Annals of Emerging Technologies in Computing (AETiC)
Vol. 5, No. 5, 2021

Research Article

A Formal OLAP Algebra for NoSQL based Data Warehouses

Shreya Banerjee[1,*], Sourabh Bhaskar[2], Anirban Sarkar[3] and Narayan C. Debnath[1]

[1]Eastern International University, Binh Duong, Vietnam
shreya.banerjee@eiu.edu.vn; narayan.debnath@eiu.edu.vn
[2]Sardar Vallabhbhai National Institute of Technology, India
sourabhb440@gmail.com
[3]National Institute of Technology, India
sarkar.anirban@gmail.com
*Correspondence: shreya.banerjee@eiu.edu.vn

Received: 17th October 2020; Accepted: 15th November 2020; Published: 20th March 2021

Abstract: NoSQL solutions are started to be increasingly used in modern days' Data Warehouses (DW). However, business analysts face challenges when performing On Line Analytical Processing (OLAP) queries on these NoSQL systems. The lack of uniform representation of various OLAP operations over different types of NoSQL based DWs is one of them. In addition, deficiency of precise semantics in OLAP operations create obstacles to effective query interpretation over distinct types DWs. This paper is aiming to deal with aforementioned challenges. Formal and rigorous specification are represented in this paper for different kinds of OLAP operators and operations. These precise specifications are capable to analyse business queries. Further, the proposed formal specifications are implemented in a document-oriented database using a suitable case study. In addition, the proposed approach aids efficient visualization techniques of data cubes over NoSQL based DWs.

Keywords: *Data Cube; NoSQL Data Warehouses; OLAP query algebra; Ontology*

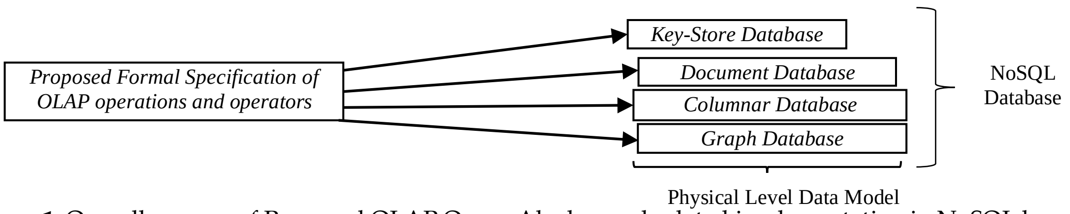
1. Introduction

Modern Data Warehouses (DW) solutions demand to act more in internet-style than to enforce the user to act within predefined structures [1]. Consequently, nowadays DWs need to handle a variety of subject areas, diverse data sources and heterogeneous data types like structured, semi-structured and unstructured. Accordingly, On Line Analytical Processing (OLAP) operations require dealing with related business queries based on that irregular information [2]. To manage these new characteristics of DWs, business analysts focuses on using of NoSQL databases.

Flexible deployment, high read/write efficiency as well as scaling to very large data sets – these are remarkable features of NoSQL databases [3]. Yet, these databases are categorized based on various data models at physical level such as Document Store, Key-Value stores, Graph databases and Column-Family store [4]. Each physical level data model has their own approach towards handling OLAP algebra. In general, every kind of NoSQL database has a query language of its own. For example, Cassandra database has developed Cassandra Query Language (CQL); MongoDB query language is used in MongoDB database; Neo4j database has Cypher query language etc. [5]. Thus, lack of a common specification of OLAP operations over different NoSQL databases make serious problems when DWs using these databases are required to be portable. This challenge creates a research question, that how to provide a uniform standard towards OLAP operations for distinct types of NoSQL based DWs.

Shreya Banerjee, Sourabh Bhaskar, Anirban Sarkar and Narayan C. Debnath, "A Formal OLAP Algebra for NoSQL based Data Warehouses", *Annals of Emerging Technologies in Computing (AETiC)*, Print ISSN: 2516-0281, Online ISSN: 2516-029X, pp. 154-161, Vol. 5, No. 5, 20th March 2021, Published by International Association of Educators and Researchers (IAER), DOI: 10.33166/AETiC.2021.05.019, Available: http://aetic.theiaer.org/archive/v5/v5n5/p19.html.

This paper is aiming to address the aforementioned research question. The research methodology followed in this paper is described next. Ontology is applied to resolve the challenge. It is defined as an explicit specification of shared conceptualization [6]. Axioms are used to enable the ontology to provide enriched and formal semantics towards related concepts. OLAP operations on different NoSQL based DWs are varied due to both syntactic and semantic differences. These variances need to be decreased to get a standard specifications of OLAP operation over disparate NoSQL based DWs [7]. An ontology based specification can provide common conceptualization towards the elements of DW domain in terms of concepts and related axioms. Thus, syntactic differences can be omitted. In this context, the ontology driven conceptual model described in [8] is adopted to express a set of OLAP operators and operations formally. Further, semantics differences among OLAP operations can also be omitted with the help of ontology. Figure 1 has described the overall process. Although, the proposed conceptualization is implemented in a document-oriented database, it can also be implemented in other NoSQL based DWs.



Proposed Formal Specification of OLAP operations and operators
Key-Store Database
Document Database
Columnar Database
Graph Database
NoSQL Database
Physical Level Data Model

Figure 1. Overall process of Proposed OLAP Query Algebra and related implementation in NoSQL based DWs

2. Related Work

In literature, several research works exist related to formalization and implementation of OLAP queries on NoSQL based DW. In [9], authors have described ways to implement columnar NoSQL DW (CN-DW) and OLAP queries in Hbase. In [10], authors are using OLAP queries to know about the popularity in recent tweeter trends. In [11], authors used dice and drill-down operation to evaluate the performance on different enterprise scenarios of columnar family. In [12], authors have proposed an approach where an ontology serves as superimposed conceptual layer between multidimensional data and business analysts. The Ontology based OLAP is proposed using UML (Unified Modelling Language) diagram. In [13], a model is described for extracting OLAP dimensions from document-oriented SQL database based on parallel similarity techniques. In [14], authors have presented a Personalization System based on three interrelated ontologies - resources ontology, DW ontology, and domain ontology. They presented these three ontology models in UML and in OWL (Web Ontology Language). However, in all these approaches any common formal specifications of OLAP operations over distinct NoSQL DWs are not provided.

Majority of existing works described OLAP operations specific to its physical level implementations. However, very few works have focused on formal representation of OLAP algebra. Moreover, very few proposals have addressed how to adapt flexible data for OLAP in NoSQL based DW systems. In this context, this paper proposed a universal OLAP interface for disparate NoSQL based DWs. The proposed uniform OLAP interface is devised based on formal semantics of OLAP operators and operations and further implemented in a document-oriented NoSQL based DW.

3. Summarization of Ontology Driven Conceptual Modelling of NoSQL based Data Warehouses

The conceptual model described in [8] has three main layers namely - *Collection* (Top-Most layer), *Family* (Intermediate Layer) and *Attribute* (Bottom-Most Layer). *Attribute* layer realizes the measure and dimension attributes of DWs. *Family* layer represents fact and dimension hierarchies in DW. Further, the data cubes based on facts are mapped towards *Collection* layer. *Attribute* layer has its construct types - *Attribute* (*AT*). Likewise, *Family* layer has construct type – *Family* (*FA*) and *Collection* layer has construct type *Collection* (*col*). *AT* is the group of all possible instances of a data item. This can be classified in two types namely- *Measure Attribute* (*M_{AT}*) and *Dimension Attribute*

(D_{AT}). *FA* is constructed by grouping several semantically related *AT*. It can be of two types - *Fact Family* (*FF*) and *Dimension Family* (*DF*).

FF has single level. A *DF* can be decomposed into multiple levels to form the dimension hierarchies. *Col* is created from group of *FF* those are semantically related. Thus, from the top level a whole DW can be observed as a group of *col*s. Cube can be created from *FF* and realized as a *col*. Further, using different relationships, distinct types of constructs in the conceptual model are linked with each other. These relationships are of two kinds –Inter-layer kind and Intra-layer kind. *Containment* and *Inverse Containment* relationships are included towards both intra-layer kind and inter-layer kind relationships category. Further, *Association* relationship can only be included towards Inter-layer kind relationship group. In addition, different relationships of this conceptual has distinct properties such as *Cardinality*, *Modality*, and *Ordering*. Figure 2 has illustrated the conceptual model described in [8].

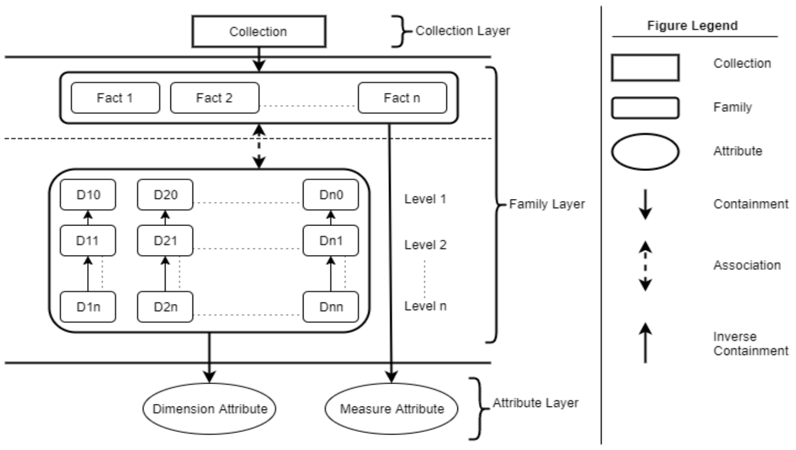


Figure 2. Conceptual model for NoSQL based data warehouses

4. Proposed OLAP Algebra for NoSQL based DWs

Proposed OLAP algebra is classified in two groups. Those groups are OLAP operators and OLAP operations. Two operators are included in the first category namely, *Select* and *Aggregate* operator. On the other hand, five types of operations are included in the second category. These five operations used those two operators.

4.1. Proposed OLAP operators

Formal representations of two OLAP operators are proposed next.

(a) ***Select Operator*** **(π):** This operator will extract the dimension and its hierarchy from dimension family depending on some predicate p. This can be atomic predicate, denoted as p or it can be a composite predicate denoted as $p_1 < op > p_2 < op > \ldots < op > p_n$. In the composite predicate, <op> acts as a logical operator such as AND, OR etc. The p can be either dimensional family (*DF*) or dimension hierarchy (*DFH*). The algebraic notation of the operator is

$$\pi_p\,(DF) = DF_o$$

Here *DF* is the original dimension family on measure and DF_o is the output dimension family on measure after the restriction. Null predicate operator will return the original *DF*. Hence

$$\pi_\emptyset(DF) = DF$$

(b) ***Aggregate Operator*** **(α):** The aggregate operator will perform the grouping function *GF* on measure attribute (M_{AT}) of the specified set of *DFs* in a cube *C*. The *GF* is the relational aggregation function, which will operate on the M_{AT} only. These *GFs* can be *SUM, MIN, MAX, AVG*, and *COUNT*. The algebraic notation of the aggregate operator is

$$\alpha_{GF(M_{AT})\,\{DF1\ V\ DF2\ V\ DF3\ \ldots\ V\ DFn\}}(C)$$

4.2. Proposed OLAP Operations

In this section, five OLAP operations are formally specified.

(a) ***Slice operation*** **(*sl*)**: The slice operation pick out one specific dimension from an input cube and provides a new sub-cube. The algebraic notation for the slice operation is

$$sl(C) = \alpha_{GF(M_{AT})\{DF\},CON}(C)$$

Here, *CON* is the condition defined as,

$$CON = \pi_p(DF)$$

(b) ***Dice operation*** **(*di*):** The dice operation picks two or more dimensions from an input cube and provides a new sub-cube. The algebraic notation for the dice operation is

$$di(C) = \alpha_{GF(M_{AT})\{DF\},CON}(C)$$

Here, *CON* is the condition defined as,

$$CON = \pi_{p1}(DF1) < op > \pi_{p2}(DF2) \ldots < op > \pi_{pn}(DFn)$$

(c) ***Roll-up operation*** **(*Rup*):** The Roll-up operation performs aggregation on a data cube by moving down a dimension in the dimensional hierarchy or by adding a new dimension. The algebraic notation for the Roll-up operation is

$$Rup(DF_{ij})(C) = \alpha_{GF(M_{AT})\{DF_{i(j+1)}\}}(C)$$

Here, the roll-up operation going from higher granularity to lower granularity by increasing the value of *j* by *1* for one level roll-up. If roll-up operation is *2* or more than *2* level up, then the operation (*Rup*) is computed in every level and give the result. Here, *j and i* are used only for indexing purpose and have numerical positive integer. *i* is defined for *DF* and *j* is defined for dimensional hierarchy.

(d) ***Drill-down operation*** **(*Ddn*):** Drill-down is the reverse operation of roll-up. The drill-down operation is performed by stepping up in the dimension hierarchy. Thus, it goes to higher granularity from lower granularity. The algebraic notation for the drill-down operation is

$$Ddn(DF_{ij})(C) = \alpha_{GF(M_{AT})\{DF_{i(j-1)}\}}(C)$$

(e) ***Pivot operation*** **(*pvt*):** The pivot operation delivers an alternate presentation of a data by rotating the data axes in a view. Thus, this operation is also called as rotation. It is about analyzing the combination of pair of selected dimension. The algebraic notation for the pivot operation is

$$\begin{aligned} Pvt(C) &= \alpha_{GF(M_{AT})\{DF1,DF2\}^T}(C) \\ &= \alpha_{GF(M_{AT})\{DF2,DF1\}}(C) \end{aligned}$$

5. Illustration of Proposed OLAP Algebra Using a Case Study

In this section, proposed OLAP algebra is illustrated using a case study described in [8]. The case study is based on sales and shipping. Sales of different products can be done in sale branches. Branches can be located in multiple locations. Shipping can have multiple shippers who will shipped the product from one location to another.

Collections (*Cubes* created from *Fact Families*)
FACT FAMILY 1 (*SALES*)
FACT FAMILY 2 (*SHIPPING*)
SALES(*Location, Branch, Product, Time, units sold, dollars sold*)
SHIPPING (*Location, Shipper, Product, Time, units shipped, dollars shipped*)
Location (location_Id, pin code, {street}, city_Id)
City (city_id, city, state_Id)
State (state_Id, state, country_Id)
Country (country_Id, country)
Branch (branch_Id, branchName)
Product (product_Id, product_Name, productType_Id)
ProductType (productType_Id, productType_Name)
Time (time_Id, time, day_Id)
Day (day_Id, day, month_Id)
Month (month_Id, month, year_Id)
Year (year_Id, year)
Shipper (shipper_Id, shipperName, locaton_Id)

Nomenclature

Collections: In Capitalize and **bold**;
Fact Families: in UPPERCASE and *italic*
Dimension Families: in Capitalize and *italic*
Measure Attributes: in lowercase and *italic*
Dimension Attributes: in lowercase
Optional Construct Type: within {}

Figure 3. Key elements of the specified case study

This case study has two facts – *Sales* and *Shipping*. These two facts may have multiple dimensions with hierarchy. For example, *Sales* is associated with four dimensions - *Location, Branch, Product,* and *Time*. Further, two facts can share their dimensions. Several dimensions have hierarchy and specific attributes. For example, dimension *Time* has hierarchy – *Time→Day→Month→Year* and several attributes such as *Time Id,* and *Time*. In addition, each fact are associated with two measures. For example, *Shipping* is associated with measures *Units Shipped* and *Dollars Cost*. In some cases, attributes of specific dimension either is changed or absent.

Distinct features of this described case study is irregular. This requires flexible data representation. Consequently, NoSQL databases are required to demonstrate these data set in DWs. Figure 3 represents the main elements of the case study as described in [8]. Data cubes related to the case study can be realized through distinct *cols* based on different *FF.*

Several queries founded on the proposed OLAP algebra are demonstrated next using the described case study.

Query 1: Find the derived dimension of *Time* for the month *"November"*.

Select operator (π) is required to accomplish this query. The formal expression of the query is as,

$$\pi_{Time.Day.Month.month="November"}(Time) = T$$

It will yield the derived dimension called *T*, which will be contain the instance of *time_id, time, day_id, day, month_id, month, year_id,* and *year* related to month=*"November"*.

Query 2: The total number of Electronics product type units sold across all of the dimensions (Time, Location, Branch, and Shipper).

Slice operation is required to accomplish this query. The formal expression of the query is as,

$$sl(C) = \alpha_{SUM(unit\_sold)\{Product.productType.productType\_Name\},CON}\ (C)$$
$$CON = (\pi_{Product.productType.productType\_Name}(Product))$$

Here, slice operation is accomplished for the dimension *"Product"* based on the criterion P*roduct.ProductType.productType_Name = "Electronics".*

Query 3: The total unit sold for a particular product type *"electronics"*, city *"Durgapur"* and month *"November"*.

Dice operation is required to accomplish this query. The formal expression of the query is as,

$$di(C) = \alpha_{SUM(units_{sold})\{A,Location.City,Time.Day.Month.month\},CON}\ (C)$$

A=$\{product.ProductType.productType_{Name}$

$$CON = ((\pi_{Product.productType.productType\_Name="Electronics"}(Product))$$
$$\cup\ (\pi_{Location.City.city="Durgapur"}\ (Location))$$
$$\cup\ (\pi_{Time.Day.Month.month="November"}\ (Time)))$$

The dice operation is performed on the cube using the following selection criteria. The criteria involves three dimensions - *Product Type Name = Electronics, City = Durgapur,* and *Month = November.*

Query 4: Find the total unit sold across all product by increasing the aggregation levels of time: from Day to Year (Day→Month→Year).

The roll-up operation is required to accomplish this query.

First step: $Rup(DF_{41})(\mathrm{C}) = \alpha_{\mathrm{SUM(units\_sold)}\{DF_{4(1+1)}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{DF_{42}\}}\ (\mathrm{C})$

Second step: $Rup(DF_{42})(\mathrm{C}) = \alpha_{\mathrm{SUM(units\_sold)}\{DF_{4(2+1)}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{DF_{43}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{Time.Day.Month.Year\}}\ (\mathrm{C})$

Query 5: Find the total units sold across all product by decreasing the aggregation level on time: from year to day (Year→Month→Day).

Drill-down (*Ddn*) operation is required to accomplish this query.

First step: $Ddn(DF_{43})(\mathrm{C}) = \alpha_{\mathrm{SUM(units\_sold)}\{DF_{4(3-1)}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{DF_{42}\}}\ (\mathrm{C})$

Second step: $Ddn(DF_{42})(\mathrm{C}) = \alpha_{\mathrm{SUM(units\_sold)}\{DF_{4(2-1)}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{DF_{41}\}}\ (\mathrm{C})$

$= \alpha_{\mathrm{SUM(units\_sold)}\{Time.Day\}}\ (\mathrm{C})$

Query 6: Analyze the total units sold by product and location.

Pivot operation is required to accomplish this query. The formal expression of the query is as,

$$Pvt(C) = \alpha_{SUM(units\_sold)\{Location,Product\}^T}\ (C)$$
$$= \alpha_{SUM(units\_sold)\{Product,Location\}}\ (C)$$

6. Implementation of Proposed Algebra

In this section, proposed OLAP operators and operations are implemented using MongoDB. The case study described in section *5* is used to illustrate the implementation. The transformation between the conceptual model and MongoDB is described in [8]. Figure 4(a) represents the general form of *Select* operator. Figure 4(b) specifies the general form of *Aggregate* operator. In these figures, D_{ij} is the dimension with related hierarchy. Dimension number is represented through *i* and *j* is used for changing hierarchy level in a particular *i*th dimension. The *XYZ* represents M_{AT}.

db.cube.select**([{**"$match"**:{**"$D_{10}.D_{11}. \dots .D_{1n}$":"value","$D_{20}.D_{21}. \dots .D_{2n}$":"value",..., "$D_{n0}.D_{n1}. \dots .D_{nn}$":"value" **}}])**

(a)

db.cube.aggregate([{"$group":{_id:{*P1*:"\$ $D_{10}.D_{11}. \dots .D_{1n}$",*P2*:"\$ $D_{20}.D_{21}. \dots .D_{2n}$", …,
Pn:"\$ $D_{n0}.D_{n1}. \dots .D_{nn}$"},XYZ:{$GF:"\$$M_{AT}$"}}}])

(b)

Figure 4. (a) General Implementation form of Select operator; **(b)** General Implementation form of Aggregate operator

In these general forms, a cube represents a dimension, or a fact with related dimensions or a view. A dimension and a fact with related dimensions can be implemented as "Collection" in MongoDB [8]. Further, views can be implemented in different ways. The first view is created for a cube that includes both facts (*shipping* and *sales*) present in the case study. This cube also comprises three shared dimensions between these two facts. Figure 5 has illustrated this view. The second view is created for a cube that includes the fact *shipping* and corresponding dimension hierarchy. Figure 6 has illustrated this view. Likewise, the third view can be created for a cube that includes the fact *sales* and corresponding dimension hierarchy.

```
db.createView('dbview','sales', [ {"$lookup":{"from":"shipping","localField":"location_Id---{"$unwind":"$collection5_doc"},
{"$lookup":{"from":"branch","localField":"branch_Id-----
{"$lookup":{"from":"product","localField":"product_Id-----
{"$lookup":{"from":"time","localField":"time_Id-------
{"$lookup":{"from":"location","localField-----
{"$lookup":{"from":"shipper","localField":"collection5_doc.shipper_Id","foreignField":"Shipper.shipper_Id","as":"collection6_
doc"}}, {"$unwind":"$collection6_doc"},
{"$project":{"dollars_cost":"$collection5_doc.dollars_cost","units_shipped":"$collection5_doc.units_shipped","Shipper":"$colle
ction6_doc.Shipper","Branch------}}])
```

Figure 5. MongoDB based illustration of a view consisting of both sales and shipping fact

```
db.createView('shippingView','shipping',[{"$lookup":{"from":"shipper","localField":"shipper_Id","foreignField":"Shipper.shipper
_Id","as":"collection1_doc"}},{"$unwind":"$collection1_doc"},
{"$lookup":{"from":"location","localField------
{"$lookup":{"from":"product","localField-------
{"$lookup":{"from":"time","localField":"time_Id-----
{"$project":{"Shipper":"$collection1_doc.Shipper","Product":"$collection3_doc.Product","Time":"$collection4_doc.Time",---}}])
```

Figure 6. MongoDB based illustration of a view consisting of only shipping fact

Next, query examples described in section *5* are implemented based on proposed OLAP operators and operations in MongoDB.

(1). ***Slice operation*** **(*sl*)**:

Query1: The total number of Electronics product type units sold across all of the dimensions (Time, Location, Branch, and Shipper). The implementation of the above query is specified in figure 7. This query realizes the slice operation of OLAP algebra. In the above query the *"salesView"* will be sliced by the predicate *productType_Name="electronics"*.

(2). ***Dice operation*** **(*di*)**:

Query2: The total unit sold for a product type electronics, city Durgapur and month *"November"*.

The above query is implemented as specified in figure 8. Results of the above query realize dice operation by restricting two dimensions of data cube. The *salesView* will be diced by the predicate *productType_Name="electronics"* and *year="2017"*.

```
db.salesView.aggregate([{"$match": {"Product.ProductType.productType_Name":"electronics"}},
{"$group":{_id:"$Product.ProductType.productType_Name",total_cost:{$sum:"$units_sold"}}}])
```

Figure 7. MongoDB based illustration of the Query 1

```
db.salesView.aggregate([ {"$match":{"Product.ProductType.productType_Name":"electronics","Time.Day.
Month.Year.year":2017}}, {"$group":{_id:{productType:"$Product.ProductType.productType_Name",Year:
"$Time.Day.Month.Year.year"}, total_cost:{$sum:"$units_sold"}}}])
```

Figure 8. MongoDB based illustration of the Query 2

(3). *Roll-up operation* (*Rup*):

Query 3: Find the total unit sold across all product by increasing the aggregation levels of time: from Day to Year (*Day→Month→Year*).

This query can be executed as systematic as specified in figure 9. The query is realized by *Roll-up* operation. According to proposed *Roll-up* operation, Intermediate Cube IC_1 and IC_2 are generated. IC_2 is Roll-up output of IC_1, which is Roll-up output of C_0.

Query C0

```
db.salesView.aggregate([{"$group":{_id:{Day:"$Time.Day.day"}, total_cost:{$sum:"$units_sold"}}}])
```

Intermediate Result 1: IC1

```
db.salesView.aggregate([{"$group":{_id:{Month:"$Time.Day.Month.month"},total_cost:{$sum:"$units_sold"}}])
```

Intermediate Result 1: IC2

```
db.salesView.aggregate([{"$group":{_id:{Year:"$Time.Day.Month.Year.year"},total_cost:{$sum:"$units_sold"}}}])
```

Figure 9. MongoDB based illustration of the Query 3

(4). *Drill-down operation* (*Ddn*):

Query 4: Find the total units sold across all product by decreasing the aggregation level on time: from year to day (*Year→Month→Day*). This query can be executed as systematic as specified in figure 10. The query is realized by *Drill-down* operation. According to proposed *Drill-down* operation, Intermediate Cube IC_1 and IC_2 are generated. IC_2 is *Drill-down* output of IC_1, which is *Drill-down* output of C_0.

Query C0

```
db.salesView.aggregate([{"$group":{_id:{Year:"$Time.Day.Month.monthYear.year"},total_cost:{$sum:"$units_sold"}}}])
```

Intermediate Result 1: IC1

```
db.salesView.aggregate([{"$group":{_id:{Month:"$Time.Day.Month.month"},total_cost:{$sum:"$units_sold"}}}])
```

Intermediate Result 1: IC2

```
db.salesView.aggregate([{"$group":{_id:{Day:"$Time.Day.day"}, total_cost:{$sum:"$units_sold"}}}])
```

Figure 10. MongoDB based illustration of the Query 4

(5). *Pivot operation* (*pvt*):

Query 5: Analyze the total dollars sold in respect to product and Time and vise-versa.

This query realizes the pivot operation. It rotates or transposes the data axes to view the data from different perspective. The implementation of the query is represented in figure 11. Figure 11(a) has represented total dollars sold in respect to Product and Time. On the other hand, figure 11(b) has represented total dollars sold in respect to Time and Product.

```
db.salesView.aggregate([{"$group":{_id:{productType:"$Product.ProductType.productType_Name",Year:"$Time.Day.Month.Year.
year"}, total_cost:{$sum:"$dollars_sold"}}}])
```

(a)

```
db.salesView.aggregate([{"$group":{_id:{Year:"$Time.Day.Month.Year.year",productType:"$Product.ProductType.produ
ctType_Name"}, total_cost:{$sum:"$dollars_sold"}}}])
```

(b)

Figure 11. MongoDB based illustration of the Query 5

7. Conclusion

The lack of uniform representation of OLAP operations over distinct NoSQL based DWs make them less portable. Addressing this challenge, in this paper, an ontology based formal and rigorous specification of OLAP operations are proposed. The main contribution of the proposed work is to provide uniform precise syntax and semantics towards different OLAP operators and operations. These proposed formal specifications are independent of any physical level implementation. Thus, proposed operators are able to be applied in distinct type of NoSQL based DWs. Further, the proposed formal specification is implemented in a document-oriented database MongoDB. Moreover, the proposed approach is suitable for web-scale analytical applications. Future work will include automated query answering through incorporating prescribed formal semantics of OLAP operators in a rule based reasoner. Besides this, another important future work will be automated conversion of formal operators towards specific NoSQL based DWs.

References

[1] Bicevska Z., and Oditis I. (2017). Towards NoSQL-based Data Warehouse Solution, In ICTE 2016, Procedia Computer Science, Riga Technical University, pp. 104-111, Latvia, doi: 10.1016/j.procs.2017.01.080.

[2] Chevalier M., Malki M. El, Kopliku A., Teste O., and Tournier R. (2015). Benchmark for OLAP on NoSQL technologies comparing NoSQL multidimensional data warehousing solution. In 9th Int. Conf. on Research Challenges in Information Science (RCIS), IEEE, pp. 480-485, Athens, DOI: 10.1109/RCIS.2015.7128909.

[3] Oditis I., Bicevska Z., Bicevskis J., and Karnitis G. (2018). Implementation of NoSQL-based Data Warehouses, Baltic Journal of Modern Computing. Riga. vol. 6, no. 1, pp. 45-55, DOI: 10.22364/bjmc.2018.6.1.04.

[4] Hecht R., and Jablonski S. (2011). NoSQL evaluation: A use case oriented survey. In Cloud and Service Computing (CSC '11), IEEE Computer Society, pp. 336-341, Hong Kong, China, DOI: 10.1109/CSC.2011.6138544.

[5] Xu J., Shi M., Chen C., Zhang Z., Fu J., and Liu C.H. (2016). ZQL: A Unified Middleware Bridging Both Relational and NoSQL Databases. In 14th Int. Conf. on Dependable, Autonomic and Secure Comp., 14th Int. Conf. on Pervasive Intelligence and Comp., 2nd Intl Conf. on Big Data intelligence and Comp. and Cyber ScienceandTechnologyCongress (DASC/ PiCom/ DataCom/ CyberSciTech), IEEE, pp. 730-737, Auckland, DOI: 10.1109/DASC-PICom-DataCom-CyberSciTec.2016.129.

[6] Guarino N., Oberle D., and Staab S.(2009). What is an Ontology?. In Handbook on Ontologies, 2nd ed. S. Staab, R. Studer, Eds. Verlag, Berlin Heidelberg, Germany: Springer, pp.1-17, DOI: 10.1007/978-3-540-92673-3.

[7] Max C., El Malki M., Kopliku A., Teste O., and Tournier R.(2016). Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document', In 10th Int. Conf. on Research Challenges in Information Science (RCIS), IEEE, pp. 1-11, Grenoble, DOI: 10.1109/RCIS.2016.7549351.

[8] Banerjee S., Bhaskar S., Sarkar A., and Debnath N. C. (2020). A Unified Conceptual Model for Data Warehouses. In Global Research Conference (GRaCe, 2020), Malaysia, 16th-18th October.

[9] Boussahoua M., Boussaid O., and Bentayeb F. (2017). Logical Schema for Data Warehouse on Column-Oriented NoSQL Databases. In Proc. Database and Expert Systems Applications (DEXA 2017), Lecture Notes in Computer Science, Springer, Cham, pp.247-256, Bratislava, Slovakia, DOI: 10.1007/978-3-319-64471-4_20.

[10] Cuzzocrea A., De Maio C., and Fenza G. (2016). Towards OLAP Analysis of Multidimensional Tweet Streams. In ACM Eighteenth International Workshop on Data Warehousing and OLAP, pp. 992–999, ACM, NY, USA, DOI: 10.1145/2811222.2811233.

[11] Scabora L. C., Brito J. J., Ciferri R. R., Ciferri C. D. A. (2016). Physical Data Warehouse Design on NoSQL Databases. In 18th Int. Conf. on Enterprise Information Systems, Lecture Notes in Business Information Processing, Springer, pp. 111-118, Poland, DOI: 10.5220/0005815901110118.

[12] Neumayr B., Anderlik S., and Schrefl M. (2012). Towards ontology-based OLAP: Datalog-based reasoning over multidimensional ontologies. In fifteenth international workshop on Data warehousing and OLAP (DOLAP), pp. 41-48, ACM, NY, USA, DOI: 10.1145/2390045.2390053.

[13] Davardoost F., Babazadeh Sangar A. and Majidzadeh K. (2020). Extracting OLAP Cubes From Document-Oriented NoSQL Database Based on Parallel Similarity Algorithms. Canadian Journal of Electrical and Computer Engineering. vol. 43, no. 2, pp. 111-118, Spring 2020, DOI: 10.1109/CJECE.2019.2953049.

[14] El Sarraj L., and Espinasse B. (2014). An Ontology-Driven Personalization Approach for Data Warehouse Exploitation. International Journal on Advances in Software. vol.7, no.1, pp. 253–265.