Explain Plan and SQL Trace the Two Approaches for RDBMS Tuning

Hitesh Kumar Sharma, Mr. S.C. Nelson

University of Petroleum & Energy Studies, Dehardun, Uttarakhand, India
hksharma@ddn.upes.ac.in, cnelson@ddn.upes.ac.in

Probably the best way to determine whether your SQL statements are properly optimized is by using the Oracle SQL Trace facility and the EXPLAIN PLAN command. You can use the SQL Trace facility and the Oracle program TKPROF, which is used to translate trace files, to trace production SQL statements, and gather statistics about those statements. You use SQL Trace to gather information into a trace file; the Oracle program TKPROF formats the trace information into useful, understandable data.

1 Introduction

The EXPLAIN PLAN command is used to display the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. By analysing the execution plan the Oracle optimizer has chosen, and knowing your data and application, you should be able to determine whether the optimizer has chosen the correct execution plan for your application. After using EXPLAIN PLAN, you can rewrite your SQL statements to take better advantage of such things as indexes and hash keys. By analysing the output, you may be able to providehints that the Oracle optimizer can use to take better advantage of your knowledge of your data. By using hints, you may be able to take better advantage of features such as the Oracle Parallel Query option. By the end of this chapter, you should be able to execute SQL statements using both SQL Trace and EXPLAIN PLAN and be able to analyse the output from these statements. You should also understand the value of registering applications for later use when tracking performance problems. These Oracle options can greatly improve the stability and performance of your system.

2. Tuning Considerations

The data warehouse is tuned to allow several large processes to run at maximum throughput. There is usually no concern for response times. We may have to tune both Oracle and the server operating system. The following sections look first at Oracle and then at the server operating system.

3. Server OS Tuning

We may have to tune the server OS to provide for a large number of processes (if we are using the Parallel Query option) and optimal I/O performance. Some of the things we may have to tune in the server OS are listed here; remember that some OSes may not require any tuning in these areas:

- **Memory.** Tune the system to reduce unnecessary memory usage so that Oracle can use as much of the system's memory as possible for the SGA and server processes. We may also need significant amounts of memory for sorts.
- **Memory enhancements.** Take advantage of 4M pages and ISM, if they are available. Both features can improve Oracle performance in a data warehouse environment.
- **I/O.** If necessary, tune I/O to allow for optimal performance and use of AIO.
- **Scheduler.** If possible, turn off preemptive scheduling and load balancing. In a data warehousing system, allowing a process to run to completion (that is, so that it is not preempted) is beneficial.

- **Cache affinity.** We may see some benefits from cache affinity in a data warehousing system because the processes tend to run somewhat longer. The server operating system is mainly a host on which Oracle does its job. Any work done by the operating system is essentially overhead for Oracle. By optimizing code paths and reducing OS overhead, we can enhance Oracle performance.

4. Hardware Enhancements

For a data warehouse, several hardware enhancements can improve performance. These hardware enhancements can be beneficial in the area of CPU, I/O, and network, as described in the following sections.

4.1. CPU Enhancements

Enhancing the CPUs on our SMP or MPP system can provide instantaneous performance improvements, assuming that we are not I/O bound. The speed of CPUs is constantly being improved as are new and better cache designs. For SMP or MPP machines, the process of enhancing the CPU may be as simple as adding an additional CPU board. Before we purchase an additional processor of the same type and speed, however, consider upgrading to a faster processor. For example, upgrading from a 66 MHz processor to a 133 MHz processor may provide more benefit than purchasing an additional 66 MHz CPU with the added benefit that we now have the option of adding more 133 MHz CPUs. Because of the complexity and run time required by these queries, we can benefit from more and faster CPUs. SMP and MPP computers provide scalable CPU performance enhancements at a fraction of the cost of another computer. When upgrading our processors or adding additional processors, remember that our I/O and memory needs will probably increase along with the CPU performance. Be sure to budget for more memory and disk drives when we add processors.

4.2. I/O Enhancements

We can enhance I/O by adding disk drives or purchasing a hardware disk array. The data warehouse can benefit from the disk striping available in both hardware and software disk arrays.

Using Oracle data file striping can also help the performance of our data warehouse.

If our system performs only one query at a time and we are not taking advantage of the Oracle Parallel Query option, we may not see a benefit from a hardware or software disk array. In this specific case, we do not recommend OS or hardware striping; we should use traditional Oracle striping. Because we are executing only one query at a time without using the Parallel Query option, the I/Os to the data files are purely sequential on the table scans. This scenario is somewhat rare; any variance from “pure table scans” results in degraded performance. Hardware and software disk arrays have the added benefit of optional fault tolerance. We should first choose the correct fault tolerance for our needs and then make sure that we have sufficient I/O capabilities to achieve the required performance level. If we use fault tolerance, we will most likely have to increase the number of disk drives in our system. Another benefit of hardware disk arrays is caching. Most disk arrays on the market today offer some type of write or read/write cache on the controller. The effect of this cache is to improve the speed of writing to the disk; the cache also masks the overhead associated with fault tolerance. If our queries often perform table scans, we may see good improved performance with disk controllers that take advantage of read-ahead features. Read-ahead occurs when the controller detects a sequential access and reads an entire track (or some other large amount of data) and caches the additional data in anticipation of a request from the OS. Unlike an OLTP system in which this is just wasted overhead, in the data warehouse where we are performing DSS queries, it is likely

that we will need that data soon; if we do, it will be available very quickly. Enhancements to the I/O subsystem almost always help in a data warehouse environment because large amounts of data are accessed. Be sure that we have a sufficient number of disk drives, properly configured. An I/O bottleneck is usually difficult to work around. As with all types of systems, a well-tuned application is very important.

5. Fault Tolerance Consideration

Because the data warehouse contains so much data, we can take one of two approaches to dataprotection:

- **Protect everything.** Because there is so much data and so many disks in use, everything must be protected. The large number of disks in use increases the possibility of a disk failure. The massive amount of data increases the time needed for backup and recovery.
- **Conserve cost.** Because there are so many disks involved, it may be cost prohibitive to use RAID-1 or disk mirroring. When we mirror the disks, we double the number of disks.

In a data warehousing system, a good compromise is to use a fault tolerant method such as RAID-5 for the data files. We can be somewhat selective and use RAID-1 on volumes with heavy update activity and RAID-5 on volumes with more read activity. Remember that the performance penalty for RAID-5 is only on writing; we can achieve excellent read performance from RAID-5.

6. Hardware Considerations

When choosing hardware to use for a data warehousing system, consider these factors:

- **Low user load.** Not many concurrent processes/threads simultaneously access the systemunless we take advantage of the -Parallel Query option.
- **High I/O load.** I/Os are concurrent and heavy, with mostly random I/O.
- **Huge amounts of data.** Data warehousing systems typically involve massive amounts of data. We must make sure that our system can support the high volumes of data we will be using.
- **Low network traffic during runtime, possibly high during load.** During the execution of typical decision support queries against our data warehouse, there is very little network activity. When data is being loaded or updated from other sources (possibly our OLTP systems), the network activity can be quite high.

If we can take advantage of the Oracle Parallel Query option, many different processes will use the machine at once; an SMP or MPP machine should scale very well. Because an SMP architecture uses CPUs based on the processes that are available to be run, if we always have a runnable process available for each CPU, we should see good scaling by adding additional processors. With an MPP machine, we see a similar effect but on a much larger scale. Because there is much random access to the disks, we can benefit from a disk array. We prefer hardware striping to OS striping because hardware striping does not incur any additional overhead for the operating system and does not take up valuable CPU cycles. If hardware striping is not available, OS striping is adequate. Network traffic may or may not be an issue to our data warehousing system. If necessary, segment the network or add faster network hardware. A network bottleneck is an easy problem to solve: simply add more and faster hardware.

7. SQL Trace

The SQL Trace facility and the Oracle program TKPROF are designed to give performance information about individual SQL statements. You can use this information to determine the characteristics of those statements. You can enable SQL Trace for a session or for an entire instance. Of course, because this facility gathers an abundance of information about SQL statement functionality and performance, SQL Trace has an effect on the performance of the system. If you use SQL Trace on a single session, the effect is fairly minimal, but if you use SQL Trace on an entire instance, you will see a substantial effect on the performance of the system. Avoid running SQL Trace on an entire instance for this reason.

7.1. SQL Trace Initialization

Before you run SQL Trace, you must make sure that certain Oracle initialization parameters are set:

Parameter Description

Parameter	*Description*
TIMED_STATISTICS	Setting TIMED_STATISTICS to TRUE enables SQL and some of the dynamic performance tables to c timed statistics such as CPU and elapsed times. E timed statistics incurs significant overhead becaus Oracle operations are now being timed; avoid thi parameter except when necessary.
MAX_DUMP_FILE_SIZE	Specifies the maximum size of trace file dumps in blocks. Set this fairly low to avoid filling up the fi system with trace files. If the SQL Trace output fi being truncated, increase this value.
USER_DUMP_DEST	This parameter specifies the destination for the tr The default destination is the same as for system c on your OS.

7.2. Controlling SQL Trace

You can enable the SQL Trace facility on a per-session basis or for the entire instance. The following sections explain how to enable and disable SQL Trace for both of these cases.

Enable SQL Trace for a Session

To enable SQL Trace for a session, use this Oracle command:

```
ALTER SESSION
SET SQL TRACE = TRUE;
```

Alternatively, you can use the Oracle procedure:

```
RDBMS_SESSION.SET_SQL_TRACE.
```

To enable SQL Trace for a session other than your own, you can use the Oracle procedure RDBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION with the arguments SID, Serial#, and TRUE. To determine the values for SID and Serial#, use the following SQL statements:

```
SQL> SELECT sid, serial#,
osuser
2 FROM v$session
3 WHERE osuser = 'Ed Whalen';
SID SERIAL# OSUSER
---------- ------- ---------
---
7 4 Ed Whalen
```

To turn SQL Trace on for that session, use the Oracle stored procedure as follows:

```
SQL> EXECUTE
RDBMS_system.set_sql_trace_in
_session(7,4,TRUE);
```

PL/SQL procedure successfully completed.

Disable SQL Trace for a Session

To disable SQL Trace for a session, use this Oracle command:

```
ALTER SESSION
SET SQL TRACE = FALSE;
```

The SQL Trace facility is also disabled when your session disconnects from Oracle. To disable SQL Trace for a session other than your own, use the Oracle procedure RDBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION with the arguments SID, Serial#, and FALSE as shown here:

```
SQL> EXECUTE
RDBMS_system.set_sql_trace_in
_session(7,4,FALSE);
```

PL/SQL procedure successfully completed.

Enable SQL Trace for an Instance
To enable SQL Trace for your instance, set the Oracle initialization parameter SQL_TRACE to TRUE. Doing so enables SQL Trace for all users of this instance for the duration of the instance.

Disable SQL Trace for an Instance
The SQL Trace facility cannot be disabled for the entire instance without shutting down the Oracle instance and setting the Oracle initialization parameter SQL_TRACE to FALSE. Alternatively, you can remove the parameter because its default value is FALSE. When SQL Trace is enabled for the entire instance, it is still possible to disable it on a per session basis. You can disable SQL Trace on a per-session basis with the SQL statement shown in the preceding section.

7.3. SQL Trace Functionality
Once SQL Trace is enabled, it gathers the following information:

- **Parse, execute, and fetch counts.** These counts can give you vital information about the efficiency of the SQL statements.
- **CPU and elapsed times.** This information can tell you which statements take the most time to execute.
- **Physical and logical reads.** This information can help you determine the effectiveness of the database buffer pool.
- **Number of rows processed.** This information can be used as an indication that more rows are being processed than you expected, thus indicating a problem.
- **Library cache misses.** This information can show you the effectiveness of the shared SQL area and how well you are reusing already parsed SQL statements.

SQL Trace puts this information into a trace file in an unreadable form. You then use the Oracle program TKPROF to format the trace information into useful, understandable data.

7.4. Interpreting SQL Trace
This section looks at some of the statistics available from SQL Trace and how to interpret them. For each SQL statement executed, SQL Trace provides the following information:

Parameter	*Description*
count	Number of times the OCI procedure was executed. (The OCI interface is the standard set of calls used to access the Oracle database.)
cpu	CPU time in seconds executing. This value is the amount of time Oracle used to process the statement.
elapsed	Elapsed time in seconds executing. This value is equivalent to the user's response time.
disk	Number of physical reads of buffers from disk. This value tells you how many reads actually missed the buffer cache and had to go to physical disk.
query	Number of buffers gotten for consistent read. This value represents the number of buffers retrieved in consistent mode. Consistent mode guarantees consistent reads throughout the transaction; it is used for most queries.
current	Number of buffers gotten in current mode (usually for update). In current mode, the data blocks gotten reflect the value at that instant in time.
rows	Number of rows processed by the fetch or execute call. This value gives you an idea of how many instructions have been executed.

By looking at each of these parameters, you can get an idea of how your SQL statements are being processed and which statements are taking the most time. By analysing which statements are taking the longest, you may be able to find some inefficiencies you can correct. The SQL Trace facility was enabled in a session by using this Oracle command:

```
EXECUTE
RDBMS_system.set_sql_trace_in
_session(7,3,TRUE);
```

In another session (with SID = 7 and Serial# = 3), the following SQL statement was executed:

```
SELECT
SUBSTR(dogname,1,20)      "Dog
Name",
SUBSTR(description,1,20)
"Breed",
SUBSTR(owner,1,20) "Owner"
FROM
dogs, breeds
WHERE
dogs.breed = breeds.breed
```

```
ORDER BY
dogs.breed;
```

The SQL Trace facility was later disabled using this Oracle command from the fist session:

```
EXECUTE
RDBMS_system.set_sql_trace_in
_session(7,3,FALSE);
```

Following the execution of the SQL statements, the trace file was translated by running TKPROF as follows:

```
tkproforclshad.trctrace.out
sys=no explain=ed/ed
```

In this syntax, the following are true:
`orclshad.trc` Trace file generated by SQL Trace trace.out Where I want the output to go sys=no Indicates that no SYS or recursive SQL statements should be printed explain=ed/ed Specifies that I also want to generate EXPLAIN PLAN output TKPROF generated the output file shown in Listing

The EXPLAIN PLAN Command

The EXPLAIN PLAN command shows you the execution plan that the Oracle optimizer has chosen for your SQL statements. With this information, you can determine whether the Oracle optimizer has chosen the correct execution plan based on your knowledge of the data and the application. You can also use EXPLAIN PLAN to determine whether any additional optimization should be done to your database (for example, the addition of an index or the use of a cluster). The EXPLAIN PLAN command is used to display the execution plan chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements. After using EXPLAIN PLAN, you can rewrite your SQL statements and see whether the new SQL statement is better optimized than theoriginal statement. By analysing the output, you may be able to provide hints that the Oracle optimizer can use to take better advantage of the data.

EXPLAIN PLAN Initialization

When you run SQL statements with the EXPLAIN PLAN command, the output of EXPLAIN PLAN is put into a table with the default name plan_table. You must create this table before you can run EXPLAIN PLAN. The table can be created in one of two ways:

Using the UTLXPLAN.SQL script provided by Oracle.
Creating the plan_table table by hand.

The plan_table table is define as follows:

```
SQL> describe plan_table
Name Null? Type
----------------------------
-- -------- ----
STATEMENT_ID VARCHAR2(30)
TIMESTAMP DATE
REMARKS VARCHAR2(80)
OPERATION VARCHAR2(30)
OPTIONS VARCHAR2(30)
OBJECT_NODE VARCHAR2(128)
OBJECT_OWNER VARCHAR2(30)
OBJECT_NAME VARCHAR2(30)
OBJECT_INSTANCE NUMBER(38)
OBJECT_TYPE VARCHAR2(30)
OPTIMIZER VARCHAR2(255)
SEARCH_COLUMNS NUMBER(38)
ID NUMBER(38)
PARENT_ID NUMBER(38)
POSITION NUMBER(38)
OTHER LONG
```

You do not have to name the table plan_table. You can direct EXPLAIN PLAN to use a table of another name if you want.

Invoking EXPLAIN PLAN

Invoke the EXPLAIN PLAN command with the following Oracle command sequence:

```
EXPLAIN PLAN
```

```
SET STATEMENT_ID = 'Testing
EXPLAIN PLAN'INTO plan_table
FOR
SQL Statement;
```

STATEMENT_ID should reflect the statement's function so that you can recognize it at a later time. The plan_table parameter is the name of the table you created as described in the preceding section. If the INTO clause is omitted, the command defaults to the name plan_table.

Here is an example of a completed command:

```
SQL> EXPLAIN PLAN
2 SET STATEMENT_ID = 'Testing
EXPLAIN PLAN'
3 INTO plan_table
4 FOR
5 SELECT
6  SUBSTR(dogname,1,20)  "Dog
Name",
7     SUBSTR(breed_name,1,20)
"Breed",
8 SUBSTR(owner,1,20) "Owner"
9 FROM
10 dogs, breeds
11 WHERE
12 dogs.breed = breeds.breed
13 ORDER BY
14 dogs.breed;
```

Explained.

The results of the EXPLAIN PLAN are written into the table plan_table. The following sectionexplains how to retrieve the information in that table.

Extracting EXPLAIN PLAN Results

The output of EXPLAIN PLAN is written to the table specified in the EXPLAIN PLAN command (by default, to the table named plan_table). You must extract this information in order to look at the results of EXPLAIN PLAN. The results can be displayed with a query such as this:

```
SELECT          SUBSTR(LPAD('
',2*(LEVEL-
1))||operation,1,30)
||' '||SUBSTR(options,1,15)
||'
'||SUBSTR(object_name,1,15)
||'  '||SUBSTR(DECODE(id,  0,
'Cost = '||position),1,12)
"Statement Execution Plan",
SUBSTR(optimizer,   1,   10)
"Optimizer"
FROM
plan_table
START WITH
id = 0 AND statement_id =
'Testing EXPLAIN PLAN'
CONNECT BY PRIOR
id = parent_id
AND
statement_id    =    'Testing
EXPLAIN PLAN';
```

This query results in the following output:

```
Statement Execution Plan Optimizer
----------------------------
------SELECT STATEMENT Cost =
CHOOSE
MERGE JOIN
SORT JOIN
TABLE ACCESS FULL BREEDS
SORT JOIN
TABLE ACCESS FULL DOGS
6 rows selected.
```

If the optimizer had chosen a cost-based approach, the cost of the query would have been reflected in the first line of the optimization plan. Any features such as parallel query are also reflected here. With this information, you can tell whether your SQL statements take advantage of indexes, clusters, or hash clusters. If you use EXPLAIN PLAN, you can see precisely how your SQL statement is being executed and what effect any changes you make to the SQL statements have on the execution plan. If you change your SQL statements to take advantage of an index or a cluster, for example, you can see an immediate improvement. EXPLAIN PLAN output is ideal for pointing out your execution plan and may indicate that where you thought

you were takingadvantage of an index, you actually were not.

8. Registering Applications

When you register an application, the name and the actions performed by that application are stored in the database to assist with debugging and performance tuning efforts. When an application is registered, its name and actions are recorded in the V$SESSION and V$SQLAREA views. This information can be used later to track problems. To register an application, use the following procedures, available in the RDBMS_APPLICATION_INFO package:

Procedure Description

SET_MODULE Used to set the name of the module currently being run.

SET_ACTION Used to set the name of a certain action currently being performed.

SET_CLIENT_INFO Used to set up information for the client information field.

READ_MODULE Reads the current values of the module and action fields for the current session.

READ_CLIENT_INFO Reads the current client information field for the currently running session.

By registering the application, you can track many different parameters. Some of the values available through V$SQLAREA are given here:

- Memory used
- Number of sorts
- Number of executions
- Number of loads
- Number of parse calls
- Number of disk reads
- Number of buffer gets
- Number of rows processed

These parameters can provide valuable information when you are trying to debug various modules within your application. The information is enhanced by the addition of actions, which can further identify sections of your application.

9. Conclusion

Determining whether your SQL statements are properly optimized can be as important as anything else you can do to tune your system. An improperly tuned SQL statement can nullifyany work you have done to optimize the database system. A well-tuned server system thatis handling hundreds or thousands of unnecessary SQL statements can be perceived to havepoor performance when, in reality, there is just an abundance of excess work being done.

The Oracle SQL Trace facility and the EXPLAIN PLAN command can be valuable tools in debugginginefficient SQL code. The SQL Trace facility and its companion program TKPROF cangive valuable information into such areas as these

References

[1] Hitesh Kumar Sharma, Aditya Shastri, Ranjit Biswas, “A Framework for Automated Database TuningUsing Dynamic SGA Parameters and Basic Operating System Utilities”, Database Systems Journal”, Academy of Economic Studies-Bucharest, Romania.

[2] Hitesh Kumar Sharma, Sandeep Kumar, Sambhav Dubey, Pawan Gupta, “Auto-selection and management of dynamic SGA parameters in RDBMS”, Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference.

[3] Hitesh Kumar Sharma, Aditya Shastri, Ranjit Biswas, “SGA Dynamic Parameters: The Core Components of Automated Database Tuning”, Database Systems Journal”, Academy of Economic Studies-Bucharest, Romania.

[4] S. Elnaffar, W. Powley, D. Benoit, and P. Martin, “Today’s DBMSs: How

Autonomic are They?", Proceedings of the 14th DEXA Workshop, Prague, 2003, pp. 651-654.

[5] D. Menasec, Barbara, and R. Dodge, "Preserving Qos of E-Commerce Sites through Self-Tuning: A Performance Model Approach", Proceedings of 3rd ACM-EC Conference, Florida, 2001, pp.224-234.

[6] D. G. Benoit, "Automated Diagnosis and Control of DBMS resources", EDBT Ph.D Workshop, Konstanz, 2000.

[7] B. K. Debnath "SARD: A Statistical Approach for Ranking Database Tuning Parameters" 2007.

[8] K. P. Brown, M. J. Carey, and M. Livny, "Goal-Oriented Buffer Management Revisited", Proceedings of ACM SIGMOD Conference, Montreal, 1996, pp. 353-364.

[9] P. Martin, H. Y. Li, M. Zheng, K. Romanufa, and W. Poweley, "Dynamic Reconfiguration Algorithm: Dynamically Tuning Multiple Buffer Pools", Proceedings of 11th DEXA conference, London, 2002, pp.92-101.

[10] P. Martin, W. Powely, H. Y. Li, and K. Romanufa, "Managing Database Server Performance to Meet Qos Requirements in Electronic Commerce System", International Journal of Digital Libraries, Vol. 8, No. 1, 2002, pp. 316-324.

[11] S. Duan, V. Thummala, S. Babu, "Tuning Database Configuration Parameters with iTuned", *VLDB '09,* August 2428, 2009, Lyon, France.

[12] H. K. Sharma, A. Shastri, R. Biswas " Architecture of Automated Database Tuning Using SGA Parameters" , *Database Systems Journal* vol. III, no. 1/2012.

[13] A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era", IBM Systems Journal, Vol. 42, No. 1, 2003, pp. 5-18.

[14] H. K. Sharma, A. Shastri, R. Biswas "A Framework for Automated Database Tuning Using Dynamic SGA Parameters and Basic Operating System Utilities", *Database Systems Journal vol. III, no. 4/2012.*

[15] P. S. Yu, M. S. Chen, H. U. Heiss, S. H. Lee, "On Workload Characterization of Relational Database Environments", IEEE Transactions on Software Engineering", Vol. 18, No. 4, 1992, pp.347-355.

[16] J. Seok Oh, S. Ho Lee, " Resource Selection for Autonomic Database Tuning", Korea Research Foundation .

Dr. Hitesh Kumar Sharma: Author is an Assistant Professor (Senior Scale) in University of Petroleum& Energy Studies, Dehradun. He has published 40+ research papers in International Journal and 10+ research papers in National Journals.

Christalin Nelson. S: Author is an Assistant Professor (Selection Grade) in University of Petroleum& Energy Studies, Dehradun. He has published 40+ research papers in International Journal and 12 research papers in National Journals. He is Head of Department of Analytics.