Computing Data Cubes and Aggregate Query Processing

Anna Rozeva

Abstract: *The paper is dealing with data cubes, multidimensional data structures built from data warehouse for OLAP purposes. Performing aggregations over cube dimensions and the way they are stored is considered to be a problem worth optimization. A two-tier multilevel list structure for storing cubes has been proposed. Algorithms for tiers' setup and maintenance when records are loaded into the data warehouse have been designed. An overview of analytical queries has been presented. Algorithm for processing analytical query over data cubes stored in the proposed structure has been outlined.*

Key words: *Data Warehouse, Data Cube, Cube Computation, Storing a Cube, OLAP, Query Processing.*

INTRODUCTION

Data warehouse is a data store serving the purposes of decision support [2]. It is system architecture for information processing with application in decision making. As a result basic design issues for the data warehouse are query throughput and response time. The data model providing for such a purpose is the multidimensional one. The basis of the model consists in viewing a numeric value, i.e. measure as being dependent on a set of attributes, dimensions. In a data warehouse of a retail business appropriate measure is sales and dimensions product sold, customer and time of sale. The most typical database schema for the multidimensional model is the star schema. Dimensions are represented by dimension tables and measure is contained in the fact table. The relation of each dimension table to the fact is achieved by means of a foreign key. The fact table can be perceived as a cube. Cube being a logical entity giving away values of a certain fact arising at an intersection of a combination of dimensions – fig.1.

sale	Product	Client	Date	Amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

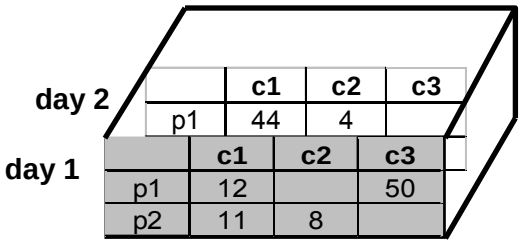

Fig.1. Fact table represented by 3-dimensional cube

The cube, corresponding to the sales fact provides for viewing the sales measure by product, by client, by date. Cubes have been designed as logical models serving on-line analytical processing (OLAP) applications. OLAP provides powerful and fast tools for reporting on data. The basic notion of OLAP consists in computing multiple related group-bys and aggregate measure values over dimensions. For improving response time data cubes are designed to hold measure values together with summarized information (aggregates). The purpose of the cube is to output measure of a fact at a particular aggregation level. The set up of a cube consists in computing aggregated measures grouped by all subsets of the dimensions. Precomputing aggregates and storing them in the cube provides for speeding analytical queries. There are two basic approaches for physical implementation of cube computation – ROLAP and MOLAP. The difference between the two OLAP physical models is in the kind of structure data is stored in – relational tables or numeric arrays. In ROLAP a cell of the cube is represented as a record, part of the attributes identifies the location of the record in the multidimensional space of the cube and another attribute holds the data value contained in the cell. In MOLAP the

elements of the array hold numeric values only and the position in the array represents the connection of numeric values to the dimensions, i.e. the dimensions are implemented by the dimensions of the array. Further on in the paper MOLAP cube computation issues are presented, the problem of sparseness being examined and a multilevel list structure for handling it has been outlined. Application issues connected with structure's maintenance and performance of analytical queries are pointed out.

CUBE COMPUTATION

- **Aggregations over Data Cube**

Most of the OLAP queries over the cubes require aggregation of measures at some level. The following aggregations can be computed for the cube from fig.1: sales grouped by product, customer and date; sales by product and customer; sales by product and date; sales by customer and date; sales by product; sales by customer; sales by date and total sales. By 3 dimensions there are 2^3=8 granularities at which aggregated sales can be computed. For consistent representation of the aggregated values they are denoted as "*". The aggregates produced from the cube in fig.1. are shown in fig.2.

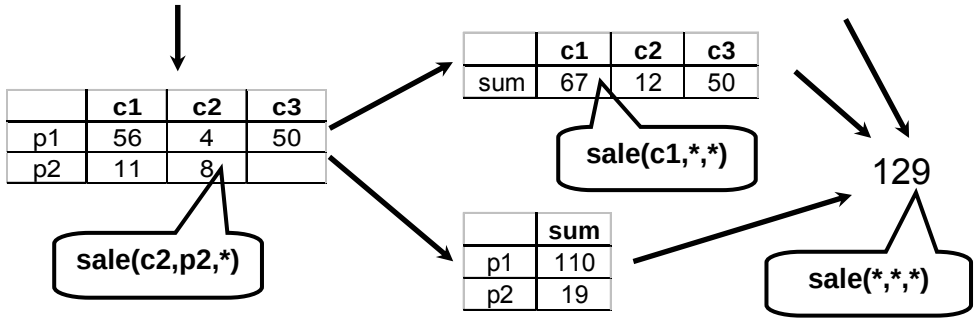

Fig.2. Granularities of aggregated values for data cube

A granularity for aggregated values in the cube is referred to as a subcube. This is a region of the cube with smaller dimension. A d-dimensional cube has 2^d subcubes. Each subcube can be computed from the base data. Some subcubes can be directly computed from other subcubes. A cube therefore can be represented as a set of subcubes – fig.3.

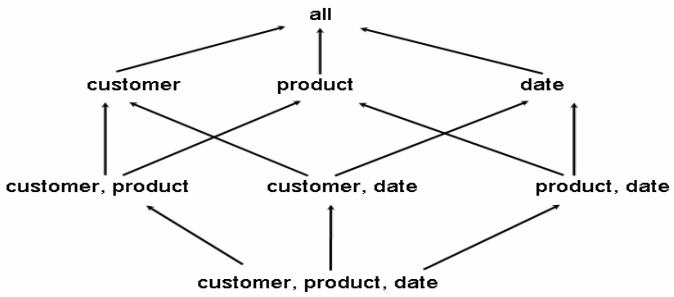

Fig.3. Data cube – subcube lattice

An edge between two subcubes shows that one subcube can be computed from the other. Edges are from finer to coarser granularity. The coarser granularity is specified by one attribute less than the finer one. The SQL notation for computing aggregations over cube dimensions implies several Group-Bys and can be expressed by CUBE operator [4]:

```
SELECT Product, Customer, Date, Sum(Sales)
FROM Sales_Table
CUBE BY Product, Customer, Date
```

The main issue of data cube computation concerns aggregations along dimensions and combinations of dimensions and their storage for further effective access by OLAP queries.

- **Overview of Storage Techniques for Data Cubes**

MOLAP implementation of data cubes deals with arrays and position based access to the values therein. A problem arising with the array representation of a cube is sparseness. Sparseness with respect to some attributes means small relation cardinality compared to the cross product of attribute domains. Sparseness is due to large domain sizes of some Cube By attributes and/or large number of Cube By attributes in the query. Algorithms for implementing a cube as an array aiming to overcome sparseness are presented in [1], [3], [6] and [10]. The one from [6] computes the various subcubes using generated pipelined paths. The idea is in sharing computation across paths. Path generation implies the underlying relation to be sorted in a particular attribute ordering. The algorithm from [1] builds subcube with a single attribute first and then subcube with two attributes, etc. At the cost of duplicating work computation of records which don't meet minimum support is reduced. The algorithm from [10] implements the technique of chunking for dealing with sparse data. It concerns overlapping group-bys in different computations to reduce memory requirements. In case of insufficient memory to hold the cubes several passes over the input data will be needed. A tree-like structure for storing the cube has been designed in [3].

- **Proposal of Two-Tier Structure for Storing Data Cube**

We've designed a structure for storing the cube which is based on implementing the subcube lattice shown in fig.3. The cube storage architecture is proposed to be a two-tier one. The lowest level of the lattice denoted (customer, product, date) represents the subcube without any aggregation, i.e. facts from the fact table. This subcube is stored as a first tier. The rest of the subcubes containing aggregation on one or more dimensions form the second tier of the storage structure. These are records with at least one "*" element as shown in fig.2. We'll examine a cube with small number of dimensions and moderate attribute cardinalities. All aggregates for existing facts will be computed. The elements of the second tier are computed from the first one. The first tier provides for answering queries concerning detail data. We'll consider now a storage structure for the tiers.

The facts of the first tier are organized in a multilevel list structure. The number of levels corresponds to the number of dimensions. The number of elements in a level corresponds to the dimension's cardinality. Dimensions are ordered by cardinalities and in this order they are set as levels of the structure. The first level contains the elements of the least cardinality dimension. In the root of the storage structure pointers are set up to the elements of the first level. The structure is implemented by scanning the fact table. For each record the dimensions' members are checked in the order by which levels have been decided to be set up. For the corresponding member of the first level pointer is created for the next lower level and further on to the next level, etc. The fact value is stored in the record at the last level. In the process of scanning the fact table the lists are checked and if the elements of the levels have already been created the fact value is aggregated (usually summed) with the value in the record at the last level. Otherwise new elements of the corresponding levels are created as well as fact value records at the last level. For the table from fig.1 the initial setup of the first tier multilevel storage structure is performed in the following way:

1. Choose a dimension to represent the first index level - D_1. Level elements are d_{11}, d_{12}, …, d_{c1}, c_i is the cardinality of D_i in a fact table FT. Order the other dimensions as D_2 and D_3;
2. Initialize c1 pointers pi from the root to the first level elements;
3. For each record in FT check D_1, go to the corresponding element in the first level;

4. Check for pointer pi to next level dimension D_2 element; if it exists – check for dimension D_3 element; if it exists sum the fact table value with the one (V) in the record from the last level;

5. If dimension D_2 element doesn't exist in the second level of the storage structure - initialize pointer pi to the D_2 element and further on to D_3 element, create record for storing the fact value (V) of the current FT record.

Repeat steps 3, 4 and 5 for all fact table records.

The measures (aggregated facts AV) in the second tier are computed from the first tier by traversing pointers from the root to the last level. The root of the second tier represents the most aggregated subcube – (*, *, *). The next level subcubes are the ones with aggregates on two dimensions – (D1, *, *), (*, D2, *), (*, *, D3). Each of them contains as many samples as the cardinality of the dimension is. AV denotes the aggregated measure computed from the facts of the first tier. The next lower level contains AVs for the less aggregated subcubes - (D1, D2, *), (D1, *, D3), (*, D2, D3) - fig.4.

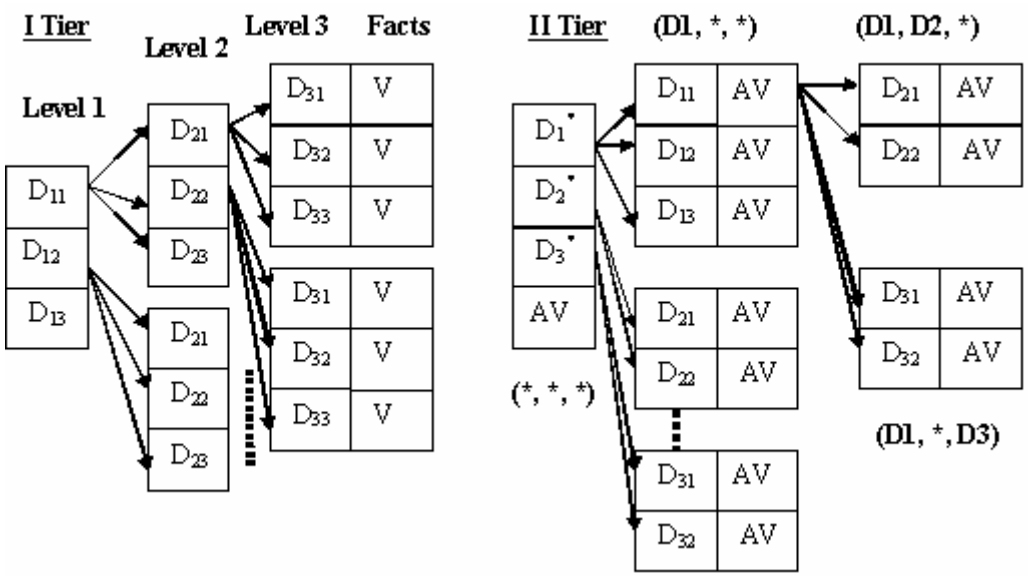

Fig.4. Multilevel list structure for the subcube lattice

- **Storage Structure Maintenance**

After the initial setup of the two tiers of the multilevel storage structure it has to be maintained and updated when new records are appended to the warehouse fact table. Maintenance issues concern:

1. Update of the facts V in the first tier and the corresponding AVs in the subcubes of the second tier.

2. Update the levels with new dimension members.

Update of Vs is to be performed when new fact is appended to the table for a combination of dimension members that have already been placed into the levels of the first tier. In this case the corresponding pointers to the V values are traversed and the new value is summed with the one in the last level record. Further on update of AVs in the second tier is performed. This is done on the subcubes of all granularities by following the pointers. When a record is appended with a dimension member that hasn't been created in the tiers' levels update has to be performed on the tier structure itself. This implies initializing pointers and creating new records to hold the corresponding fact value in the first tier. This is done by performing the setup procedure from the previous subsection. The second tier update implies the setup of pointers for the new samples of the subcubes as well as update of the AVs of the corresponding higher granularity subcubes.

OLAP QUERIES OVER DATA CUBE

A data cube is set up to enhance the performance of OLAP queries. Researches concerning this topic have developed in three main directions, i.e. data storage structures [7], view materialization [5] and indexing. Our previous work on indexing data warehouse tables is presented in [8] and [9]. Our current work implements storage structures techniques. Further on we'll explain the way a cube stored in the structure shown in the previous section processes OLAP queries. OLAP query selects data from subcubes of the data cube. Major operations are slicing, dicing, rolling-up, drilling-down and pivoting. These operations are implemented in the CUBE operator shown in the introduction by means of aggregation, subtotalling, cross tabulation and grouping. An example of OLAP query for the cube shown in fig.1. is: "Output the sum of p1 product sales to customers c1 and c2 between d1 and d5". The query criteria may be:

- Single value from the domain of a dimension (p1);
- Partial set, i.e. subset of domain values for the other dimension (c1, c2);
- Contiguous range in the domains of another dimension (d1…d5).

The number of dimensions specified in the query represents its degree. Formally a query can be represented [3] by a record with a degree determined by the number of specified dimensions and elements - the dimensions with a notation showing the type for each of it – value, partial set or range. Following the notation for the storage structure presented in fig.4. the sample query that was stated will have the following formal expression:

(a) Q = (d1, d2, d3) = (D11, D21, [D31, D32]).

Aggregate values denoted in fig.2. by "*" can represent query values too. The formal expression of such a query will be:

(b) Q = (d1, d2, d3) = (D11, *, [D31, D32]).

The queries stated so far are full, meaning that they specify criteria for all cube dimensions. Further on the way of processing a full query over the cube structure shown in fig.4 will be explained. The query marked with (a) doesn't include "*" elements in its criteria. The values that'll result from it can be found by processing the first tier of the cube by following the pointers corresponding to the stated values. The (b) query includes "*" so it'll be processed over the second tier. Pointers to the corresponding subcubes will be followed and the AV values will be output. Pseudo code of the algorithm for processing an OLAP query over the cube's two-tier storage structure is shown in fig.5.

```
Input: Query(D11, *, [D31, D32]) / Query (D11, D21, [D31, D32]);
    If D11 = "*" Or D21 = "*" Or D31 = "*" Then
            Process Second_Tier (D11, *, [D31, D32]);
            Root.D1*
            Subcube(D1, *, *).D11
            Subcube(D1, *, D3)
            Output (D31.AV, D32.AV);
        Else
            Process First_Tier (D11, D21, [D31, D32]);
            Level1.D11
            Level2.D21
            Output (D31.V, D32.V);
    End;
```

Fig.5. Algorithm for query processing over the two-tier storage structure

The query criteria are checked and the tier serving for its answering is determined. The first query is processed over the second tier. Following the pointer of D_{11} to subcube's (D_1, *, D_3) sample the aggregated measures AV for D_{31} and D_{32} are output. The second query is processed over the first tier and the result values V are located by following the pointer of D_{11} from the first level to D_{21} from the second to D_{31} and D_{32} at the last one.

CONCLUSIONS AND FUTURE WORK

The storage structure for a data cube that has been designed consists of two tiers – one for the finest granularity cube data and the other for the aggregated values. Pointer-based multilevel list structure implements both tiers. The multilevel list follows the representation of a cube as subcube lattice. The separation of data by granularity provides for faster location of aggregates by accessing the aggregate tier only. Algorithms for list setup and maintenance after data load have been designed. This storage structure overcomes the problem with sparseness in the array-based methods by storing existing data values only. Classification of analytical queries and a formal description of OLAP query have been presented. Algorithm for designing query execution plan as well as for tier processing has been outlined.

Future work is intended in implementation of hierarchies for cube dimensions and tuning the storage structure for supporting hierarchical dimensions. A matter of interest represents processing analytical queries implying selection of aggregates satisfying certain condition and optimized multilevel list search.

REFERENCES

[1] Beyer, K., R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg Cubes. In Proceedings of the 1999 ACM SIGMOD Conference on Management of Data. Philadelphia. 1999

[2] Chaudhuri, S., U. Dayal. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record (ACM Special Interest Group on Management of Data). 26:1. pp 65-74. 1997

[3] Fu, L., J. Hammer. CubiST: A New Algorithm for Improving the Performance of Ad-hoc OLAP Queries. In Proceedings of the ACM Third International Workshop on Data Warehousing and OLAP (DOLAP). Washington DC. 2000

[4] Gray, J., S. Chaudhuri, A. Bosworth, A. Layman, et.al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. Data Mining and Knowledge Discovery. 1:1. pp. 29-53. 1997

[5] Gupta, A., V. Harinarayan, D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In Proceedings of the Eight International Conference on Very Large Databases. Zurich. pp. 358-369. 1995

[6] Ross, K., D. Srivastava. Fast Computation of Sparse Data Cubes. In Proceedings of the 23rd International Conference on Very Large Databases. Athens. 1997

[7] Ross, K., K. Zaman. Optimizing Selections over Data Cubes. In Proceedings of IEEE International Conference on Scientific and Statistical Database Management. Berlin, 2000. IEEE Computer Society

[8] Rozeva, A. Index Structure for the Fact Table of a Star-Join Schema and Template Query Processing. In Proceedings of the International Conference on Computer Systems and Technologies CompSysTech'2003, pp. II.14-1- II.14-6. 2003

[9] Rozeva, A. Bitmap Indexes – Enhancing the performance of Multidimensional Ad Hoc Queries. In Proceedings of International Scientific Conference Computer Science'2004. Sofia. 2004

[10] Zhao, Y., P. Deshpande, J. Naughton. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. SIGMOD Record (ACM Special Interest Group on Management of Data). 26:2. pp 159-170. 1997

ABOUT THE AUTHOR

Assoc.Prof. Anna Rozeva, PhD, Department of Computer Systems and Informatics, University of Forestry, Sofia, Phone: +359 2 91907 340, E-mail: arozeva@ltu.bg.