MULTIDIMENSIONAL DATA CUBE APPROACH VERSUS MATERIALIZED QUERY APPROACH

Sonali Chakraborty[1], Dr. Jyotika Doshi[2]
[1]Gujarat University, [2]GLS University

Abstract

For Online Analytical Processing, multidimensional cubes serve as a dimensional structure of star schema. It stores data warehouse aggregates having n-dimensions. Millions of aggregates are possible for large enterprise requiring huge storage space for precomputing and materializing all the cuboids. Probability that user will require all aggregates on all members of each dimension is very less. Materialized query approach stores query fired by the user in a relational database with its result, timestamp, threshold and frequency without the need of updating it with every data warehouse refresh. Incremental updates are done on the results only when query is fired next time, hence reducing processing time. Another advantage of proposed approach is the possibility of storing non-aggregate results.

Index Terms: Multidimensional Data Cube, Materialized Query, Saving storage space, OLAP

I. INTRODUCTION

Materialized query approach suggests storing executed query along with its result and other factors like timestamp, frequency and threshold in a relational database. If an equivalent query is fired next time, result extraction from data warehouse can be avoided, or only incremental updates can be done; thus saving processing time.

The probability that user will be interested and will require all aggregates on all members of each dimension is very less. To eliminate storage of so many results calculated in anticipation in case of multidimensional cubes, materialized queries are proposed. For materialized queries there is no requirement of precomputing and storing results. Instead, the query is stored or materialized only when the user fires the query for the first time. Updation of all materialized queries is not done with every data warehouse refresh. They are updated only when next time the query is fired. To avoid load on the database storing the queries, all the queries will be evaluated periodically based on timestamp and the threshold values defined for each query.

[1, 2] "A data cube allows data to be modelled and viewed in multiple dimensions defined by dimensions and facts". They do not confine data to two or three dimensions but are n-dimensional. In case of three dimensional cubes, eight types of aggregations or queries are possible, i.e. 2 n, where n is the total number of dimensions. The total number of cells in the cube will be (total members in dimension 1) x (total members in dimension 2) x (total members in dimension3). Cube need not have an equal number of members in each dimension.

To understand the proposed approach, consider an example of an insurance company.

The stored data is about customers enrolled for various policies.

Dimensions involved in this example are like customer's id/name, birth date, gender, marital status, city, annual income, policy name, policy category, distribution type etc.

Members (distinct values of a dimension) of some dimensions are as follows:

Dimension	Members	Number of members
gender	Male, Female	2
city	Ahmedabad, Mumbai, Kolkata, Delhi	4
marital status	Single, Married, Divorced, Widow	4
policy category	Life-Mandatory (11), Life-Non mandatory (10), Nonlife-mandatory (0,1), Nonlife-non mandatory (00)	4

Table 1: Members of dimensions gender, city, marital status and policy category.

Measures (usually aggregate) of interest may be average annual income, total customers having specific type of policy etc.

Fact is a collection of related data items. It contains values of dimensions of interest and values of measures.

It can be well understood with following examples.

Fact 1: "The average annual income of the "Male" customers living in "Mumbai" enrolled for policies under category "11" is Rs. 5 lakhs".

Here, the dimensions are gender, city, policy category while 5 is a measure (average annual income) calculated considering these dimensions.

Consider another fact:

Fact 2: "The average annual income of the "Male" customers having marital status "1" living in "Mumbai" enrolled for policies under category "11" is Rs. 5 lakhs".

The dimensions used in Fact 2 are gender, city, policy category and marital status.

II. LITERATURE REVIEW

Panos Vassiliadis, Timos Sellis [3] stated that data cubes provide the functionality needed for summarizing, viewing, and consolidating the information available in data warehouses. They performed comparisons between relational-oriented, cube-oriented, defined standards like TPC-D, OLEDB and statistical models like OOM85, RR91 standards. Panos Vassiliadis [4] introduces some simple cube operations. He proposed a model for multidimensional databases based on the notion of base cube. It is used for calculation of results of cube operations and support series of operations on cube. They preserve the results of previous operations with applied aggregate functions. They also provided mapping of multidimensional model to relational model and to multidimensional arrays using a mapping function.

Venky Harinarayan et al. [5] discussed that for query optimization; some cells can be materialized instead of computing them from raw data every time. They investigated which cells are to be materialized when it becomes too expensive to materialize all cells. A lattice framework is used for expressing the dependencies among views. Then a greedy algorithm which works on this lattice picks the views which are to be materialized considering the constraints.

N. Colossi et al. [6] define metadata extensions that will help multidimensional schema designers to describe the structure of schemas to multidimensional query and analysis tools. They described Web services for OLAP providing metadata for multidimensional data and XML query results. They discussed various SQL extensions, such as improving grouping operations to reduce the number of queries to access cube data hence, increasing efficiency of the query that is executed, automatic summary tables or materialized query tables to reduce query times and extent of redundant query processing.

Anindya Datta, Helen Thomas [7] propose a model of data cube and an algebra for supporting OLAP operations on that cube providing a means to concisely express complex OLAP

queries. Usually OLAP products view measures as functions of dimensions hence, making dimensions and measures set static. This restricts users from generating queries based on measure restrictions. They demonstrated the capabilities of proposed algebra which allows uniform treatment to dimensions and measures i.e. query dimensions by restricting measures.

Rakesh Agrawal et al. [8] propose a data model based on hypercube and some algebraic operations which provides semantic foundation to multidimensional databases by extending their current functionality. The approach provides a symmetric treatment to dimensions as well as measures and also provides support for multiple hierarchies along dimensions and support for ad hoc aggregates.

Prasad Deshpande et al. [9] provide a framework for computing and evaluating the cube. They present algorithms using heuristics based on sorting which tries to minimize disk accesses by overlapping cuboid computation and hence reducing the number of sorting steps.

Seok-Ju Chun et al. [10] proposed an algorithm for reducing cost by maintaining search efficiency using an index structure referred as Δ-tree. It is a hierarchical data structure storing information about the updated cells in the data cube.

Jayavel Shanmugasundaram et al. [11] propose a cube compression technique based on statistical clustering the data. They suggested that by estimating the probability density of the data, a compact data representation supporting aggregate queries can be build.

Chang Li, X. Sean Wang [12] developed an algebraic query language called as grouping algebra as an extension of relational algebra. These relational operations are then used for manipulating basic groupings for obtaining complex groupings.

Carlos A. Hurtado et al. [13] discussed the issue of dimension updates required for adapting multidimensional database. Structural updates of dimensions also take place such as addition of categories or modification in hierarchical structure. With these updates materialized aggregate views i.e. cubes must be maintained. In their approach, they proposed a model for updating domains of dimensions and for structural updates of the dimensional hierarchies. They framed an algorithm to maintain materialized aggregate views over dimension level.

Paper	***Description***
Panos Vassiliadis [4]	• Proposed a model for multidimensional databases based on the notion of base cube. • Used for calculation of results of cube operations and support series of operations on cube. • Provided mapping of multidimensional model to relational model and to multidimensional arrays using a mapping function.
Venky Harinarayan et al. [5]	• Investigation of cells for partial materialization. • Lattice framework expressing dependencies among views. • Greedy algorithm working on lattice, picking the views to be materialized considering the constraints.
N. Colossi et al. [6]	• Described web services for OLAP which provides metadata for multidimensional data and XML query results. • Discussed SQL extensions like improving grouping operations to reduce number of queries in cube. Hence, increasing efficiency of the query that is executed, automatic summary tables or materialized query tables are generated to

	reduce query times and extent of redundant query processing.
Anindya Datta, Helen Thomas [7]	• Model of data cube and algebra to concisely express complex OLAP queries. • Proposed algebra allows uniform treatment to dimensions and measures i.e. query dimensions by restricting measures.
Rakesh Agrawal et al. [8]	• Data model based on hypercube and algebraic operations providing semantic foundation to multidimensional databases. • Provides a symmetric treatment to dimensions and measures • Support for multiple hierarchies along dimensions and for ad hoc aggregates.
Prasad Deshpande et al. [9]	• Framework for computing and evaluating the cube. • Algorithms using heuristics based on sorting for minimizing disk accesses by overlapping cuboid computation and hence reducing the number of sorting steps.
Seok-Ju Chun et. al [10]	• Index hierarchical data structure referred as Δ-tree storing information about the updated cells in the data cube. • Hybrid approach for providing an approximate or precise result with respect to OLAP range-sum queries.
Jayavel Shanmugasundaram et al. [11]	• Cube compression technique based on statistical clustering the data. • By estimating the probability density of the data, a compact data representation supporting aggregate queries can be build.
Chang Li , X. Sean Wang [12]	• Developed an algebraic query language called as grouping algebra as an extension of relational algebra. • Relational operations are used for manipulating basic groupings for obtaining complex groupings. • Also includes order related operations for retrieving sorted results.
Carlos A. Hurtado et al. [13]	• Model for updating domains of dimensions and for structural updates of the dimensional hierarchies. • Framed an algorithm to maintain materialized aggregate views i.e. cubes over dimension level.

Table 2. Summary of related work

III. STORING RESULTS USING MULTIDIMENSIONAL CUBE APPROACH AND PROPOSED MATERIALIZED QUERY APPROACH

For the above mentioned dimensions and members, n-dimensional cubes will be formed. One may derive many facts from the cube by applying different aggregates such as sum, average, max and min.

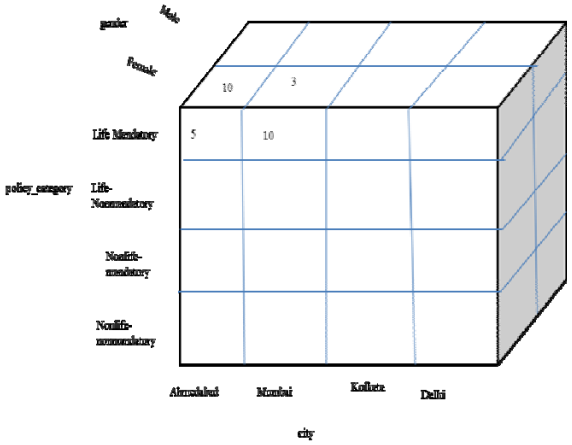

Fig.1. A 3-D data cube calculating measures for the average income of customers having dimensions gender, policy category and city.

Total number of cells without hierarchies = number of members in gender x number of members in policy category x number of members in city i.e. (2 x 4 x 4 = 32 cells).

For the fourth dimension, marital status having four members, cuboids generated are as follows:

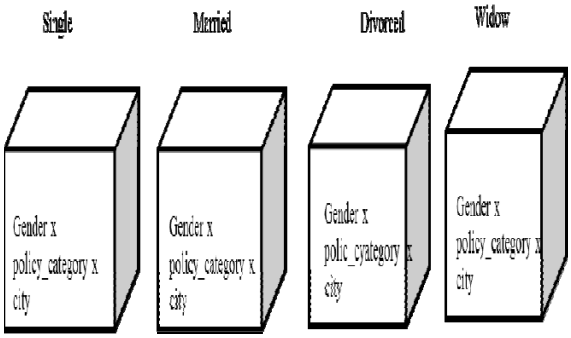

Fig. 2. A 4-D data cube calculating measures for the average income of customers having dimensions gender, policy category and city, marital status.

To find aggregate for 200 cities, for 100 types of policies over 30 distribution channels, a 3 D cube will have 6, 00,000 aggregates.

For given set of dimensions, cuboids are generated for each possible subsets of dimension. This results into lattice of cuboids forming a data cube. In this way, 2n cuboids or group-by can be computed, where n is the number of dimensions. This huge storage space requirements problem is referred to as curse of dimensionality [1].

The equivalent SQL query for the 3-Dimensional cube for Fact1 is:

SELECT Avg (customer.annual_income) AS AvgOfannual_income, customer.city, customer.gender, category.type
FROM ((category INNER JOIN policy ON category.cat_id = policy.cat_id) INNER JOIN cust_policy ON policy.pol_id = cust_policy.po_id) INNER JOIN customer ON cust_policy.c_id = customer.c_id
GROUP BY customer.city, customer.gender, category.type;

SQL query for Fact 2:

SELECT Avg (customer.annual_income) AS AvgOfannual_income, customer.city, customer.gender, category.type, customer.marital_status
FROM ((category INNER JOIN policy ON category.cat_id = policy.cat_id) INNER JOIN cust_policy ON policy.pol_id = cust_policy.po_id) INNER JOIN customer ON cust_policy.c_id = customer.c_id
GROUP BY customer.city, customer.gender, category.type, customer.marital_status;

Result generated when SQL query is fired for Fact 2.

Avgincome	city	gender	policycateg	marital_status
9	ahmedabad	0	10	1
2	ahmedabad	1	11	0
2	mumbai	0	00	0
3	mumbai	1	00	1
9	mumbai	1	10	1
3	mumbai	1	11	1

Fig. 3. Average annual income calculated after performing group by on four dimensions (city, gender, policy category, marital status).

For Fact 2, if average income of customers is to be calculated for marital status =1, the results satisfying the criteria is fetched from the previous stored result.

Avgincome	city	gender	policycateg	marital_status
9	ahmedabad	0	10	1
3	mumbai	1	00	1
9	mumbai	1	10	1
3	mumbai	1	11	1

Fig. 4: Results fulfilling criteria "marital status = 1" extracted from previous stored result

Results of the query can also be grouped based on data warehouse refresh dates.
E.g. Query for Fact 2 was last fired on "20-12-2016" (dd-mm-yyyy) (timestamp for Fact 2 query).
Data warehouse refresh were done on "01-09-2016", "15-12-2016".
When the query was fired on "20-12-2016", it extracted the results till data warehouse refresh date of "15-12-2016".

Avgincome	ger	marita	city	policycateg	dwupdate
2	0	0	mumbai	00	15-12-2016
9	0	1	ahmedabad	10	15-12-2016
2	1	0	ahmedabad	11	01-09-2016
3	1	1	mumbai	00	01-09-2016
3	1	1	mumbai	11	01-09-2016

Fig. 5. Average annual income of customers as per data warehouse update on "15-12-2016".

Data warehouse refresh was later done on "22-12-2016" and again the query for Fact 2 is fired on "25-12-2016".

Avgincome	ger	marita	city	policycateg	dwupdate
9	1	1	mumbai	10	22-12-2016

Fig. 6. Average annual income of customers for the data warehouse refresh after 15-12-2016.

Next time when any materialized query is fired incremental updates are done from warehouse data and is appended with the past result. Its result, timestamp, frequency values are updated.

Avgincome	ger	marita	city	policycateg	dwupdate
2	0	0	mumbai	00	15-12-2016
9	0	1	ahmedabad	10	15-12-2016
2	1	0	ahmedabad	11	01-09-2016
3	1	1	mumbai	00	01-09-2016
9	1	1	mumbai	10	22-12-2016
3	1	1	mumbai	11	01-09-2016

Fig.7. Average annual income of customers grouped by data warehouse refresh dates.

The probability that fired query is not completely equivalent to materialized query is high. Variation among queries can be with respect to fields used in query for performing group-by. The varying fields can be overwritten / added / removed from the materialized query. Query will be executed from the data warehouse and the result is updated.

Example: In Fact 2, the average income of customers was calculated based on dimensions city, gender, policy category and marital status. Next time when the query is fired average income of customers is to be calculated excluding dimension city.

Avgincome	gender	policycateg	marital_status
2	0	00	0
9	0	10	1
3	1	00	1
9	1	10	1
2	1	11	0
3	1	11	1

Fig. 8.Average annual income of customers grouped by gender, policy category, marital status.

Unlike multidimensional cubes, materialized queries can also be fired for non-aggregate output.

Example: Finding names of the policies enrolled by the customers staying in "Mumbai"

```
SELECT policy.pol_name, customer.city
FROM policy INNER JOIN (cust_policy INNER
JOIN customer ON cust_policy.c_id =
customer.c_id) ON policy.pol_id =
cust_policy.po_id
GROUP BY policy.pol_name, customer.city
```

HAVING (((customer.city) ="mumbai"));

pol_name
healthplus
megalife
vehplus

Fig. 9. Non-aggregate output for a materialized query

Threshold values vary depending on results fetched by the queries.

For e.g. Management generates monthly reports for getting count of customers enrolling for various policies. The related query will be fired at least once in a month. Whereas calculating total premium collected in a year grouped by policies will require the query to be fired annually. While saving query in database, these thresholds are defined.

Frequency of the query is updated every time it is executed. If any query is inactive i.e. frequency is less than defined threshold, then it is removed from the relational table. Evaluation of frequencies of materialized queries and removal of inactive queries saves storage space.

Following table shows comparison between multidimensional cubes and materialized query approach.

Multidimensional Cubes	***Materialized query***
• Aggregates are computed for all dimensions irrespective of queries fired.	• Query is executed and stored only when it is fired by the user for the first time.
• If no records for any combinations of dimensions or members of dimensions exist, cells will still be generated with NULL or zero values.	• Rows will not be generated for any combination of dimensions or members whose record does not exist.
• Huge storage space is required to store aggregates. Number of cells required will be 2n; n is the number of dimensions.	• Less storage space required compared to multidimensional cubes.
• Storage space can be reduced by performing partial computation.	• Storage space is reduced by periodically evaluating frequencies of queries and removing them if no longer required.
• In partial computation, user defined criteria has to be configured. Criteria may not be same for all queries.	• Threshold value is decided by the user for each query and it depends on the nature and type of query.
• Usually multidimensional cubes are refreshed with every data warehouse refresh.	• Materialized query is refreshed only when it is fired next time. Only incremental updates are done on it.
• Measures are calculated which are generally aggregate values.	• Query having non aggregate output are also executed.
• Cuboids are generated for each combination of dimension.	• No separate relational tables are generated for each combination of dimension.
• Can be implemented using ROLAP and MOLAP structures. [1,2]	• Will use only relational database approach for storing results.

Table 3. Comparison of multidimensional cubes and materialized query

IV. CONCLUSIONS

Materialized query is a memory efficient approach which is stored along with its result only when it is fired by the user. Factors stored with the query like frequency, threshold helps eliminating non-frequent queries. These properties reduce the storage requirements. No rows are allocated for the combination of dimensions or members for which records do not exist. With materialized query updating incrementally, query processing time reduces compared to multidimensional cubes.

REFERENCES

[1] Jiawei Han, Micheline Kamber, and Jian Pei. Data Mining-Concepts and Techniques, Third Edition, Morgan Kaufman Publishers.

[2] G. K. Gupta. 2014. Introduction to Data Mining with Case Studies, PHI Learning Private Limited

[3] Panos Vassiliadis and Timos Sellis., "A Survey of Logical Models for OLAP databases," *ACM SIGMOD Record*, Volume 28, Issue 4, Dec.1999, 64 – 69.

[4] Panos Vassiliadis, "Modeling Multidimensional Databases, Cubes and Cube Operations," *Proceedings of Tenth International Conference on Scientific and Statistical Database Management*, 1998.

[5] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman, "Implementing Data Cubes Efficiently," *Proceedings of the 1996 ACM SIGMOD International Conference on Management of data*, 205-216.

[6] N. Colossi, W. Malloy, and B. Reinwald, "Relational extensions for OLAP,"*IBM Systems Journal*, VOL 41, NO 4, 2002.

[7] Anindya Datta and Helen Thomas, "The Cube Data Model: A Conceptual Model and Algebra for On-Line Analytical Processing in Data Warehouses'" *Decision Support Systems*, Volume 27, Issue 3, December1999, 289-301.

[8] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi, "Modeling Multidimensional Databases," *Proceedings 13th International Conference on Data Engineering*, 232-243.

[9] Prasad Deshpande, Sameet Agarwal, Jeffrey Naughton, and Raghu Ramakrishnan, "Computation of Multidimensional Aggregates," *Proceedings 22nd, VLDB Conference*.

[10] Seok-Ju Chun, Chin-Wan Chung, Ju-Hong Lee, and Seok-Lyong Lee, "Dynamic Update Cube for Range-Sum Queries," *Proceedings of the 27th VLDB Conference*.

[11] Jayavel Shanmugasundaram, Usama Fayyad, and P. S. Bradley, "Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimensions," *Proceedings of the fifth ACM SIGKDD International conference on Knowledge discovery and data mining*, 223-232.

[12] Chang Li and X. Sean Wang, "A Data Model for Supporting On-Line Analytical Processing," *Proceedings of the fifth International Conference on Information and knowledge management*, 81-88.

[13] Carlos A. Hurtado, Alberto O. Mendelzon, and Alejandro A. Vaisman, "Maintaining Data Cubes under Dimension Updates," *Proceedings 15th International Conference on Data Engineering.*