Efficient Cuboid Materialization in Data Cube

Prashant R
Department of Information Technology
Maharaja Surajmal Institute of Technology, GGSIPU, New delhi-110059, India
prashantraghu999@gmail.com

Suman Mann
Department of Information Technology
Maharaja Surajmal Institute of Technology, GGSIPU, New delhi-110059, India
sumanmann2007@gmail.com

Eashwaran R
Department of Computer Science
Maharaja Surajmal Institute of Technology, GGSIPU, New delhi-110059, India
eashwaranraghu@gmail.com

***Abstract*—In the field of business intelligence, we require the analysis of multidimensional data with the need of it being fast and interactive. Data warehousing and OLAP approaches have been developed for this purpose in which the data is viewed in the form of a multidimensional Data cube which allows interactive analysis of the data in various levels of abstraction presented in a graphical manner. In data cube there may arise a need to materialize a particular cuboid given that only the core cuboid has been materialized, in this paper we propose an algorithm to do this in an optimal way such that the intermediate cuboids require less space and time to generate.**

***Keywords*—-cuboid materialization, lattice, data warehouse, OLAP**

I. Introduction

The basic principle of Data Warehousing is to provide statistics/data to the business users for well calculated and planned strategic decisions. And the data stored is in the form of a multidimensional model. A data warehouse is basically of 3 main types:

1. Operational Data Store

2. Enterprise Data Warehouse

3. Data Mart

A Data Warehouse works as a primary repository where data from various origins materialize. Further Data might be categorized into Structured, Semi-structured and Unstructured data forms.

In the past, the various organizations used simple and non-complex data warehousing, but over time they modified it and made it more advanced and intricate.

The various general stages of the use of data warehouse are as follows:

1. Offline Operational database: In this stage data is copied from OS to a different server so this way the operational server's performance is not affected as the processing would be done in the server which has the copied data.

2. Offline Data Warehouse: Data in the Data Warehouse is automatically updated and mapped to reach the objectives.

3. Real-time Data Warehouse: In this one, Data Warehouses are updated when there is a change in the database, like a transaction.

These systems provide OLAP tools for interactive analysis of multidimensional data.

OLAP is the acronym for Online Analytical Processing. In OLAP, information/data are classified by dimensions. OLAP(cubes) are usually pre-compiled to improve query time in relational databases.

One of the most coherent data structures is Data Cube, which is used to represent multidimensional aggregates in Data Warehouse systems.

A data cube refers to a 3D range of values that are used to view data from the multidimensional model in which the information is computed and stored.

The data cube that is used is defined by two entities, Facts, and Dimensions. Facts are used to analyze dimensions with their numerical values. Dimensions are the rules with respect to which an organization stores their records.

Data cubes are categorized into two categories:-

1. Relational OLAP

2. Multidimensional Data Cube

In data warehouse systems, query response time largely depends on the efficient computation of data cube. However, creating data cube is normally time and memory consuming. So, we normally materialize cuboids of a Data Cube. To do this, there are three possibilities:

1. Materialize the whole Data Cube: This is the best solution in terms of the query response time. However, computing every cuboid and storing them will take maximum space if the data cube is very large, which will affect indexing and the query response time.

2. No Materialization: In this one, Cuboids are computed as and when required. So, the query response time fully depends on the database which stores the raw data.

3. Partial materialization: This is the most feasible solution. In this approach, some cuboids or cells of a cuboid are pre-computed. However, the problem is how to select these cuboids and cells to be pre-computed. Generally, cuboids and cells which can help in computing other cells or cuboids, are pre-computed.

All the view materialization algorithms are required to use some data structures to represent the data cube. One useful data structure is data cube lattice or Lattice of Cuboids, which is being used by our algorithm Lowest first described in this paper.

Rest of the paper is organized as follows: Section 2, discusses previous work done in related areas of study. Section 3, describes our approach to pathfinding. Section 4, provides a proof by induction to our theorem based on which the algorithm is proposed. Section 5, presents the algorithm lowest first for cuboid materialization. Section 6, implements the algorithm to an example data cube and then verifies the resultant path. Section 7, presents the conclusion.

II. Related Work

There has been some synchronous work on the problem of computing and selection of data cube [2][16][14][5][15][1]. Shukla et al. [2] proposed a greedy algorithm which was a modified version of what Harinarayan et al. presented, which proposed a greedy algorithm that works on the lattice and pick the right views to materialize, subject to various constraints for the selection of data cube. This had an efficiency problem which was solved by Shukla et al. [17]. whose work proposed PBS (Pick by Size) that selects data cube according to the cube size. H. Gupta [13] proposed a polynomial time greedy heuristic framework that uses AND view (each view has a unique evaluation), OR view (any view can be computed from any of its related views), and AND-OR view graph. Shukla et al. [18] proposed three different algorithms, considering the view selection problem for multi-cube data models, SimpleLocal, SimpleGlobal, and Complex Global which picks aggregates for precomputation from multi-cube schemas.

A heuristic algorithm was proposed by C. Zhang et al. [15], for determining a set of materialized views based on the idea of reusing temporary results from the execution of global queries with the help of Multiple View Processing Plan (MVPP). This algorithm had a major flaw, it didn't consider the system storage constraints. For this special case of OR view graphs, Himanshu Gupta et al. [12] designed an approximation greedy algorithm. To demonstrate that materialized views is practical and effective as compared to heuristic approaches J. Yang et al. [15] used Genetic Algorithm. W. Yang [7] proposed a greedy repaired genetic algorithm to select the set of materialized cubes and found that solution can greatly reduce the amount of query cost as well as the cube maintenance cost. I. Mami et al. [11] presented a constraint programming-based approach to address the view selection problem. A framework for materialized view selection was proposed by K. Aouiche et al. [8] that exploited a data mining technique (clustering), in order to determine clusters of similar queries. He also proposed a view merging algorithm that builds a greedy process and candidate for selecting a set of views to materialize. I. Mami [10] game surveys of the technique for selection of cube.

M.P. Deshpande [17][6] proposed a sorting-based algorithm, for computing the cube that overlaps the computation of different group-by operations using the least possible memory for each computation of cube in the lattice of the cuboid. A Lvanova and R. Boris [9] designed data cube based on the Object-oriented conceptual model. S. Sen [4] [3] proposed an algorithm for finding the optimal path in the lattice of the cuboid, which is based on two operations roll-up and drill-down for finding the optimized path to traverse between data cubes. Now the researchers are focusing on designing efficient algorithms for the complete cube.

III. Lowest First Approach

In Lattice of cuboids, there may arise a situation where we require to materialize a particular cuboid given that only core cuboid is presently materialized. We propose an algorithm named Lowest First, that does this by searching and materializing those intermediate cuboids that have the lowest number of rows present and hence low space and time complexities.

For example let us consider a Lattice of cuboids having a total of 4 dimensions {A,B,C,D} having values A=10, B=20, C=5, D=15. If we had to materialize cuboid <A> having cuboid <A,B,C,D> already materialized we have multiple ways to do this.

i) One is by starting from <A,B,C,D> to <A,B,C> and then to <A,B> and finally we reach <A>

ii) Another path may be to start from <A,B,C,D> to <A, C,D> to <A,D> and finally to <A>.

In total we have $\binom{3}{2}+\binom{3}{1}$ number of ways of doing so, which we can try all to compare and decide the most efficient path but our proposed algorithm finds the most efficient path by sorting and hashing the product of cardinalities of dimensions in each cuboid without the need to actually materialize intermediate cuboids and find the most efficient path.

First, we represent the cost of a path as a mathematical expression using which as a parameter we compare the efficiency of various paths and later optimize it. The expression involves the cardinalities of dimensions each intermediate cuboid has where cardinality of a dimension is the number of elements in the set containing all unique values a dimension has. For a better explanation, let's get back to our example of the Data cube with 4 dimensions {A,B,C,D} and we are required to find an efficient path starting from <A,B,C,D> to <A> by observation we can say that the path will involve two intermediate cuboids one having 3 dimensions and the other having 2 dimensions materializing both will result in our target cuboid, and both of these intermediate cuboids must contain all dimensions present in our target cuboid which is {A} in our case.

$$a_1 a_2 a_3 + a_4 a_5 \qquad (1)$$

is the expression for the cost where a_1, a_2, a_3 are the cardinalities of the dimensions present in intermediate cuboid having 3 dimensions and a4, a5 are the cardinalities of dimensions present in intermediate cuboid having 2 dimensions. We also know that a_4, a_5 must be present in a_1, a_2, a_3 so for simplicity we assume $a_1 = a_4$ and $a_2 = a_5$

so that our expression becomes

$$a_1 a_2 a_3 + a_1 a_2 \qquad (2)$$

The target dimension {A} must be present in both intermediate cuboids hence we assume a_1 to be A and take is common, so our expression further reduces to

$$A(a_2 a_3 + a_2) \qquad (3)$$

where $A = a_1$

Now we need to find an optimal injection between the domain set $\{a_2, a_3\}$ containing elements present in our expression (3) to codomain containing dimensions {B,C,D} that are present in our source cuboid but absent in target cuboids we call this set difference set.

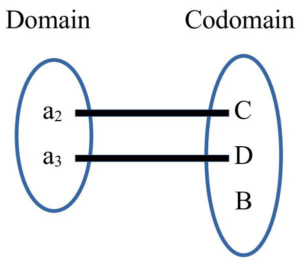
The way our algorithm finds this optimal injection is by performing the following operations.

Steps:

1)Sort the difference set based on cardinality of each dimension.

2)Map each element of domain set in order starting from a_1 to a_n to elements of difference set in sorted ascending order.

So the resultant mapping for our example data cube and particular path finding will look like.



where the sorted difference set in ascending order is {C,D,B} based on their values 5,15,20 respectively and the domain set is {a_2, a_3} so our expression finally becomes

$$C*D+C$$

which means the optimal path from <A,B,C,D> to <A> is

<A,B,C,D> => <A,C,D> => <A,C> => <A>

and the intermediate cuboids are <A,C,D> and <A,C> which out of all other possibilities require least space and time to materialize.

Now we provide a proof by induction to why the algorithm results in an optimal path.

IV. Mathematical Proof

Theorem 1. *Minimum value of expression*

$$a_1 a_2 a_3 \ldots a_n + a_1 a_2 a_3 \ldots a_{n-1} + \ldots. + a_1 a_2 + a_1 \qquad (4)$$

where $n \in W$ *is obtained when the set containing all the variables present in the expression* (4) *in order starting from* a_1 *to* a_n *is mapped to a set containing* m *elements where* $m \geq n$ *and* $m \in W$ *in sorted ascending order.*

Proof.

Let the codomain to which we map be

$$\{A_1, A_2, A_3, \ldots., A_m\} \qquad (5)$$

and assume it is present in sorted ascending order

so $A_1 \leq A_2 \leq A_3 \leq A_4 \ldots.. \leq A_m \qquad (6)$

Base Case. $n=1$

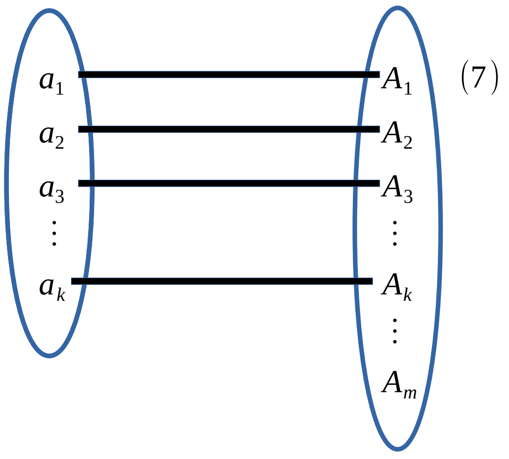
when $n=1$ the domain becomes $\{a_1\}$ and the expression is reduced to a single term containing only a_1 now according to our theorem we map the variable a_1 to the first element of our sorted codomain A_1 which is the minimum possible value of the expression a_1 as A_1 is the minimum value among all elements of set (5) Hence base case is true.

Inductive assumption. $n=k$

We suppose our theorem is true for $n=k$ which then implies for the expression

$a_1 a_2 a_3 \ldots a_k + a_1 a_2 a_3 \ldots a_{k-1} + \ldots. + a_1 a_2 + a_1$ the optimal mapping that will result in minimum possible value is

Fig. 1 Set Mapping for inductive assumption



and the optimal value of expression for $n=k$ becomes

$$A_1 A_2 A_3 \ldots A_k + A_1 A_2 A_3 \ldots A_{k-1} + \ldots. + A_1 \qquad (8)$$

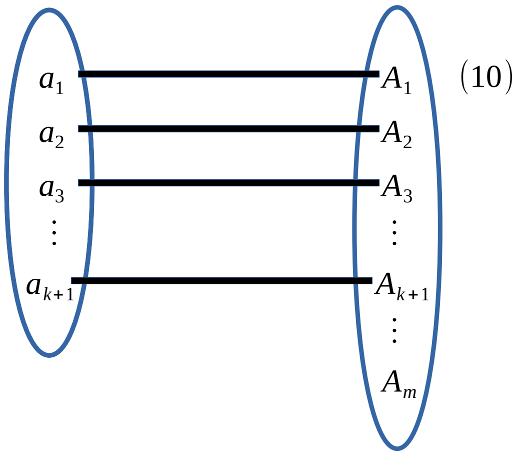
For $n=k+1$

we need to show using **Fig. 1** that the value of expression

$$a_1 a_2 a_3 \ldots a_{k+1} + a_1 a_2 a_3 \ldots a_k + \ldots. + a_1 a_2 + a_1 \qquad (9)$$

is minimum when the mapping is as shown in

Fig. 2 Optimal mapping for $n=k+1$



and to show this we split the expression (9) into 2 parts

$$\underbrace{A_1 A_2 A_3 \ldots A_{k+1}}_{Part 1} + \underbrace{A_1 A_2 A_3 \ldots A_k + \ldots. + A_1 A_2 + A_1}_{Part 2}$$

Part 2 of the expression is same as (8) which by our inductive assumption for $n=k$ is minimum.

Now we need to show whether part 1 is minimum possible which is true as to select k+1 non negative integers from a set such that their product is minimum is only possible when the lowest k+1 terms are chosen and this is what we are doing by selecting $A_1, A_2, A_3, \ldots, A_{k+1}$.

Hence for $n=k+1$ our theorem holds true and **Fig. 2** is the optimal mapping of elements which proves our theorem to be true.

V. Algorithm

Now we describe the algorithm in a step by step way which can be replicated in any programming environment.

Preprocessing:

Time complexity: $O(n\log(n))$

1) User input the values and names of all dimensions.

2) Check if they are valid.

3) Create an array named sorted_dimensions by sorting the dimensions given by the user based on the values of those dimensions in ascending order.

Query:

Time complexity: $O(n)$

1) User input for the source and target cuboids.

2) Ensure the number of dimensions in source cuboid is greater than that in the target cuboid and if not then swap source cuboid with target cuboid.

3) Create a hash map for dimensions of source cuboid, Iterate over the array of dimensions of target cuboid and for each dimension check whether the hash map of source cuboid contains this dimensions if not then push it a new array named difference_set which after fully iterating over the target dimensions will contain dimensions that aren't present in source cuboid but present in target cuboid.

4) Create Hash map named hash_diff_set of the elements of difference_set to reduce access time complexity down to $O(1)$ from $O(n)$.

5) Iterate over the sorted_attr containing all the dimensions in sorted order by values, and for each element check if it exists in hash_diff_set, if present then push that dimension with its value to a new array named sorted_diff_set so that after iterating over sorted_attr we have sorted_diff_set containing dimensions that have possibility to be present in intermediate cuboids for a given query in $O(n)$ time complexity.

6) Now the path from source cuboid to target cuboid is starting from Target cuboid and then to an intermediate cuboid containing dimensions of source cuboid and in addition the first dimension present in sorted_diff_set in its sorted ascending order, and from here to the next intermediate cuboid having dimensions of the present cuboid and in addition to this also the second dimension present in sorted_diff_set and continuing this procedure till we reach the Source cuboid.

VI. Implementation With Example

Let us consider a lattice of cuboids of a data cube having a total of 3 dimensions namely Quarter, Brand and Region where cardinality of Quarter is 4 (Q_1,Q_2,Q_3,Q_4), cardinality of Brand is 2 (B_1,B_2) and cardinality of Region is 6 (R_1,R_2,R_3,R_4,R_5,R_6) and the measure is Sales which is a number. We are required to find the optimal way of materializing cuboid <Q> starting from <Q,B,R>.
Now we use our algorithm to find what the optimal path is for the given Data cube, source cuboid, and target cuboid.

Preprocessing:

1) We create an array named sorted_dimensions by sorting the dimensions {Q,B,R} based on the values of those dimensions in ascending order and the result is sorted_dimensions = [B,Q,R] as values of B=2, Q=4 and R=6.

Query: <Q,B,R> => <Q>

1) Source Cuboid = <Q,B,R>
Target Cuboid = > <Q>

2) Ensure dimensions in target cuboid are strictly lower than that in source cuboid.
3) To create difference_set we first create a hash map of source dimensions and iterate over target cuboid checking whether this dimension exists in the hash map or not and if not present then push it to difference_set. So after doing this our difference_set = [B,R].

4) Create a hash map of difference_set.

5) Iterate over sorted_attr created during preprocessing and for each dimension in it check whether it exists in the hash map of difference_set if present then push it to a new array sorted_diff_set. After doing this we'll have an array named sorted_diff_set which contains the elements of difference_set in sorted ascending order.
sorted_diff_set = [B,R].

6) Our path now is starting from Target cuboid <Q> and to the intermediate cuboid having dimensions of target cuboid and the first dimension in sorted_diff_set ie, {B} and then the second dimension and so on.
Hence Path given by algorithm **<Q> => <Q,B> => <Q,B,R>** is the optimal way to materialize cuboid <Q> starting from <Q,B,R>

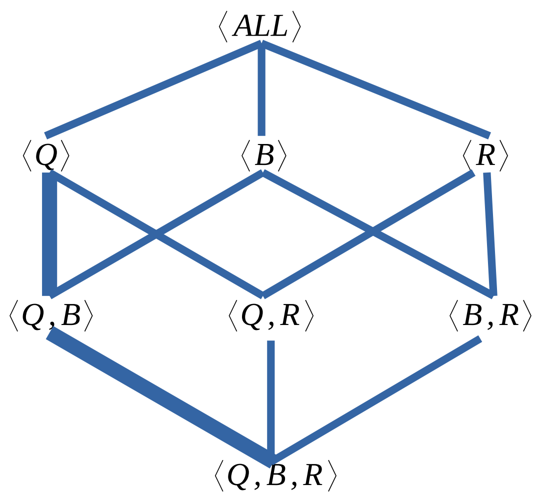
Now we'll verify this result by comparing it will all other possibilities. All the paths we could've taken to materialize cuboid <Q> are

1) <Q,B,R> => <Q,R> => <Q>, cost = sum of cost of materialization of intermediate cuboids ie, cost(<Q,R>) which is product of values of dimensions present in the cuboid = 4*6=24.

2) <Q,B,R> => <Q,B> => <Q>, cost = 4*2 =8 similar to how we calculate for previous path.

Comparing the costs we can see that the path 2) is optimal compared to path 1) and as we only have 2 paths in our example we can be sure this is the most efficient way to materialize cuboid <Q>.

Fig. 3 Lattice of cuboids for the example with three Dimensions {Q,B,R} and optimal path highlighted from <Q,B,R> to <Q>.



VII. Conclusion

In this paper, we have proposed an algorithm which has a preprocessing time complexity of $O(nlog(n))$ and query time complexity of $O(n)$ that finds the optimal path of materializing a cuboid starting from core cuboid in a lattice of cuboids or data cube and with the Preprocessing it can extend its use case to massive pathfinding queries.

References

[1] X. Li, J. Han, and H. Gonzalez, "High dimensional OLAP: A Minimal cubing approach, "in Proc. 2004 Int. Conf. Very Large Databases (VLDB'04), Toronto Canada, Aug. 2004, pp.528-539.

[2] V. Harinarayan, Rajaraman, A., Ullman, "Implementing Data Cubes Efficiently", In ACM SIGMOD International Conference on Management of Data, ACM Press, New York (1996) pp.205-216.

[3] S. Soumya, C. Nabendu, "Efficient traversal in data warehouse based on concept hierarchy using Galois Connections, In Proc. of Second Int. Con on Emerging applications of information technology,2011, pp 335- 339

[4] S. Soumya, C. Nabendu, C. Agostino, "Optimal space and time complexity analysis on the lattice of cuboids using Galois connections for the data warehousing" In proc 2009, Inter. conf. on computer science and convergence information technology, pp1271- 1275.

[5] Stefanovic, N., Han, J., Koperski, K.: Object-Based Selective Materialization for Efficient Implementation of Spatial data cubes. IEEE Transaction on Knowledge and Data Engineering, 2000, pp 938-958

[6] M.P. Deshpande, S. Agarwal, J.F. Naughton, R. Ramakrishnan "Computation of Multidimensional Aggregates" University of Wisconsin Madison, Technical Report,1997

[7] L.Y. Wen, K.I. Chung, "A genetic algorithm for OLAP data cubes" Knowledge and information systems, January 2004, volume 6, Issue1, pp 83-102.

[8] K. Aouiche, P. Jouve, and J. Darmont. Clustering-based materialized view selection in data warehouses. In ADBIS'06, volume 4152 of LNCS, pages 81–95, 2006.

[9] I. Antoaneta, R Boris, "Multidimensional models constructing data cube" Int. conference on computer systems and technologies-CompSysTech'2004, V-5pp1-7

[10] I. Mami and Z. Bellahsene," A survey of view selection method" SIGMOD Record, March 2012 (Vol. 41, No. 1), pp20-30

[11] I. Mami, R. Coletta, and Z. Bellahsene, "Modeling view selection as a constraint satisfaction problem", In DEXA, pp396-410,2011

[12] H. Gupta, I.S. Mumick, Selection of views to materialize under maintenance cost constraint. In Proc.7th International Conference on Database Theory (ICDT'99) Jerusalem, Israel, pp. 453–470, 1999

[13] H. Gupta, "selection of views to materialize in a Data Warehouse", ICDT, January 1997, Delphi Greece.

[14] G. Sanjay, C. Alok, "Parallel Data cube Construction for high performance On_line analytical processing", IEEE.Inter. Confer.1997, pp10-14

[15] C. Zhang and J. Yang, "Genetic algorithm for materialized view selection in data warehouse environments," Proceedings of the International Conference on Data Warehousing and Knowledge Discovery, LNCS, vol.1676, pp. 116-125, 1999

[16] Antoaneta Ivanova, Boris Rachev, "Multidimensional models – Constructing data cube", International Conference on Computer Systems and TechnologiesCompSysTech'2004.

[17] A. Shukla, PM Deshpande, JF Naughton, "materialized view selection for multidimensional datasets", Proceeding of the 24th international conference on very large databases, New York, August 1998, pp 488-499.

[18] A. Shukla, Deshpande PM, Naughton JF, "materialized view selection for multi-cube data models" 7th international conference on extended database technology, Germany, March 2000, Springer, pp 269-284.