

NEW!

**CramSession**Comprehensive **Study Guides**

A+  
Adobe  
C++  
Cisco CCNA

**Your Trusted  
Study Resource  
for  
Technical  
Certifications**

Written by experts.  
The most popular  
study guides  
on the web.

In Versatile  
PDF file format

Check out these great features  
at [www.cramsession.com](http://www.cramsession.com)

> **Discussion Boards**

<http://boards.cramsession.com>

> **Info Center**

<http://infocenter.cramsession.com>

> **SkillDrill**

<http://www.skilldrill.com>

> **Newsletters**

<http://newsletters.cramsession.com/default.asp>

> **CramChallenge Questions**

<http://newsletters.cramsession.com/signup/default.asp#cramchallenge>

> **Discounts & Freebies**

<http://newsletters.cramsession.com/signup/ProdInfo.asp>

INFORMATION TECHNOLOGY

# Oracle 8 Performance Tuning

Version 3.0.0

Microsoft Office  
Microsoft Windows 2000  
Microsoft Windows XP  
Network Security  
Network+  
Networking  
Nortel Networks  
Novell  
Oracle  
Proxy Server  
Red Hat Linux  
SAIR Linux  
SANS  
SCO  
Server+  
SQL  
Sun Solaris  
Unix  
Visual Basic  
Web Design

**Notice:** While every precaution has been taken in the preparation of this material, neither the author nor Cramsession.com assumes any liability in the event of loss or damage directly or indirectly caused by any inaccuracies or incompleteness of the material contained in this document. The information in this document is provided and distributed "as-is", without any expressed or implied warranty. Your use of the information in this document is solely at your own risk, and Cramsession.com cannot be held liable for any damages incurred through the use of this material. The use of product names in this work is for information purposes only, and does not constitute an endorsement by, or affiliation with Cramsession.com. Product names used in this work may be registered trademarks of their manufacturers. This document is protected under US and international copyright laws and is intended for individual, personal use only.

For more details, visit our [legal page](#).



**CramSession**  
Prepare for Success!



# Oracle 8 Performance Tuning

Version 3.0.0

**NOTICE:** Got the **NEWest Version?**  
Make sure by clicking here!

## Abstract:

This study guide will help you to prepare for the Oracle Exam 1Z0-014, Oracle 8.0 Performance Tuning. Exam topics include Business Requirements, Utilities and Dynamic Performance Views, Database Configuration, Tuning SQL, the Shared Pool, and the Buffer Cache, Sort Operations, and Monitoring and Detecting Lock Contention.

Find even more help here:

- > **Feedback & Discussion Board for this exam**
- > Read & Write Reviews of this study guide
- > Rate this Cramsession study guide



## Contents:

Ratios To Memorize .....	3
Business Requirements and Tuning .....	3
Oracle Alert, Trace Files, and Events .....	4
Utilities and Dynamic Performance Views .....	5
Tuning Considerations For Different Applications .....	6
SQL Tuning .....	7
Tuning the Shared Pool .....	9
Tuning the Buffer Cache .....	10
Latch and Contention Issues .....	12
Tuning the Redo Log Buffer .....	13
Database Configuration and I/O Issues .....	14
Using Oracle Blocks Efficiently .....	16
Optimize Sort Operations .....	17
Rollback Segment Tuning .....	18
Monitoring and Detecting Lock Contention .....	19
Tuning with Oracle Expert .....	20



### Ratios To Memorize

1. V\$LIBRARYCACHE gives the pins/reload ratio for the library cache. The GETHITRATIO column should be .95 or higher. The RELOADS column should be equal to or less than one percent of the PINS column.
2. The GETMISSES to GETS columns in V\$ROWCACHE should have a ratio less than 15% percent.
3. The buffer cache hit ratio should be 90% or higher.
4. The hit ratio for a latch should be .98 or higher. Anything lower indicates latch contention.
5. If in V\$LATCH the MISSES to GETS ratio is higher than 1%, there is a problem with redo allocation latch contention.
6. If in V\$LATCH the IMMEDIATE\_MISSES to IMMEDIATE\_GETS ratio is higher than 1%, there is a problem with redo copy latch contention.
7. Query V\$WAITSTAT to see if there is rollback segment header block contention. A value over zero in the UNDO HEADER column (undo = rollback) indicates contention.
8. Query V\$SYSTAT to see whether sorts are occurring on disk or in memory. For OLTP systems 95% of sorts should be happening in memory.

### Business Requirements and Tuning

- Before beginning Oracle specific tuning you should check for bottlenecks on the operating system level. In particular, determine if there is excessive paging to the swap file and whether other applications are contending for system resources (such as RAM, CPU and disks) with Oracle. Eliminate use of the swap file altogether if possible. Ideally, move non-Oracle applications to another server. If this is not possible at least ensure these applications use their own set of hard disk spindles.
- As the Oracle DBA you are the main person approached to improve rdbms performance. The overall goal of tuning should be to make a noticeable difference in performance from the end user's point of view. Tuning is not a one-time job but rather an iterative process which you do over and over, refining each time around.
- You should tune in the following order, which moves from the most local to most global effect as well as the most to least amount of performance gain. You always want to make the most localized changes possible before making changes that will effect the entire database system.
- **Example:** Changing one SQL query to start using indexed columns will potentially have a greater performance payoff and will be less likely to adversely affect other users and applications than changing the size of the db buffer cache.



1. **Database Design:** Many times you will not be able to participate at this level but when possible, being able to lay down a good design initially will help prevent a lot of performance nightmares from ever occurring.
  2. **Application SQL Statements:** This is the overriding cause of poor performance. Work with application programmers to write SQL statements most efficiently.
  3. **Memory Tuning:** Make sure the SGA is tuned correctly. Neither too small nor so large it causes paging to the swap file. The SGA is composed of three parts: 1) the data block buffer cache, 2) the redo log cache and 3) the shared pool. All three need to be tuned properly for optimal performance.
  4. **Disk I/O:** Make sure your tablespaces are laid out correctly so that you don't have one type of tablespace adversely affecting another. An example of this is putting a rollback segment tablespace on the same disk as your data tablespace. Also verify you have storage parameters set optimally.
  5. **Internal Memory Structure Contention:** Verify one internal process isn't waiting upon another for long periods of time. This is done by monitoring and adjusting latches correctly.
- Oracle defines **paging** as when a block of data is moved from physical memory to virtual memory on disk. Oracle defines **swapping** as when an entire process is moved from physical memory to virtual memory.

### Oracle Alert, Trace Files, and Events

- Trace files are Oracle log files that give the status of Oracle events. The main overall trace file that monitors the instance is called the **alert.log**. It is located in the directory specified by the **BACKGROUND\_DUMP\_DEST** initialization parameter. Background process (e.g., SMON, DBWR, etc.) trace files go here too.
- Trace files (except for alert.log) usually end with a .trc extension.
- User process trace files go in the directory specified by the **USER\_DUMP\_DEST** initialization parameter.
- The initialization parameter **MAX\_DUMP\_FILE\_SIZE** specifies the maximum size for user process trace files. In contrast, the alert.log trace file cannot have a specified size limit and must be truncated manually.
- The V\$ views are dynamic performance views based on the X\$ tables. X\$ tables are internal tables which hold information about the instance. Both are owned by SYS and populated at instance startup. V\$ views can be seen by anyone with the "**SELECT ANY TABLE**" object privilege while X\$ tables can only be viewed by SYS.
- **V\$SYSTEM\_EVENT**, **V\$SESSION\_EVENT**, **V\$SESSION\_WAIT** are three views which show information about processes waiting for resources.
- **V\$SYSTEM\_EVENT** shows total waits for events since the database started.



- **V\$SESSION\_EVENT** shows total waits for events since the database started grouped by session.
- **V\$SESSION\_WAIT** shows the most information about event waits per session.
- If the **WAIT\_TIME** column of **V\$SESSION\_WAIT** has a value over 0 this indicates the session's last wait time. If the value is zero, the session is currently waiting. If the value is -1, the last wait was so small it wasn't measurable. If the value is -2, you need to enable **TIMED\_STATISTICS** (see below) because Oracle isn't currently gather any time information.

### Utilities and Dynamic Performance Views

- **UTLBSTAT.SQL** and **UTLESTAT.SQL** (B for begin and E for end) are scripts for collecting a variety of performance statistics. These scripts are located in \$ORACLE\_HOME/rdbms/admin.
- Before running **UTLBSTAT.SQL** and **UTLESTAT.SQL** you should set the initialization parameter **TIMED\_STATISTICS** equal to **TRUE**. This will give CPU time statistics. Alternatively you can enable and disable **TIMED\_STATISTICS** on the fly by issuing: "**ALTER SYSTEM SET TIMED\_STATISTICS=TRUE**" (or FALSE).
- **UTLBSTAT.SQL** and **UTLESTAT.SQL** must be run from Server Manager (not SQL\*Plus) either connecting **internal** or as **sysdba**. First run **UTLBSTAT.SQL** after the instance has been running for a while through a full spectrum of normal usage. Let it run for a given amount of time. Then end the statistics gathering by running **UTLESTAT.SQL**. This will generate a file called '**report.txt**'.
- Make sure to turn off **TIMED\_STATISTICS** after collecting statistics since this slows down the database. Also, you need to take into consideration this overhead when reading the results of **report.txt**. For example: the numbers from **report.txt** will show the database running at 90% of its capability, not 100%.
- If you shut down the database while running **UTLBSTAT.SQL** but before running **UTLESTAT.SQL**, you need to start the entire process over again.
- **Report.txt** contains the following sections:
  - Library cache statistics.
  - Numbers on a variety of statistics for the duration of the time **UTLBSTAT.SQL** was gathering data.
  - System wide wait events.
  - Average length of the dirty buffer write queue.
  - I/O statistics by datafile and tablespace.
  - Latch statistics.
  - Rollback segment statistics.
  - The current init<sid>.ora parameters.



- Row (dictionary) cache statistics.
- There is a set of GUI applications that come with the Oracle Enterprise Edition version of Oracle Enterprise Manager called the Oracle Performance Pack. These programs help monitor and optimize database performance:
  - **Oracle Expert:** assists in configuring and tuning your database (see below section for more detail).
  - **Lock Manager:** monitors locks.
  - **Performance Manager:** lets you monitor real time performance statistics and view them in various ways.
  - **Tablespace Manager:** lets you monitor segment storage in tablespaces. Also lets you defragment tablespaces (know as coalescing).
  - **TopSessions:** like the Unix 'top' command, lets you monitor the top resource intensive user sessions.
  - **Oracle Trace:** lets you monitor performance within user applications.
- **V\$SYSSTAT** is the main view for system performance on the instance level. You can also use V\$SYSSTAT to monitor client-server traffic.
- **V\$SESSION** gives connection information for all user sessions.
- 

### Tuning Considerations For Different Applications

- Know the difference between **OLTP**, **DSS** and **hybrid** systems.
- **OLTP** stands for online transaction processing system and is characterized by a high volume of inserts, updates and deletes (DML). The data is constantly changing in an OLTP system and the system may require 24x7 uptime. An example of an OLTP database is an ecommerce book site that handles online orders.
- **DSS** stands for decision support system. Is also commonly known as a data warehouse or OLAP: online analytical processing system. DSS is a read-only database consisting of massive amounts of data. Only selects are performed from the user end. Joins and full table scans are common. The data is not dynamic but rather is updated periodically. An example of an OLAP database is a DMV record archive of all automobile registrations within the last five years, updated on a monthly basis.
- In DSS systems, since most queries are full table scans, you may consider dropping the use of indexes altogether on some tables to improve performance.
- A **hybrid** system would be one that is both an OLTP database and a DSS database but at different times. An example of a hybrid database is a purchase order database that is used for querying during the daytime. But at



night, new orders are inserted, current ones are modified, and fulfilled orders are deleted.

- A good way to deal with a hybrid system is to have two sets of parameter files, one optimized for OLTP and one for DSS. When it comes time to switch from one use to the other, stop the database and start with the new parameter settings.

## SQL Tuning

- There are two optimizers built into Oracle: **Rules Based** and **Cost Based**.
- The **Rules Based Optimizer** (aka **RBO**) is a set of 15 rules Oracle uses to determine the fastest path of execution for a given SQL statement.
- The **Cost Based Optimizer** (aka **CBO**) determines all possible paths of execution and assigns a cost to each one. It chooses what it finds to be the least expensive path. You must be running **ANALYZE** on your tables and indexes to generate statistics to use the **Cost Based Optimizer**.
- It is very important to update statistics on a regular basis to provide meaningful data for the CBO work with. Performance can suffer tremendously by using outdated statistics.
- You can set the CBO goal to either 1) fastest overall throughput for a SQL statement or 2) fastest initial response time for a SQL statement.
- You set the optimizer method at the instance level by specifying the **OPTIMIZER\_MODE** initialization parameter. The possible values are:
  - **CHOOSE**: Oracle decides whether to use CBO or RBO. If statistics are present for any table in a SQL statement, Oracle will use CBO with the goal of fastest overall throughput. If no statistics are available, Oracle will use RBO. This is the default value.
  - **RULE**: Oracle will use **RBO** all the time whether or not statistics are present.
  - **ALL\_ROWS**: Oracle will use CBO with the goal of fastest statement throughput whether or not statistics are present (or current).
  - **FIRST\_ROWS**: For all statements, Oracle will use CBO with the goal of fastest initial response time.
- You can set the optimizer at the session level by issuing the command:
  - **ALTER SESSION SET OPTIMIZER GOAL=*optimizer mode***
- Use **EXPLAIN PLAN FOR *sql statement*** to see how the optimizer is executing your statement. This puts the execution plan in the **PLAN\_TABLE**.
- For each user that needs to run **EXPLAIN PLAN** you need to first run the **UTLXPLAN.SQL** script to create the **PLAN\_TABLE** table in their schema.
- Execution plans are read inside out, top to bottom.



- **SET AUTOTRACE ON** to automatically obtain every statement's execution plan.
- **SQL Trace** is used to gather user session statistics. It generates a trace file which is generated in **USER\_DUMP\_DEST**. It contains the same data as **AUTOTRACE** with the following additional information:
  - Parse, fetch and execute counts
  - Cpu and elapsed time
  - Number of logical and physical reads
  - Number of rows processed
  - Library cache misses
- **SQL Trace** can be turned on at the instance level by setting the initialization parameter **SQL\_TRACE=TRUE**.
- SQL Trace can also be turned on for an individual session by issuing: "**ALTER SESSION SET SQL\_TRACE=TRUE;**"
- -You can also turn SQL Trace on in the current session by executing the procedure **SYS.DBMS\_SESSION.SET\_SQL\_TRACE(TRUE)**.
- Finally, you can turn on **SQL Trace** for another session by executing the procedure: **SYS.DBMS\_SESSION.SET\_SQL\_TRACE\_IN\_SESSION(sid, serial #,TRUE)**.
- **TKPROF** is used to read the output of a trace file created by **SQL Trace**.
- The **DMBS\_APPLICATION\_INFO** package is created by the **DBMSUTIL.SQL**. It allows you to track resource usage and performance data for PL/SQL procedures. **DBMSUTIL.SQL** is called by the **CATPROC.SQL** script so it should already exist in your database.
- **Star schemas** are commonly used in DSS databases and consist of one large **fact table** and several small **lookup tables** (aka **dimension tables**). The fact table holds all the data while the lookup tables hold more detailed information about a given column in the fact table.
- An example of a star schema is a purchase order table which has among others, the following columns: **part\_number** and **shipping\_number**. There is a corresponding lookup table called **part\_number** which gives detailed item information. Likewise, there is another lookup table called **shipping\_number** which gives detailed information about how each order was shipped.
- Each lookup table has a **primary key** to **foreign key** relationship to the fact table (a child-parent relationship). However, in a star schema there is no relation between the various lookup tables among themselves. Hence, the "star" formation.
- A **star query** is a join between a fact table and one or more lookup tables.
- Star queries can only be optimized using the cost based optimizer; there are no rules for star queries in RBO.
- A **hash join** is a special type of equijoin that works well with joins between a large and small table. Oracle loads up all the rows from each table in the join



- into memory and breaks them up into partitions. Next, Oracle takes a partition from the smaller of the two tables and creates a hash table out of it. Oracle then uses the corresponding partition from the larger table to query the hash table with. This makes for extremely fast joins.
- The initialization parameter **HASH\_JOINED\_ENABLED** controls whether hash joins are allowed. The default value is TRUE. This can also be changed at the session level via ALTER SESSION or at the statement level via the /\*+ USE\_HASH (tablename) \*/ hint.
  - Like star queries, hash joins can only be done using CBO.
  - Clauses that can create poor SQL response time: **DISTINCT** (this causes sorts), **HAVING** (a group function - use **WHERE** instead) and **CONNECT BY** (performs a tree walk).

### Tuning the Shared Pool

- The **shared pool** is composed of the 1) **library cache**, 2) **row cache** and if **MTS** is used, 3) the **User Global Area (UGA)**.
- The size of the shared pool is set by the initialization parameter **SHARED\_POOL\_SIZE** (in bytes).
- The **library cache** contains shared parse information for SQL statements. This is the chief area to monitor in the shared pool.
- Programmers can use PL/SQL packages to promote SQL statement reuse and increase library cache hits.
- Programmers can use bind variables instead of constants to promote SQL reuse. This is useful for OLTP but not DSS.
- **V\$LIBRARYCACHE** gives the pins to reload ratio for the library cache. The **GETHITRATIO** column should be .95 or higher.
- In **V\$LIBRARYCACHE**, the value for **RELOADS** should be equal to or less than one percent of the value of **PINS**. If it is higher, you should increase the size of the shared pool.
- **V\$SQLAREA** gives information about shared cursors and the first part of the SQL statements they contain.
- **V\$DB\_OBJECT\_CACHE** gives information about PL/SQL packages and views. The **SHAREABLE\_MEM** column specifies how much memory of the library cache an object is consuming.
- **V\$SQLTEXT** gives the full SQL statements contained in shared cursors.
- **V\$DB\_OBJECT\_CACHE** gives detailed information about cached tables, rows and PL/SQL objects.
- The row (dictionary) cache stores data dictionary information.
- The **V\$ROWCACHE** view gives the hit/miss ratio for the dictionary cache.
- The **GETMISSES** to **GETS** columns in **V\$ROWCACHE** should have a ratio of less than 15% percent.
- This ratio is expressed in **report.txt** as **GET\_MISS** to **GET\_REQS**.



- You can pin **packages, procedures, triggers** and **cursors** into the shared pool using **DBMS\_SHARED\_POOL.KEEP( )**.
- The **DBMS\_SHARED\_POOL** package is created by running **DBMSPOOL.SQL** and **PRVTPPOOL.SQL**.
- The initialization parameter **SHARED\_POOL\_RESERVED\_SIZE** specifies how much of the shared pool you want to set aside for the **reserved list**. This is an area of the library cache to store large objects in. Objects smaller than the value specified by the initialization parameter **SHARED\_POOL\_RESERVED\_MIN\_ALLOC** will not be allowed on the reserved list.
- The **User Global Area (UGA)** is an additional third area of the shared pool when Oracle runs in **Multithreaded Server (MTS)** mode. When running in MTS mode, user information stored in the PGA in dedicated server mode is stored in the UGA instead. Hence, the UGA needs to be taken into consideration for overall sizing of the shared pool. Specifically you need to calculate the additional amount of memory required by the shared server sessions and open cursors.
- By default, there can be a maximum of 50 open (i.e., cached) cursors per user.
- By default, there are 250 bytes per user per open cursor for the shared pool.

### Tuning the Buffer Cache

- The size of the **buffer cache** is determined by: **DB\_BLOCK\_SIZE \* DB\_BLOCK\_BUFFERS**.
- **DB\_BLOCK\_SIZE** cannot be changed without recreating the database.
- The buffer cache contains the **dirty buffer write queue**, which holds dirty block buffers (i.e., modified blocks) until they can be written out to disk. A **least recently used (LRU)** algorithm is used to decide which buffers to move out of the buffer cache so that new buffers can be read in.
- **DBWR** is the background process that writes the dirty block buffers (located in the dirty buffer write queue) to disk.
- Server processes (**S000..S999**) read data blocks from datafiles into the buffer cache. Server processes check first to determine if the data blocks already exist in the buffer cache.
- The **V\$SYSSTAT** view shows the **buffer cache hit ratio**.
- The buffer cache hit ratio should be 90% or higher.
- The **V\$RECENT\_BUCKET** view shows the effects of increasing the buffer cache by x amount of blocks. The value in the **ROWNUM** column shows the number of proposed block additions minus one (it starts with zero). The value in the **COUNT** column shows how many more buffer cache hits you will get by increasing the buffer cache by the proposed amount of blocks.
- To use **V\$RECENT\_BUCKET**, stop the database and set the **DB\_BLOCK\_LRU\_EXTENDED\_STATISTICS** initialization parameter to the



- maximum number of blocks you are considering adding. Next, start the database. After the instance has been running for a while under normal conditions query **V\$RECENT\_BUCKET**.
- **V\$CURRENT\_BUCKET** shows the effect of decreasing the buffer cache. The value in the **ROWNUM** column shows the number of proposed block subtractions. The value in **COUNT** column shows how many buffer cache hits you would still have after this decrease.
  - To use **V\$CURRENT\_BUCKET**, stop the database and set the **DB\_BLOCK\_LRU\_STATISTICS** initialization parameter to **TRUE**. Next, start the database. After the instance has been running for a while under normal conditions query **V\$CURRENT\_BUCKET**.
  - Gathering statistics for **V\$RECENT\_BUCKET** and **V\$CURRENT\_BUCKET** consumes CPU cycles and hence is a performance hit to the database. Therefore, you should disable these views by resetting their respective initialization parameters to the default value (i.e., 0 and FALSE) as soon as you are finished using them.
  - You should increase the buffer cache if:
    - The cache hit ratio is too low (below 90%).
    - There are increasing values for **free buffer inspected** (check **V\$SYSSTAT** or report.txt).
    - There is contention for the cache buffer's chain latch (check **V\$LATCH**).
    - There are buffer busy waits (check **V\$SYSTEM\_EVENT** or **V\$SESSION\_WAIT**).
  - On multi-CPU systems, **DB\_BLOCK\_LRU\_LATCHES** specifies how many **LRU latches** (aka **buffer chain latches**) are available for the database. The default value is one half the number of CPUs and can be increased to double the number of CPUs. On single CPU systems you can only have one LRU latch.
  - A new feature in Oracle 8 is optional **multiple buffer pools** within the buffer cache. Using these buffer pools lets you explicitly define which objects you want to remain in the buffer cache and which you want to quickly unload. There are three buffer pools:
    1. **Keep Buffer Pool**: For objects you want to remain in the buffer cache as long as possible.
    2. **Recycle Buffer Pool**: For objects you want unloaded from the buffer cache as soon as the transaction that uses them is complete.
    3. **Default Buffer Pool**: The default buffer pool that db blocks are stored in if not otherwise specified. Applications based on Oracle 7.3 and earlier will always use the default buffer pool.



- By default, the keep and recycle buffer pools are not enabled. To specify space for them, you must define the initialization parameters **BUFFER\_POOL\_KEEP** and **BUFFER\_POOL\_RECYCLE**. You must also specify how many buffer chain latches you want allocated for each pool. **NOTE:** You must have at least 50 buffer blocks for every LRU latch you assign.
- **Syntax:** BUFFER\_POOL\_KEEP(buffers:50000, lru\_latches:5)
- The size of the default buffer pool is not explicitly defined. Instead it is equal to the size of the entire buffer cache (value of DB\_BLOCK\_BUFFERS) minus the number of blocks allocated to the keep buffer pool and/or the recycle buffer pool.
- The buffer blocks for the keep and recycle buffer pools are taken from the buffer cache. Likewise, their LRU latches are taken from the latches allocated for the entire buffer cache as specified by DB\_BLOCK\_LRU\_LATCHES. Therefore, neither the number of buffer blocks nor the number of LRU latches allocated to the keep and recycle buffer pools can - taken together - equal or exceed the values allocated to the buffer cache as a whole. If you do over-allocate either db blocks or LRU latches by accident, the database will not mount.
- Use the **CACHE hint** (/\*+ CACHE \*/) in your SQL statements to ensure that the selected table will be cached.
- You can also use the **CACHE clause** when creating a table to make sure it will always be cached.
- Only cache small, frequently used tables. By default, large tables will fill the buffer cache (unless assigned to the keep or recycle buffer pool).
- **V\$CACHE** shows what objects are currently in the buffer cache. Run **CATPARR.SQL** to create this view. While CATPARR.SQL is for Parallel Server environments, V\$CACHE is useful in single instance environments as well.
- The initialization parameter **CACHE\_SIZE\_THRESHOLD** specifies how many blocks per table will be cached. This is helpful to prevent buffer cache crowding.

### Latch and Contention Issues

- What locks are to tables, latches are to internal memory structures. Latches regulate access to instance resources by system and user processes.
- The **V\$LATCH** view gives real time detailed latch information. The **HIT\_RATIO** column gives the hit ratio for each latch. The **SLEEPS** column indicates how many times a process has had to wait to obtain a latch. Latch statistics are also listed in **report.txt**.
- The hit ratio for a latch should be .98 or higher. Anything over indicates latch contention.



## Tuning the Redo Log Buffer

- The **LOG\_BUFFER** initialization parameter specifies how big (in bytes) the **redo log buffer** is.
- Query **V\$SYSTEM\_EVENT** to see if there are waits for '**log buffer space**'. If so, increase the redo log buffer size. You can also obtain this information from **V\$SYSSTAT** by looking at the number of '**redo buffer allocation retries**' there are for a given user process.
- Query **V\$SYSTEM\_EVENT** to determine if there are waits for '**log file parallel write**'. If so, this means there is I/O contention for redo log files. Stripe the redo log files across several disks to alleviate this problem.
- You can forego creating redo log information during **ALTER** and **CREATE** statements as well as SQL\*Loader direct-path loads by specifying the **NOLOGGING** clause. This greatly speeds up creation of large tables and indexes. NOLOGGING replaces the **UNRECOVERABLE** keyword used in Oracle 7.3.
- Since redo information is not generated for direct-path loads on databases in NOARCHIVELOG mode, the NOLOGGING option for SQL\*Loader direct-path loads is only applicable for databases in ARCHIVELOG mode.
- There are two types of latches associated with the redo log buffer. The first is the **redo allocation latch**. This latch is required for a process to allocate space for writing to the redo log buffer. There is only one redo allocation latch per instance. This ensures serial logging to the redo logs. On single CPU systems, once a process obtains the redo allocation latch, it allocates space in the redo log buffer and then writes the data to it in one step.
- The second type of latch is the **redo copy latch**. This latch exists only in multi-CPU systems. The redo copy latch is used to break up the allocation and writing processes into two steps. Processes needing to write data over a given amount of size (see below for specifics) must release the redo allocation latch and get a redo copy latch before writing to the redo log buffer. This frees up the redo allocation latch a lot faster than is possible on a single-CPU system.
- The **V\$LATCH** view give statistics about redo allocation latch and redo copy latch contention. For redo allocation latch requests (**WILLING\_TO\_WAIT** type) the columns to monitor are **GETS**, **MISSES** and **SLEEPS**.
- For redo copy latch requests (**IMMEDIATE** type) the columns to monitor are **IMMEDIATE\_GETS** and **IMMEDIATE\_MISSES**.
- If the **MISSES** to **GETS** ratio is higher than 1%, there is a problem with redo allocation latch contention.
- If the **IMMEDIATE\_MISSES** to **IMMEDIATE\_GETS** ratio is higher than 1%, there is a problem with redo copy latch contention.
- Any process with a redo log entry larger than **LOG\_SMALL\_ENTRY\_MAX\_SIZE** (specified in bytes) must give up the **redo allocation latch** and obtain a **redo copy latch** before writing its entry into the **redo log buffer**.



- So, to reduce contention for the redo allocation latch, decrease the value for **LOG\_SMALL\_ENTRY\_MAX\_SIZE**.
- The **LOG\_SIMULTANEOUS\_COPIES** initialization parameter specifies how many **redo copy latches** there are. The default value for this parameter is equal to number of CPUs on the system. However, you can increase this value up to twice the number of CPUs to help reduce redo copy latch contention.

### Database Configuration and I/O Issues

- Set the **LOG\_CHECKPOINT\_INTERVAL** initialization parameter (specified in OS block size) to a larger size than your redo logs and set **LOG\_CHECKPOINT\_TIMEOUT** to zero (the default) to ensure that checkpoints only occur at log switches.
- You can further reduce the frequency of checkpoints by increasing the size of the redo logs. However, the larger the redo logs, the longer it will take for recovery.
- Set the **LOG\_CHECKPOINTS\_TO\_ALERT** initialization parameter to **TRUE** so that checkpoint beginning and ending times are logged to the **alert.log** file.
- Avoid having small, frequent checkpoints since this will interrupt database operations unnecessarily (DBWR, LGWR and CKPT are all involved) and hinder performance. On the other hand, avoid making checkpoints so infrequent that when they do occur, they seize up the entire database for a noticeable amount of time. You want to avoid these "pregnant pauses" and tune checkpoints so they happen infrequently as possible without noticeably halting database operations.
- **DB\_BLOCK\_CHECKPOINT\_BATCH** specifies the maximum number of blocks that a DBWR process can write in a single batch during a checkpoint. Increasing this value can speed up checkpoint times, but making it too large can give poor response times as well.
- The initialization parameters **DISK\_ASYNCH\_IO** and **TAPE\_ASYNCH\_IO** specify whether the operating system supports asynchronous I/O for hard drives and tape drives respectively (most do). The default value is TRUE.
- If your operating system doesn't support asynchronous disk and/or tape I/O, you can enable them on the Oracle level by specifying the following initialization parameters:
  - **ARCH\_IO\_SLAVES**
  - **LGWR\_IO\_SLAVES**
  - **DBWR\_IO\_SLAVES**
  - **BACKUP\_DISK\_IO\_SLAVES**
  - **BACKUP\_TAPE\_IO\_SLAVES**



- These parameters start up multiple I/O server processes (called "slave processes") that attempt to have the same effect as asynchronous I/O. All of these I/O server processes are disabled by default and should only be used if **1)** there are bottlenecks in the above listed processes and **2)** your operating system either doesn't support asynchronous I/O or does so poorly.
- You can specify multiple DBWR processes with the **DB\_WRITER\_PROCESSES** initialization parameter. You can have up to ten simultaneous DBWR processes (**DBW0** through **DBW9**) to assist in writing data from the db buffer cache to datafiles. This can be very useful in servers with a high number of CPUs.
- **NOTE:** You cannot have multiple DBWR processes and DBWR I/O slaves at the same time. If you have a value other than zero for **DBWR\_IO\_SLAVES** you will only have one DBWR process regardless of the value for **DB\_WRITER\_PROCESSES**.
- The **SYSTEM** tablespace should only contain data dictionary objects, PL/SQL packages, triggers and the initial rollback segment.
- Always explicitly define the default tablespace and temporary tablespace when creating users. Otherwise the **SYSTEM** tablespace will be used. This is very bad and will kill performance due to sorts taking place in **SYSTEM**. Additionally if the **SYSTEM** tablespace fills up, the database will halt and you may have to recreate the instance!
- Your database should have a minimum of six tablespaces:
  1. **SYSTEM** for the data dictionary and initial rollback segment.
  2. **DATA** for tables.
  3. **INDEX** for indexes.
  4. **TEMP** for sorting.
  5. **RBS** for rollback segments.
  6. **USER** for user created tables.
- Rollback and temporary segments tend to fragment by expanding and contracting frequently. Therefore they should each have dedicated tablespaces and disks.
- You should place datafiles so that I/O is distributed evenly. Consider what sort of object segments each tablespace will contain. Put table and index tablespaces on separate disks.
- **V\$FILESTAT** tells the number of reads and write done to each datafile. Use this view and report.txt to determine if I/O is evenly distributed or not.
- Avoid heavy I/O on the **SYSTEM** tablespace. Heavy I/O may occur by not specifying the default and temporary tablespace when creating users. If so, users object and query sorts will be written and read from **SYSTEM**.



- If the row cache isn't big enough to allow the entire data dictionary to exist in memory, additional I/O will occur in the SYSTEM tablespace due to data dictionary reads.
- When using hardware or software striping, make sure the stripe block size is a multiple of the **DB\_FILE\_MULTIBLOCK\_READ\_COUNT** initialization parameter.
- Manually stripe tablespaces only if hardware or the operating system doesn't support it. You should only consider it if you are in an OLAP environment where a lot of full table scans are happening and you are using Parallel Query.
- You can manually stripe by spreading the tablespace over two datafiles, one on a different drive. Set **MINEXTENTS** greater than 1 and then create the table so that each extent fills up more than half of each datafile. You can also allocate extents to datafiles explicitly.
- **NOTE:** I don't know of any modern operating system or hardware architecture that doesn't allow for hardware or software RAID. However, I included the above information because you may be asked about this on the exam.

### Using Oracle Blocks Efficiently

- **PCTUSED** is relevant for deletes. If you have a lot of inserts and deletes on a table, set PCTUSED high. On DSS systems PCTUSED is irrelevant.
- **PCTFREE** and **PCTUSED** taken together should be less than 100.
- Set PCTFREE to zero in DSS environments.
- Setting PCTFREE too low in OLTP systems will cause **row migration**.
- **Migrated** and **chained rows** cause excessive disk I/O. So does inefficient use of block space which causes more blocks to be read than is necessary. The goal is to determine how your tables are being used and set the block storage parameters optimally, neither overcrowding nor wasting space.
- The **CHAIN\_CNT** column of **DBA\_TABLES** tells you how many chained or migrated rows a table has.
- The **CHAINED\_ROWS** table gives detailed information about each chained or migrated rows in a table. This table is created by the **UTLCHAIN.SQL** script. Use the **LIST CHAINED ROWS** clause of the **ANALYZE** command to populate the **CHAINED\_ROWS** table.
- Query **V\$WAITSTAT** to determine if **freelist** contention is high. This view doesn't tell you what tables are experiencing freelist contention. You need to figure this out on your own!
- If there is freelist contention for a table due to heavy usage, drop the table and recreate it with more freelists.
- Index blocks are updated sequentially and entries must be maintained in a certain order. Therefore, only if an index block is completely empty is it put back on the **freelist**. Since this causes inefficient space usage, rebuild



- indexes (**ALTER INDEX *index* REBUILD**) on high usage OLTP tables frequently.
- Freelists for a table (or index) are kept in its **segment header**.
  - **DB\_BLOCK\_SIZE** should be a multiple of the OS block size. Make it as large as possible.
  - Set **DB\_FILE\_MULTIBLOCK\_READ\_COUNT** to a multiple of **DB\_BLOCK\_SIZE**. This determines how many blocks are read at once in a full table scan. This is very important for DSS systems.
  - The **highwater mark** for a table is increased 5 blocks at a time. Full table scans read up through the high water mark.
  - **DELETE** does not reduce the highwater mark count on a table. **TRUNCATE** does. So does **ALTER TABLE *tablename* DEALLOCATE UNUSED**.

### Optimize Sort Operations

- Index creation causes sorting since when an index is built it needs to be sorted first.
- The following clauses cause sorting to occur: **ORDER BY, GROUP BY, DISTINCT, UNION, INTERSECT** and **MINUS**.
- Sorts in memory occur much faster than sorts on disk. The goal is to have as much sorting as possible happen in memory. However, no matter how much RAM a server has, large sorts will take place on disks (e.g., index rebuild on a million row table). This means that tuning the disk I/O for sorts is crucial for efficient database operations.
- Oracle will sort in memory if the sort can fit in the value specified by the initialization parameter **SORT\_AREA\_SIZE** (in bytes). If not, Oracle will break the sort into multiple **sort runs**. Only one set of the data will be in memory for each run. The rest of the sort data will be written to disk until it is needed.
- The **sort area** is part of the **PGA** in dedicated server mode and is part of the **UGA** in **MTS** mode.
- When a sort is complete the sort area will shrink to the size specified by **SORT\_AREA\_RETAINED\_SIZE**. Only make this value lower than **SORT\_AREA\_SIZE** if you are extremely short on RAM. The shrinking and growing of the sort area will give a performance hit to the system.
- You should create at least one temporary tablespace (**CREATE TABLESPACE *name* TEMPORARY**) to handle disk sorts. When creating a temporary tablespace, specify all extents to be the same size and a multiple of **SORT\_AREA\_SIZE**. **MAXEXTENTS** is not a valid storage parameter for temporary tablespaces.
- You should stripe temporary tablespaces over as many hard disk spindles as possible.



- There is only one sort segment per temporary tablespace. The sort segment will be created the first time a sort uses the tablespace and will grow as needed. Each sort can grab a separate extent from this segment as needed.
- When a process needs to sort on disk it looks in the **sort extent pool** (SEP) in the SGA to get a free extent.
- **V\$SORT\_SEGMENT** gives detailed information regarding sort extents. The **EXTENT\_HITS** column tells how many times a free segment was found in the sort extent pool.
- Query **V\$SYSTAT** to see whether sorts are occurring on disk or in memory.
- 95% of sorts should be happening in memory for OLTP systems.
- Set **SORT\_DIRECT\_WRITES** to **TRUE** if you have a lot of memory to allocate for sorts. Direct write sorts bypass the buffer cache and instead assign buffers directly to the server process performing the sort. While this option can speed up performance, it consumes more RAM and temporary extents than conventional sorts. Again, make sure your disk I/O is well tuned and evenly distributed!
- The initialization parameters **SORT\_WRITE\_BUFFERS** and **SORT\_WRITE\_BUFFER\_SIZE** specify how much memory to allocate for direct write sorts.
- **SORT\_WRITE\_BUFFERS** should be between 2 and 8.
- **SORT\_WRITE\_BUFFER\_SIZE** should be between 32K and 64K.
- The default value for **SORT\_DIRECT\_WRITES** is **AUTO**. This means that Oracle will use direct write buffers when SORT\_AREA\_SIZE is at least ten times the size of the minimum direct write buffer configuration for your operating system (usually 64K).
- Memory allocated to direct write buffers should be less than 1/10th the amount allocated for **SORT\_AREA\_SIZE**.

### Rollback Segment Tuning

- Rollback segments are used for the three Rs: transaction **rollback**, instance **recovery** and query **read consistency**.
- **Rule of 4** for sizing number of rollback segments: # of concurrent transactions divided by 4. If less than 4+4, round up to a multiple of 4. No more than 50 total.
- If you get an "**ORA-01555: snapshot too old (rollback segment too small)**" error, this means a rollback segment is too small for a given transaction to complete and that it has overwritten itself. You should increase the number of rollback segments or assign the transaction a large rollback segment.
- Use "**SET TRANSACTION USE ROLLBACK SEGMENT *rollback segment name***" to assign a specific rollback segment to a transaction. This should be done for long running batch jobs.
- Rollback segment extents should all be the same size.



- **MINEXTENTS** for rollback segments should be 20.
- **INITIAL** for rollback segments should be set to the power of 2.
- **OPTIMAL** should be set high enough to accommodate normal database activity.
- **DELETES** use the most rollback space. Next **UPDATES**, then **INSERTS**.
- **V\$SESSION** and **V\$TRANSACTION** show how much rollback information a transaction is creating.
- The **rollback transaction table** is in the **rollback segment header**.
- Query **V\$WAITSTAT** to see if there is rollback segment header block contention. A value over zero in the **UNDO HEADER** column (undo = rollback) indicates contention.

### Monitoring and Detecting Lock Contention

- **Enqueues** are what Oracle uses to manage locks.
- **DML** locks occur at the row level.
- The **V\$LOCK** view gives detailed lock information.
- A query does not hold a lock unless the **SELECT FOR UPDATE** statement is issued.
- **SELECT FOR UPDATE** statements give the user a **Row Shared (RS)** lock. This gives a **shared table lock** and an **exclusive row lock**.
- A **Share** lock on a table prevents **DML** operations happening to it.
- A **Share Row Exclusive** lock prevents **DML** and **SELECT...FOR UPDATE** on a table.
- Query **V\$SESSION** to see which row is causing lock contention.
- **DDL** locks are not usually in contention because they are held only briefly.
- The three types of locks mainly in use by applications are: **TM** (row exclusive table lock), **TX** (row exclusive lock) and **UL** (PL/SQL user locks).
- High level locking (i.e., locking a whole table instead of just a row) in application code can cause the majority of lock contention in a database. This may be because the application was coded for another rdbms such as Sybase or SQL Server.
- Long running transactions can also cause lock contention. Make sure users commit regularly and that application code is written to do so as well.
- To kill a user session which is holding a lock, query **V\$SESSION** and get the **SERIAL#** and **SID** (session ID). Next, issue **ALTER SYSTEM KILL SESSION serial #, sid**.
- **UTLLOCKT.SQL** shows a graph of which locks are being held. **CATBLOCK.SQL** must be run first to set up the views for this.
- When a **deadlock** is detected, Oracle will roll back the statement that caused the **deadlock** and issue an **ORA-00060** error message. **NOTE:** Oracle does not rollback the entire transaction - only the statement that detected the deadlock. You will have to manually rollback the rest of the transaction if necessary.



- When a deadlock occurs the **rowid** of the locking row is recorded in a trace file located in **USER\_DUMP\_DEST**.

### Tuning with Oracle Expert

- Oracle Expert is part of the Tuning Pack for Oracle Enterprise Manager. It assists in the initial configuration of a new database as well as collects data, analyzes it, and makes recommendations on an existing database. Oracle Expert can detect trends over a period of time and point out where performance bottlenecks lie.
- Oracle Expert uses the concept of a **tuning session** to focus in on a given portion of the database you want to examine. All data is collected and stored on the tuning session level. These sessions are saved to disk with an **.XDL** extension.

There are three scopes for tuning sessions:

1. **Application level:** examines SQL statements and access methods (index usage).
2. **Instance level:** examines SGA, operating system and I/O parameters.
3. **Structure level:** examines storage parameters and placement of tablespaces.
  - While you can choose more than one scope for a tuning session, it is normally most practical to keep it to one scope and then even further narrow the focus down to a few objects (e.g., a specific table or SQL query).
  - Don't try to tune tables owned by the sys or system schemas. Also, partitioned tables can't be analyzed correctly.
  - Oracle Expert is a learning tool inasmuch as it goes beyond just giving recommendations it thoroughly explains why it is making them.

Oracle Expert has seven different classes of input data:

1. **Database:** users, tablespaces, public synonyms.
2. **Instance:** data gathered from the X\$ tables and V\$ views.
3. **Schema:** tables, indexes, sequences, views, etc.
4. **Environment:** physical server attributes (RAM, CPU, disks).
5. **Workload:** what normal transactions are performed on the database. Can be gathered by Oracle Trace.
6. **Rules:** guidelines to tell Oracle how to treat collected data. Change rules if you don't like a given recommendation.



7. **control parameters:** specifies whether the database is an OLTP, DSS, or hybrid, so Oracle Expert can set goals and use specific features accordingly.

Oracle Expert generates two different types of output:

1. **Reports:** includes a) Analysis, b) Session Data and c) Recommendation Summary.
  2. **Implementation Files:** includes proposed changes to the init<sid>.ora file and SQL scripts to put into effect recommended changes (e.g. index creation).
- If you change any rules for a tuning session, you should rerun the analysis again.
  - You should sample the database frequently throughout the entire spectrum of normal database usage to give Oracle Expert the best data to work with.

Special thanks to  
[Rick Sands](#)  
and [Joe Seeley](#)  
for contributing this  
Cramsession.