GnanaSekar. A

ORACLE CERTIFIED ASSOCIATE

TABLE OF CONTENTS

1. Tuning Overview

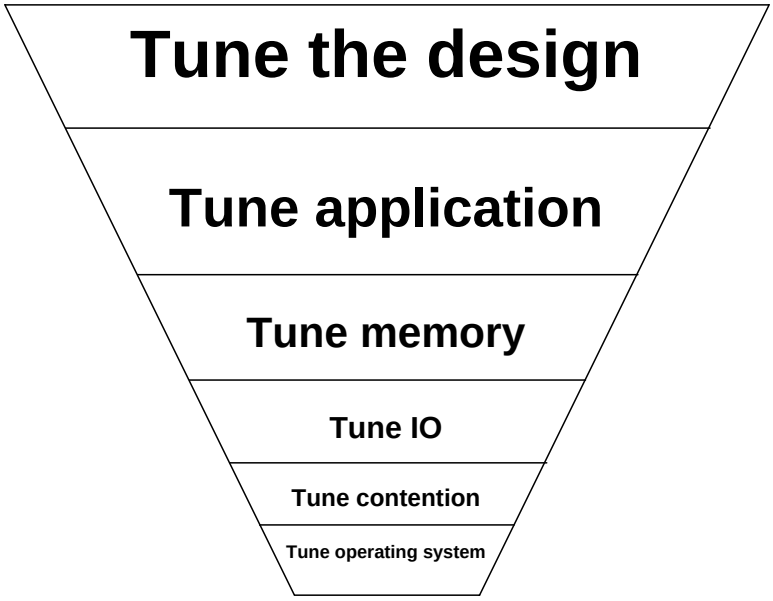
Tuning Questions

Questions	Answers
Who tunes?	• DBAs • Application designers • Application developers • System Administrators
Why tune?	The best practice of tuning is careful design of systems and applications and the majority of performance gains are realized by tuning the application. If wrong decisions were made early, or if users now expect much more from the system than they did previously, you may need to seriously consider improving performance. The longer you delay addressing the tuning process, the more it costs in time and resources.
How much tuning?	You should begin tuning with a clear idea of what you are trying to achieve. Try to quantify this as precisely as possible, in real world terms. For example: • Process 10,000 orders a day • Produce 250,000 billing statements overnight at the end of the month Tuning is an iterative process. It is not an activity you do once and then forget.

Tuning Goals

- Access the least number of blocks
- Cache blocks in memory
- Share application code
- Read and write data as fast as possible
- Ensure users do no wait for resources
- Perform backups and housekeeping while minimizing impact

Tuning Steps



Repeat the process if your goals have not yet been achieved. The rationale for this structure is that improvements early in the sequence may save you from having to deal with later issues. The first two steps are typically the responsibility of the system architects and application developers, however, the DBA may also be involved in application tuning.

2. Diagnostic Information

Alert Log File

- The Alert log file consists of a chronological log of messages and errors.
- Check the Alert log file regularly to:
  - Detect internal errors (ORA-600) and block corruption errors.
  - Monitor database operations, such as CREATE DATABASE, STARTUP, SHUTDOWN, ARCHIVE LOG, RECOVER
  - View the non default initialization parameters.
- Remove or trim it regularly after checking.
- The parameter BACKGROUND_DUMP_DEST controls the location of alert file

Background Processes Trace Files

- If an error is detected by a background process, the information is dumped into a trace file.
- BACKGROUND_DUMP_DEST controls the location of the background On UNIX, the default value is $ORACLE_HOME/rdbms/log and the name of the file is `<SID>_<processname>_<PID>.trc`.
- On NT, the default value is %ORACLE_HOME%\Rdbms80\Trace and the name of the file is `<sid><PROCESSNAME>.TRC`.processes trace files:

Dynamic Performance Views

V$EVENT_NAME

A collection of Wait Events provides information on the sessions that had to wait or must wait for different reasons.

```
SQL> SELECT name, parameter1, parameter2, parameter3
  2> FROM v$event_name;
```

NAME	PARAMETER1	PARAMETER2	PARAMETER3
buffer busy waits	file#	block#	id
library cache pin	handle addr	pin address	0*mode+name
log buffer space			
log file switch			
transaction	undo seg#	wrap#	count
...			
136 rows selected.			

V$SYSTEM_EVENT : total waits for an event, all sessions together.

```
SQL> SELECT event, total_waits, total_timeouts,
  2> time_waited, average_wait
  3> FROM v$system_event;
```

EVENT	TOTAL_ WAITS	TOTAL_ TIMEOUTS	TIME_ WAITED	AVERAGE_ WAIT
latch free	5	5	5	1
pmon timer	932	535	254430	272.993562
process startup	3		8	2.66666667
buffer busy waits	12	0	5	5
...				
23 rows selected.				

V$SESSION_EVENT : waits for an event for each session that had to wait.

```
SQL> select * from v$session_event where sid=10;
```

SID	EVENT	TOTAL_WAITS	AVERAGE_WAIT
10	buffer busy waits	12	5
10	db file sequential read	129	0
10	file open	1	0
10	SQL*Net message to client	77	0
10	SQL*Net more data to client	2	0
10	SQL*Net message from client	76	0

V$SESSION_WAIT: waits for an event for current active sessions that are waiting.

```
SQL> SELECT sid, seq#, event, wait_time, state
  2> FROM v$session_wait;
```

SID	SEQ#	EVENT	WAIT TIME	STATE
1	1284	pmon timer	0	WAITING
2	1697	rdbms ipc message	0	WAITING
3	183	rdbms ipc message	0	WAITING
4	4688	rdbms ipc message	0	WAITING
5	114	smon timer	0	WAITING
6	14	SQL*Net message from client	-1	WAITED SHORT TIME

3. Utility for Performance Tuning

UTLBSTAT and UTLESTAT Scripts

- Gather performance figures over a defined period.
- Produce a hard-copy report.
- Run the scripts from Server Manager connected as SYSDBA.
- Set TIMED_STATISTICS to TRUE.

The Dynamic views display cumulative totals since the instance started, but this is often unhelpful; if your instance is rarely shut down, the statistics may cover a long period and have little meaning.
You should gather performance figures over a defined period, probably your busiest time of day or month end, and produce a hard-copy report. You can do this with the utlbstat.sql and utlestat.sql scripts, stored in $ORACLE_HOME/rdbms/admin directory on UNIX, in C:\Orant\Rdbms80\Admin on NT.

The Statistics Report(report.txt) Contents

- Library cache statistics
- System statistics
- Wait events statistics
- Latch statistics
- Rollback contention statistics
- Buffer Busy Wait Statistics
- Dictionary cache statistics
- I/O statistics per datafile/tablespace
- Period of measurement

4. Tuning Considerations for Different Applications

OLTP and DSS

	Online Transaction Processing(OLTP)	**Decision Support Systems (DSS)**
Characteristics	high-throughput, insert and Update intensive systems. Contain large volumes of data that grow continuously and are accessed concurrently by hundreds of users	Perform queries on large amounts of data and make heavy use of full table scans
Examples	• Airline reservation systems • Large order-entry applications • Banking applications	• Analyst large volume of data • Month end reporting • Research of marketing data
Tuning Goals	• Availability • Speed • Concurrency • Recoverability	• Response time • Accuracy • Availability
Space Allocation	Explicit space allocation	• Set DB_BLOCK_SIZE to the maximum. • Set DB_FILE_MULTIBLOCK_READ_COUNT carefully. • Ensure that extent sizes are multiples of this number. • Run ANALYZE regularly.
Indexes	• Not too many (prefer B*Tree to bitmap) • Reverse key for sequence columns and Parallel Server application • Rebuilt regularly	• Evaluate the need for indexes. • Use bitmap indexes when possible. • Use index-organized tables for large data retrieval by PK. • Generate histograms for data indexes that are distributed nonuniformly.
Clusters for tables in join queries	• Index clusters for growing tables • Hash clusters for stable tables	Hash clusters for performance access.
Rollback Segment	• Are unlikely to run out of rollback space • Need enough rollback segments to prevent contention • Extent size can be relatively small. • MINEXTENTS to 20 avoids extent allocation	Needs fewer, larger rollback segments
Others	• Use constraints instead of application code. • Make sure that code is shared. • Use bind variables rather than literals.	• Parse time is less important. • Execution plan must be optimal. • Bind variables are problematic. • Partitioning of large tables

Hybrid Systems

Many Oracle customers who are maintaining large databases often want to use the same instance for transaction processing and for decision support. It is impossible to tune your data storage optimally for both needs, and the applications will contend for resources.

- Memory use: Higher value during day time and lower value during night time.
  - SHARED_POOL_SIZE
  - DB_BLOCK_BUFFERS
  - SORT_AREA_SIZE
- Parallel query:
  - Reconfigure parameters for DSS.
- Rollback segments:
  - More small rollback segments during the day
  - Fewer, large rollback segments at night
- Multi-threaded server:
  - For peak-time use, not for DSS

5. SQL Tuning

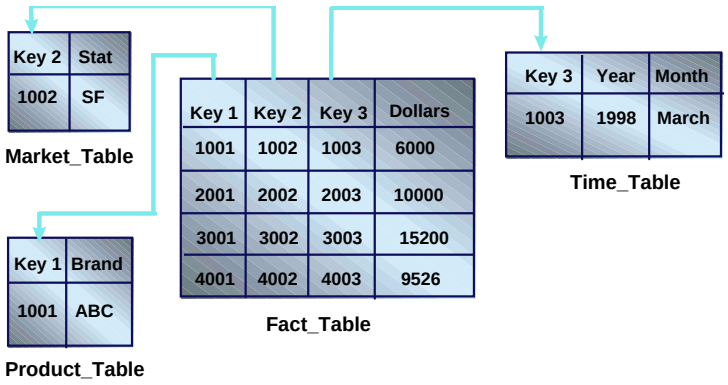
Optimizer Modes

Rule-Based Optimization	In this mode, the server process chooses its access path to the data by examining the query. This optimizer has a complete set of rules for ranking access paths. Experienced Oracle developers often have a very good understanding of these rules, and tune their SQL accordingly. The rule-based optimizer is syntax driven, in that it uses the statement syntax to determine what execution plan will be utilized.
Cost-Based Optimization	In this mode, the optimizer examines each statement and identifies all possible access paths to the data. It then calculates the resource cost of each access path and chooses the least expensive. The costing is based mainly on the number of logical reads. The cost-based optimizer is statistics-driven in that it uses statistics generated for the objects involved in the SQL statement to determine the most effective execution plan. The cost-based optimizer will be used if any object in the SQL statement has had statistics generated for it.

Setting Optimizer Mode

Instance Level	Use the parameter OPTIMIZER_MODE = **CHOOSE** : The optimizer uses the cost-based mode if statistics are available for the tables. Otherwise, it uses rule-based optimization. **RULE :** Uses rule-based optimization **FIRST_ROWS** : Minimizes immediate response time (possibly at the expense of overall response time) **ALL_ROWS** : Minimizes total response time.
Session Level	Developers can set this option using the ALTER SESSION command. `ALTER SESSION SET OPTIMIZER_GOAL = value;` The possible values are the same as for the parameter OPTIMIZER_MODE.
Statement Level	You can code hints into a statement, as shown below. `SELECT /*+ FIRST_ROWS */ * FROM scott.emp;` Possible optimizer hints are RULE, FIRST_ROWS, and ALL_ROWS.

Star Query



Characteristics:

- A join between a fact table and a number of lookup tables.
- Lookup tables are joined to the fact table using a PK-FK join.
- The fact table contains key values and measures.
- The fact table key is normally a concatenated index.

The cost-based optimizer recognizes star queries and builds a star query execution path if the fact and lookup tables are appropriately structured. The **STAR_TRANSFORMATION_ENABLED** parameter specifies whether a cost-based query transformation is applied to star queries. The default value is TRUE.

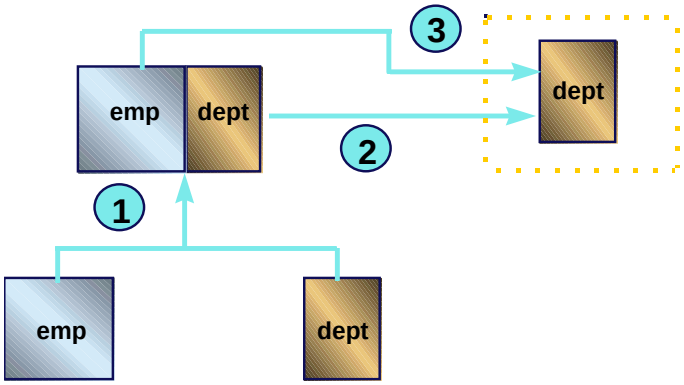
Example

```
SELECT sum(dollars)
FROM fact_table, time_table, product_table, market_table
WHERE market_table.stat = 'New York' AND
product_table.brand = 'Mybrand' AND
time_table.year = '1998' AND
time_table.month = 'March' AND
product_table.key1 = fact_table.key1 AND
market_table.key2 = fact_table.key2 AND
time_table.key3 = fact_table.key3;
```

Hash Joins

Hash joins provide an efficient mechanism for joining two tables where one table may be significantly larger than the other, for example, where an employee table containing possibly thousands of rows is joined with a department table that contains a small number of rows.

Example



1. A hash join is performed between the EMP and DEPT tables. This results in a full table scan of each table.
2. Since the DEPT table is the smaller of each, a hash table is built based on it and placed into memory.
3. The larger of the two tables, the EMP table, is then used to probe the hash table.

Diagnostic Tools

Numerous diagnostic tools are available for evaluating the performance of SQL statements. Each provides a developer or DBA with a varying degree of information.

Tool 1 : EXPLAIN PLAN

Steps	Detail
Create the plan table.	Create the table called **plan_table** by runing the following script `SQL> @$ORACLE_HOME/rdbms/admin/utlxplan`

Run the SQL statement.	`SQL> EXPLAIN plan for`
	`  2> UPDATE emp`
	`  3> SET salary = salary * 1.10`
	`  4> WHERE dept_id = ( select id from dept`
	`  5> WHERE name = 'Finance')`
	`  6> /`

Query the plan table.	`SQL> select id, operation, options, object_name`
	`  2> from plan_table;`
	
	`ID OPERATION  OPTIONS  OBJECT_NAME`
	`-- ---------- ---------- ----------`
	`0  UPDATE STATEMENT`
	`1  FILTER`
	`2  TABLE ACCESS  FULL  EMP`
	`3  TABLE ACCESS  BY ROWID  DEPT`
	`4  INDEX  RANGE SCAN  DEPT_NAME_UK`

Tool 2 : SQL TRACE and TKPROF

Steps	Detail
1. Set initialization parameters	MAX_DUMP_FILE_SIZE = 5M USER_DUMP_DEST = <directory> TIMED_STATISTICS = true
2. Switch on SQL trace	`SQL> ALTER SESSION SET sql_trace= TRUE;`
3. Run the application	Run the SQL statement as normal.
4. Switch off SQL trace	`SQL> ALTER SESSION SET sql_trace= FALSE;`
5. Format the trace file	`$ tkprof ora_1945.trc pf1.txt explain=scott/tiger`

6. Interpret the output	`call  count cpu  elapsed disk  query current rows`
	`------- ----- ---- ------- ----- ----- ------- -----`
	`Parse  2  0.11 0.22  00  2  0  0`
	`Execute 4  0.00 0.00  00  4  0  0`
	`Fetch  428  0.00 0.06  600  806  808  6400`
	`------- ----- ---- ------- ----- ----- ------- -----`
	`total  434  0.11 0.28  600  806  814  6400`

Tool 3 : Autotrace

Steps	Explanation
1	Creating the PLAN_TABLE using the utlxplan.sql script.
2	Grant the trace role to the user by executing the **plustrce.sql** script. This grants select privileges to V$ views.
3	**SQL> set autotrace on**

6. Tuning the Shared Pool

Reduce misses in Library Cache

Tuning Goals	• Make sure that users can share statements • Prevent statements from being aged out by allocating enough space • Avoid invalidation that induce reparsing
Terminology	• **GETS**: The number of lookups for objects of the namespace • **PINS**: The number of reads or executions of the objects of the namespace • **RELOADS**: The number of library cache misses on the execution step causing implicit reparsing of the statement and block
Diagnostic method and guidelines	1. The gethitratio on the library cache is greater than 90 % **SQL> select gethitratio from v$librarycache** **2> where namespace = 'SQL AREA';** 2. The pinhitratio on the library cache is less than 1 % **SQL> select sum(pins), sum(reloads),** **2> sum(reloads)/sum(pins)** **3> from v$librarycache;** **sum(pins) sum(reloads) sum(reloads)/sum(pins)** **--------- ------------ ----------------------** **2641 10 .00378644**

Avoid Fragmentation in Library Cache

Tuning Goals	• Reserving space for large memory requirements • Pinning often-required large objects • Eliminating large anonymous PL/SQL blocks • Reducing UGA consumption of MTS connections
Diagnostic method and guidelines	• Define the global space necessary for stored objects (packages, views, and so on) • Define the amount of memory used by the usual SQL statements • Reserve space for large memory requirements to avoid misses and fragmentation **SHARED_POOL_RESERVED_SIZE** **SHARED_POOL_RESERVED_MIN_ALLOC** • Pin often-used objects in library cache by using the dbms_shared_pool package **SQL> EXECUTE dbms_shared_pool.keep('package_name');** • Convert large anonymous PL blocks into small anonymous blocks calling packaged functions

Tune the Data Dictionary Cache

Tuning Goals	**Avoid dictionary cache misses**
Terminology	**GETS**: Number of requests on objects **GETMISSES**: Number of requests resulting in cache misses
Diagnostic method and guidelines	Keeping the ratio of the sum of GETMISSES to the sum of GETS less than 15% **SQL> select parameter, gets, getmisses** **2> from v$rowcache;** **PARAMETER GETS GETMISSES** **dc_objects 143434 171** **dc_synonyms 140432 127**

7. Tuning the Buffer cache

Tuning Goals	• Servers find data in memory • 90% hit ratio for OLTP
Measurement of the hit ratio	**From V$SYSSTAT:** **SQL> SELECT 1 - (phy.value / (cur.value + con.value))** **2> "CACHE HIT RATIO"** **3> FROM v$sysstat cur, v$sysstat con, v$sysstat phy** **4> WHERE cur.name = 'db block gets'** **5> AND con.name = 'consistent gets'** **6> AND phy.name = 'physical reads';** **CACHE HIT RATIO** **--------------** **.908160337** **From report.txt:** **Statistic Total Per Transact Per Logon Per Second** **--------------- ------ ------------ --------- ----------** **consistent gets 121754 1117.07 6764.11 50.73** **db block gets 20628 189.25 1146 8.6** **physical reads 104695 960.5 5816.94 43.62**
Tuning Techniques	• Increase buffer cache size • Use multiple buffer pools • Cache tables

Tuning Techniques 1 : Increase buffer cache size

Determining the Impact of Adding Buffers

If your hit ratio is low, you may want to test the impact of adding more buffers. Oracle can collect statistics that estimate the performance gain that would result from increasing the size of your buffer cache. With these statistics, you can estimate how many buffers to add to your cache. To collect the statistics:

1. Set the initialization parameter DB_BLOCK_LRU_EXTENDED_STATISTICS to the number of buffers that you might add.

DB_BLOCK_LRU_EXTENDED_STATISTICS = 200

2. Startup the database for normal use.
3. After a period of normal running, query the virtual table V$RECENT_BUCKET.

Examples

To find the number of additional cache hits that you would incur by increasing the cache size; for example from 100 to 120 buffers you could use the following query:

```
SQL> SELECT SUM(count) ach
  2> FROM v$recent_bucket
  3> WHERE rownum < 20;
```

You can then find the impact on the cache hit ratio by including the additional cache hits in the formula used previously. Because the additional buffers are reducing physical I/Os, you subtract the ach value from phy.value.

```
SQL> SELECT 1-((phy.value-ach)/(cur.value+con.value)) "CACHE HIT RATIO"
  2> FROM v$sysstat cur, v$sysstat con, v$sysstat phy
  3> WHERE cur.name = 'db block gets' AND
  4> con.name = 'consistent gets' AND
  5> phy.name = 'physical reads';
```

Determining the Impact of Adding Buffers
If your hit ratio is high, your cache is probably large enough to hold your most frequently accessed data. In this case, you may be able to reduce the cache size and still maintain good performance. Oracle can collect statistics to predict buffer cache performance based on a smaller cache size. Examining these statistics can help you determine how small you can afford to make your buffer cache without adversely affecting performance. To collect the statistics:

1. Set the initialization parameter **`DB_BLOCK_LRU_STATISTICS = TRUE`**.
2. Startup the database up for normal use.
3. After a period of normal running, query the virtual table V$CURRENT_BUCKET.

Tuning Techniques 2 : Using Multiple Buffer Pools

The DBA may be able to improve the performance of the database buffer cache by creating multiple buffer pools. Objects are assigned to a buffer pool depending on how the objects are accessed. Oracle8 has three buffer pools:

- **KEEP**: Used to retain objects in memory that are likely to be reused. Keeping these objects in memory reduces I/O operations.
- **RECYCLE**: Used to eliminate blocks from memory that have little chance of being reused. Flushing these blocks from memory sooner, the space that would be used by their cache buffers can be allocated to other objects.
- **DEFAULT**: The pool always exists. It is equivalent to the single buffer cache.

Defining Multiple Buffer Pools

In init.ora parameter file

```
...
DB_BLOCK_BUFFERS = 20000
DB_BLOCK_LRU_LATCHES = 6
BUFFER_POOL_KEEP=(«BUFFERS:14000»,«LRU_LATCHES:1»)
BUFFER_POOL_RECYCLE=(BUFFERS:2000,«LRU_LATCHES:3»)
...
```

- Pool blocks taken from DB_BLOCK_BUFFERS
- Latches taken from DB_BLOCK_LRU_LATCHES
- At least 50 blocks per latch
- DBA can define 1, 2, or 3 pools

Enabling Multiple Buffer Pools

```
CREATE INDEX cust_idx …
STORAGE (BUFFER_POOL KEEP …);

ALTER TABLE customer
STORAGE (BUFFER_POOL RECYCLE);

ALTER INDEX cust_name_idx
REBUILD
STORAGE (BUFFER_POOL KEEP);
```

Tuning Techniques 3 : Caching Tables

- Enable caching during full table scans by:
  - Creating the table with the CACHE clause
  - Altering the table with the CACHE clause
  - Using the CACHE hint in a query

- Guidelines:
  - Do not overcrowd the cache
  - CACHE_SIZE_THRESHOLD limits cached blocks

8. Tuning the Redo Log Buffer

Tuning Goal	Ensuring there is adequate space so that log space requests from server processes and transactions are satisfied.
Terminology	**redo log space requests** : Total # of waits when requesting the space in redo log buffer **redo entries** : Total # of requests of redo log buffer space
Measurement	The ratio should be less than 1/ 5000. **From V$SYSSTAT** **SQL> select (req.value*5000)/entries.value "Ratio"** **2> from v$sysstat req, v$sysstat entries** **3> where req.name = 'redo log space requests'** **4> and entries.name = 'redo entries'** **Ratio** **--------------** **11.678932** **From report.txt** **Statistic Total Per Transact Per Logon Per Second** **redo entries 232898 77632.67 46579.6 189.5** **redo log space requests 544 181.33 108.8 .44** **redo log space wait time 39507 13169 7901.4 32.15**

Further Investigations

- There is disk I/O contention on the redo log files. Check that the redo log files are stored on separate, fast devices.

- The V$SESSION_WAIT view indicates through the “log buffer space” event if there are any waits for log switch to occur.

```
SQL> select sid, event, seconds_in_wait, state
  2> from v$session_wait
  3> where event like 'log%';

SID    EVENT                                SECONDS_IN_WAIT    STATE
----   ---------------------------------    ---------------    -------
5      log buffer space                     110                WAITING
9      log file switch (archiving needed)   119                WAITING
```

The SECONDS_IN_WAIT value of the “log buffer space” event indicates the time spent waiting for space in the redo log buffer because the log switch does not occur. This is an indication that the buffers are being filled up faster than LGWR is writing. This may also indicate disk I/O contention on the redo log files.

- Check in the V$SYSTEM_EVENT view the number of occurrences of the event “log file switch completion” that identifies the log file switch waits because of log switches.

```
SQL> select event, total_waits, time_waited, average_wait
  2> from v$system_event
  3> where event like 'log file switch completion%';
```

Increase the size of the redo log files and or add groups.

- DBWR has not completed checkpointing the file when the LGWR needs the file again. LGWR has to wait.
  - Check in the alert.log file the existence of the message “CHECKPOINT NOT COMPLETE”.
  - Check in the V$SYSTEM_EVENT view the number of occurrences of the event “log file switch (checkpoint incomplete)” that identifies the log file switch waits because of incomplete checkpoints.

```
SQL> select event, total_waits, time_waited, average_wait
  2> from v$system_event
  3> where event like 'log file switch (check%';
```

  - Check the frequency of checkpoints and set the appropriate values for the LOG_CHECKPOINT_INTERVAL and LOG_CHECKPOINT_TIMEOUT parameters.
  - Check the size and number of redo log groups.

- LOG_BLOCK_CHECKSUM is set to TRUE, and therefore adds performance overhead.

9. Database Configuration and I/O Issues

Tune Tablespace Usage	• Reserve the SYSTEM tablespace for data dictionary objects only. • Stripe table and index data using the partitioning option • Separate tables and indexes into separate tablespaces • Create separate rollback tablespaces for rollback segments • Store very large database objects in their own tablespace • Create one or more temporary tablespaces for sorting
Tune File Placement	• Separate data files and redo log files • Striping table data by manual striping, RAID or Oracle8 partitioning. • Reduce disk I/O unrelated to Oracle
Tune Full Table Scan	• Investigate the need of full table scans • Specify the initialization parameter **DB_FILE_MULTIBLOCK_READ_COUNT** – to determine the number of database blocks the server reads at once – to influence the execution plan of the cost based optimizer

<table>
<tr><td>Tune I/O Balancing</td><td>Query V$FILESTAT to find out the number of disk I/O per disk file. Summarize all of the I/Os on data files on a per disk basis to find out the data files most likely to cause a disk bottleneck.

Column Description
FILE# File number (join to FILE# in V$DATAFILE for the name)
PHYRDS Number of physical reads done
PHYWRTS Number of physical writes done
PHYBLKRD Number of physical blocks read
PHYBLKWRT Number of physical blocks written
READTIM Time spent doing reads
WRITETIM Time spent doing writes

SVRMGR> SELECT phyrds,phywrts,d.name
2> FROM v$datafile d, v$filestat f
3> WHERE d.file#=f.file# order by d.name;

PHYRDS PHYWRTS NAME
------- --------- --------------------
806 116 /DISK1/sys01.dbf
168 675 /DISK1/temp01.dbf
26 257 /DISK2/rbs01.dbf
8 8 /DISK3/user01.dbf
65012 564 /DISK4/scott_dat.dbf
8 8 /DISK4/scott_ind.dbf
6 rows selected</td></tr>
<tr><td>Tune Redo Log File Sizing</td><td>• Size redo log files to minimize contention
• Have enough groups to prevent waiting
• Store redo log files on separate fast devices
• Query the dynamic performance views V$LOGFILE and V$LOG</td></tr>
<tr><td>Tune Checkpoints</td><td>From report.txt

Statistic Total Per Trans Per Logon
---------- ------ ------ -----
DBWR checkpoint write reque 531 5.26 31.24
background checkpoints comp 318 3.15 18.71
background checkpoints star 319 3.16 18.76

• Size the online redo log files to cut down the number of checkpoints
• Add online redo log groups to increase the time LGWR starts to overwrite
• Regulate checkpoints with the initialization parameters LOG_CHECKPOINT_INTERVAL LOG_CHECKPOINT_TIMEOUT
• Specify the initialization parameter DB_BLOCK_CHECKPOINT_BATCH to reduce the performance impact of checkpoints</td></tr>
</table>

10. Using Oracle Blocks Efficiently

Avoid dynamic extent allocation

When database operations cause the data to grow and exceed the space allocated, Oracle extends the segment. Dynamic extension, extending the segment when executing an INSERT or UPDATE statement, reduces performance, because the server executes several recursive SQL statements to find free space and add the extent to the data dictionary.

To avoid dynamic extension:

- Size the segment appropriately by:
  - determining the maximum size of your object
  - choosing storage parameters that allocate extents large enough to accommodate all of your data when you create the object

  When determining the segment size, the DBA should allow for the growth of data. For example, allocate enough space for the current data and for any data that will be inserted into the segment over the next year.

- Monitor the database for segments that are about to dynamically extend and extend them with an ALTER TABLE/INDEX/CLUSTER command.

Example : To display segments with < 10% free blocks, then allocate extents to such segments.

```
SQL> SELECT owner, table_name, blocks, empty_blocks
  2> FROM dba_tables
  3> WHERE empty_blocks / (blocks+empty_blocks) < 0.1;

OWNER   TABLE_NAME    BLOCKS   EMPTY_BLOCKS
------  ------------  -------  -------------
HR      EMP           1450     50
HR      REGION        460      40

SQL> ALTER TABLE hr.emp ALLOCATE EXTENT;
Table altered.
```

Large extents or small extents

Advantages of Large Extents

- Large extents avoid dynamic extents, because segments with larger extents are less likely to need to be extended.
- Larger extents can have a small performance benefit because Oracle can read one large extent from disk with fewer multi block reads than would be required to read many small extents. To avoid partial multi block reads, set the extent size to a multiple of 5*DB_FILE_MULTIBLOCK_READ_COUNT. Multiply by five, because Oracle tries to allocate blocks on five blocks boundaries. By matching extent sizes to the I/O and space allocation sizes, the performance cost of having many extents in a segment will be minimized. However, for a table that never has a full table scan operation per, it makes no difference in terms of query performance whether the table has one extent or multiple extents.
- For very large tables, operating system limitations on file size may force the DBA to allocate the object with multiple extents.

- The performance of searches using an index is not affected by the index having one extent or multiple extents.
- Extent maps list all the extents for a certain segment. For MAXEXTENTS UNLIMITED, these maps are in multiple blocks. For best performance, you should be able to read the extent map with a single I/O. Performance degrades if multiple I/Os are necessary for a full table scan to get the extent map. Also, a large number of extents can degrade data dictionary performance, because each extent uses space in the dictionary cache.

Disadvantages of Large Extents

- Because large extents require more contiguous blocks, Oracle may have difficulty finding enough contiguous space to store them.
- Because the DBA sizes the segment to allow for growth, some of the space allocated to the segment will not be initially used.

Large blocks or small blocks

Small Oracle Blocks

Advantages

- Small blocks reduce block contention, since there are fewer rows per block.
- Small blocks are good for small rows.
- Small blocks are good for random access, because it is unlikely that a block will be reused after it is read into memory, a smaller block size makes more efficient use of the buffer cache. This is especially important when memory resources are scarce, because the size of the database buffer cache is limited.

Disadvantages

- Small blocks have relatively large overhead.
- You may end up storing only a small number of rows per block, depending on the size of the row. This may cause additional I/Os.

Large Oracle Blocks

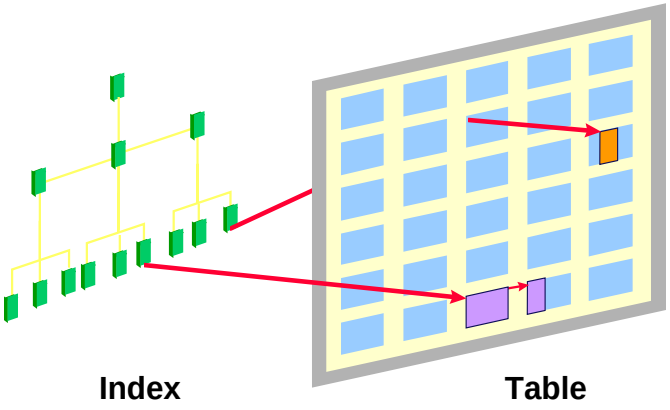
Advantages

- There is relatively less overhead and thus more room to store useful data.
- Large blocks are good for sequential reads.
- Large blocks are good for very large rows.
- Larger blocks improve performance of index reads. The larger blocks can hold more index entries in each block which reduces the number of levels in large indexes. Fewer index levels mean fewer I/Os when traversing the index branches.

Disadvantages

- A large block size is not good for index blocks used in an OLTP type of environment, because they increase block contention on the index leaf blocks.
- Space in the buffer cache will be wasted if you are doing random accesses to small rows and have a large block size. For example, with an 8 kilobyte block size and a 50 byte row size, you would be wasting 7,950 bytes in the buffer cache when doing random access.

Migration and Chaining



- In the first case, called *chainin* g, the row is too large to fit into an empty data block. In this case, Oracle stores the data for the row in a chain of one or more data blocks. Chaining can occur when the row is inserted or updated. Row chaining usually occurs with large rows, such as rows that contain a LOB. Row chaining in these cases is unavoidable.
- However, in the second case, called *migratio* n, an UPDATE statement increases the amount of data in a row so that the row no longer fits in its data block. Oracle tries to find another block with enough free space to hold the entire row. If such a block is available, Oracle moves the entire row to the new block. Oracle keeps the original row piece of a migrated row to point to the new block containing the actual row; the ROWID of a migrated row does not change. Indexes are not updated, so they point to the original row location.
- Migration and chaining have a negative affect on performance:
  - INSERT and UPDATE statements that cause migration and chaining perform poorly, because they perform additional processing.
  - Queries that use an index to select migrated or chained rows must perform additional I/Os.
- Migration is caused by PCTFREE being set too low; there is not enough room in the block for updates. To avoid migration, all tables that are updated should have their PCTFREE set so that there is enough space within the block for updates.

Detecting Chaining and Migration

- By ANALYZE command

```
SQL> ANALYZE TABLE sales.order_hist LIST CHAINED ROWS;
Table analyzed.

SQL> SELECT owner_name, table_name, head_rowid
  2> FROM chained_rows
  3> WHERE table_name = 'ORDER_HIST';
```

OWNER_NAME	TABLE_NAME	HEAD_ROWID
SALES	ORDER_HIST	AAAAluAAHAAAAA1AAA
SALES	ORDER_HIST	AAAAluAAHAAAAA1AAB
...		

- From report.txt

```
Statistic                      Total    Per transaction ...
-------------------------      -----    --------------- ...
table fetch continued row      495      .02 …
...
```

Eliminating Migrated Rows

1. Analyze table . . . list chained rows;
2. Copy the rows to another table
3. Delete the rows from the original table
4. Insert the rows from step 2 back into the original table

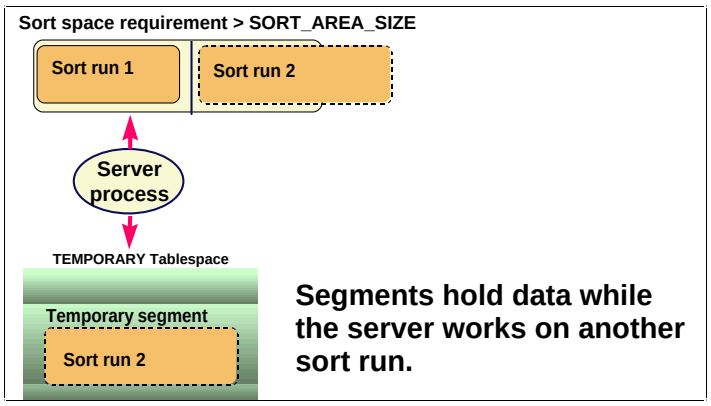
Step 4 eliminates migrated rows, because migration only occurs during an UPDATE.

11. Tuning Sorts Operation

Operations Requiring Sort

- Index creation
- Parallel insert operation involving index maintenance
- ORDER BY or GROUP BY clauses
- DISTINCT values selection
- UNION, INTERSECT, or MINUS operators use
- Sort-Merge joins
- ANALYZE command execution

Sort Process

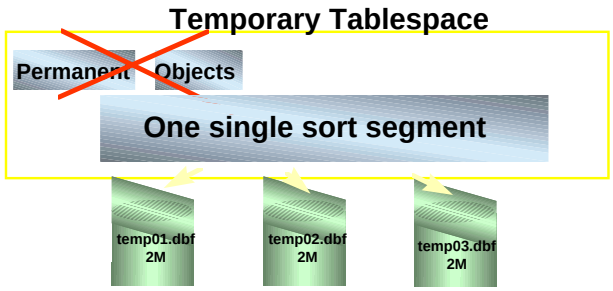


Oracle server sorts in memory if the work can be done within an area smaller than the value (in bytes) of the parameter **SORT_AREA_SIZE**. If the sort needs more space than SORT_AREA_SIZE.

1. The data is split into smaller pieces, called sort runs; each piece is sorted individually.
2. The server process writes pieces to temporary segments on disk; these segments hold intermediate sort runs data while the server works on another sort run.
3. The sorted pieces are merged to produce the final result.

Temporary Space Segment

- Is created by first sort
- Extends as demands are made on it
- Comprises extents, used by different sorts
- Is described in SGA in Sort Extent Pool (SEP)



Is created with CREATE TABLESPACE ... TEMPORARY command

Tuning Techniques

Avoid sort operations whenever possible	• Use NOSORT to create indexes • Use UNION ALL instead of UNION • Use index access for table joins • Create indexes on columns referenced in ORDER BY clause • Select the columns for analysis • Prefer ESTIMATE to COMPUTE for large objects

Ensure that sorting is done in memory where possible	`SQL> select disk.value "Disk", mem.value "Mem",`
	`2> (disk.value/mem.value)*100 "Ratio"`
	`3> from v$sysstat mem, v$sysstat disk`
	`4> where mem.name = 'sorts (memory)'`
	`5> and disk.name = 'sorts (disk)';`
	
	`Disk Mem Ratio`
	`--------- --------- ---------`
	`23 206 11.165049`
	
	• The ratio of disk sorts to memory sorts should be less than **5%.**
	• Increase the size of SORT_AREA_SIZE if the ratio is > 5%.

Bypass the buffer cache for large sorts	If your application uses many large sorts, you can write the sort runs directly to disk by setting the parameter **SORT_DIRECT_WRITES** to TRUE or AUTO. If you use direct writes, each sort uses its own memory buffers. It does not use the buffer cache. The sort writes an entire buffer for each I/O operation.
Allocate temporary space appropriately	Since sorts are done in memory if they are smaller than SORT_AREA_SIZE, you should consider this value when setting extent sizes: • Choose INITIAL and NEXT values as integer multiples of SORT_AREA_SIZE allowing an extra block for the extent header. • Set PCTINCREASE to 0.

12. Tuning Rollback Segment

Tuning Goals

- Transactions should never wait for access to rollback segments.
- Rollback segments should not extend during normal running.
- Users and utilities should try to use less rollback.
- No transaction should ever run out of rollback space.
- Readers should always see the read-consistent images they need.

Measuring performance of rollback segments

Header Contention	**From V$ROLLSTAT** **SQL> select sum(waits)* 100 /sum(gets) "Ratio",** **2> sum(waits) "Waits", sum(gets) "Gets"** **3> from v$rollstat;** **Ratio Waits Gets** **--- --- ---** **0.296736 5 1685** • The ratio of the sum of WAITS to the sum of GETS should be less than 5%. • If not, create more rollback segments. **From report.txt** This section of the `report.txt` output displays the same information as the WAITS and GETS columns of the V$ROLLSTAT view, but the statistics cover a longer period of time. **UNDO_SEGMENT TRANS_TBL_GETS TRANS_TBL_WAITS** **--- --- ---** **0 8 0** **1 749 118** **2 878 95**

Rollback Segment Contention	Compare the number of waits for each class of block with the total number of requests for data over the same period of time. **SQL> select class, count from v$waitstat** **2> where class like '%undo%';** **CLASS COUNT** **------------------ ------** **system undo header 0** **system undo block 0** **undo header 7** **undo block 0** **SQL> select sum(value) from v$sysstat** **2> where name = 'consistent gets';** **SUM(VALUE)** **----------** **47774** If the ratio is greater than 1%, consider creating more rollback segments.

Guidelines of Rollback Segment Number and Size

- OLTP : OLTP applications are characterized by frequent concurrent transactions, each of which modifies a small amount of data. Assign to OLTP transactions small rollback segments. The reasonable rule of thumb is one rollback segment for four transactions.
- BATCH : Assign large rollback segments to transactions that modify large amounts of data. Such transactions generate large rollback entries. If a rollback entry does not fit into a rollback segment, Oracle extends the segment. Dynamic extension reduces performance and should be avoided whenever possible. Allow for the growth of the rollback segments by creating them in large or autoextensible tablespaces, with unlimited .MAXEXTENTS
- For exceptionally long transactions, you may want to assign a large rollback segment using the following syntax:

  SQL> SET TRANSACTION USE ROLLBACK SEGMENT large_rbs;
- Choose the INITIAL storage parameter from the list 8 KB, 16 KB, 32 KB, 64 KB for small transactions, and 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB and so on for larger transactions. Size them large enough to prevent wrapping (an entry wraps into the next extent when it cannot find enough space in the current extent).
- Use the same value for NEXT as for INITIAL. Since PCTINCREASE is 0, all the other extents will have the same size as the NEXT.
- Make all your rollback segments the same size. If you do not, they are likely to become the same size over time anyway. The smaller ones extend as they are used by large transactions.
- Set MINEXTENTS to 20. This makes it unlikely that the rollback segment needs to grab another extent because the extent that it should move into is still being used by an active transaction.

Transaction Size and Rollback Segments

INSERT command	Inserts use little rollback space; only the ROWID is kept.
UPDATE command	The amount used for updates depends on how many columns are being updated.
DELETE command	Deletes are expensive for rollback segments; they need to store the actual row itself. If you can use TRUNCATE instead, performance improves.
Index maintenance	Indexed values generate more rollback, because the server process must change values in the index as well as in the table. For updates on indexed columns, the Oracle server records in the rollback segment the old data value, the old index value, and the new index value.

Using Less Rollback

- The design of the application should allow users to commit regularly.
- Developers should not code long transactions.
- Import
  - Set COMMIT = Y
  - Size the set of rows with BUFFER_SIZE
- Export
  - Set CONSISTENT=N
- SQL*Loader
  - Set the COMMIT intervals with ROWS

13. Monitoring and Detecting Lock Contention

Locking Management

The Oracle server automatically manages locking. Oracle's default locking mechanisms lock data at the lowest level of restrictiveness to guarantee data consistency while allowing the highest degree of data concurrency. Oracle always lock at the lowest and least restrictive level, the row level, and not at the table level during DML statements.

Two Main Types of Locks

DML or data locks:

- Table level locks
- Partition level locks
- Row level locks

DDL or dictionary locks

- Exclusive DDL locks:
  - DROP TABLE statement
  - ALTER TABLE statement
- Shared DDL locks:
  - CREATE PROCEDURE statement
  - AUDIT statement
- Breakable parse locks:
  - Invalidating Shared SQL area

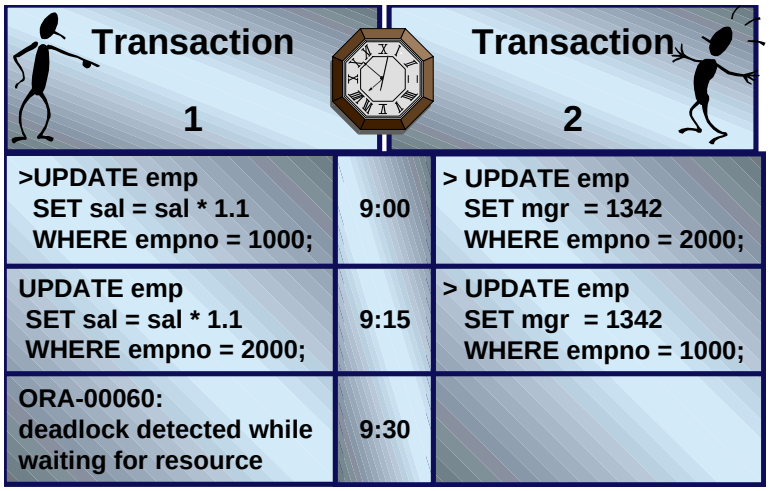
Possible Causes of Lock Contention

The Oracle server locks are inexpensive and efficient, and most sites do not have problems with locking. If locks do cause contention, it is often because

- Developers have coded in unnecessarily high locking levels
- Developers have coded in unnecessarily long transactions
- Users are not committing changes when they should
- The application uses the Oracle server in conjunction with other products that impose higher locking levels

Deadlocks

A ***deadlock*** can arise when two or more users wait for data locked by each other.



Assuming the second update in Transaction 1 detects the deadlock, the Oracle Server rolls back that statement and returns the message. Although the statement which caused the deadlock is rolled back, the transaction is not and you should receive an ORA-00060 error. Your next action should be to roll back the remainder of the transaction.
A deadlock situation will be recorded in a trace file in the USER_DUMP_DEST directory. It is advisable to monitor trace files for deadlock errors to determine if there are problems with the application. The trace file will contain the row ids of the locking rows.

14. Contention Issues

What is Latches Contention

In an Oracle environment, memory structures are held in a consistent state for a short period while a process is accessing them. This is necessary to ensure that the structure does not change while it is being accessed.
Latches are utilized to ensure that these structures do not change. When multiple processes attempt to acquire these latches, contention exists.
The object in tuning for latch contention is to minimize the contention between processes when latches are required.

Contention of Redo Copy and Redo Allocation Latch

Terminology	Different types of latches exist in the Oracle environment. The statistics for each category of latch, WILLING-TO-WAIT and IMMEDIATE can be found in different columns in the V$LATCH view. **WILLING-TO-WAIT** : If the latch requested with a willing-to-wait request is not available, the requesting process waits a short time and requests the latch again. The process continues waiting and requesting until the latch is available. – **GETS**: Shows the number of successful willing-to-wait requests for a latch – **MISSES**: Shows the number of times an initial willing-to-wait request was unsuccessful – **SLEEPS**: Shows the number of times a process waited and requested a latch after an initial willing-to-wait request **IMMEDIATE**: If the latch requested with an immediate request is not available, the requesting process does not wait, but continues processing. – **IMMEDIATE GETS**: This column shows the number of successful immediate requests for each latch. – **IMMEDIATE MISSES**: This column shows the number of unsuccessful immediate requests for each latch.

Redo Log Buffer Latch Processing	Each Server Process: 1. Obtains the redo copy latch 2. Obtains the redo allocation latch 3. Performs allocation 4. Releases allocation 5. Copies under the copy latch 6. Releases the copy latch
Tuning Goal	• Minimize contention for the redo log copy latch by server processes that need to write to the redo log buffer. • Each server process should hold a redo log copy latch for as little time as possible.
Diagnostic Tools	V$LATCH
Guidelines of Tuning	• If MISSES/GETS > 1% or I_MISSES/I_GETS + I_MISSES > 1%: • Reduce contention for that latch: • Redo Copy Latch Increase **log_simultaneous_copies** • Redo Allocation Latch Decrease **log_small_entry_max_size**

Contention of LRU Latch

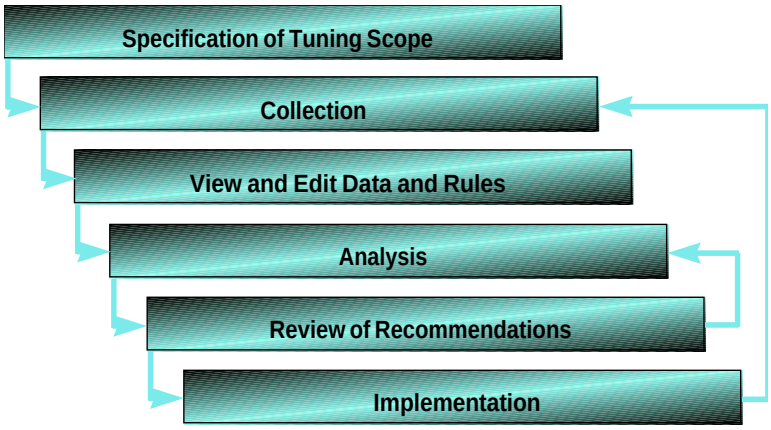
What is LRU latches	LRU latches regulate the least recently used (LRU) lists used by the database buffer cache. By default Oracle sets the number of LRU latches to one-half the number of CPUs, with a minimum of one. Each latch controls a minimum of 50 buffers.
Tuning Goal	Ensure there are a sufficient number of LRU latches for the data buffer cache so that contention between server processes is minimized. Balance the number of latches with the number of CPUs.
Diagnostic Tools	V$LATCHNAME V$SYSTEM_EVENT V$SESSION_EVENT V$BUFFER_POOL_STATISTICS
Guidelines of Tuning	If the Hit % for the LRU latch < 1%: Increase the number of LRU latches by setting the parameter **db_block_lru_latches** Maximum number of latches is the lower of: – number of cpus * 2 * 3 or – number of buffers / 50

Contention of Free Lists

What is Free List?	When an insert operation on an object occurs, the free list is used to determine what blocks are available for inserts. Many server processes can contend for the same free list if many inserts are occurring. This results in free list contention as server processes incur waits.
Tuning Goal	Ensure that there are a sufficient number of them to minimize contention between many server processes.
Diagnostic Tools	V$SESSION_WAIT V$SYSTEM_EVENT V$WAITSTAT DBA_EXTENTS
Guidelines of Tuning	1. Query v$session_wait 2. Query dba_extents 3. Get free lists for segment 4. Recreate the object in question

15. Tuning With Oracle Expert in OEM

Oracle Expert Tuning Methodology



Oracle Expert provides automated performance tuning with integrated rules.

Steps	Explanation
1. Setting the scope of the tuning session	Set the scope of the tuning session to tell Oracle Expert what aspects of you database you want to tune.
2. Collecting the data	To get comprehensive performance recommendations, Oracle Expert collects the following classes of data: database, instance, schema, environment, and workload.
3. Viewing and editing the collected data and rules	Once you have collected the various pieces of tuning data, you can view and edit that data. The data is organized in a hierarchical format. You can also view and edit the rules and attributes that Oracle Expert uses to make its recommendations.

4. Analyzing the data, and generating recommendations	When you have collected and edited the data as needed, Oracle Expert performs an analysis to generate tuning recommendations.
5. Reviewing the Oracle Expert recommendations	After Oracle Expert has analyzed the data, you can review the recommendations and decide which to accept. If you do not want to accept all the recommendations, have Oracle Expert generate a new recommendation. The new analysis takes into account the interdependencies of your preferences.
6. Generating scripts for implementing the recommendations.	When you are satisfied with all of your inputs and the resulting recommendations, Oracle Expert can generate parameter files and implementation scripts to implement at your convenience.

16. Optimizing for Load

Monitoring Dispatchers

```
SQL> SELECT network "Protocol",
  2> SUM(busy) / ( SUM(busy) + SUM(idle) ) "Total Busy Rate"
  3> FROM v$dispatcher
  4> GROUP BY network;
```

```
Protocol   Total Busy Rate
--------   ---------------
decnet     .004589828
tcp        .029111042
```

- A value of 0.5 means that a dispatcher has been busy **50%** of the time. This is too high, and you would need to add more dispatchers using the following command.

```
ALTER SYSTEM SET mts_dispatchers = ‘protocol, number’;
```

Monitoring Shared Servers

- If you underestimate the number of shared servers you need, the Oracle8 server starts up more shared servers dynamically. The extra servers are removed again when they are idle. This means that there is less need to monitor servers than dispatchers.
- However, you may have started up more shared servers than you need. If you specify a value for MTS_SERVERS in the parameter file, the Oracle8 server does not kill these processes, even if they are idle, so you may want to check the loading.
- You may also want to find out whether the number of servers is getting near the value of MTS_MAX_SERVERS or (even worse) is bringing the number of processes close to the value of the PROCESSES parameter. You remove shared servers by using the following command:

```
ALTER SYSTEM SET MTS_SERVERS = number;
```

Shared Servers and Memory Usage

- Some user information goes into the shared pool
- Overall memory demand should still decrease
- Shared servers use UGA(User Global Area) for sorts
- UGA stored in Large Pool if configured

Frequently Asked Questions in Oracle8 : Performance Tuning

Q1. When is cost base optimization triggered?

Generally, the CBO can change the execution plan when you:

1. Change statistics of objects by doing an ANALYZE;
2. Change some initialization parameteres (for example: hash_join_enabled,sort_area_size,db_file_multiblock_read_count).

Q2. How can one optimize %XYZ% queries?

It's possible to improve these queries by forcing the optimizer (using hints) to scan all the entries from the index instead of the table.

If the index is physically smaller than the table (usually the case) then it can take less time to scan the entire index than to scan the entire table..