IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-5, May 2018 E-ISSN: 2347-2693

Deriving Aggregate Results with Incremental Data using Materialized Queries

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

1*Gujarat University, Ahmedabad, Gujarat
2 GLS University, Ahmedabad, Gujarat

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

Available online at: www.ijcseonline.org

18/May/2018, Published: 31/May/2018

***Abstract*—** OLAP queries perform analytical processing on enterprise warehouse data. These queries are implemented using aggregate as well as non-aggregate functions. Result extraction using OLAP queries involves traversal through huge number of warehouse records. For repeated queries, processing time can be saved by storing queries along with its result and other parameters like timestamp, frequency, threshold in relational database MQDB. With periodic data warehouse refresh, incremental results for the frequent queries are processed using data marts and results are combined with existing results. This paper depicts the methodology to derive results based on different aggregate functions giving the effect of incremental data. Some aggregate functions may require other measures to be stored for compiling results.

***Keywords*—** Data warehouse, Materialized queries, Aggregate functions, Deriving incremental results

I. INTRODUCTION

For decision making in an enterprise, management performs analytical processing on large amount of warehouse data. Data analysis is done by OLAP (Online Analytical Processing) queries using various aggregate functions such as average/ mean, sum, count, minimum, maximum, variance, standard deviation. Generating results using data warehouse is relatively time consuming as traversal through huge number of records is done. For frequent OLAP queries, query execution time can be reduced by storing queries along with its results and metadata information such as timestamp, frequency, threshold etc. [1][2][3]. When same query is fired next time, it fetches results from MQDB in case of no incremental updates [2]. This results into significant reduction in query result retrieval time. In case of incremental updates, only incremental records from data warehouse are processed and existing results are combined with incremental results [3]. Authors have suggested using data marts [4] to store incremental data and process query incremental updates.

When query involves aggregate functions, it requires some processing to compute final result using existing and incremental results. The method of compiling aggregate results varies with the nature of function. Some functions need additional measures to be used in computing combined results. This paper depicts the process of deriving combined results using existing and incremental results for queries involving aggregate functions.

The paper is organized as follows: Section II deals with related literature. Suggested methodology for deriving results is explained in Section III while implementation of the approach is illustrated with examples in section IV. Section V concludes the research work.

II. RELATED LITERATURE

Dimitri Theodoratos and Timos Sellis [5] state that high query performance and low view maintenance cost are in conflict with each other. High query performance can be obtained by storing in the data warehouse the results of all the queries of interest. Here, maintenance cost of materialized queries might be high. Authors suggested that by materializing appropriately selected set of views in the data warehouse, the total query processing cost and the view maintenance cost can be kept at an acceptable level. The authors discuss as how to select such set of views where the solution is a negotiation between fully materializing all queries of interest and keeping replicas of all base data needed for answering the queries on the other hand. They formulated the problem by determining set of views for a given set of queries of interest against the data warehouse such that all queries can be answered using this set of view and the operational cost is minimal. The problem is modelled using a state space search algorithm after representing the views using multiquery graphs and with assumption that there are no space restrictions in data warehouse.

Author P.Karthik et. al [6] discuss ways of tuning an SQL query so that the time consumed by the query during runtime is decreased. The optimizer predicts the cost of using alternative access method used for resolving a particular query using statistics on tables and indexes and finds the best query plan in terms of I/O cost. The authors highlighted certain rules for query tuning used in their project such as rewriting the query using UNION instead of OR, replacing relational operators using BETWEEN, using formulas without attributes, avoiding join if not necessary, avoiding DISTINCT keyword, same nested query and temporary relations if not necessary and breaking a long query into parts. Authors implemented the rules assuming non-parallel or non-distributed database environment where database is not geographically separated.

Authors Patrick O'Neil and Dallan Quass [7] observed that OLAP queries with aggregates and grouping can be evaluated using indexing and clustering. They introduced a third index called Group set indexes using Bit-Sliced indexing and Projection indexing.

Ziyu Lin et. al [8] discuss about the query contention and scalability issue which deploys real time data warehouse solutions. The contention between SELECT queries and multiple inserts causes limitations to the scalability of the data warehouse. The authors deal with this issue using multi-level cache and depicted architecture called “dynamic multi-level caches”. For any query arriving the system, it is redirected to the corresponding cache for data access depending on its requirements. Though query load is distributed across multi-level cache instead of blocking one cache it has to be updated with different length cycles between real time and 24 hours.

Surajit Chaudhuri [9] quoted that the two key components of query evaluation component are query optimizer and query execution engine. He states that execution of query optimizer is critical since throughput for execution plans may vary. To solve query optimization search problem, a search space, cost estimation technique and an enumeration algorithm for searching through execution space must be provided. He discusses the set of algebraic transformations to preserve the equivalence in an optimizer namely: commuting between operators, reduction of multi-block queries to single-block, using semi-join techniques for optimizing multi-block queries. He discussed about the statistics and cost estimation for each plan in search space by using statistical summaries of data but restricted the consideration for memory resource. Also, this optimizer technology is not discussed for Object oriented systems and database systems using multimedia and web context for fuzzy queries and decision support systems.

Prasan Roy et. al. [10] highlighted that many times there are lot of common sub expressions in complex queries. Authors addressed this problem of optimizing queries with common subexpressions referred as multi-query optimization which is based on AND-OR DAG query representation. Greedy strategy picks the subexpression iteratively giving maximum benefit i.e. reduction in cost if the subexpression is materialized and reused. Their algorithm is restricted for only single query with intra-query common subexpressions. They have not considered multi query optimization of nested queries as well as parameterized queries having different parameter values.

Ashish Gupta et. al [11] introduce generalized projections (GPs), an extension for eliminating duplicate projections. The approach extends algorithms for SQL queries using distinct projections to derive algorithms for queries using aggregations like sum, min, max, avg and count. They addressed a problem in data warehousing as how to answer an aggregate query using materialized aggregate views on base tables. Authors derived a transformation rule by uniting with previous proposed transformation rules. The new rules includes coalescing of multiple aggregate computations into single computations, using arithmetic inequality selection conditions introducing and eliminating aggregate computations and pushing down aggregate computations of a join. In this approach authors did not consider correlated subqueries. Also they have considered aggregates where the aggregate value can be computed from aggregates over subsets. Example: average is calculated in terms of sum and count.

Sara Cohen et. al [12] commented that evaluating a view and then rewriting them will yield the same result as evaluating the original query. Their proposed approach is based on syntactic characterizations of the equivalence of disjunctive aggregate queries. For a specific operator, several types of queries using views as candidates for rewritings have been introduced. Then the candidate is unfolded by replacing each occurrence of a view predicates along with its definition hence, obtaining a regular aggregate query. The candidate will have more complex operator than the specified operator. Authors considered unnested queries or views with union, or using operators like min, max, count and sum. Their approach is limited to unnested queries and views are with union, and those employing operators like min, max, count and sum.

Jonathan Goldstein and Per-Ake Larson [13] commented that materialized views involving aggregate functions can improve query processing time. They presented a fast and scalable algorithm limited to only SPJG views to determine if a part or all of a query can be computed from the materialized views. A transformation based optimizer generates all possible rewritings of a query expression, then estimating their costs, and chooses the one with the lowest cost. Rewritten query expressions are generated by applying local transformation rules on subexpressions of the query. By applying a rule substitute expression is produced equivalent to original expression. One such transformation rule is view matching i.e. computing a subexpression from materialized views.

III. SUGGESTED METHODOLOGY

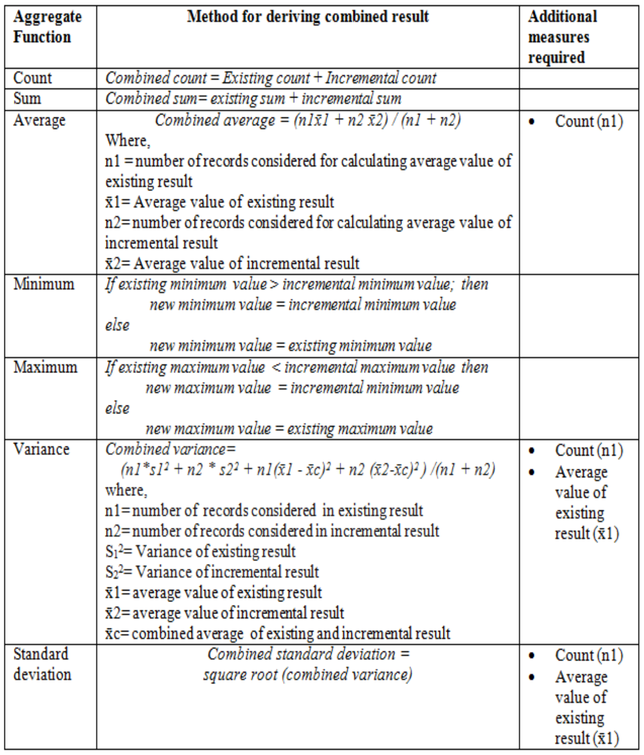
Executed OLAP queries are stored in relational database MQDB (Materialized Query Database) along with other parameters like results, timestamp, frequency, threshold and number of records [1][2] [3].

For an equivalent query, incremental updates, if required, are processed using data marts. Existing results are combined with incremental results to generate updated results. Result file and query metadata is updated.

OLAP queries might implement aggregate functions. The behaviour of aggregate functions while compiling existing and incremental results varies with aggregate functions. Also, to regenerate aggregate measure using existing results and incremental results, one may require storing other measures as per need.

The methods using which aggregate results can be computed are shown in Table 1.

Table 1 Methods for computing aggregate results



Aggregate Function	Method for deriving combined result	Additional measures required
Count	*Combined count = Existing count + Incremental count*	
Sum	*Combined sum= existing sum + incremental sum*	
Average	*Combined average = $(n1\bar{x}1 + n2\ \bar{x}2) / (n1 + n2)$* Where, n1 =number of records considered for calculating average value of existing result $\bar{x}1$= Average value of existing result n2=number of records considered for calculating average value of incremental result $\bar{x}2$= Average value of incremental result	• Count (n1)
Minimum	*If existing minimum value > incremental minimum value; then* *new minimum value = incremental minimum value* *else* *new minimum value = existing minimum value*	
Maximum	*If existing maximum value < incremental maximum value then* *new maximum value = incremental minimum value* *else* *new maximum value = existing maximum value*	
Variance	*Combined variance=* $(n1{*}s1^2 + n2 * s2^2 + n1(\bar{x}1 - \bar{x}c)^2 + n2\ (\bar{x}2-\bar{x}c)^2\ ) /(n1 + n2)$ where, n1=number of records considered in existing result n2=number of records considered in incremental result S_1^2= Variance of existing result S_2^2= Variance of incremental result $\bar{x}1$= average value of existing result $\bar{x}2$= average value of incremental result $\bar{x}c$= combined average of existing and incremental result	• Count (n1) • Average value of existing result ($\bar{x}1$)
Standard deviation	*Combined standard deviation =* *square root (combined variance)*	• Count (n1) • Average value of existing result ($\bar{x}1$)

IV. IMPLEMENTATION

To understand the method of deriving aggregate results using incremental results, we consider an example of an organization providing education facilities all over India. Data for the example considered here is collected from http://censusindia.gov.in.

Consider the following instances of OLAP query with reference to the collected data.

Query 1: Display total number of graduate females for each state.

```
select dw_states.st_name, sum(dw_zones.graduate_f)
from dw_zones, dw_states
where dw_zones.st_code = dw_states.st_code
group by dw_zones.st_code
```

Query 2: List average number of males and females pursuing technical diploma course for different age groups.

```
select dw_age.age_value , avg (dw_zones.f_diploma), avg
(dw_zones.m_diploma) from dw_zones, dw_age
where dw_zones.age_id = dw_age.age_id
group by age_id
```

Query 3: Display the town and the state to which it belongs having minimum number of literate males and females.

```
select dw_states.st_name , min(dw_zones.literate_m),
min(dw_zones.literate_f)
from dw_zones,  dw_states
where dw_zones.st_code = dw_states.st_code
group by dw_zones.st_code
```

Query 4: Find maximum number of illiterate and below primary level males and females for each state.

```
select dw_states.st_name,  max (dw_zones.m_illiterate), max
(dw_zones.f_illiterate),
            max (dw_zone.m_belprimary), max
            (dw_zones.f_belprimary)
from dw_zones, dw_states
where dw_zones.st_code= dw_states.st_code
group by dw_zones.st_code;
```

Query 5: Count the number of towns considered for analysis for each state

```
select dw_states.st_name, count(dw_town.town_name)
from dw_town, dw_states
where dwt_town.st_code=dw_states.st_code
group by dw_town.st_code
```

Query 6: Find the deviation in number of females pursuing primary education for each state

```
select dw_states.st_name, stddev(dw_zones.primary_f)
from dw_zones, dw_states
where dw_zones.st_code = dw_states.st_code
group by dw_zones.st_code
```

Query 7: Find the variance in number of below primary education for males in each town.

```
select dw_town.town_name, var(dw_zones.belprimary_m)
from dw_zones, dw_town
where dw_zones.town_code=dw_town.town_code
group by dw_zones.town_code
```

A. Initialization

This phase generates identifiers for the tables, fields and aggregate functions. It is executed during application load time. Identifiers for tables and fields are application/ domain specific. Generation of identifiers are depicted in detail in [1] [2].

For the application discussed here, the identifiers defined are as follows:

i. **Table identifiers** (table_name, table_id): (dw_states, 01), (dw_town, 02), (dw_age, 03), (dw_zones, 04)

ii. **Field identifiers** For illustration we depict field identifier generation for table dw_states.
(field_name, field_id): (st_code, 01), (st_name, 02), (entry_date, 03)

iii. **Function identifiers** (func_name, func_id): (sum, 01), (avg, 02), (min, 03), (max, 04), (count, 05), (stddev, 06), (var, 07), (group by, 08)

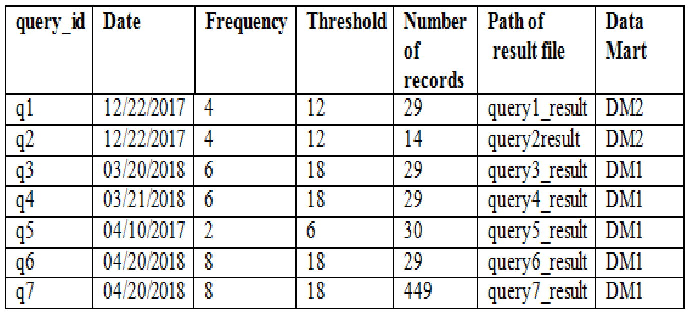
B. Storing queries

Using the assigned identifiers as discussed in Initialization, the queries are stored in "Stored_query" table of MQDB as depicted in Table 2. Corresponding metadata information is stored in "Materialized_query" table of MQDB shown in Table 3.

Table 2 "Stored_query" table of MQDB [1][2][3]

sq_id	query_id	Table_id	Field_id	Func_id
sq1	q1	04	22	01
sq2	q1	01	02	00
sq3	q1	04	02	08
sq4	q2	03	02	00
sq5	q2	04	20	02
sq6	q2	04	19	02
sq7	q2	04	04	08
sq8	q3	04	07	03
sq9	q3	04	08	03
sq10	q3	01	02	00
sq11	q3	04	02	08
sq12	q4	04	05	04
sq13	q4	04	06	04
sq14	q4	04	09	04
sq15	q4	04	10	04
sq16	q4	01	02	00
sq17	q4	04	02	08
sq18	q5	02	03	05
sq19	q5	01	02	00
sq20	q5	02	02	08
sq21	q6	04	12	06
sq22	q6	01	02	00
sq23	q6	04	02	08
sq24	q7	04	09	07
sq25	q7	02	03	00
sq26	q7	04	03	08

Table 3 "Materialized_query" table of MQDB [1][2]



query_id	Date	Frequency	Threshold	Number of records	Path of result file	Data Mart
q1	12/22/2017	4	12	29	query1_result	DM2
q2	12/22/2017	4	12	14	query2result	DM2
q3	03/20/2018	6	18	29	query3_result	DM1
q4	03/21/2018	6	18	29	query4_result	DM1
q5	04/10/2017	2	6	30	query5_result	DM1
q6	04/20/2018	8	18	29	query6_result	DM1
q7	04/20/2018	8	18	449	query7_result	DM1

C. Processing equivalent queries with aggregate functions

When a query is fired, it is first searched in "Stored_query" table of MQDB, for its equivalent query. Process to determine equivalence between two queries is illustrated by the authors in [1][2][3]. For processing incremental updates of the query, data mart [4] is used. Existing results are combined with incremental results to generate updated results. Methods for combining existing results with incremental results for queries involving aggregate functions varies with the function as described in Table 1.

We illustrate deriving aggregate results using existing and incremental results for Query 2 and Query 7.

Example 1: Compiling Average value for Query 2
Average value stored as existing result for **Males and Females pursuing diploma belonging to age_id = 'g1'** is **664.42** and **681.40** respectively.
Incremental average value calculated using data mart for the same criteria is **993.93** and **1112.83** respectively.
Hence combined average calculated using method discussed in Table 1 is shown in Table 4.

Table 4 Combined average derived for Query 2

Gender	n1	x̄1	n2	x̄2	Combined average
Males	722	664.42	43	993.93	682.94
Females	722	681.40	43	1112.83	705.65

Here, we need to stored **n1** for each record in result file for calculating combined average value. Result file for the query with additional attribute will be as shown in Table 5.

Table 5 Updated result file for Query 2 with additional attribute (n)

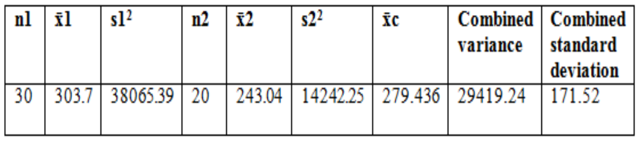
Gender	Average number pursuing diploma	Number of records (n)
Males	682.94	765
Females	705.65	765

Where,
Number of records (n) = total count of records considered for calculating combined average value (n1+ n2)

Example 2: Compiling Variance for Query 7
Variance calculated **38065.39** for town name **'Jaipur'** considering **30** entries is stored as existing result. Incremental variance value calculated considering **20** new

entries for the same town from data mart is **14242.25**. Hence, deriving combined variance is shown in Table 6.

Table 6 Combined variance for Query 7



n1	x̄1	s1²	n2	x̄2	s2²	x̄c	Combined variance	Combined standard deviation
30	303.7	38065.39	20	243.04	14242.25	279.436	29419.24	171.52

Hence, for deriving combined results for aggregate functions, majorly for average, standard deviation and variance; additional measures are required to be stored. Count of records considered for calculating average value and average value for calculating combined average in case of variance or standard deviation needs to be stored in result file. In this case, query result file with additional attributes is shown in Table 7.

Table 7 Updated result file for Query 7 with additional attribute (n, x̄1)

Town name	**Variance for below primary male**	**Number of records (n)**	**Average value (x̄1)**
Jaipur	29419.24	50	279.436

Where,
Number of records (n) = total count of records considered for calculating average value (n1+ n2)
Average value (x̄1) = Combined average calculated (x̄c)
(x̄1 is updated with new combined average every time combined variance or combined standard deviation is calculated)

V. Conclusion

Storing queries and then compiling existing and incremental results, eliminates the need to traverse through huge number of records in data warehouse. This significantly reduces query execution time for frequent OLAP queries. For deriving combined results related to aggregate functions especially in case of calculating average, standard deviation and variance, storing additional measures like count and average value in result file is required

References

[1] S. Chakraborty and J. Doshi, "*Data Retrieval from Data Warehouse Using Materialized Query Database*," International Journal of Computer Sciences and Engineering, Vol.6(1), Jan 2018, E-ISSN: 2347-2693, pages 280-284.

[2] S. Chakraborty and J. Doshi, "*Performance Evaluation of Materialized Query*," International Journal of Emerging Technology and Advanced Engineering, vol. 8, Issue 1, pages 243-249, January 2018.

[3] S. Chakraborty and J. Doshi, "*Materialized Queries with Incremental Updates*," 3rd International Conference on Information and Communication Technology for Intelligent Systems, Springer Smart Innovation, Systems and Technologies (SIST). Series: http://www.springer.com/series/8767. [Presented, Ahmedabad, 6-7th April, In Press].

[4] S. Chakraborty and J. Doshi, "*An Approach for Creating and Maintaining Dependent Data Marts using Materialized Queries' Information,*" International Journal of Scientific Research in Science, Engineering and Technology, vol 4, Issue 1, pages 1527-1533, JanuaryFebruary,2018.

[5] D Theodoratos, T Sellis, "*Data Warehouse Configuration,*"Proceedings of the 23rd VLDB Conference Athens, Greece, 1997.

[6] P. Karthik, G.Thippa Reddy, E.Kaari Vanan, "*Tuning the SQL Query in order to Reduce Time Consumption*," IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 4, No 3, July 2012, ISSN (Online): 1694-0814.

[7] P O'Neil, D Quass, "*Improved Query Performance with Variant Indexes,*" Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Pages 38-49.

[8] Z Lin, D Yang, G Song, T Wang, "*Dealing with Query Contention Issue in Real-time Data Warehouses by Dynamic Multi-level Caches,*" Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on Computer and Information Technology.

[9] S Chaudhuri, "*An Overview of Query Optimization in Relational Systems*," PODS '98 Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Pages 34-43.

[10] P Roy, S. Seshadri, S. Sudarshan, S Bhobe, "*Efficient and Extensible Algorithms for Multi Query Optimization*," Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, Pages 249-260.

[11] A Gupta, V Harinarayan, D Quass, "*Aggregate-Query Processing in Data Warehousing Environments*," Proceedings of the 21st VLDB Conference, Zurich, Swizerland, 1995.

[12] S Cohen, W Nutt, A Serebrenik, "*Rewriting Aggregate Queries Using Views*," Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Pages 155-166.

[13] 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.

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 from GLS (Gujarat Law Society) University, Ahmedabad, India. She has published six research papers in International Journals.

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.