International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:9, 2017

Materialized View Effect on Query Performance

Yusuf Ziya Ayık, Ferhat Kahveci

***Abstract*—**Currently, database management systems have various tools such as backup and maintenance, and also provide statistical information such as resource usage and security. In terms of query performance, this paper covers query optimization, views, indexed tables, pre-computation materialized view, query performance analysis in which query plan alternatives can be created and the least costly one selected to optimize a query. Indexes and views can be created for related table columns. The literature review of this study showed that, in the course of time, despite the growing capabilities of the database management system, only database administrators are aware of the need for dealing with archival and transactional data types differently. These data may be constantly changing data used in everyday life, and also may be from the completed questionnaire whose data input was completed. For both types of data, the database uses its capabilities; but as shown in the findings section, instead of repeating similar heavy calculations which are carrying out same results with the same query over a survey results, using materialized view results can be in a more simple way. In this study, this performance difference was observed quantitatively considering the cost of the query.

***Keywords*—**Materialized view, pre-computation, query cost, query performance.

I. Introduction

TIME is precious and users do not like waiting, and as such computer programs are expected to be faster; especially in client/server-based programming, where this fact is even more important. In the case of database overload, many users may have to wait longer. The operations on very large data with complex queries have various challenges.

Queries run very fast in database management systems. Although many of them include sorting, calculating, and merging operations, they complete the process in less than a second. Various optimization techniques are applied to improve the performance of constant and frequently used queries over multiple tables. These query optimization techniques are usually arranged to get the best query performance. Optimized and frequently used queries can be stored in databases as views. In addition, some queries that require heavy and high capacity operations are pre-calculated. In this way, obtained results are saved at specific time intervals to the materialized view. Thus, query performance and saving time are achieved.

II. Conceptual Framework

A. Data Query Language

The users/applications interact with the database through the data query language. The data query language is a special-purpose programming language for database. The structure of the database is dominant factor in determining the query language, which is used to access data in the database [1]. SQL (Structured Query Language) is one of the most common structured query languages that performs a number of operations, such as read, update, add new data, delete data, etc. in a database, and it has three components: Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL) [2].

B. SQL Query Optimization

Despite the use of high speed servers supported by powerful hardware, the response time is sometimes extended due to a query. Query optimization is used to overcome this problem. Query optimization is an important issue for SQL developers and database administrators (DBAs). In query optimization, the data access techniques must be well-defined. In addition, determining proper rules, semantic transformation, and choosing the optimal query techniques should be well-understood [3]. Different techniques are used to optimize queries. Generally, alternative query plans with the same result are created and compared for query performance in database systems. There are many rules for alternative query formation. However, these rules always may not give good results. Rules that give good results for some queries may not give very good results for other queries [4].

C. Views

Views are a type of query which is used to form a virtual table displaying the data from one or more tables. Although data entry is possible, data cannot be recorded; that is why they are called virtual tables. The desired columns of the real tables are represented, so that they are advantageous in terms of security as they allow users to access certain parts of the tables [5].

D. Indexed Tables

Indexed tables allow access to the desired entry directly, instead of scanning the entire table in order to access the corresponding entry. An index table is created apart from the data table. A created index of a table contains the relevant table keys and the addresses on a disk [6]. Thus, queries over indexed tables have less access time on the disk.

E. Pre-Computation

Pre-computation is the action of generating a result table to avoid the same calculations being repeated by performing pre-calculations before the queries are executed. It is usually used for complex structured queries to compute results. A simple example; instead of calculating the value of π in queries, the pre-calculated value can be used. The materialized view term is used for the process of keeping the pre-calculation results in

Yusuf Ziya Ayik is with the Atatürk University, Turkey (e-mail: ziyaayik@atauni.edu.tr).

database systems [7].

F. Materialized View

The difference between materialized view and the views is the physical space occupancy. The materialized views hold the query results at the moment that they were created. Whereas the views are storing a SQL query, materialized views are storing both the SQL query and the query results. Therefore, the query results of the materialized views must be updated periodically [8], [9] (Fig. 1).

Fig. 1 Updating materialized view results

G. Query Performance Analysis

Database management systems have specific query performance analysis tools. In this study, the most used tools of database management systems were researched. The top three database systems were Oracle, MySQL and Microsoft SQL Server based on the Db-Engines Web site monthly database popularity ranking on October 2016 [10].

Oracle, Microsoft SQL Server and MySQL use Cost-Based Optimizer (CBO). The query optimization component of the database system forms query execution plans. In this way, each alternative query plan returns cost information based on input/output, processor, memory, and resource consumption, also the number of rows, and the size of the initial dataset (Fig. 2). At the end, the least costly query plan is chosen by the optimizer [11]-[13].

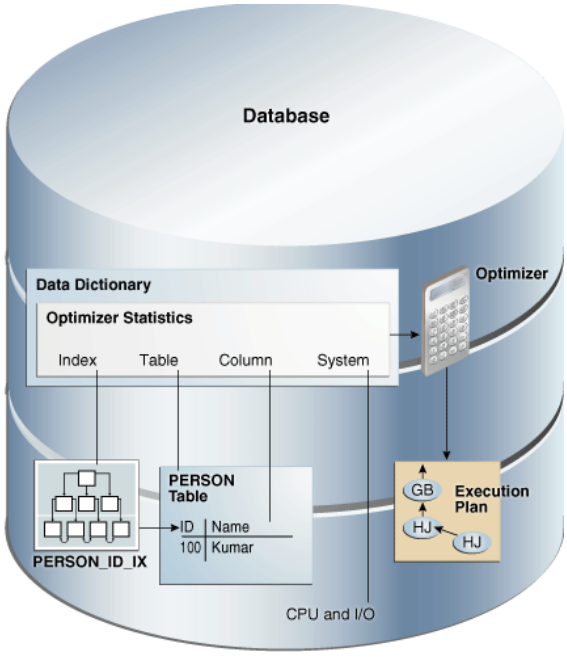

Fig. 2 The operation of the database optimization component [14]

III. Method

A. Purpose of Research

The purpose of this research is to point out concepts about query performance and experimentally observe the cost of the SQL query that gives the same result in cases of optimized, index, views, and materialized view usage.

B. Research Model

In the research, most frequently used basic concepts that speed up the queries in database management systems were researched and defined. These are query optimization, indexing, use of views and materialized views in general. Query cost differences were evaluated based on the "cost" variant of Oracle database management system`s cost-based optimization [15].

C. Data Collection

The cost data obtained by the PL/SQL Developer program's Explain Plan Window was saved according to the cases in which the SQL query optimizing, index, views, and materialized view usage. These data were arranged in tabular form to create a graphic and the cost of the questionnaire was evaluated in different cases.

D. Findings

Tested queries in the study were given in this section together with screen outputs and results. The test query that finds the average of the first question within the survey results was called "original".

Original: select avg(value) from test_survey having question_no=1 group by question_no

When the query was checked, it was seen that "having" was used. "Having" filters the results after all records were scanned in the table. For this reason, the use of "having" in "select" queries should be avoided [3]. In Fig. 3, the cost value of the "original" query was calculated in the "cost" column of the Explain Plan Window of the PL / SQL Developer program. For the "optimized" query, only related rows were scanned by using "where" instead of "having". In this case the optimized original query is as follows:

Optimized: select avg(value) from test_survey where question_no=1 group by question_no

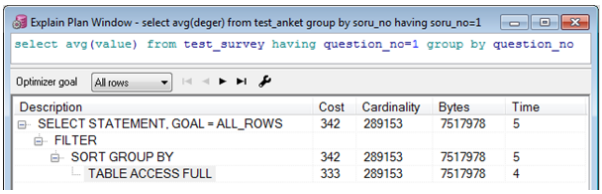

Fig. 3 Cost value in the query Explain Plan Window

Firstly, in the case of without view; non-indexed and indexed, and then in the case of with view; non-indexed, indexed and materialized costs of the original query and optimized query were recorded (Table I).

TABLE I
THE COSTS OF QUERIES IN DIFFERENT CASES

	Without View		With View		Materialized View
	Non-indexed	*Indexed*	*Non-indexed*	*Indexed*	
Original	341	147	341	147	2
Optimized	334	48	334	48	2

The index, named as "index_test_survey" has been created for the cost calculation of the indexed table case. The index operation was performed on the "question_no" and "value" columns of the "test_survey" table, because the original query computes the average value of a given question. The query for this operation is as follows:

Query 1. "create index index_test_survey on TEST_SURVEY (question_no, value);"

In case of using the view for cost calculation, "view_test_survey" view was created with the following query.

Query 2. "create view view_test_survey as (select * from test_survey);"

In case of using materialized view for cost calculation, "mt_test_survey" was created with the following query.

Query 3. "create materialized view mt_test_survey as "select avg(value) from test_survey having question_no=1 group by question_no;"

The materialized view "mt_test_survey" keeps both the original query sequence and the result, and also update parameters can be assigned. As seen in Query 4, just the materialized view table was queried to obtain the result of the original query. Instead of performing any heavy calculations through the query, the pre-calculated result was listed, which significantly reduced the query cost.

Query 4. "select * from mt_test_survey;"

IV. CONCLUSION AND DISCUSSION

Table I shows the values of query costs in different cases. There were 10 different costing cases for original and optimized queries: Without view/non-indexed, without view/indexed, with view/non-indexed, with view/indexed and materialized view cases. These cost values were graphically shown in Fig. 4.

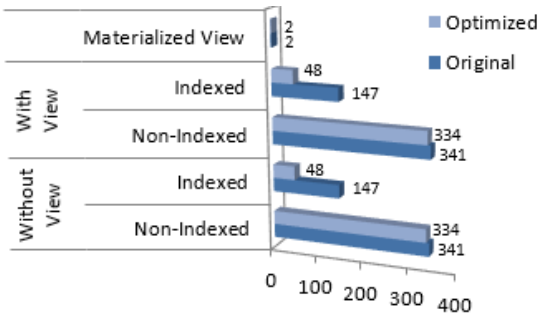

Fig. 4 The cost of query according to different cases

The cost of the original query was higher than the cost of the optimized query in all cases except the materialized view. This difference became clearer by using indexed tables. Optimization of the query improved the query performance, because cost reduction was achieved through optimization of original query.

Regarding the use of indexes, there was a decrease in query cost relative to the non-indexed case both for optimized and the original query. The decline in the indexed use of the optimized query was more than the indexed use of the original query. In other words, both optimizing query and indexing table significantly reduced the cost.

Regarding the costs associated with the use of the view, the indexed/non-indexed costs were same both for with view and without view cases. For this reason, there was no effect of using the view on the query performance. As noted at “Views” subtitle of the article, views provide some advantages, such as security and ease of use, because they just keep the pre-defined query sequence.

Finally, regarding the cost of the materialized view in Fig. 4, it was obvious that there was a cost difference relative to other cases. Given the materialized view, it had very little cost for both the original query and the optimized query. This small amount of cost indicates the best performance through very little consumption of memory, processor and input/output units. In terms of the materialized view principles, use of optimized query will affect the performance only during the update process of the results.

Many techniques and tools for query optimization and indexing are generally used to improve query performance. Besides, today's database management systems have the ability to improve query performance through the query optimizer tool. Furthermore, the query can be slightly improved by the user through optimizing or indexing efforts. However, the performance improvement provided by the materialized view performance is farther than those achievements. For this reason, the materialized view is a concept which should not be ignored during query performance improvement. It is predicted that the use of the materialized view, for tables those data entries are in certain periods or data entry is completed instead of tables with the transactional data, will provide a high performance improvement in terms of the general aspect of the database management system.

REFERENCES

[1] MEB, Veritabanında Sorgular, Ankara, Turquia: Ministry of Education, 2012.
[2] IEEE, SWEBOK version 3.0, Piscataway, New Jersey, USA: IEEE Computer Society Products and Services, 2014.
[3] N. Kumari, "SQL server query optimization techniques," *International Journal of Scientific and Research Publications,* vol. 2, no. 6, pp. 1-4, 2012.
[4] A. Ö. Uysal, "Veritabanı sistemlerinde sorgu optimizasyonlarının veri analiz teknikleriyle geliştirilmesi," *Yıldız Technical University, Graduate School of Natural and Applied Sciences,* 2011.
[5] MEB, Veritabanı Yönetimsel Fonksiyonları, Ankara, Turquia: Ministry of Education, 2013.
[6] Y. Özkan, Veri madenciliği yöntemleri, Istanbul, Turquia: Papatya Yayıncılık Eğitim, 2013.
[7] J. Han, M. Kamber and J. Pei, Data Mining Concepts and Techniques 3rd Edition, Waltham, Massachusetts, USA: Elsevier Inc., 2012.
[8] C. J. Date, The Relational Database Dictionary: A Comprehensive

Glossary of Relational Terms and Concepts, with Illustrative Examples, Sebastopol, California, USA: O'Reilly Media, 2006.
[9] L. Ergüder, "Materialized View," 25 06 2013. (Online). Available: http://www.injavawetrust.com/oracle-ders-40-views-05-materialized-view/. (Accessed 11 10 2016).
[10] DB-Engines, "DB-Engines Ranking of Relational DBMS," 2016. (Online). Available: http://db-engines.com/en/ranking/relational+dbms. (Accessed 14 10 2016).
[11] Oracle, "Optimizer Statistics Concepts," 2016. (Online). Available: https://docs.oracle.com/database/121/TGSQL/tgsql_statscon.htm#TGSQL351. (Accessed 14 10 2016).
[12] M. Pilecki, "Optimizing SQL Server Query Performance," *TechNet Magazine*, 2007.
[13] MySQL, "Understanding the Query Execution Plan," 2016. (Online). Available: http://dev.mysql.com/doc/refman/5.7/en/execution-plan-information.html. (Accessed 14 10 2016).
[14] Oracle, "Query Optimizer Concepts," 2016. (Online). Available: https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm#TGSQL192. (Accessed 14 10 2016).
[15] K. Yagoub and P. Gongloor, "SQL Performance Analyzer," Oracle Corporation, Redwood Shores, California, USA, 2007.