Journal Homepage: - www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/14058
DOI URL: http://dx.doi.org/10.21474/IJAR01/14058

RESEARCH ARTICLE

A PROACTIVE APPROACH FOR DATABASE PERFORMANCE TUNING

Rajan B.
Performance Engineering Service Line, HCL Technologies.

Manuscript Info

Manuscript History
Received: 10 November 2021
Final Accepted: 14 December 2021
Published: January 2022

Keyword:-
Database Performance Tuning Query Tuning

Abstract

The Oracle database is a trendsetter and highly tunable software product. It's very flexible to allow you to make small adjustments that affect the database performance. By tuning you can tailor its performance to best meet your needs. Performance tuning cannot be performed best after a system is put into production. To achieve performance targets of response time, and throughput, one must proactively analyze database design, implementation and tune ahead in the life cycle.

Introduction:-

Oracle database tuning is a wide area and one who provides proactive solution would be successful. Performance tuning is a large and complex area. There is a common question raised by a DBA that "*Where to start and what to do for tuning the performance?*" Here we go a proactive method on improving the performance of your database. Please do not wait until someone reports a problem about slow or poor performance of an application. A most common mistake made is, to spend effort for increasing the application performance in limited area like SQL tuning. A DBA must be aware of other tuning areas, which needs attention and a defined tuning methodology, covering each area, other than SQL tuning. SQL tuning can be a very successful method for increasing performance, but it must be used as a part of a larger and more effective performance tuning methodology.

Here we are focusing on three major areas to improve the Oracle performance by a proactive approach.

1. Parameter Initialization
2. Space Management
3. SQL Tuning

I. Parameter Initialization

Across technologies, as configuration changes is critical factor for influencing better performance, Oracle database instance needs initialization parameter configuration change in the *init* file. This configuration changes, influences the behavior of running database which includes the performance as well.

The simplest way to manage instance memory is to allow the Oracle database instance to automatically manage and tune it for you. To do that, target memory size initialization parameter (MEMORY_TARGET) and optionally a maximum memory size initialization parameter (MEMORY_MAX_TARGET) needs to be configured. Oracle instance automatically tunes to the target memory size, redistributing memory as needed between the SGA and PGA. Since, the target memory initialization parameter is dynamic, it can be changed anytime without restarting the database. The maximum memory size serves as an upper limit so that it does allow setting the target memory size, above maximum memory size. As this configuration change automatically gives enough memory for Oracle instance to operate, it does not require explicit manual change of increasing the total instance memory size. Certain SGA

Corresponding Author:- Rajan B.
Address**:-** Performance Engineering Service Line, HCL Technologies.

components, cannot easily shrink or must remain at a minimum size, the database instance prevents from setting up the target memory size too low.

Find below the three steps to calculate the minimum value for MEMORY_TARGET as follows:
(i) Determine the current sizes of SGA_TARGET and PGA_AGGREGATE_TARGET by entering the following SQL*Plus command:

```
SHOW PARAMETER TARGET
```

SQL plus displays the values of all initialization parameters with the string TARGET in the parameter name

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----
archive_lag_target                   integer     0
db_flashback_retention_target        integer     1440
fast_start_io_target                 integer     0
fast_start_mttr_target               integer     0
memory_max_target                    big integer 0
memory_target                        big integer 0
pga_aggregate_target                 big integer 90M
sga_target                           big integer 272M
```

(ii) Run the following query to determine the maximum instance PGA allocated since the database was started:

```
select value from v$pgastat where name='maximum PGA allocated';
value in MB
120
```

(iii) Compute the maximum value between the query result from step 2b and PGA_AGGREGATE_TARGET. Add SGA_TARGET to this value.

```
memory_target = sga_target + max(pga_aggregate_target,
                                 maximum PGA allocated)
```

From the above results:
Memory_Target = 272 + max (90,120)
= 272 + 120 = 392 M

Hence the minimum MEMORY_TARGET should be 392 M based on the above results.

The below tables (table 1 & 2) has the necessary parameters which needs to be taken care during the installation of the oracle database.

Table I:- Necessary Initialization parameters.

Parameter	Description	Recommendation
DB_NAME	Name of the database	This should match the ORACLE_SID environment variable.
DB_DOMAIN	Location of the database	This should be in Internet dot notation
OPEN_CURSORS	Limit on the maximum number of cursors (active SQL statements) for each session	The setting is application-dependent; 500 is recommended
CONTROL_FILES	Configuration files	Set to contain at least two files on different disk drives to prevent failures from control file loss.
DB_FILES	Number of files assign to the database	Set to the maximum number of files that can assigned to the database.

Table II:- Important Initialization parameter which influences the performance

Parameter	Description	Recommendation
COMPATIBLE	Specifies the release with which the Oracle database must maintain compatibility.	If application designed for a specific release and you are actually installing a later release, then you might want to set this parameter to the version of the previous release.
DB_BLOCK_SIZE	Sets the size of the blocks stored in the database files and cached in the SGA.	The range of values depends on the operating system, but it is typically 8192 for transaction processing systems and higher values for database warehouse systems.
SGA_TARGET	Specifies the total size of all SGA components.	If SGA_TARGET is specified, then the buffer cache DB_CACHE_SIZE, JAVA_POOL_SIZE, LARGE_POOL_SIZE, SHARED_POOL_SIZE memory pools are automatically sized.
PGA_AGGREGATE_TARGET	Specifies the target aggregate PGA memory available to all server processes attached to the instance.	
PROCESSES	Sets the maximum number of processes that can be started by that instance.	This is the most important primary parameter to set, because many other parameter values are deduced from this.
SESSIONS	This is set by default from the value of processes.	If you are using the shared server, then the deduced value is likely to be insufficient.
UNDO_MANAGEMENT	Specifies the undo space management mode used by the database.	The default is AUTO. If unspecified, the database uses AUTO.
UNDO_TABLESPACE	Specifies the undo tablespace to be used when an instance starts.	

Few Oracle Hidden Parameters

In addition, to the above parameters, there are few Oracle hidden Parameters that help to boost the performance. The below parameters, take advantage of hardware – specific performance features:

```
_in_memory_undo=false
_cursor_cache_frame_bind_memory = true
_db_cache_pre_warm = false
_in_memory_undo=false
_check_block_after_checksum = false
_lm_file_affinity
```

II. Space Management

When we talk about the space management in Oracle performance tuning, the following three parameters needs to be taken care

1. Configuring Undo Space
2. Sizing Redo Log Files
3. Creating Temporary Tablespace

1. Configuring Undo Space:
Oracle needs undo space to keep information for read consistency, for recovery, and for actual rollback statements.

Oracle provides automatic undo management, which completely automates the management of undo data. A database running in an automatic undo management mode transparently creates and manages undo segments. Oracle Corporation strongly recommends using automatic undo management, because it significantly simplifies database management and removes the need for any manual tuning of undo (rollback) segments.

Adding the UNDO TABLESPACE clause in the CREATEDATABASE statement sets up the undo tablespace. Set the UNDO_MANAGEMENT initialization parameter to AUTO to operate your database in automatic undo management mode.

The V$UNDOSTAT view contains statistics for monitoring and tuning undo space. Using this view, Oracle can better estimate the amount of undo space required for the current workload. Oracle also uses this information to help tune undo usage in the system. The V$ROLLSTAT view contains information about the behavior of the undo segments in the undo tablespace.

2. Sizing Redo Log Files:
The size of the redo log files influence the performance, because the behavior of the database writer, and archived processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Undersized log files increase checkpoint activity and reduce performance.

Although the size of the redo log files does not affect LGWR performance, it can affect DBWR and checkpoint behavior. Checkpoint frequency is affected by several factors, including log file size and the value of the FAST_START_MTTR_TARGET initialization parameter. If the FAST_START_MTTR_TARGET parameter is set to limit the instance recovery time, Oracle automatically tries to checkpoint as frequently as necessary. Under this condition, the size of the log files should be large enough to avoid additional checkpoints due to the under sized log files. The optimal size can be obtained by querying the OPTIMAL_LOGFILE_SIZE column from the V$INSTANCE_RECOVERY view. The below picture shows the REDO LOG structure with two disks

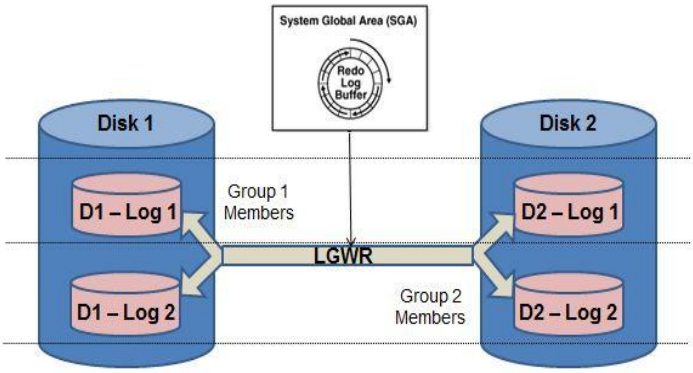

Fig. 1:- Redo Log Files.

Planning the Redo Log files:
1. Multiplexing Redo Log Files
2. Placing Redo Log Members on Different Disks
3. Planning the Size of Redo Log Files
4. Planning the Block Size of Redo Log Files
5. Choosing the Number of Redo Log Files
6. Controlling Archive Lag

It may not always be possible to provide a specific size recommendation for redo log files, but redo log files in the range of a hundred megabytes to a few gigabytes are considered reasonable. Size your online redo log files according to the amount of redo your system generates. A rough guide is to switch logs, once every twenty minutes.

3. Creating Temporary Tablespace:
Appropriately configuring the temporary tablespace helps optimize disk sort performance. Temporary tablespaces can be dictionary-managed or locally managed. Oracle Corporation recommends the use of locally managed temporary tablespaces with a *UNIFORM* extent size of 1 MB.

You should monitor temporary tablespace activity to check how many extents are being allocated for the temporary segment. If an application extensively uses temporary tables, as in a situation when many users are concurrently using temporary tables, the extent size could be set smaller, such as 256K, because every usage requires at least one extent. The EXTENTMANAGEMENTLOCAL clause is optional for temporary tablespaces because all temporary tablespaces are created with locally managed extents of a uniform size. The default for SIZE is 1M.

III. SQL Tuning
SQL tuning is the iterative process of improving SQL statement performance to meet specific, *measurable and achievable goals*. SQL tuning implies fixing problems in deployed applications. In contrast, application design sets the security and performance goals before deploying an application.

After identifying application problem, the below standard performance tuning implies
1. Reduce user response time
2. Improve throughput

SQL tuning involves the following basic steps:
1. Identifying high load or top SQL statements
2. Verifying that the execution plans produced by the query optimizer for these statements perform reasonably
3. Implementing corrective actions to generate better execution plans for poorly performing SQL statements

There are many areas to discuss when we talk about SQL tuning. As we focused on proactive tuning approach there are few things that need to be taken care during development phase itself.
1. Index tuning
2. Table partitioning
3. Scalar sub-query
4. Hints usage
5. Session tuning

1. Index tuning is the major in proactive performance tuning to improve the speed of SQL queries. Oracle includes many new indexing algorithms that dramatically increase the speed of the queries. There are below indexing parts which helps to increase the speed of queries.
   a. Bitmap index
   b. Function based index

a. ***Bitmap Index:*** Oracle bitmap indexes are very different from standard b-tree indexes. In bitmap structures, a two-dimensional array is created with one column for every row in the table being indexed. Each column represents a distinct value within the bitmapped index. This two-dimensional array represents each value within the index multiplied by the number of rows in the table. At row retrieval time, Oracle decompresses the bitmap into the RAM data buffers so it can be rapidly scanned for matching values. These matching values are delivered to Oracle in the form of a Row-ID list, and these Row-ID values may directly access the required information.

The real benefit of bitmapped indexing occurs when one table includes multiple bitmapped indexes. Each individual column may have low cardinality. The creation of multiple bitmapped indexes provides a very powerful method for rapidly answering difficult SQL queries.

For example, assume there is an insurance company database with numerous low-cardinality columns such as customer_type, insurance_made_year etc. and each column contains less than 100 distinct values by themselves and a b-tree index would be fairly useless in a database of 20 million insurers. However, combining these indexes together in a query can provide blistering response times a lot faster than the traditional method of reading each one of the 20 million rows in the base table. For example, assume we wanted to find Male Senior Citizen placed an insurance in 2012:

Select name, Insurnace_no from Insurance where cust_type = 'Sr_Ctz' and Gender = 'M' and year = 2012;

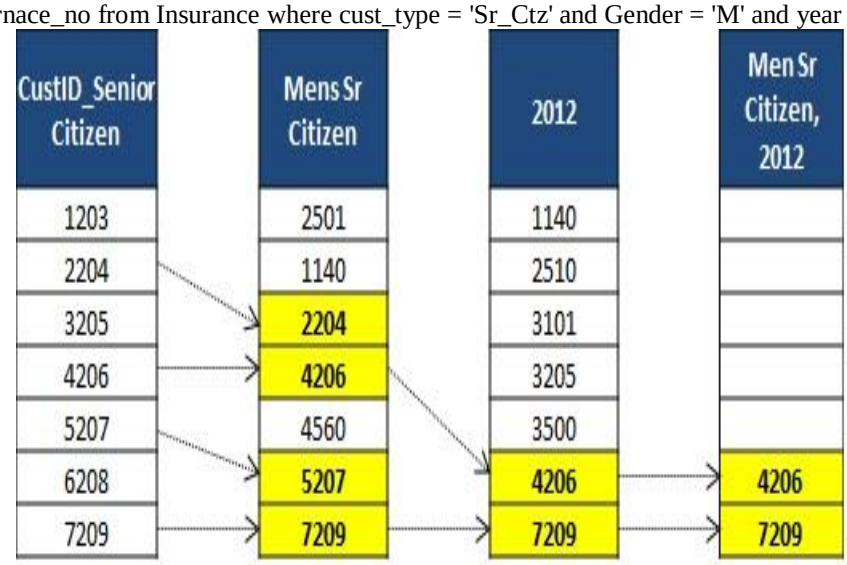

Fig. 2:- Bitmap Index Flow.

b. Function based Index: One of the most important advances in Oracle indexing is the introduction of function-based indexing. Function-based indexes allow creation of indexes on expressions, internal functions, and user-written functions in PL/SQL and Java. Function-based indexes ensure that the Oracle designer is able to use an index for its query.

Prior to Oracle8, the use of a built-in function would not be able to match the performance of an index. Consequently, Oracle would perform the dreaded full-table scan. Example now you can do the following:

```
Create index emp_ix on emp (upper(name));... where upper(name) like 'SMITH%'
```

Now we have a way to get at the function modified data using an index. This is a benefit that makes this technique really easy, that you can index expressions on the data. The logic being used in the where clause pretty much matches the expression in the index. This means you can enable indexed access to data that previously you could not. Consider questions like these:

(i) *how do I find rows where values are null*
select * from emp where release_date is null
create index emp_ix on emp (decode(release_date,null,1,null));
select * from emp where decode(release_date,null,1,null) = 1;

(ii) *how do I find rows based on calculations*
select * from line_item where nvl(total,0) - nvl(discount,0) - nvl(mgr_reduction,0) = 0
create index line_item_ix on line_item (nvl(total,0) - nvl(discount,0) - nvl(mgr_reduction,0));

(iii) *how do I convert a range scan to an exact match query*
select * from line_item where nvl(total,0) - nvl(discount,0) - nvl(mgr_reduction,0) > 0
create index line_item_ix on line_item (decode(sign(nvl(total,0) - nvl(discount,0) - nvl(mgr_reduction,0)),1,1,null)));
select * from line_item where decode(sign(nvl(total,0) - nvl(discount,0) - nvl(mgr_reduction,0)),1,1,null)) = 1;

It would offer that function based indexes tend to be special purpose in nature. Its support specific pieces of code, that needs them and not to try use them as generic performance enhancers. In numerous situations, function based indexes yield stellar enhancements in performance.

2. Table Partitioning: Table partitioning is a divide-and-conquer approach to improving Oracle maintenance and SQL performance. Anyone with un-partitioned databases over 500 gigabytes is courting disaster. Table partitioning will increase the performance by following:

(a) ***Disk load balancing:*** Table and index partitioning allows to segregate portions of very large tables and indexes onto separate disk devices, thereby improving disk I/O throughput and ensuring maximum performance.

(b) ***Improved query speed:*** The Oracle optimizer can detect the values within each partition and access only those partitions that are necessary to service the query. Since each partition can be defined with its own storage parameters, the Oracle SQL optimizer may choose a different optimization plan for each partition.

(c) ***Faster Parallel query:*** The partitioning of objects also greatly improves the performance of parallel query. When Oracle detects that a query is going to span several partitions, such as a full-table scan, it can fire off parallel processes. Each of processes will independently retrieve data from each partition. This feature is especially important for indexes, since parallel queries do not need to share a single index when servicing a parallel query

(d) ***Before Partitioning:*** When data is stored in a non-partitioned table, without indexes, it always performs a full table scan. Consider following code:

```
CREATE TABLE bom AS SELECT ROWNUM AS n_uniq,
        MOD(ROWNUM,10) AS n10,
        MOD(ROWNUM,100000) AS n100k,
        RPAD(rownum,1000,'A') AS filler
        FROM all_source, all_source
        WHERE rownum<= 100000;

SELECT COUNT(1) FROM bom;
COUNT(1)
--------
  100000
```

Task completed in 3,449 seconds

```
SELECT COUNT(1) FROM bom WHERE n10 = 1;
COUNT(1)
--------
   10000
```

Task completed in 1,328 seconds

(e) ***After Partitioning:*** With table partitioning, a table can be physically divided into multiple smaller tables, called partitions, while logically it stays one table. This means that the code stays the same, but full partition scans will be executed instead of a full table scan.

These partitions are created based on a key. Depending on which value a certain column has, the record will be stored in a certain partition. It's important to choose a column that is often used in queries as our key

```
CREATE TABLE bom_list
 PARTITION BY LIST(n10)
   (partition part1 VALUES (1),
    partition part2 VALUES (2,3,4),
    partition part3 VALUES (DEFAULT))
 AS SELECT * FROM bom;

SELECT table_name, tablespace_name, blocks, num_rows
  FROM user_tablesWHERE table_name LIKE 'BOM%';

TABLE_NAME TABLESPACE_NAME BLOCKS  NUM_ROWS
```

```
---------- --------------- ------  --------
BOM           USERS         14449   100000
BOM_LIST                    14536   100000

SELECT   table_name,   partition_name,   high_value,   tablespace_name,blocks,
num_rows FROM user_tab_partitions WHERE table_name LIKE 'BIG_T%';

TABLE_NAME PARTITION_NAME  HIGH_VALUE TABLESPACE_NAME BLOCKS NUM_ROWS
---------- --------------- ---------- --------------- ------ --------
BOM_LIST   PART1           1          USERS             1461    10000
BOM_LIST   PART2           2, 3, 4    USERS             4364    30000
BOM_LIST   PART3           DEFAULT    USERS             8711    60000
```

The big advantage of partitioning is the possibility for "partition pruning". When we look for a value of "1" in column "N10", we know we will only find this in partition "PART1", so we don't need to access the other partitions, and our query will execute ten times as fast. When we need all values, it will scan all partition, and there will only be a small overhead.

```
SELECT COUNT(1) FROM bom_list;
COUNT(1)
--------
  100000
```

Task completed in 3,601 seconds

```
SELECT COUNT(1) FROM bom_list WHERE n10 = 1;

COUNT(1)
--------
   10000
```

Task completed in 0,537 seconds

(f) ***Index partitioning:*** The partitioning of the indexes is transparent to all SQL queries. The great benefit is that the Oracle query engine will scan only the index partition that is required to service the query, thus speeding up the query significantly. In addition, the Oracle parallel query engine will sense that the index is partitioned and will fire simultaneous queries to scan the indexes.

(g) ***Local partitioned indexes:*** Local partitioned index allows taking individual partitions of a table and indexes offline for maintenance (or reorganization) without affecting the other partitions and indexes in the table.

In a local partitioned index, the key values and number of index partitions will match the number of partitions in the base table.

```
CREATE INDEX year_idx
on all_fact (order_date)
LOCAL
(PARTITION name_idx1,
PARTITION name_idx2,
PARTITION name_idx3);
```

Oracle will automatically use equal partitioning of the index based upon the number of partitions in the indexed table.

(h) ***Global partitioned indexes:*** A global partitioned index is used for all other indexes except for the one that is used as the table partition key. Global indexes partition OLTP (online transaction processing) applications where fewer index probes are required than with local partitioned indexes. In the global index partition scheme, the index is harder to maintain since the index may span partitions in the base table.

```
CREATE INDEX item_idx
on all_fact (item_nbr)
GLOBAL
(PARTITION city_idx1 VALUES LESS THAN (100),
PARTITION city_idx1 VALUES LESS THAN (200),
PARTITION city_idx1 VALUES LESS THAN (300),
PARTITION city_idx1 VALUES LESS THAN (400),
PARTITION city_idx1 VALUES LESS THAN (500));
```

Here, we see that the item index has been defined with five partitions, each containing a subset of the index range values. Note, that it is irrelevant that the base table is in three partitions. In fact, it is acceptable to create a global partitioned index on a table that does not have any partitioning.

3. Scalar Sub-query: A scalar subquery expression is a subquery that returns exactly one column value from one row. The value of the scalar subquery expression is the value of the select list item of the subquery. If the subquery returns 0 rows, then the value of the scalar subquery expression is NULL. If the subquery returns more than one row, then Oracle returns an error.

So now most of them will raise a question whether I can use a Scalar sub-query or a Join?

Answer to this always use *Join* based on the performance background Scalar sub-query takes more burdensome to database.

Scalary sub-query

```
select (select t2.a from t2 where t2.c = t1.c),
       col_2,col_3,
       (select t2.b from t2 where t2.c = t1.c),
       (select t3.x from t3 where t3.d = t1.d),
from .....
```

User JOIN (outer join if need be) to T2 and T3, it would be:

```
select t2.a, col_2, col_3, t2.b, t3.x
from ...., T2, T3 where ....
and t2.c(+) = t1.c  and t3.d(+) = t1.d  -- outer join IF and ONLY IF
necessary
```

In this scalar sub-query caching which can change that slightly, but in general, you'll be running a query inside of another query. You want to use SET processing and currently the optimizer does not roll/merge the scalar subquery into the outer query.

Below are the statistics of a real scenario which is having scalar query and converted to join query. The scalar subquery retrieves 198 records, took 12mins 30secs but after changing to join query it took just only 4secs. The scalar query took more time because fetch more data unnecessarily. So if you are pulling data from more number of data should not use scalar query.

Statistics Name	Scalar Query Value	Normal Join Query Value
Client Statistics		
Elapsed Time	00:12:30.73	00:00:04.24
First Row Time	00:12:30.73	00:00:04.24
Number of Executions	1	1
Number of Records	198	198
Status		
Session Statistics		
Physical Reads	124	5497
Session Logical Reads	44105195	293120
CPU Used by this Session	749.64	4.26
Parse Time CPU	0.03	0.03
Parse Time Elapsed	0.04	0.03
Sorts (Memory)	30544	30592
Sorts (Disk)	0	0
Sorts (Rows)	31941	31950
Table Fetch by Rowid	356592	178360
Table Scan Blocks Gotten	43625277	104528
Table Fetch Continued Row	0	1
Table Scan Rows Gotten		21812837
Table Scans (Short Tables)	60709	30566
Table Scans (Long Tables)	0	0
Consistent Changes	0	0
Consistent Gets	44105194	293119

FIG. 3:- Comparison Between Scalar Query And Normal Join Query.

4. Hints usage in query: Hints give specific information that we know about our data and application. A way to override the default query optimization in the DBMS. Influence the execution plan of query

Why using hints?
Oracle comes with an optimizer that promises to optimize a query's execution plan. But it does not always choose the best execution plan. Using hints may improve the performance by changing the execution plan oracle takes.
It really helps to improve the response time for few queries and Oracle recommends using the hints in some places even though if the query is optimized.

For Example:
(i) FIRST_ROWS(n)
It is recommended to use the FIRST_ROWS(n) in cases where it requires the first n number of rows in the shortest possible time. For example, consider the following PL/SQL block that uses a cursor to retrieve the first 10 hits of a query and uses the FIRST_ROWS(n) hint to optimize the response time:

```
declare
cursor c is
select /* FIRST_ROWS(10) */ article_id from articles_tab
where contains(article, 'Oracle')>0 order by pub_datedesc;

begin
for i in c
loop
insert into t_s values(i.pk, i.col);
exit when c%rowcount> 11;
end loop;
end;
```

(ii) Parallel Queries :

```
SELECT /*+ PARALLEL(s,16) */ SUM(amount_sold)
FROM sales s  WHERE s.time_id
BETWEEN TO_DATE('01-JAN-2005','DD-MON-YYYY')
AND TO_DATE('31-DEC-2005','DD-MON-YYYY');
```

5. Session Tuning: Before start troubleshooting for slow performance database, one has to first understand that the database itself is never slow or fast—it has a consistent speed. The sessions connected to the database, however, slow down when they hit a bump in the road. To resolve a session performance issue, you need to identify the bump and remove it. Fortunately, it's very easy to do in most of the cases.

The first step to resolving a session performance issue is to ascertain what the database session is doing at any instant of time. An Oracle database session is always in one of three states:

Idle Not doing anything—just waiting to be given some work.
Processing Doing something useful—running on the CPU.
Waiting Waiting for something, such as a block to come from disk or a lock to be released.

You can query the database's current state using a view called V$SESSION.

How difficult is it to get information about what's causing the session to stop? It's actually ***very*** easy: Oracle database is instrumented to talk about what the database sessions are doing. All you need to do is to listen attentively or, more precisely, look for that information in the right place, and that place is a view called V$SESSION. Everything you need for your analysis is in this view.

How to identify and solve the session issue?
Update in one table
Example:

```
update t1
set col2 = 'x' where col1 = 1;
```

The output will show "1 row updated," indicating that the row was updated. Do not issue a COMMIT after the statement. By not committing, you will force the session to get and hold a lock on the first row of the T1 table. Now go to the second session and issue the following SQL statement:

```
update t1 set col2 = 'y' where col1 = 1;
```

This statement will hang. Why? The answer is simple: the first session holds a lock on the row, which causes the second session to hang and the user to complain that the session is slow. To know what the second session is doing, the first thing one has to understand the need to check is the STATE column in V$SESSION:

```
Select sid, state from v$session
where username = ' DEERAJ';
SID   STATE
____  ___________________
3346  WAITING
2832  WAITED KNOWN TIME
```

The important point is that session 2832 is ***not waiting*** for anything right now, meaning that it's working productively. To check what is blocking can view the EVENT column in the V$SESSION. Below query displays state of the query.

```
Select sid, state, event from v$session
where username = ' DEERAJ';
```

```
SID   STATE               EVENT
_____ ________________  ____________________________
2832  WAITED KNOWN TIME  SQL*Net message from client
3346  WAITING            enq: TX - row lock contention
```

The session is waiting because it wants to lock one or more rows, but another session has already placed locks on the row or rows. Unless that other session commits or rolls back the transaction, session 3346 will not get the lock it needs and will have no choice but to wait.

Getting the amount of time a session has been waiting makes sense for sessions that are waiting ***right now***, but what about the sessions that are working now? Recall that the EVENT column shows not only the event a session is experiencing now but also the last wait event the session has experienced. Another column—WAIT_TIME—in the same V$SESSION view shows how long that wait lasted.

Below query to display the session, session state and wait details

```
col "Description" format a50
selectsid,
decode(state,'WAITING','Waiting','Working')state,
decode(state,'WAITING','So far '||seconds_in_wait,
       'Last waited '||wait_time/100)||' secs for
       '||event)  "Description"
from v$session where username = 'DEERAJ';
```

Output:

```
SID  STATE     Description
____ _______  ____________________________________
2832  Working  Last waited 2029 secs for SQL*Net message from client
3346  Waiting  So far 743 secs for enq: TX - row lock contention
4208  Waiting  So far 5498 secs for SQL*Net message from Client
```

a. ***Diagnosis of Locking:*** The output of listing 2 provides enough information to enable you to make a diagnosis about the performance of these three sessions. Session 4208 is idle, so any complaints that session 4208 is slow just aren't related to the database. Any performance issues related to this session could be related to a bug in the code that's going through an infinite loop or high CPU consumption on the application server. The performance troubleshooting focus has to be redirected toward the application client.

The story of session 3346 is different. This session is truly a bottleneck to the application. Now that it is understandable, to know why this session appears slow—it is waiting for a row lock—the next logical question is which session holds that lock.

b. ***Session Parameter Tips:*** If processes and session value are set as 1000 and 1248 in the database respectively. If, using 11.2.0.1, when DB is restarted, it shows sessions as 1524. It confuses to see higher value than the value set for sessions:

```
SQL> alter system set processes=1000 scope=spfile;
SQL> alter system set sessions=1248 scope=spfile;
```

It is clear that, that the session parameter is derived as a function of the processes parameter *It looks like:*

11R1 - sessions = (1.5 * PROCESSES) + 22
11R2 - sessions = (1.1 * PROCESSES) + 5

If sessions parameter was set as a low value, than the derived value, then Oracle automatically bumps it to above derived value.

If session parameter was set higher value, than the derived value, Oracle will consider our set value.

Conclusion:-

Realizing the value of proactive tuning approach on database performance tuning, will bring out the true potential of identifying and addressing the performance issue during application development cycle, well before any impact to end users. There are many varieties of tools available in the market which can be used for performance tuning. These tools are used to diagnose and proactively tune an Oracle database.

There are some key tools which used for proactive performance. Such as – SQL optimizer – can analyze for query tuning, index tuning. Enterprise manager – can be used to monitor an Oracle database for performance related. Spotlight – This is can be used to monitor for memory and also overall performance of Oracle database.

At HCLT, Performance Engineering Service Line group is providing exclusively various performance engineering services not only to tune but also to identify and fix the various potential performance issues and application stability related bottlenecks.

References:-

1. Tuning guide from Oracle Corporation -http://docs.oracle.com/cd/B19306_01/server.102/b14211/toc.htm
2. Tuning guide from Oracle DBA - http://www.dba-oracle.com
3. Oracle DBA forum - https://forums.oracle.com/thread/918658
4. Expert Indexing in Oracle Database 11g - Publisher Apress – Written by Sam Alapati, Darl Kuhn, Bill Padfield
5. Expert Oracle Database Architecture, 2nd Edition - Publisher Apress – Written by Sam Alapati, Darl Kuhn, Bill Padfield
6. Oracle Performance Tuning, 2nd Edition – Publisher - O'Reilly Media – by Mark Gurry, Peter Corrigan.