rules for key-value databases like Riak.

***The destination schema D is faithful to the source schema S if it does not cause the system to fall into inconsistency, i.e., D does not violate any constraint of S [28].**

Bibliography

[1] "Replication and sharding in mongodb tutorial — simplilearn." https://www.simplilearn.com/ replication-and-sharding-mongodb-tutorial-video.

[2] "Nosql tutorial: Learn nosql features, types, what is, advantages." https://www.guru99.com/nosql-tutorial.html.

[3] Wikipedia contributors, "Mongodb — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=MongoDB&oldid= 927312653, 2019. [Online; accessed 21-November-2019].

[4] Dan Sullivan. 2015. NoSQL for Mere Mortals (1st. ed.). Addison-Wesley Professional.

[5] "Introduction to document databases with mongodb — derickrethans." https://derickrethans.nl/introduction-to-document-databases. html.

[6] "Mongodb - datatypes - tutorialspoint." https://www.tutorialspoint. com/mongodb/mongodb_datatype.htm.

[7] "db.collection.find() — mongodb manual." https://docs.mongodb.com/ manual/reference/method/db.collection.find/.

[8] "Mongodb - aggregation - tutorialspoint." https://www.tutorialspoint. com/mongodb/mongodb_aggregation.htm.

[9] "Mongodb indexing tutorial with example." https://beginnersbook. com/2017/09/mongodb-indexing-tutorial-with-example/.

[10] "Mysql master-slave replication tutorial — toptal." https://www.toptal. com/mysql/mysql-master-slave-replication-tutorial.

[11] "What is blob data type in mysql?." https://www.tutorialspoint.com/ What-is-BLOB-data-type-in-MySQL.

[12] "Openedge." https://documentation.progress.com/output/ua/ OpenEdge_latest/index.html#page/dmsrf/blob-limitations.html.

[13] "Databases and collections — mongodb manual." https://docs.mongodb. com/manual/core/databases-and-collections/.

[14] "Difference between fact table and dimension table." https://www.guru99.com/fact-table-vs-dimension-table.html.

[15] C. Ballard, D. M. Farrell, A. Gupta, C. Mazuela, and S. Vohnik, Dimensional Modeling: In a Business Intelligence Environment. Vervante, 2006.

[16] Aggregate (data warehouse) - wikipedia." https://en.wikipedia.org/wiki/Aggregate_(data_warehouse).

[17] Prakash D., Data warehouse design using Document Stores, internal report, dept. of CSE, NIIT University, Neemrana, Rajasthan.

[18] "What is data warehouse? types, definition & example." https://www.guru99.com/data-warehousing.html.

[19]"Data warehousing - relational olap - tutorialspoint." https://www.tutorialspoint.com/dwh/dwh_relational_olap.htm.

[20] "Data warehousing - multidimensional olap - tutorialspoint." https://www.tutorialspoint.com/dwh/dwh_multidimensional_olap.htm.

[21] D. Prakash., "Nosolap: Moving from data warehouse requirements to nosql databases," in Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE, , pp. 452–458, INSTICC, SciTePress, 2019.

[22] Wikipedia contributors, "Nosql — Wikipedia, the free encycloedia." https://en.wikipedia.org/w/index.php?title=NoSQL&oldid=954869460, 2020.

[23] N. Gupta, Comparison of relational database and MongoDB, internal report, dept. of CSE, IIIT Delhi, New Delhi.

[24] M. Chevalier, M. El malki, A. Kopliku, O. Teste, and R. Tournier, "Implementing multidimensional data warehouses into nosql," ICEIS, vol. 1, 04 2015.

[25] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Big data warehouse: Building columnar nosql olap cubes," International Journal of Decision Support System Technology, vol. 12, pp. 1–24, 01 2020.

[26] Chevalier, Max and El Malki, Mohammed and Kopliku, Arlind and Teste, Olivier and Tournier, Ronan Implementation of Multidimensional Databases with Document-Oriented NoSQL. (2015) In: 17th International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2015) in 26th DEXA Conferences and Workshops, 1 September 2015 - 4 September 2015 (Valencia, Spain).

[27] R. Yangui, A. Nabli, and F. Gargouri, "Automatic transformation of data warehouse schema to nosql data base: Comparative study," Procedia Computer Science, vol. 96, pp. 255 – 264, 2016. Knowledge-Based and Intelligent Information Engineering Systems: Proceedings of the 20th International Conference KES-2016.

[28] T. Schaub, G. Friedrich, and B. O'Sullivan, ECAI 2014: 21st European Conference on Artificial Intelligence 18-22 August 2014, Prague, Czech Republic. Ios Pr Inc, 2014.

[29] D. Prakash, "Direct conversion of early information to multi-dimensional model," in Database and Expert Systems Applications - 29th International Conference, DEXA 2018, Regensburg, Germany, September 3-6, 2018, Proceedings, Part II (S. Hartmann, H. Ma, A. Hameurlain, G. Pernul, and R. R. Wagner, eds.), vol. 11030 of Lecture Notes in Computer Science, pp. 119–126, Springer, 2018.

[30] "Star and snowflake schema in data warehouse." https://www.guru99.com/star-snowflake-data-warehousing.html.

[31] "Mongodb: Normalizationvsdenormalization-dev." https://dev.to/damcosset/mongodb-normalization-vs-denormalization#:~:text=What%20is%20normalization%3F,multiple%20documents%20will%20reference%20it.

[32] "Datawarehousing-olap-tutorialspoint." https://www.tutorialspoint.com/dwh/dwh_olap.htm.

[33] "Data cube operations - sql queries - perficient blogs." https://blogs.perficient.com/2017/08/02/data-cube-operations-sql-queries/.

[34] "Relationalquerylanguages—relationalcompleteness dbmsenotes." http://dbmsenotes.blogspot.com/2014/08/relational-query-languages-relational.html

Appendix A

Output File

Description:

The file in JavaScript format shows the output generated by the tool bases on the conversion rules given in the section 5.1. Here, P5 is the project name.

Filename:

P5.js

Appendix B

Output File (with values)

Description:

The file in JavaScript format shows the manual data inserted into the file to run the MongoDB queries directly using the following command:

mongo localhost:27017 "dbScripts/P5_withvalues.js"

Here, P5 is the project name.

Filename:

P5_withvalues.js

Curriculum Vitae (CV)

NEHA GUPTA

Email: neha18113@iiitd.ac.in

DOB: October 08, 1992

Education

Indraprastha Institute of Information Technology, New Delhi — CGPA (Current): 6.7
M. Tech – CSE (2018 – 2020)

NRI Institute of Information Science and Technology, Bhopal — CGPA: 8.13
B. Tech – CSE (2010 – 2014)

St. Xavier's Sr. Sec. Co-Ed School, Bhopal (M.P.) — Percentage: 74
CBSE (2009 – 2010)

St. Xavier's Sr. Sec. Co-Ed School, Bhopal (M.P.) — Percentage: 78.6
CBSE (2007 – 2008)

Work Experience

Assistant System Engineer – Trainee (Team Member) (Dec,14 – Feb,15)
TCS ILP Training — Team Size-4
Project – Managing Student's Data
Objective - To manage student's data of a college using CRUD operations.
Programming Language – Visual C#

Systems Engineer (Java / Web Developer) (Mar,15 – Jun,18)
TCS
Project – Khajane II (Karnataka IFMS)
Objective - Computerization of all the treasuries in the state of Karnataka and connecting them to a central server at the state secretariat.
Programming Language and Tools – Java, iReport, Eclipse, DB2
Technologies – JavaScript, Hibernate, JSP, CSS, Spring

Projects and Papers

M. Tech Thesis (Aug,19 – July,20)
Guide: Dr. Naveen Prakash — Team Size-1
Objective – Representing Dimensional Model in MongoDB
Programming Language and Tools – MongoDB Shell, MS SQL Server, Java, JavaFX

Independent Study (May,19 – July,19)
Guide: Dr. Naveen Prakash, Dr. Samaresh Chatterji — Team Size-1
Objective – Comparison of Relational database and MongoDB

Emotion, Gender and Text Classification on Speech (Feb,19 – Apr,19)
Guide: Dr. Richa Singh Team Size-2
Objective – Classification of emotional state, spoken text, and gender on human speech using Machine Learning techniques.
Programming Language and Tools – Python, Jupyter Notebook

Term Paper (Feb,19 – Apr,19)
Guide: Dr. Naveen Prakash Team Size-2
Objective – Aspect-Oriented Programming

Interests and Hobbies

- Research – NoSQL Databases, Data Warehouse
- Technical – Software development (Java), Machine Learning (Python)
- Travelling
- Painting

Declaration: The above information is correct to the best of my knowledge.
Neha Gupta
Date: May 22, 2020