A Formal OLAP Algebra for NoSQL based Data Warehouses

Shreya Banerjee *
Eastern International University, Binh Duong Province, Vietnam,
shreya.banerjee@eiu.edu.vn

Sourabh Bhaskar
Parul Institute of Engineering & Technology, Parul University, Vadodara, India,
sourabhb440@gmail.com

Anirban Sarkar
National Institute of Technology Durgapur, Durgapur, India,
sarkar.anirban@gmail.com

Narayan C. Debnath
Eastern International University, Binh Duong Province, Vietnam,
narayan.debnath@eiu.edu.vn
Overview

01 Motivation and Contribution
02 Ontology Driven Conceptual Modelling of NoSQL based Data Warehouses
03 Proposed OLAP Algebra for NoSQL based DWs
04 Illustration of Proposed OLAP Algebra Using a Case Study
05 Implementation of Proposed Algebra
INTRODUCTION

- Modern Data Warehouses (DW) solutions demand to act
  - In internet-style than to enforce the user to act within predefined structures

- Consequently, nowadays DWs need to handle
  - diverse data sources
  - heterogeneous data types: structured, semi-structured, and unstructured

- On Line Analytical Processing (OLAP) operations require to
  - deal with related business queries based on those irregular information
  - business analysts focuses on using of NoSQL databases
Motivation

- NoSQL databases are categorized based on various data models at physical level
  - Key-Value stores, Document Store, Column-Family store and Graph databases
- Each NoSQL database has implemented its own query language
- Lack of a common specification of OLAP operations over different NoSQL databases make serious problems
  - DWs using these databases are required to be portable
- How to provide a uniform standard towards OLAP operations for distinct types of NoSQL based DWs?
Contribution

- The proposed work is aiming to address the research question
- Ontology is applied to resolve the challenge
  - It is defined as an explicit specification of shared conceptualization
  - 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
  - Syntactic differences can be omitted using an ontology based specification that can provide common conceptualization
  - Semantic differences among OLAP operations can also be omitted with the help of ontology
Contribution

Figure 1. Overall process of Proposed OLAP Query Algebra and related implementation in NoSQL based DWs
Brief Discussion of Ontology Driven Conceptual Modelling of NoSQL based Data Warehouses [8]

Figure 2. Conceptual model for NoSQL based data warehouses
Brief Discussion of Ontology Driven Conceptual Modelling of NoSQL based Data Warehouses [8]

- The conceptual model is realized as a layered organization
  - Collection (Top-Most layer)
  - Family (Intermediate Layer)
  - Attribute (Bottom-Most Layer)

- $AT$ is the group of all possible instances of a data item
  - 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 ($DF$)

- The entire DW can be viewed as set of Collections from the top level
Brief Discussion of Ontology Driven Conceptual Modelling of NoSQL based Data Warehouses [8]

- Different constructs in the conceptual model are connected with each other using different relationships
  - Intra-layer kind and Inter-layer kind

- Containment and Inverse Containment relationships are
  - included towards both Inter-layer kind and intra-layer kind relationships category

- Association relationship can only be Inter-layer kind relationship

- Different relationships of this conceptual has distinct properties
  - Cardinality, Modality, and Ordering

Contd.
Proposed OLAP Algebra for NoSQL based DWs

- Proposed OLAP Algebra is classified in two groups
  - OLAP Operators
  - OLAP Operations
- Select and Aggregate operators are OLAP Operators
- Five proposed OLAP operations are: Slice, Dice, Roll-up, Drill-down, and Pivot operation
Proposed OLAP Operators

- **Select Operator** ($\pi$): This operator will extract the dimension and its hierarchy from dimension family depending on predicate $p$
  - Either $P$ can be atomic predicate, denoted as $p$
  - Or it can be a composite predicate denoted as $p_1<op>p_2<op>....<op>p_n$ where $<op>$ is a logical operator like AND, OR etc.
  - The $p$ can be either *Dimensional family* (DF) or *Dimension Hierarchy* (DFH)

The algebraic notation of the operator is

$$\pi_p (DF) = DF_o$$

- Null predicate operator will return the original $DF$

$$\pi_\emptyset (DF) = DF$$
Proposed OLAP Operators

- **Aggregate Operator** $(\alpha)$: The aggregate operator will perform the grouping function $GF$ on measure attribute $(M_{AT})$ of the set of $DF$s in a cube $C$
- The $GF$ is the relational aggregation function operating on the $M_{AT}$ only
- These $GF$s can be $SUM$, $MIN$, $MAX$, $AVG$, and $COUNT$
- The algebraic notation of the aggregate operator is

$$\alpha_{GF(M_{AT})}\{DF1 \, V \, DF2 \, V \, DF3 \, ... \, V \, DFn\}(C)$$
Proposed OLAP Operations

- **Slice operation** (\(sl\)): The slice operation selects one particular dimension from a given 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,

  \[
  CON = \pi_p(DF)
  \]

- **Dice operation** (\(di\)): The dice operation selects the two or more dimensions from a given 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,

  \[
  CON = \pi_{p1}(DF1) < op > \pi_{p2}(DF2) ... < op > \pi_{pn}(DFn)
  \]
Proposed OLAP Operations

- **Roll-up operation (Rup):** The Roll-up operation performs aggregation on a data cube by stepping down in the dimensional hierarchy.

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

- **Drill-down operation (Ddn):** Drill-down is performed by stepping up in the dimension hierarchy.

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

- **Pivot operation (pvt):** The pivot operation is also known as rotation. It rotates (transpose $T$) the data axes in view in order to provide an alternative presentation of data.

\[ Pvt(C) = \alpha_{GF(M_{AT})\{DF_{1,DF_{2}}\}}^T(C) = \alpha_{GF(M_{AT})\{DF_{2,DF_{1}}\}}(C) \]
Illustration of Proposed OLAP Algebra Using a Case Study

➢ 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

➢ This case study has two facts – Sales and Shipping
  ❑ Sales is associated with four dimensions - Location, Branch, Product, Time
  ❑ Shipping is associated with four dimensions - Location, Shipper, Product, Time
  ❑ Two facts share three dimensions - Location, Product, and Time
  ❑ Several dimensions have hierarchy : Time→Day→Month→Year
Illustration of Proposed OLAP Algebra Using a Case Study

- several dimensions have specific attributes
  - *Time Id*, and *Time*
- In addition, each fact are associated with two measures
  - Shipping is associated with measures Units Shipped and Dollars Cost
- In some cases, attributes of specific dimension is either changed or absent
  - Distinct features of this described data set is highly irregular and require flexible representation.
  - NoSQL databases are required to demonstrate these data set in DWs

Contd.
Illustration of Proposed OLAP Algebra Using a Case Study

Collections (Cubes created from Fact Families)

**FACT FAMILY 1 (SALES)**

SALES(Location, Branch, Product, Time, units sold, dollars sold)

**FACT FAMILY 2 (SHIPPING)**

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, location_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 {}
Illustration of Proposed OLAP Algebra Using a Case Study

- Example of several queries

  - Query 1: Find the derived dimension of Time for the month “November”.
    - Select operator ($\pi$) 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 contain the instance of $time_id$, $time$, $day_id$, $day$, $month_id$, $month$, $year_id$, and $year$ related to $month$=“November”
Illustration of Proposed OLAP Algebra Using a Case Study

Example of several queries

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,

\[ sl(C) = \alpha_{\text{SUM}(\text{unit\_sold})}\{\text{Product.productType.productType\_Name}\}, CON (C) \]

\[ CON = (\pi_{\text{Product.productType.productType\_Name}}(\text{Product})) \]

- Here, slice is performed for the dimension “Product” using the criterion

\[ \text{Product.ProductType.productType\_Name} = \text{“Electronics”} \]
Illustration of Proposed OLAP Algebra Using a Case Study

Example of several queries

Query 3: 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:  \[ R_{up}(DF_{41})(C) = \alpha_{SUM(\text{units\_sold})\{DF_{4(1+1)}\}}(C) \]

\[ = \alpha_{SUM(\text{units\_sold})\{DF_{42}\}}(C) \]

Second step:  \[ R_{up}(DF_{42})(C) = \alpha_{SUM(\text{units\_sold})\{DF_{4(2+1)}\}}(C) \]

\[ = \alpha_{SUM(\text{units\_sold})\{DF_{43}\}}(C) \]

\[ = \alpha_{SUM(\text{units\_sold})\{\text{Time\_Day\_Month\_Year}\}}(C) \]

Contd.
The proposed OLAP algebra is implemented in MongoDB – a Document Oriented Database

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

(a)

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

(b)

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

```javascript
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"----
Shipper"":"collection6_doc.Shipper","Branch"----}
]}

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

```javascript
db.createView('shippingView','shipping',[{"$lookup":{"from": "shipper","localField": "shipper_Id"--"foreignField": "Shipper.shipper_Id","as": "collection1_doc"},{"$unwind": "$collection1_doc"},
{"$lookup":{"from": "location","localField": "--"localField"----","Time": "$collection4_doc.Time","="-----}}]

Figure 6. MongoDB based illustration of a view consisting of consisting of only shipping fact
```
Implementation of Proposed Algebra

Query 1: The total number of Electronics product type units sold across all of the dimensions (Time, Location, Branch, and Shipper). [Slice Operation]

```javascript
db.salesView.aggregate([{
  "$match": {
    "Product.ProductType.productType_Name": "electronics"
  }
},
{
  "$group": {
    "_id": "$Product.ProductType.productType_Name",
    "total_cost": {
      "$sum": "$units_sold"
    }
  }
}])
```
Query 2: The total unit sold for a product type electronics, city Durgapur and month “November”. [Dice Operation]

```javascript
const query = 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"
    }
  }
}]);
```
Implementation of Proposed Algebra

Query 3: Find the total unit sold across all product by increasing the aggregation levels of time: from Day to Year (Day→Month→Year). [Roll-up Operation]

Query $C_0$
$db.salesView.aggregate([\{"$group":{_id:{Day:"$Time.Day.day"}, total_cost:{$sum:"$units_sold"}}\}])$

Intermediate Result 1: $IC_1$
$db.salesView.aggregate([\{"$group":{_id:{Month:"$Time.Day.Month.month"},total_cost:{$sum:"$units_sold"}}\}])$

Intermediate Result 2: $IC_2$
$db.salesView.aggregate([\{"$group":{_id:{Year:"$Time.Day.Month.Year.year"},total_cost:{$sum:"$units_sold"}}\}])$

Contd.
Conclusion and Future Work

➢ Novelties of the proposed work
   □ To provide common formal syntax and semantics towards different OLAP operators and operations
   □ Proposed formal specifications are
     ▶ independent of any physical level implementation
     ▶ able to be applied in distinct type of NoSQL based DWs
     ▶ implemented in a document-oriented database MongoDB

➢ Future Work will include
   □ automation of query answering through a rule based reasoner
   □ automated conversion of these formal operators towards native NoSQL based DWs
THANK YOU