Oracle Database Performance Tuning (Note Sheets)

Based on Oracle 9i Ed. 1.1

Editor: Ahmed Baraka

Document Purpose

This document is edited to be a quick hand book reference on Oracle 9i Performance Tuning. It is simply a summary from the resources below with concentration on the practical code.

Version History

Version	Individual Name	Date	Updates
1.0	Ahmed Baraka	Sept, 2003	Initial document.
1.1	Ahmed Baraka	July,2007	Minor Mdifications on format and information.

Resources

Resource Name	Description
Oracle 9i Performance Tuning Study Guide, by Joseph C. Johnson, Sybex, 2002	A book published by Sybex
Oracle 9i Database Performance Tuning Guide and Reference, Product No. A96533-02, October 2002, Oracle	Online Oracle 9i documentation library
Expert Oracle9i Database Administration, by Sam R. Alapati, Apress, 2003	Book published by Apress ISBN:1590590228
Internet Articles	

Contents

Introduction to Performance Tuning

Tuning Development Systems

Oracle's Top-Down Tuning Methodology

First Tuning the Data Design

Second Tuning the Application Design

Third Tuning Memory Allocation

Fourth Tuning I/O and Physical Structure

Fifth Tuning Resource Contention

Sixth Tuning the Underlying Platform(s)

Tuning Production Systems

Oracle's Production Performance Tuning Principles

First Define the problem clearly and then formulate a tuning goal.

Second Examine the host system and gather Oracle statistics.

Third Compare the identified problem to the common performance problems identified by Oracle in the Oracle9i Database Performance documentation. (see next section)

Fourth Use the statistics gathered in the second step to get a conceptual picture of what might be happening on the system.

Fifth Identify the changes to be made and then implement those changes.

Sixth Determine whether the objectives identified in step one have been met. If they have, stop tuning. If not, repeat steps five and six until the tuning goal is met.

Performance Tuning Documentation

Typical documentation (a.k.a. "deliverables") for a performance tuning project will include:

1. System or technical architecture document
2. Network architecture
3. Database architecture
4. Capacity plan
5. Change management plan
6. Test plans and procedures
7. Maintenance plan

Plan for your deliverables early in the project.

Top Ten Mistakes Found in Oracle Systems

1. Bad Connection Management

The application connects and disconnects for each database interaction. This problem is common with stateless middleware in application servers. It has over two orders of magnitude impact on performance, and it is totally unscalable.

2. Bad Use of Cursors and the Shared Pool

Not using cursors results in repeated parses. If bind variables are not used, then there is hard parsing of all SQL statements. This has an order of magnitude impact in performance, and it is totally unscalable. Use cursors with bind variables that open the cursor and execute it many times. Be suspicious of applications generating dynamic SQL.

3. Getting Database I/O Wrong

Many sites lay out their databases poorly over the available disks. Other sites specify the number of disks incorrectly, because they configure disks by disk space and not I/O bandwidth.

4. Redo Log Setup Problems

Many sites run with too few redo logs that are too small. Small redo logs cause system checkpoints to continuously put a high load on the buffer cache and I/O system. If there are too few redo logs, then the archive cannot keep up, and the database will wait for the archive process to catch up.

5. Serialization of data blocks in the buffer cache due to lack of free lists, free list groups, transaction slots (`INITRANS`), or shortage of rollback segments.

This is particularly common on INSERT-heavy applications, in applications that have raised the block size to 8K or 16K, or in applications with large numbers of active users and few rollback segments.

6. Long Full Table Scans

Long full table scans for high-volume or interactive online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Long table scans, by nature, are I/O intensive and unscalable.

7. In Disk Sorting

In disk sorts for online operations could indicate poor transaction design, missing indexes, or poor SQL optimization. Disk sorts, by nature, are I/O-intensive and unscalable.

8. High Amounts of Recursive (`SYS`) SQL

Large amounts of recursive SQL executed by `SYS` could indicate space management activities, such as extent allocations, taking place. This is unscalable and impacts user response time. Recursive SQL executed under another user ID is probably SQL and PL/SQL, and this is not a problem.

9. Schema Errors and Optimizer Problems

In many cases, an application uses too many resources because the schema owning the tables has not been successfully migrated from the development environment or from an older implementation. Examples of this are missing indexes or incorrect statistics. When migrating applications of known performance, export the schema statistics to maintain plan stability using the `DBMS_STATS` package. Likewise, optimizer parameters set in the initialization parameter file can override proven optimal execution plans. For these reasons, schemas, schema statistics, and optimizer settings should be managed together as a group to ensure consistency of performance.

10.Use of Nonstandard Initialization Parameters

These might have been implemented based on poor advice or incorrect assumptions. In particular, parameters associated with `SPIN_COUNT` on latches and undocumented optimizer features can cause a great deal of problems that can require considerable investigation.

Sources of Tuning Information

Alert Log File

- The alert log file consists of a chronological log of messages and errors.
- The later log file destination is defined by the parameter BACKGROUND_DUMP_DEST
- Check the alert log file regularly to:
  - Detect internal errors (ORA-600) and block corruption errors.
  - Monitor database operations.
  - View the nondefault initialization parameters.
- Remove or trim the file regularly after checking.

User Trace Files

- A user trace file contains statistics for traced SQL statements in that session.
- It location is specified by USER_DUMP_DEST
- User trace files can also be created by:
  - Backup control file to trace
  - Database SET EVENTs

Activating User Tracing

- **Instance-Level Tracing**
  By setting the parameter `SQL_TRACE=TRUE`, all processes against the instance will create their own trace files.
- **Session-Level Self Tracing**
  ```
  ALTER SESSION SET SQL_TRACE=TRUE;
  DBMS_SESSION.SET_SQL_TRACE(TRUE);
  ```
- **Session-Level DBA Tracing**
  ```
  SYS.DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(9,2,TRUE);
  ```

Managing Trace Files

The parameter `MAX_DUMP_FILE_SIZE` allows you to limit the size to which a user trace file can grow. This value can be expressed in either operating system blocks or bytes.

```
MAX_DUMP_FILE_SIZE=10M
```

Performance Tuning Views

Displaying Statistics

V$STATNAME

All kinds of system wide statistics are cataloged in this view.

STATISTIC# : statistic number

NAME : statistic name

CLASS : class number

1 - User instance activity
2 - Redo redo log buffer activity
4 - Enqueue locking
8 - Cache database buffer cache
16 – OS
32 - Real Application Clusters
64 – SQL like table access
128 – Debug

V$SYSSTAT

The server displays all calculated system statistics in the v$sysstat view. You can query this view to find cumulative totals since the instance started. Among its columns: STATISTIC# , NAME , CLASS , VALUE

V$SESSTAT

This view lists user session statistics. To find the name of the statistic associated with each statistic number, link it to the V$STATNAME view.

V$MYSTAT

This view displays the statistics of the current session.

SGA Global Statistics

V$SGASTAT

Query this view to find cumulative totals of detailed SGA usage since the instance started.

POOL : designates the pool in which the memory in NAME resides.

NAME : SGA component name.

BYTES :memory size in bytes.

Wait Statistics

V$EVENT_NAME

This view displays all possible wait events.

V$SYSTEM_EVENT

Cumulative on waits for an event are stored in this view; i.e total waits for an event for *all* sessions together since instance startup.

EVENT : the name of the wait event

TOTAL_WAITS : the total number of waits

TOTAL_TIMEOUTS : the total number of timeouts

TIME_WAITED : total amount of time waited in hundredths of a second

AVERAGE_WAIT : the average amount of time waited

V$SESSION_EVENT

This view shows, by session, the total waits for a particular event since instance startup.

MAX_WAIT : the maximum time waited

V$SESSION_WAIT

This view lists the resources or events for which active sessions are waiting.

P*n*TEXT	Description of nth additional parameter
P*n*	value of nth additional parameter
P*n*RAW	value of nth additional parameter in hexadecimal

`WAIT_TIME` a nonzero value is the session's last wait time. A zero value means the session is currently waiting (see STATE below).

`SECONDS_IN_WAIT` if WAIT_TIME = 0, then SECONDS_IN_WAIT is the seconds spent in the current wait condition. If WAIT_TIME > 0, then SECONDS_IN_WAIT is the seconds since the start of the last wait, and SECONDS_IN_WAIT - WAIT_TIME/100 is the active seconds since the last wait ended.

`STATE`: wait state takes one of the following (numbers are value of WAIT_TIME):

0 WAITING (the session is currently waiting)

-2 `WAITED UNKNOWN TIME` duration of last wait is unknown)

-1 `WAITED SHORT TIME` (last wait <1/100th of a second)

>0 WAITED KNOWN TIME (WAIT_TIME=duration of last wait)

Note: For the list of wait events, refer to "Oracle9i Database Reference, Release 2 (9.2), Part No. A96536-02, Appendix A Oracle Wait Events".

Note: Set the TIMED_STATISTICS parameter to TRUE to retrieve values in the WAIT_TIME column.

Segment Blocks Contention Statistics

The V$WAITSTAT view contains statistics about contention for segment blocks:

`CLASS` Short description of the type of segment that the block belongs to

`COUNT` Number of times a wait to access a block occurred

`TIME` Total time spent waiting for access to blocks

Level of Statistics Collection

The level of statistic collection on the database is determined by the value of the STATISTICS_LEVEL parameter:

- **Basic**: No advisory or other statistical data is collected. You can manually set other statistic collection parameters such as TIMED_STATISTICS and DB_CACHE_ADVICE.
- **Typical**: This is the default value. Data is collected for segment level statistics, timed statistics and all advisories. The value of other statistic collection parameters is overridden.
- **All**: Collection is made of all the Typical level data, the timed operating system statistics and the row source execution statistics. The value of other statistic collection parameters is overridden.

Note: Query V$STATISTICS_LEVEL to determine which parameters are affected by the STATISTICAL_LEVEL parameter. Common ones are: TIMED_STATISTICS, TIMED_OS_STATISTICS, DB_CACHE_ADVICE

Oracle Supplied Tuning Utilities

UTLBSTAT.SQL and UTLESTAT.SQL

Running the UTLBSTAT.SQL Script

The UTLBSTAT.SQL script is executed using the following command in SQL*Plus:

Unix: `@$ORACLE_HOME/rdbms/admin/utlbstat.sql`

Windows: `@%ORACLE_HOME%\rdbms\admin\utlbstat.sql`

Running the UTLESTAT.SQL Script

A meaningful amount of time should be allowed to pass between the time that UTLBSTAT.SQL is run and the time UTLESTAT.SQL is run.

The script calculates metrics between the statistics in the stats$begin tables, which were created by UTLBSTAT.SQL at the beginning of the tuning period, and the stats$end tables created by UTLESTAT.SQL at the end of the tuning period.

Then it generates a text file, REPORT.TXT, which contains the results of these calculations.

Using STATSPACK

Interactive STATSPACK Installation

After logging on as a SYSDBA, run the %ORACLE_HOME%\rdbms\admin\SPCREATE.SQL script. Enter appropriate information when prompted for the PERFSTAT user's password, default tablespace, and temporary tablespace.

Taking a Statspack Snapshot

After logging on as SYS or PERFSTAT, execute the procedure STATSPACK.SNAP. Alternatively you can run its overloaded function STATSPACK.SNAP which returns the snap_id of the snapshot just taken.

Note: For better performance analysis, set the initialization parameter TIMED_STATISTICS to TRUE.

Automating Statistics Gathering

Using DBMS_JOB to Collect Statistics

```
DBMS_JOB.SUBMIT(:jobno, 'STATSPACK.SNAP;', trunc(sysdate+1/24,'HH'),
'trunc(SYSDATE+1/24,''HH'')', TRUE, :instno);
```

This is included in SPAUOT.SQL script

Use the DBMS_JOB.INTERVAL procedure to change the interval of statistics collection.

```
EXECUTE DBMS_JOB.INTERVAL( job_number,'SYSDATE+(1/48)');
```

Note: In order to use DBMS_JOB to schedule snapshots, you must set the JOB_QUEUE_PROCESSES initialization parameter to greater than 0 in the initialization file.

Using Operating System Utility

- Use cron on UNIX, to schedule snapshots.
- On Windows platforms, you can use the at utility on Windows NT or System Tools > Scheduled Tasks on Windows 2000.

Running a Statspack Performance Report

Running the Statspack Report

This is a general instance health report that covers all aspects of instance performance.

```
connect perfstat/pswd
@%ORACLE_HOME%\rdbms\admin\spreport
```

Note: It is not correct to specify begin and end snapshots where the begin snapshot and end snapshot were taken from different instance startups.

Running the SQL Report

This report displays statistics and the complete SQL text.

```
@%ORACLE_HOME%\rdbms\admin\sprepsql
```

Note: Hash value for the SQL statement can be obtained from the Statspack spreport report.

Gathering Optimizer Statistics on the PERFSTAT Schema

For best performance when running the performance reports, collect optimizer statistics for tables and indexes owned by PERFSTAT

```
DBMS_STATS.GATHER_SCHEMA_STATS(OWNNAME=>'PERFSTAT',CASCADE=>TRUE)
```

Configuring the Amount of Data Captured in Statspack

Snapshot Levels

Levels >= 0 General Performance Statistics

Levels >= 5 Additional Data: SQL Statements. This level includes all statistics gathered in the lower levels, as well as performance data on SQL statements with high resource usage.

Levels >= 6 Additional Data: SQL Plans and SQL Plan Usage. A level 6 snapshot gathers valuable information for determining whether the execution plan used for a SQL statement has changed.

Levels >= 7 Additional data: Segment Level Statistics. A level 7 snapshot gathers information which determines what segments are more heavily accessed and contended.

Levels >= 10 Additional Statistics: Parent and Child Latches This level includes all statistics gathered in the lower levels, as well as parent and child latch information.

Note: The default level set at installation is level 5.

SQL Thresholds

SQL statements gathered by Statspack are those that exceed one of six predefined threshold parameters:

- `i_executions_th` : number of executions of the SQL statement. [100]
- `i_disk_reads_th` : number of disk reads performed by the SQL statement. [1,000]
- `i_parse_calls_th` : number of parse calls performed by the SQL statement. [1,000]
- `i_buffer_gets_th` : number of buffer gets performed by the SQL statement. [10,000].
- `i_sharable_mem_th` : size of sharable memory used by the SQL statement. [1 Mb]
- `i_version_count_th` : version count for the SQL statement. [20]

Segment Statistic Threshold

- `i_seg_phy_reads_th`: number of physical reads on a segment. [1000]
- `i_seg_log_reads_th`: number of logical reads on a segment. [10000]
- `i_seg_buff_busy_th`: number of buffer busy waits for a segment [100]
- `i_seg_rowlock_w_th`: number of row lock waits for a segment [100]
- `i_seg_itl_waits_th`: number of ITL (interested transaction list) waits for a segment [100]
- `i_seg_cr_bks_sd_th`: number of consistent reads blocks served by the instance for the segment (RAC) [1000]
- `i_seg_cu_bks_sd_th`: number of current blocks served by the instance for the segment (RAC) [1000]

Note: Snapshot level and threshold information used by the package is stored in the `STATS$STATSPACK_PARAMETER` table.

Changing the Values for Snapshot Levels and SQL Thresholds

Temporarily Using New Values

```
STATSPACK.SNAP(i_snap_level=>6);
```

Saving New Defaults

```
STATSPACK.SNAP(i_snap_level=>10, i_modify_parameter=>'true');
```

or

```
STATSPACK.MODIFY_STATSPACK_PARAMETER (i_snap_level=>10, i_buffer_gets_th=>10000,
i_disk_reads_th=>1000);
```

Specifying a Session ID

If you want to gather session statistics and wait events for a particular session, specify the session ID in the call to Statspack.

```
STATSPACK.SNAP(i_session_id=>3);
```

Managing and Sharing Statspack Performance Data

Sharing Data Through Export
Use export and import utilities.

Removing Unnecessary Data

Purge unnecessary data from the PERFSTAT schema using the SPPURGE.SQL script.

Truncating All Statspack Data

To truncate all performance data indiscriminately, use SPTRUNC.SQL. This script truncates all statistics data gathered.

Removing Statspack

To deinstall Statspack, connect as a user with SYSDBA privilege and run the script SPDROP from SQL*Plus

Graphical Performance Tuning Tools

Oracle Enterprise Manager Console

- **Instance Manager** Starting, stopping, and managing Oracle instances
- **Schema Manager** Creating and managing Oracle database objects.
- **Security Manager** Creating and managing Oracle users, privileges, and roles
- **Storage Manager** Creating and managing tablespaces and datafiles
- **Replication Manager** GUI environment for managing Oracle snapshots and replication features
- **Workspace Manager** Virtual workspace that allows changes to be made to the data in the database.

Oracle Diagnostics Pack

- **Capacity Planner** Estimates future system requirements based on current workloads
- **Performance Manager** Presents real-time performance information in a graphical format
- **TopSessions Monitors** user session activities including resource consumption and locking
- **Trace Data Viewer** Graphical tool for viewing trace file output
- **Lock Monitor** Monitors locking information for the connected user sessions.
- **Top SQL Monitors** user session activities and identifies the SQL statements that are being executed and the resources they are consuming.
- **Performance Overview** Single-screen summary of the overall performance of the database.

Oracle Tuning Pack

- **Oracle Expert** Wizard-based database tuning tool based on the Oracle Tuning Methodology.
- **SQL Analyze** Analyzes and rewrites SQL statements to improve their performance.
- **Index Tuning Wizard** Helps identify which indexes are being utilized by application users, and the effectiveness of those indexes.
- **Outline Management** Used to manage Stored Outlines.
- **Reorg Wizard** Assists with table and index reorganization tasks like moving segments to new tablespaces and rebuilding segments with new sizing specifications.
- **Tablespace Map** Graphically shows you were each segment's individual extents are stored within a Tablespace's datafiles.

SQL Application Tuning and Design

Writing Efficient SQL

Here are some standard guidelines to ensure efficient SQL code.

Efficient Where Clauses

Using SQL Functions

- If you use SQL functions (for example, the SUBSTR and INSTR, TO_DATE, and TO_NUMBER functions) in the where clause, make sure you use a function-based index.

Using the Right Joins

- Try to use equijoins wherever possible.
- Performing filtering operations early reduces the number of rows to be joined in later steps.
- Join in the order that will produce the least amount of rows as output to the parent step.

Using the Case Statement

- When you need to calculate multiple aggregates from the same table, avoid writing a separate query for each aggregate. It is more efficient to use the case statement in this case, as it will enable you to compute multiple aggregates from the table with just a single read of the table.

Efficient Subquery Execution

- Subqueries perform better when you use IN rather than EXISTS. Oracle recommends using the IN clause if the subquery has the selective where clause. If the parent query contains the selective where clause, use the EXISTS clause rather than the IN clause.

Using Where Instead of Having

- Wherever possible, use the where clause instead of the having clause. The where clause will restrict the number of rows retrieved at the very outset.

Finding Inefficient SQL

The V$SQLAREA view holds all the SQL statements executed since instance start-up and gather statistical information about them from the shared pool area. Following are columns that could not be self descriptive:

SQL_TEXT	is the text of the SQL statement (first 1,000 characters)
ELAPSED TIME	is the elapsed time for parsing and execution.
PARSE_CALLS	is the combined soft and hard parse calls for the statement.
LOADS	is the number of times the statement was reloaded into the shared pool.
SHARABLE_MEMORY	is the total shared memory used by the cursor.
PERSISTENT_MEMORY	is the total persistent memory used by the cursor.
RUNTIME_MEMORY	is the total runtime memory used by the cursor.

Note: use the HASH_VALUE of the SQL statement to get its full text from the V$SQLTEXT view.

To capture statements with with both high disk reads and high logical reads:

```
SELECT hash_value, executions, buffer_gets, disk_reads,
 parse_calls
 FROM V$SQLAREA
 WHERE buffer_gets > 100000 OR disk_reads > 100000
 ORDER BY buffer_gets + 100*disk_reads DESC;
```

To find out the number of rows processed for each statement.

```
SELECT hash_value, rows_processed,
 buffer_gets, disk_reads, parse_calls
 FROM V$SQLAREA
 WHERE buffer_gets > 100000
 OR disk_reads > 100000
 ORDER BY buffer_gets + 100*disk_reads DESC;
```

Here's a query that sorts the top five queries that are taking the most CPU time and the most elapsed time to complete:

```
SELECT hash_value, executions,  round (elapsed_time/1000000, 2) elapsed_seconds,
 round(cpu_time/1000000, 2) cpu_secs from
  (select * from v$sql order by elapsed_time desc) where rownum <6
```

The new Oracle9*i* views V$SQL_PLAN and V$SQL_PLAN_STATISTICS are highly useful for tracking the efficiency of execution plans.

Measuring SQL Performance

The TKPROF Utility

TKPROF is a utility that anylyzes session-trace files and provides readable formatted files about the statements in the trace file. Following is a list of TKPROF command line options

`EXPLAIN` =user/password

Generates an Explain Plan for each statement in the trace file by using the PLAN_TABLE in the schema specified by this command. This option must include a valid Oracle username and password.

`TABLE` =schema.tablename

Using the schema specified by the `EXPLAIN` option, the execution plan is stored in the specified table rather than default table, PLAN_TABLE.

`SYS` =yes|no

Indicates whether the formatted file should include recursive SQL statements (i.e., statements implicitly issued against the data dictionary in support of the primary SQL statements).

`RECORD` =filename

Specifies the filename into which the SQL statements in the trace file should be written. Allows you to isolate the SQL statements without any accompanying resource usage information. Recursive SQL will be omitted.

`PRINT` =integer

Limits the number of SQL statements included in the resulting formatted trace file to the specified value.

`INSERT` =filename

Creates an SQL script that, when executed, will create a table called `TKPROF_TABLE` and then populate this table with the raw statistics contained in this trace file.

`AGGREGATE` =yes|no

Determines whether the statistics from multiple issuers of the same SQL statement will be counted together or independently.

`WAITS` =yes|no

Calculates summary statistics for wait events that occurred within the trace.

`SORT`

Determines the order in which the statements in the formatted trace file will be sorted. It can be equaled to a set of zero or more of the following sort options:

Sort Options for the Parse Phase

`PRSCNT` Number of times the statement was parsed
`PRSCPU` Amount of CPU time spent parsing the statement
`PRSELA` Elapsed time spent parsing the statement
`PRSDSK` Number of disk reads that occurred while the statement was being parsed
`PRSMIS` Number of Library Cache misses while the statement was being parsed
`PRSCU` Number of data or index buffers read from the SGA while the statement was being parsed
`PRSQRY` Number of rollback buffers read from the SGA while the statement was being parsed.

Sort Options for the Execute Phase

`EXECNT` Number of times the statement was executed
`EXECPU` Amount of CPU time spent executing the statement
`EXEELA` Elapsed time spent executing the statement
`EXEDSK` Number of disk reads that occurred while the statement was being executed
`EXEQRY` Number of rollback buffers read from the SGA while the statement was executing
`EXECU` Number of data buffers read from the SGA while the statement was executing
`EXEROW` Number of rows that were processed while the statement was executing
`EXEMIS` Number if Library cache misses that occurred while the statement was executing

Sort Options for the Fetch Phase

`FCHCNT` Number of row fetches that occurred
`FCHCPU` Amount of CPU time spent fetching rows
`FCHELA` Elapsed time spent fetching rows
`FCHDSK` Number of disk reads that occurred while the rows were being fetched
`FCHQRY` Number of rollback blocks read from the SGA while the rows were being fetched
`FCHCU` Number of data blocks read from the SGA while the rows were being fetched
`FCHROW` Number of rows fetched by the statement

The Explain Plan Facility

Generating Explain Plans Using TKPROF

This method is time consuming when formatting a large trace file.

```
tkprof ora_1234.trc trace.txt sys=no explain=user/pswd@PROD
```

Generating Explain Plans Using EXPLAIN PLAN FOR

1. Create a PLAN_TABLE table using the utlxplan.sql script (one time step)
2. Issue the command
   EXPLAIN PLAN FOR SELECT ...
3. Use utlxpls.sql or utlxplp.sql to query content of the PLAN_TABLE.
   The first script, utlxpls.sql excludes any Parallel Query information in the Explain Plan. The second script, utlxplp.sql, includes Parallel Query information in the Explain Plan.

The AUTOTRACE Utility

Preparing to Use AUTOTRACE

1. Create a PLAN_TABLE table using the utlxplan.sql script for any schema wishing to use the utility.
2. Create a database role called PLUSTRACE by running an Oracle-supplied script called plustrce.sql while logged on as the user SYS. This script is located in /sqlplus/admin.
3. Grant the new PLUSTRACE role to every user who will be using the AUTOTRACE utility.

Using AUTOTRACE

AUTOTRACE can be activated in a SQL*Plus session by issuing the SET AUTOTRACE ON command.

```
SQL> set autotrace on
SQL> SELECT dist.distributor ...
```

Options for AUTOTRACE Utility

ON Displays query results, execution plan, and statistics

ON STATISTICS Displays the query results and statistics, but no execution plan

ON EXPLAIN Displays query results and execution plan, but no statistics

TRACEONLY Displays the execution plan and the statistics, but no query results (The query is still executed even though the query results are not shown).

TRACEONLY STATISTICS Displays the statistics only, no execution plan or query results (The query is still executed even though the query results are not shown).

OFF Turns off the AUTOTRACE utility.

SQL Tuning Information in STATSPACK Output

Tuning information is found in four sections of the STATSPACK output file:

- SQL ordered by Gets this is useful to tune share pool
- SQL ordered by Reads this is useful to tune disk I/O
- SQL ordered by Executions this is useful for pinning PL/SQL packages
- SQL Executions ordered by Parse Calls this is useful to identify statements that could be potentially re-written to improve SQL reuse.

Understanding the Oracle Optimizer

Rule-Based Optimization

The rule-based optimizer (RBO) uses a set of predefined “rules of thumb” to decide how to best execute a query.

Cost-Based Optimization

The CBO relies heavily on the presence of statistics about the tables and indexes in the database. These statistics are stored in the data dictionary, which includes the following:

- The size of each table or index
- The number of rows each table or index contains
- The number of database blocks used by each table or index
- The length of each table row
- The cardinality of the column data in indexed columns

Using ANALYZE command

ANALYZE command is used to analyze an individual table and index in order to gather its statistics. Oracle, however, recommends using DBMS_STATS instead.

```
ANALYZE INDEX last_name_idx COMPUTE STATISTICS
```

This command gathers the statistics for the named index.

```
ANALYZE TABLE employee COMPUTE STATISTICS
```
This command gathers the statistics for the EMPLOYEE table by examining every row in the table. It will also gather statistics on all the indexes associated with the EMPLOYEE table.

```
ANALYZE INDEX distributor_id_pk ESTIMATE STATISTICS
```
Using ESTIMATE in place of COMPUTE causes Oracle to estimate table and index statistics using a sample of the table or index data. The default sample size is 1,064 rows.

```
ANALYZE TABLE distributors ESTIMATE STATISTICS SAMPLE 500 ROWS
```
ANALYZE INDEX distributor_id_pk ESTIMATE STATISTICS SAMPLE 35 PERCENT
Those commands specify their sample sizes in terms of rows or a percentage of the overall table.

```
ANALYZE TABLE distributors ESTIMATE STATISTICS FOR COLUMNS district_id SIZE 200
```
This command would limit the analysis of the table to the just the DISTRICT_ID column. The SIZE parameter specifies how many slices to divide the column's values into before examining the cardinality of data within each slice. The more slices that are taken, the more accurate a picture the CBO will have regarding the distribution of the data within the column. The default value for SIZE is 75. The acceptable range of values is 1 through 254.

```
ANALYZE TABLE employee DELETE STATISTICS
```
DELETE option can be used to remove statistics if needed.

Options of the FOR clause of the ANALYZE command

`FOR TABLE` Gathers only table statistics, without column statistics
`FOR COLUMNS` Gathers column statistics for only the specified columns
`FOR ALL COLUMNS` Gathers columns statistics only, on all columns
`FOR ALL INDEXES` Gathers the statistics for all indexes on this table, but no table statistics
`FOR ALL INDEXED COLUMNS` Gathers columns statistics on only columns that are indexed
`FOR ALL LOCAL INDEXES` Gathers column statistics on all local indexes of a partitioned table

Histograms

Histogram statistics can be used to improve the performance of SQL statements whose underlying tables have nonuniformly distributed data.

```
ANALYZE TABLE finaid COMPUTE STATISTICS FOR COLUMNS award SIZE 100
```

Using DBMS_UTILITY

DBMS_UTILITY is used to to analyze every table and index in a user's schema without having to individually specify each segment by name.

```
DBMS_UTILITY.ANALYZE_SCHEMA('SCOTT','ESTIMATE',0,40,'FOR ALL INDEXED COLUMNS SIZE 40')
```

SCOTT The schema to be analyzed.
ESTIMATE The type of analysis to be done. If COMPUTE is used, the next two arguments have no effect.
0 The number of rows to use in the estimate. If the estimate will be specified as a percentage of rows, enter a zero here.
40 The percentage of the total rows to be used in the estimate. If the previous argument is nonzero, then this argument has no effect.
FOR ALL INDEXED COLUMNS SIZE 40 Additional options as shown above.

```
DBMS_UTILITY.ANALYZE_DATABASE('ESTIMATE',0,40,'FOR ALL INDEXED COLUMNS SIZE 40 ')
```
This is to analyze the entire database.

Note: Using the ANALYZE_DATABASE procedure will cause statistics to be gathered for the SYS schema. You should use the DBMS_UTILITY.ANALYZE_SCHEMA('SYS', 'DELETE ') command to remove them.

Using DBMS_STATS

About DBMS_STATS

- The DBMS_STATS package offers several additional analyzing options that are not available in the DBMS_UTILITY package.
- Oracle Corporation strongly recommends that you use the DBMS_STATS package rather than ANALYZE to collect optimizer statistics.

Creating Statistics Table

- Use DBMS_STATS.CREATE_STAT_TABLE to create a table for storing statistics in a specific schema. It will be used for stattab parameter in the other procedures.
  DBMS_STATS.CREATE_STAT_TABLE ('hr', 'mystattab')

Gathering CPU and I/O statistics for the system

```
DBMS_STATS.GATHER_SYSTEM_STATS (
    gathering_mode VARCHAR2 DEFAULT 'NOWORKLOAD',
    interval INTEGER DEFAULT NULL,
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL)
```

`gathering_mode` mode values are:
- `'NOWORKLOAD'`: no workload is required to capture system activity.
- `'INTERVAL'`: captures system activity during a specified interval.
- `'START'|'STOP'`: start/stop capturing system activity. Interval value is ignored.

`interval`: time, in minutes, to gather statistics. This parameter applies only when `gathering_mode='INTERVAL'`.

`stattab` table where to save the current statistics.

`statid` to associate with these statistics within `stattab`.

`statown` schema containing `stattab`.

If the database has a different kinds of activities in specific times (like OLTP at day time and generating reports at night), you can gather system statistics during these periods into user statistics tables. Then, schedule the process of importing the statistics into the dictionary in its appropriate period.

Gathering Index Statistics

```
DBMS_STATS.GATHER_INDEX_STATS (
  ownname  VARCHAR2,
  indname  VARCHAR2,
  partname VARCHAR2 DEFAULT NULL,
  estimate_percent  NUMBER DEFAULT NULL,
  stattab  VARCHAR2 DEFAULT NULL,
  statid   VARCHAR2 DEFAULT NULL,
  statown  VARCHAR2 DEFAULT NULL,
  degree   NUMBER   DEFAULT NULL,
  granularity VARCHAR2 DEFAULT 'DEFAULT',
  no_invalidate BOOLEAN DEFAULT FALSE);
```

`partname` name of partition.

`estimate_percent` the valid range is [0.000001,100] (NULL means compute) or use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` (*recommended*)

`degree` degree of parallelism (NULL means use of table default value that was specified by the `DEGREE` clause in the `CREATE/ALTER INDEX` statement). Use the constant `DBMS_STATS.DEFAULT_DEGREE` for the default value based on the initialization parameters.

`granularity` The granularity of statistics to collect (only pertinent if the index is partitioned):
- `'DEFAULT'` - gathers global and partition-level statistics
- `'SUBPARTITION'` - gathers subpartition-level statistics
- `'PARTITION'` - gathers partition-level statistics
- `'GLOBAL'` - gathers global statistics
- `'ALL'` - gathers all (subpartition, partition, and global)

`no_invalidate` dependent cursors are not invalidated if this parameter is set to `TRUE`.

You should analyze the table after creating a **function-based index**, to allow Oracle to collect column statistics equivalent information for the expression.

Gathering Table, column, and index statistics

```
DBMS_STATS.GATHER_TABLE_STATS (
  ownname VARCHAR2,
  tabname VARCHAR2,
  partname VARCHAR2 DEFAULT NULL,
  estimate_percent NUMBER DEFAULT NULL,
  block_sample BOOLEAN DEFAULT FALSE,
  method_opt DEFAULT 'FOR ALL COLUMNS SIZE 1',
  degree NUMBER DEFAULT NULL,
  granularity VARCHAR2 DEFAULT 'DEFAULT',
  cascade BOOLEAN DEFAULT FALSE,
  stattab VARCHAR2 DEFAULT NULL,
  statid VARCHAR2 DEFAULT NULL,
  statown VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT FALSE);
```

`block_sample` whether or not to use random block sampling instead of random row sampling.

`method_opt` accepts:

`FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`

`FOR COLUMNS [size clause]`

`column [size_clause] [,column [size_clause]...]`

where `size_clause` is defined as:

`size_clause := SIZE {integer | REPEAT | AUTO |SKEWONLY}`

- `integer`: number of histogram buckets. [1,254].
- `REPEAT`: collects histograms only on the columns that already have histograms.
- `AUTO`: Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.

`SKEWONLY`: Oracle determines the columns to collect histograms based on the data distribution of the columns.

Gathering Statistics for all objects in a schema

```
DBMS_STATS.GATHER_SCHEMA_STATS (
  ownname VARCHAR2,
  estimate_percent NUMBER DEFAULT NULL,
  block_sample BOOLEAN DEFAULT FALSE,
  method_opt DEFAULT 'FOR ALL COLUMNS SIZE 1',
  degree NUMBER DEFAULT NULL,
  granularity VARCHAR2 DEFAULT 'DEFAULT',
  cascade BOOLEAN DEFAULT FALSE,
  stattab VARCHAR2 DEFAULT NULL,
  statid VARCHAR2 DEFAULT NULL,
  options VARCHAR2 DEFAULT 'GATHER',
  [objlist OUT ObjectTab,]
  objlist OUT ObjectTab,
  statown VARCHAR2 DEFAULT NULL,
  no_invalidate BOOLEAN DEFAULT FALSE,
  gather_temp BOOLEAN DEFAULT FALSE);
```

`options` Further specification of which objects to gather statistics for:

- `GATHER`: gathers statistics on all objects in the schema.
- `GATHER AUTO`: gathers all necessary statistics automatically.
- `GATHER STALE`: gathers statistics on stale objects.
- `GATHER EMPTY`: on objects which currently have no statistics.
- `LIST AUTO`: returns a list of objects to be processed with `GATHER AUTO`.
- `LIST STALE`: returns list of stale objects
- `LIST EMPTY`: returns list of objects which currently have no statistics.

`objlist` list of objects found to be stale or empty.

`gather_temp` gathers statistics on global temporary tables.

Before gathering new statistics for a particular schema, use the `DBMS_STATS.EXPORT_SCHEMA_STATS` procedure to backup existing statistics.

Gathering Statistics for all objects in a database

```
DBMS_STATS.GATHER_DATABASE_STATS (
  estimate_percent NUMBER DEFAULT NULL,
  block_sample BOOLEAN DEFAULT FALSE,
  method_opt DEFAULT 'FOR ALL COLUMNS SIZE 1',
  degree NUMBER DEFAULT NULL,
  granularity VARCHAR2 DEFAULT 'DEFAULT',
  cascade BOOLEAN DEFAULT FALSE,
  stattab VARCHAR2 DEFAULT NULL,
```

```
statid VARCHAR2 DEFAULT NULL,
options VARCHAR2 DEFAULT 'GATHER',
[objlist OUT ObjectTab,]
statown VARCHAR2 DEFAULT NULL,
gather_sys BOOLEAN DEFAULT FALSE,
no_invalidate BOOLEAN DEFAULT FALSE,
gather_temp BOOLEAN DEFAULT FALSE);
```

`gather_sys` gathers statistics on SYS objects.

Transferring Statistics

The `IMPORT_*` procedures

```
DBMS_STATS.IMPORT_TABLE_STATS(ownname, tabname, partname, stattab, statid, cascade,
statown, no_invalidate)

DBMS_STATS.IMPORT_COLUMN_STATS(ownname, tabname, colname, partname, stattab, statid,
statown, no_invalidate)

DBMS_STATS.IMPORT_INDEX_STATS(ownname, indname, partname, stattab, statid, statown,
no_invalidate)

DBMS_STATS.IMPORT_SYSTEM_STATS(stattab, statid, statown)

DBMS_STATS.IMPORT_SCHEMA_STATS(ownname, stattab, statid, statown, no_invalidate)

DBMS_STATS.IMPORT_DATABASE_STATS(stattab, statid, statown, no_invalidate)
```

Default values of parameters:
`partname: NULL, statid: NULL, cascade: TRUE, statown: NULL, no_validate: FALSE`

The `EXPORT_*` procedures

```
EXPORT_COLUMN_STATS
EXPORT_INDEX_STATS
EXPORT_SYSTEM_STATS
EXPORT_TABLE_STATS
EXPORT_SCHEMA_STATS
EXPORT_DATABASE_STATS
```

For instance, to move statistics of a specific schema from a production database to a development database, you can follow the following steps:

1. In the production database, create the `STATS` table in the source schema (preferably in the `TOOLS` tablespace). This table will be used to hold the production schema statistics.
   `EXECUTE DBMS_STATS.CREATE_STAT_TABLE ('JOE', 'STATS','TOOLS')`
2. Capture the current statistics for the source schema and store them in the newly created STATS table.
   `DBMS_STATS.EXPORT_SCHEMA_STATS('JOE','STATS')`
3. Use the Oracle Export utility from the OS command line to export the contents of the `STATS` table from the production database.
4. Move the export dump file from the production server to the Development server.
5. In the Development database, create the `STATS` table in the destination schema in the `TOOLS` tablespace.
6. Use the Oracle Import utility from the operating system command line to import the `STATS` dump file created on the production server into the `STATS` table on the Development server.

Move the statistics in destination schema's `STATS` table into the Development database's data dictionary.
`DBMS_STATS.IMPORT_SCHEMA_STATS('JOE','STATS')`

Gathering Automated Statistics

1. You must bring either the tables of a specific schema or a complete database into the monitoring mode using the following procedures:
   `DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING (ownname, monitoring [TRUE])`
   `DBMS_STATS.ALTER_DATABASE_TAB_MONITORING`
   or
   Use the `ALTER TABLE...MONITORING` (or `NOMONITORING`) statement.
2. You can enable automated statistics gathering by setting up a recurring job that invokes `DBMS_STATS.GATHER_TABLE_STATS` with the `GATHER STALE` option at an appropriate interval for your application.

Note: Objects are considered stale when 10% of the total rows have been changed.

Viewing Statistics using DBMS_STATS

Use DBMS_STATS.GET_*_STATS procedures to view the statistics stored in the data dictionary or in a statistics table.

```
DBMS_STATS.GET_COLUMN_STATS (
ownname, tabname, colname, partname [NULL], stattab [NULL], statid [NULL], distcnt
OUT NUMBER, density OUT NUMBER, nullcnt OUT NUMBER, srec OUT StatRec,
avgclen OUT NUMBER, statown VARCHAR2 [NULL]);
```

distcnt number of distinct values.
density column density.
nullcnt number of NULLs.
srec structure holding internal representation of column minimum, maximum, and histogram values.
avgclen average length of the column (in bytes).
statown schema containing stattab (if different than ownname).

```
DBMS_STATS.GET_INDEX_STATS(
ownname, indname, partname, stattab [NULL],
statid [NULL], numrows OUT, numlblks OUT, numdist OUT, avglblk OUT, avgdblk OUT,
clstfct OUT, indlevel OUT, statown [NULL]);
```

numrows Number of rows in the index (partition).
numlblks Number of leaf blocks in the index (partition).
numdist Number of distinct keys in the index (partition).
avglblk Average integral number of leaf blocks in which each distinct key appears for this index (partition).
avgdblk Average integral number of data blocks in the table pointed to by a distinct key for this index (partition).
clstfct Clustering factor for the index (partition).
indlevel Height of the index (partition).

```
DBMS_STATS.GET_SYSTEM_STATS (
status OUT VARCHAR2, dstart OUT DATE,
dstop OUT DATE, pname VARCHAR2, pvalue OUT NUMBER,
stattab [NULL], statid [NULL], statown [NULL]);
```

status: COMPLETED, AUTOGATHERING, MANUALGATHERING, BADSTATS
dstart date when statistics gathering started.
dstop date when statistics gathering stopped.
If status = AUTOGATHERING, the future finish date is returned.
If status = BADSTATS, the must-finished-by date is returned.
pname The parameter name to get, which can have one of the following values:
- sreadtim: average time to read single block (random read), in milliseconds
- mreadtim: average time to read an mbrc block at once (sequential read), in milliseconds
- cpuspeed: average number of CPU cycles per second, in millions
- mbrc: average multiblock read count for sequential read, in blocks
- maxthr: maximum I/O system throughput, in bytes/sec
- slavethr: average slave I/O throughput, in bytes/sec

pvalue The parameter value to get.

```
DBMS_STATS.GET_TABLE_STATS(
ownname, tabname, partname [Null], stattab [NULL],
statid [NULL], numrows OUT NUMBER, numblks OUT NUMBER, avgrlen OUT NUMBER, statown
[NULL])
```

numrows Number of rows in the table (partition).
numblks Number of blocks the table (partition) occupies.
avgrlen Average row length.

Using Histograms

In general, create histograms on columns that are used frequently in WHERE clauses of queries and have a highly skewed data distribution.
Histograms are not useful when:

- The column data is uniformly distributed.
- The column is unique and is used only with equality predicates.

Creating Histograms

```
DBMS_STATS.GATHER_TABLE_STATS ('scott', 'emp', METHOD_OPT => 'FOR COLUMNS SIZE AUTO
sal');
```

Viewing Histograms

Query the appropriate data dictionary view:

DBA_HISTOGRAMS
DBA_PART_HISTOGRAMS
DBA_SUBPART_HISTOGRAMS
DBA_TAB_COL_STATISTICS

Note: DENSITY is calculated as 1 divided by NUM_DISTINCT.

Using the OEM Console to Gather Statistics

- The OEM Console can also be used to gather schema statistics. By selecting Tools -> Database Wizards -> Analyze from the OEM Console menu, the Analyze Wizard will be displayed.
- By default, the Analyze Wizard uses the DBMS_STATS package to perform its statistical analysis.

Viewing Table and Index Statistics using Data Dictionary Views

DBA_TABLES

Statistics for analyzed tables can be found in the BA_TABLES data dictionary view.

NUM_ROWS Number of rows in the table

BLOCKS Number of blocks below the High Water Mark

EMPTY_BLOCKS Number of blocks above the High Water Mark

AVG_SPACE Average available free space in the table

CHAIN_CNT Number of chained or migrated rows

AVG_ROW_LEN Average length, in bytes, taken up by row data and row overhead

AVG_SPACE_FREELIST_BLOCKS Average free space, in bytes, of the blocks on the table's Freelist

NUM_FREELIST_BLOCKS Number of blocks on the table's Freelist

SAMPLE_SIZE Size of the sample used to last analyze the table

LAST_ANALYZED Date the table was last analyzed

DBA_INDEXES

BLEVEL Number of levels between the index's root block and its leaf blocks

LEAF_BLOCKS Number of leaf blocks in the index

DISTINCT KEYS Number of distinct values in the indexed column—a measure of the column's cardinality

AVG_LEAF_BLOCKS_PER_KEY Average number of leaf blocks used for each distinct key value in the index

AVG_DATA_BLOCKS_PER_KEY Average number of data blocks for each distinct key value in the index

CLUSTERING_FACTOR A measure of how ordered the data is in the index's underlying table

NUM_ROWS Number of rows contained in the index

SAMPLE_SIZE Size of the sample used to last analyze the index

LAST_ANALYZED Date the index was last analyzed.

DBA_TAB_COL_STATISTICS

TABLE_NAME Name of the table

COLUMN_NAME Name of the table column

NUM_DISTINCT Number of distinct values in the column—a measure of cardinality

LOW_VALUE Lowest value in the column

HIGH_VALUE Highest value in the column

DENSITY A measure of how dense the column data is; a derived value defined as 1/NUM_DISTINCT

NUM_NULLS Number of values in the column that are NULL

NUM_BUCKETS Number of "slices" the data was divided into when it was last analyzed

SAMPLE_SIZE Size of the sample used to last analyze the table

LAST_ANALYZED Date the table was last analyzed

GLOBAL_STATS Indicates whether statistics were generated using partition data

USER_STATS Indicates whether the statistics were entered by the user

AVG_COL_LEN Average length, in bytes, of the column

Choosing an Optimizer Approach and Goal

Optimizer approaches are either CBO or RBO. CBO the highly recommended approach by Oracle.

Optimizer goal is either to get best ***throughput*** (least resources to process all rows) or ***response time*** (least resource to process first row). Which approach to use is affected by the following factors:

- **OPTIMIZER_MODE** initialization parameter**:** can take one of the following values:
  - CHOOSE The optimizer chooses between a cost-based approach and a rule-based approach, depending on whether statistics are available. This is the default.
  - ALL_ROWS The optimizer uses a cost-based approach for all SQL statements in the session and optimizes with a goal of best throughput.
  - FIRST_ROWS_n The optimizer uses a cost-based approach and optimizes with a goal of best response time to return the first n number of rows; n can equal 1, 10, 100, or 1000.
  - FIRST_ROWS The optimizer uses a mix of cost and heuristics to find a best plan for fast delivery of the first few rows. This value is retained for backward compatibility only, with the first_rows_n mode being the latest version of this mode.
  - RULE The optimizer chooses a rule-based approach for all SQL statements regardless of the presence of statistics.

  Note: This parameter is dynamic by ALTER SESSION statement.
- **SQL Hints:** In SQL Statement level, you can use any of the following hints: FIRST_ROWS(n), FIRST_ROWS, ALL_ROWS, CHOOSE, RULE.
  SELECT /*+ RULE */ ...
- **CBO Statistics in the Data Dictionary:** You can collect exact or estimated statistics about physical storage characteristics and data distribution in schema objects by using the DBMS_STATS package (recommended method) or the ANALYZE statement. (see *Gathering Optimizer Statistics* section)

Note The CBO's behavior varies slightly from one release of Oracle to another. Oracle9i includes a new init.ora parameter, OPTIMIZER_FEATURES_ENABLE, which can be used to specify the Oracle CBO version you wish to use. For example, setting OPTIMIZER_FEATURES_ENABLE=8.1.7 will make the Oracle9i instance use the Oracle8i CBO.

Plan Stability

Stored Outlines

- Oracle9i maintains predefined execution plans of SQL statements in the data dictionary in the form of stored outlines. Plan stability is most useful when you cannot risk any performance changes in an application.
- Plan stability is normally useful for statements that handle static data.
- Oracle uses hints to record stored plans.

Recommendations to Apply on OUTLN Schema

Alter the OUTLN schema. This schema stores all the stored outlines for the database.

1. For security purposes, change the OUTLN schema's password. The default password is 'OUTLN'.
2. Alter the OUTLN schema's default and temporary tablespaces to something other than the SYSTEM tablespace.

Move the two tables that the OUTLN schema uses to store outlines (OL$HINTS and OL$NODES) out of the SYSTEM tablespace and into OUTLN's new default tablespace. The Export and Import utilities, or the ALTER TABLE ... MOVE... command can be used for this operation.

Enabling Plan Stability

Setting of the following parameters must be consistent across execution environments for outlines to function properly:

- QUERY_REWRITE_ENABLED
- STAR_TRANSFORMATION_ENABLED
- OPTIMIZER_FEATURES_ENABLE

Creating Outlines

- Oracle creates stored outlines automatically when you set the parameter CREATE_STORED_OUTLINES to TRUE.

Note: You must ensure that schemas in which outlines are to be created have the CREATE ANY OUTLINE privilege.

Note: The CREATE_STORED_OUTLINES, USE_STORED_OUTLINES, and USE_PRIVATE_OUTLINES parameters are system or session specific. They are not initialization parameters. For more information on these parameters, see the Oracle SQL Reference.

You can create stored outlines for specific statements using the CREATE OUTLINE statement.

Using Category Names for Stored Outlines

- The CREATE_STORED_OUTLINES parameter lets you specify a category name:

  TRUE outlines will be produced in a category named DEFAULT

  'cat_name' Oracle assigns all subsequently created outlines to that category until you reset the category name.

  FALSE suspends outline generation.

```
SQL> ALTER SESSION SET CREATE_STORED_OUTLINES=employee_queries;
SQL> SELECT lastname FROM employee WHERE job_title ='MGR';
SQL> ALTER SESSION SET CREATE_STORED_OUTLINES=false;
```

- The CREATE OUTLINE statement allows for specification of a category.

```
CREATE OR REPLACE OUTLINE mgr_last_name
FOR CATEGORY employee_queries ON
SELECT lastname
FROM employee
WHERE job_title = 'MGR'
```

Using Stored Outlines

To use stored outlines when Oracle compiles a SQL statement, set the parameter USE_STORED_OUTLINES to TRUE or to a category name. Setting to FALSE will suspend outline use.

Managing Stored Outlines

Using Outline Manager

The Outline Manager GUI tool can be used to:

- Create stored outlines
- Modify existing stored outlines:
  - change the access method
  - modify the join order of the SQL statement behind the stored outline
  - change the category that the stored outline is assigned to
  - drop a stored outline.

Using OUTLN_PKG

- DROP_UNUSED Drops outlines in current schema that have not been used since they were created.
- DROP_BY_CAT Drops the specified category and all outlines contained in it.
- UPDATE_BY_CAT Creates the new category name specified and moves all the outlines in the old category specified into the new category.

```
OUTLN_PKG.DROP_UNUSED
OUTLN_PKG.DROP_UNUSED('EMPLOYEE_QUERIES ')
OUTLN_PKG.UPDATE_BY_CAT('DEFAULT','DS_Q')
```

Using ALTER and DROP OUTLINE

The ALTER OUTLINE and DROP OUTLINE commands can also be used to manage stored outlines. Options of ALTER OUTLINE are the following:

- REBUILD Generates a new execution plan for the specified stored outline using current statistics
- RENAME TO Changes the name of the stored outline to the new name specified
- CHANGE CATEGORY TO Changes the category to which the stored outline belongs

Note: If the stored outline that is dropped is the last outline in its category, the category will also be dropped. Oracle does not maintain categories that do not contain stored outlines.

Viewing Stored Outline Data

DBA_OUTLINES View

NAME the name of the stored outline.

OWNER the schema name of the stored outline's owner.

CATEGORY the category the stored outline is associated with.

USED indicates whether the outline has ever been used (values are USED/UNUSED/UNDEFINED)

TIMESTAMP date and time the stored outline was created

VERSION Oracle RDBMS version that created the stored outline

SQL_TEXT The actual text of the original SQL query associated with the stored outline.

DBA_OUTLINE_HINTS View

NODE The identifier indicating to which SQL statement the hint belongs

STAGE stage in computation process where the hint was applied

JOIN_POS the position of the table in the SQL query's original join order

HINT The actual text of the hint used in the original SQL query associated with the stored outline

Private Outlines

About Private Outlines

A Private Outline is an outline that resides within an individual user's session. As such, any changes made to the outline will affect only that session while all other sessions will continue to use the unaltered public stored outline.

Private Outlines Requirements

- Tables must be created in the user's schema to hold the private copy of the public stored outline. These tables can be built by executing the DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES procedure.
- The user must have the CREATE ANY OUTLINE privilege as well as SELECT privileges on the tables involved in the outline's query.

Creating a Private Outline

This example creates a private outline called PRIVATE_OL from a public stored outline called PUBLIC_OL:

1. Copy the public stored outline's definition into the private outline using the CREATE PRIVATE OUTLINE private_ol FROM public_ol command.
2. Alter the private outline using one of the techniques discussed in the previous section and then re-sync the outline using the CREATE PRIVATE OUTLINE private_ol FROM PRIVATE private_ol command.
3. Activate private outlines in the session using the ALTER SESSION SET USE_PRIVATE_OUTLINES=TRUE command.
4. Execute the query referenced in the outline and then use the EXPLAIN PLAN or TKPROF utilities to examine the effectiveness of the new private outline.
5. If you decide that you would like to make your private outline available to all other users, you can make it public by issuing the CREATE OR REPLACE OUTLINE public_ol FROM PRIVATE private_ol command.
6. Deactivate private outlines in the session using the ALTER SESSION USE_PRIVATE_OUTLINES=FALSE command.

How to Tell If an Outline Is Being Used

```
SELECT OUTLINE_CATEGORY, OUTLINE_SID
FROM V$SQL
WHERE SQL_TEXT LIKE 'SELECT COUNT(*) FROM emp%'
```

- OUTLINE_CATEGORY if an outline was applied, then this column contains the category to which the outline belongs. Otherwise, it is NULL.
- OUTLINE_SID tells you if this particular cursor is using a public outline (value is 0) or a private outline (session's SID of the corresponding session using it).

Materialized Views

About Materialized Views

A materialized view stores the physical results of the view in its own segment, separate and distinct from the underlying table on which the view is based.

Features:

- Materialized views are often used in data warehouses to increase the speed of queries on very large data tables.
- Queries that benefit from the use of materialized views often involve joins between tables or aggregations such as SUM.
- Materialized views can be used to replicate data, which was formerly achieved using the CREATE SNAPSHOT statement.
- Materialized view segment can be stored in its own tablespace and can be indexed and partitioned.

Materialized View Requirements

Initialization Parameters and Privileges

JOB_QUEUE_PROCESSES

The number of background job queue processes to start—must be set to a value of 1 or more when using materialized views.

QUERY_REWRITE_ENABLED

Allows optimizer to dynamically rewrite queries to take advantage of materialized views when set to TRUE.

QUERY_REWRITE_INTEGRITY

Determines the degree to which the data consistency is to be adhered to when accessing materialized views. Options are:

- ENFORCED: query rewrites only occur when Oracle can guarantee data currency (default)
- TRUSTED: query rewrites can occur when declared relationships exist, but without complete data currency
- STALE_TOLERATED: query rewrites can occur even when the view's data and the underlying table data are not current.

OPTIMIZER_MODE

Must be set to one of the cost-based optimizer modes.

- The QUERY_REWRITE_ENABLED and QUERY_REWRITE_INTEGRITY parameters can also be set at the session level using the ALTER SESSION command.
- The system privilege GLOBAL QUERY REWRITE and/or QUERY REWRITE must be granted to users who wish to alter their session in this manner.

There are also two SQL hints, REWRITE and NOREWRITE, which can also be used to turn on and off the dynamic rewriting of queries when using materialized views.

Using Materialized Views

1. Determine the statements for which you would like to create materialized views. It is likely that these statements will involve large fact tables that are joined to one another and include summary functions.

   Note: Oracle provides a *Summary Advisor* utility that can be used to identify SQL statements that may make good candidates for materialized views.

2. Decide **synchronization option** between the view and the underlying base tables. You will specify one of the flowing options during the creation of the view:
   - NEVER : no synch will occur
   - COMPLETE: During a refresh, the materialized view is truncated and then completely repopulated with data from the base tables in the query.
   - FAST: During a refresh, the materialized view is populated only with data that has changed in the base table since the last re-sync. This refresh is performed using the view's log data or by ROWID.
     Fast refresh requirements:
     - The master table or master materialized view has a materialized view log.
     - Each join column in the CREATE MATERIALIZED VIEW statement must have an index on it.
   - FORCE: Oracle attempts to refresh the view using the FAST refresh method first. If the FAST refresh method is not available, then the COMPLETE refresh is used. FORCE is the default sync mode.

3. Determine the **refresh frequency mode**. There are three possible refresh modes:

- ON COMMIT: The materialized view data will be refreshed with every committed transaction on the base tables.
- By Time: By using the START WITH and NEXT options during view creation, the materialized view data can be refreshed at specified times on specified dates. These refresh times and dates are submitted as jobs to the Oracle job queue.
- ON DEMAND: You can also manually refresh the contents of a materialized view.

Creating Materialized View Log

In order for a materialized view to refresh in the FAST mode, the materialized view logs should be created on the underlying table(s).

```
CREATE MATERIALIZED VIEW LOG ON customers WITH PRIMARY KEY, ROWID;
```

Creating Materialized Views

```
CREATE MATERIALIZED VIEW cust_sales_mv
PCTFREE 0
TABLESPACE demo
BUILD IMMEDIATE
REFRESH COMPLETE
START WITH SYSATE
  NEXT SYSDATE+3/24 -- every 3 hours
ENABLE QUERY REWRITE
AS
SELECT ...
```

The options that can be specified during the creation of the materialized view:

BUILD IMMEDIATE
causes Oracle to build the materialized view and populate it with data immediately after the command is executed.

BUILD DEFERRED
causes Oracle to build the structure of the materialized, but does not populate the materialized view with data until the first refresh occurs.

ON PREBUILT TABLE

causes Oracle to use a preexisting Oracle table that already contains the data that is in the view definition, rather than building a new structure to hold the same data.

Refreshing Materialized Views

Manual Refreshing

Refreshes can be generated manually using the DBMS_MVIEW package and one of its three procedures:

- REFRESH Refreshes the specified materialized view
- REFRESH_DEPENDENT Refreshes all materialized views that utilize this table
- REFRESH_ALL_MVIEWS Refreshes all materialized views in the schema that have not been refreshed since the last bulk load of data to the base tables

```
DBMS_MVIEW.REFRESH ('EMP_BY_DISTRICT')
DBMS_MVIEW.REFRESH_DEPENDENT ('EMPLOYEE')
DBMS_MVIEW.REFRESH_ALL_MVIEWS
```

Scheduled Refreshing

```
Begin
 v_job_num NUMBER;
BEGIN
DBMS_JOB.SUBMIT( v_job_num,
 'DBMS_MVIEW.REFRESH (''EMP_BY_DISTRICT'');',
 SYSDATE,
 -- every Sunday at 2:00 A.M.
 'NEXT_DAY(TRUNC(SYSDATE),''SUNDAY'') + 2/24'
 );
END;
```

or

```
ALTER MATERIALIZED VIEW EMP_BY_DISTRICT
REFRESH
START WITH TO_DATE('16-06-2005 01:00:00','dd-mm-yyyy hh24:mi:ss')
NEXT TRUNC(SYSDATE+1)+(1/24)
```

Disabling Rewrite on Materialized Views

- Set the parameter QUERY_REWRITE_ENABLED=FALSE dynamically or statistically.
- Set QUERY_REWRITE_ENABLED=FALSE at the session level using the ALTER SESSION command.
- At the statement level, use the NOREWRITE hint in a query.

Note: Users need the QUERY REWRITE or GLOBAL QUERY REWRITE system privilege in order to enable and disable the query rewrite functionality on materialized views in their own schema, or another user's schema, respectively.

Dropping Materialized Views

```
DROP MATERIALIZED VIEW emp_by_district;
```

Minimizing I/O Using Indexes

Index Types

- B-Tree index
- Compressed B-Tree
- Bitmap index
- Function-based Index
- Reverse Key Index (RKI)
- Index Organized Table (IOT)

What Columns to Index

Following are guidelines to decide on what columns to index:

- Index columns with high selectivity (that there are few rows with identical values).
- Index all important foreign keys.
- Index all predicate columns.
- Index columns used in table joins.
- For the most common queries in your application, create any necessary composite indexes

B-Tree Indexes

- B-Tree Indexes very useful when executing queries that will return only a small number of the overall rows found in a table.
- The best candidates for B-Tree indexes are those columns having high cardinality in their data. These columns should also be frequently used in SQL statement WHERE clauses.
- Indexes that have four or more levels between their root block and their deepest leaf blocks are good candidates for rebuild.

```
SELECT index_name, blevel
FROM dba_indexes
WHERE blevel >= 4
```

- Another indication that an index may benefit from a rebuild is evident when the deleted entries in the index represent more than 20 percent of the overall entries.

```
ANALYZE INDEX emp_lname_idx VALIDATE STRUCTURE
SELECT (DEL_LF_ROWS_LEN/LF_ROWS_LEN)*100 "Wasted Space" FROM index_stats
WHERE name = 'EMP_LNAME_IDX';
```

  Note: About the VALIDATE STRUCTURE:
  - It verifies the integrity of each data block in the index and checks for block corruption.
  - It places information into the INDEX_STATS view and allows for the monitoring of space used by an index.
  - This clause does not confirm that each row in the table has an index entry or that each index entry points to a row in the table.

Index Reorganization

Drop and Re-Create Index

- Dropping and re-creating an index is the most time-consuming and resource-intensive technique for rebuilding an index.

Using the ALTER INDEX ... REBUILD

- This method is an effective way to quickly rebuild an index because the existing index entries are used to create the new index using the fast full scan feature.
- This command has ONLINE option.
- This method can also be used to move an index to a new tablespace using TABLESPACE option.
- This method can be used to change the storage characteristics.
- Using this technique to rebuild indexes requires that additional disk space is temporarily available.

Using the ALTER INDEX ... COALESCE

- The COALESCE option coalesces the leaf blocks within the same index branch index.
- This minimizes potential locking issues associated with the rebuild process.
- The COALESCE option cannot be used to move an index to a new tablespace.

Compressed B-Tree Indexes

- Compressed B-Tree index eliminated duplicate occurrences of the indexed column value to reduce the overall amount of space needed to store the index. This is practically useful in case of large tables, particularly in a data warehouse or a Decision Support System.
- Compressed indexes are built by adding the keyword COMPRESS to the end of the CREATE INDEX statement.
- Alternatively, non-compressed indexes can be compressed using the ALTER INDEX statement:

```
ALTER INDEX emp_hist_lname_idx REBUILD COMPRESS
```

Bitmap Indexes

- For columns with low cardinality, a Bitmap index may be the most effective indexing option.
- Bitmap indexes create a binary mapping of the rows in the table and store that map in the index blocks.
- This type of index is particularly useful on columns of large, low-DML activity tables that contain very few distinct values.
- Increasing the size of each of the parameters SORT_AREA_SIZE and PGA_AGGREGATE_TARGET generally enhances the speed with which bitmap indexes can be created and manipulated.
  - o SORT_AREA_SIZE size in bytes of the buffer where sorted bitmap column and row ID information is stored until batch commit.
  - o PGA_AGGREGATE_TARGET manages the amount of memory, in bytes, assigned to bitmap index creation and bitmap merges following an index range scan.

Function-Based Indexes

- Function-Based index is created based on a function or operation applied to a table's column. Both B-Tree and Bitmapped indexes can be created as function-based.

```
CREATE INDEX emp_fname_upper_idx
ON employee (UPPER(fname));
```

Note: The parameter QUERY_REWRITE_ENABLED must be set to TRUE in order to use function-based indexes.

Reverse Key Indexes

- The Reverse Key Index (RKI) is a special type of B-Tree index. The RKI is useful when an index is built on a column that contains sequential numbers.

```
CREATE INDEX employee_emp_id_idx
ON employee (emp_id) REVERSE ;
```

```
ALTER INDEX emp_id_idx REBUILD REVERSE;
```

Note: Reverse Key indexes are only useful for equality and non-equality searches. Queries that perform range scans (e.g., using BETWEEN, >, <) on columns that are Reverse Key indexed will not be able to use the index and will cause full table scans.

Index Organized Tables (IOT)

- An index-organized table IOT differs from an ordinary table in that the data for the table is held in its associated index. Table data will be physically stored in order by the key.
- If you access the table using its primary key, an IOT will return the rows more quickly than a traditional table.
- There is only one mapping table to maintain per IOT, even if there are multiple Bitmap indexes on that IOT.
- All index-organized tables must have a primary key constraint defined on the column that will be the basis of the index. An IOT cannot contain unique constraints or be clustered.

```
CREATE TABLE employee_history
(employee_id number primary key,
 ... )
 ORGANIZATION INDEX
 TABLESPACE appl_idx
 PCTTHRESHOLD 25
 INCLUDING first_name
 OVERFLOW TABLESPACE appl_of
 MAPPING TABLE;
```

ORGANIZATION INDEX Specifies that this table is an IOT.

PCTTHRESHOLD Specifies what percentage of the entire data block to hold open in order to store the row data associated with a primary key value, which must be between 1 and 50 (default 50).

INCLUDING Specifies at which column to break a row into two pieces when a row's length exceeds the size set aside in PCTTHRESHOLD.

OVERFLOW TABLESPACE Specifies the tablespace where the second half of the row data will be stored when the row's length exceeds the size set aside in PCTTHRESHOLD. Thus it allows to keep the B-Tree structure densely clustered to allow more rows per leaf block.

Note: Segment containing the overflow data is described as OVERFLOW segment.

MAPPING TABLE Causes the creation of an associated mapping table that is needed when creating bitmap indexes on the IOT.

Mapping Table

Specify MAPPING TABLE to instruct Oracle to create a mapping of local to physical ROWIDs and store them in a heap-organized table. This mapping is needed in order to create a bitmap index on the index-organized table.

Because of the use of this intermediate mapping table, over time, the Bitmapped index entries may not accurately reflect the true mappings between the logical and physical row IDs in the indexed IOT. You can see the extent to which the mapping table is out of sync with the Bitmap index by issuing the following query:

```
SELECT index_name, pct_direct_access
FROM dba_indexes
WHERE pct_direct_access IS NOT NULL;
```

Only IOT tables will have non-null values for PCT_DIRECT_ACCESS. Any index whose PCT_DIRECT_ACCESS value exceeds 30 percent should have its Bitmapped index rebuilt in order to maintain its effectiveness.

Using Bitmap Join Indexes

Bitmap join indexes (BJIs) *prestore* the results of a join between two tables in an index, and thus do away with the need for an expensive runtime join operation. It is designed for tables with primary key (*dimension* table) and foreign key (*fact* table) relationship between them.

```
SELECT sum(s.quantity)
FROM sales s, customers c
WHERE s.customer_id = c.customer_id
  AND c.city = 'DALLAS';
```

```
CREATE BITMAP INDEX cust_orders_BJI
 on sales (c.cust_id) -- index on the fact table
 FROM sales s, customers c
 WHERE c.cust_id = s.cust_id
 LOCAL -- use this keyword if sales is partitioned
 TABLESPACE users;
```

```
SELECT index_name, index_type, join_index
 FROM user_indexes
 WHERE table_name='SALES';
```

```
INDEX_NAME                INDEX_TYPE    JOI
------------------------- ------------- ---
CUST_ORDERS_BJI           BITMAP        YES
```

Index Data Dictionary Views

DBA_INDEXES

INDEX_TYPE Specifies what type the index is:

- NORMAL (Standard B-Tree)
- NORMAL/REV (Reverse Key)
- FUNCTION-BASED NORMAL
- BITMAP
- IOT—TOP (Index Organized Table)
- CLUSTER
- LOB (Large Object)

DBA_SEGMENTS

You use DBA_SEGMENTS to display overflow segment.

```
SELECT segment_name, segment_type, tablespace_name
FROM dba_segments
WHERE segment_name LIKE '%IOT%';
```

DBA_TABLES

In order to more clearly see the relationship between an IOT and its associated overflow segment, you can query DBA_TABLES as shown below:

```
SELECT table_name, iot_name, iot_type, ablespace_name
FROM dba_tables
WHERE table_name = 'EMPLOYEE_HISTORY'
OR iot_name = 'EMPLOYEE_HISTORY';
```

Note: You will observe that the entry for TABLESPACE_NAME for the OIT table is NULL since the actual segment that stores the data is an index, not a table.

Identifying Unused Indexes

- You can use the following command to activate monitoring on an index called EMP_LAST_NAME_IDX:

```
ALTER INDEX hr.emp_last_name_idx MONITORING USAGE;
```

- After a period of normal user activity, you can disable the monitoring using the following command:

```
ALTER INDEX hr.emp_last_name_idx NOMONITORING USAGE;
```

- You can then query the INDEX_NAME and USED column in the V$OBJECT_USAGE dynamic performance view to determine whether the index was accessed during the monitoring period.

Partitions

Range Partitioning

Range partitioning uses ranges of column values to determine into which partition a row of data will be inserted.

```
CREATE TABLE student_history
  (student_id NUMBER(10),
  degree VARCHAR2(3),
  graduation_date DATE,
  final_gpa NUMBER)
  PARTITION BY RANGE (graduation_date)
    (PARTITION p_1997 VALUES LESS THAN
      (TO_DATE('01-JUN-1997','DD-MON-YYYY')) TABLESPACE hist_tab01,
    PARTITION p_1998 VALUES LESS THAN
      (TO_DATE('01-JUN-1998','DD-MON-YYYY')) TABLESPACE hist_tab02,
    PARTITION p_1999 VALUES LESS THAN
      (TO_DATE('01-JUN-1999','DD-MON-YYYY')) TABLESPACE hist_tab03,..
```

- The partition key can be composed of up to 16 columns.
- There cannot be any gaps in the partition ranges.
- All rows stored in a partition will have values less than, and not equal to the upper bound for that partition.
- Range-partitioned tables cannot contain columns with LONG or LONG RAW datatypes.
- Updates that would cause a record to move across partition boundaries are not allowed unless the ENABLE ROW MOVEMENT clause is specified at table creation.
- Attempts to insert records that do not fit into any defined partition ranges will cause an ORA-14400: Insert partition key is beyond highest legal partition key error.

Unevenness of data among the partitions can cause parallel operations like Parallel Query and Parallel DML to perform poorly. If this is the case, then Hash partitioning could be considered in place of Range partitioning.

List Partitioning

List partitions are similar to Range partitions except they are based on a set of specified values rather than a range of values.

```
CREATE TABLE student_history
  ( student_id NUMBER(10),
    degree VARCHAR2(3),
    graduation_date DATE,
    final_gpa NUMBER)
  PARTITION BY LIST (degree)
    (PARTITION p_undergrad VALUES ('BS','BA','BBA','BFA')
     TABLESPACE hist_tab01,
    PARTITION p_graduate VALUES ('MA','MBA','MFA','MS')
     TABLESPACE hist_tab02,
    PARTITION p_doctorate VALUES ('PHD')
     TABLESPACE hist_tab03);
```

Values that do not meet any of the conditions of the partition key will raise an ORA-14400 error.

Hash Partitioning

Hash partitions use a hashing algorithm to assign inserted records into partitions. This has the effect of maintaining a relatively even number of records in each partition, thus improving the performance of parallel operations like Parallel Query and Parallel DML.

The following syntax would create a table with six partitions, with systemgenerated names.

```
  CREATE TABLE student_history
    (student_id NUMBER(10),
     degree VARCHAR2(3),
     graduation_date DATE,
     final_gpa NUMBER)
     PARTITION BY HASH (student_id)
      PARTITIONS 6
      STORE IN (hist_tab01, hist_tab02, hist_tab03,
              hist_tab04, hist_tab05, hist_tab06);
```

- The partition key should have high cardinality—the column should contain unique values or very few duplicate values.
- Hash partitions work best when applications retrieve the data from the partitioned table via the unique key. Range lookups on hash partitioned tables derive no benefit from the partitioning.

- When they are properly indexed, hash partitioned tables improve the performance of joins between two or more partitioned tables.
- Updates that would cause a record to move across partition boundaries are not allowed.
- Local hash indexes can be built on each partition.

Oracle recommends that the total number of partitions be a power of two in order to maximize the effectiveness of the hash partition algorithm.

Composite Prtitioning

As the name implies, composite partitioning represents a combination of both range and hash partitioning. This type of partition is useful when Range partitioning is desired, but when the resulting number of values within each range would be uneven.

Composite partitioning creates Range partitions that are in turn subdivided into hashed subpartitions:

```
CREATE TABLE student_history
  (student_id NUMBER(10),
  degree VARCHAR2(3),
  graduation_date DATE,
  final_gpa NUMBER)
  PARTITION BY RANGE (graduation_date)
  SUBPARTITION BY HASH(student_id) SUBPARTITIONS 4
  STORE IN (hist_tb01,hist_tb02,hist_tb03,ist_tb04)
    (PARTITION p_1997 VALUES LESS THAN
      (TO_DATE('01-JUN-1997','DD-MON-YYYY')),
    PARTITION p_1998 VALUES LESS THAN
      (TO_DATE('01-JUN-1998','DD-MON-YYYY')),
    PARTITION p_1999 VALUES LESS THAN
      (TO_DATE('01-JUN-1999','DD-MON-YYYY')),
    PARTITION p_2000 VALUES LESS THAN
      (TO_DATE('01-JUN-2000','DD-MON-YYYY')),
    PARTITION p_2001 VALUES LESS THAN
      (TO_DATE('01-JUN-2001','DD-MON-YYYY')),
    PARTITION p_ERROR VALUES LESS THAN (MAXVALUE));
```

- The partitions are logical structures only; the table data is physically stored at the sub-partition level.
- Composite partitions are useful for historical, date-related *queries* at the partition level.
- Composite partitions are also useful for parallel operations (both *query* and *DML*) at the subpartition level.

Partition-wise joins are also supported through the use of composite local indexes. Global indexes are not supported.

Indexing Partitioned Tables

Partitioned Table Index Categories

A partitioned table should be indexed in order to take maximum advantage of the performance enhancing benefits of partitioned tables.

Global versus Local Partition Indexes

Local Partition Indexes

Partitioned indexes are said to be *local* when the number of partitions in the index match the underlying table's partitions on a one-to-one basis.

- Local partitioned indexes can use any of the four partition types.
- Oracle automatically maintains the relationship between the table's partitions and the index's partitions. If a table partition is merged, split, or dropped, the associated index partition will also be merged, split, or dropped.
- Any bitmapped index built on a partitioned table must be a local index.

Global Partition Indexes

Partitioned indexes are said to be *global* when the number of partitions in the index do not match the underlying table's partitions on a one-to-one basis.

- Although global indexes can be built on a range, list, hash, or composite partitioned table, the global partitioned index itself must be Range partitioned.
- The highest partition of a global index must be defined using the MAXVALUE parameter.
- Performing maintenance operations on partitioned tables can cause their associated global indexes to be placed into an invalid state. Indexes in an invalid state must be rebuilt before users can access them.

- Creating a global partitioned index with the same number of partitions as its underlying table (i.e. simulating a local partitioned index) does not substitute for a local partitioned index. The indexes would not be considered the same by the CBO despite their similar structures.

Prefixed versus Non-prefixed Partition Indexes

Prefixed Partition Indexes

Partitioned indexes are said to be *prefixed* whenever the left-most column of the index is the same as the underlying table's partition key column.

Non-prefixed Partitioned Indexes

Partitioned indexes are said to be *non-prefixed* when the left-most column of the index is not the same as the underlying table's partition key column.

- This category of indexes cannot be created as Global indexes.

Note: Traditional non-partitioned B-Tree indexes can also be built on partitioned tables. These indexes are referred to as ***global, prefixed non-partitioned indexes***.

Creating Partitioned Table Index

Creating a Local, Prefixed Partition Index

```
CREATE INDEX student_history_lp_idx ON student_history (graduation_date) LOCAL
```

Creating a Local, Non-prefixed Partition Index

```
CREATE INDEX student_history_lnp_idx
ON student_history (final_gpa) LOCAL
```

Creating a Global, Prefixed Partition Index

```
CREATE INDEX student_history_gp_idx
ON student_history (graduation_date)
GLOBAL PARTITION BY RANGE (graduation_date)
  (PARTITION p_199n VALUES LESS THAN
    (TO_DATE('01-JUN-1999','DD-MON-YYYY')),
  PARTITION p_200n VALUES LESS THAN
    (TO_DATE('01-JUN-2009','DD-MON-YYYY')),
  PARTITION p_error VALUES LESS THAN (MAXVALUE));
```

Using the Clause UPDATE GLOBAL INDEXES

- When you perform DDL on a table partition, if a global index is defined on table, then Oracle invalidates the entire index, not just the partitions undergoing DDL.
- Using the clause UPDATE GLOBAL INDEXES you obtain the following benefits:
  - The clause updates the global index partition you are changing during the DDL operation, eliminating the need to rebuild the index after the DDL and avoiding problems with the UNUSABLE status.
  - Global indexes remain intact and available for use by data manipulation language (DML) statements even for sessions that have not enabled the skipping of unusable indexes.
  - Global indexes remain intact and available for use by data manipulation language (DML) statements
- However, Global indexes are ***not*** maintained during the operation of the DDL command and therefore can not be used by any concurrent query.

How the Partitioned Tables Affect the Optimizer

CBO can eliminate partitions that will not be part of the query result. This quick elimination of unneeded partitions to improve query performance is referred to as *partition pruning*.

Partitioned tables and indexes also help the CBO develop effective execution plans by influencing the manner in which joins are performed between two or more partitioned tables. When performed in parallel using Oracle Parallel Query, there are two possible join methods that are considered by the CBO:

Full Partition-wise Joins

Partition-wise joins occurs any time two tables are joined on their partition key columns. Partition-wise joins can be performed either serially or in parallel.

Partial Partition-wise Joins

Partial partition-wise joins occur any time two tables are joined on the partition key columns of one of the two tables. Partial partition-wise joins can only be performed in parallel.

Accurate table and index statistics are vital for efficient CBO performance. You can use DBMS_STATS.GATHER_TABLE_STATS and DBMS_STATS.GATHER_INDEX_STATS to gather statistics about table partitions and their indexes.

Data Dictionary Views Related to Partitions

DBA_PART_TABLES
Contains details about each partitioned table

DBA_TAB_PARTITIONS
Contains details about each table's individual partitions

DBA_TAB_SUBPARTITIONS
Contains details about each table's individual subpartitions

DBA_PART_INDEXES
Contains details about each partitioned index

DBA_IND_PARTITIONS
Contains partition details about index partitions

DBA_IND_SUBPARTITIONS
Contains partition details about index subpartitions

DBA_PART_KEY_COLUMNS
Contains the partition key columns for all partitioned tables and indexes

DBA_LOB_PARTITIONS
Contains partition details about large object partitions

DBA_LOB_SUBPARTITIONS
Contains partition details about large object subpartitions

DBA_PART_COL_STATISTICS
Contains columns statistics for partitioned tables

DBA_PART_HISTOGRAMS
Contains histogram statistics for partitioned tables

DBA_PART_LOBS
Contains details about each partitioned large object

DBA_SUBPART_COL_STATISTICS
Contains subpartition column statistics

DBA_SUBPART_HISTOGRAMS
Contains subpartition histogram statistics

DBA_SUBPART_KEY_COLUMNS
Contains the partition key column for each subpartitioned table or index.

Index and Hash Clusters

A cluster is a group of one or more tables whose data is stored together in the same data blocks.
There are two types of clusters available in Oracle9i: the Index cluster and the *Hash cluster*.

Index Clusters

Index clusters are used to store the data from one or more tables in the same physical Oracle blocks.
In general, clustered tables should have these attributes:

- Always be queried together and only infrequently on their own
- Have little or no DML activity performed on them after the initial load
- Have roughly equal numbers of child records for each parent key

Note: Normal B-Tree indexes do not store null key values, whereas cluster indexes store null keys.

Creating a Cluster

1. Issue the CREATE CLUSTER command:

```
CREATE CLUSTER teacher_student
 (teacher_id number)
```

```
SIZE 1000
STORAGE (initial 500K next 500K pctincrease 0);
TABLESPACE apps_clst;
```

The parameter SIZE specifies how many cluster keys we expect to have per Oracle block. If we have a 2048-byte (2K) Oracle block size, then this parameter indicates that we expect to get about two (2048/1000 = 2.048) cluster keys per block.

Choosing the proper value for SIZE is difficult if all the cluster key values (teachers) don't have the same number of clustered records (students). However, setting this value properly is critical to gaining optimal cluster performance.

2. Create an index for the cluster

```
CREATE INDEX TEACHER_STUDENT_IDX
 ON CLUSTER teacher_student
 TABLESPACE apps_idx
```

The index is automatically created on the cluster key column, TEACHER_ID.

3. Create tables in the cluster

```
CREATE TABLE TEACHER
  (teacher_id number,
  last_name varchar2(20),
  first_name varchar2(20),
  room_number number)
  CLUSTER teacher_student (teacher_id);
CREATE TABLE student
  (student_id number,
  last_name varchar2(20),
  first_name varchar2(20),
  teacher_id number)
  CLUSTER teacher_student (teacher_id);
```

Hash Clusters

Hash clusters are used in place of a traditional index to quickly find rows stored in a table. In general, Hash cluster tables should have these attributes:

- Have little or no DML activity performed on them after the initial load
- Have a uniform distribution of values in the indexed column
- Have a predictable number of values in the indexed column
- Be queried with SQL statements that utilize equality matches against the indexed column in their WHERE clauses

OLTP vs. DSS Tuning Requirements

Tuning OLTP Systems

Online Transaction Processing (OLTP) systems tend to be accessed by large numbers of users doing short DML transactions.

- Users of OLTP systems are primarily concerned with throughput.
- OLTP systems need enough B-Tree and Reverse Key indexes to meet performance goals but not so many as to slow down the performance of DML activities.
- Bitmap indexes are not a good choice for OLTP systems because of the locking issues they can cause.
- Table and index statistics should be gathered regularly.
- OLTP indexes should also be rebuilt frequently.

Tuning DSS Systems

Decision Support Systems (DSS) and data warehouses tend to have very little if any DML activities, except when data is mass loaded or purged at the end of the week, month, quarter, or year. Users of these systems are concerned with response time, which is the time it takes to get the results from their queries.

- DSS makes heavy use of full table scans so the appropriate use of indexes and Hash clusters are important. Index-organized tables can also be important tuning options for large DSS systems.
- Bitmap indexes may be considered where the column data has low cardinality but is frequently used as the basis for queries.
- Consideration should be given to database block size, initialization parameters related to sorting, and the possible use of the Oracle Parallel Query option.
- Database statistics should also be gathered following each data load.

Tuning the Shared Pool

Principles

Cache Hit :finding a matching SQL statement in the Shared Pool.

Cache Miss :not finding a matching statement and having to perform the parse operation.

Components of the Shared Pool

Library Cache

Library Cache is the location where Oracle caches the SQL and PL/SQL statements that have been recently issued by application users.

Many of these components for an SQL statement can be viewed in four dynamic performance views:

V$SQL SQL statement text and statistics for all cached SQL statements including I/O, memory usage, and execution frequency

V$SQLAREA similar to V$SQL , but displays only information for SQL statements that are cached in memory, parsed, and ready for execution

V$SQLTEXT the complete text of the cached SQL statements along with a classification by command type

V$SQL_PLAN execution plan information for each cached SQL statement

V$DB_OBJECT_CACHE view can be used to determine what types of database objects are being referenced by the application's SQL statements.

Data Dictionary Cache

Whenever an SQL or PL/SQL statement is issued, the data dictionary must also be examined to make sure that the tables referenced in the statement exist, that the column names and data types are correct, and that the application user issuing the statement has sufficient privileges on the segments involved.

User Global Area

In the shared server architecture, the User Global Area is used to cache application user session information.

Measuring the Performance of the Shared Pool

The primary indicator of the performance of the Shared Pool is the cache-hit ratio.

Measuring Library Cache Performance

Using V$LIBRARYCACHE to Monitor the Library Cache

This view contains statistics about library cache performance and activity.

NAMESPACE: the library cache namespace.

Rows of the view with the following NAMESPACE values reflect library cache activity for SQL statements and PL/SQL blocks: SQL AREA, TABLE/PROCEDURE, BODY, and TRIGGER.

Rows with other NAMESPACE values reflect library cache activity for object definitions that the server uses for dependency maintenance: INDEX, CLUSTER, OBJECT, PIPE, JAVA SOURCE, JAVA RESOURCE, and JAVA DATA.

GETS: the total number of requests for information or lock on the corresponding item

GETHITS: number of times an object's handle was found in memory

GETHITRATIO: the ratio of GETHITS to GETS

PINS: Shows the number of executions of SQL statements or procedures

PINHITS: number of times all of the metadata pieces of the library object were found in memory

PINHITRATIO: the ratio of PINHITS to PINS

RELOADS: Shows the number of times, during the execution phase, that the shared SQL area containing the parsed representation of the statement is aged out of the library cache to make room for another statement. The Oracle server implicitly reloads the statement and parses it again.

INVALIDATIONS: Shows the number of statements that have been made invalid due to the modification of a dependent object. Invalidations also cause reloads.

- According to Oracle, well-tuned OLTP systems can expect to have GETHITRATIO and PINHITRATIO of **90 percent** or higher for the SQL Area portion of the Library Cache. DSS and data warehouse applications often have lower hit ratios because of the ad hoc nature of the queries against them.
- Oracle considers well-tuned systems to be those with Reload Ratios (SUM(reloads)/SUM(pins)) of less than **1 percent**.

Using STATSPACK to Monitor the Library Cache

- Library cache performance statistics are included in the STATSPACK section headed *Instance Efficiency Percentages (Target 100%)*.

- STATSPACK also includes a second section, headed by *Library Cache Activity for DB*, which shows the same information about the invalidations and reloads.

Using Enterprise Manager to Monitor the Library Cache

The Performance Manager component of the Oracle Enterprise Manager GUI tool contains several sources of Library Cache performance information.

Measuring Data Dictionary Cache Performance

Using V$ROWCACHE to Monitor the Data Dictionary Cache

```
SELECT 1 - (SUM(getmisses)/SUM(gets))
 "Data Dictionary Hit Ratio"
 FROM v$rowcache
```

Note: Oracle recommends that consideration be given to tuning the Shared Pool if the overall Data Dictionary Cache hit ratio is less than **85 percent**.

Using STATSPACK to Monitor the Data Dictionary Cache

Data Dictionary Cache performance indicators can be found in the output from the STATSPACK utility in the section headed as *Dictionary Cache Stats for DB*.

Improving Shared Pool Performance

Make it bigger

Making the Shared Pool larger lessens the likelihood that cached information will be moved out of either cache by the LRU mechanism.

The V$DB_OBJECT_CACHE and V$SQLAREA dynamic performance views can be used to arrive at an estimated size for the Library Cache.

```
SELECT SUM(SHARABLE_MEM) -- in bytes
FROM v$db_object_cache
plus
SELECT SUM(SHARABLE_MEM)
FROM v$slqarea
WHERE EXECUTIONS>10
```

Make room for large PL/SQL statements

An area from Library Cache can be reserved for large PL/SQL packages called *Shared Pool Reserved Area*. This is set by the parameter SHARED_POOL_RESERVED_SIZE.

The value of SHARED_POOL_RESERVED_SIZE is specified in bytes and can be set to any value up to 50 percent of the value specified by SHARED_POOL_SIZE (by default it is 50 percent).

Oracle recommends that you aim to have REQUEST_MISSES and REQUEST_FAILURES near zero in V$SHARED_POOL_RESERVED when using the Shared Pool Reserved Area.

If you issue the following:

```
dbms_shared_pool.aborted_request_threshold(10000)
```

This allows a large PL/SQL object to flush up to 10,000 bytes from the Shared Pool's LRU list. If an application user exceeds this imposed limit, Oracle will return an out of memory error to the user.

Keep important PL/SQL code in memory

Frequently used PL/SQL objects can be pinned in the SHARED_POOL_RESERVED_SIZE area.

1. Create DBMS_SHARED_POOL by running %ORACLE_HOME%\rdbms\admin\dbmspool.sql
2. Pin or unpin a package using KEEP or UNKEEP procedures
   ```
   DBMS_SHARED_POOL.KEEP ('APPROVE_PO')
   ```
3. To identify pinned objects
   ```
   SELECT owner, name, type
   FROM v$db_object_cache
   WHERE kept='YES'
   ```

To determine which objects to pin, you can activate Oracle's auditing feature on the objects that your are considering pinning in operation times. Then specify heavily used objects.

You can make the database to automatically load the pinned objects into the memory when it starts up by using AFTER STARTUP ON DATABASE trigger.

Encourage code reuse

Application developers should use coding standards that specify the appropriate use of case, spacing, and lines in the application code. This it to increase the likelihood of a statement to find a similar cached statement in the memory.

Generally, it is recommended to use bind variables instead of using literals in SQL statements. However, bind variables may confuse the cost-based optimizer, which is forced to make a blind guess at how the value in the container might affect the optimal execution plan.

Tune Library Cache–specific initialization parameters

OPEN_CURSORS

This parameter limits how many cursors can be opened by a user's session. The default values (50) is usually not adequate for most applications.

CURSOR_SPACE_FOR_TIME

When this parameter is set to TRUE, shared SQL areas are pinned in the Shared Pool. This prevents the LRU mechanism from removing a shared SQL area from memory unless all cursors that reference that shared SQL area are closed.

To switch it to TRUE you should confirm that the value in the RELOADS column of V$LIBRARYCACHE is consistently zero.

Oracle recommends that applications which utilize Oracle Forms should ***not*** set CURSOR_SPACE_FOR_TIME to TRUE.

SESSION_CACHED_CURSORS

This parameter lets you specify the number of session cursors to cache. Repeated parse calls of the same SQL statement cause the session cursor for that statement to be moved into the session cursor cache instead of reopening a new cursor.

To tune value of this parameter, you can examine the session statistic session cursor cache hits in the V$SYSSTAT view. If this statistic is a relatively low percentage of the total parse call count for the session, then consider setting SESSION_CACHED_CURSORS to a larger value.

Oracle recommends configuring this parameter for Oracle Forms based applications.

CURSOR_SHARING

FORCE Allows two SQL statements, which differ only by a literal value, to share parsed code cached in the Shared Pool. The difference in the literal values must not change the meaning of the statement.

SIMILAR Allows two SQL statements, which differ only by a literal value, to share parsed code cached in the Shared Pool. The difference in the literal values must not change the meaning of the statement or its cached execution plan.

EXACT Two SQL statements must match exactly in order to share the parsed code cached in the Shared Pool. This is the default value.

- Using CURSOR_SHARING = SIMILAR (or FORCE) can significantly improve cursor sharing on some applications that have many similar statements, resulting in reduced memory usage, faster parses, and reduced latch contention.
- Setting CURSOR_SHARING to SIMILAR (or FORCE) prevents any outlines generated with literals from being used if they were generated with CURSOR_SHARING set to EXACT.
- Oracle does not recommend setting CURSOR_SHARING to FORCE in a DSS environment.

Tuning the Database Buffer Cache

Possible States for Database Buffer Cache Buffers

Free	Buffer is not currently in use.
Pinned	Buffer is currently in use by a Server Process..
Clean	A buffer that was just released from a pin, and has been marked for immediate reuse if the contents are not requested again because the data in the buffer was not changed.
Dirty	Buffer is not in use, but contains committed data that has not yet been written to disk by Database Writer

Measuring the Performance of the Database Buffer Cache

Performance measurement indicators:

- Buffer Busy Wait events
- Free Buffer Inspected waits
- Free Buffer Wait events
- Cache hit ratio

Buffer Busy Waits

This is the number of times user Server Processes waited for a free buffer to become available. These waits occur whenever a buffer requested by user Server Processes is already in memory, but is in use by another process.

How to measure

```
SELECT event, total_waits
FROM v$system_event
WHERE event = 'buffer busy waits'
```

Or in the STATSPACK output, under a section named "*Buffer Pool Statistics..*".

Causes and Resolution

If value of this measure is high or is increasing constantly, then determine the wait statistics for each class of buffer by querying the v$waitstat view. Common buffer classes that have buffer busy waits include:

Data Block

If the contention is on tables or indexes (not the segment header):

- Check for SQL statements using unselective indexes.
- Check for right-hand-indexes (that is, indexes that are inserted at the same point by many processes; for example, those which use sequence number generators for the key values).
- Consider using automatic segment-space management or increasing free lists to avoid multiple processes attempting to insert into the same block.
- The v$session_wait view will provide the file and block numbers (in the P* columns) for those blocks that have the most frequent block waits. These blocks can then be mapped to which object they belong.

Undo Header

Displays contention on rollback segment header: If you are not using automatic undo management, then add more rollback segments.

Undo Block

Displays contention on rollback segment block: If you are not using automatic undo management, consider making rollback segment sizes larger.

Free Buffer Inspected

This is a measure of how many buffers on the LRU list are inspected by a process looking for a free buffer (writing a new block) before triggering DBWn to flush the dirty buffers to disk.
A closely related statistic is *dirty buffer inspected*, which represents the total number of dirty buffers a user process found while trying to find a free buffer.

How to measure

```
SELECT name, value
FROM v$sysstat
WHERE name ='free buffer inspected'
```

Or in the STATSPACK output, under a section named "*Instance Activity Stats for..*".

Free Buffer Waits

This wait event indicates that a server process was unable to find a free buffer and has requested the database writer to make free buffers by writing out dirty buffers.

How to measure

```
SELECT event, total_waits
FROM v$system_event
WHERE event = 'free buffer waits'
```

Or in the STATSPACK output, under a section named "*Buffer Pool Statistics..*".

Causes and Resolution

- The I/O system is slow.
  Solution: Check that the files are equally distributed across all devices. If that produces no effect get faster disks or place offending files onto faster disks.
- The I/O is waiting for resources, such as latches.
  Solution: Check that the files are equally distributed across all devices. If that produces no effect get faster disks or place offending files onto faster disks.
- The buffer cache is so small that DBWn spends most of its time cleaning out buffers for server processes.
  Solution: Increase the buffer cache size.

The buffer cache is so large that one DBWn process cannot free enough buffers in the cache to satisfy requests.
Solution: Decrease the buffer cache size or initialize more database writer processes.

Measuring the Cache Hit Ratio Using V$SYSSTAT

Four of statistics tracked by V$SYSSTAT are used when calculating the performance of the Database Buffer Cache:

Physical Reads	This statistic indicates the number of data blocks (i.e., tables, indexes, and rollback segments) read from disk into the Buffer Cache since instance startup.
Physical Reads Direct	This statistic indicates the number of reads that bypassed the Buffer Cache because the data blocks were read directly from disk instead. Because direct physical reads are done intentionally by Oracle when using certain features like Parallel Query, these reads are subtracted from the Physical Reads value when the Buffer Cache hit ratio is calculated.
Physical Reads Direct (LOB)	This statistic indicates the number of reads that bypassed the Buffer Cache because the data blocks were associated with a Large Object (LOB) datatype.
Session Logical Reads	This statistic indicates the number of times a request for a data block was satisfied by using a buffer that was already cached in the Database Buffer cache. For read consistency, some of these buffers may have contained data from rollback segments.

```
SELECT 1 - (phy.value - lob.value - dir.value)/  ses.value "CACHE HIT RATIO"
FROM v$sysstat ses, v$sysstat lob,
     v$sysstat dir, v$sysstat phy
WHERE ses.name = 'session logical reads'
AND dir.name = 'physical reads direct'
AND lob.name = 'physical reads direct (lob)'
AND phy.name = 'physical reads'
```

According to Oracle, a well-tuned OLTP system should have Database Buffer Cache hit ratios of **90 percent** or higher.

Measuring the Cache Hit Ratio Using STATSPACK

In a section named "*Instance Efficiency Percentages (Target 100%)*", you will see a measurement of *Buffer Hit%*.

Improving Buffer Cache Performance

Make It Bigger

The larger the Database Buffer Cache, the less likely cached buffers are to be moved out of the cache by the LRU List.

The size of the Database Buffer Cache is determined by the following initialization parameters:

DB_BLOCK_SIZE	Defines the primary block size (in bytes) for the database. Default is 2048 in Windows 2000 platform. SYSTEM and TEMP tablespaces always have the primary block size.
DB_CACHE_SIZE	Determines the size of the Default Buffer Pool for buffers with the primary block size. Syntax DB_CACHE_SIZE=integer [K|M|G] Parameter class Dynamic: ALTER SYSTEM
DB_KEEP_CACHE_SIZE	Determines the size of the Keep Buffer Cache. The default size is 0KB. It uses the primary block size as its buffer size. Parameter class Dynamic: ALTER SYSTEM
DB_RECYCLE_CACHE_SIZE	Determines the size of the Recycle Buffer Cache. The default size is 0KB. Also uses the primary block size as its buffer size.
DB_nK_CACHE_SIZE	If nKB is not the primary block size, specifies the size of the Buffer Cache for segments with a nKB block size. n ranges from 2 to 32.

Note: Unlike Oracle8i, where the memory for the Keep and Recycle pools was taken from the memory allocated to the Default Pool, Oracle9i independently assigns the memory to each of the three Buffer Pool types.

Note: Buffer pools are assigned to a segment, so option with multiple segments can have blocks in multiple buffer pools.

Note: There are at least 50 blocks per LRU latch for each pool.

Dynamically Increasing the Buffer Cache Size

```
ALTER SYSTEM SET db_cache_size = 100M
```

- The size specified must be a multiple of the granule size.
- The total size of the Buffer Cache, Shared Pool, and Redo Log Buffer cannot exceed the value specified by SGA_MAX_SIZE.

Configuring the Buffer Cache Advisory Feature

DB_CACHE_ADVICE

OFF turns off the Buffer Cache Advisory feature and releases any memory or CPU resources allocated to it.

READY pre-allocates memory to the Buffer Cache Advisory process, but does not actually start it.

ON memory will be allocated by Buffer Cache Advisory and it will start gathering statistics about the buffer cache's performance.

Note: you may get an *ORA-04031 unable to allocate n bytes of shared memory* error when you dynamically try to set DB_CACHE_ADVICE to ON.

V$DB_CACHE_ADVICE

NAME name of the Buffer Pool being referenced.

BLOCK_SIZE block size (in bytes).

ADVICE_STATUS status of the Advisory (OFF, READY, ON).

SIZE_FOR_ESTIMATE cache size estimate used when predicting the number of physicals reads, (in MB).

BUFFERS_FOR_ESTIMATE same as SIZE_FOR_ESTIMATE (in buffers).

ESTD_PHYSICAL_READ_FACTOR ratio comparing the estimated number of physical reads to the actual number of reads that occurred against the current cache, null if no physical reads occurred.

ESTD_PHYSICAL_READS estimated number of physical reads that would have occurred if the cache size specified by SIZE_FOR_ESTIMATE and BUFFER_FOR_ESTIMATE had actually been used for the DB_CACHE_SIZE.

Use Multiple Buffer Pools

- Oracle recommends you consider caching in the Keep Pool frequently accessed segments whose total size is less than 10 percent of the Default Pool size.
- Oracle recommends that you consider caching in the Recycle Pool segments whose blocks will not be accessed outside of individual transactions, and segments whose total size is more than twice the size of the Default Pool.

Using V$BH and DBA_OBJECTS to Identify Caching Candidates

V$BH

This view gives the status and number of pings for every block in the buffer cache. It can be used to obtain number of blocks taken up by which segments.

BLOCK# The block number of the block cached in the Buffer Cache

OBJD The Object ID of the object associated with the blocks that are cached in the Buffer Cache.

```
SELECT obj.owner,obj.object_name,obj.object_type,
   COUNT(DISTINCT bh.block#) "Num. Buffers"
  FROM dba_objects obj, v$bh bh
  WHERE obj.object_id = bh.objd
    AND owner != 'SYS'
  GROUP BY obj.owner,obj.object_name,obj.object_type
ORDER BY 4 DESC
```

Using V$CACHE and DBA_USERS to Identify Caching Candidates

- The V$CACHE view contains information from the block header of each block in the SGA as related to particular database objects. It can also be used to obtain number of blocks used by which segments.
- Before using the V$CACHE view, you must create it by running the script CATPARR.SQL. This script should be run while logged in as the user SYS.

```
SELECT username "Owner", name "Seg. Name",
   kind "Seg. Type",
   COUNT(DISTINCT block#) "Num. Buffers"
  FROM  v$cache, dba_users
  WHERE v$cache.owner#=dba_users.user_id
  AND username !='SYS'
  GROUP BY name, username, kind
  HAVING COUNT(DISTINCT block#) > 10
ORDER BY 3 DESC
```

Assigning Segments to Pools

```
ALTER TABLE apps.employee
STORAGE (BUFFER_POOL KEEP)

ALTER TABLE apps.division
STORAGE (BUFFER_POOL RECYCLE)
```

You can view which segments have been assigned non-Default Buffer Pools by querying BUFFER_POOL in the DBA_SEGMENTS.

Monitoring the Performance of Multiple Buffer Pools

V$BUFFER_POOL

This view displays information about all buffer pools available for the instance.

NAME The name of the Buffer Pool. Possible values: DEFAULT, KEEP, RECYCLE.

BLOCK_SIZE The block size (in bytes)

CURRENT_SIZE The current size of the Buffer Pool (in megabytes)

V$BUFFER_POOL_STATISTICS

This view displays statistics about all buffer pools.

NAME The name of the Buffer Pool

DB_BLOCK_GETS Number of requests for segment data that were satisfied by using cached segment buffers

CONSISTENT_GETS Number of requests for segment data that were satisfied by using cached rollback blocks for read consistency

PHYSICAL_READS The number of requests for segment data that required that the data be read from disk into the Buffer Cache

The formula 1 - (physical_reads / (db_block_gets + consistent_gets)) will retrieve hit ratio of every buffer pool.

Cache Tables in Memory

Tables designated as being *cache tables* do not have their buffers placed at the least recently used end of the LRU List when they are accessed via an FTS (Full Table Scan). Instead, these buffers are placed at the most recently used end of the LRU List just as if they had not been full table scanned.

```
CREATE TABLE phone_list .. CACHE
ALTER TABLE employee CACHE
```

You can also dynamically CACHE or NOCACHE tables using the appropriate hint:

```
SELECT /*+ CACHE */ last_name, first_name
FROM employee
```

Tuning Other SGA Areas

The SGA and PGA in the Shared Server

Cursor state and user session data will be stored in the large pool area, if configured. Otherwise they will be stored in SGA.

Configuring Oracle Shared Server

- Required initialization parameters
  - DISPATCHERS
  - SHARED_SERVERS
- Optional initialization parameters
  - MAX_DISPATCHERS
  - MAX_SHARED_SERVERS
  - CIRCUITS
  - SHARED_SERVER_SESSIONS

Monitoring Oracle Shared Server Performance

Calculating busy ratio of Shared Server process:

```
SELECT name,
DECODE(busy+idle,0,0,ROUND((busy/(busy+idle))*100,4)) "Busy Rate"
FROM  v$shared_server
WHERE status != 'QUIT'
```

High or increasing Shared Server busy ratios indicate that additional Shared Server processes may need to be spawned at instance startup.

Calculating average waiting time of request queue processes

Monitor these statistics while your application is running:

```
SELECT
DECODE(totalq,0,0,ROUND(SUM(wait)/SUM(totalq),4) "Avg Dispatcher Wait"
FROM v$queue
WHERE type='COMMON'
GROUP BY totalq
```

WAIT column is total time that all items in the queue spent waiting in 100ths of a second.

Calculating busy ratio of Dispatchers

V$DISPATCHER

NETWORK Protocol used by the Dispatcher

STATUS Current status of the Dispatcher. Possible Dispatcher statuses are:

- WAIT (idle)
- SEND (sending a message)
- RECEIVE (receiving a message)
- CONNECT (establishing connection)
- DISCONNECT (disconnecting a session)
- BREAK (performing a break)
- TERMINATE (terminating a connection)
- ACCEPT (accepting connections)
- REFUSE (refusing connection)

OWNED Number of virtual circuits owned by this Dispatcher

BUSY Total time (in 100ths of second) that the Dispatcher was busy

IDLE Total time (in 100ths of second) that the Dispatcher was idle

```
SELECT name "DISPATCHER", network,
(ROUND(SUM(busy)/(SUM(busy)+SUM(idle)),4))*100 "BUSY_RATE"
FROM v$dispatcher
GROUP BY name, network
```

Oracle recommends adding additional Dispatcher processes if the Dispatcher busy rate consistently exceeds **50 percent**.

Calculating user processes waiting for dispatchers to accept their requests (contention for dispatchers)

```
SELECT
  DECODE(totalq,0,0,ROUND(SUM(wait)/SUM(totalq)),4) "AVG DISPATCHER WAIT"
FROM v$queue q, v$dispatcher d
WHERE q.paddr = d.paddr
```

```
AND q.type='DISPATCHER'
GROUP BY totalq
```

High or increasing waits for Dispatcher processes may indicate a need to spawn additional Shared Server processes at instance startup.

Another method is to use V$DISPATCHER_RATE view:

V$DISPATCHER_RATE

NAME Name of the Dispatcher

CUR_IN_CONNECT_RATE Current rate at which the Dispatcher is servicing requests from application users

MAX_IN_CONNECT_RATE Maximum rate at which the Dispatcher has serviced requests from application users

```
SELECT NAME, cur_in_connect_rate - max_in_connect_rate "VARIANCE"
FROM v$dispatcher_rate
```

Large variances between current and maximum Dispatcher servicing rates indicate that additional Dispatcher processes may be required to improve Shared Server performance.

Overview of the cumulative activity of the Shared Server environment

V$SHARED_SERVER_MONITOR

MAXIMUM_CONNECTIONS The maximum number of virtual circuits used by the Shared Server options since instance startup

MAXIMUM_SESSIONS The maximum number of Shared Server sessions that have been utilized since instance startup

SERVERS_STARTED Total number of additional Shared Servers that were automatically started by the Shared Server option since the instance was started—beyond the number of Shared Servers specified by the SHARED_SERVERS parameter

SERVERS_TERMINATED Total number of Shared Servers that were automatically shut down by the Shared Server option since the instance was started

SERVERS_HIGHWATER The maximum number of Shared Servers running since the instance was started—both those that were started explicitly in the SHARED_SERVERS and those started implicitly by PMON.

Improving Shared Server Performance

Increase the Size of Related SGA Components

The following query can be used to determine the maximum amount of memory that has been used to store user session and cursor state information since instance startup:

```
SELECT SUM(value) "Total UGA Bytes"
FROM v$sesstat s, v$statname n
WHERE n.name = 'session uga memory max'
AND s.statistic# = n.statistic#
```

The result of this query should be added to the shared pool size or you can configure large pool size.

Increasing the Number of Shared Servers

Whenever the allocated Shared Servers are unable to service the incoming requests for database activity, the PMON background process will dynamically start additional Shared Server processes, up to the limit specified by the MAX_SHARED_SERVERS value.

You can also explicitly add additional Shared Server processes either dynamically or manually.

```
ALTER SYSTEM SET MAX_SHARED_SERVERS = 10
```

Increasing the Number of Dispatchers

Dispatchers can only be added dynamically or manually.

```
ALTER SYSTEM SET dispatchers = 'tcp, 6'
```

Large Pool Concepts

Large Pool can be configured to save memory space used by UGA data, RMAN utility and Parallel Query (PQ) instead of using the shared pool for this purpose.

Configuring the Large Pool

To create a Large Pool, set the parameter LARGE_POOL_SIZE to the desired size (using K or M suffix). This parameter is static.

Measuring the Performance of the Large Pool

```
SELECT name, bytes FROM v$sgastat
WHERE pool = 'large pool'
```

```
NAME        BYTES
----------- --------
PX msg pool 2347781
free memory 612030
```

Java Pool Concepts

Java pool is used to store session-specific Java code and application variables.

Configuring the Oracle Java Environment

The JAVA_POOL_SIZE parameter is used to configure Java pool size in the SGA. This parameter is static.

However the related parameters are the following:

SHARED_POOL_SIZE Shared Pool memory is used by the JVM when Java classes are loaded and compiled and when Java resources in the database are accessed.

JAVA_POOL_SIZE All other Java-related session data is cached in the Java Pool memory structure. Default size is 20M

JAVA_SOFT_SESSIONSPACE_LIMIT When memory is allocated to a Java process, the amount of memory requested is compared to this

limit. If the request exceeds this limit, a message is written to a user trace file. Default value is zero; maximum value is 4GB.

JAVA_MAX_SESSIONSPACE_SIZE When memory is allocated to a Java process, the amount of memory requested is compared to this limit. If the request exceeds this limit, an out-of memory error (ORA-29554) occurs. Default value is zero; maximum value is 4GB.

Measuring the Performance of the Java Pool

Measuring Java Pool Performance Using V$SGASTAT

```
SELECT name, bytes
FROM v$sgastat
WHERE pool = 'java pool';
```

```
NAME          BYTES
------------- ---------
free memory   58720256
memory in use 277821
```

Measuring Java Pool Performance Using STATSPACK

The section named "*SGA breakdown difference*" displays the value of Java pool at the beginning of the STATSPACK monitoring and its value by the end of the snapshot period.

Tuning Redo Mechanisms

Performance And Safety Tradeoffs

Operating a production database in non-Archive Log Mode just to avoid additional I/O is not advised.

Measuring the Performance of the Redo Log Buffer

The performance measurement is considered by the number and length of waits that user Server Processes experience when trying to place entries in the Redo Log Buffer.

Using V$SYSSTAT

- Considered statistics are
  - redo entries reports the number of entries that have been placed into the Redo Log Buffer by the user Server Processes since instance startup.
  - redo buffer allocation retries refers to the number of times user Server Processes had to wait and then retry placing their entries in the Redo Log Buffer because LGWR had not yet written the current entries to the online redo log.
- Therefore *Redo Log Buffer Retry Ratio* can be calculated by the following:

```
SELECT retries.value/entries.value "Redo Log Buffer Retry Ratio"
FROM v$sysstat retries, v$sysstat entries
WHERE retries.name = 'redo buffer allocation retries'
AND entries.name = 'redo entries'
```

- Oracle recommends that this *Redo Log Buffer Retry Ratio* should be less than 1 percent.
- redo log space requests in V$SYSSTAT measures how often LGWR is waiting for a Redo Log switch to occur when moving from the current Online Redo Log to the next. This will lead to the same problem described above. Possible reasons:
  - too small redo logs.
  - I/O contention.

Tip of Sizing the Redo Log Buffer

If your database requires extra memory for database buffer cache and the server memory does not have more free space, then think of utilizing Redo log buffer by reducing its size to 1M and releasing the rest to the buffer cache.

Using V$SESSION_WAIT

To display sessions waiting for log buffer:

```
SELECT username, wait_time, seconds_in_wait
FROM v$session_wait, v$session
WHERE v$session_wait.sid = v$session.sid
AND event LIKE '%log buffer space%'
```

Using STATSPACK Output

- The section headed *Instance Efficiency Percentages* indicates the performance of the Redo Log Buffer in terms of a ratio called the Redo NoWait%.
- The section headed *Instance Activity Stats* contains the same statistical information found in the V$SYSSTAT dynamic performance view.

Improving Redo Log Buffer Performance

Make It Bigger

Use the static parameter LOG_BUFFER to set size of redo buffer. The default value is 512KB, or 128KB × the number of CPUs in the server, whichever is greater.

Note: The LOG_BUFFER parameter value must be a multiple of the operating system block size.

Reduce Redo Generation

Using UNRECOVERABLE Keyword

```
CREATE TABLE employee_history
AS SELECT ..
UNRECOVERABLE
```

Tables created in this manner do not generate any redo information for the inserts generated by the CREATE statement's sub-query.

Note: The UNRECOVERABLE/RECOVERABLE keywords cannot be used when creating partitioned tables, index-organized tables, or tables containing Large Objects (LOBS).

Note: The UNRECOVERABLE/RECOVERABLE keywords are deprecated and have been replaced with LOGGING and NOLOGGING, respectively. Oracle recommends that you use the LOGGING and NOLOGGING keywords.

Using NOLOGGING Keyword

LOGGING|NOLOGGING option affect all DML made against the table of the following types:

- Direct Path loads using SQL*Loader
- Direct load inserts using the /*+ APPEND */ hint

Also those keywords let you specify whether creation of a database object will be logged in the redo log.

This option is supported in the following statements:

- CREATE TABLE
- CREATE TABLE AS SELECT
- ALTER TABLE
- CREATE INDEX
- ALTER INDEX REBUILD
- CREATE TABLESPACE

Note: Specifying the NOLOGGING attribute on a table means that any data that was loaded using the direct path methods will be lost if media recovery is required before the new data that was added has been included in a scheduled cold or hot database backup.

Measuring the Performance of Checkpoints

Using V$SYSTEM_EVENT

waits to look for in V$SYSTEM_EVENT are:

Checkpoint Completed This event shows how often waits occurred for the checkpoint to complete its activities. High or steadily increasing values for this event indicate that checkpoints need to be tuned.

Log File Switch (Checkpoint Incomplete) This event shows how often the Online Redo Log switched from one log to the next, before the checkpoint from the previous log switch had time to complete. When this occurs, the in-progress checkpoint is abandoned and a new checkpoint is begun. Because incomplete checkpoints cause excess I/O that do not provide any recovery benefits, frequent occurrences of this event indicate that checkpoint activity should be tuned.

Using V$SYSSTAT

Two statistics should be considered: background checkpoints started and background checkpoints completed. Discrepancies between these two statistics indicate that checkpoints are starting, but not completing. Any time the difference between these two values is greater than 1, checkpoint tuning should be considered.

Note: The number of background checkpoints started and the number of background checkpoints completed will occasionally differ by one checkpoint if the query is executed when a checkpoint is in progress.

Using STATSPACK

Checkpoint performance statistics are shown in STATSPACK report in the section *Instance Activity Stats*.

Using the Alert Log

"Checkpoint not complete" error will be included in Alert Log whenever the redo logs are switching too fast and enough time for a checkpoint that has been started to complete normally.

Note: more verbose checkpoint information will also be recorded in the Alert log when the initialization parameter LOG_CHECKPOINTS_TO_ALERT is set to TRUE.

Improving Checkpoint Performance

Make it Bigger

One way to reduce the number of checkpoint events in the database is to increase the size of the Redo Logs.

To balance the recovery and performance implications of checkpoints, Oracle recommends sizing your Redo Logs so that they switch approximately every 20 to 30 minutes.

Setting Checkpoint-related Parameters

FAST_START_MTTR_TARGET

This parameter is useful when Service Level Agreements dictate the maximum allowable outage related to an instance failure.

This is a dynamic parameter and can take values from zero to 3,600 seconds. Zero value means disabling the functionality.

Note: This parameter is superseded by the FAST_START_IO_TARGET and LOG_CHECKPOINT_INTERVAL parameters if they are configured.

V$INSTANCE_RECOVERY
This view allows you to monitor effectiveness of FAST_START_MTTR_TARGET. Its columns are the following:

TARGET_MTTR Indicates the target Mean Time to Recover. This will usually match the value specified in the init.ora, but may be slightly higher if limited system resources will not allow the specified target to be maintained.

ESTIMATED_MTTR Indicates the estimated mean time to recover the instance if it were to fail at that moment.

Same statistics are included in *Instance Recovery Stats* section in STATSPACK output.

Measuring the Performance of Redo Logs

Using V$SYSTEM_EVENT

Events to consider are:

Log File Switch Completion This event indicates the amount of time that LGWR waited for a log switch to complete. High or increasing values for this event indicate that Redo Logs may be a performance bottleneck.

Log File Parallel Write This event indicates how long it took for LGWR to write the redo entries from the Redo Log Buffer to the Online Redo Logs. High or increasing values for this event can indicate that the Redo Logs are a performance bottleneck.

Using OS Utilities

Use OS utilities to monitor I/O on devices containing log files.

In unix, you can use sar (System Activity Reporter) and iostat utilities.

Improving Redo Log Performance

- separate redo logs from other database files.
- place redo logs on the fastest devices available

Measuring the Performance of Archiving

Filling the Archive Location

When the archive destination becomes full, database processing halts, and an error message is written to the database Alert log.

Solutions:

- Once space is made available in the archive destination, archiving will resume.
- You can also temporarily change the archive location using the following command:

ALTER SYSTEM ARCHIVE LOG ALL TO '/u04/oradata/PROD'

ARC0 Processes Causing Waits at Log Switch

LGWR keeps waiting for Archiver to archive a log file.

How to monitor the problem

- V$ARCHIVE_PROCESSES can be used to know status of Archiver: idle or busy. (see next section to details about this view)
- The log file switch (archiving needed) event appears in the V$SYSTEM_EVENT indicates how often the writing of the contents of the Redo Log Buffer to the Online Redo Log by LGWR was interrupted by a wait for the ARC0 Oracle background process to copy the contents of the Redo Log to the archive destination.

Solutions

- LGWR automatically starts new Archive processes any time it detects that the current number of Archiver processes is insufficient to keep up with the switching of the Redo Logs.

The DBA can set the parameter LOG_ARCHIVE_MAX_PROCESSES to a value between 2 and 10 (1 is the default).

Improving the Performance of Archiving

- Add more Redo Logs
- Making the existing Redo Logs bigger
- Create additional ARCn processes

- To monitor ARCn processes, use V$ARCHIVE_PROCESSES

  PROCESS Archiver Identity (e.g., ARC0, ARC1, ARC2).

  STATUS possible values are STOPPED, SCHEDULED, STARTING, ACTIVE, STOPPING, TERMINATED.

  LOG_SEQUENCE The log sequence number of the Redo Log that is currently being archived.

  STATE The current state of the ARCn process (IDLE or BUSY).

- Tune Archive Destinations

  Use LOG_ARCHIVE_DEST_n with LOG_ARCHIVE_MIN_SUCCEED_DEST, LOG_ARCHIVE_DEST_STATE_n or LOG_ARCHIVE_DUPLEX_DEST to archive in different destinations.

  specify a device with fast I/O capabilities for the location for your archived Redo Log files.

How to Monitor

The dynamic data dictionary views V$ARCHIVE_DEST and V$ARCHIVED_LOG can be used to monitor the location and status of the archive activity in each archive destination.

Tuning Disk I/O

The Objectives

The goals when tuning physical I/O are generally these:

- Minimize physical I/O by properly sizing the SGA.
- Tuning any remaining physical I/O when it is required.

Tuning Tablespace and Datafile I/O

Measuring Datafile I/O

Using V$FILESTAT, V$DATAFILE, and V$TEMPFILE
V$FILESTAT

FILE#	The Datafile's internal identifier
PHYRDS	Number of physical reads done on that Datafile
PHYWRTS	Number of physical writes done to that Datafile
AVGIOTIM	Average time, in milliseconds, spent performing I/O on that Datafile
MINIOTIM	Minimum time, in milliseconds, spent performing I/O on that Datafile
MAXIOWTM	Maximum time, in milliseconds, spent writing to that Datafile
MAXIORTM	Maximum time, in milliseconds, spent reading from that Datafile

Note: The parameter TIMED_STATISTICS must be set to TRUE.

Using STATSPACK
I/O statistics are provided in STATSPACK in the section named "*Tablespace IO Stats*"

Improving Datafile I/O

- Minimizing Non-Oracle I/O
- Using Locally Managed Tablespaces
- Balancing Datafile I/O
  - Separate tables and indexes into their own tablespaces.
  - Segregate Datafiles by Drive and Controller
  - Strip a Datafile so that it is stored across several devices.
    - Automatic stripping can be achieved by placing the datafile into a RAID disk.
    - Manual striping is accomplished by creating a tablespace made up of several datafiles, where each datafile is placed on a separate physical device. Use ALTER TABLE .. ALLOCATE EXTENT (DATAFILE '..' SIZE nM) statement to strip a table across multiple datafiles.
    - The parameter DB_FILE_MULTIBLOCK_READ_COUNT determines the maximum number of database blocks that are read in one I/O operation by a user's Server Process whenever a full table scan read operation is performed. Its default value is 8.
      Setting this parameter to a value greater than 8 may increase efficiency of full table scans.

Tuning DBWR Performance

Measuring DBWR I/O

Using V$SYSTEM_EVENT
Occurrences of the following events in V$SYSTEM_EVENT are possible indicators of I/O performance problems related to DBWR:

- buffer busy waits indicates that waits are being experienced for buffers in the Database Buffer Cache.
- db file parallel write indicates DBWR may be experiencing waits when writing many blocks in parallel.
- free buffer waits indicates that user Server Processes have been experiencing waits when moving dirty buffers to the Dirty List while searching the LRU for a free buffer.
- write complete waits indicates that user sessions have been experiencing waits for buffers to be written from the Database Buffer Cache by DBWR.

More details are in section *Free Buffer Waits*.

Using V$SYSSTAT

The following statistics figures in V$SYSSTAT are also useful for measuring the performance of DBWR:

- o redo log space requests indicates that a wait occurred for a redo log to become available following a log switch. If DBW0 is not writing the contents of the Database Buffer Cache fast enough, redo log space requests may result.
- o DBWR buffers scanned indicates the total number of buffers in the Database Buffer Cache that were examined in order to find dirty buffers to write.
- o Dividing DBWR buffers scanned by the value for DBWR lru scans will yield the average number of buffers examined with each LRU scan. High or steadily increasing values for the Average Number of Buffers Scanned may indicate that DBW0 is not performing its write activities efficiently.

Using STATSPACK

Same statistics can be found in the STATSPACK output in the section named *Instance Activity Stats*.

Improving DBW0 I/O

Two parameters can be used to tune the performance of DBWR's activities:

- **DBWR_IO_SLAVES**
  specifies the number of Database Writer slave processes to start at instance startup. Its default is zero. Database Writer slave processes can only perform write operations, not move buffers from the LRU List to the Dirty List in the Database Buffer Cache. The DBWR continues to perform all of the DBWR-related work, apart from performing I/O.

  The purpose of these slaves is to simulate asynchronous I/O on systems that only support synchronous I/O.

  Note: Any time Database Writer spawns I/O slave processes, other background process that perform disk I/O (i.e., Log Writer, Archiver, and the Recovery Manager utility) will also have I/O slaves created to support their I/O activities.

  If your server OS natively supports asynchronous I/O, you can set the parameter DISK_ASYNCH_IO to TRUE (the default) causes Oracle to use the OS's asynchronous I/O functionality.
- **DB_WRITER_PROCESSES**
  This parameter indicates number of DBWR processes to start. Its maximum value is 10.

Multiple DBWR or multiple slaves

- Multiple Database Writer processes generally have better throughput than Database Writer I/O slaves.
- If the Buffer Cache wait events are strictly related to DBW0 I/O activity, then starting I/O slaves is appropriate.
- If the Buffer cache wait events are occurring for activities related to internal Buffer Cache management (i.e., free buffer waits events), then starting multiple Database Writer processes to handle those structures would be more suitable.

Note: If you specify a non-zero value for DBWR_IO_SLAVES, then the value for DB_WRITER_PROCESSES has no effect.

Tuning Segment I/O

The overriding goal when tuning Segment I/O is to minimize the number of blocks that must be accessed in order to retrieve requested application data.

Improving Segment I/O

- Use dynamic extent allocation tablespaces
- Sizing extents: large extents provide better performance. The drawback is the possibility that there may not be a large enough set of contiguous Oracle blocks available when an extent is required.
- Sizing blocks: OLTP system tend to use small block sizes whereas DSS systems tend to use large block sizes. Block size also affects row chaining and migration (See next section).
- Empty blocks above High Water Mark HWM for static segments.
  If no more rows will be added to a segment, release blocks above HWM to the tablespace.

  Identifying size of blocks above HWM

  - o After using the ANALYZE command, the EMPTY_BLOCKS column in DBA_TABLES will show the number of blocks that a table has above its HWM.
  - o The unused space above the HWM can also be determined using the DBMS_SPACE.UNUSED_SPACE procedure.

To release blocks above HWM

ALTER TABLE tablename DEALLOCATE UNUSED

- In full table scan FTS, empty blocks below the HWM are also scanned. Move the HWM to the appropriate level.

To estimate number of empty blocks below the HWM

```
SELECT blocks "Total blocks",
(t.avg_row_len * t.num_rows)/s.block_size "Blocks needed below HWM",
blocks - ((t.avg_row_len * t.num_rows)/s.block_size) "Wasted blocks below HWM"
FROM dba_tables.t, dba_tablespaces s
WHERE t.tablespace_name = s.tablespace_name
AND t.table_name=..
```

To move a table's HWM to the appropriate level

- o Export, drop or truncate, and then re-import the table.
- o Use the ALTER TABLE .. MOVE command to rebuild the table.

Chaining and Migration of Oracle blocks

A chaining Row

A chaining row is spill over into two or more blocks. The only way to fix a chained row is to either decrease the size of the insert or increase the Oracle block size.

Rows Migration

If the update to the row causes the row to grow larger than the space available in the block specified by PCTFREE, Oracle moves (or migrates) the row to a new block.

Row migration can be minimized by setting PCTFREE to an appropriate value.

To identify row chaining and migration

- The ANALYZE command populates the CHAIN_CNT column of DBA_TABLES with the total number of rows using more than one block to store data: chaining plus migrating rows.
- Query V$SYSSTAT for occurrences of the statistic table fetch continued row.
- The same statistics table fetch continued row will be found in the STATSPACK report output.

Resolving row migration

- Export, drop or truncate, and then re-import the table
- Use the ALTER TABLE ... MOVE command to rebuild the table
- Identify and then reinsert the migrated rows as follows:
  1. ANALYZE the table containing migrated rows.
  2. Use the Oracle-supplied script UTLCHAIN.SQL to build a table called CHAINED_ROWS.
  3. Issue the following ANALYZE statement
     ANALYZE TABLE sales LIST CHAINED ROWS
  4. Copy the chained rows to a temporary table so that you can subsequently delete them from the original table, and then reinsert them into it.
  5. Reanalyze the table with the COMPUTE STATISTICS option. Any non-zero value remaining in the CHAIN_CNT column of DBA_TABLES indicates that those rows are chained, not migrated.

Tuning Sort I/O

Tuning goals:

- minimize sort activity
- perform sorting in memory (not in harddisk) whenever possible.

SQL operations causing sorts

- ORDER BY
- GROUP BY
- SELECT DISTINCT
- UNION
- INTERSECT
- MINUS
- ANALYZE
- CREATE INDEX
- Joins between tables on columns that are not indexed

Parameters affecting sorting

SORT_AREA_SIZE

specifies in bytes the maximum amount of memory Oracle will use for a sort.

After the sort is complete, but before the rows are returned, Oracle releases all of the memory allocated for the sort, except the amount specified by the SORT_AREA_RETAINED_SIZE parameter. After the last row is returned, Oracle releases the remainder of the memory.

Parameter class Dynamic: ALTER SESSION, ALTER SYSTEM .. DEFERRED

Note: Oracle does not recommend using the SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE parameter unless the instance is configured with the shared server option. Oracle recommends that you enable automatic sizing of SQL working areas by setting PGA_AGGREGATE_TARGET instead.

Note: When a parallel query is used to perform a sort, the total amount of sorting memory will equal to SORT_AREA_SIZE * 2 * degree of parallelism.

SORT_AREA_RETAINED_SIZE

This parameter specifies (in bytes) the maximum amount of the user global area (UGA) memory retained after a sort run completes. This memory is released back to the UGA, not to the operating system, after the last row is fetched from the sort space.

Parameter class Dynamic: ALTER SESSION, ALTER SYSTEM .. DEFERRED

Range of values: From the value equivalent of two database blocks to the value of SORT_AREA_SIZE

PGA_AGGREGATE_TARGET

This parameter specifies the target aggregate PGA memory available to all server processes attached to the instance.

You must set this parameter to enable the automatic sizing of SQL working areas used by memory-intensive SQL operators such as sort, group-by, hash-join, bitmap merge, and bitmap create.

Syntax: integer [K | M | G]

Default value: 0

Parameter class: Dynamic: ALTER SYSTEM

Range of values: 10 MB to 4096 GB – 1

WORKAREA_SIZE_POLICY

This parameter is used to determine whether the overall amount of memory assigned to all user processes is managed explicitly or implicitly.

When set to MANUAL, the size of each user's sort area will be equivalent to the value of SORT_AREA_SIZE for all users. When set to a value of AUTO, Oracle will automatically manage the overall memory allocations so that they do not exceed the target specified by PGA_AGGREGATE_TARGET.

Default value: If PGA_AGGREGATE_TARGET is set, then AUTO

If PGA_AGGREGATE_TARGET is not set, then MANUAL

Parameter class Dynamic: ALTER SESSION, ALTER SYSTEM

Measuring Sort I/O

Using V$SYSSTAT

The V$SYSSTAT dynamic performance view has two statistics, sorts (memory) and sorts (disk), that can be used to monitor sort user activity.

Oracle's recommended tuning goal with regard to sorting is to have at least **95 percent** of all sorts happen in memory.

Using STATSPACK

In the *Instance Efficiency Percentages* section, you will see percentage of sorts occurred in memory.

In the *Instance Activity Stats* section, you will see number of sorts occurred in memory and disk. Also you will see number of rows that were sorted.

In the *Load Profile* section, you will see the average of number of sorts per second and average of sorts per transaction.

Improving Sort I/O

- **Avoid Sorts**
  - SQL Statement Syntax
    - Use UNION ALL operator instead of the UNION operator.
    - Avoid the use of the INTERSECT, MINUS, and DISTINCT keywords.
  - Appropriate Use of Indexes
    - Index columns that are referenced by ORDER BY and GROUP BY clauses in application SQL statements.
    - Make sure that all foreign key columns are indexed to reduce sorting when tables are joined via primary and foreign key columns.
  - Index Creation Overhead
    - When possible, use NOSORT option of the CREATE INDEX statement.
  - Statistics Calculation Overhead

- Consider using the ESTIMATE option instead COMPUTE when gathering table and index statistics.

- **Make It Bigger**

Large value of SORT_AREA_SIZE improves sorts performance. On the other hand, when high number of users are concurrently running sort operations, you will risk excessive usage of server memory.

- **Managing Temporary Tablespaces**
  - Use a default temporary tablespace for the database.
  - Strip the tempfile on multiple devices can provide excellent performance improvements.
  - Using V$SORT_SEGMENT
    The V$SORT_SEGMENT view allows you to monitor the size and growth of the sort segment that resides in the temporary tablespace. MAX_SORT_BLOCKS column can be used to set the optimized value for SORT_AREA_SIZE

```
SELECT tablespace_name, current_users,
max_sort_blocks
FROM v$sort_segment
```

  - Using V$SORT_USAGE
    V$SORT_SEGMENT does not allow you to see which individual user are causing large sorts to disk. The V$SORT_USAGE view does include a USER column which can be used to monitor sort activity by user. However the V$SORT_USAGE view only contains entries when at least one sort is in progress.
  - Identifying SQL statement causing high sorts

```
SELECT sess.username,
sql.sql_text,
sort.blocks
FROM v$session sess,
v$sqltext sql,
v$sort_usage sort
WHERE sess.serial# = sort.session_num
AND sort.sqladdr = sql.address
AND sort.sqlhash = sql.hash_value
AND sort.blocks > 200
```

Tuning Rollback Segment I/O

The tuning goals are:

- Making sure that database users *always* find a rollback segment to store their transaction before-images without experiencing a wait.
- No transaction, however large or exceptional, should ever run out of rollback space.
- Making sure that database users always get the read-consistent view they need to complete their transactions.
- Making sure that the database rollback segments do not cause unnecessary I/O.

Measuring Rollback Segment I/O

Measuring Rollback Segment Header Contention

Users may experience a wait for access to rollback segment header blocks in the buffer cache.

Using V$SYSTEM_EVENT

Search for undo segment tx slot statistic.

Ideally, the value of the AVERAGE_WAIT statistic should be consistently at or near zero.

Using V$WAITSTAT

This view lists block contention statistics. This table is only updated when timed statistics are enabled.

```
SELECT class, count
FROM v$waitstat
WHERE class IN ('undo header','system undo header')
```

Ideally, the counts for these waits should be consistently at or near zero.

Using V$ROLLSTAT

The V$ROLLSTAT view contains detailed information regarding the behavior of the rollback segments in the database.

USN	Undo Segment Number, the internal system-generated number assigned to the rollback segment
GETS	Number of times a user's Server Process needed to access the rollback segment header and did so successfully

WAITS Number of times a user Server Process needed to access the rollback segment header and experienced a wait

```
SELECT n.name, s.usn,
DECODE (s.waits,0,1,1-(s.waits/s.gets))
 "RBS Header Get Ratio"
FROM  v$rollstat s, v$rollname n
WHERE s.usn = n.usn
ORDER BY usn
```

Oracle recommends that a well-tuned OLTP system have a rollback segment header get ratio of **95 percent** or higher for all rollback segments**.**

Using STATSPACK
Rollback statistics in STATSPACK output can be found in *Rollback Segment Stats*.

Rollback Segment Extent Contention
The following query shows how to determine whether waits for access to the blocks within the individual extents of each rollback segment are occurring in the database:

```
SELECT class, count
FROM v$waitstat
WHERE class IN ('undo block','system undo block')
```

Divide the total number of requests (using the following select statement) by the value obtained from the select above to calculate wait ratio for rollback segment extent blocks.

```
SELECT value
FROM v$sysstat
WHERE name = 'consistent gets'
```

If the wait ratio for rollback segment extent blocks exceeds 1 percent, Oracle recommends that you consider tuning your rollback segments.

Rollback Segment Transaction Wrapping
When a transaction's before-image exceeds the size of the extent to which it has been allocated, the transaction will wrap into the next extent if it is available.
It is better to avoid, if possible, occurrence of dynamic rollback extent wrapping.

Using V$ROLLSTAT to Measure Rollback Segment Wrapping
To see how often transactions have wrapped from one extent to another since instance startup:

```
SELECT n.name, s.usn, s.wraps
FROM v$rollname n, v$rollstat s
WHERE n.usn = s.usn
```

Frequent wrapping indicates that the extent sizes of each rollback segment may be too small.

Using STATSPACK to Measure Rollback Segment Wrapping
Consider the Rollback Segment Stats section.

Dynamic Extent Allocation of Rollback Segments
When a transaction's before-image exceeds the size of the extent to which it has been allocated, but cannot wrap to the next extent because there are still active transactions in that extent, the rollback segment will add a new extent and wrap into that extent instead.

Using V$SYSTEM_EVENT to Measure Dynamic Undo Segment Extent Allocation
The statistic undo segment extension in the V$SYSTEM_EVENT dynamic performance view records how long user Server Processes had to wait for rollback segments to add extents to handle transaction processing:

```
SELECT event, total_waits, time_waited, average_wait
FROM v$system_event
WHERE event like '%undo%'
```

High or steadily increasing values for these wait statistics indicate that you either have too few rollback segments, too small rollback segments, or both.

Using V$ROLLSTAT to Measure Dynamic Undo Segment Extent Allocation
The V$ROLLSTAT view contains a column called EXTENDS. This column indicates how often the rollback segment was forced to add extents to support database transaction activity:

```
SELECT n.name, s.usn, s.extends
FROM v$rollname n, v$rollstat s
WHERE n.usn = s.usn
```

Using STATSPACK to Measure Dynamic Undo Segment Extent Allocation
The statistics can be found in the *Rollback Segment Stats* section.

Improving Rollback Segment I/O

- **Add More Undo Segments**
- **Make Undo Segments Bigger**

- Using V$TRANSACTION and V$SESSION to Monitor Rollback Segment Usage

```
SELECT s.osuser, s.username, t.used_ublk
FROM v$session s, v$transaction t
WHERE s.taddr = t.addr;
```

- **Explicitly Managing Rollback Segments**

  Very large transactions, like those associated with batch processing runs, frequently require large rollback segments. You can dedicate one or two large rollback segments for those transactions.

  1. `CREATE PRIVATE ROLLBACK SEGMENT rbsbatch`
     `STORAGE (INITIAL 10M NEXT 10M)`
     `TABLESPACE rbs`
  2. `ALTER ROLLBACK SEGMENT rbsbatch ONLINE`
  3. `SET TRANSACTION USE ROLLBACK SEGMENT rbsbatch`
     or
     `EXECUTE DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT('rbsbatch')`
  4. Now, start the batch processing job.
  5. Next, from a database session other than the one that is executing the batch job, immediately take the large rollback segment offline. The rollback segment will not actually go offline until the batch transaction is complete.
  6. `ALTER ROLLBACK SEGMENT rbsbatch OFFLINE`

  Note: Any commit that occurs within the batch job will cause the large rollback segment to go offline and become unavailable. Therefore, you must bring the rollback segment back online and reissue the `SET TRANSACTION` or `DBMS_TRANSACTION` command after each commit to keep the batch job from using some other rollback segment.

- **Minimizing Rollback Segment Activity**

  Minimize the number and size of entries that are written to the rollback segments. This goal can be accomplished by:

  - Using the COMMIT=Y option when performing database imports
  - Forgoing the use of the CONSISTENT option when using the database EXPORT utility
  - setting an appropriate COMMIT value when using the SQL*Loader utility.

- **Using Automatic Undo Management Features**

  Undo tablespaces dynamically manage the size and number of undo segments. `UNDO_MANAGEMENT` and `UNDO_TABLESPACE` parameters are used to enable using Undo tablespace in a database.

  Other related parameters are the following:

  - UNDO_RETENTION specifies (in seconds) how long to retain undo information after the transaction that generated it, commits. The default value is 900 seconds. This parameter is designed to help minimize Snapshot Too Old errors caused when a transaction needs overwritten before-image data for read consistency.
  - UNDO_SUPPRESS_ERRORS (TRUE|FALSE) used to suppress errors that would otherwise occur when commands that are appropriate in RBU mode (e.g., SET TRANSACTION USE ROLLBACK SEGMENT ...) are used in AUM (Automatic Undo Management) mode.
  - You can switch the database active undo tablespace using the following command:

    ALTER SYSTEM SET UNDO_TABLESPACE=undo
  - Transactions in the old tablespace keep using the same tablespace until those transactions complete and the UNDO_RETENTION period has passed.

Monitoring the Performance of System-Managed Undo

Using DBA_TABLESPACE

This view can be used to list undo tablespaces created in the database. The column CONTENTS equals to UNDO.

USING V$UNDOSTAT

This view contains historical statistics about to monitor the usage of the system-managed undo tablespace, presented as a histogram.

Each row in the V$UNDOSTAT view stores undo statistics from a previous 10-minute interval.

BEGIN_TIME	Time and date when the undo statistics monitoring period began.
END_TIME	Time and date when the undo statistics monitoring period ended.
UNDOTSN	The ID of the tablespace where the undo activity took place. Same as TS# in V$TABLESPACE
UNDOBLKS	The number of undo blocks used to store before-image data between the begin and end time.

TXNCOUNT	The total number of transactions executed between the begin and end time.
MAXQUERYLEN	The time (in seconds) that the longest query took to execute between the begin and end time. Use this value to determine the appropriate value for UNDO_RETENTION.
UNXPBLKREUCNT	The number of undo blocks that were needed to maintain read consistency for another transaction, but which were reused to hold another transaction's before-image instead. Use this value to measure the likelihood of Snapshot Too Old errors occurring in the database.

Tuning Contention

Latch Contention

A latch is a specialized type of lock that is used to serialize access to a particular memory structure or serialize the execution of kernel code. Using latches, Oracle makes sure that no two processes are accessing the same data structure simultaneously.
Types of latches: *Willing-to-Wait latch* and *Immediate (no-Wait) latch*.

V$LATCH is used to monitor the activity latches.

NAME	The name of the latch
GETS	The number of times a Willing-to-Wait latch was acquired without waiting
MISSES	The number of times a Willing-to-Wait latch was not acquired and a wait resulted
SLEEPS	The number of times a process had to wait before obtaining a Willing-to-Wait latch
IMMEDIATE_ GETS	The number of times an Immediate latch was acquired without waiting
IMMEDIATE_ MISSES	The number of times an Immediate latch was not acquired and a retry resulted

Measuring Latch Contention

To determine if latch contention is even occurring in your instance, search for latch free event in V$SYSTEM_EVENT

Important Latch Types
From a tuning perspective, the following latches are important:

- Shared Pool Latch
- Library Cache Latch
- Cache Buffers LRU Chain Latch
  Frequent waits for the `Cache Buffer LRU Chain` latch can be caused by two possible sources:
  - Inefficient application SQL that results in excessive full table scans or inefficient execution plans.
  - Database Writer is unable to keep up with write requests. See “Tuning Disk I/O,” section
- Cache Buffers Chains Latch
- Redo Allocation Latch
- Redo Copy Latch

Using V$LATCH to Identify Latch Contention

```
SELECT name "Willing to Wait Latch",
gets, misses, wait_time
FROM v$latch WHERE misses != 0
```

Using STATSPACK Output to Identify Latch Contention

- A section entitled *Top 5 Wait Events* indicates whether latch free event waits have been occurring frequently in the instance.
- To specifically determine which latches are experiencing the waits, examine *Latch Activity* section.

Tuning Latch Contention

DBAs uses evidence of latch contention as an indicator of possible areas for tuning improvement in the database’s other structures, such as the SGA. Once identified, the structure related to the latch contention is then tuned, not the latch itself.

Free List Contention

If your application has many users performing frequent inserts, the application user’s Server Process may experience waits when trying to access the Free List for a frequently inserted table.

Measuring Free List Contention

Using V$SYSTEM_EVENT
Search for the event buffer busy wait in V$SYSTEM_EVENT.

However, Finding the buffer busy waits in your V$SYSTEM_EVENT view does not in itself indicate a problem with Free List contention. The V$WAITSTAT view should also be examined to make sure that Free List contention is occurring during processing.

Using V$WAITSTAT

Non-zero wait statistics for two classes of blocks in particular, free list and segment header, indicate that Free List contention is occurring in the database.

Once you determine that Free List contention is present, you must next determine which segments are experiencing the contention using V$SESSION_WAIT and DBA_SEGMENTS.

Using V$SESSION_WAIT and DBA_SEGMENTS

```
SELECT s.segment_name, s.segment_type, s.freelists
FROM dba_segments s, v$session_wait w
WHERE w.p1 = s.header_file
AND w.p2 = s.header_block
AND w.event = 'buffer busy waits'
```

Tuning Free List Contention

- Adding Additional Free Lists

```
ALTER TABLE hr.employee
STORAGE (FREELISTS 5)
```

- Moving Segments to Automatic Segment-space Managed Tablespaces

```
ALTER TABLE hr.employee MOVE TABLESPACE appl_data
```

Lock Contention

Oracle's locking mechanisms are similar to the latching mechanisms, but locks are used to protect access to data, not memory structures.

Locks are also less restrictive than latches. In some cases, several users can share a lock on a segment. This is not possible with latches, which are never shared between processes.

When two users try modifying the same row in the same segment, at the same time, lock contention results.

Table-level or Row-level Locking

This can be controlled using the parameter ROW_LOCKING. This parameter takes value of INTENT (table-level) or ALWAYS (row-level) which is the default.

Lock Levels

Lock Types

- DML or data lock
- DDL or Dictionary Lock: used to lock tables when users are creating, altering, or dropping tables.

Lock Modes

- Exclusive: locks a resource until the transaction that took out vfthe lock is complete.
- Share: locks the resource, but allows other users to also obtain additional share locks on the same resource.

Lock Kinds

- Implicit: Oracle performs the locking actions for the user.
- Explicit: lock you manually request a specified lock type.

DML Locks

Kind of Lock	Mode (Symbol)	Command	Description
Implicit	Row Exclusive (RX)	`INSERT`, `UPDATE`, `DELETE`	Other users can still perform DML on any other row in the table.
Implicit	Table Row Share (RS)	`SELECT ... FOR UPDATE`	Other users can still perform DML on the rows in the table that were not returned by the SELECT statement.
Implicit	Share (S)	`UPDATE` and `DELETE` on parent tables with Foreign Key relationships	Users can still perform DML on any other row in either the parent or child table as long as an index exists on the child table's Foreign Key column.

		to child tables	
Implicit	Share Row Exclusive (SRX)	DELETE on parent tables with Foreign Key relationships to child tables	Users can still perform DML on any other row in either the parent or child table as long as an index exists on the child table's Foreign Key column.*
Explicit	Exclusive (X)	LOCK TABLE ... IN EXCLUSIVE MODE	Other users can only query the table until the locking transaction is either committed or rolled back.

* In versions prior to 9i, Oracle will take out a more restrictive lock on the child table whenever appropriate indexes on the Foreign Key columns of a child table are not present.

DDL Locks

Mode (Symbol)	Command	Description
Exclusive (X)	CREATE, DROP, ALTER	Prevents other users from issuing DML or SELECT statements against the referenced table until after the CREATE, DROP, or ALTER operation is complete
Shared (S)	CREATE PROCEDURE, AUDIT	Prevents other users from altering or dropping the referenced table until after the CREATE PROCEDURE or AUDIT operation is complete
Breakable Parse	-	Used by statements cached in the Shared Pool; never prevents any other types of DML, DDL, or SELECT by any user

Note: All lock modes above are implicit.

Measuring Lock Contention

Using V$LOCK

The V$LOCK dynamic performance view contains data regarding the locks that are being held in the database at the time a query is issued against the view.

SID	Identifier for session holding or acquiring the lock
TYPE	Type of user or system lock. The user type locks are: TM - DML enqueue TX - Transaction enqueue UL - User supplied The locks on the system types are held for extremely short periods of time.
ID1, ID2	Lock identifier #1 and #2 (depends on type)
LMODE	Lock mode in which the session holds the lock: 0 - none 1 - null (NULL) 2 - row-S (SS) 3 - row-X (SX) 4 - share (S) 5 - S/Row-X (SSX) 6 - exclusive (X)
REQUEST	Lock mode in which the process requests the lock. (same list as above)

To see who is holding the lock and on which objects:

```
SELECT s.username,
DECODE(l.type,'TM','TABLE LOCK',
'TX','ROW LOCK', NULL) "LOCK LEVEL",
o.owner, o.object_name, o.object_type
FROM v$session s, v$lock l, dba_objects o
WHERE s.sid = l.sid
AND o.object_id = l.id1
AND s.username IS NOT NULL
```

However, this query does not explicitly indicate which user is blocking another user.

Using V$LOCKED_OBJECT
The V$LOCKED_OBJECT view also lists all the locks currently held by every user on the system. However, it also includes blocking information showing which user is performing the locking transaction that is causing other application users to experience a wait.

```
SELECT LPAD(' ',DECODE(l.xidusn,0,3,0))
||l.oracle_username "User Name",
o.owner, o.object_name, o.object_type
FROM v$locked_object l, dba_objects o
WHERE l.object_id = o.object_id
ORDER by o.object_id, 1 desc
```

Using DBA_WAITERS
The DBA_WAITERS view contains information about user sessions that are waiting for locks to be released by other user sessions.

WAITING_SESSION	The SID of the user session that is currently waiting to obtain a lock
HOLDING SESSION	The SID of the user session that currently holds the lock that the user shown in WAITING_SESSION wishes to obtain
LOCK_TYPE	The type of lock held by the user associated with the HOLDING_SESSION
MODE_HELD	The lock mode held by the user associated with the HOLDING_SESSION
MODE_REQUESTED	The lock mode requested by the user associated with the WAITING_SESSION
LOCK_ID1	Internal lock identifier for the first (usually Share) lock being held by user associated with the HOLDING_SESSION
LOCK_ID2	Internal lock identifier for the second (usually Row Exclusive) lock being held by the user associated with the HOLDING_SESSION

```
SELECT sw.username "WAITING USER",
 bu.username "LOCKING USER",
 dw.lock_type, dw.mode_held, dw.mode_requested
FROM dba_waiters dw, v$session sw, v$session bu
WHERE dw.waiting_session = sw.sid
  AND dw.holding_session = bu.sid
```

Using DBA_BLOCKERS
DBA_BLOCKERS view contains only one column, HOLDING_SESSION, which displays the SIDs of the user sessions that are blocking the lock requests of other application users.

```
SELECT s.sid, s.username
FROM dba_blockers db, v$session s
WHERE db.holding_session = s.sid
```

Note: The DBA_BLOCKERS and DBA_WAITERS data dictionary views are created by executing the CATBLOCK.SQL script.

Note: you can also use the Oracle-supplied script, UTLLOCKT.SQL to display locking information.

Tuning Lock Contention

- Change the application code so less-restrictive locking is used
- Contact the blocking user and ask them to commit or roll back their changes
- Resolve the lock contention using an SQL command

```
ALTER SYSTEM KILL SESSION '12,8881'
```

Operating System Tuning

Resources to Tune

- Memory
- Disk I/O
- CPU

SGA Locking

- Setting the parameter LOCK_SGA=TRUE will prevent the OS from paging or swapping any portion of the SGA by locking the entire SGA in physical memory.
- This parameter is not available on Windows 2000 servers.

Different System Architectures

Uniprocessor	Servers with a single CPU and memory area. I/O, bus, and disk resources are used exclusively by this server.
Symmetric Multiprocessor (SMP)	Servers with two or more CPUs, but one memory area. I/O, bus, and disk resources are used exclusively by this server.
Massively Parallel Processor (MPP)	Two or more servers connected together. Each server has its own CPU, memory area, I/O bus, disk drives, and a copy of the OS.
Non-Uniform Memory Access (NUMA)	Two or more SMP systems connected together. Each server has its own CPU, but the memory areas for each server is shared as one large memory area across all servers. A single copy of the OS is run across all servers.

CPU-related Initialization Parameters

If new CPUs are added to a database server, Oracle will dynamically change the default settings for several CPU-related parameters.

`CPU_COUNT`

Specifies the number of CPUs in the server.

`PARALLEL_THREADS_PER_CPU`

Used to establish the default degree of parallelism when using Oracle's parallel execution features.

`FAST_START_PARALLEL_ROLLBACK`

The number of parallel rollback processes is derived from this value (`HIGH`, `LOW`, `FALSE`) and the number of CPUs.

`LOG_BUFFER`

Maximum size is 512K or 128K × the number of CPUs, whichever is higher.

`PARALLEL_MAX_SERVERS`

specifies the maximum number of parallel execution processes and parallel recovery processes for an instance.

If you set this parameter too low, some queries may not have a parallel execution process available to them during query processing. If you set it too high, memory resource shortages may occur during peak periods, which can degrade performance.

Oracle's Parallel Execution Options

- **Parallel Query** When configured, Oracle's Parallel Query feature allows a single query to spawn multiple query slaves. Each of these query slaves works on a portion of the query before the individual results are combined into a final result set.
- **Parallel DML** Some insert, update, and delete activities can be performed in parallel using Oracle's Parallel DML (PDML) features.
- **Parallel ANALYZE** Table and index statistics can be gathered in parallel if the DBMS_STATS package is being used.
- **Parallel Index Creation** Indexes that are being created on large tables can be created in parallel if the PARALLEL keyword is included in the CREATE INDEX command.

Understanding Resource Manager

Resource Manager Components

Resource consumer group
A group of users who have similar needs for database resources and who are governed by similar rules for allocating resources to those users. A user can be a member of several resource groups, but only one group can be active at a time.

Resource plan
A collection of resource plan directives that are used to determine how resources are allocated to users.

Resource plan directive
Specifies how many resources are allocated to each resource plan.

Resource Manager Resources

- The **amount of CPU** allocated to each user who is a member of the resource consumer group
- The **degree of parallelism** allowed for Parallel Queries performed by a user who is a member of the resource consumer group
- The **amount of undo segment space** that a user who is a member of a resource group is allowed to consume when performing a long running transaction
- The **total number of active, concurrent sessions** that a given resource consumer group is allowed to have at one time
- The **maximum expected total time of a database action** taken by a user who is a member of the resource consumer group

Managing CPU Resources

- The CPU resources are allocated using the ***Emphasis method***.
- The Emphasis method uses percentages, expressed in terms of a value between 0 and 100, to assign CPU resources to each of the eight priority levels.
- The percentage assigned to a resource consumer group at a given level determines the maximum amount of CPU that users in that consumer group can consume at that level. If excess CPU resources remain after all users at that level have been serviced, then the remaining CPU resources will be made available to users at the next level, and so on.

How the Emphasis Method Manages Parallel Query Resources

- Resource manager uses the Absolute method to control Parallel Query resources. This means that the maximum degree of parallelism is a specific (i.e., absolute) value.

Note: The CPU and Parallel Query resource limits do not affect processing unless these resources are scarce at the system level. For example, if excess CPU resources exist on your server, then CPU resource management will have no effect. This would allow a process assigned little or no resources via a PlanDirective to be able to acquire as many CPU resources as it needs—even more than the amount specified in its assigned directive.

Managing Undo Resources

Oracle9i introduces a new Plan Directive called the UNDO_PLAN. Using this Plan Directive you can control the amount of undo segment space a user is allowed to consume when executing a large DML transaction.

Managing Workload Resources

- Through a mechanism called the ***Active Session Pool***, Oracle9i allows you to limit the number of active concurrent sessions for a particular resource group.
- Once the members of the resource group reach the number of allowed active concurrent sessions, any subsequent requests for active sessions will be queued by Resource Manager instead of being processed immediately.
- Non-zero values in the CURRENT_QUEUE_DURATION column of the V$SESSION dynamic performance view indicate that a session is currently queued by Resource Manager.
- The maximum number of sessions is specified using the ACTIVE_SESS_POOL_P1 Resource Manager parameter. The default is 1,000,000.
- The maximum time, in seconds, that a session will be queued before aborting is specified using the QUEUEING_P1 Resource Manager parameter. The default is 1,000,000.

Managing Execution Duration

- Using the new Oracle9i resource plan directive called MAX_ESTIMATED_EXEC_TIME, you can specify the maximum length of time that a database operation can take to complete.
- Once set, Resource Manager will use the statistics in the CBO to generate an estimated completion time for each database operation. This estimate is then compared to this specified maximum allowed time. If the operation is going to take longer than the specified duration, then the operation will not be started.

Configuring Resource Management

I. Granting Resource Manager Privileges

- Using an Oracle-supplied PL/SQL package called DBMS_RESOURCE_MANAGER_PRIVS grant ADMINISTER_RESOURCE_MANAGER privilege to users who are going to be allowed to manage Resource Groups, Plans, and Directives.

  dbms_resource_manager_privs.grant_system_privilege(grantee_name => 'REGI', privilege_name => 'ADMINISTER_RESOURCE_MANAGER', admin_option => FALSE)
- To revoke the privilege:

  dbms_resource_manager_privs.revoke_system_privilege(revokee_name => 'REGI', privilege_name => 'ADMINISTER_RESOURCE_MANAGER')
- To display which database users have been granted the ADMINISTER_RESOURCE_MANAGER privilege, use the DBA_RSRC_MANAGER_SYSTEM_PRIVS view.

II. Creating the Resource Manager Objects

1. Creating the Pending Area

EXECUTE dbms_resource_manager.create_pending_area

The database can have only one pending area at a time.

2. Creating Resource Consumer Groups RCG

The following RCGs are created on database creation

`SYS_GROUP`

This group has the highest priority for resource access. The users `SYS` and `SYSTEM` are assigned to this RCG by default.

`LOW_GROUP`

This group receives the lowest priority for database resources. Every user has access to this RCG.

`DEFAULT_CONSUMER_GROUP`

Every user who is not explicitly assigned to another RCG is assigned to this RCG by default.

`OTHER_GROUPS`

The group to which all users belong when they are not members of the specific resource plan that is currently active in the instance.

To create your own RCG:

```
dbms_resource_manager.create_consumer_group(consumer_group => 'POWER_USERS', comment
=> 'User who will always receive preference for resources')
```

At this point, the new POWER_USERS resource consumer group is stored in the pending area.

3. Creating Resource Plans

```
dbms_resource_manager.create_plan( plan => 'FUNCTIONAL', comment => 'Functional power
users')
dbms_resource_manager.create_plan( plan => 'TECHNICAL', comment => 'Technical power
users')
```

Note: Three default resource plans, called `SYSTEM_PLAN`, `INTERNAL_PLAN`, and `INTERNAL_QUIESCE`, are created at database creation.

4. Creating Plan Directives

```
dbms_resource_manager.create_plan_directive( plan => 'FUNCTIONAL', group_or_subplan
=> 'POWER_USERS', comment => 'Plan directive for functional power users', cpu_p1 =>
80, parallel_degree_limit_p1 => 5)
dbms_resource_manager.create_plan_directive( plan => 'TECHNICAL', group_or_subplan =>
'POWER_USERS', comment => 'Plan directive for technical power users', cpu_p1 => 100,
parallel_degree_limit_p1=>8)
```

Each plan directive must also include a reference to the default resource consumer group `OTHER_GROUPS`:

```
dbms_resource_manager.create_plan_directive( plan => 'FUNCTIONAL', group_or_subplan
=> 'OTHER_GROUPS', comment => 'OTHER_GROUP default', cpu_p2 => 50,
parallel_degree_limit_p1 => 1)
dbms_resource_manager.create_plan_directive( plan => 'TECHNICAL', group_or_subplan =>
'OTHER_GROUPS', comment => 'OTHER_GROUP default', cpu_p2 => 50,
parallel_degree_limit_p1=>1)
```

5. Validating Consumer Resource Group, Plans, and Directives

```
dbms_resource_manager.validate_pending_area()
```

6. Saving Resource Consumer Groups, Plans, and Directives

```
dbms_resource_manager.submit_pending_area
```

7. Assigning Resources to Users

```
dbms_resource_manager_privs.grant_switch_consumer_group( grantee_name => 'BETTY',
consumer_group => 'POWER_USERS', grant_option => FALSE)
```

It is usually preferred to specify a default RCG that will be active for the user when they initially connect to the database:

```
dbms_resource_manager.set_initial_consumer_group( user=>'BETTY', consumer_group =>
'POWER_USERS')
```

If a user is not assigned a default consumer group, their initial RCG will be the public group `DEFAULT_CONSUMER_GROUP`.

8. Setting Instance Resource Objectives

- The active resource plan for the instance can be defined using the `RESOURCE_MANAGER_PLAN` initialization parameter.
- The parameter can be dynamically changed by `ALTER SYSTEM`
- If it is not set, then the features of Resource Manager are disabled.

9. Changing active RCG dynamically for a connected user

The active RCG for a user can also be changed dynamically within the application:

```
dbms_session.switch_current_consumer_group( new_consumer_group => 'POWER_USERS',
old_consumer_group =>'POWER_USERS', initial_group_on_error => FALSE)
```

10. Changing active RCG dynamically for a session

If your application connects to the database with the same database username for all application users, in this case using `switch_consumer_group_for_user` cannot work.

In this situation, you can use the `SWITCH_CONSUMER_GROUP_FOR_SESS` procedure in the `DBMS_RESOURCE` package to change the active RCG for an individual user's session.

```
dbms_resource_manager.switch_consumer_group_for_sess( session_id=>13,
session_serial=>14, consumer_group => 'POWER_USERS')
```

11. Switching consumer group dynamically

If they have been granted the privileges allowing them to switch to that particular consumer group, non-DBA users can also change their active resource consumer group by using the `SWITCH_CURRENT_CONSUMER_GROUP` procedure in the `DBMS_SESSION` package.

Determining Application User CPU Utilization

One way to determine how many CPU resources users are consuming is to compare the CPU usage for a particular user session to the CPU usage for the entire instance:

- CPU usage for a particular user session can be obtained from value column of v$session view where statistic# = 12.
- CPU usage for the entire instance can be obtained from value column of v$sysstat where name equal to 'cpu used by this session'

Monitoring Resource Manager

V$SESSION

The RESOURCE_CONSUMER_GROUP column in V$SESSION displays the current RCG for each session.

V$RSRC_CONSUMER_GROUP

This view contains details about the resource allocations that have been made to the active resource consumer groups.

Name	The name of the resource consumer group
ACTIVE_SESSIONS	The number of sessions that are currently assigned to this resource consumer group
EXECUTION_WAITERS	The number of sessions assigned to this resource consumer group that are waiting to be allocated resources before they can execute

REQUESTS	The total number of database requests that have been executed by users in this resource consumer group
CPU_WAIT_TIME	The total time that users in this resource consumer group have had to wait for access to the CPU (in 100ths of a second)
CPU_WAITS	The total number of times that users in the resource consumer group have waited for access to the CPU
CONSUMED_CPU_TIME	The total amount of CPU time that users in this resource consumer group have consumed (in 100ths of a second)
YIELDS	The number of times users in this resource consumer group have had to yield the CPU to users in other resource consumer groups with greater priority
SESSIONS_QUEUED	The number of user sessions assigned to this resource consumer group that are waiting to become active

Ideally, users with high priority in the database should be experiencing few CPU waits (EXECUTION_WAITERS and CPU_WAIT_TIME) and not have many unserviced sessions (SESSIONS_QUEUED).

V$RSRC_PLAN

This view displays the names of all the currently active resource plans.

DBA_RSRC_CONSUMER_GROUPS

This view contains information on each resource consumer group in the database.

CONSUMER_GROUP
The name of the resource consumer group.

CPU_METHOD
The CPU allocation method used to assign CPU resources to this consumer group.

COMMENTS
The comments (if any) specified for the resource consumer group at its creation.

STATUS
The status of the resource consumer group. New, uncommitted RCGs have a status of PENDING. Committed RCGs have a status of ACTIVE.

MANDATORY
Indicates whether or not the resource consumer group is mandatory. Mandatory RCGs cannot be deleted or modified.

DBA_RSRC_CONSUMER_GROUP_PRIVS

This view shows which users or roles in the database have been granted to each resource consumer group.

GRANTEE
The username or role that has been granted a resource consumer group

GRANTED_GROUP
The name of the resource consumer group the user or role has been granted

GRANT_OPTION
Indicates whether the user has the ability to pass the privilege, assigned via the resource consumer group, on to other users

INITIAL_GROUP
Indicates whether the resource consumer group is the user's default RCG

DBA_RSRC_PLANS

This view shows all the resource plans that have been created in the database.

PLAN
The name of the plan.

NUM_PLAN_DIRECTIVES
The number of directives within the plan.

CPU_METHOD
The method used to allocate CPU resources to the plan (Emphasis method).

ACTIVE_SESS_POOL_MTH
Allocation method used for Active Session Pool resources.

PARALLEL_DEGREE_LIMIT_MTH
The method used to allocate parallel resources to the plan (Absolute method).

QUEUING_MTH
Allocation method used for Queuing session resources.

COMMENTS
The comments defined at plan creation (if any).

STATUS
The status of the resource plan. New uncommitted plans have a status of PENDING. Committed plans have a status of ACTIVE.

MANDATORY
Indicates whether or not the resource plan is mandatory.

DBA_RSRC_PLAN_DIRECTIVES

This view displays the specific resource allocations that were assigned to each resource plan.

PLAN
The name of the plan to which this directive has been assigned.

CPU_Pn
The emphasis percentage assigned to CPU allocation at level n.

COMMENTS
The comments defined at plan creation (if any).

STATUS
The status of the resource plan: PENDING or ACTIVE.

MANDATORY
Indicates whether or not the resource plan is mandatory.

DBA_RSRC_MANAGER_SYSTEM_PRIVS

This view contains information about which users have been granted the ADMINISTER_RESOURCE_MANAGER privilege.

DBA_USERS

The DBA_USERS view has a single column, INITIAL_RSRC_CONSUMER_GROUP, which indicates the resource consumer group that has been assigned to a user as his default.