

# STORING DATA IN A DOCUMENT-ORIENTED DATABASE AND IMPLEMENTED FROM A STRUCTURED NESTING LOGICAL MODEL

Y. Hiyane, A. Benmakhlof and A. Marzouk

Computer, Networks, Mobility and Modeling Laboratory (IR2M),  
Faculty of Science and Technology,  
University Hassan 1st, BP 577, 26000 Settat, Morocco

## **ABSTRACT**

*Given the exponential increase in data volumes, the variety of data types, and the complexity of processing, researchers have begun to look at another type of database model that is moving away from the classic rules of the relational model. Among the NoSQL models proposed in a set of research works, there is the document-oriented model. This model relies on the key-document pair. The documents are JSON or XML type. A single key can thus retrieve all information hierarchically and nested, which would require multiple joins in the SQL model. In this work we propose a method of storing data in a nested document-oriented database. The latter will be obtained from a standardized conceptual model by applying the transformation rules from the data conceptual model (DCM) to the nested document-oriented model (NDOM). A performance study will also be carried out in this work to show NDOM model ability to process massive data through aggregation queries.*

## **KEYWORDS**

*Big data, No-Relational, Conceptual data modelling, The NoSQL logical data model, Nested document-oriented model, Json MongoDB, Analysis axes.*

## **1. INTRODUCTION**

Nowadays, in the air of big data, the use of non-relational database management systems continues to increase. The data models managed by the relational database management systems are begin to show their limits of the need to increase the quantity and the diversity of data to be processed on the one hand, and to reduce the response time to complex queries. Among the non-relational models more and more use can be cited the document-oriented model (DOM) [1][2][3]. In a NoSQL database document-oriented, a row in a relational table is a structured document (eg, a JSON object or an XML document), and a table is a collection of documents [4]. The difference with relational data structuring is that a collection can contain documents of different structures and the documents themselves can contain other nested documents (dynamic diagram) [5]. Several logical data models have been proposed by authors based on normalised data conceptual models such as the UML class diagram. a partial-nesting logic model has been proposed by K.Shinoù and all [6] where the partial normalised of the conceptual data model (CDM) is maintained and the other part is denormalized by nesting. in another research work, the author has proposed a document-oriented logical model with total and structured nesting (NDOM) [7]. This last model is obtained by applying transition rules from the CDM to the DOM. The absence of joins in this model will significantly reduce the response time to requests that request data from multiple collections. But on the other hand, we will be confronted with a set of anomalies that are

due to denormalization. The major problem will be the insertion of new records in this type of database, which is completely denormalized, while avoiding redundancies.

In this work we propose a method to allow the fast insertion of the records in a non-relational database obtained from a document-oriented model with total nesting NDOM, using a python script which allows multiple insertion into different databases based on Json files. An experimental study of performance is also carried out in this work, in order to put into highlight the capacity of the NDOM model to quickly process queries with syntheses on multi-collections data. A comparative study of the response times is carried out in three DBMS: MongoDB, Oracle and PostgreSQL. In this study we take as an example of application a data conceptual model of an e-commerce organization. This model will be transformed, on the one hand into a nested document-oriented logic model that will be implemented in MongoDB, and on the other hand into a relational logic model that will be implemented in Oracle and PostgreSQL.

## **2. LOGICALMODEL OF DATA-ORIENTED DOCUMENT**

The Class Model Obtained by the UML design will be used to get a non-relational logic model that is based on document-oriented data structuring. Work has been done to define rules for moving from the Conceptual Data Model (CDM) to the Document-Oriented Logic Model (DOM) [6] [7]. Each class in the class-diagram becomes a collection in the DOM. The instances of each class become documents in the collection. The attributes of each class become attributes in the collection. The associations between the classes are also transformed in the DOM. These transformations depend of the Cardinality's that are defined in the CDM [7]. In the table-1, we summarize the different rules of transition from the CDM to the DOM in the two cases: the DOM with Joins and Nested (JDOM and NDOM). For each situation of the CDM we give the different collections created in DOM with for each of them the attributes to insert.

## **3. THE DATA STORAGE IN A NESTED DOCUMENT-ORIENTED DATABASE**

### **3.1. Using Aggregate and Lookup Query**

The two models, with join and nested (JDOM and NDOM) will be used for optimal insertion of new data. These will be inserted at first in the database with join to avoid redundancies. Using an aggregation and lookup JSON query on this normalized database, an insertion of a new document will be done in the nested DB. This document will include the data of the class of the side several plus a sub-document corresponding to the class of the side 1. This new document can be removed from the normalized DB after it is inserted into the nested master collection. These different operations can be triggered automatically using a system function that we can create in the document-based database management system -mongodb- (see figure-1).

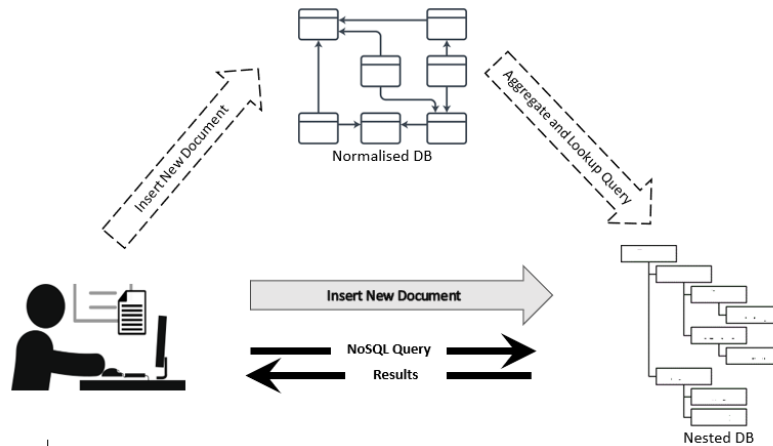


Figure 1. Diagram of insertion of a new document in a document-oriented DB with structured nesting.

- Function-1: the document insertion in the collection of the class of the side several

**InsertNewDoc\_CClassB (x, y, ...,e){**

```
db.CClassB.insert({
    "idB": x,           //Primary Key of Classe B
    "Att_B1": y,
    "Att_B2": z,
    .
    .
    "idA": e,          //Foreign Key of Classe A
});
InsertNewDoc_CP(x);
db.CClassB.remove({"idB": a});}
```

- Function-2: the insertion of the same document into the nested main collection:

```
InsertNewDoc_CP(x){var result = db.CClassB.aggregate([{"$match": {"idB": x }},
{
    $lookup:
    {
from: "CClassA",
localField: "idA",
foreignField: "idA",
as: "NClassA"
    }]);
db.CP.insert(result.toArray());
}
```

### 3.2. Using Python Script to Build Nesting Model

In this section we propose a method to build the nested model that will later be used to optimize the response time of very complex analysis requests, the idea is to execute a python script that takes input of Json objects. generated from the relational production database and subsequently nested in a logical way that respects the NDOM model already explained the script will have an

intermediary role between the production database and the nested database as we can obviously use it to insert into several databases with different structures in order to make comparisons based on the same data

#### 4. PERFORMANCE STUDY:

In order to demonstrate the advantages of structured nesting modelling, we will perform, in this part, a comparative study of response times for complex analysis queries. The conceptual data model used example to conduct this study is given in Figure-2. It is a class diagram that models the data of an e-commerce activity [6]. The implementation of the logical document-oriented model will be done in the MongoDB DBMS, while the relational logic model will be implemented in the Oracle and PostgreSQL. To study the performance of each model we will execute several types of analysis queries with grouping. these queries are given according to one, two, three and four analysis axes. The syntax of these queries will be given in SQL under Oracle and then in JSON under MongoDB.

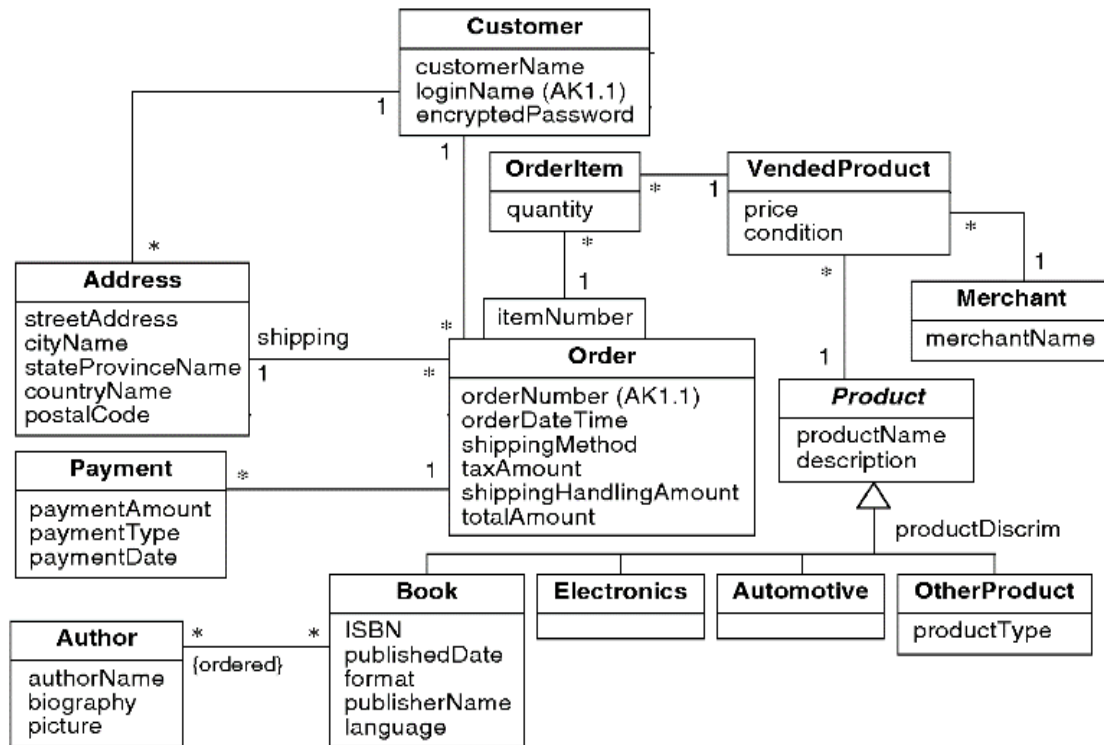
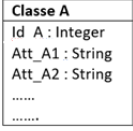
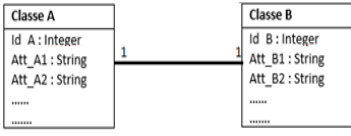
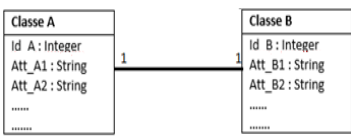
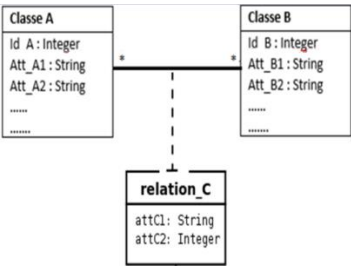
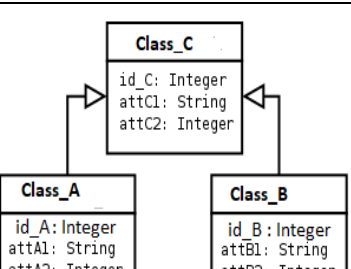


Figure 2. Conceptual Data Model (Class Diagram)

Tableau-1 : Rules for switching from CDM to JDOM and NDOM

MCD	JDOM	NDOM
	$C^{classA} = \{Id\_A, Att\_A1, Att\_A2,.. \}$	$C^{classA} = \{Id\_A, Att\_A1, Att\_A2,.. \}$
	$C^{ClassA} = \{id_A, Att_{A1}, Att_{A2}, .., id_B \}$ $C^{ClassB} = \{id_B, Att_{B1}, att_{B2}, .. \}$	$C^{ClassA} = \left\{ \begin{matrix} id_A, Att_{A1}, \\ Att_{A2}, \dots, id_B, \\ id_B, \\ Att_{B1}, att_{B2}, \dots \end{matrix} \right\}$
	$C^{ClassB} = \{id_B, Att_{B1}, Att_{B2}, .., id_A \}$ $C^{ClassA} = \{id_A, Att_{A1}, att_{A2}, .. \}$	$C^P = \left\{ \begin{matrix} id_B, Att_{B1}, \\ Att_{B2}, \dots, id_A, \\ id_A, \\ Att_{A1}, att_{A2}, \dots \end{matrix} \right\}$
	$C^{ClassB} = \{id_B, Att_{B1}, Att_{B2}, \dots \}$ $C^{ClassA} = \{id_A, Att_{A1}, att_{A2}, \dots \}$ $C^{ClassAB} = \{id_A, id_B, att_{C1}, att_{C2} \}$	$C^{AB} = \left\{ \begin{matrix} id_A, id_B, att_{C1}, att_{C2}, \\ id_A, \\ Att_{A1}, att_{A2}, \dots, \\ id_B, A \\ att_{B1}, att_{B2}, \dots \end{matrix} \right\}$
	$C^{ClassC} = \{id_C, Att_{C1}, Att_{C2}, type \dots \}$ $C^{ClassA} = \{id_A, id_C, Att_{A1}, att_{A2}, \dots \}$ $C^{ClassB} = \{id_B, id_C, Att_{B1}, att_{B2}, \dots \}$	$C^P = \left\{ \begin{matrix} id_C, att_{C1}, \\ att_{C2}, type, \\ id_A, id_C, \\ Att_{A1}, att_{A2}, \dots, \\ id_B, id_C, \\ Att_{B1}, att_{B2}, \dots \end{matrix} \right\}$

#### 4.1. Query1: One Analysis Axe: Total Amount of Sales by Customer

##### SQL Oracle:

```
SELECT customers.CustomerID, customers.CustomerName, Sum(Price*Quantity) AS CA
FROM customers INNER JOIN (adress INNER JOIN (orders INNER JOIN (orderitem INNER
JOIN vendedproduct ON vendedproduct.VendedProductID=orderitem.VendedProductID) ON
orderitem.OrderNumber=orders.OrderNumber) ON orders.AdressID=adress.AdressID) ON
adress.CustomerID=customers.CustomerID GROUP BY customers.CustomerID,
customers.CustomerName;
```

##### JSON Mongodb:

```
db.OrderItemIMB.aggregate([ {$group: {'_id':{Client:"$ORDER.ADRESS.CUSTOMERS"}},
```

total: { \$sum: { \$multiply: [ "\$VENDEDPRODUCT.Price" , "\$Quantity" ] } } } )

	Mongodb	Postgresql	Oracle
100 K	0,079	18	11
1M	0,52	183,6	121,9
5M	0,589	918	784,3
10 M	9,81	1773,6	1492,8

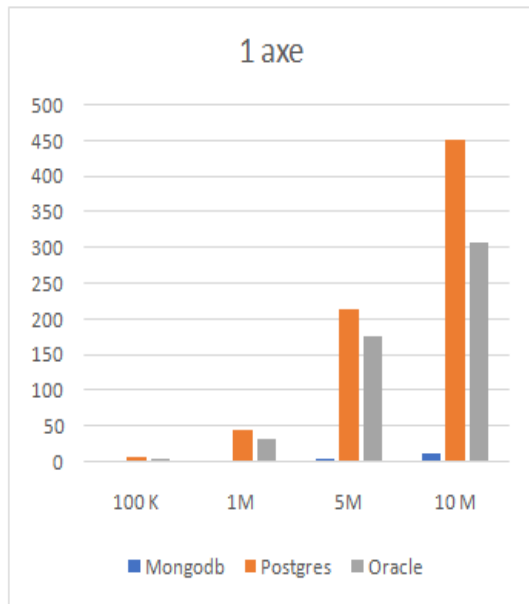


Figure 3. Response time of the query-1 in the three DBMS for different number of processed line

#### 4.2. Query2: Two Analysis Axes: Total Sales by Customer and Year

##### SQL Oracle :

```
select Customers.CustomerID, Customers.CustomerName, EXTRACT(year from
Orders.DateOrder) as ANN, SUM(Price*quantity) as CA
FROM (((CUSTOMERS INNER join ADRESS on
CUSTOMERS.CUSTOMERID=ADRESS.CUSTOMERID) inner join ORDERS on
ADRESS.ADRESSID=ORDERS.ADRESSID)
inner join ORDERITEM on ORDERITEM.ORDERNUMBER=ORDERS.ORDERNUMBER)
inner join VENDEDPRODUCT on vendedproduct.vendedproductid =
ORDERITEM.VENDEDPRODUCTID
group by Customers.CustomerID, Customers.CustomerName, EXTRACT(year from
Orders.DateOrder);
```

##### JSON Mongoddb:

```
db.OrderItemIMB.aggregate([ { $group: { _id: { Client: "$ORDER.ADRESS.CUSTOMERS",
ANN: "$ORDER.ANN" } , total: { $sum: { $multiply: [ "$VENDEDPRODUCT.Price"
, "$Quantity" ] } } } } ])
```

	Mongodb	Postgresql	Oracle
100 K	0,0704	1,8	0,92
1M	0,426	51,6	31,1
5M	0,509	282	178
10 M	7,86	544,8	382

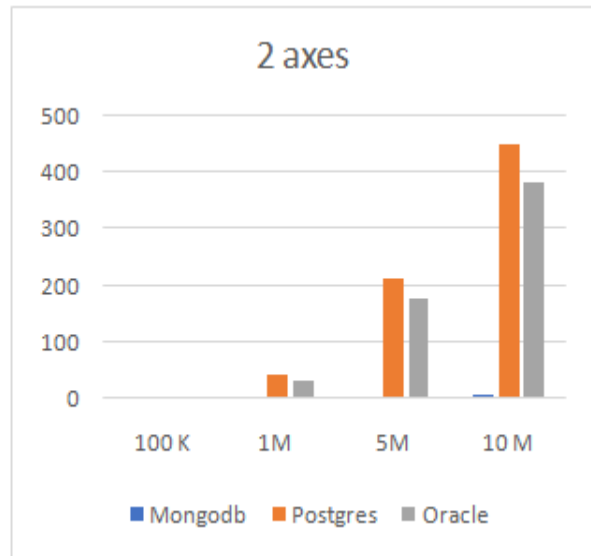


Figure 5. Response time of the query-2 in the three DBMS for different number of processed lines

### 4.3. Query3: Three Analysis Axes: Total Sales by Customer, Product Type and Year

#### SQL Oracle

```
SELECT Customers.CustomerID, Customers.CustomerName, Extract(year from DateOrder) AS
ANN, Products.type,
Sum(Quantity*Price) AS CA FROM Products INNER JOIN (VendedProduct INNER JOIN
(((Customers INNER JOIN Adress ON Customers.CustomerID = Adress.CustomerID)
INNER JOIN Orders ON Adress.AdressID = Orders.AdressID) INNER JOIN OrderItem ON
Orders.OrderNumber = OrderItem.OrderNumber) ON VendedProduct.VendedProductID =
OrderItem.VendedProductID) ON Products.ProductID = VendedProduct.ProductID
GROUP BY Customers.CustomerID, Customers.CustomerName, Extract(year from DateOrder),
Products.type;
```

#### JSON Mongodb :

```
db.OrderItemIMB.aggregate([{$group: { _id: {Client:"$ORDER.ADRESS.CUSTOMERS",
TYPE:"$VENDEDPRODUCT.PRODUCT.type",ANN:"$ORDER.ANN"} ,
total: { $sum: { $multiply: [ "$VENDEDPRODUCT.Price" , "$Quantity" ] } } } ]])
```

	Mongodb	Postgresql	Oracle
100 K	0,079	18	11
1M	0,52	183,6	121,9
5M	0,589	918	784,3
10 M	9,81	1773,6	1492,8

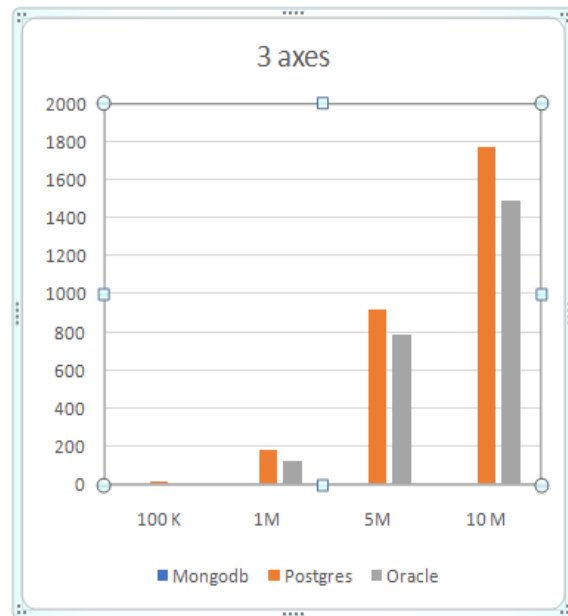


Figure 5. Response time of the query-3 in the three DBMS for different number of processed lines

#### 4.4. Query4: Four Analysis Axes: Total Sales By Merchant, Customer, Product Type And Year

##### SQL Oracle

```
SELECT Merchants.MerchantID, Merchants.MerchantName, Customers.CustomerID,
Customers.CustomerName, Extract(year from DateOrder) AS ANN, Products.type,
Sum(Price*Quantity) AS CA FROM Merchants INNER JOIN
(Products INNER JOIN(VendedProduct INNER JOIN (Customers INNER JOIN (Adress INNER
JOIN (Orders INNER JOIN OrderItem ON Orders.OrderNumber = OrderItem.OrderNumber)
ON Adress.AdressID = Orders.AdressID) ON Customers.CustomerID = Adress.CustomerID) ON
VendedProduct.VendedProductID = OrderItem.VendedProductID)
ON Products.ProductID = VendedProduct.ProductID) ON Merchants.MerchantID =
VendedProduct.MerchantID
GROUP BY Merchants.MerchantID, Merchants.MerchantName, Customers.CustomerID,
Customers.CustomerName, Extract(year from DateOrder), Products.type;
```

##### JSON Mongodb

```
db.OrderItemIMB.aggregate([{$group: {
_id:{Merchant:"$VENDEDPRODUCT.MERCHANT",
Client:"$ORDER.ADRESS.CUSTOMERS",TYPE:"$VENDEDPRODUCT.PRODUCT.type",
ANN:"$ORDER.ANN"} , total: { $sum: { $multiply:["$VENDEDPRODUCT.Price"
,$Quantity]}}}])
```



	Mongoddb	Postgresql	Oracle
100 K	0,072	24	14
1M	0,543	246	189,5
5M	0,702	1222,8	976,2
10 M	10,4	2417,4	1778,5

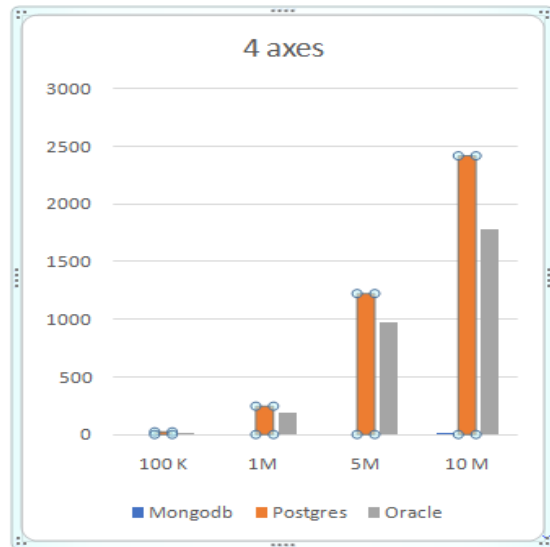


Figure 6. Response time of the query-4 in the three DBMS for different number of processed lines

#### 4.5. Results Analysis

By analysing these four graphs (3-6), we clearly see that the response time in the MongoDB database is still very low compared to the response time required to execute the same queries in the Oracle and PostgreSQL databases. For example, for a four-axis analysis query the response time to process 29,960 lines decreases by -87.83% compared to the Oracle data base and by -95.85% compared the PostgreSQL data base.

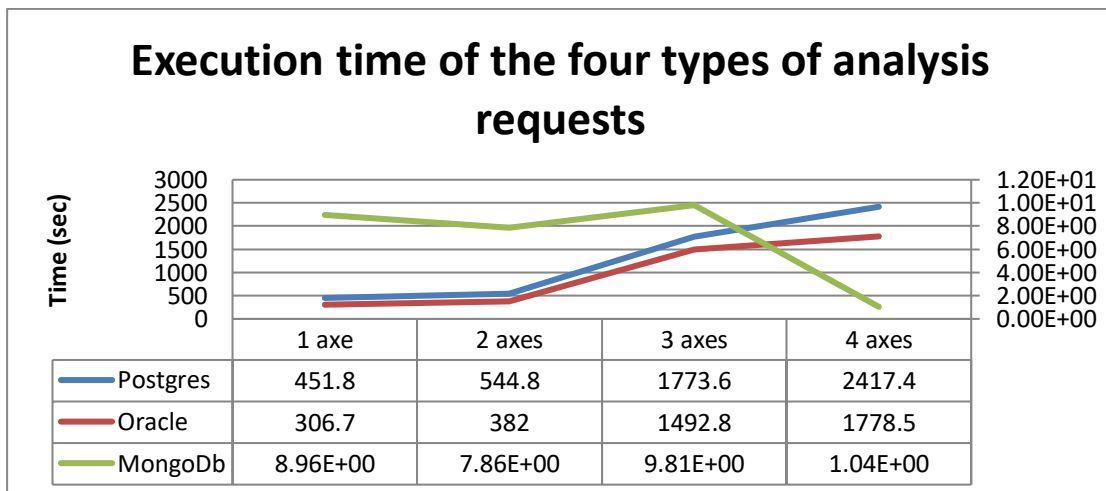


Figure 7. Response time according to analysis axes (10 M)

In figure-7 below we represent the execution time of the four types of analysis query in the three DBMSs and for a number of treated lines of 12800. We note that in the case of logic modelling with structured nesting, the number of analysis axes does not influence the response time of the

queries. The complexity of the analysis queries not increases the response time in the case of the nested model implemented in MongoDB.

The significant optimization in the execution of requests and the response time already presented in the figures (3-6) is due to the advantages of the NoSQL Model as well as the process of structuring data in a nested way used in the NDOM model, we can explain this enormous optimization by the absence of joins in the proposed model however in the normalized relational model the joins are the main factor responsible for the increase in response time of complex requests also that's why we note that the response time increases exponentially for the Standardized Relational Model illustrated in our experiments by Oracle and PostgreSQL also we can conclude that with the increase in the number of data handled and the complexity of axes used in the requests implies an increase in response time in a proportional manner however this is not the case for the NoSQL model illustrated by MongoDB in our experiments it is very clear that the response time in MongoDB is not influenced by the complexity of the requests nor by the amount of data handled for future work we will focus on the introduction of indexes in the NDOM model, an idea which will be able to optimize response time even more we use MongoDB, Oracle, PostgreSQL as database and Python, JSON as technical tools to Power databases also we have proposed a method which allows the automatic transformation of a class diagram to the NDOM Model while respecting passing rules

## 5. CONCLUSION

In this study we have proposed a practical method to feed a database implemented from a document-oriented model with structured total nesting. The problem of the strong redundancy present in this model is overcome by the use of a document-oriented database with join. The latter represents an intermediate data base allowing to easily insert new data because of its normalization. This new data is passed to the nested database using a system function that we can create in the MongoDB database management system. This function will encapsulate a JSON request for aggregation and search which will extract the new data, which are inserted beforehand in the standardized DB, then complete them with all the attached data, then insert them into the nested DB, and finally delete them from normalized database.

In this work we also did a performance study of the nested document-oriented model NDOM by comparing the response times of a set of synthesis queries with clustering in both relational and no-relational document-oriented models. The conceptual data model (CDM) used to conduct this study models the data of an e-commerce activity. The implementation of the document-oriented logic model is carried out in the MongoDB DBMS while the relational logic model is implemented in Oracle and PostgreSQL RDBMS. This study clearly demonstrated the advantage of NDOM. Indeed, the response time of the different analysis requests is much smaller than the one used to answer the same requests and on the same amounts of data in the case of an MR implemented in Oracle or PostgreSQL.

Another more remarkable result in this study is that in the NDOM the degree of complexity (number of joins) of the query does not have much impact on the response time. On the other hand, in relational model, the response time increases strongly with the number of joins.

## REFERENCES

- [1] Benmakhlouf, A., 2018, "NoSQL Implementation of a Conceptual Data Model: UML Class Diagram to a Document-Oriented Model.", "International Journal of Database Management Systems (IJDMS)".
- [2] Chaudhuri, S., Dayal, U., 1997. An overview of data warehousing and olap technology. *SIGMOD Record*, 26, ACM, pp. 65–74.
- [3] Colliat, G., 1996. Olap, relational, and multidimensional database systems. *SIGMOD Record*, 25(3), ACM, pp. 64–69.
- [4] Cuzzocrea, A., Bellatreche, L., Song, I.-Y., 2013. Data warehousing and olap over big data: Current challenges and future research directions. 16th Int. Workshop on Data Warehousing and OLAP (DOLAP), ACM, pp. 67–70.
- [5] Dede, E., Govindaraju, M., Gunter, D., Canon, R. S., Ramakrishnan, L., 2013. Performance evaluation of a mongodb and hadoop platform for
- [6] scientific data analysis. 4th Workshop on Scientific Cloud Computing, ACM, pp. 13–20.
- [7] Dehdouh, K., Boussaid, O., Bentayeb, F., 2014. Columnar nosql star schema benchmark. *Model and Data Engineering*, LNCS 8748, Springer, pp. 281–288.
- [8] Golfarelli, M., Maio, D., and Rizzi, S., 1998. The dimensional fact model: A conceptual model for data warehouses. *Int. Journal of Cooperative Information Systems*, 7, pp. 215–247.
- [9] Gray, J., Bosworth, A., Layman, A., Pirahesh, H., 1996. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. *Int. Conf. on Data Engineering (ICDE)*, IEEE Computer Society, pp. 152-159.
- [10] Han, D., Stroulia, E., 2012. A three-dimensional data model in hbase for large time-series dataset analysis.
- [11] Vajk, T., Feher, P., Fekete, K., Charaf, H., 2013. Denormalizing data into schema-free databases. 4th Int. Conf. on Cognitive Infocommunications (CogInfoCom), IEEE, pp. 747–752.
- [12] Vassiliadis, P., Vagena, Z., Skiadopoulos, S., Karayannidis, N., 2000. ARKTOS: A Tool For Data Cleaning and Transformation in Data Warehouse Environments. *IEEE Data Engineering Bulletin*, 23(4), pp. 42-47.
- [13] TPC-DS, 2014. Transaction Processing Performance Council, Decision Support benchmark, version 1.3.0, <http://www.tpc.org/tpcds/>.
- [14] Wrembel, R., 2009. A survey of managing the evolution of data warehouses. *Int. Journal of Data Warehousing and Mining (ijDWM)*, 5(2), IGI Publishing, pp. 24–56.
- [15] A B M Moniruzzaman and Syed Akhter Hossain, 2013, "NoSQL Database: New Era of Databases for Big-data Analytics - Classification, Characteristics and Comparison,". *International Journal of Database Theory and Application*.
- [16] Veronika Abramova, Jorge Bernardino and Pedro Furtado, 2014, "Experimental Evaluation Of NOSQL DataBase", *International Journal of Database Management Systems (IJDMS) Vol.6, No.3, June 2014*

- [17] Y.Hiyane, A.Benmakhlouf, A.Marzouk, 2018, "Storing data in NOSQL data warehouses." Proceeding of International Conference on Control, Automation and Diagnosis, IEEE Publications.
- [18] Christine Niyizamwiyitira and Lars Lundberg, "Performance Evaluation Of SQL and NOSQL DataBase Management Systems in a Cluster", International Journal of Database Management Systems (IJDMS) Vol.9, No.6, June 2017

## **AUTHOR**

**Hiyane Youssef** Doctoral researcher at the Faculty of Science and Technology of Settat Hassan First University. 2018. "Storing data in NOSQL data warehouses: Document-Oriented Model " proceedings ICCAD\_18: Paper70.

