ISSN : 2231-0711

SQL Performance Tuning

Ch.V.N.Sanyasi Rao[#1], Tiruveedula Gopi Krishna[*2]

[#] *Associate Project Manager, HCL, Bangalore, India*

[1] vnsanyasirao77@gmail.com

[*] *Sirt University*

[2] *Lecturer, Department of Computer Science, Hoon, Libya*

[2] gktiruveedula@gmail.com

***Abstract*— Database performance is one of the most challenging aspects of an organization's database operations. Database tuning can be an incredibly difficult task, particularly when working with large-scale data where even the most minor change can have a dramatic positive or negative impact on performance. A well-designed application may still experience performance problems if the SQL it uses is poorly constructed. It is much harder to write efficient SQL than it is to write functionally correct SQL. As such, SQL tuning can help significantly improve a system's health and performance. The key to tuning SQL is to minimize the search path that the database uses to find the data.**

***Keywords*— SQL tips, optimizer modes, SQL tuning, index hint.**

I. Introduction

When an application submits a SQL query to the database server, the server first parses the SQL. It checks the SQL syntax, for security access and prepares the query for execution. Query optimization involves determining the optimal path for executing the query. Each database comes with built-in intelligent algorithms to figure out the best possible way to execute the query. For complex queries involving joins between eight different tables, the optimizer could spend as much as 30 minutes to find an effective execution path before the server actually executes the query. The server uses either a cost-based optimizer or a rule-based optimizer to figure out the best execution path for the query.

II. Tips To improve performance

Hints are instructions that you include in your SQL statement for the optimizer. Using hints, you can specify join orders, types of access path, indexes to be used, and the intended optimization goals. You must place the hints within /*+ <hint> */, and you should place them after the SELECT key word. [1], [2].

Note: If a user doesn't follow the syntax, the optimizer will not prompt the user with a syntax error. Instead, it will treat it as no hint.

A. Optimizer Modes

The optimizer goal determines the overall approach the optimizer takes in determining an execution plan. The following statement returns the rows as soon as it finds a few:

```
SELECT /*+ FIRST_ROWS */ distinct customer_name
FROM customer;
```

However, the following query waits until all the rows are retrieved and sorted before returning them to the client:

```
SELECT /*+ ALL_ROWS */ distinct customer_name
FROM
 customer ORDER BY customer_name;
```

You can set the optimizer modes at the session level or at the query level. PL/SQL procedures that run multiple queries would need the session-level setting.

B. Index Hint

Indexes play a very important role in SQL tuning. Indexes allow the table data to be indexed and organized, which in turn enables faster retrieval. Merely creating an index does not speed up the query execution. You must make sure that the query's execution plan uses the hinted index. In the following query, when the optimizer uses the index hint, it will be forced to use the specified index for the search on last_name:

```
SELECT /*+ index(cust_table_last_name_indx) */
 distinct author_names FROM devx_author_names
WHERE
 author_last_name ='DON%';
```

When you do a explain plan on this query, you will see the optimizer using this index. You can also instruct the optimizer to choose between a subset of indexes using /*+ index(indx1, indx2) */.

Note: Creating the index does not speed up the query execution. The index needs to be analyzed. The syntax for analyzing the index is:

```
analyze index <index_name> compute statistics;
```

C. Join Queries

If the query involves joining two or more tables, the database server provides various hints to speed up the queries. A typical join query involves performing a search of the inner table for each row found in the outer table.

For example, suppose table A has 100 rows and table B has 1,000 rows. Logically, the query would run faster if for each row from table B it did a lookup for a matching row in table A. The opposite join could take as much as 10 times longer to execute.

D. *ORDERED Hint*

The ordered hint instructs the optimizer to join the tables in the order in which they appear in the FROM clause. Even though the optimizer picks a different join order based on the computed statistics, this hint forces the optimizer to overwrite its choice of join order. The syntax is /*+ ORDERED */.

E. *USE_NL Hint*

Use nested loop joins when the subset of data between the two joining tables is small. Because nested loop joins fetch the data as soon as possible, they are the preferred joins when either the data doesn't need to be sorted or you need the queries to return quickly.

For the previous tables A and B example, the inner table (second table) would be table A:

```
SELECT /*+ ORDERED USE_NL(A) */
FROM B, A
Where A.column1 = B.column2;
```

F. *USE_HASH Hint*

This hint is best suited for joining tables that have a large subset of data. A hash join offers better throughput and is best suited for sorting and ordering queries. When the size of the tables is large, the hash table size becomes pretty large and it requires more CPU and memory.

The syntax for the hint is
/*+ USE_HASH (table_name) */.

G. *USE_MERGE Hint*

The merge hint requires that both inputs be sorted on the merge columns, which are defined by the equality (WHERE) clauses of the join predicate. Because each input is sorted, the merge join operator gets a row from each input and compares them. For example, for inner join operations, the rows are returned if they are equal. If they are not equal, whichever row has the lower value is discarded and another row is obtained from that input. This process repeats until all rows have been processed. The syntax for this hint is
/*+USE_MERGE(table_name) */.

III. USE THE SQL TUNING THAT FITS

Before you start tuning the query, understand the tables and their data and apply the hints. The hints you use in the query must be tailored to each query's requirement. No one solution fits all in SQL tuning. The best bet would be to try different hints and time the queries. Select the best hint and do a cost analysis on the hint to make sure that you don't overuse the server resources [3].

Below are some Tips to Improve performance.

1) Understand the data. Look around table structures and data. Get a feel for the data model and how to navigate it.
2) If a view joins 3 extra tables to retrieve data that you do not need, don't use the view!
3) When joining 2 views that themselves select from other views, check that the 2 views that you are using do not join the same tables!
4) Avoid multiple layers of view. For example, look for queries based on views that are themselves views. It may be desirable to encapsulate from a development point of view. But from a performance point of view, you loose control and understanding of exactly how much task loading your query will generate for the system.
5) Look for tables/views that add no value to the query. Try to remove table joins by getting the data from another table in the join.
6) WHERE EXISTS sub-queries can be better than join if can you reduce drastically the number of records in driver query. Otherwise, join is better.
7) WHERE EXISTS can be better than join when driving from parent records and want to make sure that at least on child exists. Optimizer knows to bail out as soon as finds one record. Join would get all records and then distinct them!
8) In reports, most of the time fewer queries will work faster. Each query results in a cursor that Reports has to open and fetch. See Reports Ref Manual for exceptions.
9) Avoid NOT in or NOT = on indexed columns. They prevent the optimizer from using indexes. Use where amount > 0 instead of where amount != 0.
10) Avoid writing where project_category is not null. nulls can prevent the optimizer from using an index.
11) Consider using IN or UNION in place of OR on indexed columns. ORs on indexed columns causes the optimizer to perform a full table scan.
12) Avoid calculations on indexed columns. Write WHERE approved_amt > 26000/3 instead of WHERE approved_amt/3 > 26000.
13) Avoid this: SUBSTR(haou.attribute1,1,LENGTH(':p_otc')) = :p_otc). Consider this: WHERE haou.attribute1 like :p_otc||'%' .
14) Talk to your DBA. If you think that a column used in a WHERE clause should have an index, don't assume that an index was defined. Check and talk to your DBA if you don't find any.
15) Consider replacing outer joins on indexed columns with UNIONs. A nested loop outer takes more time than a nested loop unioned with another table access by index.
16) Consider adding small frequently accessed columns (not frequently updated) to an existing index. This will enable some queries to work only with the index, not the table.
17) Consider NOT EXISTS instead of NOT IN.
18) If a query is going to read most of the records in a table (more than 60%), use a full table scan.
19) Try to group multiple sub queries into one.

IV. Beyond the Simple stuff

- If you want to actually understand what you are doing, here are a few things that you need to start playing with:
- Get into EXPLAIN_PLAN. There are multiple way of doing this. The less user friendly is to simply issue this in SQL*Plus: explain plan set statement_id = 'HDD1' for <Your DML SQL statement>;
- Look at the trace from Oracle Reports. It tells you how much time it spends on each query. With r25: C:\ORANT\BIN\R25RUN32.EXE module=p:\old\bcmtrka1_hdd.rdf userid=opps/opps@new tracefile=p:\trace3.txt trace_opts=(trace_all).
- Use the SQL Trace by issuing an alter session set sql_trace=true; then look at it with TKPROF <something>.trc <something>.lis sort=(EXECPU).

If you remember nothing else.

- Don't apply these guidelines blindly, EXPERIMENT: compare one method to another. Do NOT expect that one trick will work all the time.
- Educate yourself: read, read, read. It SAVES time!.

V. Tips for Avoiding Problematic Queries

These tips provide a solid foundation for two outcomes: making a SQL statements perform better, and determining that nothing else can be done in this regard (i.e., you have done all you can with the SQL statement, time to move on to another area). [4],[5].

The 17 tips are listed below.

1) Avoid Cartesian products

2) Avoid full table scans on large tables

3) Use SQL standards and conventions to reduce parsing

4) Lack of indexes on columns contained in the WHERE Clause.

5) Avoid joining too many tables.

6) Monitor V$SESSION_LONGOPS to detect long running operations.

7) Use hints as appropriate

8) Use the SHARED_CURSOR parameter.

9) Use the Rule-based optimizer if I is better than the Cost-based optimizer.

10) Avoid unnecessary sorting

11) Monitor index browning (due to deletions; rebuild as necessary)

12) Use compound indexes with care (Do not repeat columns)

13) Monitor query statistics

14) Use different tablespaces for tables and indexes (as a general rule; this is old-school somewhat, but the main point is reduce I/O contention)

15) Use table partitioning (and local indexes) when appropriate (partitioning is an extra cost feature)

16) Use literals in the WHERE clause (use bind variables)

17) Keep statistics up to date.

VI. Conclusions

This paper provides some helpful tips related to performance improvements against any relational database. We hope that after reading this paper the reader can modify their design and/or queries within their application that returns optimum result and saves development time.

Acknowledgment

We would like to thank all our lab assistants' timely co-operation and guidance in completing the paper. We greatly acknowledge the management and staff of our organization for their support and for providing lab facilities.

References

[1] Oracle9i Database Performance Guide and Reference. Connie Dialeris Green,2002.

[2] Oracle Enterprise Manager Database Tuning with the Oracle Tuning Pack Release 9.0.1 wiki pedia. Part Number A86647-01.

[3] SQL Performance Tuning Peter Gulutzan, Trudy Pelzer Addison-Wesley Professional, 2003.

[4] Oracle9i Performance Tuning: Optimizing Database Productivity by Hassan Afyouni (Thompson Course Technology, 2004).

[5] Oracle SQL tuning - Tune individual SQL statements *Oracle Tips by* Burleson Consulting.