JCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Volume-6, Issue-1 E-ISSN: 2347-2693

Data Retrieval from Data Warehouse Using Materialized Query Database

Sonali Chakraborty[1*], **Jyotika Doshi**[2]

1[*]Gujarat University, Ahmedabad, Gujarat, India
[2]GLS University, Ahmedabad, Gujarat, India

[*]*Corresponding Author: chakrabartysonali@gmail.com*

Available online at: www.ijcseonline.org

Received: 06/Jan/2018, Revised: 10/Jan/2018, Accepted: 25/Jan/2018, Published: 31/Jan/2018

Abstract— Decision making in an organization requires aggregate as well as non- aggregate results, computed from data stored in data warehouse. Performance in case of result extraction from a data warehouse is an important factor. Probability that the same query is fired more often is high. This results into frequent analysis of warehouse data for fetching same results or results with incremental updates. This paper discusses an approach for storing such frequent queries along with their result, timestamp, frequency and threshold in a separate database. Past results are fetched from database and only incremental updates are done through data marts. This approach may improve performance removing or reducing execution time.

Keywords— Data warehouse, Data mart, materialized query, faster execution

I. Introduction

OLAP queries are fired by the organization for decision making. Results are retrieved from data warehouse; a repository of huge amount of data. Probability that a same query is fired frequently is high. Frequent access to data warehouse for retrieving results will lead to fetching same data again and again and thus requiring more processing time. To avoid frequent data warehouse access, there are some ways like using materialized views or multidimensional cubes.

In this paper, we suggest another approach to avoid execution of frequent queries. Results of high frequency queries can be materialized and stored in a separate database. Other details such as timestamp, frequency and threshold are also stored along with query and its results. Next time when the same query is fired, only incremental updates if necessary, are done through data marts, hence reducing data warehouse access time. Data marts are loaded with copy of most recent records during data warehouse refresh. Result retrieval from data marts will result into faster execution of query compared to data warehouse access.

Rest of the paper is organized as follows: Section II deals with related work. In section III problems with the current approaches are highlighted. In section IV methodology of suggested approach along with examples and model diagram is described. Section V concludes the research work.

II. Related Work

Issues dealing with materialized views are summarized in a tabular format in [22]. Approach of querying a multi version data warehouse by extending SQL language is discussed in [1] while in [2] authors identify factors for selecting a proper indexing technique for data warehouse applications. It also identifies factors to be considered for building a proper index on base data. Literature [3-11] deal with various issues and techniques used for maintaining and using materialized views. Algorithms for incrementally maintaining materialized views are discussed in [3], while maintenance expression for dealing with materialized views with aggregates is discussed in [4]. Algorithm for maintaining views when data sources are updated is shown in [5] and a model for keeping the view current according to the changes in the underlying database is depicted in [6]. The incremental maintenance problem of an SQL view in case of database updates using DAG is shown in [7]. Authors [8] discuss view maintenance overhead issues and deal using a lazy view maintenance approach. Literature [9] deals with the issue of computing answers to SQL queries with grouping and aggregates in case of multiset tables. Authors [10] designed algorithms SWEEP and NESTED SWEEP for incremental view maintenance. Determining a part or all of a query can be computed from materialized view is discussed in [11].

Panos Vassiliadis [12] proposed a model for multidimensional databases based on the notion of base cube and provides mapping of multidimensional model to relational model and to multidimensional arrays using a mapping function. Authors [13] explained the three categories of attributes of cube models while in [14] a lattice framework is used to investigate the cells to be materialized when it becomes too expensive to materialize all cells. A model of data cube and algebra to concisely express complex OLAP queries is discussed in [15]. Data model based on

hypercube is shown in [16]. A framework for computing and evaluating the cube is depicted in [17]. Issue of frequent change of data elements in a cube and hence the response time getting affected by the update cost and the search cost of the cube is dealt in [18] using an index hierarchical data structure referred as Δ-tree. Authors [19] discuss about the cube compression technique based on statistical clustering the data. Authors [20] developed an algebraic query language called as grouping algebra as an extension of relational algebra and formalized multidimensional data model for OLAP with its basic component as multidimensional cube.
A method for storing queries and their corresponding results is discussed in [23] where Index is maintained to keep track of queries and their results.

III. PROBLEMS WITH CURRENT APPROACHES

As per paper published by us [22] the problems / issues associated with materialized views and multidimensional cubes are summarized as below:

- Materialized views improve query processing time majorly in case of aggregate queries [11]. But view maintenance is necessary as the data sources are updated [5].
- Views are complex. Hence, it is better and cheaper to maintain them incrementally. Incremental updates can be done by applying the changes made to the base data rather than re-computing the view from the scratch [4].
- Though materialized views speed up query processing, for ensuring correct results, they should be kept up to date when accessed by a query. They can be maintained eagerly i.e. in the same transaction as the base tables are updated and these updates bear the cost of view maintenance. Overhead issues arise for maintaining them which increases as multiple views are maintained. Hence, results into poor response time for updates.
- Instead of forcing for updates, some database systems support the deferred maintenance approach, i.e. view maintenance can be delayed till the user explicitly triggers. This may lead to out-of-date views producing incorrect results. Materialized views will no longer be automatic and transparent. Query users need to have knowledge about the views used by a query, their maintenance and requirement of updation [8].
- Multidimensional data cubes are the logical model for OLAP (Online Analytical Processing). They provide the functionality needed for summarizing, viewing and consolidating the information available in data warehouse [13].
- Results are pre-computed and stored. All aggregates on all dimensions are computed in anticipation that they might be required. The problem with this is that, it occupies huge storage space for n-dimensional data cube. For materializing whole data cube having n dimensions, the number of aggregates will be 2n for snowflake schema [21].
- Few cells can be materialized instead of computing them from raw data every time. For implementation of data cube, available options are: 1) materialize the whole data cube, which will give best query response at the cost of higher storage, 2) materialize nothing resulting into computing every cell on request, 3) materialize only a part of the cube [14].

IV. SUGGESTED APPROACH

When a SQL query is fired, it is materialized and stored along with other metadata such as its result, timestamp, threshold and frequency in a separate database.

Next time, when user fires a query, first it is checked if an equivalent query exists in the database. Two queries are said to be equivalent if they fetch the same result irrespective of the order of tables and fields used in the query. Past result is extracted from the materialized query database and then it requires only incremental updates. For incremental updates, the results are retrieved from data marts.

Data marts store recent records which are loaded during data warehouse refresh. Incremental updates through data mart retrieves faster results than if fetched from data warehouse. This leads to less processing time as compared to data warehouse access, each time the query is fired.

Dimensions on which queries are fired more frequently can be stored in one data mart instead of performing join operation from multiple tables. Avoiding join operations further leads to less query execution time.

To understand the storing of materialized queries, consider the example of an insurance company. Organization stores data about the customers and the policies enrolled by them through various distribution channels. Policies vary based on category to which they belong to.

Some relational tables considered for this application are customer, policy, category, distribution, cust_policy.

Dimensions frequently used in the queries fired by organisation are customer's id, customer's name, gender, birthdate, marital status, city, state, annual income, policy name, policy category, distribution channel, maturity amount etc.

For creating data marts, requirements are gathered and then dimensions in that data mart are determined.

For example, marketing department frequently fire queries to retrieve information regarding the enrolment of policies based on customer's city, state, gender, income level, marital status, policy category, maturity amount, distribution channel etc. Data mart is created having the above required dimensions.

A. *Storing Materialized Queries Using the Identifiers*

Example:
Employee wants to get the average annual income of customers grouped according to gender, policy category, marital status and state.

For the above scenario, there are many different ways the query can be written in SQL. Some sample SQL queries generated for the above requirement are as follows:

QueryString1:

```
SELECT Avg(customer.annual_income) AS Avgincome,
customer.gender, category.cat_name,
customer.marital_status, customer.state
FROM customer, policy, category, cust_policy
WHERE category.cat_id = policy.cat_id AND
            policy.pol_id = cust_policy.po_id AND
            cust_policy.c_id = customer.c_id
GROUP BY customer.gender, category.cat_name,
customer.marital_status, customer.state;
```

QueryString 2 (Using keyword INNER JOIN)

```
SELECT Avg(customer.annual_income) AS Avgincome,
customer.gender, category.cat_name,
customer.marital_status, customer.state
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.gender, category.cat_name,
customer.marital_status, customer.state;
```

QueryString 3 (Changing the order of fields)

```
SELECT customer.gender, category.cat_name,
customer.marital_status, customer.state,
Avg(customer.annual_income) AS Avgincome
FROM customer, policy, category, cust_policy
WHERE category.cat_id = policy.cat_id AND
            policy.pol_id = cust_policy.po_id AND
            cust_policy.c_id = customer.c_id
GROUP BY customer.gender, category.cat_name,
customer.marital_status, customer.state;
```

All the above SQL queries fetch the same result. Hence, they are considered equivalent, but when saved as strings in database and compared through string matching, it will result into a string mismatch.

We have considered different logic for matching queries for equivalence as follows:

B. *Storing query along with results and other metadata considering QueryString1*

For storing queries, predefined identifiers can be assigned to each tables and fields (application specific) and to the functions. These identifiers can then be saved into the tables.

Considering given example, it generates following tables.

Table for Table identifiers can be generated as follows:

Table 2: Table_Identifier

Table name	Table_Id
customer	01
policy	02
category	03
distribution	04
cust_policy	05

Table for Field Identifiers can be generated as follows:

(Identifiers are assigned to fields of table "customer" for illustration. Identifiers are assigned to fields of other tables in a similar manner)

Table 3: Field_Identifier

Field	fld_id
c_id	01
c_name	02
gender	03
dob	04
marital_status	05
addr1	06
addr2	07
pincode	08
city	09
state	10
mob_num	11
org_type	12
annual_income	13

Table for Function Identifiers are defined as below:

Table 4: Function_Identifier

Function	func_id
Sum	01
Avg	02
Min	03
Max	04
Count	05
Stdev	06
Var	07
group by	08
where	09
order by	10
group by and where	89

Inserting details about **QueryString1** tables, fields and functions used are generated as:

Table 5: Tables, fields and functions used in the above discussed query

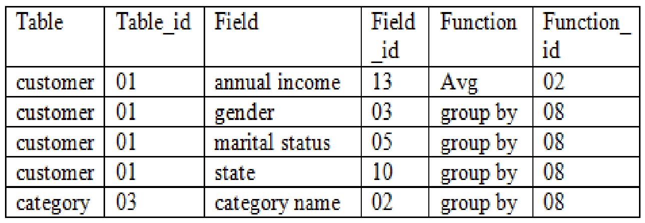

Table	Table_id	Field	Field _id	Function	Function_ id
customer	01	annual income	13	Avg	02
customer	01	gender	03	group by	08
customer	01	marital status	05	group by	08
customer	01	state	10	group by	08
category	03	category name	02	group by	08

Table for storing SQL query in the database for QueryString 1:

Hence, the query stored in the database using identifiers for will be as:

Table 6: Table: "Store_query"

sq_id	query_id	Table_id	Field_id	Function_id
sq1	q1	01	13	02
sq2	q1	01	03	08
sq3	q1	01	05	08
sq4	q1	01	10	08
sq5	q1	03	02	08

Table for storing SQL query results in the database:

Table 7: Table:"Result_metadata"

query_id	query_date	query_freq	query_threshold	path of query_result
q1	1/12/2016	2	8	avg_income.

To understand finding equivalent query from materialzed query database, assume next time the employee again wants to get the average annual income of customers grouped according to gender, policy category, marital status and state. SQL query written is in the form of QueryString 3.

Tables, fields and functions used in the query are extracted. Tables "Table_identifier", "Field_identifier" and "Function_identifier" are referred for assigning identifiers to the tables, fields and functions used in query.

Details about **QueryString 3** tables, fields and functions used are generated as below

Table 8: Tables, Fields and Functions used in QueryString 3

Table	Table_id	Field	Field_id	Function	Function_id
customer	01	gender	03	group by	08
category	03	category name	02	group by	08
customer	01	marital status	05	group by	08
customer	01	state	10	group by	08
customer	01	annual income	13	Avg	02

Table "Store_query" is checked to find if any materialized query has the same table, field and function identifier combination as that of the fired query irrespective of the order of tables stored.

In our example, it is found that query_id "q1" has the same table, fields and function combination. Hence q1 and current fired query can be considered equivalent. The result is retrieved from "Result_metadata". For incremental updates, if required, updated results are generated from data mart. Data loaded after query timestamp value is considered for generating results. New result thus generated, is appended with past result, and saved. Query frequency and query date is update.

Now we need to take care for reducing space load also in database. For this periodic evaluation of database "Result_metadata" is done based on frequency and threshold. This will help to eliminate the infrequent queries from database. Infrequent queries are those queries which have not been fired since long time, i.e. their frequency value is much less than defined threshold value. Removing such queries, helps in reducing the load on the database, making equivalence check and result retrieval faster.

Number of records to be loaded in data mart is subjective to data warehouse refresh and timestamp of the materialized queries. Lowest timestamp value is fetched from updated "Result_metadata" table. Only records loaded post that timestamp value is stored in the data mart.

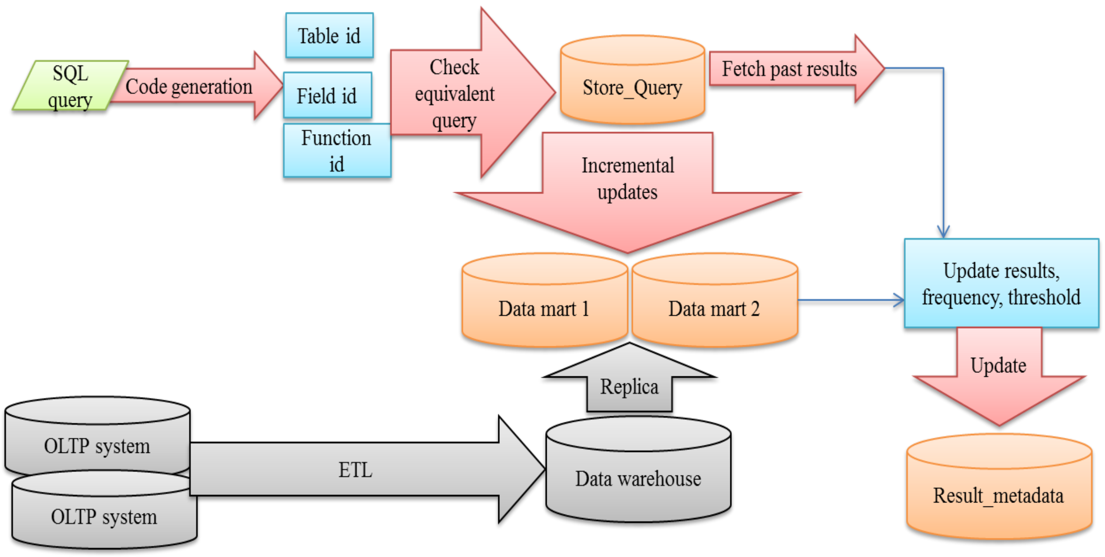

Figure 1. Storing of materialized query and performing incremental updates

V. CONCLUSION

Materialized query is stored only when it is fired by the user. If an equivalent query is found, results are fetched from the materialized query database, in case of no incremental updates. Fetching results from database consumes less time as compared to generating results using warehouse data. In case of incremental updates, updated results are generated using data from data marts. Limited records in data marts make result retrieval faster as traversal through huge records in data warehouse is eliminated. Factors like frequency, threshold, and timestamp helps in eliminating infrequent queries hence saving storage space compared to multidimensional cubes.

REFERENCES

[1] T. Morzy, R. Wrembel, "*On Querying Versions of Multiversion Data Warehouse*," DOLAP'04, November 12–13, 2004, Washington, DC, USA. Copyright 2004 ACM 1-58113-977-2/04/001.

[2] S. Vanichayobon. "*Indexing Techniques for Data Warehouses*' Queries". [Online] Available: http://www.cs.ou.edu/~database/documents/vg99.pdf [Accessed September 15, 2016]

[3] A. Gupta, I. S. Mumick, V.S.Subrahmanian, "*Maintaining Views Incrementally*," Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Pages 157-166.

[4] D. Quass, "*Maintenance Expressions for Views with Aggregation*," Views'96, June 1996, [Online]. Available: http://ilpubs.stanford.edu:8090/183/1/1996-54.pdf.

[5] Y. Zhuge, H. G.Molina, J. Hammer, J. Widom, "*View Maintenance in a Warehousing Environment*," Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, Pages 316-327.

[6] A. Gupta, H.V. Jagadish, I. S. Mumick, "*Data Integration using Self-Maintainable Views*," Advances in Database Technology — EDBT '96, Volume 1057 of the series Lecture Notes in Computer Science, pp 140-144.

[7] K. A. Ross, D. Srivastava, S.Sudarshan, "*Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time*," Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Pages 447-458.

[8] J. Zhou, P. A. Larson, H. G. Elmongui, "*Lazy Maintenance of Materialized Views*," VLDB '07 Proceedings of the 33rd International Conference on Very large Databases, Pages 231-242.

[9] D. Srivastava, S. Dar, H. V . Jagadish, A. Y.Levy, "*Answering Queries with Aggregation Using Views*," Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India, 1996.

[10] D. Agrawal, A. El Abbadi, A. Singh, T. Yurek, "*Efficient View Maintenance at Data Warehouses,*" SIGMOD '97 AZ,USA @ 1997 ACM 0-89791 -911 -419710005.

[11] J. Goldstein, P. A. Larson, "*Optimizing Queries Using Materialized Views: A Practical, Scalable Solution*," Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Pages 331-342, ISBN:1-58113-332-4.

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

[13] P. Vassiliadis, T. Sellis, "*A Survey of Logical Models for OLAP databases*," ACM SIGMOD Record, Volume 28 Issue 4, Dec.1999, Pages 64 – 69.

[14] V. Harinarayan, A. Rajaraman, J. D. Ullman, "*Implementing Data Cubes Efficiently*," Proceedings of the 1996 ACM SIGMOD International Conference on Management of data, Pages 205-216.

[15] A. Datta, H. 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, December 1999, Pages 289-301.

[16] R. Agrawal, A. Gupta, S. Sarawagi, "*Modeling Multidimensional Databases*," Proceedings 13th International Conference on Data Engineering, pages232-243.

[17] P. Deshpande, S. Agarwal, J. Naughton, R. Ramakrishnan, "*Computation of Multidimensional Aggregates*," Proceedings 22nd VLDB Conference.

[18] S. J. Chun, C. W. Chung, J. H. Lee, S. L. Lee, "*Dynamic Update Cube for Range-Sum Queries*," Proceedings of the 27th VLDB Conference.

[19] J. Shanmugasundaram, U. Fayyad, 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, Pages 223-232.

[20] C. Li , X. S. Wang, "*A Data Model for Supporting On-Line Analytical Processing*," Proceedings of the fifth international conference on Information and knowledge management, Pages 81-88.

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

[22] S. Chakraborty, J. Doshi, "*Faster Query Result Retrieval Approaches from a Data Warehouse: A Survey*," "International Journal of Current Engineering and Scientific Research (IJCESR)", Volume 4, Issue 6, 2017, ISSN (PRINT): 2393-8374, (ONLINE): 2394-0697, Pages 7-14.

[23] F. Sultan, A. Aziz, "*Ideal Strategy to Improve Data warehouse Performance*," International Journal on Computer Science and Engineering Vol. 02, No. 02, 2010, 409-415.

Authors Profile

Sonali Chakraborty is an Assistant Professor for MSc (CA & IT) at Gujarat University, Ahmedabad, India. She has completed her MSc (CA & IT) from Gujarat University, Ahmedabad, India in 2007. She has 8+ years of experience in the field of teaching. Her subjects of interest include Data Warehousing and Data Mining, Computer Graphics, Digital Image Processing, E-commerce and E-governance. She is pursuing PhD in the area of Data Warehousing and Data Mining from GLS (Gujarat Law Society) University, Ahmedabad, India. She has published three research papers in International Journal.

Dr. Jyotika Doshi is an Associate Professor for MCA at Faculty of Computer Technology, GLS University, Ahmedabad, India. She earned her PhD in computer science from Gujarat University, Ahmedabad; MCA from IGNOU, Delhi; MSc(Statistics) from M. S. University, Vadodara. She has 35+ years of experience in the academic field and 3 years experience in software development industry. Her research is in the area of Data compression. Her subjects of interest are Data structures, Database management, Data analysis, Parallel programming. She has published nearly 15 research papers in International Journals.