# OLAP ALGEBRAIC LAWS FOR IMPROVING OLAP EXPRESSION TREES

Dr. Bhavesh Patel[1], Dr. Jignesh Patel[2], Dr. Badal Kothari[3]

[1] *Assistant Professor,|Department of Computer Science,H.N.G.University, Gujarat, India*
[2] *Assistant Professor,|Department of Computer Science,H.N.G.University, Gujarat, India*
[3] *Technical Assistant,|Department of Computer Science,H.N.G.University, Gujarat, India*

## ABSTRACT

*OLAP algebra, In other words, we describe simple semantics representing a comprehensive Multi-dimensional OLAP algebra that can directly exploit the clean Object-Oriented conceptual model. In this paper, we describe a number of laws for our comprehensive OLAP algebra. To illustrate the motivation for this process, first recall that a query in traditional relational databases, written in SQL, is translated internally into an initial relational algebra expression that can be then transformed into equivalent, but more efficient ones by applying various relational algebraic rules. In order to perform better joins between the cube and dimension tables, we change the restriction of the selection operation so that it can be performed on the relevant cuboid/view alone. In summary, our comprehensive OLAP query algebra (operations and laws), grammar and metadata storage are essential components in the process of resolving OLAP queries written in native OOP languages.*

**Keyword:** *Algebraic Laws, MOLAP Algebra, Operation in MOLAP*

## Introduction

Object-Oriented OLAP queries are written at a very high level against the conceptual model, our OLAP query processor must do a lot of additional processing to supply missing details. Thus, an OLAP query is translated internally into an OLAP algebra expression that ultimately makes alternative forms of an OLAP query easier to create, explore, manipulate and optimize (e.g., push and pull operations, replace operations). Specifically, when an OLAP query is submitted to our OLAP DBMS, its query optimizer tries to find the most efficient equivalent OLAP algebra expression before evaluating it.

For example, the most common relational algebraic laws are (1) pushing the selection (_) as far as possible, (2) combining selection (_) with Cartesian product (X) to produce joins (∞), (3) introducing new projections (_) when necessary, etc. In Figure 1(a), we can see how the SQL is transformed into an initial tree of relational algebra operations. Figure 1(b) improves the initial expression by applying common relational algebraic rules in some meaningful way. Specifically, we split the two parts of the selection (starname =name) and (birthdate LIKE '%1960'). The first condition involves attributes from both sides of the product, but they are equated, so the product and selection can be combined to produce an equijoin. The latter condition is pushed down the tree.

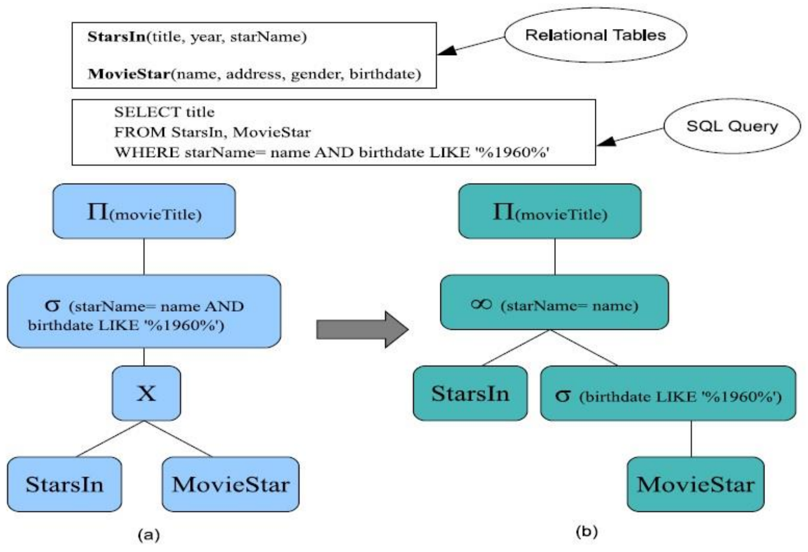

StarsIn(title, year, starName)
MovieStar(name, address, gender, birthdate)
Relational Tables
SELECT title
FROM StarsIn, MovieStar
WHERE starName= name AND birthdate LIKE '%1960%'
SQL Query
Π(movieTitle)
σ (starName= name AND birthdate LIKE '%1960%')
X
StarsIn
MovieStar
(a)
Π(movieTitle)
∞ (starName= name)
StarsIn
σ (birthdate LIKE '%1960%')
MovieStar
(b)

Figure 1: (a) Translation of SQL to an initial relational algebra expression.
(b) The effect of applying some relational algebra laws

Figure 2 illustrates an initial tree consists of operators from our OLAP algebra (i.e., SELECTION and PROJECTION). Therefore, common OLAP analysis, such as the application of OLAP query constraints or descriptive OLAP reports, could not be performed from the fact table alone since it is the dimension tables that store descriptive attributes. In other words, we generally require joins between the cube and the dimension tables because (i) the query constraints are often specified on the attributes of the dimensions and (ii) descriptive attributes make OLAP reports easier to read. Moreover, we note that whenever descriptive dimension attributes are utilized by OLAP algebra operators, *inner joins* are required between the fact table and dimension tables.

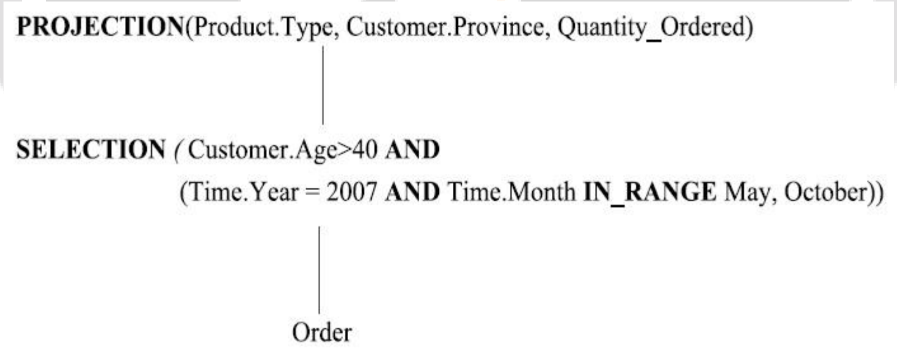

PROJECTION(Product.Type, Customer.Province, Quantity_Ordered)
SELECTION ( Customer.Age>40 AND
(Time.Year = 2007 AND Time.Month IN_RANGE May, October))
Order

Figure 2: Initial OLAP algebra tree

**Laws involving SELECTION**

A selection result depends on inner/natural joins between the cube and dimension tables in order to exclude cube rows that don't satisfy the query restriction specified on the descriptive attributes of the dimension. In order to perform better joins between the cube and dimension tables, we change the restriction of the selection operation so that it can be performed on the relevant cuboid/view alone. Let the 2-dimensional cube C = <D, F, M, BasicCube>, where D={Dim1, Dim2}, F={Dim1.Dim1ID, Dim2.Dim2ID}. Note that Dim1ID and Dim2ID are the most detailed encoded values of dimensions Dim1 and Dim2.

**LAW 1:**

SELECTION *(Dim1(C1)* AND/OR *Dim2(C2)) (C)* = SELECTION *(Dim1ID = x* AND/OR *Dim2ID = y)(C)*

Where *x* and *y* are two sets of *Dim1ID*s and *Dim2ID*s that satisfy the conditions *C1* and *C2* associated with dimensions *Dim1* and *Dim2*.

Justification: Suppose a cell *c* is in the result of the left expression. Then there must be a record *r* that satisfies the restriction on dimension *Dim1* and a record *s* that satisfies the restriction on dimension *Dim2*. Moreover, *r* and *s* must agree with *c* on every attribute that each record shares with cell *c* (*Dim1ID* and *Dim2ID*).

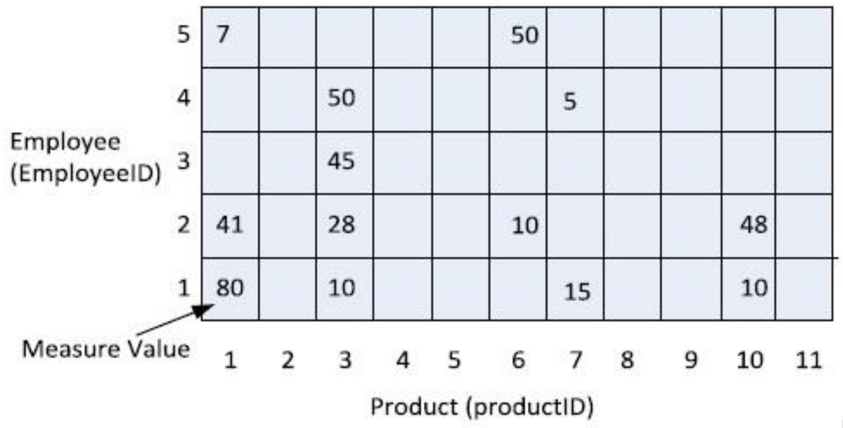


Figure 3: Two-dimensional cube with the most detailed level values

When we evaluate the expression on the right, x is a set of Dim1IDs satisfying the restriction associated with Dim1, while y is a set of Dim2IDs satisfying the restriction on dimension Dim2. Dim1ID of record r must be in set x and Dim2ID of records must be in set y. Thus a cell c1 is in the result of the right expression. Consequently, Dim1ID and Dim2ID of cell c1 must agree with one value from set x (Dim1ID of record r) and one value from set y (Dim2ID of record s). Therefore, we can say that c and c1 is the same cell.

We use the same logic if the logical operator between dimension conditions is an OR operator. Figure 3 provides an illustration of a very simple two dimensional view (called Sales) with the most detailed value stored for each dimension in the cube.

**Combining conditions**

When we have two or more consecutive SELECTION operators, we can replace them by only one SELECTION operator and connect their conditions with the AND operator(s). Thus, our second law for SELECTION is the combining law:

**LAW 2:**

SELECTION *(Cond1) (*SELECTION*(Cond2) C)* =SELECTION *(Cond1 AND Cond2)(C)*

Justification: Suppose that a cell c is in the result of the left expression. Then the result of SELECTION(Cond2)C is a sub-cube C1 that contains cell c that satisfies cond2. We apply SELECTION(Cond1) to C1. The result is a sub-cube of the left expression that contains c that satisfies also Cond2. When we evaluate the right condition, cell c will again be in the result since c satisfies Cond1 and Cond2. Since our OLAP server provides a very efficient multi-dimensional indexing scheme, this rule allows the SELECTION operation to benefit from this multi-dimensional indexing. Instead of accessing the appropriate R-tree index view to answer the first condition and then using the result cube to answer the second condition, the multi-dimensional index view can be efficiently used to answer both conditions simultaneously.

In addition to the above law, two SELECTION operators can be combined into only one SELECTION if there is a UNION operator between them. Moreover, the conditions of both SELECTIONs are connected with the OR operator. This law is written as follow:

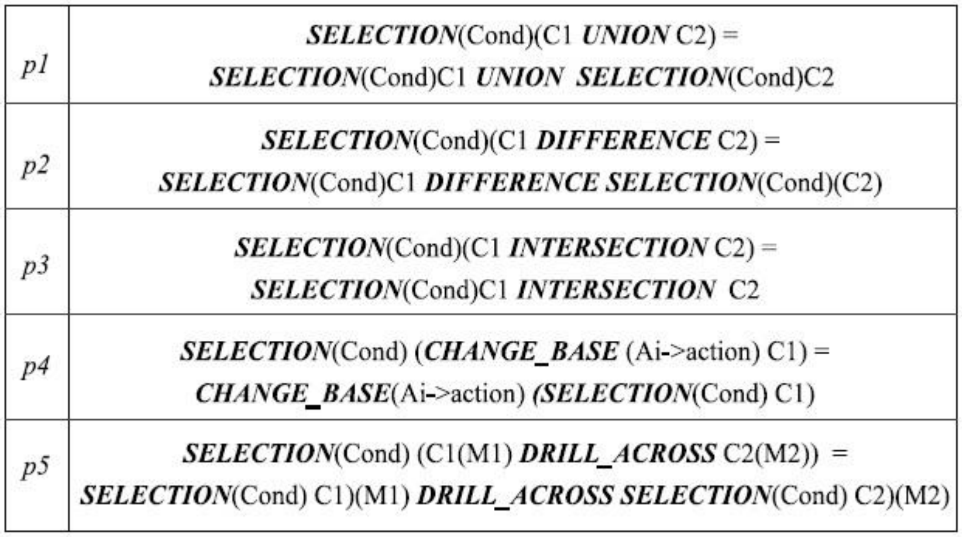
**LAW 3:**

*(*SELECTION*(Cond1)C)* UNION *(*SELECTION*(Cond2)C)* = SELECTION *(Cond1 OR Cond2)(C)*

Finally, LAW 2 and LAW 3 are very useful in any OLAP server that provides multidimensional indexing schemes (e.g., R-tree).

**Pushing laws**

Selection is a very important operation from the point of view of OLAP query optimization. In particular, Selection tends to reduce the size of the cubes. One of the most important objectives is to move the selection down the tree as far as it will go without changing what the OLAP expression tree actually does. In addition, pushing SELECTION down the tree makes it possible to be efficiently resolve the query from the apropriate multi-dimensional index view. The next family of laws allows us to push the SELECTION through other OLAP operators. Thus, we refer to this set of laws as the *pushing laws*. Figure 4 illustrates how the SELECTION can be pushed below other OLAP operators.



| | |
|---|---|
| *p1* | ***SELECTION***(Cond)(C1 ***UNION*** C2) = ***SELECTION***(Cond)C1 ***UNION*** ***SELECTION***(Cond)C2 |
| *p2* | ***SELECTION***(Cond)(C1 ***DIFFERENCE*** C2) = ***SELECTION***(Cond)C1 ***DIFFERENCE SELECTION***(Cond)(C2) |
| *p3* | ***SELECTION***(Cond)(C1 ***INTERSECTION*** C2) = ***SELECTION***(Cond)C1 ***INTERSECTION*** C2 |
| *p4* | ***SELECTION***(Cond) (***CHANGE_BASE*** (Ai->action) C1) = ***CHANGE_BASE***(Ai->action) ***(SELECTION***(Cond) C1) |
| *p5* | ***SELECTION***(Cond) (C1(M1) ***DRILL_ACROSS*** C2(M2)) = ***SELECTION***(Cond) C1)(M1) ***DRILL_ACROSS SELECTION***(Cond) C2)(M2) |

Figure 4: SELECTION *pushing laws*

**LAW 4:**

1. For a UNION, SELECTION must be pushed to both arguments of the UNION. p1 in Figure 4 illustrates this rule.
2. For a DIFFERENCE, SELECTION must be pushed to the first argument of the operator or to both arguments. For example, p2 in Figure 4 shows how we might push SELECTION to both arguments.
3. For an INTERECTION, SELECTION can be pushed to one of the arguments or both. p3 is an example of how we might push SELECTION to the first argument.
4. For CHANGE LEVEL and CHANGE BASE, SELECTION is pushed down to the argument. p4 provides an example of pushing SELECTION under CHANGE BASE.
5. For a DRILL ACROSS, SELECTION must be pushed to both arguments. p5 in Figure 4 shows this rule.

*Justification:* Suppose that a cell c is in the result of SELECTION(Cond) (C1 UNION C2). Then the result of (C1 UNION C2) has a cell c that satisfies the condition parameter of the SELECTION operator. In addition, c can be a cell found only in C1, C2, or the result of two cells from both cubes C1 and C2. When we evaluate the right expression, SELECTION(Cond)C1 UNION SELECTION(Cond) C2, c will again be in the result of the right expression, because c matches the condition and can be found from C1, C2, or the result of the union.

**Pulling laws**

Pushing a selection down an OLAP expression tree is one of the most important steps performed by the query optimizer. However, we have found that in some situations it is essential to pull the SELECTION up the expression tree as far as it will go, and then push it down all possible branches. Consider two views/cuboids (C1 and C2) of

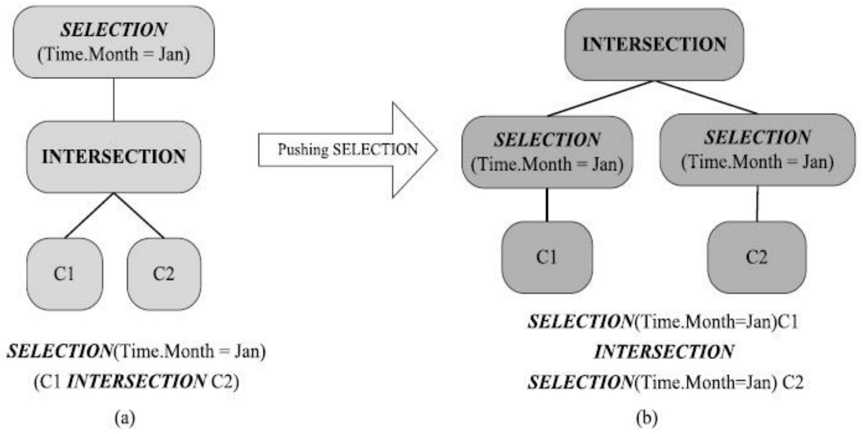

SELECTION
(Time.Month = Jan)
INTERSECTION
C1
C2
Pushing SELECTION
SELECTION(Time.Month = Jan)
(C1 INTERSECTION C2)
(a)
INTERSECTION
SELECTION (Time.Month = Jan)
SELECTION (Time.Month = Jan)
SELECTION(Time.Month=Jan)C1
INTERSECTION
SELECTION(Time.Month=Jan) C2
(b)

Figure 5: (a) Initial OLAP expression tree and (b) its equivalent after applying the SELECTION pushing laws.

(PROJECTION(Product.Number, Time.Month, Units Sold)
(SELECTION(Time.Month = Dec)C2))INTERSECTION
PROJECTION(Product.Number, Time.Month, Units Sold)C1)

The OLAP expression tree of the above OLAP algebra expression is shown in Figure 6(a). In this OLAP algebra tree, there is no way to push the SELECTION down the tree because it is already as far as it would go. We can pull the SELECTION above the INTERSECTION and then push down if, and only if, the output of the INTERSECTION contains all attributes that are mentioned within the SELECTION.

This mechanism of pulling up and then pushing down the SELECTION operator is advantageous because the size of the view C1 is reduced in the intersection. Moreover, if C1 is stored in our server, then its R-tree index can be efficiently used to find those rows satisfying the condition (Time.Month=Dec). However, without this condition all cells in C1 must be accessed and read into the main memory.

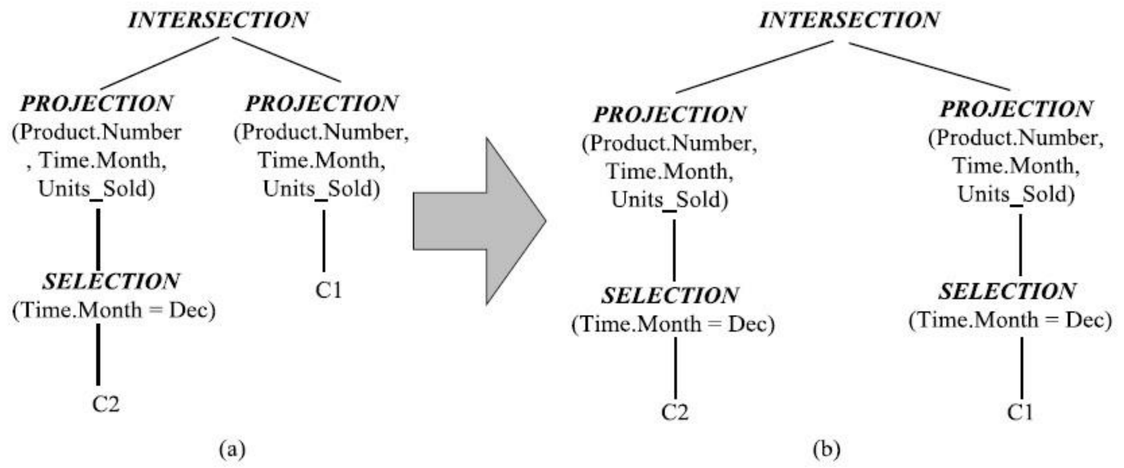

INTERSECTION
PROJECTION (Product.Number , Time.Month, Units_Sold)
PROJECTION (Product.Number, Time.Month, Units_Sold)
SELECTION (Time.Month = Dec)
C2
C1
(a)
INTERSECTION
PROJECTION (Product.Number, Time.Month, Units_Sold)
PROJECTION (Product.Number, Time.Month, Units_Sold)
SELECTION (Time.Month = Dec)
SELECTION (Time.Month = Dec)
C2
C1
(b)

Figure 6: (a) Initial OLAP expression tree. (b) Improving the initial expression by pulling SELECTION up and then pushing it down the tree.

## Conclusion

We have described a comprehensive multi-dimensional OLAP algebra. Our algebra represents all common OLAP operations reduces the complexity of using existing relational algebras to write OLAP queries (via SQL or MDX) and also subsequently allows for the optimization of OLAP queries written in native OOP languages such as Java. Moreover, we are providing optimization laws and execution algorithms that show how and why an OLAP algebra is a good idea in practice. In association with the algebra, we have developed a robust DTD-encoded OLAP query grammar that provides a concrete foundation for client language queries. The grammar, in turn, is the basis of a native language query interface that eliminates the reliance on an intermediate, string-based embedded language. Finally, the storage of the schema is done natively in XML.

In summary, our comprehensive OLAP query algebra (operations and laws), grammar and metadata storage are essential components in the process of resolving OLAP queries written in native OOP languages. Further, we will work on, how these components, as well as the storage engine, are integrated with the query compiler and execution engine to form a pure OLAP DBMS.

## References

[1] Microsoft analysis services. http://www.microsoft.com/sqlserver/2008/en/us/ analysisservices.aspx.
[2] Oracle essbase. http://www.oracle.com/us/solutions/ent-performancebi/business-intelligence/essbase/index.html.
[3] Sap. http://www.sap.com/services/education/catalog/netweaver/bi.epx.
[4] Xml for analysis specification v1.1, 2002. http://www.xmla.org/index.htm.
[5] F.N. Afrati, C. Li, and J.D. Ullman. Generating efficient plans for queries using views. *SIGMOD*, pages 319–330, 2001.
[6] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan, and S. Sarawagi. On the computation of multidimensional aggregates. *Proceedings of the 22nd International VLDB Conference*, pages 506–521, 1996.
[7] C. Cunningham, G. Graefe, and C. A. Galindo-Legaria. Pivot and unpivot: Optimization and execution strategies in an rdbms. *In International conference on Very Large Data Bases (VLDB)*, pages 998–1009, 2004.
[8] E. Franconi and A. Kamble. The gmd data model and algebra for multidimensional information. *In: Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2004).*, 3084:446–462, 2004.
[9] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. *In International conference on Data Engineering (ICDE),Washington, DC, USA*, pages 152–159, 1996. IEEE Computer Society.
[10] O. Romero and A. Abello. On the need of a reference algebra for olap. *In International conference on Data warehousing and Knowledge Discovery (DaWak)*, pages 99–110, 2007.