Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries

Jean HABIMANA

Abstract: SQL statements can be used to retrieve data from any database. If you've worked with databases for any amount of time retrieving information, it's practically given that you've run into slow running queries. Sometimes the reason for the slow response time is due to the load on the system, and other times it is because the query is not written to perform as efficiently as possible which is the much more common reason. For better performance we need to use best, faster and efficient queries. This paper covers how these SQL queries can be optimized for better performance. Query optimization subject is very wide but we will try to cover the most important points. In this paper I am not focusing on, in- depth analysis of database but simple query tuning tips & tricks which can be applied to gain immediate performance gain.

I. INTRODUCTION

Query optimization is an important skill for SQL developers and database administrators (DBAs). In order to improve the performance of SQL queries, developers and DBAs need to understand the query optimizer and the techniques it uses to select an access path and prepare a query execution plan. Query tuning involves knowledge of techniques such as cost-based and heuristic-based optimizers, plus the tools an SQL platform provides for explaining a query execution plan.The best way to tune performance is to try to write your queries in a number of different ways and compare their reads and execution plans. In this paper I proposed various techniques that you can use to try to optimize your database queries.

II. GENERAL TIPS FOR QUERY OPTIMIZATION

Each tip was tested by running both the original query and improved query while retrieving information from the Oracle 11g sample database especially on Sales schema. I recorded the average time of each query to show the speed increase of using the more efficient query.

Tip #1:

Use Column Names Instead of * in a SELECT Statement

If you are selecting only a few columns from a table there is no need to use SELECT *. Though this is easier to write, it will cost more time for the database to complete the query. By selecting only the columns you need, you are reducing the size of the result table, reducing the network traffic and also in turn boosting the overall performance of the query.

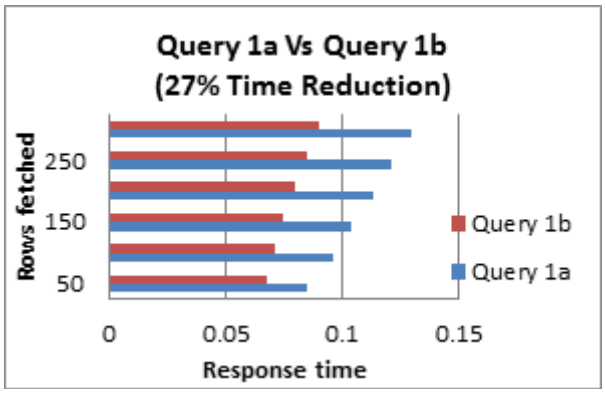
Example:

Original query:

```
SELECT * FROM SH.Sales;
```

Improved query:

```
SELECT s.prod_id FROM SH.sales s;
```



Tip #2:

Avoid including a HAVING clause in SELECT statements

The HAVING clause is used to filter the rows after all the rows are selected and it is used like a filter. It is quite useless in a SELECT statement. It works by going through the final result table of the query parsing out the rows that don't meet the HAVING condition.

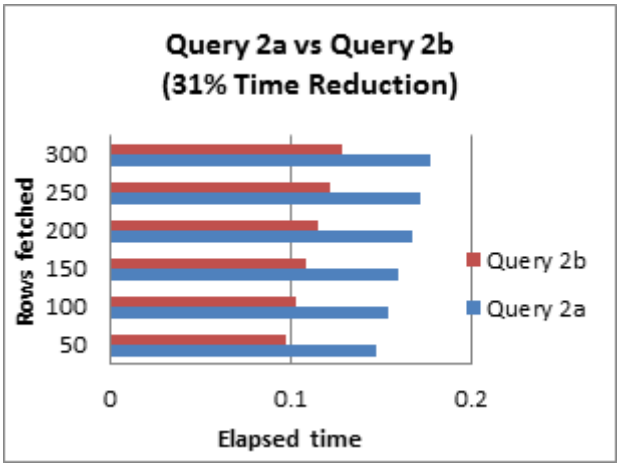
Example:

Original query:

```
SELECT s.cust_id,count(s.cust_id)
FROM SH.sales s
GROUP BY s.cust_id
HAVING s.cust_id != '1660' AND s.cust_id != '2';
```

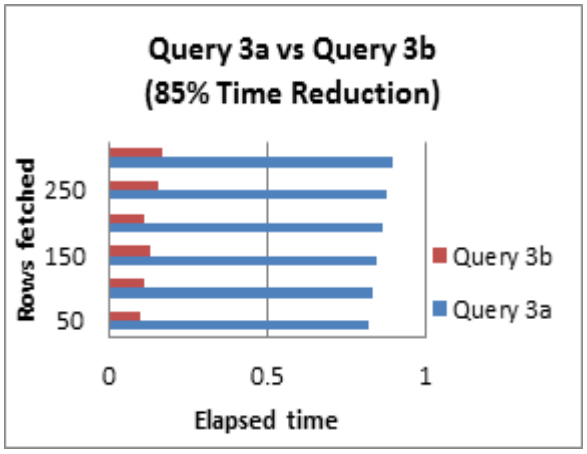
Improved query:

```
SELECT s.cust_id,count(cust_id)
FROM SH.sales s
WHERE s.cust_id != '1660'
AND s.cust_id !='2'
GROUP BY s.cust_id;
```



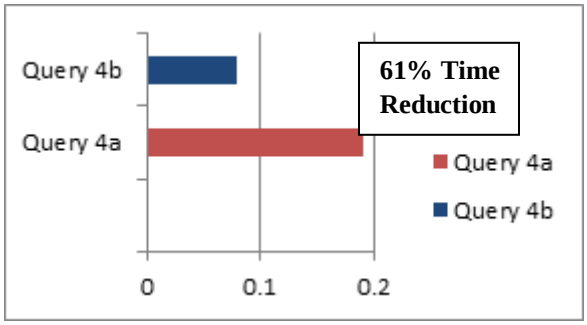
Tip #3:
Eliminate Unnecessary DISTINCT Conditions
Considering the case of the following example, the DISTINCT keyword in the original query is unnecessary because the table_name contains the primary key p.ID, which is part of the result set.

Example:
Original query:
```
SELECT DISTINCT * FROM SH.sales s
JOIN SH.customers c
ON s.cust_id= c.cust_id
WHERE c.cust_marital_status = 'single';
```
Improved query:
```
SELECT * FROM SH.sales s JOIN
SH.customers c
ON s.cust_id = c.cust_id
WHERE c.cust_marital_status='single';
```



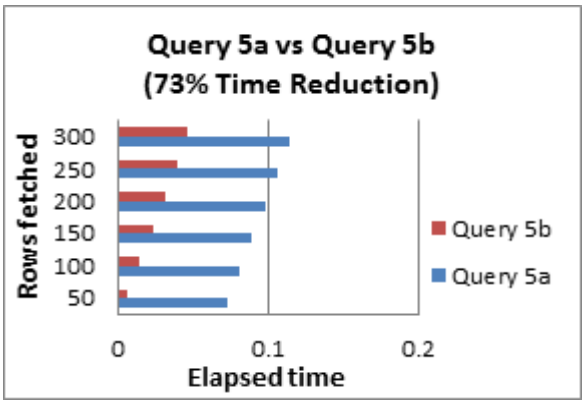
Tip #4:
Un-nest sub queries
Rewriting nested queries as joins often leads to more efficient execution and more effective optimization. In general, sub-query un-nesting is always done for correlated sub-queries with, at most, one table in the FROM clause, which are used in ANY, ALL, and EXISTS predicates. A uncorrelated sub-query, or a sub-query with more than one table in the FROM clause, is flattened if it can be decided, based on the query semantics, that the sub-query returns at most one row.

Example:
Original query:
```
SELECT *
FROM SH.products p
WHERE p.prod_id =
   (SELECT s.prod_id
     FROM SH.sales s
     WHERE s.cust_id = 100996
       AND s.quantity_sold = 1 );
```
Improved query:
```
SELECT p.*
FROM SH.products p, sales s
WHERE p.prod_id = s.prod_id
AND s.cust_id = 100996
AND s.quantity_sold = 1;
```



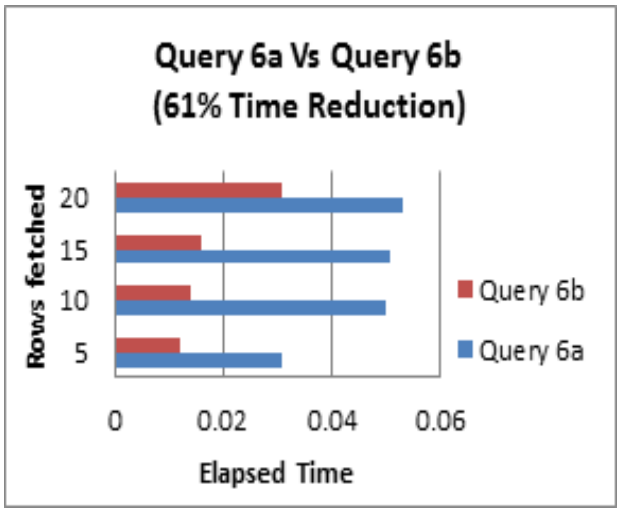
Tip #5:
Consider using an IN predicate when querying an indexed column
The IN-list predicate can be exploited for indexed retrieval and also, the optimizer can sort the IN-list to match the sort sequence of the index, leading to more efficient retrieval. Note that the IN-list must contain only constants, or values that are constant during one execution of the query block, such as outer references.

Example:
Original query:
```
SELECT s.*
FROM SH.sales s
WHERE s.prod_id = 14
  OR s.prod_id = 17;
```
Improved query:
```
SELECT s.*
FROM SH.sales s
WHERE s.prod_id IN (14, 17);
```



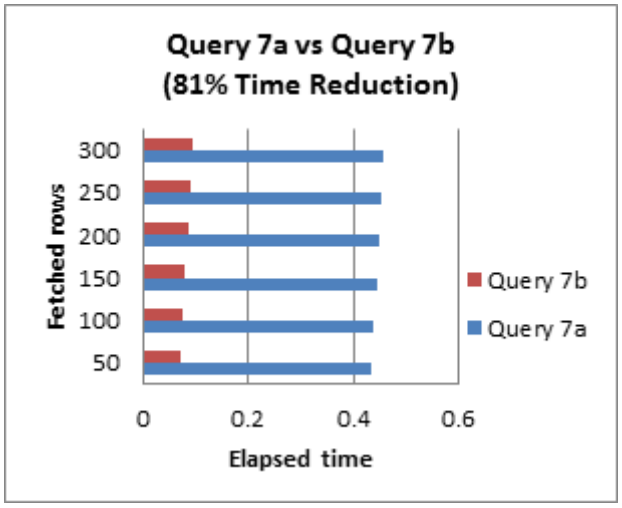
Tip #6:
Use EXISTS instead of DISTINCT when using table joins that involves tables having one-to-many relationships
The DISTINCT keyword works by selecting all the columns in the table then parses out any duplicates.Instead, if you use sub query with the EXISTS keyword, you can avoid having to return an entire table.

Example:
Original query:
SELECT DISTINCT c.country_id, c.country_name
FROM SH.countries c,SH.customers e
WHERE e.country_id = c.country_id;
Improved query:
SELECT c.country_id, c.country_name
FROM SH.countries c
WHERE EXISTS (SELECT 'X' FROM SH.customers e WHERE e.country_id = c.country_id);



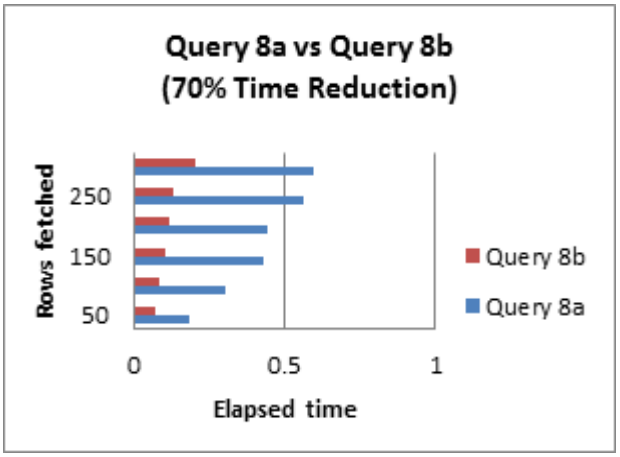
Tip #7:
Try to use UNION ALL in place of UNION
The UNION ALL statement is faster than UNION, because UNION ALL statement does not consider duplicate s, and UNION statement does look for duplicates in a table while selection of rows, whether or not they exist.

Example:
Original query:
SELECT cust_id
FROM SH.sales
UNION
SELECT cust_id
FROM customers;
Improved query:
SELECT cust_id
FROM SH.sales
UNION ALL
SELECT cust_id
FROM customers;



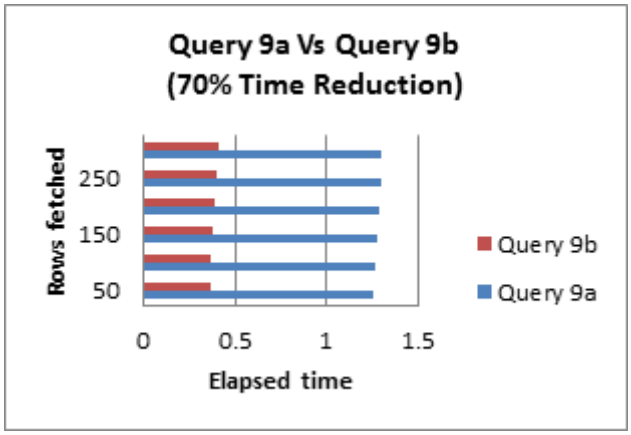
Tip #8:
Avoid using OR in join conditions
Any time you place an 'OR' in the join condition, the query will slow down by at least a factor of two.

Example:
Original query:
SELECT *
FROM SH.costs c
INNER JOIN SH.products p ON c.unit_price = p.prod_min_price OR c.unit_price = p.prod_list_price;
Improved query:
SELECT *
FROM SH.costs c
INNER JOIN SH.products p ON c.unit_price = p.prod_min_price
UNION ALL
SELECT *
FROM SH.costs c
INNER JOIN SH.products p ON c.unit_price = p.prod_list_price;



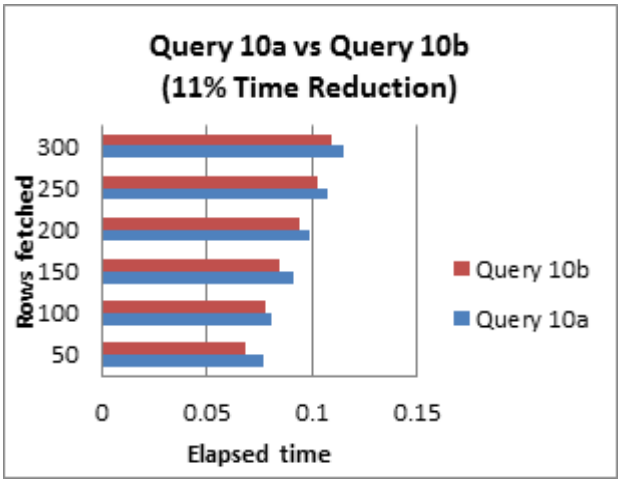
Tip #9:
Avoid functions on the right hand side of the operator
Functions or methods are used very often with their SQL queries. Rewriting the query by removing aggregate functions will increase the performance tremendously.

Example:
Original query:
SELECT *
FROM SH.sales
WHERE EXTRACT (YEAR FROM TO_DATE (time_id, 'DD-MON-RR')) = 2001 AND EXTRACT (MONTH FROM TO_DATE (time_id, 'DD-MON-RR')) =12;
Improved query:
SELECT * FROM SH.sales
WHERE TRUNC (time_id) BETWEEN TRUNC(TO_DATE('12/01/2001', 'mm/dd/yyyy')) AND TRUNC (TO_DATE ('12/30/2001','mm/dd/yyyy'));



Tip #10:
Remove any redundant mathematics
There will be times where you will be performing mathematics within an SQL statement. They can be a drag on the performance if written improperly. For each time the query finds a row it will recalculate the math. So eliminating any unnecessary math in the statement will make it perform faster.

Example:
Original query:
SELECT *
FROM SH.sales s
WHERE s.cust_id + 10000 < 35000;
Improved query:
SELECT *
FROM SH.sales s
WHERE s.cust_id < 25000;



III. CONCLUSION

Query optimization is a common task performed by database administrators and application designers in order to tune the overall performance of the database system. The purpose of this paper is to provide SQL scenarios to serve as a quick and easy reference guide during the development phase and maintenance of the database queries. Even if you have a powerful infrastructure, the performance can be significantly degraded by inefficient queries. Query optimization has a very big impact on the performance of a DBMS and it continuously evolves with new, more sophisticated optimization strategies. So, we should try to follow the general tips as mentioned above to get a better performance of queries. Optimization can be achieved with some efforts if we make it a general practice to follow the rules. The main focus was on query optimizations.

IV. REFERENCES

[1] 10 Ways to Improve SQL Query Performance http://www.developer.com/db/10-ways-to-improve-sql-query-performance.html

[2] 15 Ways to Optimize Your SQL Queries http://hungred.com/useful-information/ways-optimize-sql-queries/

[3] Optimize SQL Server queries with these advanced tuning techniques. http://www.techrepublic.com/blog/the-enterprise-cloud/optimize-sql-server-queries-with-these-advanced-tuning-techniques/.

[4] Making Queries Run Faster https://sqlschool.modeanalytics.com/advanced/faster-queries.html.

[5] Query Optimization Techniques in Microsoft SQL Serverhttp://www.dbjournal.ro/archive/16/16_4.pdf

[6] SQL Tuning or SQL Optimization. http://beginner-sql-tutorial.com/sqlquery-tuning.htm

[7] SQL Server Optimization Tips.http://santhoshgudise.weebly.com/uploads/8/5/4/7/8547208/sql_server_optimization_tips-1.doc

[8] Efficient SQL Statements https://oracle-base.com/articles/misc/efficient-sql-statements

[9] Best Way to Write SQL Query. http://www.ifadey.com/2010/11/best-way-to-write-sql-query/

[10] SQL Tuning Guidelines for Oracle - Simple yet Effective! http://askanantha.blogspot.com/2007/10/sql-tuning-guidelines-for-oracle-simple.html

[11] What are the most common SQL Optimizations. http://stackoverflow.com/questions/1332778/what-are-your-most-commonsql-optimizations

[12] Top 10 performance tuning tips for relational databases. http://web.synametrics.com/top10performancetips.htm