ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2012

CROME : Indexing The Partial Data Cubes

K.DHANASREE[1] , C.SHOBHA BINDU[2],R.SATISH[3]

Associate professor,DRKIST[1] ,Associate professor,JNTUA[2],Assistant professor,DJRIET[3]

***ABSTRACT* — In recent years multiple group -bys are computed using various OLAP applications. For the computation of group bys most of the OLAP applications use cube operator. The cube operator computes group-bys on all possible combinations of list of attributes. OLAP products typically run faster than other approaches, because its possible to index directly into the data cube structure to collect subsets of data. However for large data sets with many dimensions OLAP solutions aren't always effective. To make the user queries faster, parts of the data cube are pre computed and the aggregates are presented in cuboids. When the cube dimensions are more in number, it is difficult to search for the pre computed cuboids and there is a chance of pre computing the same cuboid redundantly. As pre computing cuboids requires much of storage space , even redundant evaluation of data cuboids is wastage of memory. This also increases the cost of evaluation and performance time. So a better indexing is needed on what cuboids are pre computed. Here we present a CROME based method of indexing ,that indexes the pre computed partial cuboids. The Crome based method calculates Crome values to index the pre computed cubes, in order to avoid redundant evaluation of the pre computed cuboids. If the user query Crome does not exist in the crome data structure then that cuboid can be pre computed.**

Keyword: olap,cube,lattice,crome,D-sequence, ordered maximal group.

I. INTRODUCTION

With years of research and development of data warehouses and technology, a large number of data warehouses have been successfully constructed and deployed in applications, and data cube has become an essential component in most data ware house systems and in some extended relational database systems. Since we are required to retrieve large number of records from different data bases we are at an urge to summarize them on multi dimensions. This multidimensional nature of data has led to OLAP applications.. Many of the modern business problems can be solved by these OLAP applications. Recently introduced data cube operator is supporting such aggregates in OLAP data bases. The pre computation of all or part of data cube can greatly reduce the response time and enhance the performance of online analytical processing . pre computation of the data cube is therefore necessary. The OLAP[1] cubes are used to summarize data by pre computations. Data cube computation and representation approaches were classified into two main categories: (i) full cube computation and (ii) partial cube computation and representation. The Full cube computation approach computes all the cells, for a given data cube, while the partial cube computation approach computes a subset of a cube cells(cuboids) for the given set of dimensions, or a smaller range of possible values for some of the dimensions.

The pre computation of the different summary views (group-bys) of a data cube is critical to improve the response time of data cube queries for On-Line Analytical Processing (OLAP). Many solutions have been proposed for generating the entire data cube. A data cube consists of two kinds of attributes : measures and dimensions. The dimensions consists of attributes like sales, city, time period etc. The measures are numeric counts like profit, total of sales etc. The pre computation of all or part of data cube can greatly reduce the response time and enhance the performance of online analytical processing. To support this goal of an OLAP application the most efficient ways are pre compute all cells in the cube, or pre compute no cells , or pre compute some of the cells. If the whole cube is pre computed then the query response time is faster. But the disadvantage is pre computation requires lot of memory. We can pre compute none of the cells in order to minimize memory requirements. The disadvantage is user query response time is slow. With these two disadvantages we pre compute the few cells. When few cells are pre computed and presented it is better to index which cells are pre computed , in order to increase the efficiency of query response time.

In this paper we present a CROME based indexing method to index the pre computed cells. The method uses two algorithms : 1. Algorithm to calculate D-sequence 2. Algorithm to generate ordered maximal groups. . While maintaining the datawarehouse the warehouse admin pre computes few cells of the data cube. The ware house admin calculates the D-sequences and Crome values for each of the possible group bys on the dimensions of the ware house cube. .These Crome values are indexed using any of the data structures . Now when the user queries for the aggregates on the dimensions, using the cube operator, the algorithms discussed in this paper extracts the dimensions from the user query , calculates the D-sequence and generates the ordered maximal groups . we then find Crome values for the user query .using these Crome values we search the already

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2012

indexed Crome values by the admin. If the Crome value exist the aggregates can be retrieved. If the Crome values doesn't exit then that cell of the cube is not pre computed and a request can be put to the admin to pre compute the required cell. There by indexing the pre computed cubes eliminates the redundant computation of the cube cells.

We have discussed about the basic data cube , its representation and its lattice form in Section II. We have discussed about our crome based indexing approach in Section III. Section III also emphases on the D-Sequence generation algorithm and on ordered maximal group generation algorithm.

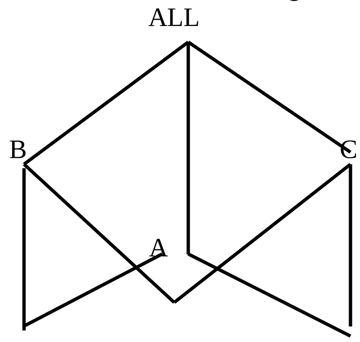
II. DATA CUBE

To retrieve and support decision queries effectively, a new operator, CUBE BY, was proposed [2]. The cube operator is a multidimensional extension of the relational operator GROUP BY. The CUBE BY operator computes group bys corresponding to all possible combinations of grouping attributes in the CUBE BY clause. A cell (cuboids) of a cube is one group by. As the cube by attributes increases cube operator becomes more expensive and the size of the data cube increases.. The huge size of a data cube makes data cube computation time-consuming [3]. Much of research work is done on the size of data cube and several methods have been introduced to reduce the size of a data cube and hence its computation time and storage overhead are reduced. Condensed cube, Dwarf , Quotient cube and indexing using QC-trees[4] are some approaches. The basic idea of all these methods is to compute the whole cube using optimized memory. Many indexing techniques are been proposed to index the data cube. Indexing techniques like sort based and hash based are efficient only for smaller dimensions.

In SQL the collection of aggregate queries can be expressed using the cube operator as follows

```
Select A, B, C sum(s)
Cube by A, B, C;
```

This query will result in the computation of 8 Group-bys: ABC, AB, BC, AC, A, B, C and ALL. ALL is the aggregate of all attributes, as shown in Fig.1 and executes them separately.



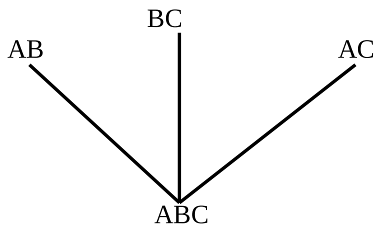


Fig .1 : Cube by A,B,C.

A lattice framework to represent the hierarchy of the group-bys was introduced in [5]. This is an elegant model for representing the dependencies in the calculations and also to model costs of the aggregate calculations. A scheduling algorithm can be applied to this framework substituting the appropriate costs of computation and communication. A lattice for the group-by calculations for a four dimensional cube is shown in Fig. 2. Each node represents an aggregate and an arrow represents a possible aggregate calculation which is also used to represent the cost of the calculation. Calculation of the order in which the GROUP-BYs are created depends on the cost of deriving a lower order (one with a lower number of attributes) group-by from a higher order (also called the *parent*) group-by.

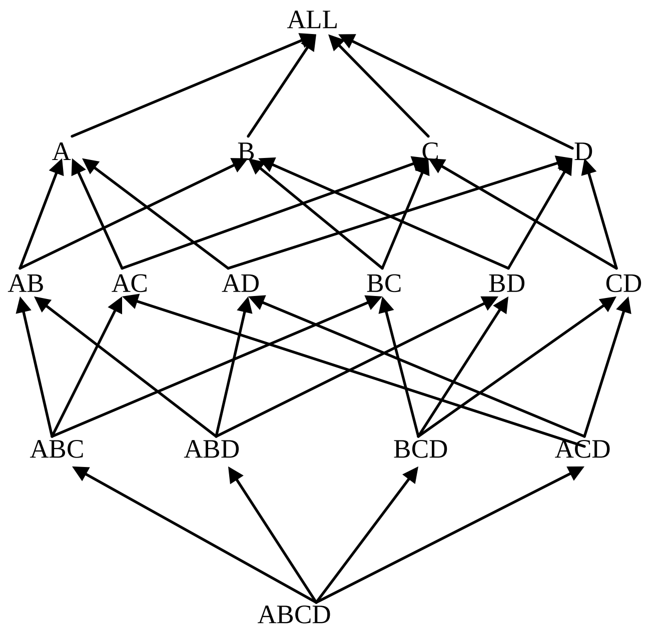


Fig. 2 : Lattice of possible group bys

When the OLAP application with the cube operator in implemented based on cube attributes the aggregations are retrieved from the already pre computed aggregates. Many of the OLAP applications provide a way to pre compute the entire cube [6] or pre compute a part of the cube [7]. In order to retrieve these pre computed cells efficiently a better indexing is necessary. Indexing on the pre computed cells enhances the query performance.

III. CROME BASED METHOD

A data cube with 4 attributes A, B, C, D have 16 possible combinations of group bys as shown in Fig.2.
ABCD, ABC, ABD, BCD, ACD, AB, AC,AD, BC,BD,CD,A,B, C, D, and ALL.

Definition :1 (Ordered Maximal group) : The combinations consisting of maximum attributes at each level in an alphabetical ordered combination.

Definition:2 (Crome codes) : Crome codes are binary equivalents of ordered numeric values of ordered maximal groups.

We built Crome codes for each group by combination as shown in Table 1

TABLE 1: Crome codes

Ordered maximal Group	Number equivalents	Binary equivalents
ABCD	0	0000
ABC	1	0001
ABD	2	0010
ACD	3	0011
BCD	4	0100
AB	5	0101
AC	6	0110
AD	7	0111
BC	8	1000
BD	9	1001
CD	10	1010
A	11	1011
B	12	1100
C	13	1101
D	14	1110
ALL	15	1111

For an SQL cube query

```
Select A, B, C, sum(s)
from   sales
cube by A,B,C;
```

We take the base node as the ordered maximal group in the cube by condition. In the above SQL query base node is ABC with n=3 attributes.

Definition : 3 (Crome ID) : we define Crome ID of a node as a number given to the node such that, no two adjacent nodes with common n-1 base node attributes should be given the same ID.

Definition : 4 (Crome Cube): The cube in which the possible group bys are given crome ids.

Definition : 5 (Minimal Crome Cube): Minimal Crome cube is the Crome cube which uses Minimal Crome IDs.

For the above SQL query the Minimal Crome Cube is

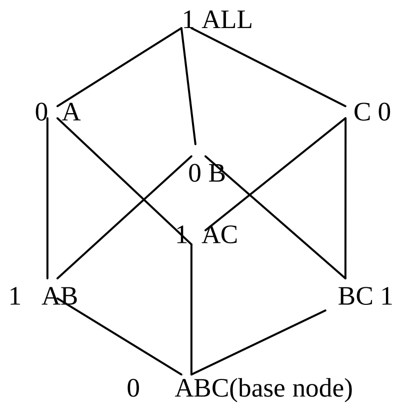


Fig. 3: Minimal Crome Cube

Definition : 5 (D-Sequence of Minimal Crome Cube): D-sequence is sequence of Crome IDs at each levels of ordered maximal groups starting from the base node.

For the minimal Crome cube of Fig. 3 the D-Sequence is
D - Sequence = (0, 1, 1, 1, 0, 0, 0, 1).

A . Crome Values

The Crome Values are taken as D-sequence together with the ordered maximal groups binary equivalents. For the cube shown in Fig.1 the Crome Values are

Crome value of Cube (ABC,AB ,AC, BC,A,B,C,ALL)
= (0001, 0101, 0110, 1000,1100,1101,1110)
+ (0,111,0,0,0,1)

B . Indexing using Crome Values

Let us suppose that we are maintaining a data cube whose partial cubes are calculated on dimensions ABCD.
Then there are 16 possible group bys-
ABCD,ABC,ABD,ACD,BCD,AB,AC,AD,BC,BD,CD,A,B,C,D,ALL.
For each group by the data base admin calculates the D-sequences as given by definition 5.

These D-sequences are stored in a structure as shown Table 2

TABLE 2: D-Sequences

Possible group by	D-Sequence
ABCD	(0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0)
ABC	(0,1,1,1,0,0,0,1)
ABD	(0,1,1,1,0,0,0,1)
AB	(0,1,1,0)
BC	(0,1,1,0)
A	(0,1)

The Crome values for each possible group by is tabulated as shown in Table 3

TABLE 3: Crome values

Possible group by	Crome values
ABCD	(0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0) + ordered maximal binary equivalents of ABCD
ABC	(0,1,1,1,1,0)+ ordered maximal binary equivalents of ABC

Now when the user queries
Select A, B, C sum(s) from sales
Cube by A, B, C;

The user system calculates the Crome values of ABC from the query using the algorithms discussed in section C and section D

C . Algorithm to calculate D-Sequence

Now when the user queries
Select A, B, C sum(s) from sales
Cube by A, B, C;
The D-sequence and ordered maximal group for the above query is calculated as follows:
The above query has 3 dimensions.
We implement the D-Sequence algorithm to generate the D-sequence.
Algorithm D-Sequence:

```
Step 1 : Take number of dimensions.
Step 2 : Generate N+1 levels
Step 3 : Generate ncn 0's at N+1 th level.
Step 4 : Generate ncn-1 1's at N th level
Step 5 : Generate ncn-2 0's at N-1 th level.
Step 6 : Generate nc1 0's at last level.
Step 7 : Generate the D-sequence by combining
         the number of 0's and 1's at each level
         with 2^n entries in the sequence.
```

For dimensions A,B,C our algorithm generates D-sequence as follows:

Number of dimensions	:	3
Number of levels	:	3+1 = 4

Level 4 :	generate 3c3	0's	:	0
Level 3 :	generate 3c2	1's	:	1,1,1
Level 2 :	generate 3c1	0's	:	0,0,0
Level 1 :	generate 3c1	1's	:	1

Combining all the entries of each level to get the D-sequence-

D-sequence- (0,1,1,1,0,0,0,1)

D . Algorithm to generate ordered maximal groups:

```
Step 1 : get dimensions from the query.
Step 2 : find number of dimensions, N.
Step 3 : set R=N
Step 4 : generate NcR maximal group combinations
         whose length is N and sort the combinations
step 5 : R=R-1;
step 6 : repeat steps 4 and 5 until R=N-R.
```

For the above query the algorithm to generate ordered maximal group works as follows :

Step 1 : get dimensions ABC
Step 2 : number of dimensions 3
Step 3 : r= 3
Step 4 : generate 3c3=1 ordered maximal group which can be either ABC or BCA or ACB. Sorting the group results in ABC.
Step 5 : R=R-1; R=2
Step 6 : repeating step 4, for R=2
We get the next ordered maximal groups as AB,AC, BC, A, B, C.

All these ordered maximal groups are given number equivalent and tabulated as shown in Table 4

TABLE 4 : Number equivalents

Ordered maximal Group	Number equivalent
ABC	0
AB	1
AC	2
BC	3
A	4
B	5
C	6
ALL	7

The binary equivalents for the numbers are tabulated as shown Table 5

TABLE 5 : Binary equivalents

Ordered maximal groups	Number equivalents	Binary equivalents
ABC	0	0000
AB	1	0001
AC	2	0010
BC	3	0011
A	4	0100
B	5	0101
C	6	0110
ALL	7	1111

Now the user system calculates the Crome value using calculated D-sequence and ordered maximal group bys. The user module then sends these Crome values to the admin module. The admin module checks these Crome value in its index structure. If the Crome value exists then the user requested data cube exists pre computed in the database and is retrieved to the user. If the Crome value doesnt exist then the admin module pre computes the data cube and stores in its data base. The Crome based indexing thus avoids the redundant pre computation of data cubes.

IV. IMPLEMENTATION AND RESULTS

To test how well our method performs, we implemented the two algorithms using C.. We constructed two separate data bases with common sales table cube. The values for each attribute is independently chosen and the tuple size taken is 24 bytes. We have noticed that when compared to B ,B+ indexing methods , Crome based method reduced the cost of calculation of redundant cubes and increased the query performance. Our indexing method is easy and convenient. We varied the dimensions of the cube from 0-25 in steps of 5 and measured the performance ratio.Normal methods could reduce a 10% cost when compared to Crome based which has reduced the cost of evaluating the redundant cubes upto 30% . Search time has reduced more than 30% when compared to other indexing techniques.

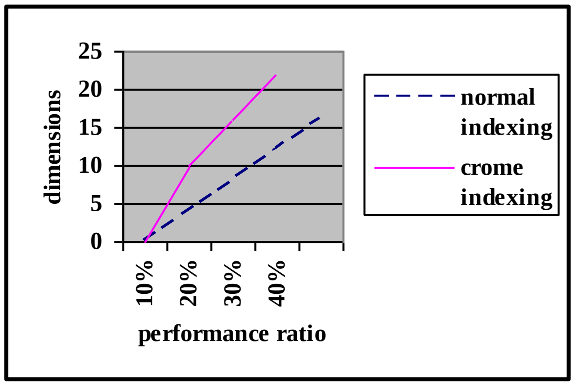


Fig. 4 : Performance comparison of crome indexing

V . CONCLUSION

The task of all OLAP or multidimensional data analysis applications is the ability to simultaneously aggregate across many sets of dimensions. Computing multidimensional aggregates is a performance bottleneck for these applications. The cube operator requires computing group bys on all possible combinations of attributes. We showed a Crome based indexing method to index the pre computed cubes .The approach has provided an indexing to increase the user query performance and eliminated the redundant evaluation cost of the data cells and showed a better performance.

REFERENCES :

[1] Goil S. and Choudhary A., "Parallel Data Cube Construction for High Performance On-Line Analytical Processing", Proc. 4th Intl. Conf. on High Performance Computing, Bangalore, India, 1997.

[2] Harinarayan V., Rajaraman A. and Ullman J.D."Implementing Data Cubes Efficiently", Proc.SIGMOD'96.

[3] Sarawagi S., Agrawal R., and Gupta A., "On Computing the Data Cube", Research Report 10026, IBM Almaden Research Center, San Jose, California, 1996.

[4] H. Gupta et al" Index selection for OLAP",proc.of the 13th ICDE, 1997.

[5] Sarawagi.S, Agrawal.R.,and Gupta.A.,"On computing the Data Cube",Research Report 10026,IBM Almaden Research Center, San Jose,California,1996

[6] A. Shukla, P. Deshpande, and J. Naughton"Materialized view selection for multidimensional datasets". In Proceedings of the 24th International,VLDB Conference,1998.

[7] K. Ross and K. Zaman. Optimizing selections over data cubes. Technical Report CUCS-011-98,Dept of Computer Science, Columbia university,1997.