# 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*

Eashwaran R
*Department of Information Technology*
*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 also referred to as lattice of cuboids there is 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

Data warehouse systems provide OLAP tools for interactive analysis of multidimensional data at varied levels of granularity. OLAP tools typically use the data cube and a multidimensional data model to provide flexible access to summarized data. For example, a data cube can store precomputed measures (like count and total sales) for multiple combinations of data dimensions (like item, region, and customer). Users can pose OLAP queries on the data. They can also interactively explore the data in a multidimensional way through OLAP operations like drill-down (to see more specialized data, such as total sales per city) or roll-up (to see the data at a more generalized level, such as total sales per country). Although the data cube concept was originally intended for OLAP, it is also useful for data mining. Multidimensional data mining is an approach to data mining that integrates OLAP-based data analysis with knowledge discovery techniques. It is also known as exploratory multidimensional data mining and online analytical mining (OLAM). It searches for interesting patterns by exploring the data in multidimensional space. This gives users the freedom to dynamically focus on any subset of interesting dimensions. Users can interactively drill down or roll up to varying levels of abstraction to find classification models, clusters, predictive rules, and outliers. This chapter focusses on data cube technology. In particular, we study methods for data cube computation and methods for multidimensional data analysis. Precomputing a data cube (or parts of a data cube) allows for fast accessing of summarized data. Given the high dimensionality of most data, multidimensional analysis can run into performance bottlenecks. Therefore, it is important to study data cube computation techniques. Luckily, data cube technology provides many effective and scalable methods for cube computation. Studying such methods will also help in our understanding and the development of scalable methods for other data mining tasks, such as the discovery of frequent patterns. preliminary concepts for cube computation. These summarize the notion of the data cube as a lattice of cuboids, and describe

The basic forms of cube materialization. General strategies for cube computation are given. Section 5.2 follows with an in-depth look at specific methods for data cube computation. We study both full materialization (that is, where all of the cuboids representing a data cube are precomputed and thereby ready for use) and partial cuboid materialization (where, say, only the more “useful” parts of the data cube are precomputed). The Multiway Array Aggregation method is detailed for full cube computation. Methods for partial cube computation, including BUC, Star-Cubing, and the use of cube shell fragments, are discussed. In Section 5.3, we study cube-based query processing. The techniques described build upon the standard methods of cube computation presented in Section 5.2. You will learn about sampling cubes for OLAP query-answering on sampling data (such as survey data, which represent a sample or subset of a target data population of interest). In addition, you will learn how to compute ranking cubes for efficient top-k (ranking) query processing in large relational datasets. In Section 5.4, we describe various ways to perform multidimensional data analysis using data cubes. Prediction cubes are introduced, which facilitate predictive modeling in multidimensional space. We discuss multifeature cubes, which compute complex queries involving multiple dependent aggregates at multiple granularities. You will also learn about the exception-based discoverydriven exploration of cube space, where visual cues are displayed to indicate discovered data exceptions at all levels of aggregation, thereby guiding the user in the data analysis process.

If we had to materialize cuboid C and there exist two ways to do so one by first materializing cuboid A with 100 rows or by either materializing cuboid B with 20 rows as the intermediate cuboid joining base cuboid and cuboid C, The cost of materializing any cuboid is the sum of costs of the cuboid that join it to the base cuboid, in our case cuboid A and B are one level away from base cuboid both joining cuboid C to the base and we know the cost of materializing any cuboid is proportional to the cost of processing the number of rows in that cuboid. Hence path involving cuboid B is more efficient compared to the path involving cuboid A as the number of rows in cuboid B are lower than that in cuboid A, furthermore, we can also say that the number of rows is nothing but the product of cardinality of each dimension of the cuboid. Our purposed algorithm named lowest first does the above explained in O(nlogn) preprocessing and O(n) for each query.

## II. RELATED WORK

## III. Lowest First Approach

The idea is to head towards the locally optimal solution by visiting cuboid that have all the attributes of current cuboid plus one more which has lowest value among the valid and unvisited attributes where being valid means they are present in the target cuboid and continuing until we reach the target. In Further sections we will prove using principle of mathematical induction how the local optima leads us to the global optima. The algorithm has a preprocessing complexity of O(nlog(n)) and query time complexity of O(n) where n is the number of total attributes. Consider a lattice of cuboids having a total of 6 attributes Q, P, M, S, U given to us that 0<=Q <=P<=S <=U <=M (values belong to set of natural number) and we have to find optimal path from <Q> to <Q, P , S ,U , M>, Now a path is optimal if the intermediate cuboids generated while traversal occupy minimum space and take minimum processing time so we write a generic expression for path value and try to optimize it, which is for the current case A1*A2*A3 + A1*A2 + A1 and now we need to find a bijection between the variables in our generic expression and the sorted difference set [P,S,U,M] which gives us the optimal value for the expression. To minimize

the whole expression what we do is minimize each term in the expression while making sure it stays valid, So starting from the last term we have · A1 is minimum when its value is P. So, A1 = P (as P has lowest value in our difference set followed by S, U and M) · A1*A2 is minimum when its value is P*S and we already have A1 =P so A2 = S. · A1 *A2* A3 is minimum when its value is P*S*U and we already have A1 =P, A2=S so A3=U and our optimal assignment order is A1 =P, A2=S, A3=U and the path becomes <Q> => <Q,P> => <Q,P,S> => <Q,P,S,U> => <Q,P,S,U,M> which gives us lowest possible space and time complexity for any possible path from source cuboid to target cuboid. .

## IV. Mathematical Proof

We propose that for any natural number n if the values of attributes are in order A1<=A2<=A3<=A4 ... <= An then the correct bijection which gives minimum value for the expression B1*B2*B3 *B4*...*Bn-1*Bn + ... + B1*B2*B3 + B1*B2 + B1 is when B1=A1, B2=A2 B3=A3, ... Bn-1=An-1 , Bn=An ,.

Base Case (n=1) for n=1 we have only one attribute, ie A1 and the expression becomes A1 which has no other possible variation hence it is the minimum possible, base case is proved hence.

Inductive Step. Fix k ≥ 1, and suppose that the following holds(assumption), that is for n=k the minimum value of expression B1*B2*B3 *B4*...*Bk-1*Bk + B1*B2*B3 *B4*...*Bk-1 + ... + B1*B2*B3 + B1*B2 + B1 is when B1=A1, B2=A2 , B3=A3,, ..., Bk-1=Ak-1 , Bk=Ak .

Now It remains to show that for n=k+1 our proposal holds, So according to what we proposed minimum value of expression B1*B2*B3 *B4*...*Bk*Bk+1 + B1*B2*B3 *B4*...*Bk-1*Bk + B1*B2*B3 *B4*...*Bk-1 + ... + B1*B2*B3 + B1*B2 + B1 is when B1=A1, B2=A2 , B3=A3,, ..., Bk-1=Ak-1 , Bk=Ak ,Bk+1=Ak+1

Now moving on to proving if this is true, we know that from our inductive assumption for n=k that expression containing terms till n=k are minimum possible and they are common in expression for n=k+1, So now we just need to show the k+1th term is minimum it can be, then the whole n=k+1 expression will become optimal, and that can done So, we have n non negative integer variables in order 0<=A1<=A2<=A3<=A4 ... <= An and if for some k+1<=n we have to select k+1 variables such that their products is minimum then it is obviously to select the least k+1 variables as they would result in least value when multiplied, So the k+1th term in our expression comes out to be B1*B2*B3 *B4*...*Bk*Bk+1 with Bk+1=Ak+1 which proves that it holds for n=k+1 also and hence using principle of mathematical induction we have proved our Hypothesis

## V. Algorithm

**Preprocessing:**

1) User input the values and names of all attributes. 2) Check if they are valid. 3) Create an array named sorted_attr by sorting the attributes given by the user based on the values of those attributes in ascending order.

**Query:**

1) User input for the source and target cuboids. 2) Check for their validity using boundary checks. 3) Create an array named difference_set containing attributes that arent present in source cuboid but are present in target cuboid(these are the attributes that would create possible paths from source to target) 4) Create Hash map named hash_diff_set of the elements of difference_set to reduce random access complexity down to O(1) 5) Iterate over the sorted_attr containing all the attributes in sorted order by values, and for each element check if hash_diff_set[sorted_attr[i]] is present, if its present then push that element to a new array named sorted_diff_set so that after iterating over sorted_attr we get sorted_diff_set containing elements of difference_set in O(n) time complexity. 6) Now the path from source to target is starting from source and then to cuboid containing attributes of source plus first element of sorted_diff_set, from there to cuboid containing first 2 elements of sorted_diff_set and so on till we finally reach target cuboid

## VI. Implementation With Example

Let us consider a lattice of cuboids having 4 attributes <P,O,I,U>, having values 20, 5, 10, 30 respectively and we have two path finding queries A)from <P> to <P,O,I,U>, B) from <P> to <P,O,U>

**Steps:**

A) Query #1 <P> to <P,O,I,U>

1) Sorting all the attributes we get [O,I,P,U]

2) Our difference_set contains attributes present in target but not in source ie, [O,I,U]

3) Sorting it using hash maps and previously sorted attribute array we get sorted_diff_set to be [O,I,U].

4) Now starting from <P>(source) we first visit <P,O>(first element of sorted_diff_set) then from <P,O> we visit <P,O,I> and then from <P,O,I> we visit <P,O,I,U> which is

our target. And this optimal path matches with the result of brute-force algorithm having O(2^n) worst-case time complexity.

B) Query #2 <P> to <P,O,U>

1) Sorting all the attributes we get [O,I,P,U]

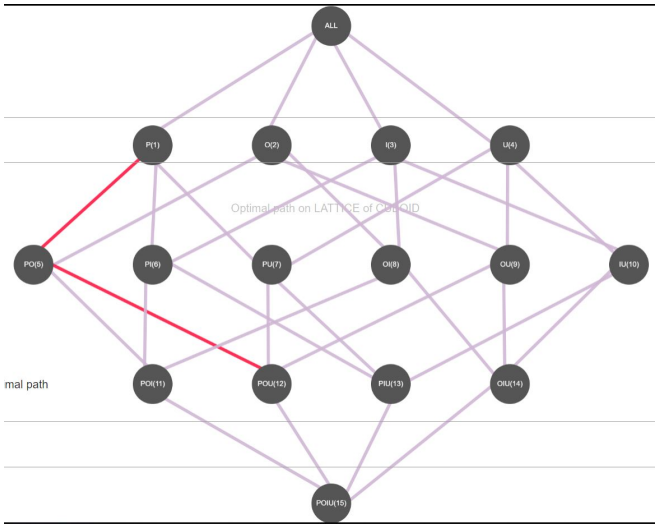
2) Our difference_set contains attributes present in target but not in source ie, [O,U]

3) Sorting it using hash maps and previously sorted attribute array we get sorted_diff_set to be [O,U].

4) Now starting from <P>(source) we first visit <P,O>(first element of sorted_diff_set) then from <P,O> we visit <P,O,U> which is our target. And this optimal path also matches with the result of brute-force algorithm having O(2^n) worst-case time complexity.

Below are the optimal paths highlighted for both Queries A and B, generated by a web implementation of the above algorithm available at https://latticeofcuboid.ml/ . The time complexity was O(nlog(n)) for the first preprocessing thereafter it is O(n) per Query which is 2 in our case.

**Query #1:**




## VII. Conclusion

In this paper we have proposed an algorithm which has a preprocessing time complexity of O(nlogn) and query time complexity of O(n) So it can handle massive queries to find optimal path in lattice of cuboids.

## References


[1] Gray, J., Bosworth, A., Layman, A., and Pirahesh, H.; "Data cube: A relational operator generalizing group-by, cross-tab, and roll-up"; Proc. of International Conf. on Data Engineering. New Orleans: IEEE Press. 1996.

[2] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao; "Efficient OLAP Operations in Spatial Data Warehouses"; Proc. of SSTD 2001, Springer-Verlag LNCS 2121, pp. 443-459, 2001

[3] Min Wang, Bala Iyer; "Efficient Roll-Up and Drill-Down Analysis in Relational Databases (1997)"; Proc. of 1997 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery.

[4] S.Bhattacharya, A.Cortesi; "A distortion-free watermark framework for relational databases"; Proc. 4th Int. Conference on Software and Data Technology (ICSOFT'09) Sofia, Bulgaria (2009).

[5] A.Cortesi, F.Logozzo; "Abstract Interpretation Based Verification of Non-Functional Requirements", Proc. COORDINATION 2005, Namur, LNCS 3454, pp 49-62 (2005).

[6] A.Cortesi; "Widening Operators for Abstract Interpretation"; Proc. 6th IEEE Int. Conf. on Software Engineering and Formal Methods, pp. 31-40, Cape Town, SA (2008).