International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510

SQL TUNING IN ORACLE - A PROGRAMMER'S GUIDE FOR TARGETED PERFORMANCE IMPROVEMENT

Rajalakshmi Thiruthuraipondi Natarajan
rajalan11@gmail.com

Abstract

SQL Tuning is the process of making changes to the code wither by rewriting or adding boosters to the queries and codes, underlying an application for improved performance and turnaround time, without compromising the functionality or user experience. As the business grows, so will the data accumulated in their systems, and with time, the code / query initially designed might not be optimized to deliver the best performance. Hence, it is the responsibility of the developers and support teams to revisit such underperforming programs and provide optimal solution so that the application will perform to the best of its ability.
This article is written for developers and support teams, who deals with the PL/SQL codes on ways to identify the high turnaround code, isolate the problematic section, identify appropriate solution and implement the same.

Index Terms – SQL Tuning, AWR, SQL Trace, Query redesign, Performance Improvement, Explain Plan

I. INTRODUCTION

Every Organization, from the day of its inception, continues accumulating data, either as part of their daily operations or service or for planning and research activities and it only keeps growing with time. The ability to harness the collected data in an accurate and timely fashion is key to the success and survival of the company. Hence it is imperative that all systems used in everyday operations be tuned for the most optimal performance. While Oracle Databases are arguably the best in the market and come in loaded with features for effective and easy storage and retrieval, there also need to be a continuous process defined by and for IT teams to perform periodic checks to capture the obvious and subtle indicators of performance degradation on these systems, re-baseline it and act as needed.
Oracle Database comes with provides a plethora of tools inbuilt to monitor the performance, which provides a detailed breakdown on the behaviour of each statement to the most granular level. This provides a great deal of information to every team such as DBA, system administrator and application developer who can use this information to fine tune the application in their own way, such as alter the database parameters, or add more hardware or make specific modifications to the code to effectively balance between the performance and cost.

II. FACTORS AFFECTING THE DATABASE PERFORMANCE

Like, every machine, the performance of an application can degrade over time, however, unlike hardware; it is due to wear and tear, but due to evolving technical landscape, such as introduction of new interfaces or systems, compatibility to integrating systems, load, etc. Among these factors, from each team's standpoint, there are certain attributes that can be adjusted to improve the performance and some out of our control.

A. Hardware & Network Limitations

From an application developer or support view, this is one of the factors that they have very little to no control. These are generally controlled at the corporate level and will be modified or upgraded only if all other options are exhausted or if the market demands to stay competitive, since it usually has significant financial and operational impact.
The physical server on which the database and applications are installed, determines the overall performance. Its obvious that a server with a higher number of CPUs and memory and storage, would perform much better than the lower ones. But such powerful configuration comes with a cost. As the load on the server gets higher, the systems need to work harder and beyond a threshold, they would compromise performance to stability.
Similarly, network is another piece of the infrastructure that can greatly impact the performance, but the application team has no control over. With organizations operating globally, network acts as the nerves systems to connect and make the data readily available to all corners of the world. Depending on the infrastructure, there is only certain amount of traffic these networks can handled and anything beyond it, would affect the turnaround time.

B. Data Volume

This is a fuzzy area, where the application team has partial control. As mentioned earlier, it is quite obvious that the larger the data, longer it takes for the engines to parse through them and return the value. While it is a corporate decision to determine what data needs to be gathered and how, application development and support team can determine how to maintain them. For example, a retail organization can decide to collect their customer spending patterns, however, the application team can design a solution on how to categorize the data like critical, non-critical or highly requested data or not and store them and create indexes accordingly to optimize the performance. But any such design is based on the assumptions made using available requirement and information and deviation or evolution might render the design obsolete and affect accuracy and performance.

C. Ineffective Coding

This is one part of the entire performance puzzle that the application development or support team has total control. Ineffective code is a piece (or whole) in its current form does not deliver the optimal performance. It might either mean it was written poorly or the current design is no longer effective, resulting in either too many iterations or physical reads.
It is natural that at the time of development, the teams focus primarily on the functionality and accuracy and the performance takes a back seat. And since it is tested with sample data, which in many cases are a fraction of what we might see in production, it usually clears the checks and works fine fore period of time, before it starts slowing down. The other reason is that the assumptions made might no longer hold good, like the design might be made that the database would grow in a certain direction, but in reality, it might not. For example, In Payables, it is a general assumption that the invoices table would be significantly higher than payments tables, since there usually is one payment for multiple invoices. However, if the company adopts partial payments or group invoices for payment is a more granular level or there is a significant check returns and reissue, the payments table might grow much bigger than anticipated and any design

iJCEM
International Journal of Core Engineering & Management
International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510

made around this assumption might not work as expected.

D. External Factors

This is a catchall of anything and everything outside of the database or application that might slow it down. Usually, the performance impact caused by this type, is only for a short period of time. There could be a heavy process or incompatible activity that is running in the system, that temporarily affects the performance. For instance, there might be a database back up activity scheduled, during which the system resources will be heavily used by this activity leaving rest of the application running slow or a report that was run with a wide range of parameters, which would need several iterations, resulting in longer runtime.

III. IDENTIFY INEFFICIENT SQL AND PREPARATION

Oracle database as a product has several options to monitor and identify the performance inbuilt within them, which generates detailed reports as requested and each piece of this report can be used by various teams such as DBAs, system administrators, developers to act on it to their capacity.

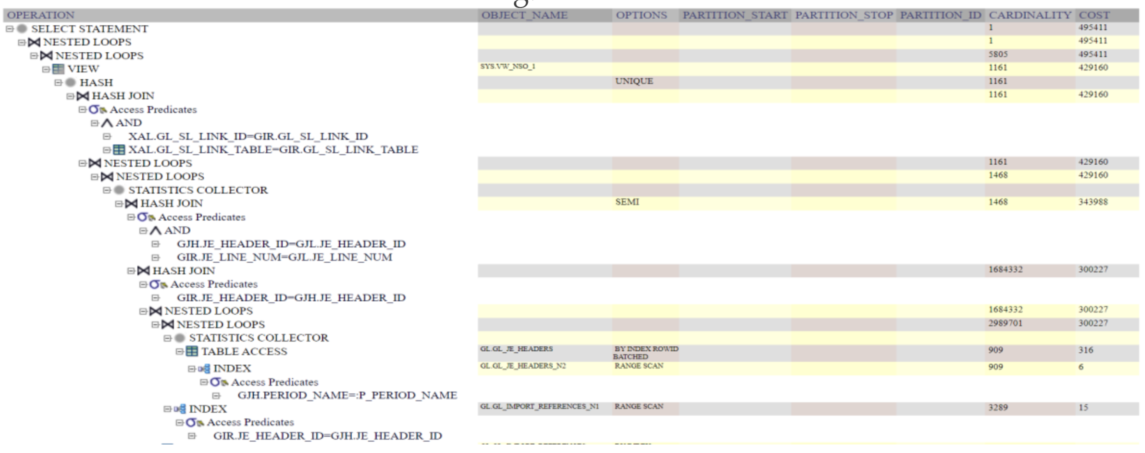
A. Trace files

Anyone who in involved with Oracle ERP applications would know the significance of trace files. These are the reports that can be enabled wither at session level or on a specific form or a concurrent program. The trace files capture every SQL that is executed by the form or the program during the time it was enabled.

B. Explain Plan

Explain Plan is the path that Oracle optimizer takes to execute a statement. It shows the tables the optimizer parses, the indexes it uses, the matching and sorting algorithm it uses, etc. The key part is the cost and cardinality associated with it. The cost in a explain plan is the resource used while executing the step, such as CPU, physical reads, etc. and cardinality is the number of rows fetched in each step. In general, higher the cost and cardinality, longer is the turnaround time.

Fig. 1



OPERATION	OBJECT_NAME	OPTIONS	PARTITION_START	PARTITION_STOP	PARTITION_ID	CARDINALITY	COST
SELECT STATEMENT						1	495411
NESTED LOOPS						1	495411
NESTED LOOPS						5805	495411
VIEW	SYS.VW_NSO_1					1161	429160
HASH		UNIQUE				1161	
HASH JOIN						1161	429160
Access Predicates							
AND							
XAL.GL_SL_LINK_ID=GIR.GL_SL_LINK_ID							
XAL.GL_SL_LINK_TABLE=GIR.GL_SL_LINK_TABLE							
NESTED LOOPS						1161	429160
NESTED LOOPS						1468	429160
STATISTICS COLLECTOR							
HASH JOIN		SEMI				1468	343988
Access Predicates							
AND							
GJH.JE_HEADER_ID=GJL.JE_HEADER_ID							
GIR.JE_LINE_NUM=GJL.JE_LINE_NUM							
HASH JOIN						1684332	300227
Access Predicates							
GIR.JE_HEADER_ID=GJH.JE_HEADER_ID							
NESTED LOOPS						1684332	300227
NESTED LOOPS						2989701	300227
STATISTICS COLLECTOR							
TABLE ACCESS	GL.GL_JE_HEADERS	BY INDEX ROWID BATCHED				909	316
INDEX	GL.GL_JE_HEADERS_N2	RANGE SCAN				909	6
Access Predicates							
GJH.PERIOD_NAME=:P_PERIOD_NAME							
INDEX	GL.GL_IMPORT_REFERENCES_N1	RANGE SCAN				3289	15
Access Predicates							
GIR.JE_HEADER_ID=GJH.JE_HEADER_ID							

C. AWR Report

If, there comes a situation to capture the performance statistics of the entire Oracle database for a given period, AWR report is the way to go. AWR (Automated Workload Repository) report is an elaborate report that can be generated between a start and end times to capture all the activities the activities in the database. This captures every query executed during the time frame and provides a drilldown into each performance factors such as CPU, DB read/write, parses, elapsed time and much more. This also displays the top queries by elapsed time, using which we can determine the problematic area(s).

D. SQL Analysis

Now that the inefficient query has been identified, the next logical step is to understand the functionality of the code, i.e., what is the intended result of the query. Any deviation to the intended output for performance is unacceptable. This analysis should be done locally, where the impacted query is analysed to deduce the result, it produces and the bigger picture where, we look at the overall functionality of the component this query is a part of and how it fits the puzzle. For example, the problematic query might pull all the journals and subledger accounting entries for a given period, however, this query might be part of a package, which is designed to derive the journal entries of a particular module in the given period. This puts a whole new perspective on how to tune the code.

IV. SQL TUNING

Depending on the statement that is causing performance issues, there are several ways to tune the query. The development team needs to strategize and find the right fit not only for the current volume, but potential increase in the long tun. The decision also needs to be made based on hardware and software capabilities, data volume and access frequencies

A. Indexes

Indexes in Oracle databases are pointers to records in a table using column(s) that closely represent the record. Indexes are typically created on columns that are key identifiers of the record or the columns that are most frequently used to retrieve the records. For instance, in Oracle, General Leader header table, header id is the primary key and an unique index that precisely represents the record, however, period name column is the most frequently used for retrieving the data, hence there are indexes created on these columns.

The need to create index can be identified if the tables in the inefficient query does not have any scope to use indexes, meaning, the columns used in the query do not have any indexes are on them. This is a good idea, if the tables involved is custom or there are attribute columns used in the query. By creating index, we are effectively creating a direct pointer to the records, there by substantially improving the performance.

The other way is to force the query to use an index by using optimizer hints. This is done in scenarios where the optimizer might not be using the right index. For example, if both sides of the where condition contains columns that are indexed, optimizer with chose based on its predefined rules to pick one of the indexes, which might not be the right fit, in which case, the hint would suggest the query to use the right index.

However, once need to be cautious while creating the indexes, since this would affect every piece of code that uses the table, which might inadvertently affect other programs. Also, index is a costly

International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510

set-up since it takes up storage to create pointers, Similarly, using hints are kind of suggestions to optimizer, which might not be used during execution.

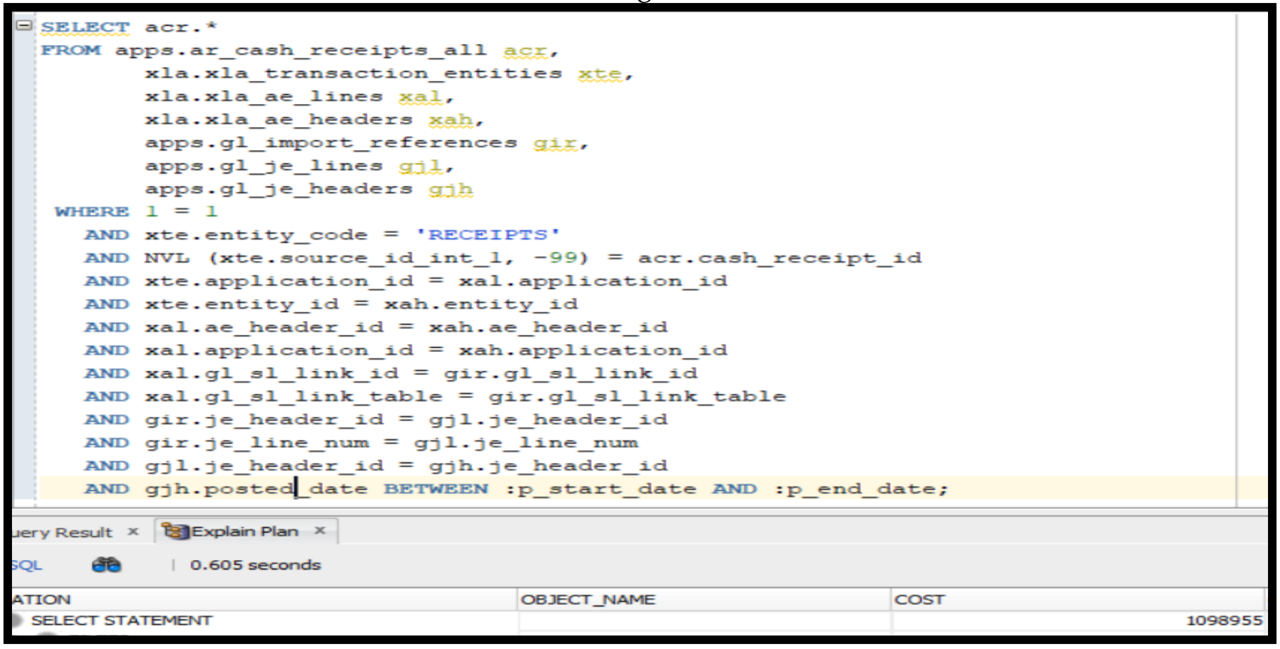
B. Explain Plan Pinning

In a scenario where a certain SQL was working fine but started having performance issues with no significant change in data volume, it is possible that the explain plan for the query might have changed. There is no one clear reason behind this chance but can be loosely tied to the change in optimizer plan because of gather schema statistics. In this scenario, the old explain can be pinned to the query, thereby effectively forcing the optimizer to use an execution path of our choosing.
As always, one need to be cautious while pinning the execution plan, since it prevents the optimizer from choosing a better plan which might be more optimal than the existing one.

C. SQL Rewriting

In this method, the problematic SQL will be analysed and rewritten to deliver quicker turnaround. The code needs to be viewed from both drilled down to individual tables and level up which contains the query and design accordingly.
The first step is breaking down the complex query into individual table's select statement, and gradually connecting the tables one by one. With each addition, the turnaround time and cost should be noted, until the table and the condition which affects the performance. Once the connection is identified and isolated, the appropriate solution needs to be found based on the significance of the table, the columns used, etc.
One of the straightforward methods is to see if the query can be simplified there by eliminating problematic table. Oracle tables contain several columns which get updated as part of each process in its own way. Though the values might be different, they would roll up to a same condition. For example, in the below queries, both would return the same result, but one is much simpler and effective than the other

Fig. 2

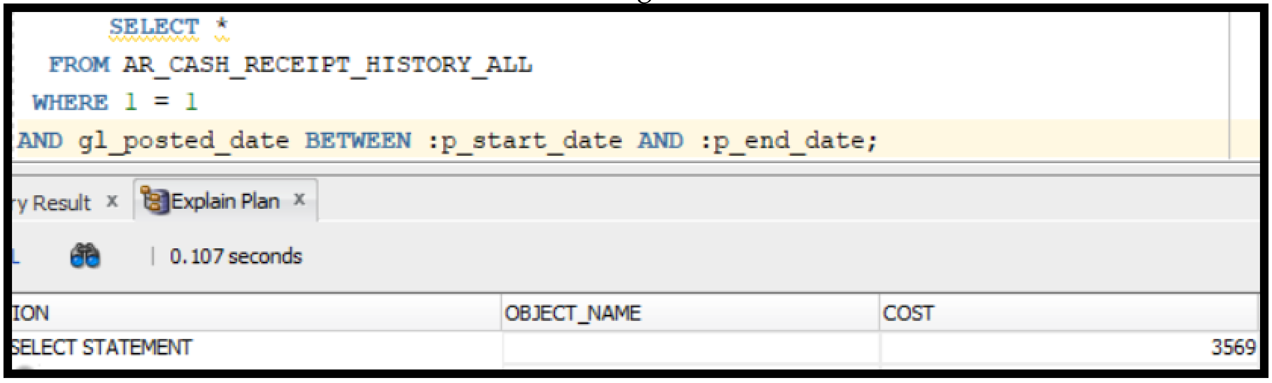


```
SELECT acr.*
FROM apps.ar_cash_receipts_all acr,
        xla.xla_transaction_entities xte,
        xla.xla_ae_lines xal,
        xla.xla_ae_headers xah,
        apps.gl_import_references gir,
        apps.gl_je_lines gjl,
        apps.gl_je_headers gjh
 WHERE 1 = 1
   AND xte.entity_code = 'RECEIPTS'
   AND NVL (xte.source_id_int_1, -99) = acr.cash_receipt_id
   AND xte.application_id = xal.application_id
   AND xte.entity_id = xah.entity_id
   AND xal.ae_header_id = xah.ae_header_id
   AND xal.application_id = xah.application_id
   AND xal.gl_sl_link_id = gir.gl_sl_link_id
   AND xal.gl_sl_link_table = gir.gl_sl_link_table
   AND gir.je_header_id = gjl.je_header_id
   AND gir.je_line_num = gjl.je_line_num
   AND gjl.je_header_id = gjh.je_header_id
   AND gjh.posted_date BETWEEN :p_start_date AND :p_end_date;
```

```
| 0.605 seconds
ATION                          OBJECT_NAME                    COST
 SELECT STATEMENT                                                    1098955
```

International Journal of Core Engineering & Management
Volume-6, Issue-11, 2021 ISSN No: 2348-9510

Fig. 3



```
      SELECT *
  FROM AR_CASH_RECEIPT_HISTORY_ALL
 WHERE 1 = 1
AND gl_posted_date BETWEEN :p_start_date AND :p_end_date;
```

```
ION                          OBJECT_NAME            COST
SELECT STATEMENT                                    3569
```

The other method is to rewrite the query either by rearranging it or breaking them down into smaller statements or separating them from the main query. The intention should be to minimize the iteration and physical reads which in turn will improve the performance.
Rearranging involves moving of the tables and conditions around within the query so that it will have least impact in the query's performance. Understanding how the parser works, would greatly help in this. For instance, when using EXISTS, the outer query executes first before the inner query, whereas while using IN, the inner query executes first, before the outer query. Similarly, if a certain table is used only for display purpose and does not contribute to filtering, then it can potentially be used as an inline sub-query.
Breaking down to smaller query is splitting a large query into smaller queries and pass the value to the next query by looping through the values. This should be done in cases where the table's conditions help in filtering the data but not used in display. Also, this is recommended, where the conditions have indexes that cannot be used together. But breaking the query, we can make the optimizer use more indexes, which can improve performance.

V. CONCLUSION

While this topic covers a lot of performance tuning ideas, one need not be limited to these methods. For a skilled performance tuning resource, there no one right solution, but often a combination of multiple pieces coming to gather to improve the efficiency. Also, it important to note that a solution which currently holds good or for a near future might not be as effective with time, change in technology or landscape and the data volume. Hence, it is imperative that performance check needs to be a continuous process and part of IT activities done at regular intervals and revalidate the solution and make necessary changes to have the application working in its best.

REFERENCES

1. "Oracle® Database - Database Performance Tuning Guide - 21c". F32091-03, August 2021
2. "Oracle® Database Performance Tuning Guide 11g Release 2(11.2)", September 2013.
3. "Oracle Database 19c: Performance Management and Tuning" - https://www.scribd.com/document/643828648/2021101184-OracleDatabase19cPerformanceManagementandTuning
4. Preeti Singh Bhadoria, Ayushi Chhabra, Sanjay Ojha #, "SQL PERFORMANCE TUNING

IN ORACLE 10g AND 11g", International Journal of Software and Web Sciences, 7(1), December 2013- February 2014, pp. 13-16

5. Gary Harrison, "Oracle sql high performance tuning", https://www.slideshare.net/slideshow/oracle-sql-high-performance-tuning/8639284, Jul, 2011
6. Mark Simmons, "Oracle SQL Tuning with OEM", https://www.linkedin.com/pulse/oracle-sql-tuning-oem-mark-simmons/, Sep, 2020
7. "SQL tuning: real-life example", https://savvinov.com/2012/03/22/sql-tuning-real-life-example/, Mar-2012
8. Quest Software Blog, "How to improve database performance with Oracle query optimization – Basic concepts", https://blog.toadworld.com/how-to-improve-database-performance-with-oracle-query-optimization-basic-concepts, Jul-2020