Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 4, 618-625 (2020)

www.astesj.com

ASTESJ

ISSN: 2415-6698

Modeling and Transformation from Temporal Object Relational Database into Mongodb: Rules

Soumiya Ain El Hayat* , Mohamed Bahaj

Department of Mathematics & Computer, Faculty of Science and Technologies, Settat, 26000, Morocco

ARTICLE INFO

Article history:
Received: .13 June, 2020
Accepted: 22 July, 2020
Online: 25 August, 2020

Keywords:
Temporal database
Mongodb
Nosql
Temporal object relational database
Migration

ABSTRACT

With such a big volume of data growing tremendously every day, and the storage of important volume of data becoming increasingly more flexible, NoSQL(Not only SQL or Non-Relational) database are designed to store a large amount of information and are growing for big data systems in web analytics. It an approach does not require any specific schema and avoid the use of the joins to store or retrieve the information. To ensure the availability and scalability many industries are now replacing their object relational database in many systems by adopting NoSQL database technology for e-business applications. It is document-oriented databases which help in grouping data more logically. This paper describes a disciplined approach of migration and proposes a model transformation from temporal Object relational database (TORDB) Into Mongo db. Also, this paper approach deals with a novel data integration methodology in order to manipulate and the store the varying time data in Json (Java Script Object Notation) documents.

1. Introduction

In the recent years, a varying Time management database is one of the most interest systems of the information technology to support temporal features associated with data. It models the information in the real world. The time is an important property to characterize a data on the web. The temporal database has appeared since the 1980s, there were a several authors produced articles and books about varying-time, but the commercial adoption has been slow [1]. During a long time, developers and researchers developed several applications that support new data type concepts bases on the time. The temporal data processing is very important in dynamically evolving systems, industry, communication systems and also in systems processing sensitive data, which incorrect change would cause a great harm [2]. The literature on varying time management offers three Periods type for temporal data support: valid time period, transaction time period and Bitemporal data which combine both to make a complete description of data. Hence, the need to retain trace and audit the change made to a data and the ability to plan based on the past or future assumptions are important uses cases for temporal [3].

*Corresponding Author: Soumiya Ain El Hayat ,FST , Settat, Morocco ,+212658521182, soumya.ainelhayat@gmail.com

Today, we notice the advent of big data. There is a revolution going on in databases system management. With the development of data acquisition technologies, the information to be stored expands strikingly in volume and velocity. NoSQL database have evolved intensely in the last years due to their flexible structure, less constrained than relational ones and offering faster access to information. Nosql is now used in many fields of industries and companies to support applications and systems not well served by relational and object relational database. NOSQL is released and widely used in many domains. Nosql database provides a mechanism for storage and retrieval of instructed data other than tabular or object relation used in relational and object relational database respectively [4]. The Nosql model fulfils the scalability problems. Nosql databases are mostly open source , non-relational , distributed and designed for large volumes of data across many clusters supporting replication and partitioning , parallel processing and what is usually called horizontal scaling [5].

One of the post popular and leading Nosql system management is Mongodb. Mongo db is an open source database based on distributed document released in 2009. It stores data as JSON-Like document with dynamic schemas (The format is called BSON) [6]. Mongodb is a document-oriented database which holds a set of collection that are similar to relational database Tables where each collection contains a set of documents. One collection can hold different document with number of fields, content and size are not similar. It is a schema less that means the mongodb can be a distributed database and does not have a predefined schema so that allows providing additional data type and inserting new fields. Mongodb document is a set of key value pairs. Key values databases provide a hash table where the unique key and a pointer to a specific set of values are stored and data can be retrieved using the key. Mongodb is suitable solution for distributing data and managing balance between instances while using replication to increase the level of availability [7].

With the information industry dependent on the time, developing rapidly in recent years, the dataset using in different system are becoming extremely large in volume with a high variety of data. For this reason, Mongodb has been invented to overcome the limitation of relational and Object Relational Database, and provides new mechanisms for managing huge amount of data that are different from the typical relational and object relational Model. In addition, the mongodb is adopted to handle the huge evolution of temporal data in distributed environments in which continue to rise in complex applications and social network.

The main goal of this approach is to provide a reliable, reasonable and efficient method to convert the schema and migrate the temporal data from the implemented temporal object relational database into Mongodb system. Our proposed approach provides a new model transformation from object relational tables including Bitemporal data features towards documents-oriented databases based on Json files. Several rules are defined to facilitate the migration process.

2. Related Works

A research work in [8] proposed a generic standard-based architecture that allows Nosql systems focusing on mongo db to be manipulated using SQL query and seamlessly interact with any software supporting JDBC. Ajit Singh demonstrated data conversion to mongo db , in order to make data more interactive and innovative using the data stored in cloud [9]. The model transformation and data migration from relational database into Mongodb taken into consideration the query characteristics of each model, in addition, an algorithm to automate the migration are discussed in [10]. Another approach presented Algorithm for automatic mapping of relational database to mongodb using entity relationship (ER) Model to provide the conceptual schema and modelling relationship between the different entities [11]. A framework for mapping MySQL database to mongodb by developing an algorithm that uses the metadata stored in relational system as input is discussed in [12]. The work in [13] described a migration process from Object relational database to Nosql document database end provided a review of different proposed approach. An overview of Nosql to evaluate the scalability and efficiency in storage of data in oriented document database case study in order to show the representational format and querying management process of Mongodb [14]. The work in [15]introduced a disciplined approach called Jschema(Temporal Json Schema) for the temporal management Json documents by creating a temporal json documents from conventional document that can vary over time, the generated document uses such features of temporal management data. The mapping Process of spatio temporal disaster data into Mongodb database using the data represented by the aspects of space and time is shown in [16].Boicea, Radulesu and agapin discussed the difference between oracle and Mongodb this comparison dealt with several criteria including theorical , Concept , restrictions and query processing of SQL database get a document oriented database Management [17].

During our criticized analysis, we noticed that many studies are devoted to the analysis and the migration from relational database into Mongodb. Most of the previous approaches don’t deal with the use of temporal data in order to make a complete description and efficient correctness to historical data. The temporal management features in Mongodb concepts was overlooked in many research, which reflects that the purpose for those researchers was only to provide methods to manipulate the data rather than on gain from the offered features of schema less of Nosql to handle a huge amount of historical data. In addition, works about the conversion from temporal database based on object relational into MongoDB are not as frequent.

From the all overhead works, we covered the transformation and the evaluation of mongodb to relational database. We conclude that mongodb has overcome many limitations of the conventional Relational and object relational database. The contribution of our work focuses on temporal aspect in emerging of temporal oriented-documents database. Our approach is based on semantic enriched mechanism which simplify the development of schema translation which enhanced by additional varying time data features. And formalize the rules to simplify the transformation process from temporal Object relational database into Mongodb which is based on temporal json file enriched with bitemporal data dimension. Our study examines the possibility to maintain and understand the structure of varying time attributes defining in TORDB tables and also to promote the storage of data into json documents using some semantics concepts.

3. Modelling and transformation Rules from TORDB into Mongodb: Phases

In this section, we present the most necessary phases for translation process. We are going to propose several rules that allow developing the determined schema. In the first, we present a comparison between temporal object relational database and Mongodb. After that we will explain the different elements composed TORDB design, and then we will define the mongodb schema including bitemporal data. In this paper, we will not explain the rules for translation of TORDB model into their equivalent in the TJson_schema.

3.1. Comparison between Object relational database and Mongo db features:

Oracle Database is an Object-relational database management system produced and marketed by oracle Corporation [18], which combine the relational database concept and object-oriented database capabilities. In object relational database, the user has the ability to create his own predefined type called UDT or User Defined Type, which is used to specify the structure and the behaviour of the data, based on Object mechanism in order to define and create complex data type. Therefore, ORDBs with time-varying features has addressed to enhance ORDB efficiency and makes a complete description of recorded data. For this reason, we create User-defined time approach. It is a time representation created by object relational techniques to satisfy specific needs of users. The User-defined time uses a bitempral data dimension which support both valid time and transaction time. The bitemporal data presents the changing knowledge of the changing world, hence, links data values with facts and also determines when the facts were valid, in order to provide history of data values and their changes over time. The following statement show the Bitemporal period Object to store valid and transaction Time attribute:

Query 1: Bitemporal Object

```
Create type Bit_period as object
( vt_Start date,
vt_end date,TT_start  date,
TT_End  date) /
CREATE TYPE bitemporal_period  IS TABLE OF
bit_period;
```

Mongodb is an open source Nosql database based on oriented document structure. It was developed during 2007 by software called 10gen Company. Mongodb documents are stored in binary form that are similar to Json document model called BSON format, that supports such primitives data type (String, integer, date, Boolean, float and binary).the main features in mongodb are collections and documents. Mongodb documents have a flexible schema in which the collection dos not impose the necessary document schema. However, the temporal object relational databases require a table schema to be declared and created before inserting the data. Any temporal object table has a certain design that shows the relationships between them. Although mongodb does not support join operations as SQL databases, the relationship between documents can be represented using either the referenced or embedded concepts. The relationship in mongodb define how various documents logically dependant to each other. The relationship in mongodb can be expressed via embedded or referenced concepts. Where, embedded documents maintain all the related data in one document. These renormalized data representations allow applications handle and retrieve data from a single document.

In the following subsection, we focus on comparing mongodb to Temporal Object relational database. The table presents the main techniques of TORDB methodology and Mongodb.

On the other hand, the referenced document stores the relationships separately between document by adding id-Field that references or links from one document to another. This approach designs the normalized relationship. Actually, the difficult part of transformation process is how we can convert the relationships of the existing temporal object relational database into Mongodb document.

Table 1 the Differences between TORDB and Mongodb features

TORDB	**Mongodb**
Database	Database
Temporal Object\Temporal Table	Temporal Collection
Row	Document
Column	Field
Data Type	Data Type
Primary Key	Id_Field
Simple UDT| REF | Nested table | Array | nested table (REF)	Referenced Document \ Embedded Document

In our work, each User Defined Time or UDT is converted to a MongoDB collection, in this example the collection name is customer contains Bitemporal period object. The customer_table holds data rows of customer_type objects. Also, the customer collection will store customer objects with the same attribute as documents in BSON (Binary encoded JSON) format including varying time attributes. Then statements below shows the creation of customer collection and how we can generate documents with bitemporal Object in mongodb :

Document1: Example of Json file with bitemporal data

```
db.createCollection(“Customer”) \\ Creation of Customer
Collection
db.Customer.insert(
{
“Customer_Id”:23
“Name”: “Soumiya”
“Bitemporal_data”:
{
“Vt_start”: “2020-02-03”    \\ Valid time start
“Vt_End”:   “2028-02-28”   \\ Valid Time End
“TT_start”: new Date()   \\ Transaction Time Start takes
the sysdate as Default value .
“TT_End”:  “9999-12-31” }} \\ Transaction Time End sets
the value of insertion of transaction time to the highest
value (“31/12/9999”)
```

3.1 Definition of the data Model

3.1.1 Temporal Object Relational Database Model (TORDB_Model):

Our approach in [19] consists in structuring a generic model of semantic enrichment. We defined the component of a TORDB Model, which constructs a comprehensive schema of the temporal object-relational model. The TORDB model is expressed as a set of temporal typed table based on structured type ST and temporal structured type TST which include the user defined Time (Bitemporal Period) for specifying temporal period interval. Each

TST is declared as a set of simple attributes and varying time attributes. In this obtained Model, the temporal attributes are based on a bitemporal object that actually defined in a nested table as a collection type.

Generally, The TORDB model is denoted as three-tuples:

$$\text{TORDBModel} = \{TT|TT=(TTs,STs,Tm)\} \quad (1)$$

where:

• TT s is set of temporal and non temporal typed table, STs is a set of temporal structured type or simple structured type , and Tm is a time-varying Period.

The sets TTs, STs and Tm are defined as follows:

$$STs = \{Sn , S, AT\} \quad (2)$$

where Sn is the name of a structured type , S is the super type of ST, and AT is a set of structured type's attributes:

$$AT=\{A|A:=\{N,T,D,NL,BitT,M\}\}, \quad (3)$$

where

N: is the name of attribute, T: data type and can be primitive, UDT or reference. NL: if the attribute allows Null value or not. D: default value. M: means if the AT is a single valued or collection valued.

BitT: denotes if the attribute contains a bitemporal object is defined:

$$BitT=\{(VT\_Start,VT\_End,TT\_start,TT\_End)\} \quad (4)$$

$$\bullet TTs=\{typedtable|Ttable=\{TTn,STn,PK,Tp\}\} \quad (5)$$

where: TTn is the name of typed table, STn is the name of the structured type based upon which TT is defined, PK : primary key, TP: means if the TT is temporal or not.

Association: For each relationship Rel where RelType= "associate with" is translated into:

$$TT\_Association = \{TT|TT=(TTn, STn, AT \cup (Ref(ST')) ,PK, Bit\_P)\} \quad (6)$$

In association relationship, we define a method for storing the reference type of the structured type ST' (REF(ST')) referencing to the class which is related as a Nested Table collection with Bitemporal data dimension.

Aggregation: each relationship Rel where RelType= "Aggregation" is expressed as a collection of UDT (User Defined Type) representing the C' class that interacts with class C.

$$TT\_Aggregation= \{ TT|TT=(TTn, STn, AT \cup (NT(UDT(ST')),PK, Bit\_P)\} \quad (7)$$

Composition: each relationship rel where RelType = "composition" is mapped into an attribute typed as a nested table maintains the attributes of the ST' corresponding to a class part C'.

$$TT\_Composition = \{ TT|TT=(TTn, STn, AT \cup NT(ST') ,PK, Bit\_P) \} \quad (8)$$

Inheritance: each relationship rel where RelType = "inheritance", the class C' inherits all the properties of its super class that are matching to ST'. For the defining the inheritance relationship in Object relational model, we add the keyword UNDER during the creation of ST that represents the child or sub class C.

$$TT\_ihneritance= \{TT|TT=(STn, ST.AT \cup Super\_T.AT, PK, Bit\_P)\} \quad (9)$$

3.1.2. *Temporal Json schema (Tjson-schema):*

Tjson-schema is a representation of Json document with historical data that is enhanced with additional semantic data to offer a new description of mongodb document. We have chosen the most commonly used in the oriented documents that able to identify the relevant collections, their documents and their relationships. The model constructs a data reference design in order to facilitate the understandability of metadata stored in Json document integrating Bitemporal Data. Also, it overcomes the complications that occur during the transformation process.

Tjson-schema can be defined as follows:

$$\text{Tjson-schema} =\{TJ|TJ=\{ Col\_name , Tdoc , RELCol\}\} \quad (11)$$

where:

- Col_name : each collection has a name , where the collection can be defined as a set of temporal and simple documents.
- Tdoc: denotes temporal Json documents including varying Time fields. Generally, the document in mongo db is a set of a key value pairs:

TDOC= {doc_ID, Fields, Bitemporal_Period, Primary Key}

- The Json document uses Object Id as a default id which is generated during the creation of mongodb document.
- Fields = means a set of fields and can be identified by the following elements:

$$Field=\{ Field\_Name , Field\_Name\} \quad (12)$$

where:

Field_Name= is the name of the field.

Field_Type= means data type (integer, string, date)

- Bitemporal_Period= means the embedded document in Json document which composed by:

Bitemporal_Period ={VT_start,VT_end,TT_start,TT_end} (13)

- Relcol=Relations Tjson-schema. Each collection has a set of relationship between documents and can be identified as follow:

Relcol ={RelType,RelName,Dircol} (14)

where:

Reltype= each collection or Document has a relationship Type with other documents. The Reltype offers 2 types of relations: Embedded document of referencing document.

Dircol= is name of doc' that is related to the document Doc.

Document 2: Account_Document extraction with Mongodb

```
{
   "_id":1345672
    "Num_Accout": 1
    "Type_Account": "Saving_Account"
    "account_h":
           {
           "VT_start": ISODATE("2010-01-01")
           "VT_End":  ISODATE("2012-11-30")
"TT_Start": ISODATE ("2010-01-03")
            "TT_End": ISODATE("2012-12-04")
              }
"Customer":{

           "_id": "145673" }}
```

3.2. Transformation Rules from TORDB into mongodb:

3.2.1. Association

R1: For the two UDTs Namely Customer_Type and Account_Type contain Bitemporal data and are related with association (1...N) relationship, using reference mechanism which allow retrieving data rapidly without using join between tables. The transformation of association in Mongodb consists on generating a new document where the Customer_Type will be referenced in Account_type document and the object type will be embedded in the both document.

Col1={NameCol1,NameDoc,Doc(Fields),Embedded(Bitemporal_Period)} (14)

Col2={NameCol2,NameDoc,Doc(Fields),Object_ID(Doc),Embedded(Bitemporal_Period)} (15)

The example below shows the structure of customer and account document, also the TORDB query Statement:

The creation will proceed according to the following syntax:

Col1={Customer,Customer_Json Document , (_id, Id_Cust, Name), Customer_h (VT_start, VT_End,TT_Start, TT_End)}

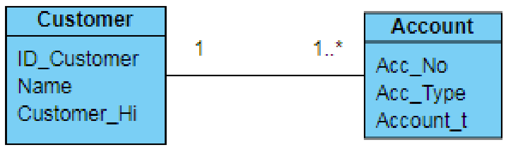

Figure1. An example of Association 1 to N relationship between Customer and Account

Col2={ Account, Account _Json Document , (_id, Num_Accout, Type_Account), Customer (_id),account_h (VT_start, VT_End,TT_Start, TT_End)}

The defined TORDB Query for the corresponding example:

Query2: creation statement for account table

```
  CREATE TYPE account_t AS OBJECT
  (acc_no NUMBER,
  acc_type varchar(20),
  Customer REF customer_T,
  account_h bitemporal_period);
CREATE TABLE account_table of account_t NESTED
TABLE  account_h STORE AS accounth_tab ;
```

On the other hand, in the case of association many to many (N,N) , the both document representing account and Branch_bank(see the example below), will be mapped in composition between the both document where each document integrates referenced document of another. The transformation result is:

Col1={NameCol1,NameDoc,Doc(Fields),Object_ID(Doc),Embedded(Bitemporal_Period)} (16)

Col2={NameCol2,NameDoc,Doc(Fields),Object_ID(Doc),Embedded(Bitemporal_Period)} (17)

The example shows the structure of branch_bank and customer Json document representing N to N Relationship:

Document 3: Extraction of branch_bank Json file

```
Banch_bank document:
  {
    "_id":45679
     "branch_pk":1
     "city": "Casablanca"
     "phone": 06453278
     "Bitemporal_Period":
            {
             "VT_start": DATE("2013-03-01")
             "VT_End":  DATE("9999-12-31")
             "TT_Start": DATE ("2013-03-03")
               "TT_End":DATE("9999-12-31")
              }
  "Customer":{
            "Customer_id": " 2334" }}
  Customer_Json Document :
  {
    "_id":2334
```

```
      "Num_Cust":
      "Name_cust":
      "Bitemporal_Period":
            {
            "VT_start": DATE("2010-01-01")
            "VT_End":  DATE("2012-11-30")
            "TT_Start": DATE ("2010-01-03")
              "TT_End": DATE("2012-12-04")}
  "Branch_bank":{
                          " _id": "45679" }}
```

3.2.2. *Composition:*

In ORDB, the composition relationship is represented by declaring Nested table in the whole class which stores all attributes of the whole part. This relationship will be converted in a strong composition in mongo db between the account and balance documents. The transformation model generates one document account contains embedded collection of balance documents. The result of composition relationship:

Col={NameCol, NameDoc, Doc (Fields), Embedded (Bitemporal_Data), Embedded(Collection (Part_ Fields))} (18)

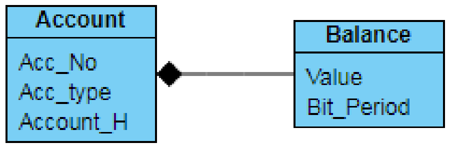

Figure2. An example of Composition Class diagram

The corresponding TJson document of the example above:

Col={ Account, Account _Json Document , (_id, Num_Accout, Type_Account), account_h (VT_start, VT_End,TT_Start, TT_End),Balance{Value,Balance_h(VT_start,VT_End,TT_Start, TT_End)})

Account document embedded the balance document:

Document4: Extraction of balance Json File

```
  {
    "_id":65789
     "Num_Accout": 2
     "Type_Account": "saving_account"
        "Bitemporal_Period":
          {
           "VT_start": "2015-01-01"
           "VT_End": "9999-12-31"
           "TT_Start":  "2015-01-01"
            "TT_End": "9999-12-31"
              }
 "Balance":[ {
          " value":324561.098

            "Blance_h":
          "VT_start": "2015-01-01"
            "VT_End":  "9999-12-31"
            "TT_Start": "2015-01-01"
             "TT_End": "9999-12-31"  }
     {
            " value": 4356.098

             " Blance_h ":
     {
        "VT_start": "2019-07-24"
            "VT_End":  "2019-09-23"
            "TT_Start": "2019-07-25"
             "TT_End": "2019-09-27"
      }]}
```

3.2.3. *Aggregation:*

An aggregation represents a binary relationship. It is a weak form of composition where the part is shareable and independent from the whole, and its properties can be linked with more than one whole class component. For example, if all the composites (whole) are deleted, the sheared part can be existed.

For the aggregation Relationship, the relation is identified a collection of (UDT) in addition of bitemporal data. Then, the branch bank can be composed by one or more than one shareable and existence independent collection. In mongodb, the Aggregation relationship will be considered as a referencing collection of _ID document represented the account document:

Col1={NameCol1,NameDoc,doc(Fields),Embedded(Bitemporal_Period)}} (19)

Col2={NameCol2,NameDoc2,Doc(Fields),Referencing(collection(Doc1)+BitemporalPeriod),Embedded(Bitemporal_Period)}} (20)

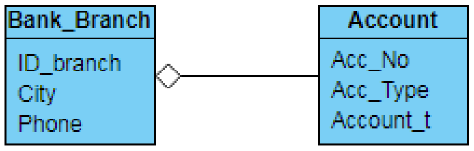

Figure3. Aggregation relationship Class

The transformation will be defined as follow:

Col1={ Account, Account _Json Document , (_id, Num_Accout, Type_Account), account_h (VT_start, VT_End,TT_Start, TT_End))

Col2={Branch_Bank, Banch_bankdocument, {_id, branch_id, city,phone},Account(collection(_ID , account_h (VT_start, VT_End,TT_Start, TT_End)} , Banch_bank_h(VT_start, VT_End,TT_Start, TT_End)}

The following TJson document presents the example above:

Document5: Temporal Json File for Balance_Bank

```
{
   "_id": 678
    "branch_id": 9
    "city": "Casablanca"
```

```
           "phone":074467543

         Accounts :[ {
           "Account":56432
          "Account_h":
                 {
            "VT_start": "2019-08-22"
                 "VT_End":  "9999-12-31"
                 "TT_Start": "2019-08-25"
                  "TT_End": "9999-12-31"  }}
          {
           "Account":980654
          " Account_h ":
                 {
            "VT_start": "2018-07-24"
                 "VT_End":  "2019-09-23"
                 "TT_Start": "2018-07-25"
                  "TT_End": "2019-09-27"  }
       }]}
```

The example below presents the TORDB Query Creation for Branch Bank and Account Tables:

Query 3: creation statement of the aggregation relationship

```
Create type Account_NT as object
( Account Account_T,
Account_h bitemporal_period) /

CREATE TYPE Account  IS TABLE OF  Account_NT ;
Create type branch_bank_type as object (
Branch_Num number,
Address varchar(40),
City varchar(20),
Accounts  Account,
Branch_h bit_period)

CREATE TABLE branch_bank_table of
branch_bank_type NESTED TABLE  accouns STORE
AS accounth_tab, NESTED TABLE branch_h  STORE
AS branch_tab ;
```

3.2.4. Inheritance:

Is called also a generalization is a relationship between two classes or more , where one entity represent a parent or super class and the other one is considered as a child or sub class . The child inherits the behavior and all the properties of the parent.

The inheritance is a very important In ORDB. For the creation statement of UDT that represents the inheritance, we add the keyword under for the sub class. This model will be transformed in Mongo db by generating two documents separately modeling transaction and transfers Types. The transfer document maintains the same structure of transaction type with Additional properties of subtype is defined in the usual way with time varying features.

Col1={NameCol1,NameDoc,Doc(Fields), Embedded(Bitemporal_Period)}} (21)

Col2={NameCol2,NameDoc,doc(Fields+doc1(Fields)), Embedded(Bitemporal_Period)}} (22)

The details are illustrated in the following example:

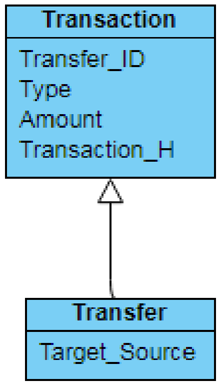

Figure4. Inheritance Relationship Example

Col1={Transaction, Transaction_Json Document doc(_id, trans_ID, Type_Trans, Amount), Transaction_h h (VT_start, VT_End,TT_Start, TT_End)}

Col2={Transfer, Transfer_Json Document ,(_id, trans_ID, Type_Trans, Amount, target_source), Transaction_h h (VT_start, VT_End,TT_Start, TT_End)}

The following TJson document presents inheritance relationship:

Document6: Temporal Json File for Transfer class

```
 {
   "_id":87654
    " trans_ID ": 34
    " Type_Trans": "transfer"
     "Account":9876
    "Amount":  23450
    "Transaction_h":
           {
      "VT_start": "2018-04-02"
           "VT_End":  "2018-04-05"
           "TT_Start": "2018-04-02"
            "TT_End": "2018-04-04" }}
 "target_source ": "soumiya}
```

Ihneritace Creation Query in TORB using Under Keyword:

Query 4: creation statement for inheritance relationship

```
Create or Replace type  Transaction_Type as object
(trans_ID number,
Type_Trans varchar(20),
Amount number,
Account REF account_type,
Transaction_h  bitemporal_period) NOT FINAL ;
Create table transaction_table of Transaction_type
NESTED TABLE Transaction_h  STORE AS
transactionh_tab;

Create or Replace type Transfer_T  UNDER
Transaction_Type (target_source varchar(20)) ;
```

4. Conclusion

This work provides a new approach to the problem of transforming and migrating of massive data based on temporal object Relational database into Temporal Json Documents using Mongodb. We presented the basics phases of modeling and converting Temporal Object relational including User defined time with Bitemporal data into oriented document database. To do that, we formalized the rules by specifying the basics steps involved in the temporal object-relational database design, in order to capture the relationship's type between objects. This solution is done by providing a TORDB Model from an Object relational database that contains user defined Time, and we use it as an input enriched with additional semantic data, which is translatable into any of the target database schemas. Furthermore, Temporal Json document design has defined including the varying time data by exploiting the range of powerful concepts provided by Nosql database. Currently, any paper deals with the transformation process and its functionalities from TORDB into Mongodb. However, this article proposes an analyst part for our future work, where we will implement a complete framework for automating the migration mechanism of the schema, structure and the data into MongoDb without any human interference.

Conflict of Interest

The authors declare no conflict of interests regarding the publication of this paper.

References

[1] K. Kulkarni., J.E. Michels, "Temporal features in SQL: 2011", ACM SIGMOD Record, **41**(3), 34--43, 2012. , https://doi.org/10.1145/2206869.2206883

[2] M. Kvet, K. Matiasko, "Transaction management inthefully temporal system" , in 20114 16th International Conference on Computer Modelling and Simulation (UKSim), Cambridge, UK, 2014. https://doi.org/ 10.1109/UKSim.2014.26

[3] M. Kaufmann, P. M. Fischer, N. May, D. Kossmann, "Benchmarking Bitemporal Database Systems: Ready for the Future or Stuck in the Past?" EDBT, 738-749, 2014. DOI: 10.5441/002/edbt.2014.80

[4] Z.Gansen, L.Qiaoying, L.Libo, L.Zijing. "Schema Conversion Model of SQL Database to NoSQL". In 2014 Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing. Guangdong, China, 2014. DOI: 10.1109/3PGCIC.2014.137

[5] Byrne, B, Nelson, David and Jayakumar, "Big Data Technology - Can We Abandon the Teaching of Normalisation?", in 2017 I9th annual International Conference on Education and New Learning Technologies, Barcelona, Spain, 2017. DOI: 10.21125/edulearn.2017.1113

[6] A. Boicea, F. Radulescu and L. I. Agapin, "MongoDB vs Oracle -- Database Comparison", In 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, Bucharest, 2012. DOI: 10.1109/EIDWT.2012.32

[7] T. Fouad, M. Bahaj. "Model Transformation From Object Relational Database to NoSQL Document database". In 2019 International Conference on Networking, Information Systems & Security, Rabat, Morocco , 2019 . https://doi.org/10.1145/3320326.3320381

[8] R. Lawrence, "Integration and Virtualization of Relational SQL and NoSQL Systems Including MySQL and MongoDB", in 2014 The International Conference on Computational Science and Computational Intelligence , Las Vegas, NV, USA, 2014 , DOI: 10.1109/CSCI.2014.56

[9] A. Singh, "Data Migration from Relational Database to MongoDB".Global Journal of Computer Science and Technology:C Software & Data Engineering ,Vol 19,Issue 2 ,2019.http://dx.doi.org/10.2139/ssrn.3372802

[10] T. Jia, X. Zhao, Z. Wang, D. Gong and G. Ding, , "Model Transformation and Data Migration from Relational Database to MongoDB", in 2016 IEEE International Congress on Big Data (BigData Congress), San Francisco, CA, 2016. DOI**:** 10.1109/BigDataCongress.2016.16

[11] L. Stanescu, M. Brezovan, D.D. Burdescu , "An Algorithm for Mapping the Relational Databases To MongoDB--A Case Study", International Journal of Computer Science & Applications, **14**(1), 2017.

[12] L.Stanescu, M.Brezovan , CS. Spahui, DD. Burdescu ,"A framework for mapping the mysql Databases to Mongodb– Algorithm, Implementation and experiments. International Journal of Computer Science and Applications, **15**(1), 65 – 82, 2018

[13] D. Chauhan, K.L. Bansal, "Using the Advantages of NoSQL: A case study on MongoDB", International Journal on Recent and Innovation Trends in Computing and Communication, **5**(2), ISSN 232/-8169.2017

[14] S. Brahmia, Z. Brahmia, F. Grandi, R. Bouaziz, "A Disciplined Approach to Temporal Evolution and Versioning Support in JSON Data Stores," In Emerging Technologies and Applications in Data Processing and Management", IGI Global, 114-133, 2019. DOI: 10.4018/978-1-5225-8446-9.ch006

[15] Y. Widyani, H. Laksmiwati and E. D. Bangun, "Mapping spatio-temporal disaster data into MongoDB",In 2016 International Conference on Data and Software Engineering (ICoDSE), Denpasa, Indonesia, 2016. DOI**:** 10.1109/ICODSE.2016.7936157

[16] S .Ain El Hayat,F. Toufik,M. Bahaj, "UML/OCL based design and the transition towards temporal object relational database with bitemporal data". Journal of King Saud University - Computer and Information Sciences, 2019. https://doi.org/10.1016/j.jksuci.2019.08.012