Comparative Analysis of Query Optimization Techniques in Database Systems

[1]Pooja Wankhade, [2]Dr. Vaishali Deshmukh

[1]ME Student, [2]Professor
[1]Computer Science and Engineering
[1]PRMITR Badnera, Maharashtra, India

***Abstract*: Due to huge growing data, performance of database systems is now critical for every organization. There exist several bottlenecks affecting this performance like memory, processor, network issues, etc. Out of these, query optimization exhibits major role in database optimization and contributes to almost 70% of database tuning [1]. Queries can be written in many different ways to achieve same set of results. Query optimization refers to various query rewriting techniques to achieve results faster. There is a need to rewrite non-optimal queries to get the same correct result in minimum possible time.**

This paper discusses about some of the query optimization techniques that can be used to achieve results from database in lesser time. In this paper, we have compared different query optimization techniques in terms of execution time. These optimized strategies yields the same result from database in minimum possible time.

***Index Terms*: Query optimization, database tuning, relational database, bulk-insert, execution time**

I. INTRODUCTION

Query optimization plays a very important role in managing application dealing with large scale databases. A request for information to a database is submitted in form of a query. A query plan is a series of steps used to perform some operations on database to achieve the intended result. There may be different ways to write the query to get similar result from database. Each method may take different amount of time to execute. The execution time may vary largely depending on the way used. With non-optimized queries, database systems might handle smaller result set within reasonable time but they can encounter performance degradation while dealing with large result sets. Queries scripted in non-optimal ways can be improved to perform better. It is possible to eliminate about 70% of database performance issues using query optimization [1].

This paper is organized as follows. Section I gives the introduction of the topic. Literature survey in section II summarizes the work related to the query optimization area. Section III discusses some of the query optimization techniques and their comparison with respect to their execution time. Section IV concludes the paper.

II. LITERATURE SURVEY

Vamsi Krishna Myalapalli, ASN Chakravarthy, "Revamping SQL Queries for Cost Based Optimization[1]" proposes various re-writing techniques for ensuring cost based optimization of queries. They say that their paper would serve as a tuning / rewriting tool to overhaul query practice and processing(s) on production databases and pragmatic results of their methodologies and explorations stronglyadvocate that operational costs and query performance and operational costs are enhanced.

This paper explains sundry novel and sophisticated query rewriting methodologies that have been implemented to tune the real time queries. This paper explained several query rewriting techniques toensure cost based optimization, as this kind of tuning is architecture and platform independent and all the rewriting methodologies that were implemented witnessed significant optimization in terms of time and resources consumed.

Vamsi Krishna Myalapalli, Bhupati Lohith Ravi Teja, "High Performance PL/SQL Programming[2]" presents different techniques that can reduce time and space complexity of a native SQL query and PL/SQL script. Their analysis reduced the rate of Context switching (an overhead) among SQL engine and PL/SQL engine. This paper enunciated appraisals for overhauling PL/SQL queries and gazes at uplifting existing tuning engineering methodologies.

Jean Habimana, "Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries[3]" proposed various general techniques that we can use to try to optimize our database queries. This paper does not focus on, in- depth analysis of database but simple query tuning tips & tricks which can be applied to gain immediate performance gain.

He suggested various tips like Using Column Names Instead of * in a SELECT statement, avoiding including a HAVING clause in SELECT statements, Un-nesting sub queries, avoiding using OR in join conditions, use of UNION ALL in place of UNION, avoiding functions on the right hand side of the operator, removing any redundant mathematics, etc.

Jintao Gao, Wenjie Liu, Hongtao Du and Xiaofang Zhang, "Batch Insertion Strategy in a Distribution Database[4]" proposes a batch insertion strategy(BIS) which includes multiple insertion strategies used to implement large data inserting, threshold optimization technology used to decrease the network cost and redirection technology used to reduce the pressure of system. BIS can flexibly and

efficiently insert large data into a distributed system. They introduced how to implement BIS, including to rebuild physical plan, overcoming network limitation, the redirection technology, and the insertion threshold optimization. Through experiments, they showed that the inserting function based on BIS can satisfy the need of financial enterprise.

Majid Khan, M. N. A. Khan, "Exploring Query Optimization Techniques in Relational Databases[5]", review different query optimization techniques and approaches discussed in the contemporary literature for both centralized and distributed databases. This paper also highlights merits of these techniques by critically analyzing them with respect to their utility and efficacy. They have discussed the existing techniques and their implementation for the sake of optimizing query. They identified some of the proposed techniques that lead towards achieving the key benefits of an optimized query as compared to an un-optimized query in terms of its throughput and response time.

Weipeng P. Yan, Per Ake Larson, "Performing Group by before join[6]" proposed a new strategy for processing SQL queries containing group-by namely pushing the group-by operation past one or more joins. They consider sql queries containing joins and group by. They say that even though the standard way of evaluating this type of queries is to perform all the join first and then the group by operation, it may be possible to perform group by early i.e. to push group by past one or more joins. According to them, early grouping may reduce the query processing cost by reducing the amount of data participating in joins.

Abhijit Banubakode, Virandra Dakhode, "Comparative Analysis of Query Optimization in the Object-Oriented Database & Relational Databases Using Clauses[7]" concludes that first if the group-by clause applied before conditional statement then there is significant cost reduction. The logic behind this optimization is based on the fact that a group-by reduces the cardinality of a relation, therefore if group by is evaluated early, it could reduce the cost of subsequent joins.

Won Kim, "On Optimizing an SQL-like Nested Query[8]" suggest to transform the nested queries to equivalent, non-nested queries that existing optimizers are designed to process more efficiently. He develops algorithms which transform nested queries to equivalent, non nested queries. The algorithms are based on alternative ways of interpreting the operations of queries which involve the four types of nesting, and may often improve the performance of nested queries by orders of magnitude.

III. QUERY OPTIMIZATION TECHNIQUES

The following are some of the basic query optimization techniques.

1. Insert statement with multiple values

We send multiple rows to be inserted into a single packet to reduce the number of network packets. This reduces overhead involved in processing each INSERT statement individually [9].
The LOAD DATA INFILE can also be used for high speed inserts. This inserts data from a text/csv file into a table at a very high speed.

2. Cartesian join optimization

Predicate pushdown: Perform predicate filtering before join to reduce size of join

Predicate pushdown is applying the condition as early as possible to reduce the number of rows participating in joins. Predicate pushdown can optimize the query by doing things like filtering data before it is transferred over the network, filtering data before loading into memory, or skipping reading entire files or chunks of files [10].

3. Group-by optimization

If group-by clause is applied before conditional statement then there is significant cost reduction. The logic behind this optimization is based on the fact that a group-by reduces the cardinality of a relation, therefore if group by is evaluated early, it could reduce the cost of subsequent joins. When group-by clause is applied after condition, search time is more and CPU cost is high. When group-by clause is applied before condition, search time is less and CPU cost is less [7].

IV. COMPARATIVE ANALYSIS WITH EXPERIMENTAL RESULTS

The schema of sample database used for analysisis as follows. Consider this as a sample office database having details about employees, their designations and projects assigned to them.
There are five tables named employee, designation, projects, emp_x_designation(employees and their designations), emp_x_projects(employees and projects assigned to them).
employee(emp_id, address)
designation(desgn_id, desgn_name)
projects(proj_id, proj_name)
emp_x_designation(exd_emp_id, exd_desgn_id)
emp_x_projects(exp_emp_id, exp_proj_id)

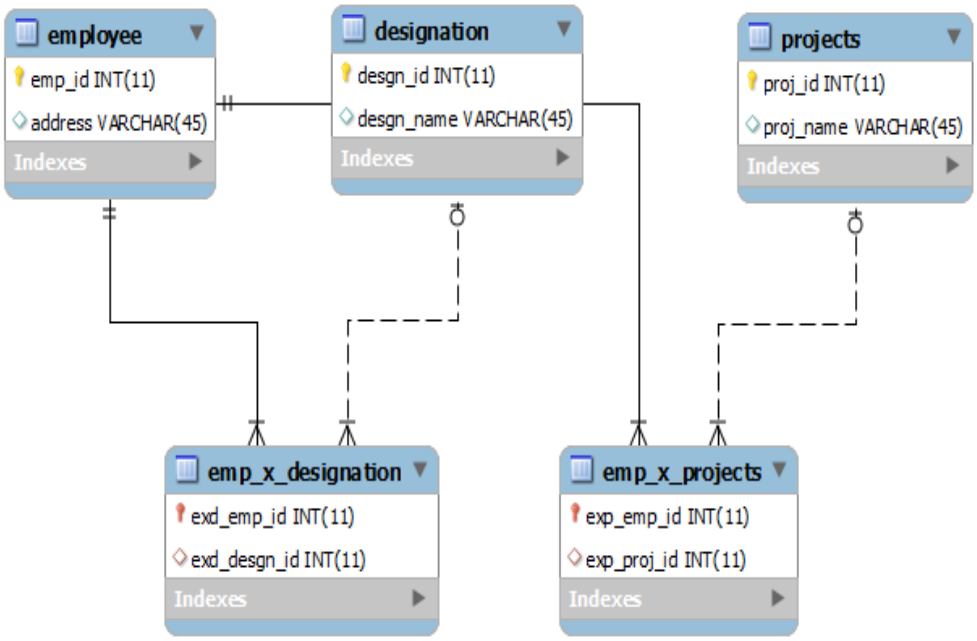

Fig.1 Schema for sample database used

In this section, we compare above mentioned query optimization techniques with respect to their execution time.

1. Simple Insert statement Vs Insert statement with multiple values

There are different ways to insert multiple values into a table.

1. One is to use simple insert statement multiple times which inserts only one row at a time.

```
INSERT into employee values (“emp_1”, “address_1”);
INSERT into employee values (“emp_2”, “address_2”);
INSERT into employee values (“emp_n”, “address_n”);
```

2. Other way is to use insert statement with multiple row values to insert data in single statement.

```
INSERT into employee values (“emp_1”, “address_1”), (“emp_2”, “address_2”),…. (“emp_n”, “address_n”);
```

3. We can also use a text/csv file to import file data into the table.
   Suppose emp-details.txt file contains delimiter separated data to be inserted into employee table.

```
LOAD DATA INFILE 'C:/temp/emp-details.txt' INTO TABLE  employee IGNORE 1 LINES;
```

For inserting **83566 rows** in employee table, 1st method using simple insert statements took around **1 hour**.
The 2nd method took only **6sec** to insert same number of rows in the same table (records were split in batches of 1000 in 2nd method).
Importing data from file took **1.76sec** for inserting same number of records.
Insert statements with multiple row values are faster because it combines the data in a single packet thereby reducing the overhead of processing every packet independently over the network.

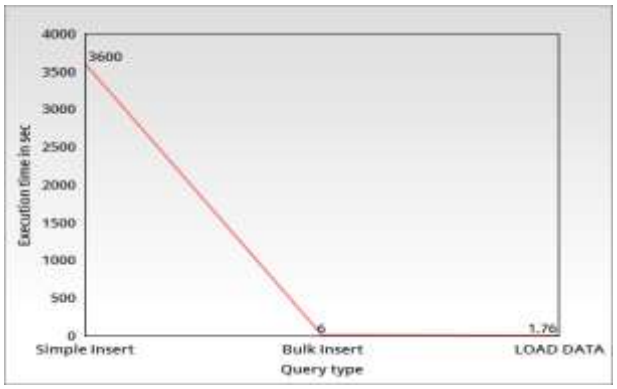

Fig.2 Execution time comparison of Insert queries

2. Cartesian join optimization

Consider a query to get id and project details of employees who are at manager level. This query is written below in two different ways.

1. First way is traditional cartesian join and then using filter condition.

   Select emp.id, epr.exp_proj_id
   from employee emp, emp_x_designation edn, emp_x_projects epr
   where emp.id=edn.exd_emp_id and emp.id=epr.exp_emp_id
   and edn.exd_desgn_id=(select desgn_id from designation where desgn_name='manager');

   1946956 row(s) returned 0.416 sec

2. Other way is to try to apply the filter condition early to reduce the number of rows participating in joins
   Select * from emp_x_projects epr,
   (select emp.id from employee emp, emp_x_designationedn where emp.id=edn.exd_emp_id and edn.exd_desgn_id=(select desgn_id from designation where desgn_name='manager')) temp where epr.exp_emp_id=temp.id;

   1946956 row(s) returned 0.015 sec

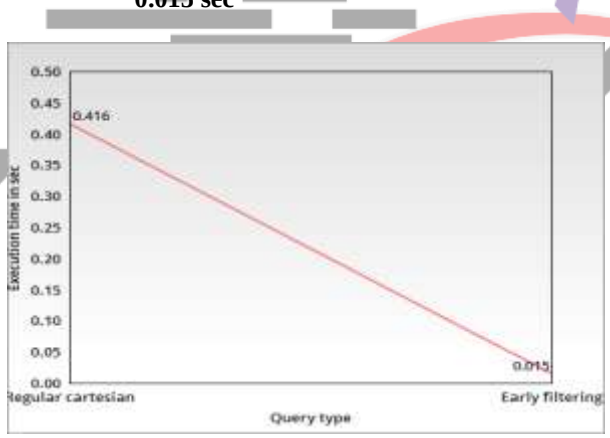

Fig.3 Execution time comparison of Cartesian join queries

3. Group-by optimization

Consider a query to get project wise count of employees except the first project.

1. One way is to apply where clause and then group by project_id

   select exp_proj_id, count(*) from emp_x_projects epr
   where epr.exp_proj_id !=1 group by epr.exp_proj_id

   57616 row(s) returned 36.395 sec

2. Another way is to apply group by first and then use the filter condition

   select exp_proj_id, count(*) from emp_x_projects epr

group by epr.exp_proj_id having epr.exp_proj_id !=1

57616 row(s) returned **15.428 sec**

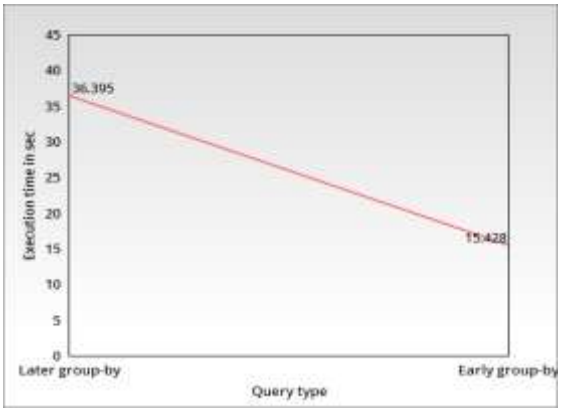

Fig.3 Execution time comparison of group-by queries

V. CONCLUSION AND FUTURE SCOPE

The size of the databases is increasing exponentially. Despite of the increasing data size, we expect that the database queries return answer in minimum time. As the queries become complex, it becomes necessary to optimize them to arrive at the results quickly. Here query optimization proves to be of great help.

In this paper, we discussed some of the query optimization techniques and compared these different rewrite strategies on the basis of their execution time. It can be seen that just by simple rewrite mechanism, it is possible to achieve the same result set from database with overwhelming reduction in response time.

We further plan to study necessary and sufficient conditions where these query optimization techniques can be applied and establish a trade-off between them. We plan to study and analyze some more query optimization techniques related to basic sql constructs like joins, group-by, etc.

REFERENCES

[1] Vamsi Krishna Myalapalli, ASN Chakravarthy, "Revamping SQL Queries for Cost Based Optimization", 2016 International Conference on Circuits, Controls, Communications and Computing (I4C).

[2] Vamsi Krishna Myalapalli, Bhupati Lohith Ravi Teja, "High Performance PL/SQL Programming", IEEE International Conference on Pervasive Computing, pp . 1 – 6, 8 -10 Jan.2015.

[3] Jean Habimana, "Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries", International Journal Of Scientific & Technology Research Volume 4, Issue 10, October 2015 ISSN 2277-8616 22.

[4] Jintao Gao, Wenjie Liu, Hongtao Du and Xiaofang Zhang, "Batch Insertion Strategy in a Distribution Database", 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS).

[5] Majid Khan and M. N. A. Khan, "Exploring Query Optimization Techniques in Relational Databases", International Journal of Database Theory and Application Vol. 6, No. 3, June, 2013

[6] Weipeng P. Yan, Per Ake Larson, "Performing Group by before join", Proceedings of 1994 IEEE 10th International Conference on Data Engineering.

[7] AbhijitBanubakode, VirandraDakhode, "Comparative Analysis of Query Optimization in the Object-Oriented Database & Relational Databases Using Clauses, International Journal of Advanced Research in Computer Science and Software Engineering Dec.2014.

[8] Won Kim, "On Optimizing an SQL-like Nested Query", ACM Transactions on Database Systems, Vol. 7, No. 3, September 1982.

[9] https://weblogs.asp.net/shahar/sql-server-for-developers-improve-data-loading-performance-using-bulk-insert-sqlbulkcopy

[10] http://bigdatums.net/2017/08/29/what-is-predicate-pushdown

[11] MatthiasJarke, Jurgen Koch, "Query Optimization in Database Systems", ACM Computing Surveys (CSUR) Volume 16 Issue 2, June 1984.

[12] D.Saisanguansat and P.Jeatrakul, "Improving Optimization Performance on PL/SQL", 15th International Conference on ICT and Knowledge Engineering (ICT&KE) 2017.

[13]Vamsi Krishna Myalapalli, ASN Chakravarthy and KeesariPrathap Reddy "Accelerating SQL Queries by Unraveling Performance Bottlenecks in DBMS Engine", 2015 International Conference on Energy Systems and Applications (ICESA 2015).