Abstract— Data Cube in the contest of data warehousing and OLAP is core operator. The data cube was proposed to pre compute the aggregation for all possible combination of dimension to answer analytical queries efficiently. It is a generalization of the group-by operator over all possible combination of dimension with various granularity aggregates. Efficient and Compressed computation of data cube are Fundamental issues. Data Warehouses tend to be order of magnitude larger than operational database in size. So by studying and comparing all these methods we can find out that which methods are applicable and suitable for which kind of Data. Here I have compared range cube with Bit cube. Each problem is of particular interest in the field of data analysis and query answering. So by comparing various methods, we can verify the trades of between time and space as per the requirements and type of problem. Different Types of measures are available for aggregating the datasets. Such as Major-Minor, Count, Sum etc. So comparative study shows that we can find which measures can be compute data cube incrementally.

Keywords: Data cube, Range cube, Bit cube, Comparison, conclusion

I. INTRODUCTION

Since the introduction of Data Warehouse and OLAP systems, research has focused in the design of efficient algorithms for cube pre-computation. Given a base table R, the data cube can be defined as a generalization of the standard GROUP-BY operator in which the aggregations of every combination of attributes, appearing in the group-by clause, are computed. An n-dimensional data cube is composed of cells of the following form (v1, v2, . . . , vn, c) where c is a list of measures. A cell can have a value vj or a * symbol to indicate that all values of that dimension have been grouped. A cell is called m-dimensional, if and only if there are exactly m (m ≤ n) values among (v1, v2, . . . , vn) which are not *. It is called a base cell if m = n and aggregate cell if m = 0 [1, 2, 7].

Different kinds of cubing are possible. The full cube computation allows the pre-computation of all parts (cuboids) of a data cube [7]; the iceberg cube introduces conditions to materialize only a subset of a cube satisfying them [5]; the closed cube compresses a cube by representing only closed cells and predicting the remaining from them [8] and shell-fragment cube selects in advance the Dimensions of interest [7]. Full and iceberg cubing computations are recognized as fundamental step for other categories. However, iceberg cubes can be efficiently computed only for measures having the anti-monotonic property [1]. Cubing algorithms perform aggregations following two major approaches: top down and bottom-up. In a top-down model, group-bys with a small number of dimensions are obtained from the ones with a higher number of dimensions through caching intermediate computations [1, 2]. Star cube integrates strength of both algorithms. Star performs good compared to top-down and bottom-up. MM cube performs better than star cube. But it requires large memory because of recursive calls. Range cube fulfills goal of both space and speed in cube computation. Range cube uses new hyper tree structure called range trie. Range tree generates compressed structure in less time. Bit cube is uses Bottom-up approach. Bit Cube uses bitmaps as lookup tables, which is inverted indexes [7], to identify values shared by the same records in a partition, allowing fast aggregation computations. Bitmaps are compressed by using an adaptation of the Word-Aligned Hybrid (WAH) [7] algorithm. Such a compression leads to a significative improvement of the performance of Bit Cube in terms of both space requirements and running time. So, Range and Bit cube both generates compressed structure in minimum run time. But which one is best?

In this paper I have compared range cube with bit cube according to parameters like dimension, cardinality, min_sup etc.
and compared two best algorithms like Range and Bit Cube. I have presented in brief them as per following.

### A. Range Cube

Range Cubing as an efficient way to compute and compress the data cube without any loss of precision. A new Data Structure, Range Trie, is used to compress and identify correlation in attribute values, and compress the input dataset to effectively reduce the computational cost. The range cubing algorithm generates a compressed cube, called range cube, which partitions all cells into disjoint ranges. Each range represents a subset of cells with the identical aggregation value as a tuple which has the same number of dimensions as the input data tuples. The range cube preserves the roll-up/drill-down semantics of a data cube [4].

Trie of size (h, b) is a tree of height h and branching factor b. All keys can be regarded as integers in range [0, bh]. Each key K can be represented as h-digit number in base b: K1K2K3...Kh. Keys are stored in the leaf level, path from the root resembles decomposition of the keys to digits. It compresses the base table into a range trie, so that it will calculate cells with identical aggregation values only once.

The traversal takes advantage of simultaneous aggregation, that is, the m - dimensional cell will be computed from a bunch of (m + 1) - dimensional cells after the initial range trie is built. At the same time, it facilitates Apriori pruning. The reduced cube size requires less output I/O time.

#### Table 1: The Base Table

<table>
<thead>
<tr>
<th>Store</th>
<th>City</th>
<th>Product</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C1</td>
<td>P1</td>
<td>D1</td>
<td>$100</td>
</tr>
<tr>
<td>S1</td>
<td>C1</td>
<td>P2</td>
<td>D2</td>
<td>$50</td>
</tr>
<tr>
<td>S2</td>
<td>C2</td>
<td>P1</td>
<td>D2</td>
<td>$200</td>
</tr>
<tr>
<td>S2</td>
<td>C2</td>
<td>P1</td>
<td>D2</td>
<td>$120</td>
</tr>
<tr>
<td>S2</td>
<td>C3</td>
<td>P2</td>
<td>D2</td>
<td>$40</td>
</tr>
<tr>
<td>S3</td>
<td>C3</td>
<td>P3</td>
<td>D1</td>
<td>$250</td>
</tr>
</tbody>
</table>

#### Fig. 1: Range Trie Constructions

#### Range Algorithm

1. Input: root1: the root node of the range trie;
2. dim: the set of dimensions of root1;
3. upper: the upper bound cell;
4. lower: the lower bound cell;
5. Output: the range tuples;
6. begin
7. for each dimension iDim 2 dim do
   8. upper[startDim] = c’s start dimension value;
   9. set upper’s dims 2 fc->key - startDim to _;
   10. set lower’s dims 2 c->key to c->KeyValue;
   11. outputRange(upper, lower, c->quant-info);
   12. if c is not a leaf node
      13. call range cubing on c recursively;
   14. g Traverse all the child nodes of the root and
      15. merge the nodes with same next startDim value;
   16. upper[iDim] = lower[iDim] = _;
   17. iDim = next StartDim;
   18. g end;

#### B. Bit Cube

Bit Cube is a bottom-up cubing algorithm. It’s a new computational model for high dimensional datasets is proposed. The method is based on the precomputation of “shell fragments” (i.e. vertical partitions of the dataset) together with their materializations. Efficiency is obtained by using “inverted indexes” (i.e. for each value in each dimension, a list of record-ids associated with it is stored). Finally, intersection among fragments is performed online by computing full group-bys on desired dimensions. Bit Cube uses inverted indexes also and following suggestion in [7], it optimizes them by using bitmaps. However, unlike previous methods, Bit Cube uses bitmaps also as lookup tables to identify values shared by the same records. Therefore, this allows to speed-up the aggregation process by reducing the number of possible candidates to group. Bit cube computes full cube and iceberg cube both. Here, I have implemented iceberg bitmap cube.

#### Fig. 2: Bitmap Data structure

In the iceberg cube construction, Bit Cube allows to prune unnecessary computations by introducing apriori based strategy. It discards partitions with a number of records less than the minimum support. For each value of dimension di meeting the constraint, its aggregation value is stored.
During the iceberg cube computation, the visit of the Bit Cube Tree is performed from right to left. Such a visit allows computing pair aggregations which will be used to speed-up higher dimensional aggregations located at the left side of the tree. For example, in the aggregation tree of Fig 2, dimensions A and D are first aggregated. Then, this knowledge is used to optimize the aggregation of A, C and D in the following way. First, a third level bitmap (iceberg bitmap) is introduced. It is associated to each dynamic vector of each dimension di.

**Definition (Iceberg Bitmap).** Let P be a partition of a base table R with respect to a value w. Let Ddi be the dynamic vector of di with respect to P. Let t be the size of D di in the partition. An iceberg bitmap Biceberg di is a bitmap of t bits where Iceberg di [v] = 1 if the group-by of the pair (w,v), where v is a value of di, satisfies the iceberg condition.

**Fig. 3:** Bit Cube Iceberg aggregation with iceberg condition

---

**Aggregate Procedure called by Bit Cube**

AggregateFirstLevel(c, size)

// dk is the dimension of c
1 for each value v of Ddk Such that Count of v ≥ minSup do //iceberg
2 WriteOutputRec;
3 if node c is not a leaf then //iceberg
4 if count of v is 1 then
5 Direct Aggregate (dk, v);
6 continue;
7 else
8 Aggregate(c, size);
End AggregateFirstLevel.

---

**III. IMPLEMENTATION WITH RESULT**

In this paper I have implemented Range cube and Bit cube. Then compare them with each other according to parameter like cardinality, tuple, min_sup, dimension etc. Figure 4 and 5 shows comparison graph between range and bit cube. We can see in both graphs that bit cube gives better performance speed wise compare than range cube because of WAH compression algorithm. Also bit cube very efficient for skewed data when cardinality and min_sup are low.

**Comparison graph based on cardinality**

<table>
<thead>
<tr>
<th>Run time in seconds</th>
<th>Range Cube</th>
<th>Bit Cube</th>
</tr>
</thead>
</table>
| Comparison based on Dimension
| Range Cube | Bit Cube |

**Fig. 4:** comparision graph based on cardinality w.r.t. D=6, min_sup= 10, T= 5000
Comparison between cube techniques
(IJSRD/Vol. 1/Issue 3/2013/0004)

System Specification:

Both the two algorithms Existing and Proposed MM Cube are coded using Microsoft Visual Studio 2008 C#.Net with MS Access 2007 database on a Core i3 M350 @ 2.27GHz, system with 3GB of RAM. The system runs Windows 7 Ultimate with Service Pack 1.

IV. CONCLUSION

Here, we analyzed and compared range cube and bit cube according to different parameter like dimension, cardinality, min_sup, tuple etc. Range and bit cube both fulfills the goal of cube computation like space and time.

Both generate compressed structure in less time. Both are different their structure wise. Range cube preserves its semantics like roll-up and drill-down. Bit cube is very faster than Range cube because of bitmap and WAH compression technique. Generally both are very useful in Data Warehouse and Olap system because of their speed and less memory requirement. But when very large and skewed data set is used then bit cube is very useful for cube computation.

ACKNOWLEDGMENTS

I would like to thank my guide, Mr. Nilesh Padariya , for his professional leadership and valuable advice.

REFERENCES

[1] Zheng Shao, Jiawei Han Dong Xing. “A MM-Cubing: Computing Iceberg Cubes by Factoring the Lattice Space”. University of Illinois at-Champaign, Urbana, IL 61801, USA, 2004

[2] Dong Xin, Jiawei Han, Xiaolei Li, Benjamin W. Wah “AStar-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration” University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA


