Oracle9*i* Database Performance Tuning

Student Guide Vol 1

D11299GC11
Production 1.1
December 2001
D34121

ORACLE®

Authors

Peter Kilpatrick
Shankar Raman
Jim Womack

Technical Contributors and Reviewers

Mirza Ahmad
Harald Van Breederode
Howard Bradley
Howard Ostrow
Alexander Hunold
Joel Goodman
John Watson
Michele Cyran
Pietro Colombo
Ranbir Singh
Ruth Baylis
Sander Rekveld
Tracy Stollberg
Connie Dialeris
Wayne Stokes
Scott Gossett
Sushil Kumar
Benoit Dagerville
David Austin
Howard Bradley
Howard Ostrow
Janet Stern
Lilian Hobbs
Maria Senise
Roderick Manalac
Sander Rekveld
Scott Gossett

Contents

1

Overview of Oracle 9*i* Performance Tuning

Objectives

After completing this lesson, you should be able to do the following:

- List the roles associated with the database tuning process
- Describe the dependencies between tuning in different development phases
- Describe service level agreements
- List the tuning goals
- List the most common tuning problems
- Describe tuning during development and production
- Describe performance and safety tradeoffs

ORACLE

Tuning Questions

- **Who tunes?**
  - **Application designers**
  - **Application developers**
  - **Database administrators**
  - **System administrators**
- **Why tune?**
- **How much tuning?**

1-3

Tuning Questions

Who Tunes?

Everyone involved with the Oracle9*i* system (system architects, designers, developers, and database administrators) should think about performance and tuning in doing his or her work.

If problems develop, it is usually the database administrators (DBA) who has to make the first attempt at solving them.

Why Tune?

The best practice of tuning is careful design of systems and applications, and the majority of performance gains are realized by tuning the application.

You are much less likely to run into performance problems if:

- The hardware can meet user demands
- Your Oracle9*i* database was carefully designed
- Your application developers write efficient SQL programs

If wrong decisions were made early, or if users expect much more from the system now than they did previously, you should seriously consider further tuning to improve performance. The database should be monitored on a regular basis to look for bottlenecks that affect performance.

Tuning Questions (continued)

How Much Tuning?

There are basically two forms of tuning:

- Speed: Short response time
- High throughput scalability: Higher load at a comparable response time or throughput.

During this course, methods to identify and resolve bottlenecks will be discussed. The result of tuning should be visible to users, either as a decrease in the time it takes to perform a task, or as an increase in the number of concurrent sessions.

Tuning is performed because either a problem already exists or the DBA wishes to prevent problems from occurring. Some examples of items to be monitored are critical table growth, changes in the statements users execute, and I/O distribution across devices. This course discusses methods to determine where waits and bottlenecks exist, and how to resolve these problems.

Tuning Phases

Tuning can be divided into different phases:

- **Application design and programming**
- **Database configuration**
- **Adding a new application**
- **Ongoing Tuning**

Tuning Phases

Application Design and Programming

Whenever possible, tuning should start at this level. With a good design, many tuning problems do not occur. For example, although it is normally good practice to fully normalized tables to reduce redundancy, this can result in a high number of table joins. By denormalizing tables the performance of the application may improve dramatically.

Database Configuration

It is important to monitor hot spots, even on fast disks. You should plan the data configuration in order to enable faster recovery times and faster data access.

Adding a New Application

When adding a new application to an existing system, the workload changes. Any major change in the workload should be accompanied by performance monitoring.

Ongoing Tuning

This is the methodology recommended for production databases. It consists of looking for bottlenecks and resolving them. Use tools to identify performance problems. By examining this data you can form an hypothesis about what is causing the bottleneck. From the hypothesis you can develop a solution and implement it. Run a test load against the database to determine if the performance problem has been resolved.

Tuning Goals

- Reducing or eliminating waits
- Accessing the least number of blocks
- Caching blocks in memory
- Response time
- Throughput
- Load
- Recovery time

Tuning Goals

Your primary goal in tuning an Oracle9*i* database is to make sure that users get responses to their statements as quickly as possible. Because "as quickly as possible" is not a precise term, the time measure must be quantified in some manner. The goal is usually measured in terms of response time, throughput, load, or recovery time.

Tuning goals can arise due to a Service Level Agreement. For example, Process A must complete within a specified time period, or a certain number of transactions per second have to be processed.

Examples of Measurable Tuning Goals

- Fewer waits
- Improved response time
- Improved database availability
- Improved memory utilization
- Improved instance hit percentages

Examples of Measurable Tuning Goals

When tuning an Oracle9*i* database environment, the DBA should establish measurable tuning goals. Without them, it will be difficult to determine when enough tuning has been performed.

- Checking for waits and bottlenecks is a good method for determining whether performance can be improved.
- Response time is how long it takes a user to receive data from a request (for example, the result set of a query), or update a table, or generate a report.
- Database availability is also a good tuning goal. Availability can be impacted due to backup and recovery, or from shutting down and starting the instance to tune parameters.
- Memory utilization is also a valid measure, because excessive paging and swapping can impact database and operating system performance. Memory utilization may also impact database hit percentages.
- Instance hit percentages provide a good baseline for determining if performance is increasing or decreasing over time.

Common Tuning Problems

- **Bad session management (usually related to middleware)**
- **Bad cursor management (usually resulting from programmer error)**
- **Bad relational designs (usually resulting from over normalization)**

Common Typing Problems

Bad Session Management

An example of this is a Web page that continually logs on and off a database. This costs the end user time while the logon procedures are followed.

Bad Cursor Management

For example, an application that does not make use of bind variables in the `where` clause. If `CURSOR_SHARING` is set to `SIMILAR` or `FORCE`, then:

(1) `select * from hr.employees where employee_id = 7900;` and

(2) `select * from hr.employees where employee_id = 7369;`

will both parse to a single cursor,

```
select * from hr.employees where employee_id = :SYS_B_0;
```

even though the SQL text is different.

Bad Relational Designs

For example, collecting the wrong columns into tables would require many table joins in order to produce output that could have been obtained from a single table.

Results of Common Tuning Problems

- **Bad session management:**
  - **Limits scalability to a point you cannot exceed**
  - **Makes the system one or two orders of magnitude slower than it should be**
- **Bad cursor management makes scalability more limited**
- **Bad relational design**
  - **Unnecessary table joins performed**
  - **Usually a result of trying to build an object interface of relational storage**

Proactive Tuning Considerations During Development

- Tune the design.
- Tune the application.
- Tune memory.
- Tune I/O.
- Tune contention.
- Tune the operating system.

Tuning Steps During Development

During the development of a new system, the following order of tuning implementation is recommended:

1. Design
2. Application
3. Memory
4. Input/output (I/O)
5. Contention
6. Operating system

Repeat the process if your goals have not yet been achieved.

The rationale for this structure is that improvements early in the sequence can save you from having to deal with problems later.

For example, if your applications use many full table scans, this may show up as excessive I/O. However, there is no point in resizing the buffer cache or redistributing disk files, if you can rewrite the queries so that they access only four blocks instead of four thousand.

The first two steps are typically the responsibility of the system architects and application developers; however, the DBA may also be involved in application tuning.

Tuning Steps During Production

- Locate the bottleneck by using tools.
- Determine the reason for the bottleneck.
- Resolve the cause.
- Check that the bottleneck has been resolved.

Tuning Steps During Production

The tuning methodology for production systems works by resolving problems before they become apparent to the users:

1. Locate a bottleneck, or a potential bottleneck, by using tools, such as `STATSPACK`, `UTLBSTAT` and `UTLESTAT`, or Oracle Enterprise Manager.
2. The bottleneck usually manifests itself as a wait event. Determine the reason for the wait event.
3. Resolve the cause of the wait. This could mean changing the size of a member of the System Global Area.
4. Check that the change has produced a positive effect on the system by running the application again, and then using the tools used in step 1.
5. Repeat the process if your goals have not yet been achieved.

The rationale for this structure is that redoing the same tuning methodology that was used for a development system wastes time. If the system requires a major overhaul, then using the development system methodology is beneficial.

Performance Versus Safety Trade-Offs

Factors that affect performance:

- **Multiple control files**
- **Multiple redo log members in a group**
- **Frequent checkpointing**
- **Backing up datafiles**
- **Performing archiving**
- **Block check numbers**
- **Number of concurrent users and transactions**

1-12

Performance Trade-Offs

There is always a trade-off with performance. In order to increase performance, something else is necessarily affected adversely. Often, recovery time is what suffers. The safer the database administrator makes the database, the slower it runs.

The decision has to be made regarding just how safe to make the database, and what level of performance is required: how many concurrent users must have access to the database, how many transactions per second must take place, and so on.

Summary

In this lesson, you should have learned how to:

- Create a good initial design
- Define a tuning methodology
- Perform production tuning
- Establish quantifiable goals
- List tuning problems
- Decide between performance and safety

2

Diagnostic and Tuning Tools

Objectives

After completing this lesson, you should be able to do the following:

- Describe how the `alert.log` file is used
- Describe how background trace files are used
- Describe how user trace files are used
- Describe the statistics kept in the dynamic performance views
- Collect statistics using STATSPACK
- Describe how STATSPACK collects statistics
- Collect statistics using Enterprise Manager
- Describe other tools used for tuning

ORACLE

2-2

Maintenance of the `alert.log` File

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

The `alert.log` File

It is important for the database administrator to check the `alert.log` file regularly to detect problems before they become serious.

The following information is logged in the `alert.log` file:

- Internal errors (ORA-600) and block corruption errors (ORA-1578 or ORA-1498)
- Operations that affect database structures and parameters, and statements such as `CREATE DATABASE`, `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- The values of all nondefault initialization parameters at the time the instance starts
- The location of the `alert.log` file is given by the `BACKGROUND_DUMP_DEST` parameter.

Tuning Using the `alert.log` File

The `alert.log` file contains the following information which can be used in tuning the database:

- **Checkpoint start and end times**
- **Incomplete checkpoints**
- **Time to perform archiving**
- **Instance recovery start and complete times**
- **Deadlock, and timeout errors**

ORACLE

2-4

Checkpointing Start and End Times

These values are written into the `alert.log` file only if the `LOG_CHECKPIONTS_TO_ALERT` parameter has been set to TRUE

Background Processes Trace Files

- The Oracle server dumps information about errors detected by any background process into trace files.
- Oracle Support uses these trace files to diagnose and troubleshoot.
- These files do not usually contain tuning information.

Background Processes Trace Files

These files are created by the background processes. In general these files contain diagnostic information, not information regarding performance tuning, however, by using events, information can be written to these files regarding performance. Database events can be set by the DBA, but usually only under the supervision of Oracle Support.

User Trace Files

- **Server process tracing can be enabled or disabled at the session or instance level.**
- **A user trace file contains statistics for traced SQL statements in that session.**
- **User trace files are created on a per server process basis.**
- **User trace files can also be created by:**
  - **Backup control file to trace**
  - **Database `SET EVENT`s**

User Trace Files

User trace files can be generated by server processes at the user's or DBA's request.

Instance-Level Tracing

This trace logging is enabled or disabled by the `SQL_TRACE` initialization parameter.

The default value is `FALSE`.

Session-Level Tracing

The following statement enables the writing to a trace file for a particular session:

```
SQL > EXECUTE dbms_system.set_sql_trace_in_session(8,12,TRUE);
```

where 8 and 12 are the system identifier and serial numbers of the connected user.

The `DBMS_SYSTEM` package is created when the `catproc.sql` script is run. This script is located in the following directory:

`$ORACLE_HOME/rdbms/admin` on UNIX systems, or

`%ORACLE_HOME%\rdbms\admin` for NT.

The following statement enables the writing to a trace file for the session of the connected user:

```
SQL> ALTER SESSION SET sql_trace=TRUE;
```

Views, Utilities, and Tools

Tools, and views, that are available to the DBA for determining performance:

- **V$xxx dynamic troubleshooting and performance views**
- **DBA_xxx dictionary views**
- **STATSPACK**
- **utlbstat.sql and utlestat.sql scripts**
- **Enterprise Manager**
- **Oracle wait events**
- **Oracle diagnostics and tuning packs**

ORACLE

2-7

Dictionary and Dynamic Views

Oracle server displays all system statistics in the V$SYSSTAT view and provides many other views for performance and troubleshooting information. You can query these views to find cumulative totals since the instance started, but this is often unhelpful; if your instance is rarely shut down, the statistics may cover a long period and have little meaning.

Oracle displays data storage statistics in DBA_*xxx* views that help you in troubleshooting the segments' storage (tables, clusters, and indexes).

STATSPACK

You can use STATSPACK to gather statistics. Similar in essence to ULBSTAT and ULTESTAT, STATSPACK also enables a DBA to collect statistics over a period of time and have the statistics stored in the database. This gives the DBA the opportunity to compare statistics over different periods of time. STATSPACK enables the DBA to collect statistics over a period of time, using several snapshots. A hard copy report can be generated from any two of the snapshots taken. In addition to the instance statistics, STATSPACK also generates information on SQL statements present in the Library Cache during the begin, or end snapshot.

The UTLBSTAT and UTLESTAT Utilities

You should gather performance figures over a defined period, probably your busiest time of day or end of month, and produce a hard-copy report.

You can do this with the `utlbstat.sql` and `utlestat.sql` scripts.

Experienced consultants usually begin a tuning project by using this utility to capture data.

Enterprise Manager

Enterprise Manager has many tools that enable the Database administrator to manage the database using a GUI tool. In order to assist in tuning there are the Diagnostics and the Tuning Packs. These packs include various managers that offer insights into many Oracle performance management areas, such as graphical monitoring, analysis, and automated tuning of Oracle databases.

Oracle Wait Events

If you are troubleshooting, you need to know when a process has waited for any resource. A list of wait events are present in the server.

Some dictionary views display the events for which sessions had to wait.

Dictionary and Special Views

The following dictionary and special views provide useful statistics after using the `DBMS_STATS` package:

- **`DBA_TABLES`, `DBA_TAB_COLUMNS`**
- **`DBA_CLUSTERS`**
- **`DBA_INDEXES`, `INDEX_STATS`**
- **`INDEX_HISTOGRAM`, `DBA_TAB_HISTOGRAMS`**

This statistical information is static until you reexecute `DBMS_STATS`.

2-10

Dictionary and Special Views

When you need to look at data storage in detail, you need to use the `DBMS_STATS` `package`, which collects statistics and populates columns in some `DBA_xxx` views.

`DBMS_STATS` populates columns in the views concerned with:

- Table data storage within extents and blocks:
  - `DBA_TABLES`
  - `DBA_TAB_COLUMNS`
- Cluster data storage within extents and blocks:
  - `DBA_CLUSTERS`
- Index data storage within extents and blocks, and indexation usefulness:
  - `DBA_INDEXES`
  - `INDEX_STATS`
- Nonindexed and indexed columns data distribution:
  - `DBA_TAB_HISTOGRAMS`
  - `INDEX_HISTOGRAM`

This package is described in more detail in the lesson “Using Oracle Blocks Efficiently.”

Dynamic Troubleshooting and Performance Views

V$ views:

- **Based on X$ tables**
- **Listed in `V$FIXED_TABLE`**

X$ tables:

- **Not usually queried directly**
- **Dynamic and constantly changing**
- **Names abbreviated and obscure**

Populated at startup and cleared at shutdown

2-11

V$ Views

- These are based on X$ tables, therefore some V$ views are available in the NOMOUNT and MOUNT stages.
- They are listed in `V$FIXED_TABLE`.
- The V$ views (actually synonyms for V_$ views) belong to the `sys` user.

X$ Tables

- These are not usually queried directly; not all the information is necessarily useful, and column names tend to be abbreviated and obscure.
- The X$ tables are memory structures that hold instance information and are available when the instance is in a `NOMOUNT` or `MOUNT` state.
- The X$ tables are dynamic, and their contents are constantly changing.
- The V$ views and the underlying X$ tables are populated at instance startup and cleared at shutdown.
- They hold timing information if you set the `TIMED_STATISTICS` `init.ora` parameter to `TRUE` or if you execute the following SQL command:

```
SQL> ALTER SYSTEM SET timed_statistics = TRUE;
```

Topics for Troubleshooting and Tuning

Systemwide Statistics

Instance/Database

V$DATABASE	T
V$INSTANCE	T
V$OPTION	T
V$PARAMETER	T/P
V$BACKUP	T
V$PX_PROCESS_SYSSTAT	T/P
V$PROCESS	T
V$WAITSTAT	T/P
V$SYSTEM_EVENT	T/P

Disk

V$DATAFILE	T/P
V$FILESTAT	T/P
V$LOG	T
V$LOG_HISTORY	T
V$DBFILE	T/P
V$TEMPFILE	P
V$TEMPSTAT	P

Memory

V$BUFFER_POOL_STATISTICS	T/P
V$DB_OBJECT_CACHE	T
V$LIBRARYCACHE	P
V$ROWCACHE	P
V$SYSSTAT	T/P
V$SGASTAT	P

Contention

V$LOCK	T/P
V$ROLLNAME	T/P
V$ROLLSTAT	T/P
V$WAITSTAT	T/P
V$LATCH	T/P

Session-Related Statistics

User/Session

V$LOCK	P
V$OPEN_CURSOR	T
V$PROCESS	T
V$SORT_USAGE	T/P
V$SESSION	T/P
V$SESSTAT	T/P
V$TRANSACTION	T
V$SESSION_EVENT	T/P
V$SESSION_WAIT	T/P
V$PX_SESSTAT	P
V$PX_SESSION	P
V$SESSION_OBJECT_CACHE	P

T: Troubleshooting
T/P: Troubleshooting/Performance
P: Performance

ORACLE

2-12

Systemwide Statistics

Views Pertaining to the Instance/Database

- `V$PX_PROCESS_SYSSTAT`: Parallel query system statistics
- `V$PROCESS`: Information about currently active processes
- `V$WAITSTAT`: Contention statistics
- `V$SYSTEM_EVENT`: Total waits for particular events

Views Pertaining to Memory

- `V$BUFFER_POOL_STATISTICS`: Buffer pools allocation on the instance (created by the `$ORACLE_HOME/rdbms/admin/catperf.sql` script)
- `V$DB_OBJECT_CACHE`: Database objects cached in the library cache
- `V$LIBRARYCACHE`: Library cache performance and activity statistics
- `V$ROWCACHE`: Data dictionary hits and misses activity
- `V$SYSSTAT`: Basic instance statistics

Systemwide Statistics (continued)

Views Pertaining to Disk Performance

- `V$FILESTAT`: Data file read/write statistics
- `V$TEMPSTAT`: Information about file read/write statistics for temporary tablespace data files

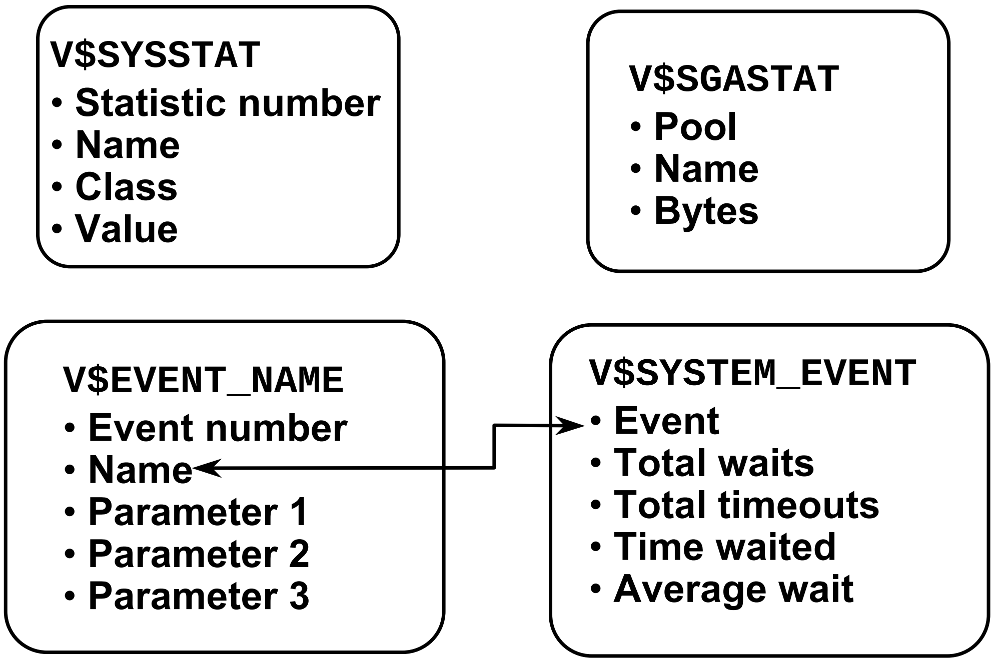
Views Pertaining to Contention

- `V$LATCH`: Statistics for each type of latch
- `V$ROLLSTAT`: Statistics for all online rollback segments
- `V$WAITSTAT`: Block contention statistics (the `TIMED_STATISTICS` init.ora parameter should be set to `TRUE`)

Session-Related Statistics

- `V$LOCK`: Locks currently held by the server and outstanding requests for a lock or latch
- `V$OPEN_CURSOR`: Cursors currently opened and parsed by each session
- `V$SORT_USAGE`: Size of temporary segments and sessions creating them; identification of processes doing disk sorts
- `V$SESSTAT`: User session statistics
- `V$SESSION_EVENT`: Information on waits for an event by a session
- `V$SESSION_WAIT`: Resources or events for which active sessions are waiting
- `V$PX_SESSTAT`: Information about the sessions running in parallel.

Collecting Systemwide Statistics



2-14

General Systemwide Statistics

All kinds of systemwide statistics are cataloged in the V$STATNAME view: about 290 statistics are available.

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.

Example

```
SQL>  SELECT name,class,value FROM v$sysstat;
NAME                        CLASS                VALUE
------------------        ------            ----------
logons cumulative              1                  6393
logons current                 1                    10
opened cursors cumulative      1                101298
table scans (short tables)    64                  6943
table scans (long tables)     64                   344
table scans (rowid ranges)    64                     0
redo entries                   2               1126226
redo size                      2             816992940
redo buffer allocation
retries                        2                  1461
redo wastage                   2               5874784
……
The results shown are only a partial display of the output.
```

General Systemwide Statistics

These statistics are classified by the topics of tuning:

- Class 1 refers to general instance activity.
- Class 2 refers to redo log buffer activity.
- Class 4 refers to locking.
- Class 8 refers to database buffer cache activity
- Class 16 refers to OS activity.
- Class 32 refers to parallelization.
- Class 64 refers to tables access.
- Class 128 refers to debugging purposes.

SGA Global Statistics

Server displays all calculated memory statistics in the `V$SGASTAT` view. You can query this view to find cumulative totals of detailed SGA usage since the instance started.

Example

```
SQL>     SELECT * FROM v$sgastat;
POOL                NAME                             BYTES
------              -------------------------        ----------
                    fixed_sga                        46136
                    db_block_buffers                 409600
                    log_buffer                       524288
shared pool         free memory                      8341616
shared pool         SYSTEM PARAMETERS                42496
shared pool         transaction                      64800
shared pool         dictionary cache                 156524
shared pool         library cache                    358660
shared pool         sql area                         551488
```

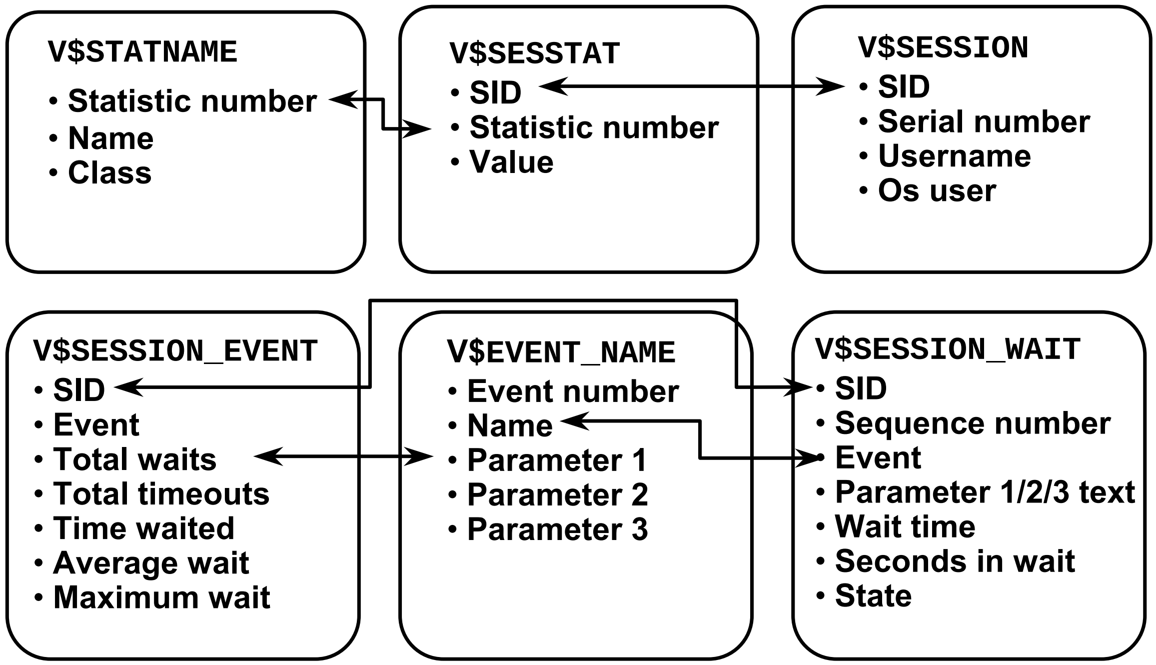
Waiting Events Statistics

All kinds of waiting events are cataloged in the V$EVENT_NAME view: about 286 events are available.

Cumulative statistics for all sessions are stored in V$SYSTEM_EVENT, which shows the total waits for a particular event since instance startup.

If you are troubleshooting, you need to know when a process has waited for any resource.

Collecting Session-Related Statistics



ORACLE

2-17

General Session-Related Statistics

Session data is cumulative from connect time.

You can display current session information for each user logged on.

The V$MYSTAT view displays the statistics of the current session.

Example: Determine the type of connection the users have.

```
SQL>    SELECT sid, username, type, server FROM v$session;
SID    USERNAME         TYPE             SERVER
-----  ------------     -----------      ----------------
  1                     BACKGROUND       DEDICATED
  2                     BACKGROUND       DEDICATED
  3                     BACKGROUND       DEDICATED
  4                     BACKGROUND       DEDICATED
  5                     BACKGROUND       DEDICATED
  6                     BACKGROUND       DEDICATED
  9    SYSTEM           USER             DEDICATED
```

Oracle server displays all calculated session statistics in the V$SESSTAT view. You can query this view to find session cumulative totals since the instance started.

General Session-Related Statistics (continued)

Example: Determine the sessions that consume more than 30,000 bytes of PGA memory.

```
SQL> select username,name,value
2  from v$statname n, v$session s, v$sesstat t
3  where s.sid=t.sid
4  and   n.statistic#=t.statistic#
5  and   s.type='USER'
6  and   s.username is not null
7  and   n.name='session pga memory'
8* and   t.value > 30000;
USERNAME    NAME                   VALUE
----------  -------------------    -----------
SYSTEM      session pga memory     468816
```

Session Waiting Events Statistics

V$SESSION_EVENT shows, by session, the total waits for a particular event since instance startup.

V$SESSION_WAIT view lists the resources or events for which active sessions are waiting.

If you are troubleshooting, you need to know when a process has waited for any resource. The structure of V$SESSION_WAIT makes it easy to check in real time whether any sessions are waiting, and why.

```
 SQL> select sid, event
 2  from V$SESSION_WAIT
 3* where wait_time = 0;
SID         EVENT
----------- ---------------------------------------
   1        pmon timer
   2        rdbms ipc message
   3        rdbms ipc message
   9        rdbms ipc message
   16       rdbms ipc message
   17       rdbms ipc message
   10       rdbms ipc message
   5        smon timer
8  rows selected.
```

You can then investigate further to see whether such waits occur frequently and whether they can be correlated with other phenomena, such as the use of particular modules.

Oracle Wait Events

- A collection of wait events provides information on the sessions that had to wait or must wait for different reasons.
- These events are listed in the `V$EVENT_NAME` view, which has the following columns:
  - `EVENT#`
  - `NAME`
  - `PARAMETER1`
  - `PARAMETER2`
  - `PARAMETER3`

ORACLE

2-19

List of Events

There are about 290 wait events in the Oracle server, including:

- Free Buffer Wait
- Latch Free
- Buffer Busy Waits
- Db File Sequential Read
- Db File Scattered Read
- Db File Parallel Write
- Undo Segment Tx Slot
- Undo Segment Extension

For the complete list, refer to the *Oracle9i Reference,* Release 9.0.1, Appendix A.

The V$EVENT_NAME View

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

```
NAME                             PARAMETER1  PARAMETER2  PARAMETER3
-------------------------------  ----------  ----------  ----------
PL/SQL lock timer                duration
alter system set mts_dispatcher  waited
buffer busy waits                file#       block#      id
library cache pin                handle addr pin address 0*mode+name
log buffer space
log file switch
(checkpoint incomplete)
transaction                      undo seg#   wrap#       count
...
286 rows selected.
```


Parameters Describing a Wait Event

Example 1: The Buffer Busy Waits event waits until a buffer becomes available. This event can be caused by a number of conditions but generally it is when a block must be brought into the buffer cache and there is already a current wait or a block in the cache needs to modified by a session and another session has been locked.

This event is accompanied by three parameters:

- `FILE#` and `BLOCK#`: These parameters identify the block number in the data file that is identified by the file number for the block for which the server needs to wait.
- `ID`: The buffer busy wait event is called from different places in the session. Each place in the kernel points to a different reason. `ID` refers to the place in the session calling this event.

Example 2: The Log File Switch (Checkpoint Incomplete) waits for a log switch because the session cannot wrap into the next log. Wrapping cannot be performed because the checkpoint for that log has not completed.

This event has no parameter.

Statistics Event Views

- `V$SYSTEM_EVENT`: **Total waits for an event, all sessions together**
- `V$SESSION_EVENT`: **Waits for an event for each session that had to wait**
- `V$SESSION_WAIT`: **Waits for an event for current active sessions that are waiting**

2-21

Statistics for Waiting Sessions

The statistics results of the sessions that had to wait or are currently waiting for a resource are stored in the `V$SESSION_EVENT` and `V$SESSION_WAIT` views.

Cumulative statistics for all sessions are stored in `V$SYSTEM_EVENT`.

The V$SYSTEM_EVENT View

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

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


The V$SYSTEM_EVENT View

V$SYSTEM_EVENT shows the total waits for a particular event since instance startup.

If you are performing ongoing tuning, you need to know when a process has waited for any resource. Therefore, it becomes useful to query this view each time the system slows down.

V$SYSTEM_EVENT contains the following columns:

- EVENT: Name of the wait event
- TOTAL_WAITS: Total number of waits for event
- TOTAL_TIMEOUTS: Total number of timeouts for event
- TIME_WAITED: Total amount of time waited for this event, in hundredths of a second
- AVERAGE_WAIT: The average amount of time waited for this event, in hundredths of a second

The V$SESSION_EVENT View

```
SQL> select sid, event, total_waits,average_wait
  2> from v$session_event where sid=10;
```

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

2-23

The V$SESSION_EVENT View

V$SESSION_EVENT shows the same information as V$SYSTEM_EVENT, except now by session. It includes the columns listed in the previous page, with an extra column for SID to identify the session. You can join the SID column to V$SESSION.SID to find user details.

You can query this view to determine all session waits since the session started.

The V$SESSION_WAIT View

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

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


The V$SESSION_WAIT View

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

Columns

- SID: Session identifier
- SEQ#: Sequence number identifying the wait
- EVENT: Resource or event waited for
- P1TEXT: Description of first additional parameter, which corresponds to the PARAMETER1 described for the V$EVENT_NAME view
- P1: First additional parameter value
- P1RAW: First additional parameter value, in hexadecimal
- P2TEXT: Description of second additional parameter, which corresponds to the PARAMETER2 described for the V$EVENT_NAME view
- P2: Second additional parameter value
- P2RAW: Second additional parameter value in hexadecimal
- P3TEXT: Description of third additional parameter, which corresponds to the PARAMETER3 described for the V$EVENT_NAME view
- P3: Third additional parameter value

V$SESSION_WAIT View (continued)

Columns (continued)

- P3RAW: Third additional parameter value in hexadecimal
- WAIT_TIME

Value	Explanation
>0	The session's last wait time
=0	The session is currently waiting
=-1	The value was less than 1/100 of a second
=-2	The system cannot provide timing information

- SECONDS_IN_WAIT: Number of seconds the event waited
- STATE: Waiting, Waited Unknown Time, Waited Short Time (less than one one-hundredth of a second), Waited Known Time (the value is stored in the WAIT_TIME column)

Note: Not all of the parameter columns are used for all events.

The TIMED_STATISTICS Initialization Parameter

Set the TIMED_STATISTICS parameter to TRUE to retrieve values in the WAIT_TIME column. It is a dynamic initialization parameter.

STATSPACK

- Installation of STATSPACK
  – $ORACLE_HOME/rdbms/admin/spcreate.sql
- Collection of statistics
  – execute STATSPACK.snap
- Automatic collection of statistics
  – $ORACLE_HOME/rdbms/admin/spauto.sql
- Produce a report
  – $ORACLE_HOME/rdbms/admin/spreport.sql
- To collect timing information, set TIMED_STATISTICS = true

STATSPACK

The STATSPACK package has been available with Oracle Database from Oracle8.1.6. When initially installed roughly 80 MBs of the Perfstat user's default tablespace is used. This may grow later with the tables storing snapshot information.

Installation of the STATSPACK Package

Installing the STATSPACK utility creates the Perfstat user, who owns all PL/SQL code and database objects created (including the STATSPACK tables, the constraints, and the STATSPACK package). During the installation you will be prompted for the Perfstat user's default and temporary tablespaces.

Collecting Statistics

Take a snapshot of performance data, log in to SQL*Plus as the Perfstat user by executing the STATSPACK.snap procedure. This stores the current values for the performance statistics in the STATSPACK tables, which can be used as a baseline snapshot for comparison with another snapshot taken at a later time.

Automatically Collecting Statistics

To compare performance from one day, week, or year to the next, there must be multiple snapshots taken over a period of time. The best method to gather snapshots is to automate the collection at regular time intervals.

The `spauto.sql script` makes use of `dbms_job`, the automated method for collecting statistics. The supplied script schedules a snapshot every hour, on the hour.This can be changed to suit the requirements of the system.

Producing a Report

To examine the change in statistics between two time periods, execute the `spreport.sql` file while being connected to the `Perfstat` user. The user is shown a list of collection periods and selects a start and end period. The difference between the statistics at each end point is then calculated and put into a file named by the user.

STATSPACK Output

Information found on the first page:

- Database and instance name
- Time at which the snapshots were taken
- Current sizes of the caches
- Load profile
- Efficiency percentages of the instance
- Top five wait events

ORACLE

First Page of the STATSPACK Report

The first page summarizes the report. The items found on this page include most of the information that the DBA requires to enhance performance. Each of these areas will be covered in greater depth during the course.

Cache Sizes

The current size of the buffer cache, shared pool, and log buffer are shown here in KB. Also included here is the value of the primary block size.

Load Profile

One value given here is per second, and the other is per transaction. What is shown includes the redo size, and physical reads, and physical writes performed during the snapshot period.

Instance Efficiency Percentages

The items listed here are ones that are most often used for tuning a database. The goal is to have all percentages at 100%, or as close as possible. Compare these values with what the normal values are for the database.

Top Five Wait Events

The full list of wait events appears later in the report, and will be dealt with later in the course. The list given here provides the DBA with the top contention items.

STATSPACK Output

Information found in the remainder of the document:

- **Complete list of wait events**
- **Information on SQL statements currently in the pool**
- **Instance activity statistics**
- **Tablespace and file I/O**
- **Buffer pool statistics**

2-29

STATSPACK Output

The report generated by STATSPACK is similar to that of utlbstat and utlestat, but STATSPACK is much better. The first page summary found with STATSPACK aids the DBA, to determine these important numbers.

STATSPACK also assists in giving information about the SQL statements that are stored in the shared SQL area, thus giving the DBA or developer better information about which statements to tune in order to best use his or her time.

Wait events are also ordered by wait time so as to put the most problematic events at the top. STATSPACK also attempts to put those wait events which are not a problem (for example, pmon timer) at the end of the list, regardless of what the time value is.

SQL Statements

Several different ordered lists of SQL statements are given. SQL statements are sorted in order of number of executions, number of parse calls, number of buffer gets, number of reads. By ordering the SQL statements by these different columns, it is easier to determine which statements are causing the heavy work load. These statements should then be tuned first, so as to get the highest return on time spent.

STATSPACK Output

Information found in the remainder of the document:

- **Rollback or undo segment statistics**
- **Latch activity**
- **Dictionary cache statistics**
- **Library cache statistics**
- **SGA statistics**
- **Startup values for `init.ora` parameters**

Rollback or Undo Segment Statistics

Because the DBA can use either rollback segments or undo segments, STATSPACK covers both options

SGA Statistics

This is made up of two lists, a memory summary, and a breakdown of each area.

Startup Values for `init.ora` Parameters

Because many `init.ora` parameters are dynamic, that is, can be changed without stopping and starting the database, there is no guarantee that the current value will be the one used if the system is restarted.

UTLBSTAT and UTLESTAT Utilities

These utilities:

- Gather performance figures over a defined period
- Produce a hard-copy report
- To fully use these, set `TIMED_STATISTICS to TRUE`
- Use the `utlbstat.sql` and `utlestat.sql` scripts
- Run the scripts from SQL*Plus connected as `SYSDBA`
- `STATSPACK` provides clearer statistics.

Begin and End Scripts

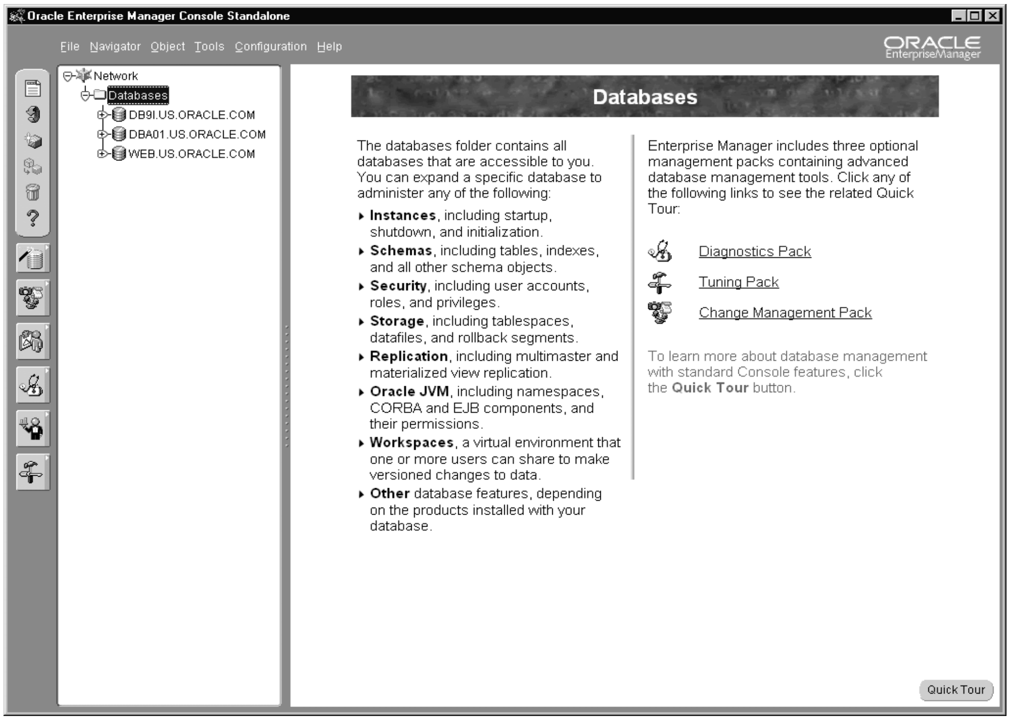
The dynamic views display cumulative totals since the instance started, but this is rarely helpful. If your instance is infrequently shut down, the statistics may cover a long period and have little meaning.

You should gather performance figures over a defined period, probably your busiest time of day or month end, and produce a hard-copy report.

You can do this by running the `utlbstat.sql` script at the beginning of the defined period and then `utlestat.sql` at the end of the period. These scripts are both stored in the `$ORACLE_HOME/rdbms/admin` directory on UNIX and in `%ORACLE_HOME%\Rdbms\Admin` on Windows.

The `ultestat.sql` script also generates a hard-copy report in the current directory. This report contains statistics for the collection period, gathered by looking at the differences between the beginning and end statistics.

Enterprise Manager Console



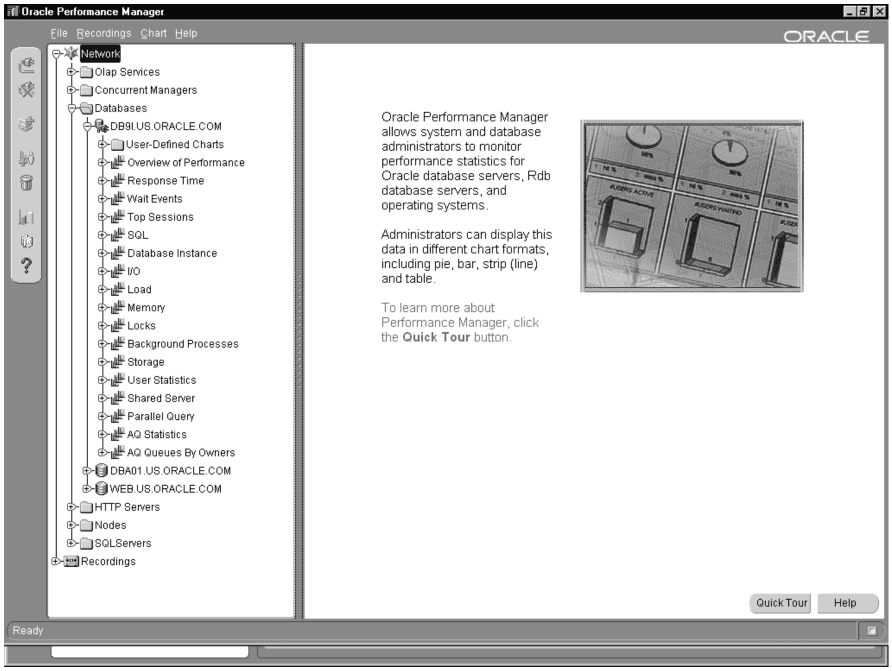
Enterprise Manager Console

From the Enterprise Manager Console the administrator can use GUI tools to manage the database. There are three packs that can be added onto the Enterprise Manager Console, these are:

- Diagnostics Pack
  - Lock Monitor
  - Performance Manager
  - Performance Overview
  - Top Sessions
  - Top SQL
  - Trace Data Viewer
- Tuning Pack
  - Oracle Expert
  - Outline Management
  - SQL Analyze
  - Tablespace Map
- Change Management Pack
  - Change Manager

The Diagnostic and Tuning packs will be used during the running of this course.

Performance Manager



Performance Manager

Performance Manager is not a part of the basic Oracle Enterprise Manager that is shipped with the database. Rather it is a part of an optional package.

Performance Manager Characteristics

The Performance Manager application captures, computes, and presents performance data in a graphical, real-time view that allows you to monitor the key metrics required to:

- Use memory effectively
- Minimize disk I/O
- Avoid resource contention

The data displayed in real-time mode can be recorded for replay.

You can also define new charts and display windows containing charts in many categories.

Predefined Charts

Seven different categories of predefined charts are available for display. Each category has a set of specific charts that focus on the parent subject.

I/O

These charts include File I/O Rate, File I/O Rate Details, Network I/O Rate, and System I/O Rate.

Performance Manager (continued)

Contention

These charts include Circuit, Dispatcher, Free List Hit %, Latch, Lock, Queue, Redo Allocation Hit %, Rollback NoWait Hit %, and Shared Server.

Database_Instance

These charts include Process, Session, System Statistics, Table Access, Tablespace, Tablespace Free Space, # Users Active, # Users Logged on, # Users Waiting, # Users Waiting for Locks, and # Users Running.

Load

These charts include Buffer Gets Rate, Network Bytes Rate, Redo Statistics Rate, Sort Rows Rate, Table Scan Rows Rate, and Throughput Rate.

Memory

These charts include Buffer Cache Hit %, Data Dictionary Cache Hit %, Library Cache Hit %, Library Cache Details, SQL Area, Memory Allocated, Memory Sort Hit %, Parse Ratio, and Read Consistency Hit %.

Top Resource Consumers

Top Resource Consumers is one of the predefined charts for database services.

Overview of Performance

The overview displays a composite of the most commonly used charts from the other categories. Use the overview charts to get an overall picture of your database activity, then drill down to the more specific reports if you need to. The set includes the following:

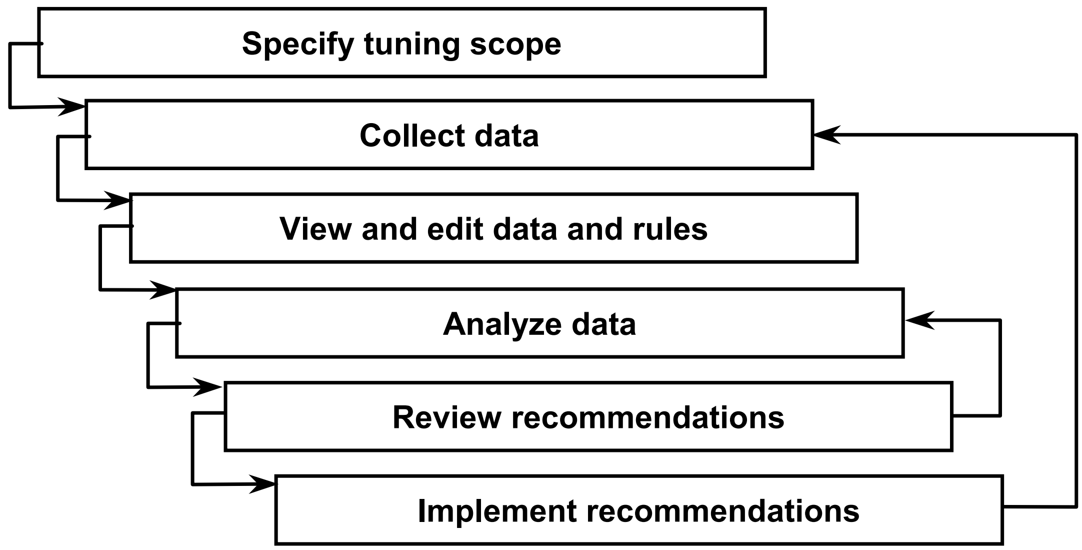
Overview of Throughput is a group chart for the Overview of Performance class. The icon depicts four small bar graphs.

Overview of Performance Default Chart is an individual chart for the Overview of Performance class. The icon depicts a bar graph.

User-Defined Charts

If you have defined any charts of your own, you can select from among them by using this category.

Overview of Oracle Expert Tuning Methodology



Oracle Expert

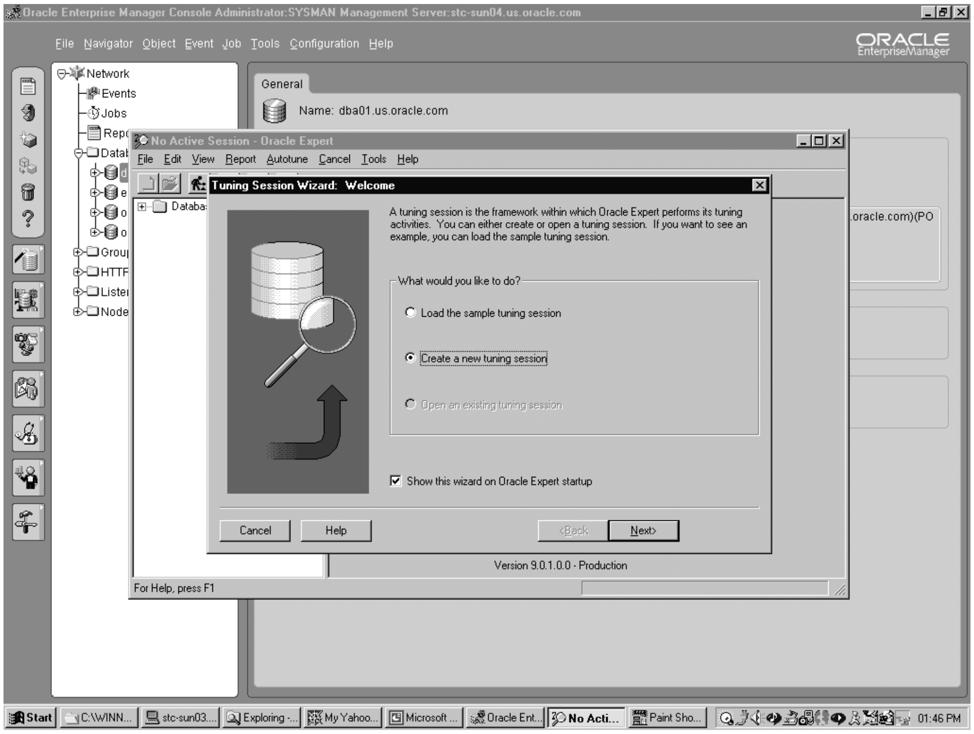
Oracle Expert is not a part of the basic Oracle Enterprise Manager that is shipped with the database. Rather it is a part of an optional package.

Methodology

Oracle Expert provides automated performance tuning with integrated rules. It automates:

- Collecting data
- Analyzing data
- Generating recommendations
- Creating scripts for recommendations implementation

Tuning Using Oracle Expert



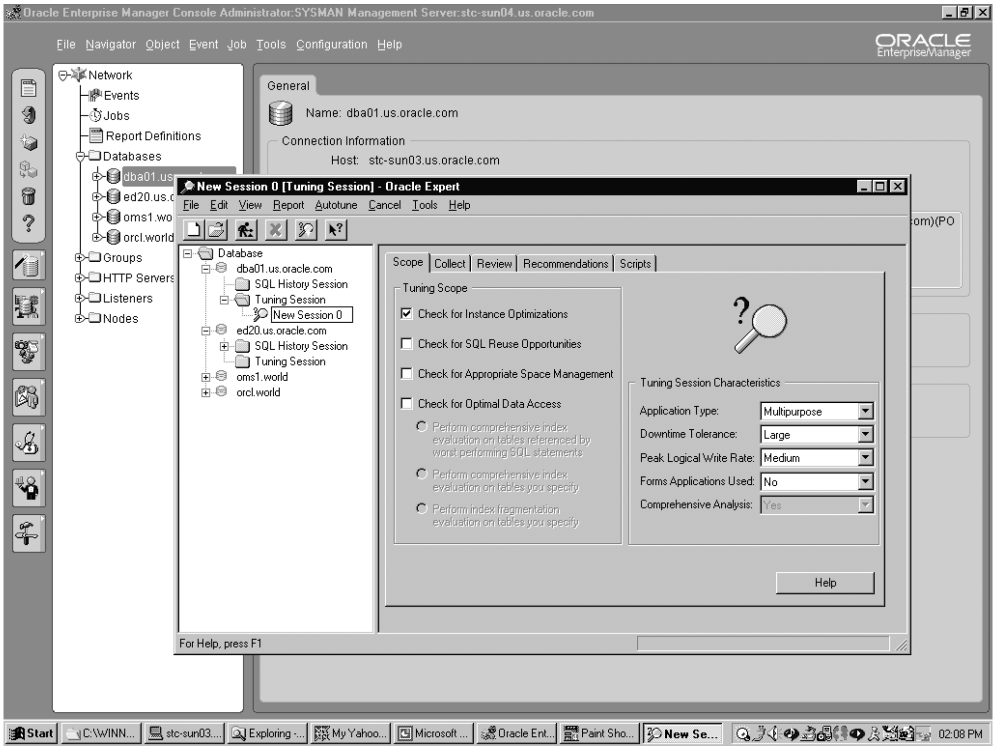
ORACLE

2-36

Tuning Using Oracle Expert

From the Enterprise Manager Console start the Oracle Expert. It is possible to use a set of data from a previous collection or to collect a new set.

Tuning Using Oracle Expert



ORACLE

2-37

Tuning Using Oracle Expert (continued)

Select the Tuning Scope. This will determine the amount of data that must be collected. The broader the scope, the more time will be required for the collection and analyze stages.

DBA-Developed Tools

- Develop your own scripts.
- Use the supplied packages for tuning.
- Schedule periodic performance checking.
- Take advantage of the EM Job service to automate the regular execution of administrative tasks.
- Take advantage of the EM Event service to track specific situations.
- Take advantage of the EM Job service to apply tasks that automatically solve problems detected by EM event service.

2-38

In-House Scripts

The utilities and Oracle tools may not provide all the statistics you need.

Therefore, you can create your own scripts to do the following:

- Check for free space in every data file
- Determine whether segments have enough space to be able to extend
- Describe schema structures to show tables and associated indexes
- Use Oracle-supplied packages (created by `$ORACLE_HOME/rdbms/admin/dbms*.sql` scripts) (refer to the Oracle Database Administration course for more information)

One of the problems with in-house scripts is that it takes time for a new DBA to learn what scripts to use to perform what tasks. Most DBA's will be familiar with STATSPACK, and / or ultbstat / utlestat reports

Summary

In this lesson, you should have learned how to:

- Use the alert log file
- Get information from background processes trace files
- Trace user SQL statements
- Collect statistics from dictionary and dynamic performance troubleshooting views
- Use the `STATSPACK` utility to collect performance data
- Retrieve wait events information

Practice 2

The goal of this practice is to familiarize yourself with the different methods of collecting statistical information. Through out this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Log on as directed by the instructor. If the database is not already started, connect to SQL*Plus using "system/manager as sysdba" then start it using the `STARTUP` command. Check that `TIMED_STATISTICS` has been set to `TRUE`; if it has not then set it using the `init.ora` file (located at `$HOME/ADMIN/PFILE`), or the alter system statement.
2. Connect to SQL*Plus as user SYSTEM, and issue a command that will create a trace file for this session. Run a query to count the number of rows in the `DBA_TABLES` dictionary view. In order to locate your new trace file easier, if possible, delete all the trace files in the `USER_DUMP_DEST` before running the trace. Disable the trace command after running the query.
3. At the operating system level view the resulting trace file located in the directory set by `USER_DUMP_DEST`. Do not try to interpret the content of the trace file as this is the topic of a later lesson.
4. Open two session, the first as user HR/HR, and the second as user system/manager. From the second session generate a user trace file for the first session using the `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` procedure. Get the user ID and SERIAL# from V$SESSION.
5. Confirm that the trace file has been created in the directory set by `USER_DUMP_DEST.`
6. Connect to SQL*Plus using "system/manager," and create a new tablespace (`TOOLS`) to hold the tables and other segments required by `STATSPACK`. This tablespace needs to be 100 MB, and should be dictionary managed (this is not a requirement of STATSPACK, but will be used later in the course). Name the data file `tools01.dbf` and place in the `$HOME/ORADATA/u05` directory.
   Note: Dictionary managed is not the default.
7. Confirm and record the amount of free space available within the `TOOLS` tablespace by querying the `DBA_FREE_SPACE` view.
8. Connect using "system/manager", then install `STATSPACK` using the spcreate.sql script located in your `$HOME/STUDENT/LABS` directory. Use the following settings when asked by the installation program.
   User's Default Tablespace = `TOOLS`
   User's Temporary Tablespace = `TEMP`
9. Query DBA_FREE_SPACE to determine the amount of free space space left in the `TOOLS` tablespace. The difference between this value and the one recorded in step 7 will be the space required for the initial installation of `STATSPACK`. Note that the amount of storage space required will increase in proportion to the amount of information stored within the `STATSPACK` tables, that is, the number of snapshots.

Practice 2 (continued)

10. Manually collect current statistics using STATSPACK by running the snap.sql script located in $HOME/STUDENT/LABS. This will return the snap_id for the snapshot just taken, which should be recorded.
11. In order to have STATSPACK automatically collect statistics every 3 minutes execute the spauto.sql script located it your $HOME/STUDENT/LABS directory. Query the database to confirm that the job has been registered using the user_jobs view.
12. After waiting for a period in excess of three minutes query the stats$snapshot view in order to list what snapshots have been collected. There must be at least two snapshots before moving to the next step.
13. Once there are at least two snapshots, start to generate a report. This is performed using the spreport.sql script found in the $HOME/STUDENT/LABS directory. The script lists the snapshot options available, and then requests the beginning snap ID and the end snap ID. The user is then requested to give a filename for the report. It is often best left to the default.
14. Locate the report file in the users current directory, then using any text editor, open and examine the report. The first page shows a collection of the most queried statistics.
15. Connect to the database as a system administrator “system/manager.”
16. Query the database in order to determine what system wait events have been registered since startup using v$system_event.
17. Determine if there are any sessions actually waiting for resources, using v$session_wait.
18. Stop the automatic collection of statistics by removing the job. This is performed by connecting as user perfstat/perfstat and querying the view user_jobs to get the job number. Then execute DBMS_JOB.REMOVE (<job number>);
19. Connect to your database using Enterprise Manager. The lecturer will supply the information required to connect to the Oracle Management Server. When connected use Enterprise Manager to explore the database. Examine items such as the number of tablespaces, users, and tables.
20. From Enterprise Manager load Oracle Expert, and create a new Tuning session. Limit the tuning scope to “Check for Instance Optimizations.” This is done in order to reduce the time taken to collect information. Collect a new set of data. Do not implement the changes that Oracle Expert recommends, as this will be done during the course.

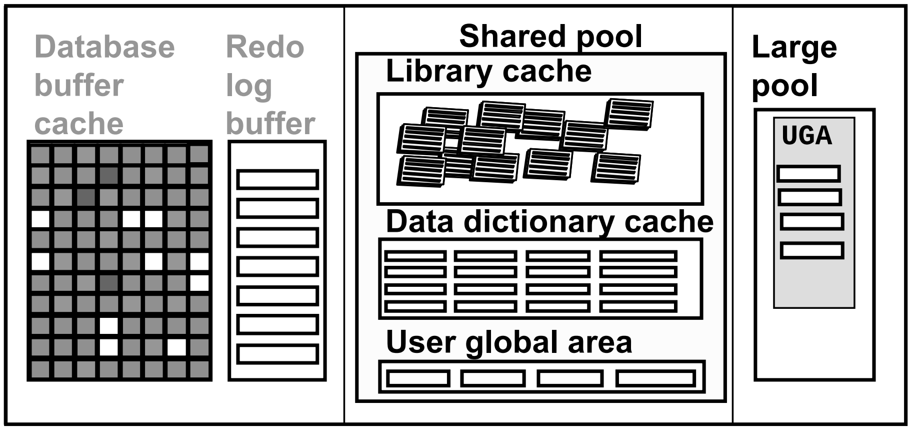
Sizing the Shared Pool

Objectives

After completing this lesson, you should be able to do the following:

- Measure and tune the library cache hit ratio
- Measure and tune the dictionary cache hit ratio
- Size and pin objects in the shared pool
- Tune the shared pool reserved space
- Describe the user global area (UGA) and session memory considerations
- List other tuning issues related to the shared pool
- Set the large pool

The System Global Area



Major components of the shared pool are:

- **Library cache**
- **Data dictionary cache**
- **User global area (UGA) for shared server connections**

Shared Pool Contents

The *shared pool* contains the following structures:

- The *library cache,* which stores shared SQL and PL/SQL code
- The *data dictionary cache*, which keeps information about dictionary objects
- The *user global area (UGA),* which keeps information about shared server connections, when Large pool is not configured.

Tuning Considerations for the Shared Pool

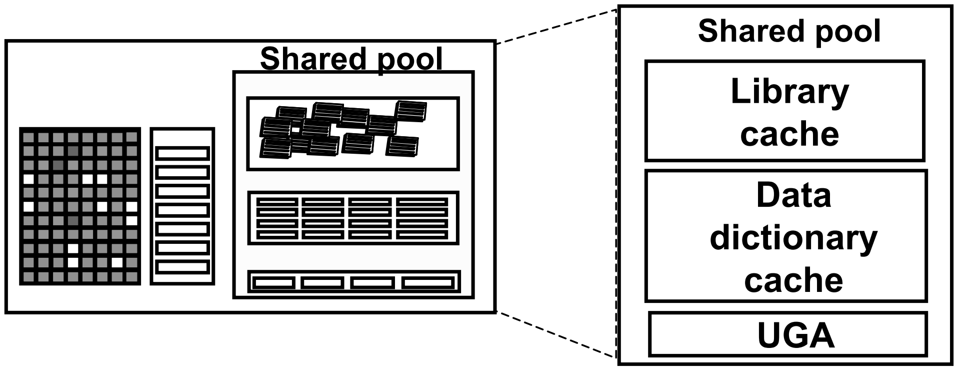
A cache miss on the data dictionary cache or library cache is usually more expensive than a miss on most other SGA memory structures. Tuning the shared pool is a priority.

When you tune the shared pool, you should concentrate on the library cache, because the algorithm that assigns memory space in the Shared_Pool prefers to hold dictionary data in memory longer than library cache data. Therefore, tuning the library cache to an acceptable cache hit ratio ensures that the data dictionary cache hit ratio is also acceptable.

If the shared pool is too small, the server must dedicate resources to managing the limited space available. This consumes CPU resources and causes contention.

If the shared pool is too large contention can also result because of the overhead of managing the fragmentation.

The Shared Pool



- **Size is defined by `SHARED_POOL_SIZE`.**
- **Library cache contains statement text, parsed code, and execution plan.**
- **Data dictionary cache contains definitions for tables, columns, and privileges from the data dictionary tables.**
- **UGA contains session information for Oracle Shared Server users when large pool is not configured.**

ORACLE

3-4

Size of the Shared Pool

You set the size of the shared pool with the `SHARED_POOL_SIZE` initialization parameter. It defaults to 8,388,608 bytes (8 MB).

The Library Cache

The library cache contains shared SQL and PL/SQL areas: the fully parsed or compiled representations of PL/SQL blocks and SQL statements.

PL/SQL blocks include:

- Procedures
- Functions
- Packages
- Triggers
- Anonymous PL/SQL blocks

The Data Dictionary Cache

The data dictionary cache holds definitions of dictionary objects in memory.

User Global Area

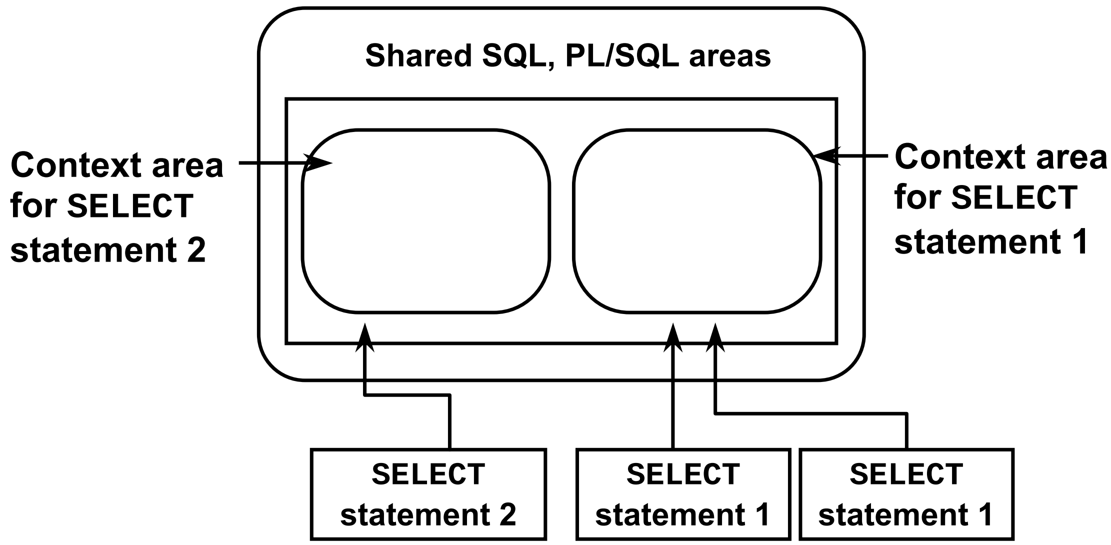
The UGA contains the session information for Oracle Shared server. The UGA is located in the shared pool when using shared server session and if large pool is not configured.

The Library Cache

- Used to store SQL statements and PL/SQL blocks to be shared by users
- Managed by a least recently used (LRU) algorithm
- Used to prevent statements reparsing

ORACLE

The Library Cache



SQL and PL/SQL Storage

The Oracle server uses the library cache to store SQL statements and PL/SQL blocks. A least recently used (LRU) algorithm is used to manage the cache.

If a user issues a statement that is already in the cache, the Oracle server can use the cached version without having to reparse it.

To find whether a statement is already cached, the Oracle server:

- Reduces the statement to the numeric value of the ASCII text
- Uses a hash function of this number

Tuning the Library Cache

Reduce misses by keeping parsing to a minimum:

- **Make sure that users can share statements**
- **Prevent statements from being aged out by allocating enough space**
- **Avoid invalidations that induce reparsing**

Tuning Goal

In an OLTP environment, reparsing should be minimized.

- If an application makes a parse call for a SQL statement and the parsed representation of the statement does not already exist in a shared SQL area in the library cache, the Oracle server parses the statement and allocates a shared SQL area.
- Ensure that SQL statements can share a shared SQL area whenever possible by using as much reusable code as possible, for example having bind variables rather than constants
- If an application makes an execute call for a SQL statement, and the shared SQL area containing the parsed representation of the statement has been deallocated from the library cache to make room for another statement, the Oracle server implicitly reparses the statement, allocates a new shared SQL area for it, and executes it. Reduce library cache misses on execution calls by allocating more memory to the library cache.
- If a schema object is referenced in the execution plan of a SQL statement and that object is later modified in any way, the parsed SQL statement held in memory is rendered invalid, and will need to be parsed again when used.

Tuning the Library Cache

Avoid fragmentation by:

- **Reserving space for large memory requirements**
- **Pinning frequently required large objects**
- **Eliminating large anonymous PL/SQL blocks**
- **Enabling the use of large pool for Oracle Shared Server connections**

Second Tuning Goal

Reduce memory fragmentation by:

- Ensuring availability of contiguous space for large memory requirements through the allocation of reserved space in the shared pool area
- Pinning frequently required large objects such as SQL and PL/SQL areas in memory, instead of aging them out with the normal LRU mechanism
- Using small PL/SQL packaged functions instead of large anonymous blocks.
- Configuring the large pool for Oracle Shared Server connections.
- Measuring session memory use by the shared server processes in Oracle Shared Server connections

Terminology

- **Gets: (Parse) The number of lookups for objects of the namespace**
- **Pins: (Execution) The number of reads or executions of the objects of the namespace**
- **Reloads: (Parse) The number of library cache misses on the execution step, thereby causing an implicit reparsing of the SQL statement**
- **Invalidations: (Parse) If an object is modified then all explain plans that reference the object would be marked invalid, and have to be parsed again**

ORACLE

Three Keywords

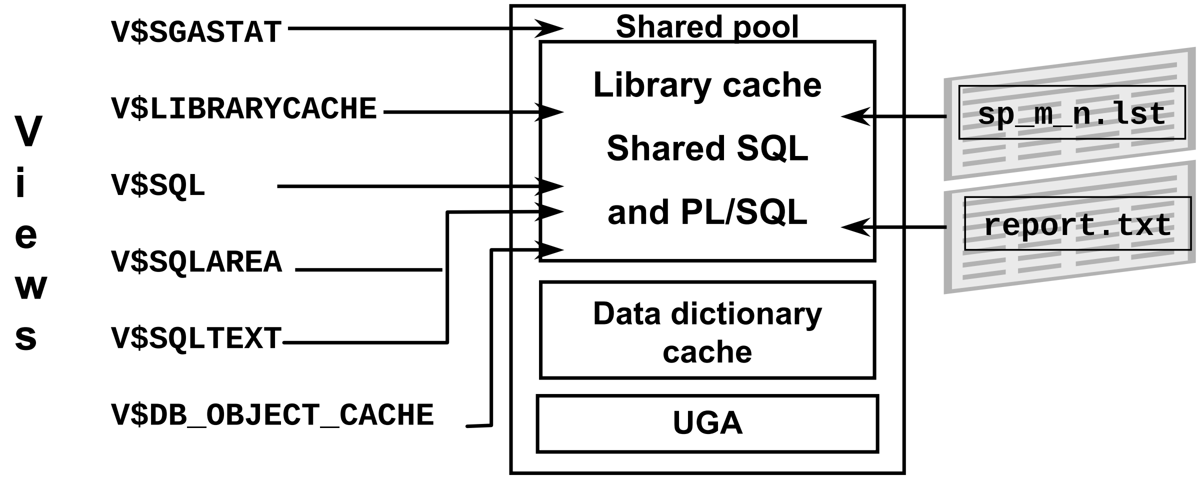
Each row in V$LIBRARYCACHE contains statistics for one type of item kept in the library cache. The item described by each row is identified by the value of the NAMESPACE column. Rows of the table with the following NAMESPACE values reflect library cache activity for SQL statements and PL/SQL blocks: 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, and PIPE.

Three keywords related to the namespace are:

- GETS: Shows the total number of requests for information on the corresponding item.
- PINS: For each of these areas, PINS shows the number of executions of SQL statements or procedures.
- RELOADS: If an execute call for a SQL statement is performed and the shared SQL area containing the parsed representation of the statement has been deallocated from the library cache to make room for another statement or because the objects the statement refers to have been invalidated, the Oracle server implicitly reloads the statement and therefore parses it again. The number of reloads is counted for each of these namespaces.
- INVALIDATIONS: When an object is modified, then it is possible that there could be a better execution path for all statements that use the object. For this reason the Oracle server will mark all executions that use a modified object as being invalid.

Diagnostic Tools for Tuning the Library Cache



Parameters affecting the components:
`SHARED_POOL_SIZE, OPEN_CURSORS`
`SESSION_CACHED_CURSORS, CURSOR_SPACE_FOR_TIME`
`CURSOR_SHARING, SHARED_POOL_RESERVED_SIZE`

3-10

Description of the Views

`V$SGASTAT` displays the sizes of all SGA structures. The contents of the shared pool are not aged out as long as free memory is available in the shared pool.

Other dynamic views that help in diagnosing performance issues related to the library cache are:

- `V$LIBRARYCACHE`: Statistics on library cache management
- `V$SQLAREA`: Full statistics about all shared cursors, and the first 1,000 characters of the SQL statement
- V$SQL: This view lists statistics on shared SQL area and contains one row for each child of the original SQL text entered. V$SQL is a similar view to V$SQLAREA, except that it does not include a GROUP BY clause that can make the V$SQLAREA view expensive to query.
- `V$SQLTEXT`: The full SQL text without truncation, in multiple rows
- `V$DB_OBJECT_CACHE`: Database objects cached, including packages; also objects such as tables and synonyms, where these are referenced in SQL statements

The `STATSPACK` report can be used to check the Instance Efficiency Percentages, Shared Pool Statistics, and the Instance Activity Stats related to the Shared Pool. In addition the report contains information on SQL statements found in the Library Cache when either of the snapshots was taken.

Are Cursors Being Shared?

- Check GETHITRATIO in V$LIBRARYCACHE:

```
SQL> select gethitratio
  2  from v$librarycache
  3  where namespace = 'SQL AREA';
```

- Find out which statements users are running:

```
SQL> select sql_text, users_executing,
 2          executions, loads
 3    from v$sqlarea;
```

```
SQL> select * from v$sqltext
 2   where sql_text like
 3   'select * from hr.employees where %';
```

3-11

Are Cursors Being Shared?

V$LIBRARYCACHE view: GETHITRATIO determines the percentage of parse calls that find a cursor to share (GETHITS/GETS). This ratio should be in the high 90s in OLTP environments. If not, you can try to:

- Improve the efficiency of your application code. This might not be an option if you do not have access to changing the application code.
- Increase the size of the Shared Pool

```
SQL> select namespace,gethitratio
  2  from v$librarycache;
NAMESPACE            GETHITRATIO
---------------     -----------
SQL AREA            .86928702
TABLE/PROCEDURE     .80073801
BODY                .6
```

Enterprise Manager

The scripts on the slide above can be executed through Enterprise Manager. Under the Database Applications menu run SQL Worksheet. Performance Manager, located under the Diagnostics Menu, can be used as a source to get the GETHITRATIO for the library cache.

Guidelines: Library Cache Reloads

Executes PROC1 —> 1st pin, 1 load
Executes PROC1 —> 2nd pin, no reload
Executes PROC1 —> 3rd pin, no reload
Executes PROC1 —> 4th pin, no reload

→ 4 pins and no reloads

- **Reloads should be less than 1% of the pins:**

```
SQL> select sum(pins) "Executions", sum(reloads)
  2         "Cache Misses", sum(reloads)/sum(pins)
  3  from v$librarycache;
```

- **If the reloads-to-pins ratio is greater than 1%, increase the value of the SHARED_POOL_SIZE parameter.**

How to Get the Reloads-to-Pins Ratio

- V$LIBRARYCACHE view: The view displays whether statements that have already been parsed have been aged out of the cache. The number of reloads should not be more than 1% of the number of pins.
- STATSPACK report: Library Cache Activity section contains the summarized information for the instance. The Instance Activity Stats section contains the detailed statistics of the activities. Some of the statistics to look for include opened cursors cumulative, parse count (failures), parse count (hard), parse count (total).

```
Instance Activity Stats for DB: ED31Instance: ed31Snaps: 1-2
Statistic                     Total   per Second    per Trans
--------------------------- ---------- ------------ -----------
parse count (failures)          2          0.0          2.0
parse count (hard)             26          0.0         26.0
parse count (total)           690          0.6        690.0
```

- report.txt output: This section indicates the same ratio for the period when utlbstat and utlestat ran.

Enterprise Manager

The scripts on the slide above can be executed through Enterprise Manager. Under the Database Applications menu run the SQL Worksheet.

How to Get the Reloads-to-Pins Ratio (continued)

Guidelines

There are two possible reasons for the reloads-to-pins ratio being greater than 1%:

- Though required by successive reexecutions, shared parsed areas have been aged out because of lack of space.
  - SOLUTION: To avoid these frequent reloads, increase the `SHARED_POOL_SIZE` `init.ora` parameter.
- Shared parsed areas are invalidated.
  - SOLUTION: Perform housekeeping (such as creating indexes) during periods of light load.

Invalidations

The number of times objects of the namespace were marked invalid, causing reloads:

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations
  2  from v$librarycache;

SQL> ANALYZE TABLE hr.employees COMPUTE STATISTICS;
```

```
SQL> select count(*) from hr.employees;

SQL> select namespace,pins,reloads,invalidations
  2  from v$librarycache;
```

3-14

When Invalidations Occur

If a schema object is referenced in a SQL statement and that object is later modified in any way, the shared SQL area becomes invalidated (marked as invalid), and the statement must be reparsed the next time it is executed, and therefore reloaded.

For example, when a table, sequence, synonym, or view is re-created, altered, or dropped, or a procedure or package specification is recompiled, all dependent shared SQL areas are invalidated.

Example

```
SQL> select count(*) from hr.employees;
SQL> select namespace,pins,reloads,invalidations from
v$librarycache;
NAMESPACE             PINS    RELOADS INVALIDATIONS
--------------------- ---------- ---------- -------------
SQL AREA              1616         12             0
…
SQL> analyze table hr.employees compute statistics;
Table analyzed.
```

When Invalidations Occur (continued)

```
SQL> select count(*) from hr.employees;
SQL> select namespace,pins,reloads,invalidations from
v$librarycache;
NAMESPACE             PINS    RELOADS INVALIDATIONS
--------------- ---------- ---------- -------------
SQL AREA              1688         14             3
…
```

Sizing the Library Cache

- Define the global space necessary for stored objects (packages, views, and so on).
- Define the amount of memory used by the usual SQL statements.
- Reserve space for large memory requirements in order to avoid misses and fragmentation.
- Pin frequently used objects.
- Convert large anonymous PL/SQL blocks into small anonymous blocks calling packaged functions.

Sizing the Library Cache

The commands required to monitor the shared pool and to pin objects in memory will be discussed later.

Anonymous Blocks

Whenever possible anonymous PL?SQL blocks of any size should be changed to stored procedures.

Cached Execution Plans

- With this feature, the Oracle server preserves the actual execution plan of a cached SQL statement in memory.
- When the SQL statement ages out of the library cache, the corresponding cached execution plan is removed.
- The main benefit of this feature is better diagnosis of query performance.

Cached Execution Plans

This feature enables the Oracle server to retain the execution plans in memory as long as the SQL statement remains in library cache. In pre-Oracle9*i* versions, the execution plan was not retained after the statement was compiled.

The Caches Execution Plans can be viewed using V$SQL_PLAN. This view has similar output, and uses, as Explain Plan, which will be discussed later in this course.

View to Support Cached Execution Plans

- A dynamic performance view, V$SQL_PLAN, can be used to view the actual execution plan information for cached cursors.
- The view contains all of the PLAN_TABLE columns (with the exception of the LEVEL column), in addition to extra columns.
- Columns that are also present in the PLAN_TABLE have the same value as those in V$SQL_PLAN.

ORACLE

3-18

New View to Support Cached Execution Plan

The V$SQL_PLAN view displays the actual execution plan information for a cached cursor. The view contains all of the PLAN_TABLE columns (with the exception of the LEVEL column) and seven extra columns. The columns present in the PLAN_TABLE have the same values as those in the V$SQL_PLAN.

Additional V$SQL_PLAN columns not found in PLAN_TABLE:

- ADDRESS: Cursor parent handle address
- HASH_VALUE: Parent statement hash value in library cache
- CHILD_NUMBER: Number using this execution plan
- DEPTH: Level of the operation in the tree

V$SQL Support For Cached Execution Plans

- V$SQL has a column, PLAN_HASH_VALUE, which references the HASH_VALUE column of V$SQL_PLAN.
- The column information is a hash value built from the corresponding execution plan.
- The column can be used to compare cursor plans the same way the HASH_VALUE column is used to compare cursor SQL texts.

View to Support Cached Execution Plan (continued)

A column, PLAN_HASH_VALUE, in the V$SQL view is a hash value built from the corresponding execution plan. The column can be used to compare cursor plans the same way the HASH_VALUE column is used to compare cursor SQL texts.

Global Space Allocation

Stored objects such as packages and views:

```
SQL> select sum(sharable_mem)
 2   from V$DB_OBJECT_CACHE;
SUM(SHARABLE_MEM)
-----------------
           379600
```

SQL statements:

```
SQL> select sum(sharable_mem)
 2   from V$SQLAREA where executions > 5;
SUM(SHARABLE_MEM)
-----------------
           381067
```


Testing Your Applications

For an existing application, you can set up a test and use the dynamic views to find out how much memory is used. Begin by setting SHARED_POOL_SIZE to a very large value (at the expense of other structures, if necessary), then run the application.

Computation of Shareable Memory Used

For stored objects such as packages and views, use the following query:

```
SQL>  SELECT SUM(sharable_mem)
2     FROM v$db_object_cache
3     WHERE type = 'PACKAGE' or type = 'PACKAGE BODY'
4      OR type = 'FUNCTION' or type = 'PROCEDURE';
```

For SQL statements, you need to query V$SQLAREA after the application has been running for a while. For frequently issued statements, you can use the following query to estimate the amount of memory being used, though this will not include dynamic SQL:

```
SQL> SELECT SUM(sharable_mem)
2       FROM v$sqlarea
3       WHERE executions > 5;
```

Computation of Shareable Memory Used (continued)

You should also allow about 250 bytes in the shared pool per user per open cursor. This can be tested during peak times with the following query:

```
SQL> SELECT SUM(250 * users_opening)
2       FROM v$sqlarea;
```

In a test environment, you can measure shareable memory by selecting the number of open cursors for a test user. You multiply the resulting value by the total number of users:

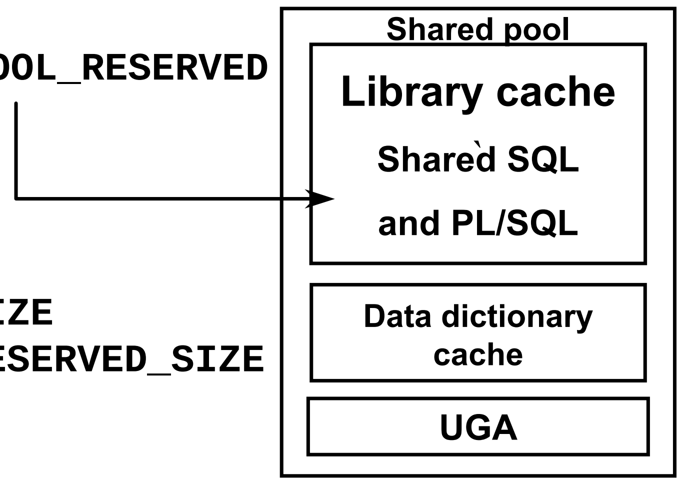
```
SQL> SELECT 250 * value bytes_per_user
2       FROM v$sesstat s, v$statname n
3       WHERE s.statistic# = n.statistic#
4       AND n.name = 'opened cursors current'
5       AND s.sid = 15;
```

Ideally, your application should have a library cache as large as the sum of the numbers above, plus a small allowance for dynamic SQL.

Large Memory Requirements

- **Satisfy requests for large contiguous memory**
- **Reserve contiguous memory within the shared pool**

V$SHARED_POOL_RESERVED



SHARED_POOL_SIZE
SHARED_POOL_RESERVED_SIZE

Why Reserve Space in the Shared Pool?

The DBA can reserve memory within the Shared Pool to satisfy large allocations during operations such as PL/SQL and trigger compilation. Smaller objects will not fragment this reserved area of the Shared Pool. This helps to ensure that the Shared Pool Reserved Area has large contiguous chunks of memory.

Initialization Parameter

The size of the Shared Pool Reserved Area, as well as the minimum size of the objects that can be allocated from the reserved list, are controlled by the `SHARED_POOL_RESERVED_SIZE` initialization parameter, which controls the amount of `SHARED_POOL_SIZE` reserved for large allocations. Set the initial value to 10% of the `SHARED_POOL_SIZE`. If `SHARED_POOL_RESERVED_SIZE` is greater that half of `SHARED_POOL_SIZE`, the server signals an error.

V$SHARED_POOL_RESERVED View

This view helps in tuning the reserved pool and space within the shared pool.

The columns of the view are only valid if the SHARED_POOL_RESERVED_SIZE parameter is set to a valid value:

```
SQL> desc V$SHARED_POOL_RESERVED
 Name      Null?            Type
 --------------------       --------      --------
 FREE_SPACE                               NUMBER
 AVG_FREE_SIZE                            NUMBER
 FREE_COUNT                               NUMBER
 MAX_FREE_SIZE                            NUMBER
 USED_SPACE                               NUMBER
 AVG_USED_SIZE                            NUMBER
 USED_COUNT                               NUMBER
 MAX_USED_SIZE                            NUMBER
 REQUESTS                                 NUMBER
 REQUEST_MISSES                           NUMBER
 LAST_MISS_SIZE                           NUMBER
 MAX_MISS_SIZE                            NUMBER
```

where:

FREE_SPACE	is the total free space in the reserved list
AVG_FREE_SIZE	is the average size of the free memory on the reserved list
MAX_FREE_SIZE	is the size of the largest free piece of memory on the reserved list
REQUEST_MISSES	is the number of times the served list did not have a free piece of memory to satisfy the request, and proceeded to start flushing objects from the LRU list

The following columns in the view contain values that are valid even if the parameter is not set:

- REQUEST_FAILURES
- LAST_FAILURE_SIZE
- ABORTED_REQUEST_THRESHOLD
- ABORTED_REQUESTS
- LAST_ABORTED_SIZE

where:

REQUEST_FAILURES	is the number of times that no memory was found to satisfy a request
LAST_FAILURE_SIZE	is the size of the last failed request

Tuning the Shared Pool Reserved Space

Diagnostic tools for tuning:
- **The V$SHARED_POOL_RESERVED dictionary view**
- **The supplied package and procedure:**
  – **DBMS_SHARED_POOL**
  – **ABORTED_REQUEST_THRESHOLD**

Guidelines:
- **Set the parameter**
  – **SHARED_POOL_RESERVED_SIZE**

Diagnostics with the V$SHARED_POOL_RESERVED View

Statistics from the V$SHARED_POOL_RESERVED view can help you tune the parameters. On a system with ample free memory to increase the SGA, the goal is to have REQUEST_MISSES equal 0, not to have any request failures, or at least to prevent this value from increasing.

Diagnostics with ABORTED_REQUEST_THRESHOLD Procedure

The ABORTED_REQUEST_THRESHOLD procedure, in the DBMS_SHARED_POOL package, enables you to limit the amount of the shared pool to flush prior to reporting an ORA-4031 error, so as to limit the extent of a flush that could occur due to a large object.

Guidelines When SHARED_POOL_RESERVED_SIZE Is Too Small

The reserved pool is too small when the value for REQUEST_FAILURES is more than zero and increasing. To resolve this, you can increase the value of SHARED_POOL_RESERVED_SIZE and SHARED_POOL_SIZE accordingly. The settings you select for these depend on your system's SGA size constraints.

This option increases the amount of memory available on the reserved list without having an effect on users who do not allocate memory from the reserved list. As a second option, reduce the number of allocations allowed to use memory from the reserved list; doing so, however, increases the normal shared pool, which may have an effect on other users on the system.

Guidelines When SHARED_POOL_RESERVED_SIZE Is Too Large

Too much memory may have been allocated to the reserved list if:

- REQUEST_MISS = 0 or not increasing
- FREE_MEMORY = > 50% of the SHARED_POOL_RESERVED_SIZE minimum

If either of these is true, decrease the value for SHARED_POOL_RESERVED_SIZE.

Guidelines When SHARED_POOL_SIZE Is Too Small

The V$SHARED_POOL_RESERVED fixed table can also indicate when the value for SHARED_POOL_SIZE is too small. This may be the case if REQUEST_FAILURES > 0 and increasing.

Then, if you have enabled the reserved list, decrease the value for SHARED_POOL_RESERVED_SIZE. If you have not enabled the reserved list, you could increase SHARED_POOL_SIZE.

Keeping Large Objects

Find those PL/SQL objects that are not kept in the library cache:

```
SQL> select * from v$db_object_cache
  2  where sharable_mem > 10000
  3  and (type='PACKAGE' or type='PACKAGE BODY' or
  4       type='FUNCTION' or type='PROCEDURE')
  5  and  KEPT='NO';
```

Pin large packages in the library cache:

```
SQL> EXECUTE dbms_shared_pool.keep('package_name');
```

3-26

Why and When to Keep Objects

Loading large objects is the primary source of Shared Pool memory fragmentation. Users' response time is affected because of the large number of small objects that need to be aged out from the shared pool to make room. To prevent these situations, keep these large or frequently required objects in the shared pool to make sure that they are never aged out of the shared pool.

- Which objects to keep:
  - Frequently required large procedural objects such as `STANDARD`, `DIUTIL` packages, and those for which shareable memory exceeds a defined threshold
  - Compiled triggers that are executed often on frequently used tables
  - Sequences, because sequence numbers are lost when the sequence is aged out of the shared pool
- When to keep them: Startup time is best, because that prevents further fragmentation.
- Flushing the shared pool using the command `ALTER SYSTEM FLUSH SHARED_POOL` does not flush kept objects.

How to Keep Objects

Use the supplied `DBMS_SHARED_POOL` package and the `KEEP` procedure to keep objects.

To create the package, run the `dbmspool.sql` script. The `prvtpool.plb` script is automatically executed at the end of the dbmspool.sql script. These scripts are not run by `catproc.sql.`

Use the `UNKEEP` procedure to remove pinned objects from the shared pool.

Anonymous PL/SQL Blocks

Find the anonymous PL/SQL blocks and convert them into small anonymous PL/SQL blocks that call packaged functions:

```
SQL> select sql_text from v$sqlarea
  2  where command_type = 47
  3  and length(sql_text) > 500;
```


Eliminating Large Anonymous PL/SQL Blocks

Two Solutions

- Find them and convert them into small anonymous PL/SQL blocks that call packaged functions.
- If an anonymous PL/SQL block cannot be turned into a package, it can be identified in V$SQLAREA and marked KEPT.

You can then keep these blocks in memory, using the appropriate supplied procedure.

```
SQL> declare x number;
 2>   begin x := 5;
 3>   end;
```

Should be written as :

```
SQL>  declare /* KEEP_ME */ x number;
 2>   begin x := 5;
 3>   end;
```

You can locate this statement with the following query:

```
SQL>  select address, hash_value
 2>  from v$sqlarea
 3>  where command_type = 47 and sql_text like '%KEEP_ME%';
```

Eliminating Large Anonymous PL/SQL Blocks (continued)

Then execute the KEEP procedure on the anonymous PL/SQL block identified by the address and hash value retrieved from the previous statement:

```
SQL>  execute dbms_shared_pool.keep('address,hash_value');
```

Note: Although we show how to pin an anonymous PL/SQL block in memory, it is recommended that all anonymous PL/SQL blocks be modified to stored procedures.

Other Parameters Affecting the Library Cache

- **OPEN_CURSORS**
- **CURSOR_SPACE_FOR_TIME**
- **SESSION_CACHED_CURSORS**
- **CURSOR_SHARING**

Initialization Parameters

The following parameters also affect the library cache:

- OPEN_CURSORS: This parameter defines the number of cursors referencing private SQL areas allocated to the user's process. A private SQL area continues to exist until the cursor is closed and therefore still exists after the completion of the statement.
  To take advantage of additional memory available for shared SQL areas, you may need to increase the number of cursors permitted for a session. Application developers should close unneeded open cursors to conserve system memory. The default value is 50.
- CURSOR_SPACE_FOR_TIME: This is a boolean parameter that defaults to FALSE. If you set it to TRUE, you choose to use space to gain time; shared SQL areas are not aged out until the cursor referencing them is closed. Therefore, make sure that there is free memory and no cache misses. Do not change this parameter unless the value of RELOADS in V$LIBRARYCACHE is consistently 0. If your application uses Forms, or any dynamic SQL, leave the setting at FALSE.

Initialization Parameters (continued)

- SESSION_CACHED_CURSORS: This parameter helps in situations in which a user repeatedly parses the same statements. This occurs in Forms applications when users often switch between forms; all the SQL statements opened for a form are closed when you switch to another one. The parameter causes closed cursors to be cached within the session. Therefore, any subsequent call to parse the statement will bypass the parse phase. (This is similar to HOLD_CURSORS in the precompilers.)
  To check that your setting is efficient, compare the "session cursor cache hits" and "parse count" session statistics in V$SESSTAT for a typical user session. If few parses result in hits, you might increase the number. Remember that this increases overall demands on memory.
  The default is 0, which means no caching.
- CURSOR_SHARING: The default behavior of this parameter allows statements to share cursors only if the statements are identical in every way. The value can be changed in order to allow statements that are similar to share cursors, if they have the same result set. This allows statements that are identical, except for a literal value, to use the same cursor, thus preventing unnecessary parsing. Note that bind variables share cursors regardless of the setting of CURSOR_SHARING.

Tuning The Data Dictionary Cache

Use V$ROWCACHE to get information on the Data Dictionary Cache

- **Content:**
  - **Definitions of dictionary objects**
- **Terminology:**
  - **`GETS`: Number of requests on objects**
  - **`GETMISSES`: Number of requests resulting in cache misses**
- **Tuning:**
  - **Avoid dictionary cache misses**

3-32

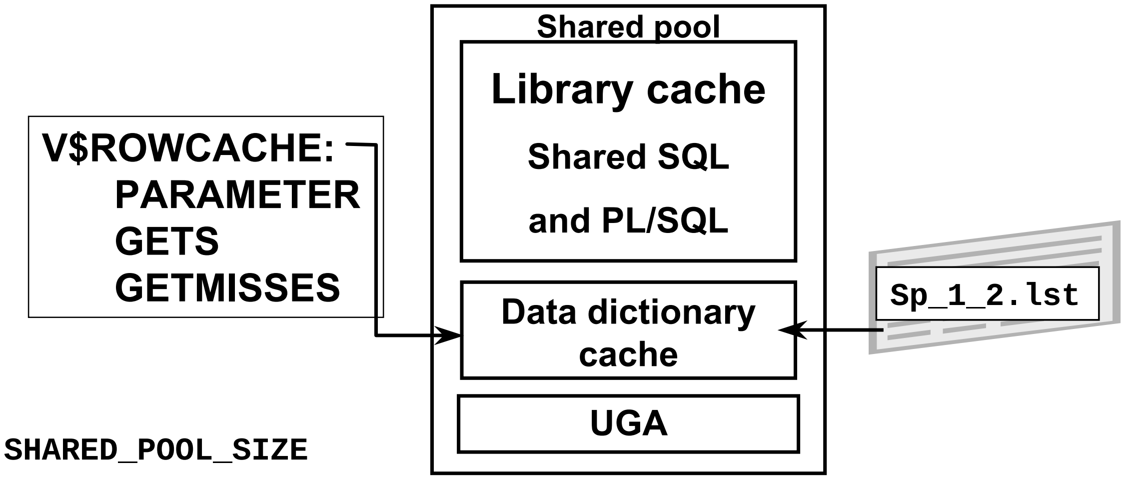
Three Key Columns of V$ROWCACHE

- `PARAMETER:` Gives the name of the Data Dictionary Cache that is being reported.
- `GETS`: Shows the total number of requests for information on the corresponding item (for example, in the row that contains statistics for file descriptions, this column has the total number of requests for file description data)
- `GETMISSES`: Shows the number of data requests resulting in cache misses

Goal

Misses on the data dictionary cache are to be expected in some cases. Upon instance startup, the data dictionary cache contains no data, so any SQL statement issued is likely to result in cache misses. As more data is read into the cache, the likelihood of cache misses should decrease. Eventually, the database should reach a "steady state" in which the most frequently used dictionary data is in the cache. At this point, very few cache misses should occur. To tune the cache, examine its activity only after your application has been running for some time.

Diagnostic Tools for Tuning the Data Dictionary Cache



Monitoring the Dictionary Cache

Use the `V$ROWCACHE` view. The columns of most interest are shown in the following table:

Column	Description
PARAMETER	Categories of data dictionary items
GETS	Requests for information on that category
GETMISSES	Requests resulting in cache misses

Sizing

You can size the dictionary cache only indirectly with the `SHARED_POOL_SIZE` parameter. The algorithm for allocating shared pool space gives preference to the dictionary cache.

Measuring the Dictionary Cache Statistics

In the Dictionary Cache Stats section of STATSPACK:

- **Percent misses should be very low:**
  - **< 2% for most data dictionary objects**
  - **< 15% for the entire data dictionary cache**
- **Cache Usage is the number of cache entries being used.**
- **Pct SGA is a percentage of usage to allocated size for that cache.**

ORACLE

Measuring Dictionary Cache Statistics

The report by the STATSPACK utility contains a section on the statistics related to the dictionary cache. An example output of the section is here:

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage	Pct SGA
dc_free_extents	215	26.5	36	0.0	150	13	57
dc_histogram_defs	29	72.4	0		0	109	92
dc_object_ids	236	6.4	0		0	373	99
dc_objects	263	9.9	0		0	513	99
dc_profiles	2	0.0	0		0	1	10
dc_rollback_segments	40	0.0	0		0	6	33
dc_segments	138	15.2	0		39	184	97
dc_tablespaces	123	0.0	0		0	6	86
dc_used_extents	57	63.2	0		57	38	67
dc_user_grants	6	16.7	0		0	5	16
dc_usernames	159	0.6	0		0	8	38
dc_users	62	1.6	0		0	7	78

…………

Tuning the Data Dictionary Cache

Keep the percentage of the sum of GETMISSES to the sum of GETS less than 15%:

```
SQL> select parameter, gets, getmisses
 2   from v$rowcache;
PARAMETER                   GETS      GETMISSES
-------------------------- --------- ---------
dc_objects                  143434      171
dc_synonyms                 140432      127
```


Goal for a Good Ratio

The percentage of the sum of all GETMISSES to the sum of all GETS should be less than 15% during normal running.

If it is higher, consider increasing SHARED_POOL_SIZE, and examine the application to see if there are objects that are constantly being created and dropped.

You cannot hope to achieve a zero value for GETMISSES, because an object definition must be loaded into the cache the first time after startup that a server needs it.

Guidelines: Dictionary Cache Misses

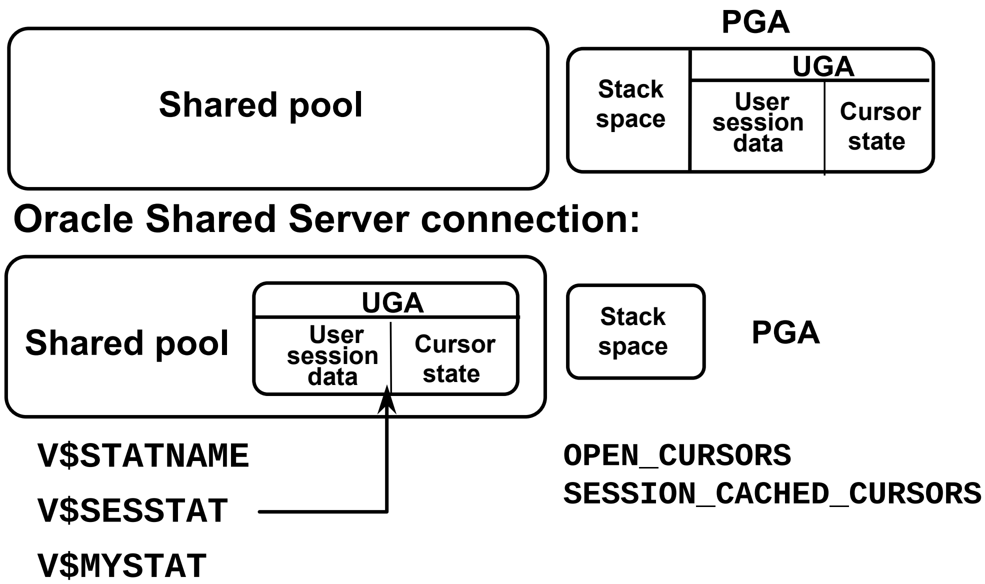
STATSPACK report output:

Cache	Get Requests	Pct Miss	Scan Reqs	Pct Miss	Mod Reqs	Final Usage	Pct SGA
dc_free_extents	2	0.0	0		0	3	3
dc_histogram_defs	11	0.0	0		0	49	92
dc_object_ids	19	0.0	0		0	440	98

Ratio from the STATSPACK Report Output

If the STATSPACK report output indicates a high GET_MISS/GET_REQ ratio for a number of items, the SHARED_POOL_SIZE should be increased.

UGA and Oracle Shared Server



3-37

The User Global Area

If you use the Oracle Shared Server, and the large pool is not configured, then user session and cursor state information is stored in the shared pool instead of in private user memory. Sort areas and private SQL areas are included in the session information. This is because shared servers work on a per-call basis, so any server may need access to any user's information. This part of the shared pool is called the user global area (UGA).

The total memory requirement for the Oracle Shared Server is no larger than if you use dedicated servers. You may need to increase `SHARED_POOL_SIZE`, but your private user memory is lower.

If you are using shared servers set the Large Pool, then the UGA will be stored there and not inside the Shared Pool.

Sizing the User Global Area

UGA space used by your connection:

```
SQL> select SUM(value) ||'bytes' "Total session memory"
  2  from V$MYSTAT, V$STATNAME
  3  where name = 'session uga memory'
  4  and v$mystat.statistic# = v$statname.statistic#;
```

UGA space used by all Oracle Shared Server users:

```
SQL> select SUM(value) ||'bytes' "Total session memory"
  2  from V$SESSTAT, V$STATNAME
  3  where name = 'session uga memory'
  4  and v$sesstat.statistic# = v$statname.statistic#;
```

Maximum UGA space used by all users:

```
SQL> select SUM(value) ||'bytes' "Total max memory"
  2  from V$SESSTAT, V$STATNAME
  3  where name = 'session uga memory max'
  4  and v$sesstat.statistic# = v$statname.statistic#;
```

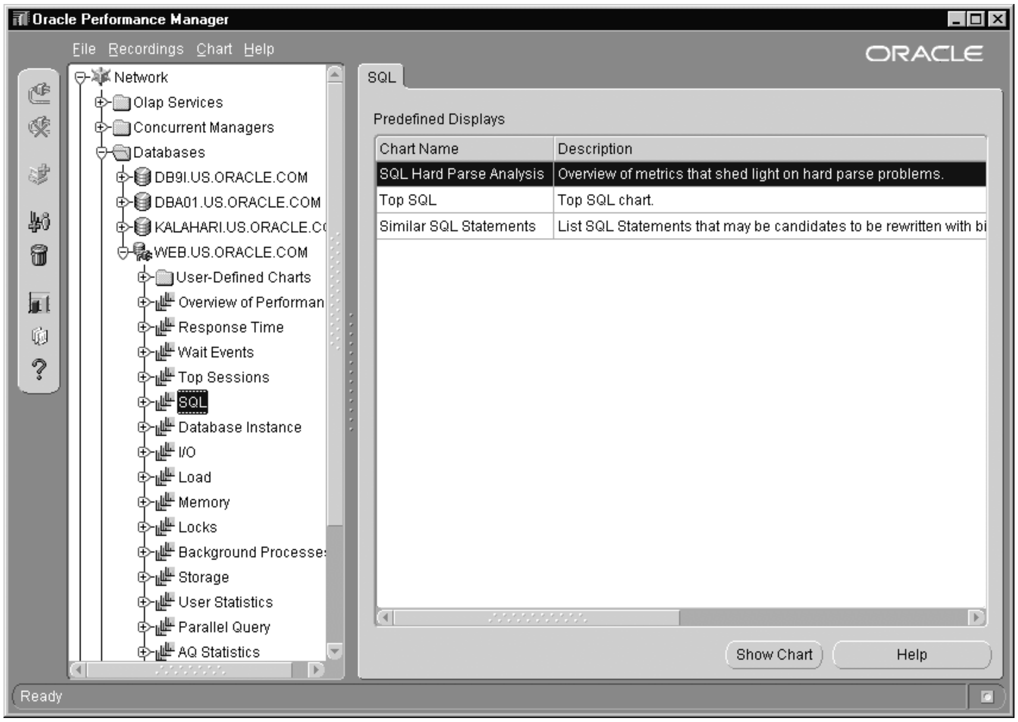
ORACLE

3-38

Required Space Measurement

For all Oracle Shared Server connections, you need to compute the amount of space required for all shared server users to put their session memory in the shared pool.

Performance Manager: Shared Pool Statistics



Performance Manager: Shared Pool Statistics

There are many charts that will give information regarding the Shared Pool Statistics. Some of these charts include:

Memory: SGA Overview

The charts give a breakdown of the SGA as a whole, and what portion is allocated to the Shared Pool. Also the amount of free space in the Shared Pool is shown.

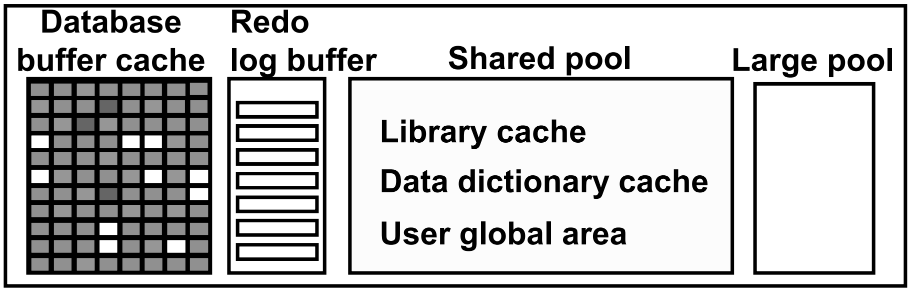
Database Instance: Library Cache Hit %

This value is the percentage of times that a requested SQL statement is found already in memory.

Top Sessions: Top Sessions

This chart shows the top ten sessions that are causing Physical Reads, Logical Reads, Commit counts and many more session statistics. Each of these statistics can be sorted in ascending or descending order.

Large Pool



- Can be configured as a separate memory area in the SGA, used for memory with:
  - I/O server processes: `DBWR_IO_SLAVES`
  - Backup and restore operations
  - Session memory for the shared servers
  - Parallel query messaging
- Is useful in these situations to avoid performance overhead caused by shrinking the shared SQL cache
- Is sized by the parameter `LARGE_POOL_SIZE`

Existence of the Large Pool

The large pool must be explicitly configured. The memory of the large pool does not come out of the shared pool, but directly out of the SGA, thus adding to the amount of shared memory the Oracle server needs for an instance at startup.

Advantages of the Large Pool

The large pool is used to provide large allocations of session memory for:

- I/O server processes
- Backup and restore operations

The memory for backup and restore operations and for I/O server processes is allocated in buffers of a few hundred kilobytes. The large pool is better able to satisfy such requests than is the shared pool.

Oracle Shared Server: By allocating session memory from the large pool for the Oracle Shared Server, the Oracle server can use the shared pool primarily for caching shared SQL and avoid additional contention, with the dependant performance reduction, due to the reduced space available for the cached SQL.

Summary

In this lesson, you should have learned how to:

- Size shared SQL and PL/SQL areas (library cache)
- Size data dictionary cache or row cache
- Size the large pool
- Size the user global area, if connections are Oracle Shared Server connections, unless the large pool is configured

Practice 3

The objective of this practice is to use diagnostic tools to monitor and tune the shared pool. Through out this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect using 'system/manager" and check the size of the shared pool.
   `SQL> show parameter shared_pool`
2. Connect as perfstat/perfstat user, execute the `$HOME/STUDENT/LABS/snap.sql` script to collect initial snapshot of statistics, and note the snapshot number by
3. To simulate user activity against the database open two operating system sessions. In session 1 connect as hr/hr and run the `$HOME/STUDENT/LABS/lab03_03_1.sql` script. In the second session connect as hr/hr and run the `$HOME/STUDENT/LABS/lab03_03_2.sql` script.
4. Connect as "system/manager" and measure the pin-to-reload ratio for the library cache by querying `v$librarycache`. Determine if it is a good ratio or not using the dynamic view.
5. Connect as "system/manager" and measure the get-hit ratio for the data dictionary cache by querying `v$rowcache`. Determine if it is a good ratio or not using the dynamic view.
6. Connect as perfstat/perfstat and run the `$HOME/STUDENT/LABS/snap.sql` script to collect a statistic snap shot and obtain the snapshot number. Record this number.
7. As user perfstat/perfstat obtain the statistics report between the two recorded snapshot IDs (from questions 2 and 6) by running the `$HOME/STUDENT/LABS/spreport.sql` script.
8. Analyze the generated report in the current directory (named `sp_3_5.lst` in the previous example). What would you consider to address if the library hit ratio (found under the heading "Instance Efficiency Percentages") is less than 98%?
9. Determine which packages, procedures, and triggers are pinned in the shared pool by querying `v$db_object_cache`.
10. Connect using "sys/oracle as sysdba" and pin one of the Oracle supplied packages that needs to be kept in memory, such as SYS.STANDARD using the `DBMS_SHARED_POOL.KEEP` procedure, that is created by running the `$ORACLE_HOME/rdbms/admin/dbmspool.sql` script.
11. Determine the amount of session memory used in the shared pool for your session by querying the `v$mystat` view. Limit the output by including the clause `where name = 'session uga memory'`
12. Determine the amount of session memory used in the shared pool for all sessions, using `V$SESSTAT` and `V$STATNAME` views.

4

Sizing the Buffer Cache

Objectives

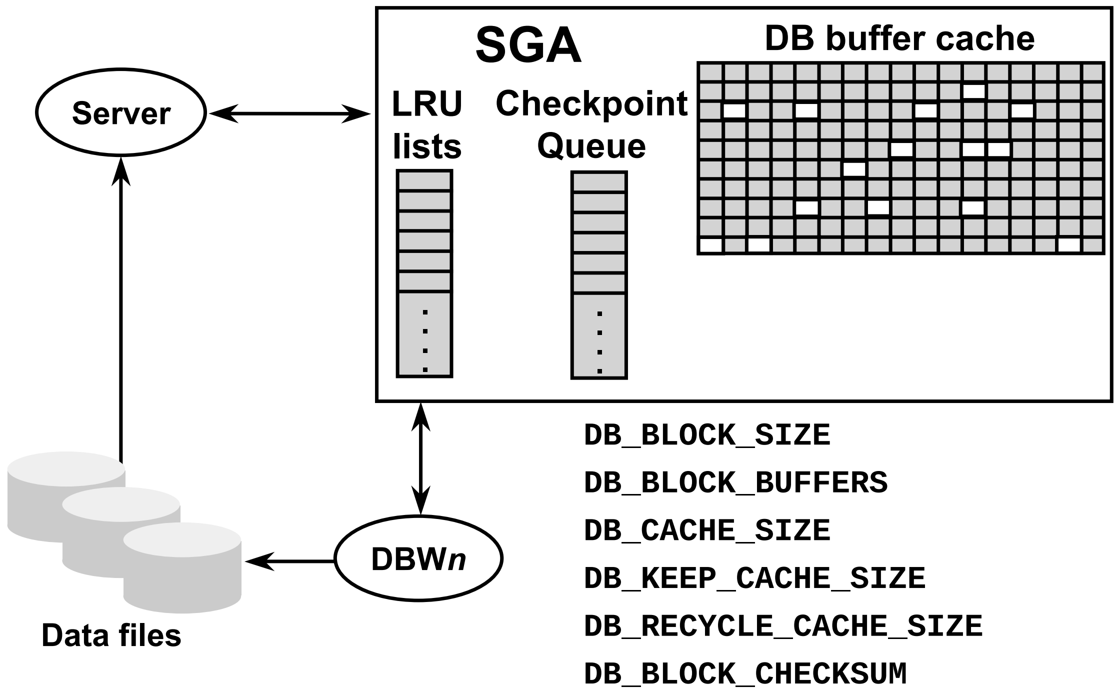
After completing this lesson, you should be able to do the following:

- Describe how the buffer cache is used by different Oracle processes
- List the tuning issues related to the buffer cache
- Monitor the use of buffer cache, and the different pools within buffer cache
- Implement dynamic SGA allocation
- Set the `DB_CACHE_ADVICE` parameter
- Create and size multiple buffer pools
- Detect and resolve free list contention
- Configure the instance to use I/O slaves
- Configure and use multiple DBWn processes

ORACLE

4-2

Overview



Buffer Cache Characteristics

The buffer cache holds copies of the data blocks from the data files. Because the buffer cache is a part of the SGA, these blocks can be shared by all users. The server processes read data from the data files into the buffer cache. To improve performance, the server process sometimes reads multiple blocks in a single read. The DBW*n* process writes data from the buffer cache into the data files. To improve performance, DBW*n* writes multiple blocks in a single write.

At any given time, the buffer cache may hold multiple copies of a single database block. Only one current copy of the block exists, but server processes may need to construct read-consistent copies, using rollback information, to satisfy queries.

In Oracle9*i*, the buffer caches can be individually sized with the following parameters:

- `DB_CACHE_SIZE` specifies the size of default buffer pool in bytes.
- `DB_KEEP_CACHE_SIZE` specifies the size of keep buffer pool in bytes.
- `DB_RECYCLE_CACHE_SIZE` specifies the size of recycle buffer pool in bytes.

The buffer pools in Oracle9*i* can be resized dynamically.

In Oracle8*i*, the buffer cache has the following characteristics.

- It is sized using the `DB_BLOCK_BUFFERS` parameter. This parameter specifies the number of blocks in the buffer cache. To find the size of the cache in bytes, multiply `DB_BLOCK_BUFFERS` by `DB_BLOCK_SIZE`.

Buffer Cache Characteristics (continued)

- The buffers in the buffer cache are managed using two lists:
  - The least recently used (LRU) list is used to keep the most recently accessed blocks in memory. The blocks on the list are organized from the most recently used (MRU) to the least recently used.
  - The checkpoint queue points to blocks in the buffer cache that have been modified but not written to disk.
- Buffers in the buffer cache can be in one of four states:
  - Free/unused - meaning the buffer is empty because the instance just started.
  - Pinned - meaning the buffer contents are being read/written by several sessions and other sessions may be waiting to access those same blocks contained in the buffers.
  - Clean - meaning the buffer is now unpinned and a candidate for immediate aging out if the current contents (data block) are not referenced again. The contents are either in synch with disk or the buffer was used to generate / hold an old snapshot of the data (CR block).
  - Dirty - buffer is no longer pinned but the contents (data block) have changed and must be flushed to disk by DBWn before it can be aged out.

Server processes use the buffers in the buffer cache, but the `DBW`*`n`* process makes buffers in the cache available by writing changed buffers back to the data files.

The Least Recently Used list monitors the usage of buffers. The buffers are sorted in accordance with the number of times that they are used. Thus, buffers that are frequently used will be found at the most recently used end, whereas those buffers that are least used are placed at the least recently used end, where they are available for being overwritten by incoming blocks. Incoming blocks are copied to a buffer from the least recently used end, which is then moved to the middle of the list. From here the buffer will work its way up or down the list depending on usage.

If the parameter `DB_BLOCK_CHECKSUM` is set to `TRUE`, then every write is given a checksum number, and the write is confirmed. This will therefore add to the performance overhead, making the write slower.

Buffer Cache Sizing Parameters in Oracle9*i*

- The buffer cache can consist of independent subcaches for buffer pools and for multiple block sizes.
- The `DB_BLOCK_SIZE` parameter determines the primary block size, which is the block size used for the `SYSTEM` tablespace and the primary buffer caches (Recycle, Keep and Default).
- The following parameters define the sizes of the caches for buffers for the primary block size:
  - `DB_CACHE_SIZE`
  - `DB_KEEP_CACHE_SIZE`
  - `DB_RECYCLE_CACHE_SIZE`

Buffer Cache Size Parameters

- Oracle9*i* supports multiple block sizes. The block size for the `SYSTEM` tablespace, specified during database creation, is referred to as the primary block size. Other tablespaces may be of different block sizes.
- For each block size there can be three buffer pools, Keep, Recycle and the Default. These different buffer caches will be dealt with in this lesson. The size of the DEFAULT buffer pool for buffers with the primary block size is defined by the parameter `DB_CACHE_SIZE`. The KEEP and RECYCLE buffer pools may be configured for the primary block size using the `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE` parameters, respectively.
- The values for the `DB_CACHE_SIZE, DB_KEEP_CACHE_SIZE`, and `DB_RECYCLE_CACHE_SIZE` parameters are specified in units of memory (KB or MB), not in number of blocks.
- The Keep and Recycle caches, defined by `DB_KEEP_CACHE_SIZE` and `DB_RECYCLE_CACHE_SIZE respectively,` do not come out of the default pool defined by `DB_CACHE_SIZE`. The values of these parameters are therefore independent of one another.

Dynamic SGA Feature in Oracle9*i*

- **The dynamic SGA feature implements an infrastructure to allow the server to change its SGA configuration without shutting down the instance.**
- **SGA is limited by `SGA_MAX_SIZE`.**
- **A dynamic SGA provides an SGA that will grow and shrink in response to a DBA command.**

Dynamic SGA

The SGA has always been a static allocation of memory, which was shared across all threads of execution. Beginning with Oracle9*i*, the dynamic SGA infrastructure allows for the sizing of the buffer cache, and shared pool without having to shut down the instance, modify the initialization parameter file, and restart the instance.

In addition, the dynamic SGA infrastructure allows limits to be set at run time on how much physical memory is used for the SGA. The parameter the limits the memory is SGA_MAX_SIZE. This amount of memory is allocated at startup of the instance, regardless of whether the individual components utilize the entire amount of memory.

You should configure instances to start with less than the maximum amount of memory the operating system makes available, and allow the DBA to modify the SGA components as needed.

Unit of Allocation in the Dynamic SGA

- **In the dynamic SGA model, the unit of memory allocation is called a granule.**
- **SGA memory is tracked in granules by SGA components.**
- **A granule is a unit of contiguous virtual memory allocation.**
- **Use `V$BUFFER_POOL` to monitor size of the buffer caches.**

4-7

ORACLE

Unit of Allocation in the Dynamic SGA

The columns in `V$BUFFER_POOL` are:

- `ID` : Buffer pool ID number
- `NAME` : Buffer pool name
- `BLOCK_SIZE` : Block size for buffers in this component
- `RESIZE_STATE` : Current state of the resize operation (values `STATIC`, `ALLOCATING`, `ACTIVATING`, or `SHRINKING`)
- `CURRENT_SIZE` : Present size of the subcache in megabytes
- `BUFFERS` : Current instantaneous number of buffers
- `TARGET_SIZE` : New size, in bytes, is shown if a resize is in progress, if the pool is `STATIC` then the current size is shown.
- `TARGET_BUFFERS` : New size, in buffer, is shown if a resize is in progress, if the pool is `STATIC` then the current size is shown.
- `PREV_SIZE` : Previous buffer pool size
- `PREV_BUFFERS` : Previous number of buffers
- `LO_BNUM, HI_BNUM , LO_SETID, HI_SETID, SET_COUNT`: Obsolete columns

The number of buffers is worked out by taking the size in bytes, and dividing by the buffer size of the relevant cache.

Granule

- SGA components are allocated and deallocated in units of contiguous memory called granules.
- The size of a granule depends on the estimated total SGA:
  – 4 MB if the estimated SGA size is less than, or equal to,128 MB
  – 16 MB otherwise

Memory Allocation

When allocating SGA structures, the value requested (for example DB_CACHE_SIZE) is rounded up in order to make in integer number of granules.

Allocating Granules at Startup

- At instance startup, the Oracle server requests SGA_MAX_SIZE bytes of address space in memory.
- As startup continues, each component will attempt to acquire the number of granules assigned.
- The minimum SGA configuration is three granules:
  - One granule for fixed SGA (includes redo buffers)
  - One granule for the buffer cache
  - One granule for the shared pool

4-9

Allocating Granules at Startup

If the sum of all the memory used by the SGA components exceeds the amount allotted by SGA_MAX_SIZE, then the value of SGA_MAX_SIZE is adjusted to meet the memory requirement.

Adding Granules to Components

- A DBA can dynamically increase memory allocation to a component by issuing an `ALTER SYSTEM` command.
- Increase of the memory use of a component succeeds only if there are enough free granules to satisfy the request.
- Memory granules are not freed automatically from another component in order to satisfy the increase.
- Decreasing the size of a component is possible, but only if the granules being released are unused by the component.

4-10

```
SQL> show parameter db_cache_size;
NAME                                TYPE        VALUE
------------------------------- ----------- ------------------
db_cache_size                        big integer 4194304

SQL> alter system set db_cache_size=8M;
System altered.
SQL> show parameter db_cache_size;
NAME                                   TYPE      VALUE
------------------------------- ----------- ------------------
db_cache_size                       big integer 8388608
```

Dynamic Buffer Cache Size Parameters

- Parameters that specify the size of buffer cache components are dynamic, and can be changed while the instance is running by means of the `ALTER SYSTEM` command:

```
ALTER SYSTEM SET DB_CACHE_SIZE = 1100M;
```

- Each parameter is sized independently.
- New cache sizes are set to the next granule boundary.
- The allocation size has the following limits:
  - Must be an integer multiple of the granule size.
  - The total SGA size cannot exceed `SGA_MAX_SIZE`.
  - `DB_CACHE_SIZE` can never be set to zero.

ORACLE

4-11

Dynamic Buffer Cache Size Parameters

The buffer cache sizing parameters are now dynamic. They can be changed while the instance is running using the `ALTER SYSTEM` command. The two components that can be resized are the shared pool, and the buffer cache. The buffer cache can be divided into caches for different database block sizes, which can be further divided into three pools default, keep and recycle. Each of these buffer caches can be resized individually.

Example: Increasing the Size of an SGA Component

- **Initial parameter values:**
  - `SGA_MAX_SIZE = 128M`
  - `DB_CACHE_SIZE = 88M`
  - `SHARED_POOL_SIZE = 32M`
- `ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`
  - **Error message indicating insufficient memory**
- `ALTER SYSTEM SET DB_CACHE_SIZE = 56M;`
- `ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`
  - **Error message indicating insufficient memory.**
  - **Check scoreboard to see if shrink has completed.**
- `ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;`
  - **The statement is now processed.**

Increasing the Size of an SGA Component

A database administrator can increase the size of the shared pool, or any buffer cache component, by issuing an ALTER SYSTEM command. The new size is rounded up to the nearest multiple of the granule size.

Increasing the memory use of a component with an ALTER SYSTEM command succeeds if there are enough free granules (SGA_MAX_SIZE minus current SGA_SIZE) to satisfy the request. The server does not start freeing another component's granules for adding. Instead, the database administrator must ensure the instance has enough free granules to satisfy the increase of a component's granule use. If the current SGA memory is less than SGA_MAX_SIZE, then the server is free to allocate more granules until the SGA size reaches SGA_MAX_SIZE.

In the example on the slide, memory is made available for the shared pool, by shrinking the buffer cache. An insufficient memory error can still result (as seen from the second error on the slide) if the instance has not completed the shrinking of the buffer cache when the granule is required for the shared pool to increase. Use the view V$BUFFER_POOL to confirm that a shrink has completed, before increasing the size of another.

When resizing SGA components remember that there is a portion of fixed memory, used by Oracle structures like the redo log buffer, and the large pool. This fixed memory will have a minimum of one granule.

Deprecated Buffer Cache Parameters

- **Three parameters have been deprecated and will be maintained for backward compatibility.**
  - **DB_BLOCK_BUFFERS**
  - **BUFFER_POOL_KEEP**
  - **BUFFER_POOL_RECYCLE**
- **These parameters cannot be combined with the dynamic size parameters.**

Deprecated Buffer Cache Parameters

The Oracle8i buffer cache size parameters DB_BLOCK_BUFFERS, BUFFER_POOL_KEEP, and BUFFER_POOL_RECYCLE are maintained for backward compatibility (so users can continue to use pre-Oracle9*i* parameter files), but are deprecated and will be made obsolete in the future.

The syntax for the BUFFER_POOL_SIZE parameters allowed the user to optionally specify the number of LRU latches for the buffer pool in addition to the number of buffers in the buffer pool. These latches were allocated out of the total number of latches specified in DB_BLOCK_LRU_LATCHES. Because DB_BLOCK_LRU_LATCHES is now obsolete, the specification of the number of LRU latches for the buffer pool, if provided, is ignored.

These parameters will continue to be static parameters. Furthermore, these parameters cannot be combined with the dynamic size parameters.

If these parameters are combined with new parameters at the start of the instance, you would get an error as follows:

```
ORA-00381: cannot use both new and old parameters for buffer
cache size specification
```

Dynamic Buffer Cache Advisory Parameter

- **The buffer cache advisory feature enables and disables statistics gathering for predicting behavior with different cache sizes.**
- **The information provided by these statistics can help DBAs size the buffer cache optimally for a given workload.**
- **The buffer cache advisory is enabled by means of the `DB_CACHE_ADVICE` initialization parameter:**
  - **This parameter is dynamic, and can be changed using `ALTER SYSTEM`.**
  - **Three values are allowed: `OFF`, `ON`, and `READY`.**

Dynamic Buffer Cache Advisory Parameter Values

- `OFF`: Advisory is turned off and the memory for the advisory is not allocated
- `READY`: Advisory is turned off, but the memory for the advisory remains allocated. Allocating the memory before the advisory is actually turned on avoids the risk of an ORA-4031 error (inability to allocate from the shared pool). If the parameter is switched to this state from `OFF`, an ORA-4031 error may be generated.
- `ON`: Advisory is turned on and both CPU and memory overhead is incurred. Attempting to set the parameter to this state when it is in the `OFF` state may lead to an ORA-4031 error when the parameter is switched to `ON`. If the parameter is in `READY` state it can be set to `ON` without error because the memory is already allocated.

View to Support Buffer Cache Advisory

- Buffer cache advisory information is collected in the V$DB_CACHE_ADVICE view.
- The view contains different rows that predict the estimated number of physical reads for different cache sizes.
- The rows also compute a physical read factor, which is the ratio of the number of estimated reads to the number of actual reads.

View to Support Buffer Cache Advisory

V$DB_CACHE_ADVICE columns:

- ID: Buffer pool ID (ranges from 1–8)
- NAME: Buffer pool name
- BLOCK_SIZE: Block size in bytes for buffers in this pool. Possible values are the standard block size, and non-standard block sizes in powers of two: 2048, 4096, 8192, 16384, or 32768.
- ADVICE_STATUS: Status of the advisory.
- SIZE_FOR_ESTIMATE: Cache size for prediction (in megabytes).
- BUFFERS_FOR_ESTIMATE: Cache size for prediction (in terms of buffers).
- ESTD_PHYSICAL_READ_FACTOR: Physical read factor for this cache size; ratio of number of estimated physical reads to the number of reads in the real cache. If there are no physical reads into the real cache, the value of this column is null.
- ESTD_PHYSICAL_READS: Estimated number of physical reads for this cache size.

Using V$DB_CACHE_ADVICE

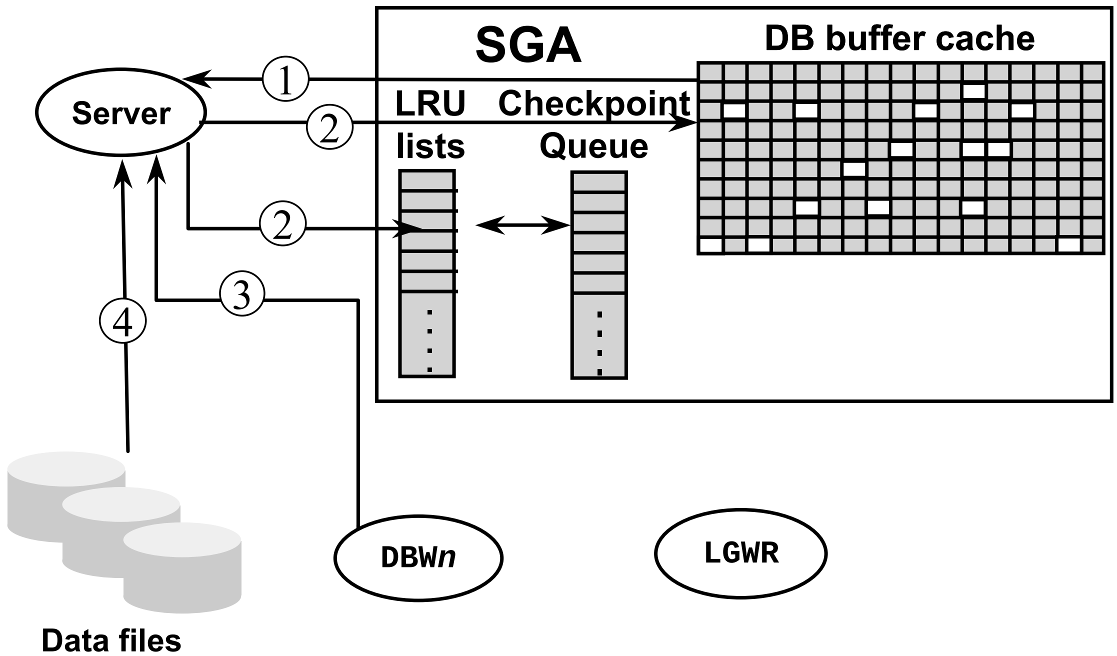
- **SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor, estd_physical_reads**
- **FROM V$DB_CACHE_ADVICE WHERE name = 'DEFAULT' AND block_size = (**
  - **SELECT value FROM V$PARAMETER**
  - **WHERE name = 'db_block_size') AND advice_status = 'ON';**

Using the V$DB_CACHE_ADVICE View

The following output shows that if the cache was 212 MB, rather than the current size of 304 MB, the estimated number of physical reads would be 17 million (17,850,847). Increasing the cache size beyond its current size would not provide a significant benefit.

```
                                        Estd Phys    Estd Phys
 Cache Size (MB)        Buffers Read Factor            Reads
---------------- ------------------------ ------------
(10%)         30          3,802       18.70  192,317,943
...........
             182         22,812        2.50   25,668,196
             212         26,614        1.74   17,850,847
             243         30,416        1.33   13,720,149
             273         34,218        1.13   11,583,180
(Current)    304         38,020        1.00   10,282,475
             334         41,822         .93    9,515,878
             364         45,624         .87    8,909,026
             395         49,426         .83    8,495,039
             424         53,228         .79    8,116,496
(150%)       456         57,030         .76    7,824,764
...
```

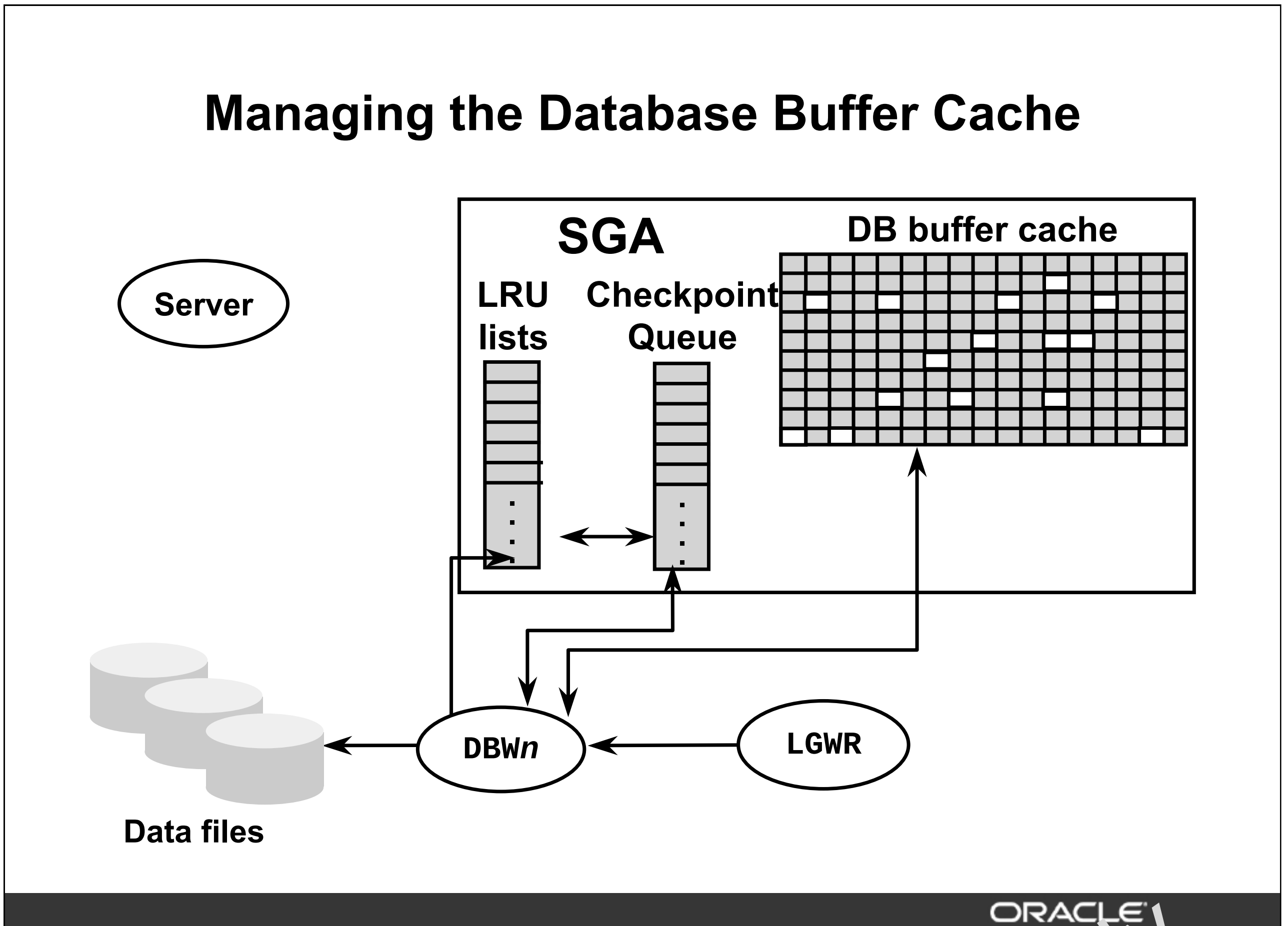
Managing the Database Buffer Cache



The Server Process and the Database Buffer Cache

When a server needs a block, it follows these steps to read the block:

1. First, the server checks whether the required block is available in the buffer cache using a hash function. If the buffer is found, it is moved to another point in the LRU list away from the LRU end. This is a logical read, because no actual I/O took place. If the buffer is not found in the buffer cache, the server process has to read the block from the data file.
2. Before reading from the data file, the server process searches the LRU list for a free buffer. All buffers that have been modified by a server process, are put on the checkpoint queue for copying back to disk during a checkpoint.
3. If the checkpoint queue exceeds its size threshold, the server signals `DBWn` to flush dirty buffers from the data buffer cache. If the server cannot find a free buffer within a search threshold, it signals `DBWn` to flush.
4. After a free buffer is found, the server reads the block from the data file into the free buffer in the database buffer cache. Oracle server process moves the buffer away from the LRU end of the LRU list.
5. If the block is not read consistent, the server rebuilds an earlier version of the block from the current block and rollback segments.



4-18

The DBW*n* Process and the Database Buffer Cache

DBW*n* manages the buffer cache by writing dirty blocks to the data files to ensure that there are free blocks for servers. DBW*n* responds to different events in the instance.

1. Checkpoint Queue Exceeds Threshold: A server process finds that the checkpoint queue has exceeded its size threshold, so it signals DBW*n* to flush. DBW*n* writes out the buffers on the checkpoint queue.
2. Search Threshold Exceeded: A server process that cannot find a free buffer on the LRU list within the search threshold signals DBW*n* to flush checkpoint queue. DBW*n* writes out dirty buffers directly from the checkpoint queue.
3. LGWR Signals a Checkpoint: When LGWR signals that a checkpoint has occurred, DBW*n* copies dirty buffers from the checkpoint queue to disk.
4. `Alter Tablespace Offline Temporary` or `Alter Tablespace Begin Backup`: When a tablespace is altered `offline temporary` or its online backup is started, DBW*n* copies the dirty buffers for that tablespace from the checkpoint queue to disk.
5. Drop Object: When an object is dropped, DBW*n* first flushes the objects dirty buffers to disk.
6. Clean Shutdown (Normal, Immediate, or Transactional)

Tuning Goals and Techniques

- **Tuning goals:**
  - **Servers find data in memory**
  - **No waits on the buffer cache**
- **Diagnostic measures**
  - **Wait events**
  - **Cache hit ratio**
  - **`V$DB_CACHE_ADVICE`**
- **Tuning techniques:**
  - **Reduce the number of blocks required by SQL statements Increase buffer cache size**
  - **Use multiple buffer pools**
  - **Cache tables**
  - **Bypass the buffer cache for sorting and parallel reads**

ORACLE

4-19

Tuning Goals

Because physical I/O takes significant time and increases CPU demand, Oracle server performance is improved when the servers find most of the blocks that they need in memory. The statistic that measures the performance of the database buffer cache is the cache hit ratio. This statistic is the ratio of the number of blocks found in memory to the number of blocks accessed. When the database buffer cache is too small, the system is slower because it is performing too many I/Os.

Diagnostic Measures

To effectively monitor the usage of the buffer cache, you can use the following mechanisms:

- Check for wait events in V$SYSTEM_EVENT, V$SESSIONS_EVENT and V$SESSION_WAIT
- Measure the cache hit ratio: Use the `V$SYSSTAT` view, or the Stats Pack utility or the `utlbstat.sql` and `utlestat.sql` scripts.
- Use the `V$DB_CACHE_ADVICE` view

Tuning Techniques

The DBA monitors the buffer cache by:

(A)Monitoring the wait events

(B)calculating the cache hit ratio from statistics collected by the Oracle server.

Tuning Techniques (continued)

To improve the cache hit ratio, the DBA can:

- Ensure that correctly tuned SQL statements are executed, thus minimizing the number of blocks that have to be accessed.
- Increase the size of buffer cache
- Use multiple buffer pools to separate blocks by access characteristics
- Configure the tables to be cached in memory

First, the DBA determines the change in the hit ratio as buffers are added or removed. As a general rule, increase buffer cache size if:

- The cache hit ratio is less than 90%
- There is adequate memory for other processes without inducing additional page faults

Increasing the size of the data buffer cache does not always improve performance. The characteristics of the application may prevent further improvement of the cache hit ratio. For example, in large data warehouse or decision support systems, which routinely use many scans of large tables, most of the data is read from disk. For such systems, tuning the buffer cache is less important and tuning I/O is vital.

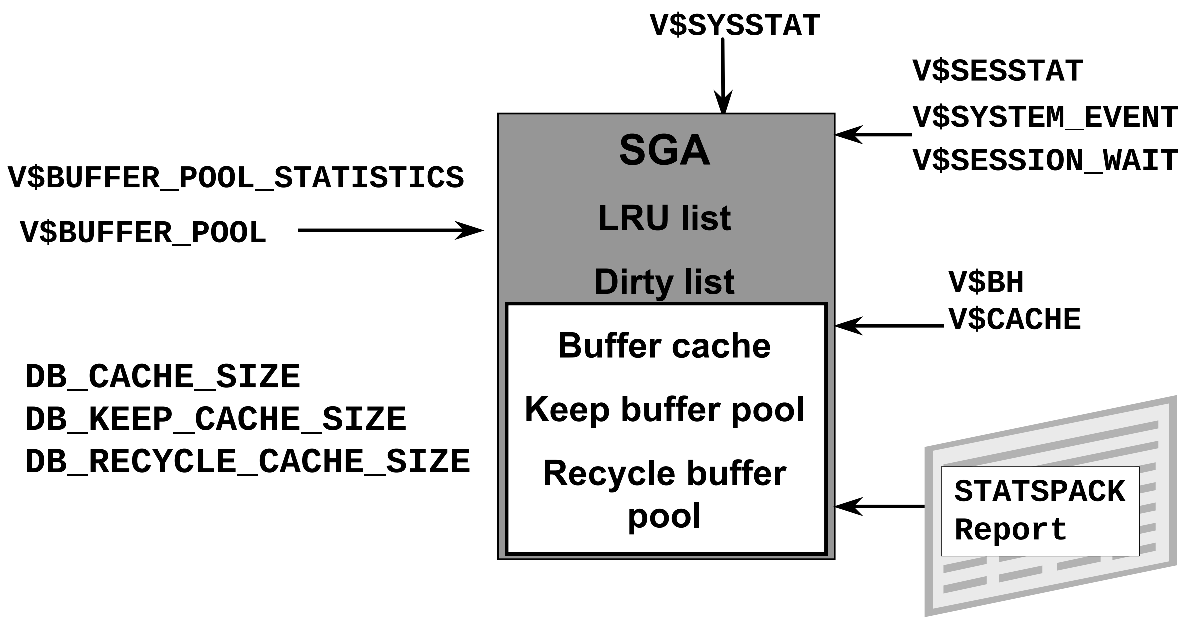
If data access characteristics are causing the low cache hit ratio, the DBA may be able to improve the ratio by defining multiple pools or caching tables.

Technical Note

You need to consider the impact of operating system caching. For example, the Oracle server may show a high rate of physical I/O that does not appear at the operating system level. This could mean that Oracle blocks, aged out of the buffer cache, are kept in the operating system cache and can be accessed very quickly. However, as a general rule it is best to bypass the operating system cache, because:

- More memory may be required to maintain duplicate blocks in memory (one block in the operating system cache and one in the database buffer cache)
- There is the CPU overhead of copying blocks from the operating system cache to the database buffer cache

Diagnostic Tools



4-21

Description of the Views

- The V$SYSSTAT and V$SESSTAT views contain the statistics used to calculate the cache hit ratio:

```
SQL> SELECT  name, value FROM v$sysstat
2>   WHERE  name in (‘session logical reads‘,
3>                   ‘physical reads`,
4>                   ‘physical reads direct’
5>                   ‘physical reads direct (lob)’);
```

- V$BUFFER_POOL: Describes multiple buffer pools, and V$BUFFER_POOL_STATISTICS shows information on individual pools. You can also monitor buffer pools with the query:

```
SQL> SELECT name, physical_reads, db_block_gets,
2>          consistent_gets
3>   FROM   v$buffer_pool_statistics;
```

- V$BH: Describes blocks held in the buffer cache

Measuring the Cache Hit Ratio

From V$SYSSTAT:

```
SQL> SELECT 1 - (phy.value – lob.value – dir.value)
   / ses.value  "CACHE HIT RATIO"
  2  FROM   v$sysstat ses, v$sysstat lob,
  3         v$sysstat dir, v$sysstat phy
  3  WHERE  ses.name = 'session logical reads'
  4  AND    dir.name = ‘physical reads direct'
  5  AND    lob.name = 'physical reads direct (lob)'
  6  AND    phy.name = 'physical reads';
```

From the STATSPACK report:

```
Statistic                   Total       Per         Per
                            Trans     Logon      Second
--------------------- ---------------   ---------
physical reads         15,238      13.0    15,238.0
physical reads direct     863       0.7       863.0
Physical reads direct(lob)  0         0           0
session logical reads 119,376     101.8   119,376.0
```


Measuring the Cache Hit Ratio

The server collects statistics on data access and stores them in the dynamic performance table V$SYSSTAT. You measure the cache hit ratio using the following system statistics:

- `physical reads`: Number of blocks read from disk
- `physical reads direct`: Number of direct reads, does not require the cache
- `physical reads direct (lob)`: Number of direct reads of large binary objects
- `session logical reads`: Number of logical read requests

Calculate the hit ratio for the buffer cache with this formula:

```
Hit Ratio = 1 – (physical reads – physical reads direct -
physical reads direct (lob) ) / session logical reads
```

Session logical reads gives the total number of read requests for data. This value includes requests satisfied by access to buffers in memory and requests that cause a physical I/O. You can multiply the ratio by 100 to convert it to a percentage.

Because these statistics are collected since the instance startup time, query them during normal working loads but not immediately after startup. Because the buffer cache is empty when the instance starts, there are more physical reads after startup.

Guidelines for Using the Cache Hit Ratio

Hit ratio is affected by data access methods:

- **Full table scans**
- **Data or application design**
- **Large table with random access**
- **Uneven distribution of cache hits**

4-23

Buffer Cache Hit Ratio Isn't Everything

- A badly tuned database can still have a hit ratio of 99% or better
- Hit ratio is only a start in determining tuning performance.
- Hit ratio does not determine if a database is optimally tuned.
- Use the Oracle Wait Interface to examine what is causing a bottleneck.
  - `V$SESSION_WAIT`
  - `V$SESSION_EVENT`
  - `V$SYSTEM_EVENT`
- Tune SQL statements

ORACLE

4-24

Using the Cache Hit Ratio

Having a good cache hit ratio is only a part of the tuning sequence. After tuning the buffer cache it is required to examine other areas in which the DBA can have a positive effect. In order to differentiate the good, from the bad, it is required to examine the Wait Statistics determined by using the Wait Interface (mentioned in Lesson 2).

Example

An application could have a good hit ratio but still have many more physical reads than necessary. A logical read is less expensive than a physical read, but there still is an associated cost to retrieving the block. In addition, the "extra" blocks that the application keeps in memory, but does not use to resolve the query, still have to be allocated memory space.

To explain further, assume two applications (A and B) return the same result set. However, Application A has a cache hit ratio of 99%, whereas Application B has a ratio of 60 %. Which application is the best? The obvious answer would be A.

However, on performing deeper investigation we notice that Application A requires 1,000,000 logical reads, and 10,000 physical reads, whereas Application B has 100 logical reads, and only 40 physical reads. Which Application is better tuned now? Obviously, with further research, the answer is B.

Guidelines to Increase the Cache Size

Increase the cache size ratio under the following conditions:

- **Any waits events have been tuned**
- **SQL statements have been tuned**
- **There is no undue page faulting.**
- **If the previous increase of the buffer cache was effective.**
- **Low Cache Hit ratio**

Guidelines

- Tuning the waits reported from the Wait Interface is the first task. The wait events will point the DBA in the direction of the bottleneck that requires tuning.
- Tuning the SQL statements often leads to the greatest benefit in performance. Tune the statements that cause the greatest load on the system. These SQL statements form a part of the STATSPACK report.
- Do not continue increasing `DB_CACHE_SIZE` if the last increase made no significant difference in the cache hit ratio. This may be because of the way that you are accessing your data, or there may be other operations that do not even use the buffer pool. For example, the Oracle server bypasses the buffer cache for sorting and parallel reads.
- Also, when looking at the cache hit ratio, bear in mind that blocks encountered during a full table scan are put at the bottom of the LRU list; therefore, repeated scanning does not cause the blocks to be cached.
- In large databases running an OLTP application, most rows are accessed either one or zero times in any given unit of time. On this basis there is little point in keeping the row (or the block that contains it) in memory for very long after its use.
- The relationship between cache hit ratio and number of buffers is far from a smooth distribution. When tuning the buffer pool, avoid the use of additional buffers that contribute little or nothing to the cache hit ratio.

Guidelines (continued)

Increasing the Cache Hit Ratio by Reducing Buffer Cache Misses

If your hit ratio is low, you may want to increase the number of buffers in the cache to improve the buffer cache hit ratio.

To make the buffer cache larger:

- Allocate more memory to the buffer cache by decreasing the size of any other component of SGA with unused memory.
- If there is room for SGA to grow (that is, if `SGA_MAX_SIZE` is not reached), use `ALTER SYSTEM` to increase the value of `DB_CACHE_SIZE`, (`DB_KEEP_CACHE_SIZE`, or `DB_RECYCLE_CACHE_SIZE`).

The DBA then collects new statistics that estimate the performance gain resulting from increasing the size of the buffer cache. With these statistics, you can estimate how many buffers to add to your cache if necessary.
For Oracle8*i* databases increase the value of the `DB_BLOCK_BUFFERS` initialization parameter. This parameter is static.

Removing Unnecessary Buffers When Cache Hit Ratio Is High

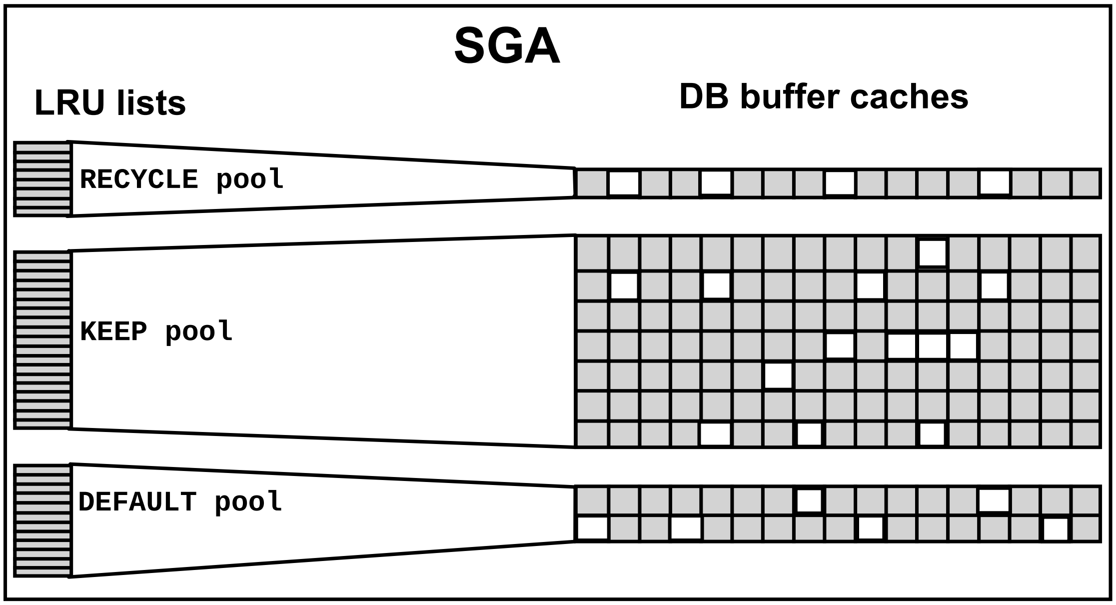
If your hit ratio is high, your cache is probably large enough to hold your most frequently accessed data. In this case, you should be able to reduce the cache size and still maintain good performance.

To make the buffer cache smaller, reduce the value of the `DB_CACHE_SIZE` initialization parameter. The minimum value for this parameter is 1 granule. You can use any leftover memory for other Oracle memory structures.

Decrease the value of the `DB_BLOCK_BUFFERS` initialization parameter for Oracle8*i* databases. This parameter is static.

The DBA then collects new statistics to calculate buffer cache performance based on a smaller cache size. Examining these statistics can help you determine how small you can afford to make your buffer cache without adversely affecting performance.

Using Multiple Buffer Pools



Multiple Buffer Pools

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

- KEEP: This pool is used to retain objects in memory that are likely to be reused. Keeping these objects in memory reduces I/O operations.
- RECYCLE: This pool is used to eliminate blocks from memory that have little chance of being reused. Flushing these blocks from memory enables you to allocate the space that would be used by their cache buffers to other objects.
- DEFAULT: The pool always exists. It is equivalent to the buffer cache of an instance without a keep or a recycle pool.

Defining Multiple Buffer Pools

In Oracle9*i*:

- Individual pools have their own size defined by:
  - `DB_CACHE_SIZE`
  - `DB_KEEP_CACHE_SIZE`
  - `DB_RECYCLE_CACHE_SIZE`
- These parameters are dynamic.
- Latches are automatically allocated by Oracle RDBMS.

4-28

Defining Multiple Buffer Pools

You can define three buffer pools: Default, Recycle, and Keep. The fact that buffers are assigned to KEEP pool does not mean that Oracle process will retain them in the cache for a longer period of time. The time that the buffers are kept in memory depends on the load put on the KEEP pool. Ideally you should limit the number of tables that are allowed to use the keep pool so as to maximize the time a block stays in the cache. In order for a table to use the KEEP pool you should specify the keep pool in the storage clause for the tables. By such segregation, you are managing to avoid contention for blocks of different (say RECYCLE or DEFAULT) pools.

Initialization Parameters:

- `DB_CACHE_SIZE:` Defines the size for the default pool in Oracle9*i*. In Oracle9*i*, this pool is individually configured and other pools do not become a part of this pool. In Oracle8*i*, `DB_BLOCK_BUFFERS` parameter defines the number of buffers belonging to the buffer cache for the instance. In Oracle8*i*, each individual buffer pool is created from this total amount; the remainder is allocated to the default buffer pool.
- `DB_KEEP_CACHE_SIZE:` (In Oracle8*i,* `BUFFER_POOL_KEEP`) Defines the size of the cache to be used as a keep cache. In Oracle9*i*, the memory blocks for the KEEP pool are independent of the `DB_CACHE_SIZE`, whereas in Oracle8*i*, memory for KEEP pool is allocated from that defined in `DB_BLOCK_BUFFERS`.

Defining Multiple Buffer Pools (continued)

- DB_RECYCLE_CACHE_SIZE: (In Oracle8*i*, BUFFER_POOL_RECYCLE) Defines the size of the buffer pool for blocks that may not be retained in memory for long. As already stated, the RECYCLE pool gets its allocation from the DB_BLOCK_BUFFERS in Oracle8*i*.

Setting the number of Latches

In Oracle9i the number of latches is set by the instance, and requires no action on the part of the DBA. However, in Oracle8i the number of latches has to be set, and the following points understood before assigning latches:

- DB_BLOCK_LRU_LATCHES: In Oracle8*i*, this parameter is used to allocate the number of LRU latches for the entire database instance (each defined buffer pool takes a latch from this total). In Oracle9*i*, this parameter is not available.
- The minimum number of buffers that must be allocated to each buffer pool is 50 times the number of LRU latches. For example, if a buffer pool has three LRU latches, it must have at least 150 buffers.There is no requirement that any buffer pool be defined for another buffer pool to be used.

Enabling Multiple Buffer Pools

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

ALTER TABLE customer
  STORAGE (BUFFER_POOL RECYCLE);

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


The BUFFER_POOL Clause

The BUFFER_POOL clause is used to define the default buffer pool for an object. It is part of the STORAGE clause and is valid for CREATE and ALTER table, cluster, and index statements. The blocks from an object without an explicitly set buffer pool go into the DEFAULT buffer pool.

The syntax is BUFFER_POOL { KEEP | RECYCLE | DEFAULT }.

When the default buffer pool of an object is changed using the ALTER statement, blocks that are already cached remain in their current buffers until they are flushed out by the normal cache management activity. Blocks read from disk will be placed into the newly-specified buffer pool for the segment.

Because buffer pools are assigned to a segment, objects with multiple segments can have blocks in multiple buffer pools. For example, an index-organized table can have different pools defined on both the index and the overflow segment.

KEEP Buffer Pool Guidelines

- **Tuning goal: Keeping blocks in memory**
- **Size: Holds all or nearly all blocks**
- **Tool: `ANALYZE ... ESTIMATE STATISTICS`**

```
SQL> ANALYZE TABLE hr.countries ESTIMATE STATISTICS;

SQL> SELECT  table_name, blocks
  2    FROM  dba_tables
  3    WHERE owner = 'HR'
  4    AND table_name = 'COUNTRIES';

TABLE_NAME     BLOCKS
---------- ----------
COUNTRIES       14
```


Tuning Goal

The goal of the keep buffer pool is to retain objects in memory, thus avoiding I/O operations. The size of the keep buffer pool is computed by adding together the sizes of all objects dedicated to this pool.

Sizing

Use `ANALYZE ... ESTIMATE STATISTICS` to obtain the size of each object. Indexes may also be put in the keep pool, where necessary. Use ANALYZE INDEX VALIDATE STRUCTURE to find the number of leaf and branch blocks used by the index.

The high-water mark is always exact, even if you use ESTIMATE STATISTICS. To get the total number of blocks required, use the views `DBA_TABLES`, `DBA_TAB_PARTITIONS, and DBA_INDEXES`.

Depending on the data access characteristics and the amount of available memory, you may not want to keep all of the blocks from all of these objects in the buffer pool. Often you can significantly decrease the size of your KEEP buffer pool and still maintain a high hit ratio. Those blocks can be allocated to other buffer pools.

The DBA must monitor objects in the KEEP pool that grow in size. An object may no longer fit in the KEEP buffer pool, in which case blocks will be flushed out of the cache.

RECYCLE Buffer Pool Guidelines

- **Tuning goal:**
  - **Eliminating blocks from memory when transactions are completed**
- **Size:**
  - **Holds only active blocks**
- **Tool:**
  - **V$CACHE**

```
SQL> SELECT  owner#, name, count(*) blocks
  2    FROM  v$cache
  3    GROUP BY owner#, name;

OWNER# NAME            BLOCKS
------ ---------- ----------
     5 CUSTOMER          147
```

4-32

Tuning Goal

The goal of the recycle buffer pool is to eliminate blocks from memory as soon as they are no longer needed. Be careful, however, not to discard blocks from memory too quickly. If the buffer pool is too small, it is possible for a block to age out of the cache before the transaction or SQL statement has completed execution. For example, an application may select a value from a table, use the value to process some data, and then update the selected row. If the block is removed from the cache after the SELECT statement, it must be read from disk again to perform the update. The block needs to be retained for the duration of the transaction.

Sizing

You can size the recycle pool by using the physical reads statistic from a tracing tool or by totaling the buffer cache blocks used by the object.

Tuning Goal (continued)

Using V$CACHE to Find Blocks in the Buffer Pool

The DBA can also monitor the number of buffer pool blocks by object using `V$CACHE`. `V$CACHE` is created by the `catclust.sql` script.

`V$CACHE`:

- Is intended for use with Real Application Clusters
- Creates a number of other views that are useful only for Real Application Clusters
- Maps extents in the data files to database objects

To determine the number of blocks required for objects in the RECYCLE pool:

- Tune the buffer cache with the RECYCLE pool disabled.
- Run `catclust.sql` to set up and populate `V$CACHE`.

During peak running times, use the following query to calculate how many blocks are currently cached for each object:

```
SQL> SELECT owner#, name, count(*) blocks
 2   FROM  v$cache
 3   GROUP BY owner#, name;
```

Sum the blocks for all objects that will be used in the RECYCLE buffer pool and divide by four to get RECYCLE pool size. You divide by four because it is assumed that one-fourth of the blocks targeted for the RECYCLE pool are active; the other three-fourths are waiting to be aged out of the cache.

RECYCLE Buffer Pool Guidelines

Tool: V$SESS_IO

```
SQL> SELECT s.username,
  2         io.block_gets,
  3         io.consistent_gets,
  4         io.physical_reads
  5  FROM   v$sess_io     io,
  6         v$session     s
  6  WHERE  io.sid  = s.sid ;

USERNAME  BLOCK_GETS CONSISTENT_GETS PHYSICAL_READS
--------  ---------- --------------- --------------
      HR       21874            2327           1344
```

ORACLE

Tracing Physical Reads

By executing statements with a SQL statement tuning tool such as Oracle Trace Manager, SQL*Plus Autotrace, or SQL trace with TKPROF, you can get a listing of the total number of data blocks physically read from disk. Also, the `V$SESS_IO` dynamic performance table provides I/O statistics by session. Because you always expect to physically read blocks from the objects in this cache, the number of physical reads for the SQL statement can be greater than or equal to the number of blocks read from the object. The reason for reading more blocks is that with the recycle pool there is the probability of having blocks aged out of the cache before the application has completed using them, therefore necessitating the block, or blocks, to be reread.

Calculating the Hit Ratio for Multiple Pools

```
SQL>
SQL> SELECT name,
            1 - (physical_reads / (db_block_gets +
                 consistent_gets)) "HIT_RATIO"
 2   FROM   sys.v$buffer_pool_statistics
 3   WHERE  db_block_gets + consistent_gets > 0;

NAME                HIT_RATIO
------------------  ----------
KEEP                .983520845
RECYCLE             .503866235
DEFAULT             .790350047
```


Description of V$BUFFER_POOL_STATISTICS

This view displays statistics (physical writes, consistent gets, free buffer waits) against the multiple buffer caches (if allocated):

Column Name	Description
NAME	Name of the buffer pool (KEEP, RECYCLE, DEFAULT)
SET_MSIZE	Maximum buffer size allowed
CNUM_REPL	Current number of buffers in replacement
CNUM_WRITE	Current number of buffers in write list
CNUM_SET	Current total number of buffers in this pool
BUF-GOT	Number of buffers that foreground got for this pool
SUM_WRITE	Number of buffers written by DBW*n* in this pool
SUM_SCAN	Number of buffers scanned by DBW*n* in this pool
FREE_BUFFER_WAIT	Free buffer waits for this pool

Identifying Candidate Pool Segments

- KEEP Pool
  - Blocks are accessed repeatedly.
  - Segment size is less than 10% of the DEFAULT buffer pool size.
- RECYCLE Pool
  - Blocks are not reused outside of transaction.
  - Segment size is more than twice the DEFAULT buffer pool size.

4-36

Trade-Offs

Remember that each object kept in memory results in a trade-off. Although it is beneficial to keep frequently accessed blocks in the cache, retaining infrequently used blocks results in less available space for other more active blocks.

Dictionary Views with Buffer Pools

```
SQL> select id, name, block_size, buffers
  2  from v$buffer_pool;

ID NAME    block_size BUFFERS
-- ------- ---------- --------
 1 KEEP          4096    14000
 2 RECYCLE       4096     2000
 3 DEFAULT       4096     4000
```


Dictionary Views

These dictionary views have a BUFFER_POOL column that indicates the default buffer pool for the given object:

- USER_SEGMENTS, DBA_SEGMENTS
- USER_CLUSTERS, ALL_CLUSTERS, DBA_CLUSTERS
- USER_INDEXES, ALL_INDEXES, DBA_INDEXES
- USER_TABLES, ALL_TABLES, DBA_TABLES
- USER_OBJECT_TABLES, ALL_OBJECT_TABLES, DBA_OBJECT_TABLES
- USER_ALL_TABLES, ALL_ALL_TABLES, DBA_ALL_TABLES

The V$BUFFER_POOL view describes the buffer pools allocated. The columns from V$BUFFER_POOL show the sizes of each buffer, in terms of bytes, and the number of buffers. Also information regarding the resizing of a buffer cache can be found here.

Caching Tables

- **Enable caching during full table scans by:**
  - Creating the table with the CACHE clause
  - Altering the table with the CACHE clause
  - Using the CACHE hint in a query
- **Guideline: Do not overcrowd the cache.**
- **Use a KEEP Pool**

ORACLE

Caching Tables

When the server retrieves blocks using a full table scan, the buffers go to the least recently used end of the LRU list. These buffers are then used next time a free buffer is required.

In order to change this behavior you must do one of the following:

- Create a table using the CACHE clause
- Alter a table using the CACHE clause
- Code the CACHE hint clause into a query

If you use one of these methods, the Oracle server places the table blocks higher on the LRU list, towards the most recently used. Use the CACHE clause when you create small lookup tables used by many users. You may overcrowd the buffer cache if you have too many cached tables.

Tables can also be effectively cached by using a KEEP pool.

Other Buffer Cache Performance Indicators

From V$SYSSTAT:

```
SQL> SELECT name, value
  2  FROM   v$sysstat
  3  WHERE  name = 'free buffer inspected';

NAME                            VALUE
--------------------------- --------
free buffer inspected             183
```


Other Performance Indicators

The buffer cache hit ratio is a measure of buffer cache performance. However, there are some other indicators of performance, that are often more useful..

Wait Statistics

You should consider increasing the buffer cache size if there are high or increasing values for the Free Buffer Inspected system statistic. This statistic is the number of buffers skipped to find a free buffer. Buffers are skipped because they are dirty or pinned.

Wait Events

You can find out whether there have been waits for buffers from `V$SYSTEM_EVENT` or `V$SESSION_WAIT`. If there are no waits, the event has not yet occurred. There are three main events to look out for:

Buffer Busy Waits

This wait indicates that there are some buffers in the buffer cache that multiple processes are attempting to access concurrently. Query `V$WAITSTAT` for the wait statistics for each class of buffer. Common buffer classes that have buffer busy waits include data block, segment header, undo header, and undo block.

Other Buffer Cache Performance Indicators

From V$SYSTEM_EVENT:

```
SQL> SELECT event, total_waits
  2  FROM   v$system_event
  3  WHERE  event in
  4         ('free buffer waits', 'buffer busy waits');

EVENT                  TOTAL_WAITS
---------------------- -----------
free buffer waits              337
buffer busy waits             3466
```

4-40

Other Performance Indicators (continued)

Buffer Busy Waits (continued)

- **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
  - V$SESSION_WAIT 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 to.

undo header

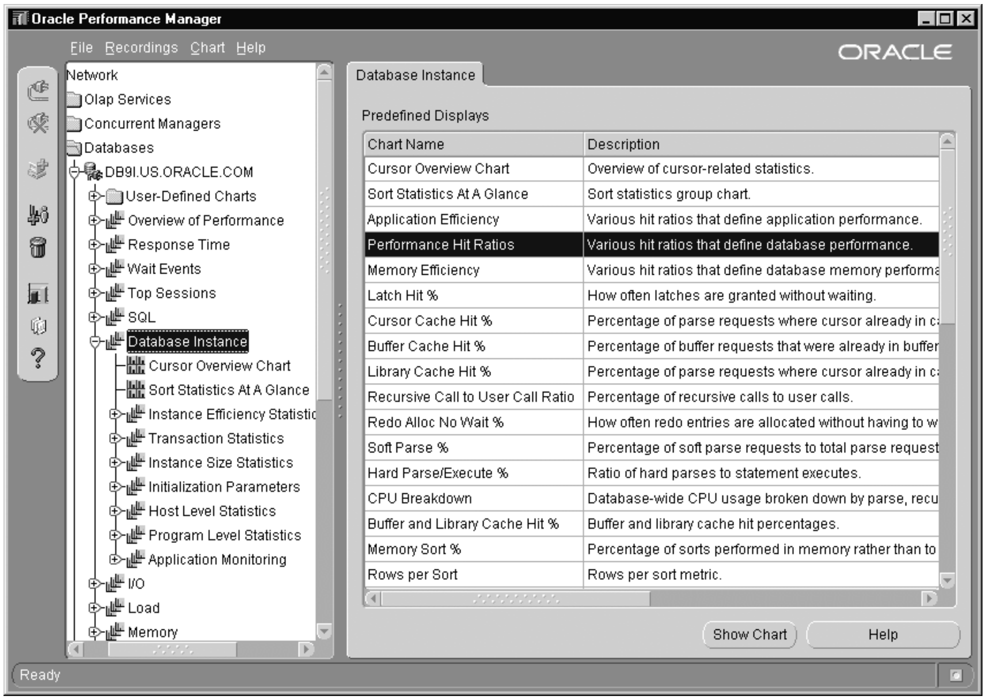
Displays contention on rollback segment header: If you are not using automatic undo management, then add more rollback segments.

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.
Free Buffer Waits
This wait event indicates that a server process was unable to find a free buffer and has posted the database writer to make free buffers by writing out dirty buffers. A dirty buffer is a buffer whose contents have been modified. Dirty buffers are freed for reuse when DBWn has written the blocks to disk.
In order to resolve the contention, DBWn has to make buffers available faster for overwriting. To achieve this, examine ways of speeding up the write process. This event is also an indication that the buffer cache is too small. Examine the hit ratios for the buffer cache in order to determine if the cache should be resized.
Causes
DBWn may not be keeping up with writing dirty buffers in the following situations:
The I/O system is slow. Solution: Check that the files are equally distributed across all devices. If that produces no affect 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 affect 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.
Actions
If this event occurs frequently, examine the session waits for DBWn to determine whether there is anything delaying DBWn.

Performance Manager



Performance Manager: Buffer Cache

There are many different charts that will give information regarding the buffer cache. Some of these charts include:

Memory: SGA Overview

The charts give a breakdown of the SGA as a whole, and what portion is allocated to the buffer Cache.

Memory: Buffer Cache

Shows what blocks are located in the buffer cache, and which files the blocks come from. This can assist in determining which files are being heaviest hit.

Load: Instance Statistics Per Second

This view gives some of the important statistics regarding the buffer cache.

Database Instance: Instance Efficiency Statistics

These charts show the hit ratios for the database.

Free Lists

- A free list for an object maintains a list of blocks that are available for inserts.
- The number of free lists for an object can be set dynamically.
- Single-CPU systems do not benefit greatly from multiple free lists.
- The tuning goal is to ensure that an object has sufficient free lists to minimize contention.
- Using Automatic Free Space Management eliminates the need for free lists, thus reducing contention on the database.

ORACLE

Using Free Lists

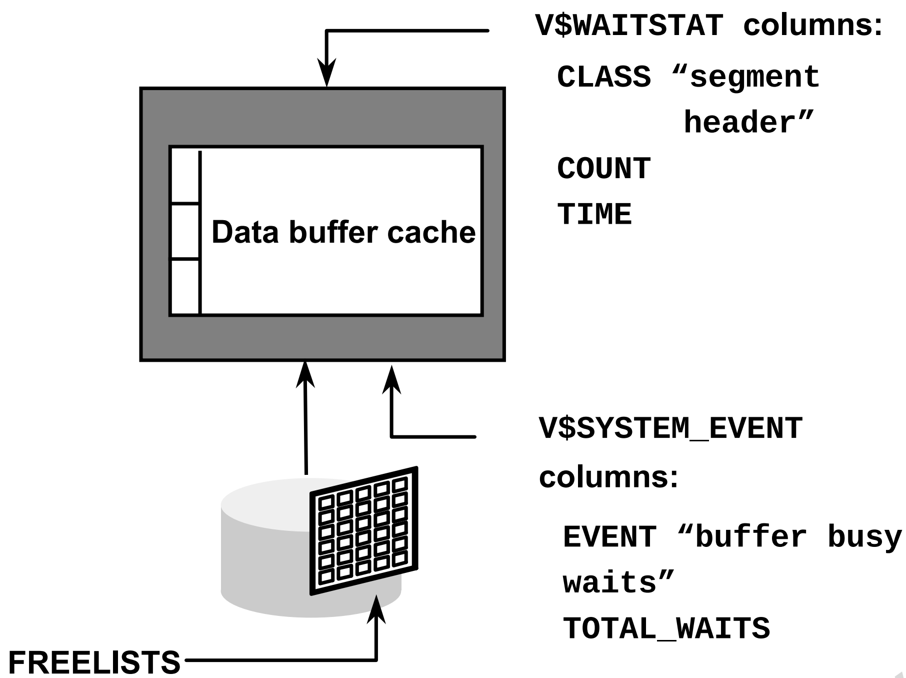
When an insert operation on an object occurs, the free list is used to determine which blocks are available for inserts. Many server processes can contend for the same free list if many inserts are occurring. This results in free list contention while server processes incur waits.

Single-CPU systems do not benefit greatly from multiple free lists, because the CPU manages one process at a time. Even in a single-CPU system, however, adding free lists may ensure that the processor is used more effectively. Care should still be taken when adding free lists.

The overall tuning goal for free lists is to ensure that there is a sufficient number to minimize contention among many server processes.

Free lists are eliminated by using Automatic Free Space Management. Instead Oracle uses bitmaps that are faster to update, and therefore cause dramatically less contention between sessions, and also between instances (when using Real Application Clusters).

Diagnosing Free List Contention



4-44

Diagnosing Free List Contention

The V$WAITSTAT and V$SYSTEM_EVENT dynamic performance views are used to diagnose free list contention problems. If queries against these views return high numbers, you must identify the object or objects.

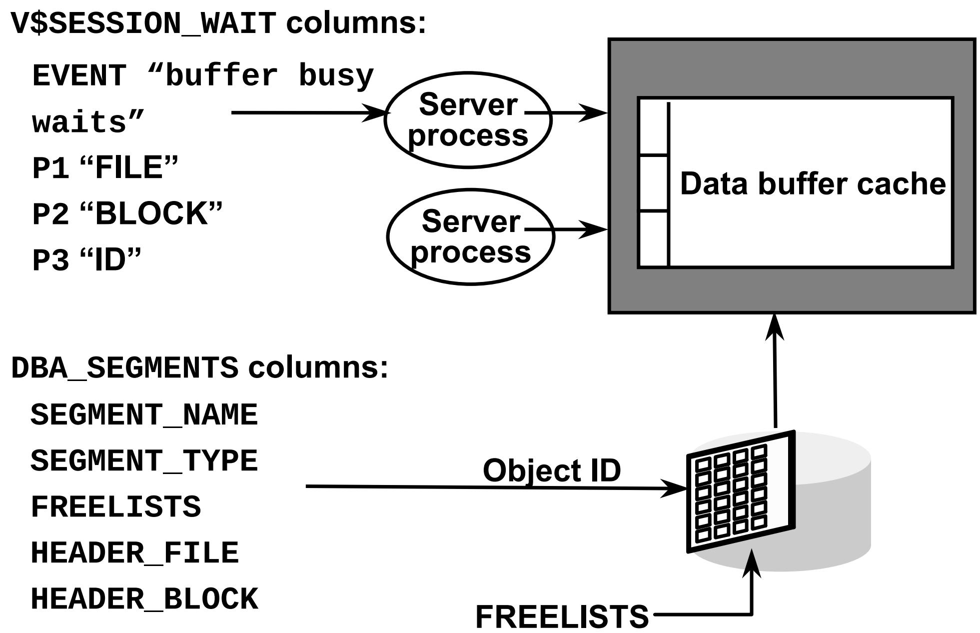
- To query V$WAITSTAT:

```
SELECT class, count, time
FROM v$waitstat
WHERE class = 'segment header';
```

- To query V$SYSTEM_EVENT:

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

Resolving Free List Contention



Identifying the Object

V$SESSION_WAIT contains the file ID and block ID of the segment incurring 'busy buffer waits'. By joining this view with DBA_SEGMENTS you can identify the segment and determine the number of free lists that currently exist for that segment, as shown below:

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

Reducing Busy Buffer Waits

To reduce buffer busy waits on:

- Data blocks:
  - Change PCTFREE and/or PCTUSED;
  - check for right-hand indexes (indexes that are inserted into at the same point by many processes);
  - increase INITRANS;
  - reduce the number of rows per block

Reducing Busy Buffer Waits (continued)

- Segment headers:
  - Use free lists or increase the number of free lists;
  - use free list groups (This can make a difference even in a single instance environment.)
- Free list blocks:
  - Add more free lists
  - In the case of Oracle Parallel Server, make sure that each instance has its own free list group. The number of free lists in a free list group can be changed by an `ALTER TABLE` statement.

Note: You cannot alter the free list storage parameter for segments in tablespaces using Automatic Segment Space Management.

Resolving Free List Contention

To increase the number of free lists for the object, do one of the following:

- Use the `ALTER TABLE` command to increase the number of `FREELISTS`.
- Move the object to a tablespace using Automatic Segment Space Management. (Discussed later in this chapter)
- Use Enterprise Manager Console, under the SCHEMA – TABLE option.

Automatic Segment Space Management

- Manages free space automatically inside database segments.
- Tracks segment free/used space with bitmaps instead of free lists
- Provides better space utilization, especially for the objects with highly varying size rows
- Specified when creating a tablespace
- Cannot be used for tables that contain one, or more, LOB columns
- Supported by the Enterprise Manager Console

4-47

Auto-Management of Free Space

With Oracle9*i,* free space can be managed automatically inside database segments. The in-segment free/used space is tracked using bitmaps, as opposed to free lists. Use of bitmaps offers the following benefits:

- Ease of use
- Better space utilization, especially for the objects with highly varying size rows
- Better run-time adjustment to variations in concurrent access
- Better multi-instance behavior in terms of performance/space utilization
- Preparation for future enhancements, such as in-space segment reorganization and in-place tablespace reorganization
- You specify auto-management of free space when you create a tablespace. The specification then applies to all segments subsequently created in this tablespace.

Tables that contain one, or more, LOB columns cannot be created in tablespace that has its free space management set to AUTO.

Tables, and Tablespaces, can be created using the Enterprise Manager Console.

Auto-Management of Free Space

- Create an auto-managed tablespace:

```
CREATE TABLESPACE BIT_SEG_TS
DATAFILE '$HOME/ORADATA/u04/bit_seg01.dbf'
SIZE 1M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

- Create a table that uses auto-management of free space:

```
CREATE TABLE BIT_SEG_TABLE
(IDNUM NUMBER)
TABLESPACE BIT_SEG_TS;
```

Multiple I/O Slaves

- Provide nonblocking asynchronous I/O requests
- Are typically not recommended if asynchronous I/O is available
- Follow the naming convention `ora_innn_SID`
- Turn asynchronous I/O on or off with `DISK_ASYNCH_IO`

I/O Slave

Asynchronous I/O behavior, if not natively available, can be simulated with the deployment of I/O slave processes. I/O slaves are specialized processes whose only function is to perform I/O. The `DBWR_IO_SLAVES` initialization parameter control I/O slave deployment. You can turn asynchronous I/O on and off with the `DISK_ASYNCH_IO` parameter. It may be necessary to turn off the asynchronous I/O facility provided by the operating system. For example, if the asynchronous I/O code of the platform has bugs or is not efficient. Usually the parameter should be left at the default value of TRUE.

The I/O Slave Mechanism

I/O slaves can be deployed by the `DBW0` process. I/O slaves for `DBW0` are allocated immediately following database open when the first I/O request is made.

The DBW0 process looks for an idle I/O slave. If one is available it gets the post. If there are no idle slaves, the I/O issuer spawns one. If the allowed number of slaves have been spawned, the issuer waits and tries again to find an idle slave. The `DBW0` continues to do all the `DBW0`-related work; for example, gathering dirty buffers into a batch. The `DBW0` I/O slave simply does the I/O on behalf of `DBW0.` That is, the writing of the batch is parallelized between the I/O slaves. This is beneficial in write-intensive environments, because the CPU time associated with issuing I/Os can be divided between the I/O slaves.

Multiple DBW*n* Processes

- Multiple DB Writer processes can be deployed with DB_WRITER_PROCESSES (DBW0 to DBW9).
- This is useful for SMP systems with large numbers of CPUs.
- Multiple processes cannot concurrently be used with multiple I/O slaves.

The Multiple DBW*n* Mechanism

Multiple DBW*n* processes can be specified by the DB_WRITER_PROCESSES parameter. Up to 10 processes (DBW0 to DBW9) can be used. In contrast to the multiple I/O slaves, which only parallelize the writing of the batch between the DBWn I/O slaves, you can parallelize the gathering as well as the writing of buffers with the multiple DBWn feature.

Therefore, DBWn processes should deliver more than the throughput of one DBW0 process with the same number *n* of I/O slaves.

Tuning DBW*n* I/O

Tune the DB Writer processes by looking at the value of the `FREE BUFFER WAITS` event

Guidelines

Consider increasing the DBW*n* processes, or configure I/O slaves if you see a high number of `free_buffer_waits` after querying the `V$SYSTEM_EVENT` view as in the following syntax:

```
SQL> SELECT total_waits
     2  FROM V$SYSTEM_EVENT
     3  WHERE EVENT = 'free buffer waits';
```

Summary

In this lesson, you should have learned how to:

- Keep frequently needed blocks in a cache
- Adjust the size of the buffer cache as necessary
- Use the buffer cache advisory feature
- Separate objects into multiple buffer pools
- Use multiple buffer pools
- Cache tables
- Reduce free list contention by creating multiple freelists
- Avoid free list contention by using Auto-Management of Free Space
- Configure multiple I/O slaves
- Use multiple DBWn processors

ORACLE

Practice 4

The objective of this practice is to use available diagnostic tools to monitor and tune the database buffer cache. Through out this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as perfstat/perfstat user, and run a statistic snapshot, and note the snapshot number. This can be performed by running the script file `$HOME/STUDENT/LABS/snap.sql`.
2. To simulate user activity against the database, connect as hr/hr user and run the `lab04_02.sql` script.
3. Connect as system/manager and measure the hit ratio for the database buffer cache using the `v$sysstat` view. Determine if it is a good ratio or not.
4. Connect as perfstat/perfstat, and run a statistic snapshot, and note the snapshot number. This can be performed by running the script file `$HOME/STUDENT/LABS/snap.sql`.
5. Generate a report from `STATSPACK` using the last two snapshots to check the buffer cache hit ratio, by running the script `$HOME/STUDENT/LABS/spreport.sql`. Then analyze the buffer hit % in the "Instance Efficiency Percentages" section.
   Note: On a production database if the ratio is bad, add new buffers, run steps 2 to 5, and examine the new ratio to verify that the ratio has improved. If the ratio is good, remove buffers, run steps 2 to 5, and verify if the ratio is still good.
6. Connect as system/manager and determine the size of the table `TEMP_EMPS` in the hr schema that you want to place in the `KEEP` buffer pool. Do this by using the `ANALYZE` command, then query the BLOCKS column of the `DBA_TABLES` view for the `temp_emps` table.
7. We intend to keep `TEMP_EMPS` in the KEEP pool. Use the "alter system" command to set `DB_KEEP_CACHE_SIZE` to 4 MB for the KEEP pool. This will generate an error due to insufficient memory in the SGA.
8. To resolve the memory problem reduce the size of the shared pool by 8M, using the "alter system" command to set the value of `SHARED_POOL_SIZE`. Then reissue the command to size the `DB_KEEP_CACHE_SIZE` to 4 MB.
   Note: In a production environment check if you have sufficient SGA size to grow, and if any other component could be reduced in size without adversely affecting performance.
9. Connect as system/manager and enable the `TEMP_EMPS` table in the hr schema for caching in the keep pool, using the storage clause of the `ALTER TABLE` command.
10. Connect as hr/hr, and run the script `$HOME/STUDENT/LABS/lab04_10.sql`. This will execute a query against the `TEMP_EMPS` table in the hr schema.
11. Connect using sys/oracle as sysdba and check for the hit ratio in different buffer pools, using the `V$BUFFER_POOL_STATISTICS` view.

5

Sizing Other SGA Structures

Objectives

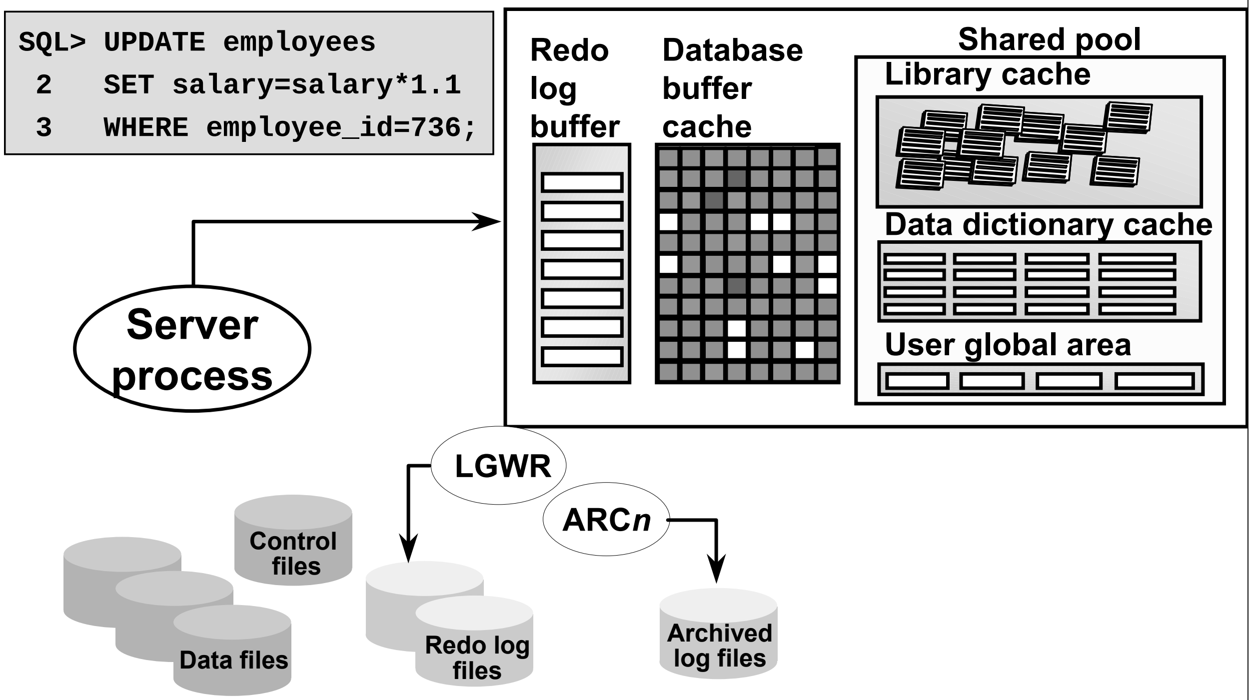
After completing this lesson, you should be able to do the following:

- Monitor and size the redo log buffer
- Monitor and size the Java pool
- Control the amount of Java session memory used by a session

ORACLE

5-2

The Redo Log Buffer



5-3

Redo Log Buffer Content

- The Oracle server processes copy redo entries from the user's memory space to the redo log buffer for each DML or DDL statement.
- The redo entries contain the information necessary to reconstruct or redo changes made to the database by DML and DDL operations. They are used for database recovery, and take up continuous, sequential space in the buffer.

Redo Entries and LGWR

- The LGWR process writes the redo log buffer to the active online redo log file (or members of the active group) on disk. It writes all redo entries that have been copied into the buffer since the last time it wrote.
- The redo log buffer is a circular buffer. Thus, server processes can copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries.

What Causes LGWR to write?

LGWR will write out the redo data from the redo log buffer when:

- A commit record when a user process commits a transaction
- Every three seconds
- When the redo log buffer is one-third full
- When a DBWn process writes modified buffers to disk, if necessary

Sizing the Redo Log Buffer

- **Adjust the `LOG_BUFFER` parameter**
- **Default value:**
  Either 512K or 128K * the value of `CPU_COUNT`, whichever is greater.

Sizing the Redo Log Buffer

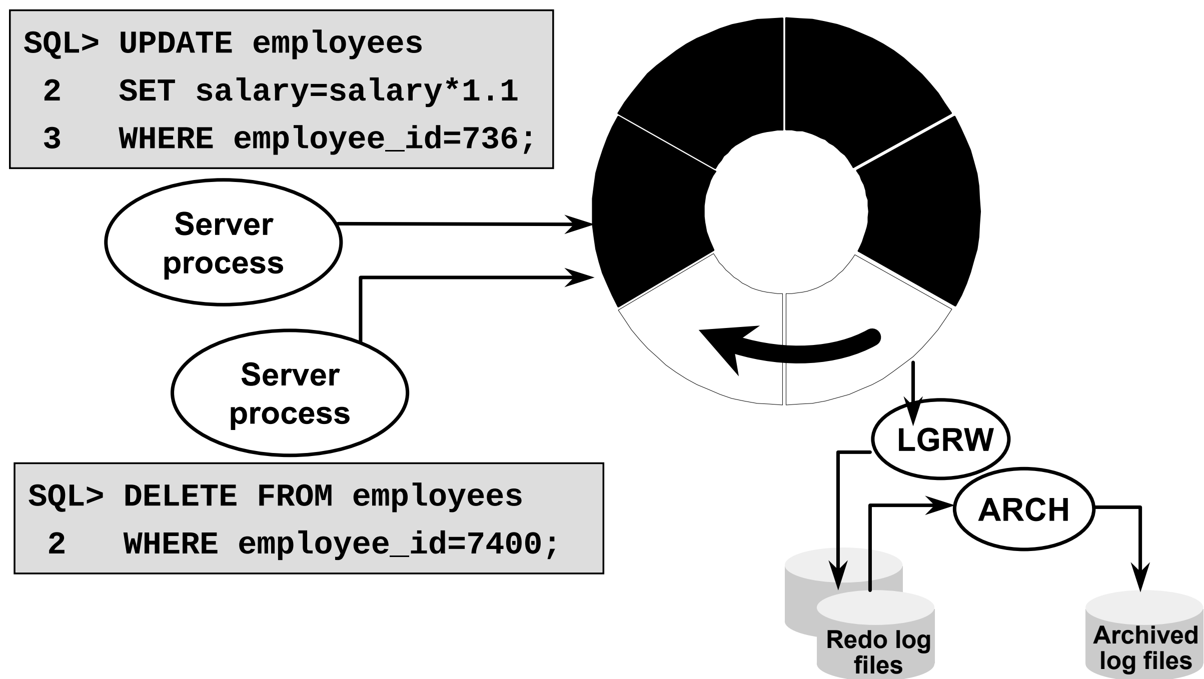
- Larger values of LOG_BUFFER reduce log file I/O, particularly if transactions are long or numerous, since the smaller the buffer the more the buffer will get one third full .
- Frequent `COMMIT` statements clear out the buffer, leading to a smaller buffer size.
- The default value of LOG_BUFFER is either 512K, or 128K * the value of CPU_COUNT, whichever is greater.

Example

```
SQL> select 'V$PARAMETER' “View name", name,
   2  to_number(value,'9999999') "Value“
   3  from v$parameter
   4  where name = 'log_buffer‘
   5  UNION
   6  select 'V$SGASTAT' “View name", name, bytes
   7  from v$sgastat
   8  where name ='log_buffer';

View name        NAME                   Value
------------     -----------            ---------
V$PARAMETER      log_buffer             120320
V$SGASTAT        log_buffer             120320
```

Diagnosing Redo Log Buffer Inefficiency



5-5

Diagnosing Problems

On machines with fast processors and relatively slow disks, the processors may be filling the rest of the redo log buffer in the time it takes the LGWR process to move a portion of the buffer out to disk.

For this reason, a larger buffer makes it less likely that new entries will collide with the part of the buffer still being written. However, if the amount of redo being is greater than the rate at which it is able to be copied out, then no matter how big the redo buffer, it will finally fill up. In these cases one must either ensure enough redo log buffer space to see the system through to the next “quiet” time, or speed up the copying process.

Server processes may request space from the redo log buffer to write new entries and not find any. They will have to wait for LGWR to flush the buffer to disk.

Tuning Goal

Tuning the redo log buffer means ensuring that the space required by server processes in the redo log buffer is sufficient. However, too much space will reduce the amount of memory that can be allocated to other areas. It is also important to note that the DBA can adopt practices that will reduce the amount of redo that must be performed. These practices will be mentioned later in this lesson.

Using Dynamic Views to Analyze Redo Log Buffer Efficiency

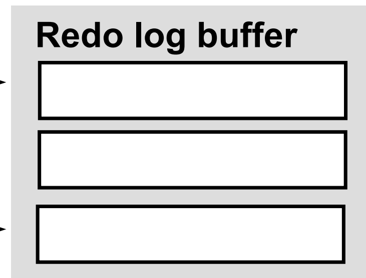
V$SESSION_WAIT

Log Buffer Space event

V$SYSSTAT

Redo Buffer Allocation Retries

Redo Entries



Dynamic Views

The V$SESSION_WAIT view indicates through the Log Buffer Space event if there are any waits for space in the log buffer because the session is writing data into the log buffer faster than LGWR can write it out.

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

SID     EVENT                    SECONDS_IN_WAIT     STATE
-----   ------                   ---------------     ---------
15      log buffer space                     110       WAITING
```

Redo Buffer Allocation Retries Statistic Ratio

The value of Redo Buffer Allocation Retries should be near 0. This number should not be greater than 1% of the redo entries. If this value increments consistently, processes have had to wait for space in the buffer.

```
SQL> select r.value “Retries”, e.value “Entries”,
  2  r.value/e.value*100 “Percentage”
  3  from v$sysstat r, v$sysstat e
  4  where r.name = 'redo buffer allocation retries‘
  5  and e.name='redo entries‘;
```

Dynamic Views (continued)

```
   Retries    Entries    Percentage
---------- ---------- -------------------
         0        189                   0
```

The wait may be caused by the log buffer being too small, by checkpointing, or by archiving. In this case you would:

- Increase the size of the redo log buffer, if necessary, by changing the value of the initialization parameter `LOG_BUFFER`.
- Alternatively, improve the checkpointing or archiving process.

The redo log buffer is normally small and a modest increase can greatly enhance throughput.

- The `SECONDS_IN_WAIT` value of the Log Buffer Space event indicates the time spent waiting for space in the redo log buffer because the log switch does not occur.
  This is an indication that the buffers are being filled up faster than `LGWR` is writing.
  This may also indicate disk I/O contention on the redo log files.
- The Redo Buffer Allocation Retries statistic in `V$SYSSTAT` view reflects the number of times a user process waits for space in the redo log buffer to copy new entries over the entries that have been written to disk. `LGWR` normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the redo log is heavy.

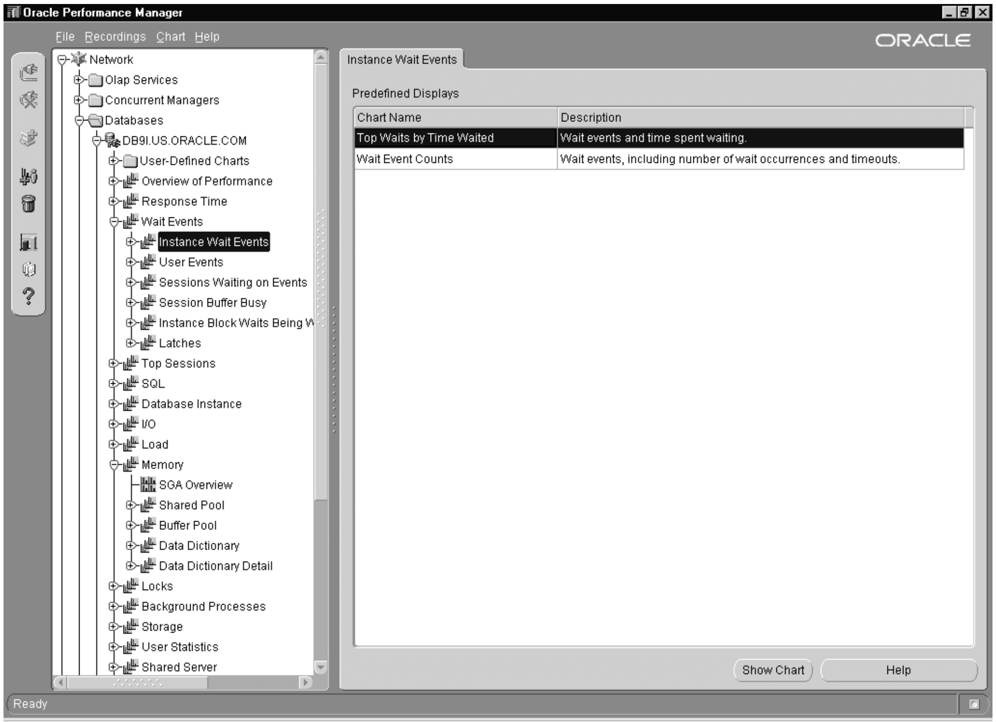
```
SQL> SELECT name, value
2   FROM   v$sysstat
3   WHERE  name = 'redo buffer allocation retries';
```

Note: The `V$SYSSTAT` view displays another statistic, Redo Log Space Requests:

```
SQL> select name, value
2  from   v$sysstat
3  where  name='redo log space requests';
```

This statistic indicates that the active log file is full and that the Oracle server is waiting for archive log space to be allocated.

Performance Manager



5-8

Performance Manager

This Performance Manager screen shot shows Wait Events that have been reported for the Instance.

Instance Wait Events

This chart shows the top wait events for the database. If the Log Buffer Space event appears this will indicate some time spent waiting for space in the redo log buffer. You should:

- Make the log buffer bigger if it is small
- Move the log files to faster disks. It is generally not a good idea to have the redo logs on striped disks.

Performance Manager – Memory

This chart provides an overview of the SGA. Two of the factors shown are the Log Buffer and the `Java_Pool`, which is dealt with later in this lesson.

Redo Log Buffer Tuning Guidelines

There should be no Log Buffer Space waits.

```
SQL> SELECT sid, event, seconds_in_wait, state
 2   FROM   v$session_wait
 3   WHERE  event = ‘log buffer space’;
```

Redo Buffer Allocation Retries value should be near 0, and should be less than 1% of redo entries.

```
SQL> SELECT name, value
 2   FROM   v$sysstat
 3   WHERE  name IN (‘redo buffer allocation retries’,
 4                   ‘redo entries’);
```


SECONDS_IN_WAIT for Log Buffer Space Event

In V$SESSION_WAIT, if the SECONDS_IN_WAIT value for the Log Buffer Space event indicates some time spent waiting for space in the redo log buffer, consider:

- Making the log buffer bigger if it is small
- Moving the log files to faster disks such as striped disks

```
SQL> select sid, event, seconds_in_wait, state
   2  from v$session_wait
   3  where event = 'log buffer space%';
  SID   EVENT      SECONDS_IN_WAIT      STATE
-----   ------     ---------------      -------------
5 log   buffer space          110       WAITING
```

Further Investigations

Investigate the possible reasons why the LGWR is slow in freeing buffers:

- There is disk I/O contention on the redo log files. Check that the redo log files are stored on separate, fast devices.
  - In the V$SYSTEM_EVENT view, check the number of occurrences of the Log File Switch Completion event, which identifies the log file switch waits because of log switches.

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

  - Increase the size of the redo log files.
- DBW*n* has not completed checkpointing the file when the LGWR needs the file again. LGWR has to wait.
  - In the alert.log file, check for the message "CHECKPOINT NOT COMPLETE."
  - In the V$SYSTEM_EVENT view, check the number of occurrences of the Log File Switch (Checkpoint Incomplete) event, which identifies the log file switch waits because of incomplete checkpoints.

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

  - Check the frequency of checkpoints and set an appropriate value for FAST_START_MTTR_TARGET.
  - Check the size and number of redo log groups.
- The archiver cannot write to the archived redo log files or cannot complete the archive operation fast enough. Therefore, it prevents the LGWR from writing.
  - Confirm that the archive device is not full and add redo log groups.
  - In the V$SYSTEM_EVENT view, check the number of the occurrences of the Log File Switch (Archiving Needed) event, which identifies the log file switch waits because of the archiving issue.

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

The LGWR process starts a new ARC*n* process whenever the current number of ARC*n* processes is insufficient to handle the workload. If you anticipate a heavy workload for archiving, such as during bulk loading of data, specify the maximum number of multiple archiver processes with the LOG_ARCHIVE_MAX_PROCESSES initialization parameter. This parameter is dynamic, and can be changed using the ALTER SYSTEM statement.

Reducing Redo Operations

Ways to avoid logging bulk operations in the redo log:

- **Direct Path loading without archiving does not generate redo.**
- **Direct Path loading with archiving can use Nologging mode.**
- **Direct Load Insert can use NOLOGGING mode.**
- **Some SQL statements can use NOLOGGING mode.**

SQL*Loader and the NOLOGGING Mode

Conventional path loading generates redo log entries just as any DML statement. When using direct path, redo log entries are not generated if:

- The database is in `Noarchivelog` mode
- The database is in `Archivelog` mode, but logging is disabled. Logging can be disabled by setting the `NOLOGGING` attribute for the table or by using the `UNRECOVERABLE` clause in the control file.

Direct Load Insert and NOLOGGING Mode

The `NOLOGGING` option:

- Applies to tables, tablespaces, and indexes
- Does not record changes to data in the redo log buffer. Some minimal logging is still carried out, for operations such as extent allocation.
- Is not specified as an attribute at the `INSERT` statement level, but is instead specified when using the `ALTER` or `CREATE` command for the table, index, or tablespace
- If NOLOGGING is set at the tablespace level, it specifies that NOLOGGING is the default option for new objects created in the tablespace, but it does not affect the NOLOGGING capabilities of objects already created in the tablespace.

Direct Load Insert and NOLOGGING Mode (continued)

- Is set before the load and is reset to LOGGING once the load completes. If a media failure occurs before a backup is taken, then all tables, and indexes that have been modified, may be corrupted.

SQL Statements that Can Use NOLOGGING Mode

Although you can set the `NOLOGGING` attribute for a table, index, or tablespace, Nologging mode only applies to a few operations on the object for which the attribute is set, such as:

- `CREATE TABLE ... AS SELECT`
- `CREATE INDEX`
- `ALTER INDEX ... REBUILD`
- `DIRECT PATH INSERT`

The following statements are nevertheless unaffected by the `NOLOGGING` attribute: `UPDATE`, `DELETE`, conventional path `INSERT`, and various DDL statements not listed above.

Note: For backward compatibility, `UNRECOVERABLE` is still supported as an alternate keyword with the `CREATE TABLE` statement. This alternate keyword may not be supported in future releases.

Monitoring Java Pool Memory

```
SQL> SELECT * FROM v$sgastat
  2  WHERE pool = 'java pool';
POOL        NAME                            BYTES
----------- --------------------- ----------
java pool   free memory                  30261248
java pool   memory in use                19742720
```

Limit Java session memory usage:

- **`JAVA_SOFT_SESSIONSPACE_LIMIT`**
- **`JAVA_MAX_SESSIONSPACE_SIZE`**

Limiting Java Session Memory

These parameters allow the DBA to limit the amount of memory used for each Java session, and are discussed later in this lesson.

`JAVA_SOFT_SESSIONSPACE_LIMIT`

When a user's session-duration Java state exceeds this size, a warning is written into an RDBMS trace file. The default is 1 MB. You use this parameter to specify a soft limit on Java memory usage in a session, as a means to warn you if something is awry.

`JAVA_MAX_SESSIONSPACE_SIZE`

When a user's session-duration Java state attempts to exceed this size, the session is killed with an out-of-memory failure. The default is 4 GB. This limit is purposely set very high so as not to be visible normally. If a user-invoked Java program is not self-limiting in its memory usage, this setting can place a hard limit on the amount of session space made available to it. When the value for this parameter is exceeded, Oracle9*i* displays the following error message:

```
ORA-29554: unhandled Java out of memory condition
```

Sizing the SGA for Java

- `SHARED_POOL_SIZE`:
  - 8 KB per loaded class
  - 50 MB for loading large JAR files
- Configure Oracle Shared Server
- `JAVA_POOL_SIZE`
  - 24 MB default
  - 50 MB for medium-sized Java application

How Oracle9*i* Enterprise Java Engine (EJE) Uses the Shared and Large Pool

The Java Engine uses about 8 KB per loaded class. The shared pool is also temporarily used by the class loader while loading and resolving classes into the database. While loading and resolving particularly large JAR files, you can use 50 MB of shared pool memory.

The UGA, when using shared servers processes, is allocated in the large pool when the `LARGE_POOL_SIZE` is included in the `init.ora` file.

How Oracle9*i* Enterprise Java Engine (EJE) Uses the Java Pool

The Java pool is a structure in the SGA that is used for all session-specific Java code and data within the EJE. During instance startup, the Java pool is allocated a fixed amount of memory equal to the `init.ora` parameter `JAVA_POOL_SIZE`.

Generally, the `JAVA_POOL_SIZE` should be set to 50 MB or higher for large applications. The default value of 24 MB should be adequate for typical Java Stored Procedure usage.

Summary

In this lesson, you should have learned how to:

- Monitor and size the redo log buffer
- Monitor and size the Java pool
- Control the amount of Java session memory used by a session

ORACLE

5-15

Practice 5

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as perfstat/perfstat and collect a snapshot of the current statistics by running the script `$HOME/STUDENT/LABS/snap.sql`. Record the snapshot id for later use.
2. Connect as user SH/SH and run the `$HOME/STUDENT/LABS/lab05_02.sql` script in the `STUDENT/LABS` directory in order to have a workload.
3. Connect as system/manager and query the `V$SYSSTAT` view to determine if there are space requests for the redo log buffer.
4. Connect as perfstat/perfstat and collect another set of statistics using the `$HOME/STUDENT/LABS/snap.sql` script. Then use `$HOME/STUDENT/LABS/spreport.sql` to generate a report using the two snapshot IDs that you have collected. From the report determine log buffer statistics. View the generated file using an editor, and locate the "log buffer space" statistic.
5. Increase the size of the redo log buffer in the `init.ora` file located in the `$HOME/ADMIN/PFILE` directory.

Database Configuration and I/O Issues

Objectives

After completing this lesson, you should be able to do the following:

- List the advantages of distributing different Oracle file types
- List reasons for partitioning data in tablespaces
- Diagnose tablespace usage problems
- Describe how checkpoints work
- Monitor and tune checkpoints
- Monitor and tune redo logs

Oracle Processes and Files

Process	Oracle file I/O			
	Data files	Log	Archive	Control
CKPT	Read/Write			Read/Write
DBWn	Write			
LGWR		Write		Read/Write
ARCn		Read	Write	Read/Write
SERVER	Read/write	Read	Write	Read/Write

Oracle Processes

While most server processes only *read* from disk, `sort direct write` server processes will *write* to disk.

Only the data file headers are accessed by the `CKPT` process. The actual table data is written out by the `DBWn` process.

The SERVER process will read the redo logs, write to the Archive logs and read / write to the control file under certain backup and recovery operations. For example, when performing manual archiving the user process that issued the statement will perform the archiving process.

Performance Guidelines

Basic performance rules are as follows:

- **Keep disk I/O to a minimum.**
- **Spread your disk load across disk devices and controllers.**
- **Use Temporary Tablespaces where appropriate**

Performance Guidelines

For very heavy OLTP applications, there are concurrency problems when using dictionary managed tablespaces because the dictionary needs to be accessed for space management operations during extent allocation. With locally managed tablespaces, there is no dictionary intervention, and therefore fewer concurrency problems.

When users are created, a temporary tablespace is designated for any disk sorts that they will need, and for the creation of temporary tables. These areas should be separate from other database objects.

If users do not have a temporary tablespace, then the SYSTEM tablespace will be used. The DBA should assign a default temporary tablespace so as to overcome this problem.

Tables and indexes should be split into separate tablespaces, because indexes and tables are often inserted into and read from simultaneously. Tables that contain `LOB` data types, for example, `BLOB` and `CLOB,` should have the LOB data placed into a tablespace separate to the actual table.

To improve sort operations, create locally managed temporary tablespaces. A locally managed temporary tablespace avoids Oracle space management operations altogether, since it does not modify data outside of the temporary tablespace or generate any redo for temporary tablespace data.

Distributing Files Across Devices

- **Separate data files and redo log files.**
- **Stripe table data.**
- **Reduce disk I/O unrelated to the database.**

Guidelines

- In general, to reduce the activity on an overloaded disk, move one or more of its heavily accessed files to a less active disk.
  - Redo log files are written sequentially by the `LGWR` process. Put the redo log files on a disk with no other activity or a low incidence of reads and writes: `LGWR` can write much faster if there is no concurrent activity.
  - Redo log files, and archive logs should be placed on different disks
  - If users concurrently access a large table, striping across separate data files and disks can help to reduce contention.
- Try to eliminate I/O unrelated to the Oracle server on disks that contain database files. This is also helpful in optimizing access to redo log files and enables you to monitor all data file activities on such disks with the `V$FILESTAT` dynamic performance view.
- Knowing the types of operation that predominate in your application and the speed with which your system can process the corresponding I/Os, you can choose the disk layout that maximizes performance.

Tablespace Usage

- Reserve the `SYSTEM` tablespace for data dictionary objects.
- Create locally managed tablespaces to avoid space management issues.
- Split tables and indexes into separate tablespaces.
- Create rollback segments in their own tablespaces.
- Store very large objects in their own tablespace.
- Create one or more temporary tablespaces.

6-6

Guidelines

Each database should have separate tablespaces specified for:

- Data dictionary objects
- Rollback segments and undo segments
- Temporary segments
- Tables
- Indexes
- Very large objects. These objects can be CLOBs, BLOBs, tables or partitions.

Most production databases have many more tablespaces than this, but the important principle is to separate data of different types and with different uses for housekeeping and backup purposes.

The SYSTEM tablespace contains only data dictionary objects owned by `sys.` No other users should have the ability to create objects in it. No user should have a quota allocated on the SYSTEM tablespace.

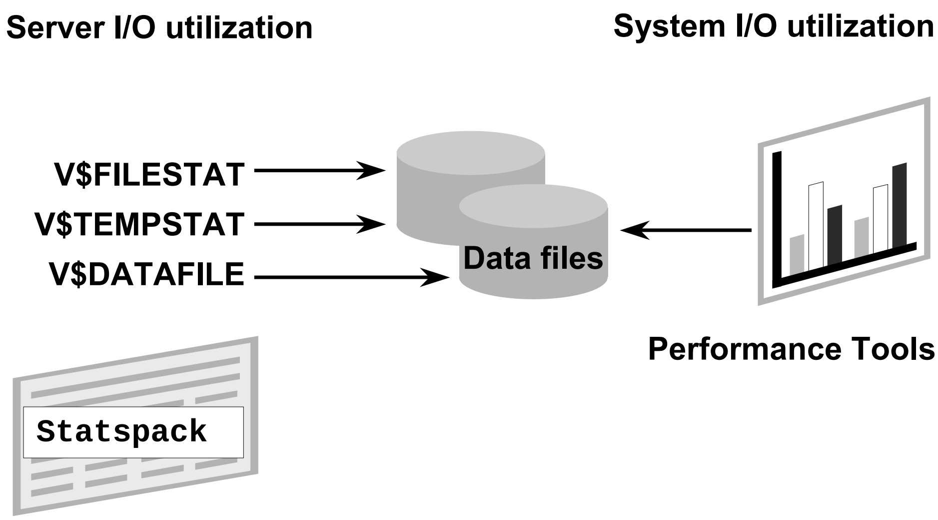
Remember that stored objects such as packages and database triggers form part of the data dictionary, but these objects do not require any space allocation on behalf of the user.

Rollback segments should use rollback segment tablespaces exclusively.

Undo segments can only exist in UNDO tablespaces thus making them exclusive.

`LOB` data types cannot be stored in tablespaces with Automatic Segment Space Management.

Diagnostic Tools for Checking I/O Statistics



Monitoring Use of Files

To monitor which files are subject to most I/O in an existing database, you can query the following:

- `V$FILESTAT` view
- `V$TEMPSTAT` view
- File I/O monitor using Enterprise Manager
- File statistics in `STATSPACK`

Using the V$FILESTAT Dynamic Performance View

You can query V$FILESTAT to find out the number of disk I/Os per disk file.

Summarize all of the I/Os on data files on a per-disk basis to find the data files most likely to cause a disk bottleneck.

V$FILESTAT contains the following columns:

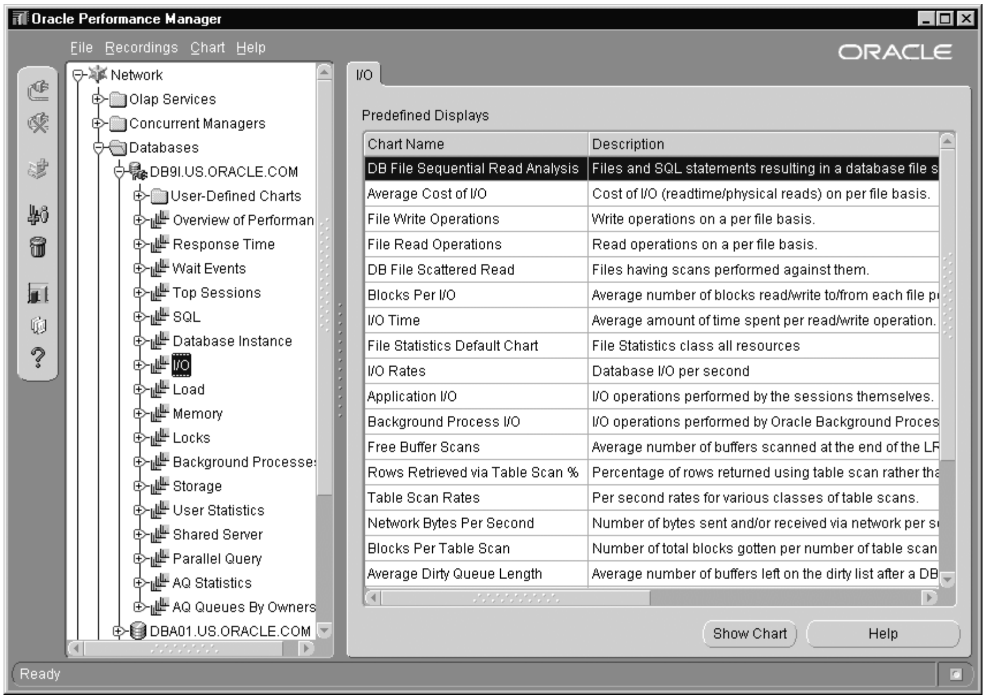
Context	**Reference**
file#	File number (join to file# in V$DATAFILE for the name)
Phyrds	Number of physical reads done
phywrts	Number of physical writes done
phyblkrd	Number of physical blocks read
phyblkwrt	Number of physical blocks written
readtim	Time spent doing reads
writetim	Time spent doing writes

Note: The last two columns contain 0 unless the TIMED_STATISTICS parameter is set to TRUE.

Use the following query to monitor these values:

```
SQL> SELECT phyrds,phywrts,d.name
  2  FROM v$datafile d, v$filestat f
  3  WHERE d.file#=f.file# order by d.name;
    PHYRDS  PHYWRTS    NAME
---------- --------    --------------------------------
       806      116    /…/u01/system01.dbf
       168      675    /…/u04/temp01.dbf
         8        8    /…/u02/sample01.dbf
        26      257    /…/u02/undots01.dbf
     65012      564    /…/u03/users01.dbf
         8        8    /…/u01/query_data01.dbf
  6 rows selected
```

Performance Manager: I/O Statistics



ORACLE

6-9

Performance Manager: I/O Statistics

Many statistics are available using the I/O set of charts. The above screen shot does not show all the charts available. Information regarding how many reads and writes are being performed per data file.

I/O Statistics

```
SQL> select d.tablespace_name TABLESPACE, d.file_name, f.phyrds, f.phyblkrd,
  2  f.readtim, f.phywrts, f.phyblkwrt, f.writetim
  3  from v$filestat f, dba_data_files d
  4  where f.file# = d.file_id
  5  order by tablespace_name, file_name;
TABLESPACE    FILE_NAME              PHYRDS  PHYBLKRD  READTIM PHYWRTS PHYBLKWRT WRITETIM
------------- ---------------------- ------ --------- -------- ------- --------- --------
UNDO1         /u02/undots01.dbf          26        26       50     257       257      411
SAMPLE        /u02/sample01.dbf       65012    416752    38420     564       564     8860
USERS         /u03/users01.dbf            8         8        0       8         8        0
SYSTEM        /u01/system01.dbf         806      1538     1985     116       116     1721
TEMP          /u04/temp01.dbf           168       666      483     675       675        0
QUERY_DATA    /u01/query_data01.dbf       8         8        0       8         8        0
6 rows selected.
```

ORACLE

I/O Statistics

The above statistics are viewed through running the SQL statement shown, however, a similar output is available in the STATSPACK report. Using either method you can observe how well the I/O load is distributed across the disk devices.

In order to correctly determine the good from the bad, it is necessary to have information regarding what type of objects are stored in what tablespaces. For this example:

The USERS tablespace holds tables created by individual users.

The SAMPLE tablespace holds tables that are require by all users on the system.

The QUERY_DATA tablespace holds tables that are read only.

The output shows which files are most active. In the example shown on the slide, the SAMPLE tablespace is being hit heavily. About 98% of the reads are being performed on the data file that contains tables, whereas the QUERY_DATA data file is having 0.001% of the disk reads performed against it. The problem that should be resolved is why the SAMPLE tablespace is being heavily read. Things to look for would be to determine how many full table scans are being performed, are indexes being used, and should you create some indexes on relevant tables.

If the load cannot be reduced, then you should spread the load between other devices.

File Striping

- **Operating system striping:**
  - **Use operating system striping software or a redundant array of inexpensive disks (RAID).**
  - **Decide the right stripe size.**
- **Manual striping:**
  - **Use the `CREATE TABLE` or `ALTER TABLE` command with the `ALLOCATE` clause.**

Operating System Striping

Your operating system may allow striping, in which what appears to be a single contiguous file is actually spread among devices. For example, you can use operating system striping software, such as a logical volume manager.

If your operating system offers striping, you should take advantage of it. You need to think about the size of the stripe, which should be a multiple of the value that you have set for `DB_FILE_MULTIBLOCK_READ_COUNT` (this will be discussed in a subsequent section).

With OS striped database files with a stripe width close to (or less than) `DB_FILE_MULTIBLOCK_READ_COUNT` x `DB_BLOCK_SIZE`, you may get two or more physical reads for each Oracle server read, because you have to access two or more disks.

The most common variety of striped file system is the redundant array of inexpensive disks (RAID). Different levels of RAID striping have varying degrees of safety checks built in.

Manual Striping

You can create tablespaces so that they are made up of multiple files, each on a separate disk. You then create tables and indexes so that they are spread across these multiple files.

Operating System Striping (continued)

You can stripe by:

- Creating objects with MINEXTENTS greater than 1, where each extent is slightly smaller than the striped data files
- Allocating extents to a file explicitly:

```
ALTER TABLE tablename
ALLOCATE EXTENT (DATAFILE 'filename' SIZE 10 M);
```

Keep in mind that striping by hand is a labor-intensive task. The Oracle server fills the extents that you have created one after another. At any given time, one extent is likely to be *hot* and the others less active. If you are using the Parallel Query feature and doing many full table scans, then striping by hand may be worthwhile.

As with many other tuning issues, you can make the right choice only when you have thorough knowledge of how the data is used.

Controller-based striping will usually outperform operating system level striping. Multiple striped file systems can be employed instead of a single large striped set.

Tuning Full Table Scan Operations

- **Investigate the need for full table scans.**
- **Configure the `DB_FILE_MULTIBLOCK_READ_COUNT` initialization parameter to:**
  - **Determine the number of database blocks the server reads at once**
  - **Influence the execution plan of the cost-based optimizer**
- **Monitor long-running full table scans with `V$SESSION_LONGOPS` view.**

Investigating Full Table Scans

If there is high activity on one disk, it is often an untuned query causing the damage. The goal of performance tuning is to increase the effectiveness with which data is accessed.

Tuning Full Table Scans

The DB_FILE_MULTIBLOCK_READ_COUNT initialization parameter determines the maximum number of database blocks read in one I/O operation during a full table scan. The setting of this parameter can reduce the number of I/O calls required for a full table scan, thus improving performance.

I/O is a function of the operating system, so there are limits specific to the operating system imposed on the setting of this parameter. The server's ability to read multiple blocks is limited by the operating system upper limit on the number of bytes that can be read in a single I/O call.

On most platforms before Oracle version 7.3, the maximum "read" memory chunk was 64 KB, so setting the DB_FILE_MULTIBLOCK_READ_COUNT parameter to 64 KB per DB_BLOCK_SIZE gave no extra performance benefit.

For most platforms running Oracle version 7.3, or later, the limit of the DB_FILE_MULTIBLOCK_READ_COUNT parameter is operating system specific. In addition, this parameter is dynamic, so individual sessions can use ALTER SESSION SET command to set a larger size for batch-type work.

Investigating Full Table Scans (continued)

Setting the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter dictates how many I/O calls are required to complete a table scan. For example, if `DB_FILE_MULTIBLOCK_READ_COUNT` is set to 16, and the Oracle block size is 4 KB, then a sequential scan of a 64 KB table can be read in one pass. This improves the speed of the table scan and overall query performance. The goal of setting the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter is to perform table scans with fewer, larger I/O operations. This is done by evaluating the number of blocks required to complete each table scan over time, then adjusting the parameter so that most scans can be performed in one I/O.

The total number of I/Os actually required to perform a full table scan also depends on other factors, such as the size of the table and whether Parallel Query is being used.

The cost-based optimizer uses all of these factors, including the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter, to determine the cost of full table scans. The cost-based optimizer favors full table scans when the cost is lower than index scans.

Table Scan Statistics

```
SQL>  SELECT name, value FROM v$sysstat
 2    WHERE name LIKE '%table scan%';
NAME                                                VALUE
------------------------------------------------ -----
table scans (short tables)                           125
table scans (long tables)                             30
table scans (rowid ranges                              0
table scans (cache partitions)                         0
table scans (direct read)                              0
table scan rows gotten                             21224
table scan blocks gotten                             804
7 rows selected.
```


Table Scan Statistics

The query on the above slide provides an overview of how many full table scans are taking place.

The values for `table scans (long tables)` and `table scans (short tables)` relate to full table scans. A `long table` is one with more than 4 blocks, and a `short table` is 4 or less blocks.

If the value of `table scans (long tables)` is high, then a large percentage of the tables accessed were not indexed lookups. Your application may need tuning, or you should add indexes. Make sure that the appropriate indexes are in place, and valid.

Monitoring Full Table Scan Operations

Determine the progress of long operations using:

```
SQL> SELECT sid, serial#, opname,
  2  TO_CHAR(start_time,'HH24:MI:SS')AS START,
  3    (sofar/totalwork)*100 AS PERCENT_COMPLETE
  4  FROM v$session_longops;
```

Use SET_SESSION_LONGOPS to populate V$SESSION_LONGOPS

```
dbms_application_info.set_session_longops(rindex, slno,
"Operation X", obj, 0, sofar, totalwork, "table",
"tables");
```

6-16

ORACLE

Monitoring Full Table Scan Operations

Users and DBAs can monitor the progress of full table scans and get an idea of the estimated completion time. The Oracle server maintains statistics tracking the progress of such operations and makes it available to the users through the V$SESSION_LONGOPS dynamic performance view.

The DBMS_APPLICATION_INFO package contains a procedure, SET_SESSION_LONGOPS, to populate the view from an application. The procedure has the following parameters:

Rindex	A token which represents the v$session_longops row to update. Set this to set_session_longops_nohint to start a new row. Use the returned value from the prior call to reuse a row.
slno	Saves information across calls to set_session_longops: It is for internal use and should not be modified by the caller.
op_name	Specifies the name of the long running task. It appears as the OPNAME column of v$session_longops. The maximum length is 64 bytes. Default value of NULL.
target	Specifies the object that is being worked on during the long running operation. For example, it could be a table ID that is being sorted. It appears as the TARGET column of v$session_longops. Default value 0.

Monitoring Full Table Scan Operations (Cont.)

context — Any number the client wants to store. It appears in the CONTEXT column of v$session_longops. Default value 0.

sofar — Any number the client wants to store. It appears in the SOFAR column of v$session_longops. This is typically the amount of work which has been done so far. Default value 0.

totalwork — Any number the client wants to store. It appears in the TOTALWORK column of v$session_longops. This is typically an estimate of the total amount of work needed to be done in this long running operation. Default value 0.

target_desc — Specifies the description of the object being manipulated in this long operation. This provides a caption for the target parameter. This value appears in the TARGET_DESC field of v$session_longops. The maximum length is 32 bytes. Default value is ‘unknown target’.

Units-Specifies the units in which sofar and totalwork are being represented. It appears as the UNITS field of v$session_longops. The maximum length is 32 bytes. Default value is NULL.

In the example as the process completes each object, Oracle will update V$SESSION_LONGOPS on the procedure's progress.

```
DECLARE
    rindex    BINARY_INTEGER;
    slno      BINARY_INTEGER;
    totalwork number;
    sofar     number;
    obj       BINARY_INTEGER;
  BEGIN
    rindex := dbms_application_info.set_session_longops_nohint;
    sofar := 0;
    totalwork := 10;
    WHILE sofar < 10 LOOP
      -- update obj based on sofar
      -- perform task on object target
      sofar := sofar + 1;
      dbms_application_info.set_session_longops(rindex, slno,
        "Operation X", obj, 0, sofar, totalwork, "table", "tables");
    END LOOP;
  END;
```

Checkpoints

The two most common types of Checkpoint are:

- **Incremental Checkpoint**
  - **CKPT updates the control file**
  - **During a Log Switch CKPT updates the control file and the data file headers**
- **Full Checkpoints**
  - **CKPT updates the control file and the data file headers**
  - **DBWn writes out all buffers on the Checkpoint Queue**

Checkpoints

Incremental Checkpoint

During the normal running of the database, buffers that get modified (dirty buffers) are added to the checkpoint queue. The checkpoint queue is a linked list of all dirty buffers, in the order in which they were first modified. A dirty buffer is only listed once in the checkpoint queue, when it is first modified. `DBWn` writes dirty blocks in the order found in the checkpoint queue. Blocks are removed from the checkpoint queue as soon as `DBWn` writes them. Therefore the first entry in the checkpoint queue is the block that has been dirty the longest time.

Each record in the redo log is assigned a redo byte address (RBA). When a block is first changed it is entered in the checkpoint queue. The RBA of the redo record recording that change is also included. All buffers that were modified prior to the first buffer in the checkpoint queue will already have been written to the data files.

Every three seconds the `CKPT` process records the RBA from the first entry in the checkpoint queue to the control file In the event of instance failure this RBA will give the location in the redo logs at which to start recovery. This RBA is referred to as the checkpoint position. At a log switch the database will, in addition, update the checkpoint information in the header block of each data file.**Note:** This Checkpoint does *not* force a write of any data to the data files.

Checkpoints

- **Two categories of Full Checkpoints**
  - **Complete**
  - **Tablespace**

6-19

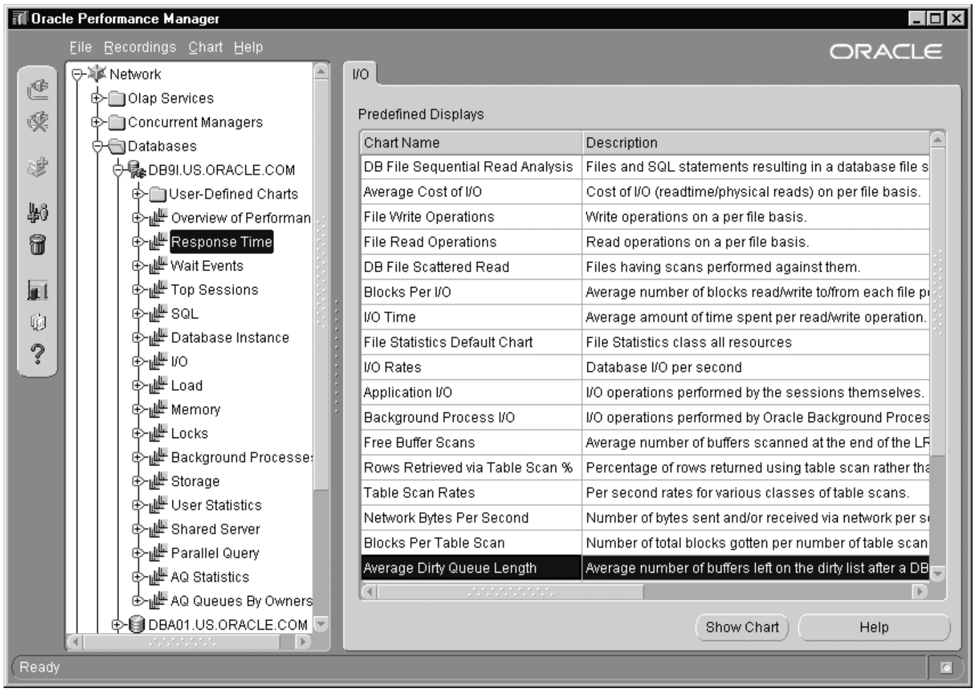
Checkpoints (continued)

Full Checkpoint

A Full Checkpoint can be broken down into two categories:

- Complete Checkpoint
  - All modified buffers are written to the data files, and then the checkpoint information is written to the control file and data file headers.
  - This form of a checkpoint is performed during an instance shutdown (not for an instance crash or shutdown abort).
  - A Full Complete Checkpoint can also be manually performed by using the `"ALTER SYSTEM CHECKPOINT;"` command
- Tablespace Checkpoint
  - Occurs when the command:
    `"ALTER TABLESPACE <tname>  BEGIN BACKUP;"`
    is issued.
  - This will start `DBWn` writing only the dirty buffers that belong to the tablespace being backed up.
  - Occurs when a tablespace is taken offline.

Performance Manager: Response Time



ORACLE

Performance Manager: Response Time

Using the Average Dirty Queue length chart under the Response Time section it is possible to determine the average length of the checkpoint queue.

Regulating the Checkpoint Queue

Regulate the checkpoint queue with the following initialization parameters:

- **FAST_START_IO_TARGET**
- **LOG_CHECKPOINT_INTERVAL**
- **LOG_CHECKPOINT_TIMEOUT**
- **FAST_START_MTTR_TARGET**

ORACLE

Regulating the Checkpoint Queue

The length of the checkpoint queue is going to be a determining factor in the period of time required to recover an instance after failure. For faster recovery a shorter queue should be maintained. However, a shorter queue means more writes for DBW*n* to perform, which can adversely affect performance. Obviously a compromise must be found.

Use the following parameters in order to control the length of the checkpoint queue:

- LOG_CHECKPOINT_TIMEOUT: The number of seconds that has passed between the checkpoint position and the last write to the redo. No block in the buffer cache will be dirty longer than this time.
- LOG_CHECKPOINT_INTERVAL: The number of *operating system* blocks in the redo log between the checkpoint position and the end of the redo log.
- FAST_START_IO_TARGET: The number of I/O operations the database should take to perform crash recovery of a single instance.
- FAST_START_MTTR_TARGET: The average number of seconds the database should take to perform crash recovery of a single instance.

In addition, the number of redo blocks between the checkpoint position and the end of the redo log will never be more than 90% of the length of the smallest log file.

Whichever of these parameters causes the shortest recovery time will be used to determine the checkpoint position.

Defining and Monitoring FASTSTART Checkpointing

- Use V$INSTANCE_RECOVERY in order to get the following information:
  - RECOVERY_ESTIMATED_IOS
  - LOG_FILE_SIZE_REDO_BLKS
  - LOG_CHKPT_TIMEOUT_REDO_BLKS
  - LOG_CHKPT_INTERVAL_REDO_BLKS
  - TARGET_MTTR
  - ESTIMATED_MTTR

6-22

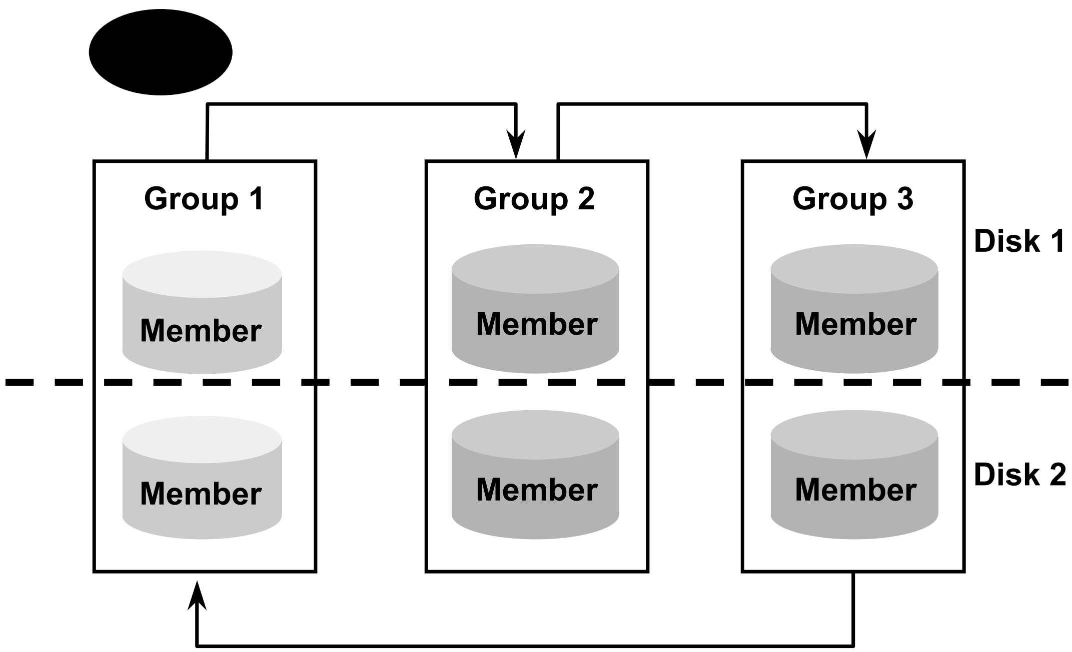
To specify a value for FAST_START_IO_TARGET, decide on the required service level after discussion with users. Translate this time to equivalent number of data file I/O by using the average I/O time statistic from the view V$FILESTAT.

Monitoring Impact of Parameters on Recovery Time

Use the V$INSTANCE_RECOVERY view to obtain the following information:

- RECOVERY_ESTIMATED_IOS: The estimated number of data blocks to be processed during recovery based on the in-memory value of the fast-start checkpoint parameter
- ACTUAL_REDO_BLKS: The current number of redo blocks required for recovery
- TARGET_REDO_BLKS: The goal for the maximum number of redo blocks to be processed during recovery. This value is the minimum of the following 4 columns.
- LOG_FILE_SIZE_REDO_BLKS: The number of redo blocks to be processed during recovery to guarantee that a log switch never has to wait for a checkpoint
- LOG_CHKPT_TIMEOUT_REDO_BLKS: The number of redo blocks that need to be processed during recovery to satisfy LOG_CHECKPOINT_TIMEOUT
- LOG_CHKPT_INTERVAL_REDO_BLKS: The number of redo blocks that need to be processed during recovery to satisfy LOG_CHECKPOINT_INTERVAL
- TARGET_MTTR: Effective mean time to recover target value in seconds. This value is calculated based on the value of the FAST_START_MTTR_TARGET parameter, and the system limitations.
- ESTIMATED_MTTR: The current estimated mean time to recover. Basically, the time recovery would take based on the work your system is doing right now.

Redo Log Groups and Members



Redo Log Groups and Members

The above diagram shows one method of assigning redo log members to disk space. This method will support redo logging adequately. If archiving is enabled, it may be necessary to have groups on different disks as well as the members so that archiving will not contend with the redo log writer.

Online redo log files are organized in groups. A group must have one or more members. All members of a group have identical contents. You should have two or more members in each group for safety, unless you are mirroring all files at a hardware level.

Online Redo Log File Configuration

- **Size redo log files to minimize contention.**
- **Provide enough groups to prevent waiting.**
- **Store redo log files on separate, fast devices.**
- **Monitor the redo log file configuration with:**
  - **V$LOGFILE**
  - **V$LOG**
  - **V$LOG_HISTORY**

Online Redo Log File Configuration

Redo log files in the same group should ideally be on separate, fast devices, because LGWR writes to them almost continuously. Properly size redo log files to minimize contention and frequency of log switches; as a guide, a redo log file should be able to contain 20 minutes of redo data. You can use the Redo Size statistics in the STATSPACK report to help determine the proper size for your redo log files.

Monitoring Redo Log File Information

You can query the V$LOGFILE and V$LOG dynamic performance views to obtain information about the name, location, size, and status of the online redo log file.

Any waits for Log File Parallel Write in V$SYSTEM_EVENT indicate a possible I/O problem with the log files.

The Oracle server does not provide a means of monitoring redo disk I/Os, so you must use operating system disk monitoring commands. On most UNIX systems, sar (system activity reporter) is useful for this purpose.

Monitoring Redo Log File Information (continued)

```
# sar -d 1 1

SunOS stc-sun101 5.6 Generic_105181-16 sun4u    02/01/01

22:30:03   device        %busy   avque   r+w/s  blks/s  avwait  avserv

22:30:04
           sd0              93     0.9     179     657     0.0     5.2
           sd0,a            93     0.9     179     657     0.0     5.2
           sd0,b             0     0.0       0       0     0.0     0.0
           sd0,c             0     0.0       0       0     0.0     0.0
           sd1               0     0.0       0       0     0.0     0.0
           sd1,c             0     0.0       0       0     0.0     0.0
           sd1,h             0     0.0       0       0     0.0     0.0
           sd6               0     0.0       0       0     0.0     0.0
```

`%busy` is the percentage of time the device was busy during the polling period, `avque` is the *average* queue size, `r+w/s` and `blks/s` are read and writes per second and blocks per second, respectively, `avwait` is the *average* wait time per request, and `avserv` is number of miliseconds per *average* seek. The `sar` command can be set up to record historical system information, making it even more useful to the DBA.

Another useful UNIX utility for monitoring disk activity is `iostat`. The output of `iostat` is simpler than that of `sar`:

```
# iostat -D
          sd0           sd1           sd6          nfs1
 rps wps util  rps wps util  rps wps util  rps wps util
   2   1  1.4    7   2  6.0    0   0  0.0    0   0  0.0
```

`rps` and `wps` are reads per second and writes per second, respectively, and `util` is the percentage of disk utilization.

Archive Log File Configuration

- **Allow the LGWR process to write to a disk different from the one the ARCn process is reading.**
- **A quick solution is to share the archiving work:**

```
ALTER SYSTEM ARCHIVE LOG ALL
TO <log_archive_dest>
```

- **Increase the number of Archive processes**
- **Change archiving speed:**
  - **LOG_ARCHIVE_MAX_PROCESSES**
  - **LOG_ARCHIVE_DEST_n**

Archive Log File Configuration

If you choose to archive, it is even more important to have more than two redo log groups.When a group switches to another group, the DBW*n* process must checkpoint as usual, and one file must be archived. You need to allow time for both of these operations before the LGWR process needs to overwrite the file again.

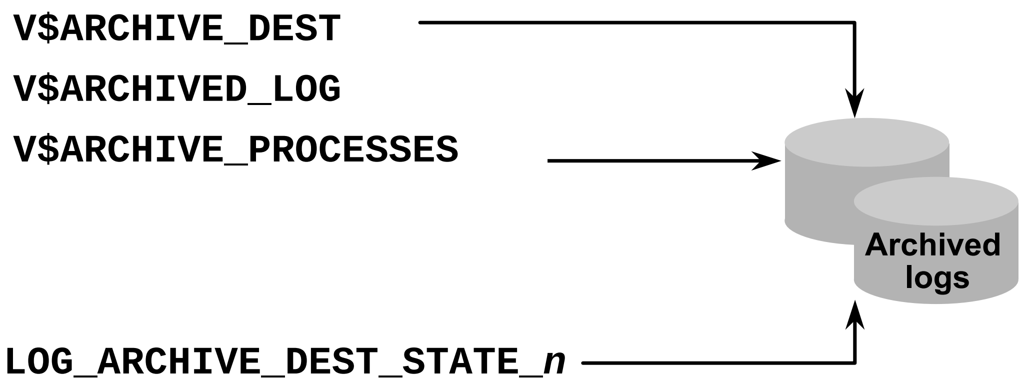
Obtaining Information about Archived Log Files and Their Location

You can query the V$ARCHIVED_LOG dynamic performance view to display archived log information from the control file, including archive log names. An archive log record is inserted after the online redo log is successfully archived or cleared (the name column is null if the log was cleared).

The V$ARCHIVE_DEST dynamic performance view describes, for the current instance, all the archive log destinations, as well as their current values, modes, and statuses.

Note: The LOG_ARCHIVE_DEST_*n* parameter is valid only if you have installed the Oracle Enterprise Edition. You can continue to use LOG_ARCHIVE_DEST if you have installed the Oracle Enterprise Edition. However, you cannot use both LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST because they are not compatible.

Diagnostic Tools



6-27

Regulating Archiving Speed

- Occasionally, in busy databases, a single `ARC0` process cannot keep up with the volume of information written to the redo logs. Oracle9*i* allows the DBA to define multiple archive processes by using the `LOG_ARCHIVE_MAX_PROCESSES` parameter.
- The `LGWR` process starts a new `ARCn` process whenever the current number of `ARCn` processes is insufficient to handle the workload. If you anticipate a heavy workload for archiving, you can get another process to share the work by regularly running a script containing the command:
  `SQL> ALTER SYSTEM ARCHIVE LOG ALL TO 'directory_name';`
- Monitor `V$ARCHIVE_PROCESSES`. There is one row for each Archive process. The STATUS column shows the state of the Archive process, (`STOPPED, SCHEDULED, STARTING, ACTIVE, STOPPING,` and `TERMINATED`). The STATE column indicates whether the process is currently `BUSY,` or `IDLE`.

```
SQL> select * from v$archive_processes;
PROCESS          STATUS     LOG SEQUENCE         STATE
-------          --------   --------------       --------
      0          ACTIVE               122        BUSY
      1          ACTIVE                 0        IDLE
      2          STOPPED                0        IDLE
```

Note: The number of processes used by the `database` is automatically set to 4 when the `DBWR_IO_SLAVES` parameter is set to a value greater than 0.

Summary

In this lesson, you should have learned how to:

- List the advantages of using different Oracle file types
- List reasons for segmenting data in tablespaces
- Diagnose tablespace usage problems
- Describe how checkpoints work
- Monitor and tune checkpoints
- Monitor and tune archive logging

ORACLE

6-28

Practice 6

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as system/manager and diagnose database file configuration by querying the `V$DATAFILE, V$LOGFILE` and `V$CONTROLFILE` dynamic performance views.
2. Diagnose database file usage by querying the `V$FILESTAT` dynamic performance view, combine with `V$DATAFILE` in order to get the data file names
3. Determine if there are waits for redo log files by querying the `V$SYSTEM_EVENT` dynamic performance view, where the waiting event is 'log file sync' or 'log file parallel write'. Some waits you can look for:

   'log file sync' waits are indicative of slow disks that store the online logs.
   'log file parallel write' is much less useful. The reason is that this event only shows how often LGWR waits, not how often server processes wait. If `LGWR` waits without impacting user processes, there is no performance problem. If `LGWR` waits, it is likely that the 'log file sync' event will also be evident.
4. Connect as perfstat/perfstat and diagnose file usage from `STATSPACK.`
   - Generate a `STATSPACK` report using `$HOME/STUDENT/LABS/spreport.sql.`
   - Locate and open the report file.
   - Examine the report, and search for the string "File IO Stats"

     Note: On a production database care should be taken in monitoring the disk, and controller usage by balancing the workload across all devices. If your examination shows a distinct over utilization of a particular data file, consider resolving the cause of the amount of I/O. For example, investigate the number of full table scans, clustering of files on a specific device, and under utilization of indexes. If after this the problem remains then look at placing the data file on a low utilization device.
5. Connect as system/manager and enable checkpoints to be logged in the alert file by setting the value of the `log_checkpoints_to_alert` parameter to `TRUE` using "alter system set" command.
6. Connect as sh/sh and execute the `$HOME/STUDENT/LABS/lab06_06.sql` script to provide a workload against the database.
7. At the operation system level use the editor to open the alert log file (located in the directory specified by `BACKGROUND_DUMP_DEST`). Then determine the checkpoint frequency for your instance by searching for messages containing the phrase "Completed Checkpoint." The time difference between two consecutive messages is the checkpoint interval

7

Optimizing Sort Operations

Objectives

After completing this lesson, you should be able to do the following:

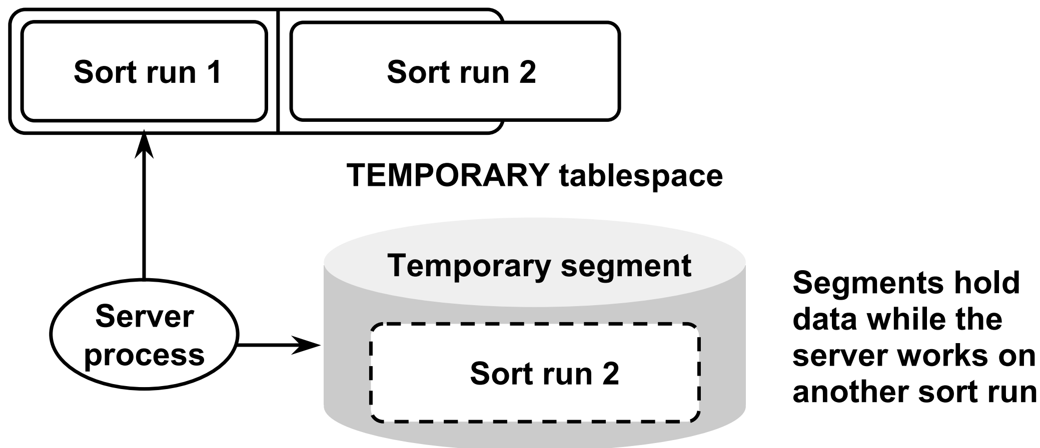
- Describe how sorts are performed
- Identify the SQL operations that require sorts
- Differentiate between disk and memory sorts
- Create and monitor temporary tablespaces
- List ways to reduce total sorts and disk sorts
- Determine the number of sorts performed in memory
- Set old and new sort parameters

ORACLE

7-2

The Sorting Process

If sort space requirement is greater than SORT_AREA_SIZE:



The Sorting Process

The server sorts in memory if the work can be done within an area smaller than the value (in bytes) of the `SORT_AREA_SIZE` parameter.

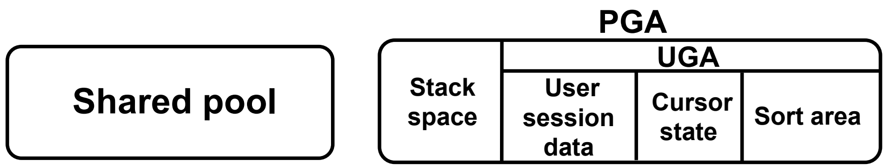
If the sort needs more space than this value:

1. The data is split into smaller pieces, called sort runs; each piece is sorted individually.
2. The server process writes pieces to temporary segments on disk; these segments hold intermediate sort run data while the server works on another sort run.
3. The sorted pieces are merged to produce the final result. If `SORT_AREA_SIZE` is not large enough to merge all the runs at once, subsets of the runs are merged in a number of merge passes.

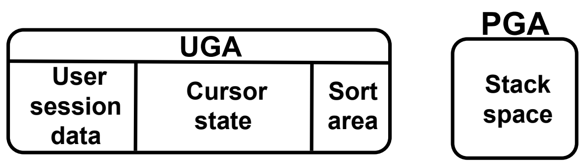
Sort Area and Parameters

The sort space is in:

- **The PGA for a dedicated server connection**



- **The shared pool for Oracle Shared Server connection**



Sort Area

The sort space:

- Is part of the PGA when connected with a dedicated server
- Is part of the shared pool when connected with Oracle Shared Server

Note: An application doing large sorts should not be using Oracle Shared Server.

Parameters

`SORT_AREA_SIZE`

- The sort area is sized with the `SORT_AREA_SIZE` `init.ora` parameter.
- It can be set dynamically, using the `ALTER SESSION` or `ALTER SYSTEM DEFERRED` command.
- The default value is dependent on the operating system.
- The default is generally adequate for most OLTP operations. Adjust it upward for DSS applications, batch jobs, or large operations.

Parameters (continued)

SORT_AREA_RETAINED_SIZE

- When the sorting is complete and the sort area still contains sorted rows to be fetched, the sort area can shrink to the size specified by the SORT_AREA_RETAINED_SIZE parameter.
- The memory is released back to the UGA for use by the same Oracle server process (not to the operating system) after the last row is fetched from the sort space.
- The default value for this parameter is equal to the value of the SORT_AREA_SIZE parameter.

SORT_AREA_RETAINED_SIZE is allocated from the UGA, which resides in shared or private memory depending on if the session has a shared or dedicated server. Anything else up to SORT_AREA_SIZE is allocated from the CGA, which is always part of the PGA. Sort memory is only allocated as needed.

Sort Area and Parameters

- An execution plan can contain multiple sorts.
- A single server needs:
  - An area of SORT_AREA_SIZE, in bytes, for an active sort
  - At least one area of SORT_AREA_RETAINED_SIZE for a join sort
- Each parallel query server needs SORT_AREA_SIZE.
- Two sets of servers can be writing at once, so:
  - Calculate SORT_AREA_SIZE × 2 × degree of parallelism
  - Add SORT_AREA_RETAINED_SIZE × degree of parallelism × number of sorts above two

7-6

Memory Requirements

Single Server Process

An execution plan can contain multiple sorts. For example, a sort-merge join of two tables may be followed by a sort for an ORDER BY clause. This gives three sorts all together.

If a single server works on the sort, then while it does the ORDER BY sort it uses:

- An area of SORT_AREA_SIZE, in bytes, for the active sort
- Two areas of the size specified by SORT_AREA_RETAINED_SIZE for the join sorts

Parallel Query Processes

If you parallelize the statement, each query server needs SORT_AREA_SIZE amount of memory.

With parallel query, two sets of servers can be working at once, so you should:

- Calculate SORT_AREA_SIZE × 2 × degree of parallelism
- If necessary, add SORT_AREA_RETAINED_SIZE × degree of parallelism × number of sorts above two

If you can still afford the memory, the optimal value for SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE with Parallel Query is one megabyte. In testing, larger values than this have not improved performance significantly.

Sizes

Usually, SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE should be set to the same value, unless:

- You are very short of memory
- You are using Oracle Shared Server

Initialization Parameters for Bitmap Indexing:

- CREATE_BITMAP_AREA_SIZE:
  - Value is static, thus can only be changed by restarting the database
  - Amount of memory allocated for bitmap creation
  - Not dynamically alterable at the session level
  - Default value: 8 MB (A larger value may lead to faster index creation. If the cardinality is very small, however, you can set a small value for this parameter. For example, if the cardinality is only two, then the value can be on the order of kilobytes. As a general rule, the higher the cardinality, the more memory is needed for optimal performance.)
- BITMAP_MERGE_AREA_SIZE:
  - Value is static, thus can only be changed by restarting the database
  - Amount of memory used to merge bitmaps retrieved from a range scan of the index
  - Not dynamically alterable at the session level
  - Default value: 1 MB (A larger value should improve performance, because the bitmap segments must be sorted before being merged into a single bitmap.)

New Sort Area Parameters

- **Parameters for automatic sort area management:**
  - **PGA_AGGREGATE_TARGET**
  - **(Ranges from 10 MB to 4000 GB)**
  - **WORKAREA_SIZE_POLICY**
  - **AUTO | MANUAL**
- **Replace all *_AREA_SIZE parameters**

New Sort Area Parameters

PGA_AGGREGATE_TARGET

PGA_AGGREGATE_TARGET specifies the target aggregate PGA memory of all server processes attached to the instance. The value of this parameter ranges from 10 MB to 4000 GB.

When setting this parameter, you should examine the total memory on your system that is available to the Oracle instance and subtract the SGA, then assign the remaining memory to PGA_AGGREGATE_TARGET.

WORKAREA_SIZE_POLICY

Accepted values are:

- AUTO

  You can specify AUTO only when PGA_AGGREGATE_TARGET is defined.
- MANUAL

  The sizing of work areas is manual and based on the values of the *_AREA_SIZE parameter corresponding to the operation (for example, a sort uses SORT_AREA_SIZE). Specifying MANUAL may result in suboptimal performance and poor PGA memory utilization.

Tuning Sorts

- **Avoid sort operations whenever possible.**
- **Reduce swapping and paging by making sure that sorting is done in memory when possible.**
- **Reduce space allocation calls by allocating temporary space appropriately.**

7-9

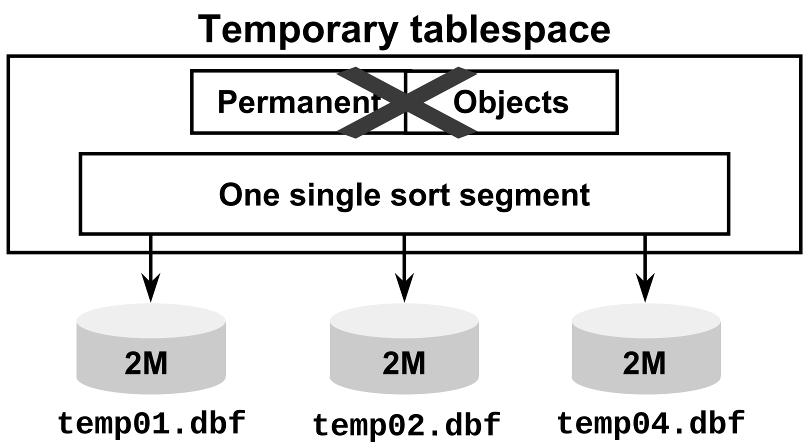
Diagnosing Problems

- You may be able to avoid sorts if the processed data has been sorted previously.
- When sorts are not too large, a sort area that is too small results in performance overheads by swapping to disk; ensure that the sort operations occur in memory, whenever possible.
- Using large chunks of memory for sorting can result in paging and swapping, and reduce overall system performance.
- If permanent tablespaces are used instead of temporary tablespaces for sorts on disk, then the frequent allocation, and deallocation, of temporary segments can cause latch contention and performance problems.

Tuning Goals

- Avoiding sort operations that are not necessary
- Optimizing memory sort and disk overhead
- Eliminating space allocation calls to allocate and deallocate temporary segments

The Sorting Process and Temporary Space



Create a temporary tablespace by using:

```
CREATE TEMPORARY TABLESPACE TEMP TEMPFILE
‘$HOME/ORADATA/u06/temp01.dbf’ size 200M;
```


Advantage of Temporary Tablespaces

Designating temporary tablespaces for sorts effectively eliminates serialization of space management operations involved in the allocation and deallocation of sort space.

A temporary tablespace:

- Cannot contain any permanent objects
- Can contain temporary objects
- Contains a single sort segment per instance for Oracle Parallel Server environments

A temporary tablespace has to have temporary files. The advantage of these files is that they do not need to be a part of the backup strategy, thus saving time during the backup process.

Every user has a temporary tablespace, by default the temporary tablespace is set to SYSTEM, unless the DBA assigns a Default Temporary Tablespace. When a user creates a temporary table, or requires a sort area, the user cannot specify a temporary storage area. It is created in the temporary tablespace assigned to that user.

Temporary Space Segments

A temporary space segment:

- Is created by the first sort
- Extends as demands are made on it
- Comprises extents, which can be used by different sorts
- Is described in the sort extent pool (SEP)

7-11

The Sort Segment

- Created at the time of the first sort operation that uses the tablespace
- Dropped when the database is closed
- Grows as demands are made on it
- Made up of extents, each of which can be used by different sort operations
- Described in an SGA structure called the sort extent pool (SEP). When a process needs sort space, it looks for free extents in the SEP.

Operations Requiring Sorts

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

Index Creation

The server process (or processes, if the index is being created in parallel) has to sort the indexed values before building the B-tree.

ORDER BY or GROUP BY Clauses

The server process must sort on the values in the ORDER BY or GROUP BY clauses.

DISTINCT Values

For the DISTINCT keyword, the sort has to eliminate duplicates.

UNION, INTERSECT, or MINUS Operators

Servers need to sort the tables they are working on to eliminate duplicates.

Sort-Merge Joins

```
SQL> select department_name, Last_name
 2>  from employees e, departments d
 3>  where e.department_id = d.department_id;
```

Sort-Merge Joins (continued)

If there are no indexes available, an equijoin request needs to:

- Perform full table scans of `EMPLOYEES` and `DEPARTMENTS` tables
- Sort each row source separately
- Merge the sorted sources together, combining each row from one source with each matching row of the other source

Before the sort operation:

```
SQL> select name, value
  2> from v$sysstat where name = 'sorts (rows)';
    NAME                                          VALUE
    ----------------- ------------------------------
    sorts (rows)                                   639330
```

After the sort operation:

```
SQL> select name, value
  2> from v$sysstat where name = 'sorts (rows)';
NAME                                              VALUE
----------------- ------------------------------
sorts (rows)                                     655714
```

`DBMS_STATS` Execution

The `DBMS_STATS` package is useful for collecting statistics on tables, indexes, and clusters to help the CBO define the best execution plans. It sorts the data to provide summarized information.

```
SQL> execute dbms_stats.gather_table_stats ('hr', employees');
  PL/SQL procedure successfully completed.
```

Avoiding Sorts

Avoid sort operations whenever possible:

- **Use `NOSORT` to create indexes.**
- **Use `UNION ALL` instead of `UNION`.**
- **Use index access for table joins.**
- **Create indexes on columns referenced in the `ORDER BY` clause.**
- **Select the columns for analysis.**
- **Use `ESTIMATE` rather than `COMPUTE` for large objects.**

7-14

ORACLE

The NOSORT Clause

Use the NOSORT clause when creating indexes for presorted data on a single-CPU machine using SQL*Loader. This clause is only valid for the data inserted into a table:

```
SQL> create index EMPLOYEES_DEPARTMENT_ID_FK on
  employees(department_id) NOSORT;
ORA-01409: NOSORT option may not be used; rows are not in
  ascending order
```

On a multi-CPU machine, it is probably quicker to load data in parallel, even though it is not loaded in order. Then you can use parallel index creation to speed up sorting.

UNION ALL

Use UNION ALL instead of UNION; this clause does not eliminate duplicates, so does not need to sort.

Nested Loop Joins

Use index access for equijoin requests:

```
SQL> select department_name, Last_name
 2>  from employees e, departments d
 3>  where e.department_id = d.department_id;
```

Nested Loop Joins (continued)

The optimizer chooses a nested loop join instead of a sort-merge join. A nested loop join does not require any sorts. The steps necessary to do this are:

1. Perform a full table scan of the `employees` table.
2. Use the `DEPARTMENT_ID` value for each row returned to perform a unique scan on the `primary key` index.
3. Use the `ROWID` retrieved from the index scan to locate the matching row in the `departments` table.
4. Combine each row returned from `employees` with the matching row returned from `departments`.

Indexes and `ORDER BY`

Create indexes on columns that are frequently referenced with `ORDER BY` statements. The server will use the index rather a sort operation, because the index is ordered. The index must be created in a manner that will match the ORDER BY clause.

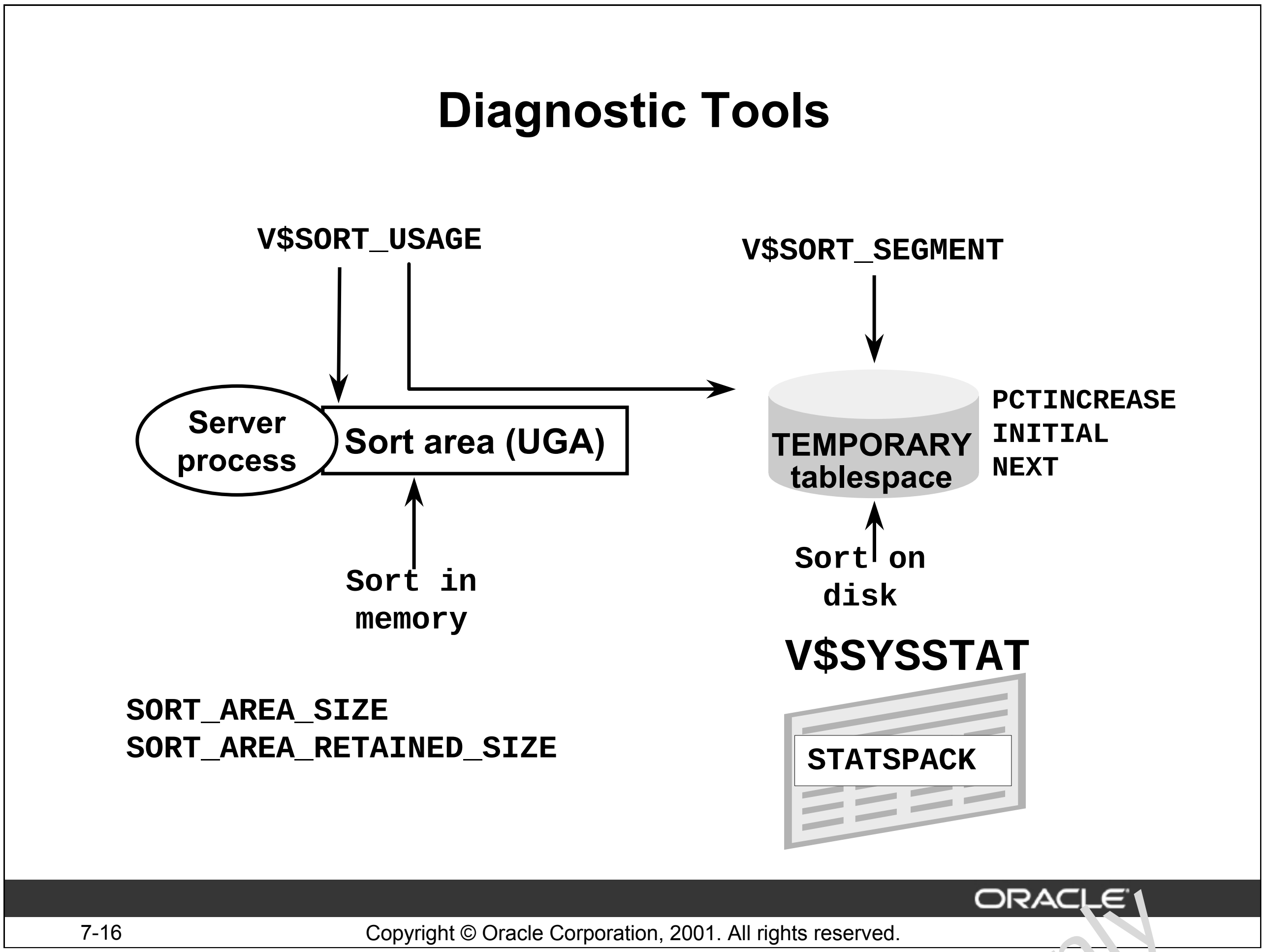
`ANALYZE FOR COLUMNS`

Collect statistics for the columns of interest only; for example, those involved in join conditions, `ANALYZE... FOR COLUMNS` or `ANALYZE...FOR ALL INDEXED COLUMNS.`

Note: The `ANALYZE... SIZE` *`n`* command creates histograms for the columns involved. Combining this clause with the `FOR ALL INDEXED COLUMNS` clause generates unnecessary histograms for primary keys and unique constraints.

`ANALYZE ESTIMATE`

The `COMPUTE` clause is more precise for the optimizer. However, it demands a large amount of sort space. The `ESTIMATE` clause is preferable for large tables and clusters.



Dynamic Views and STATSPACK Output

The V$SYSSTAT view displays the number of sorts in memory, sorts on disk, and rows being sorted:.

- `Sorts (disk)`: Number of sorts requiring I/O to temporary segments
- `Sorts (memory)`: Number of sorts performed entirely in memory
- `Sorts (rows)`: Total rows sorted in the period being monitored

```
  SQL> select * from v$sysstat where name like '%sorts%';
  STATISTIC# NAME                             CLASS     VALUE
---------------------------                   -------   ------
  161 sorts (memory)                          64        154
  162 sorts (disk)                            64        4
  163 sorts (rows)                            64        571768
```

Dynamic Views and STATSPACK Output (continued)

The STATSPACK output gives the same information. Moreover, these figures cover the period when the snapshots ran.

```
Statistic              TotalPer Transact     Per Logon     Per Second
-----------      --------- ----------     ----------    -----------
sorts (disk)             4        .02            .41            .01
sorts (memory)         154        .27           5.77            .12
sorts (rows)        571768      39.62         862.59          18.19
```

The V$SORT_SEGMENT and V$SORT_USAGE views display information about the temporary segments used and the users who used them.

Diagnostics and Guidelines

```
SQL> select disk.value "Disk", mem.value "Mem",
  2         (disk.value/mem.value)*100 "Ratio"
  3  from  v$sysstat mem, v$sysstat disk
  4  where mem.name = 'sorts (memory)'
  5  and   disk.name = 'sorts (disk)';
     Disk         Mem       Ratio
---------   ---------   ---------
       23         206   11.165049
```

- **In an OLTP system the ratio of disk sorts to memory sorts should be less than 5%.**
  - **Increase the value of SORT_AREA_SIZE / PGA_AGGREGATE_TARGET if the ratio is greater than 5%.**

7-18

Ratio

In an OLTP system the ratio of sorts (disk) to sorts (memory) should be less than 5%. If the ratio indicates a high number of sorts going to disk, increase the value of SORT_AREA_SIZE/ PGA_AGGREGATE_TARGET. This increases the size of each run and decreases the total number of runs and merges.

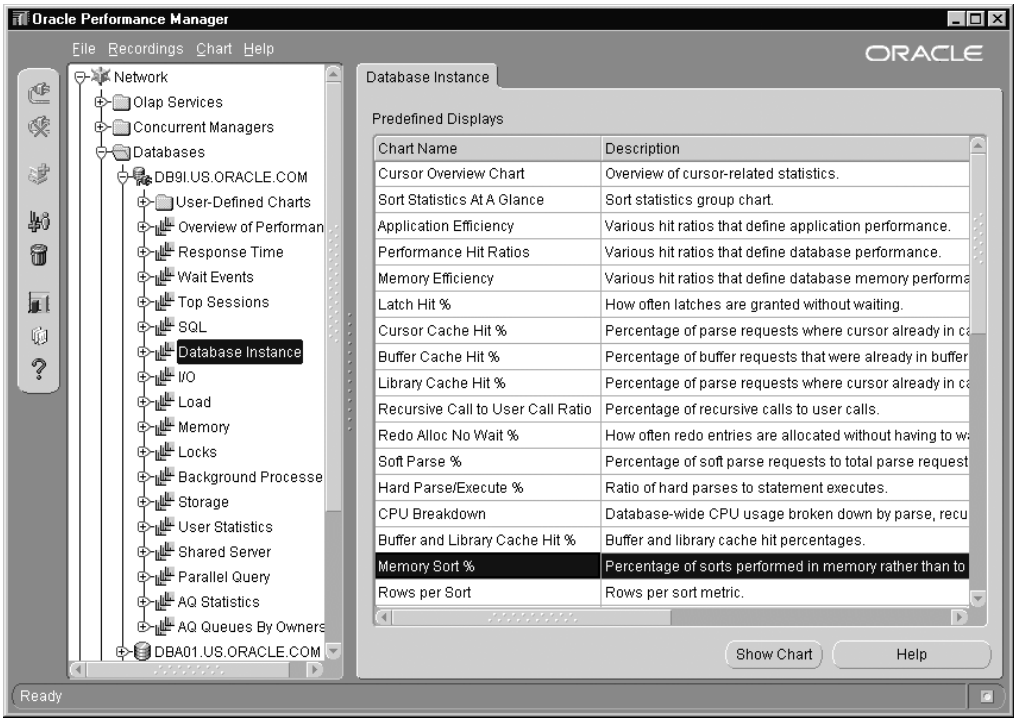
In a DSS system this ratio is not much use as due to the nature of DSS reports disk sorts will be unavoidable. Rather for a DSS system focus on eliminating unnecessary sorts.

Performance Trade-Offs for Large Sort Areas

Increasing the size of the sort area causes each server process that sorts to allocate more memory. It may affect operating system memory allocation and induce paging and swapping.

If you increase sort area size, consider decreasing the retained size of the sort area, or the size to which the server reduces the sort area if its data is not expected to be referenced soon. A smaller retained sort area reduces memory usage but causes additional I/O to write and read data to and from temporary segments on disk.

Performance Manager: Sorts



Performance Manager: Sorts

These charts show information regarding sorts. These include the maximum amount of memory used for sorts, and average number of rows per sort and the percentage of sorts performed in memory to disk.

Monitoring Temporary Tablespaces

```
SQL> select tablespace_name, current_users, total_extents,
  2           used_extents, extent_hits, max_used_blocks,
  3           max_sort_blocks
  4   from v$sort_segment;
TABLESPACE_NAME CURRENT_USERS TOTAL_EXTENTS USED_EXTENTS
EXTENT_HITS MAX_USED_BLOCKS MAX_SORT_BLOCKS
--------------- ------------- ------------- ------------
----------- --------------- ---------------
TEMP                        2             4            3
20             200              200
```

- **Default storage parameters apply to sort segments.**
- **Sort segments have unlimited extents.**

The V$SORT_SEGMENT View

This view contains information about every sort segment of the temporary tablespaces in the instance.

Column	Description
CURRENT_USERS	Number of active users
TOTAL_EXTENTS	Total number of extents
USED_EXTENTS	Extents currently allocated to sorts
EXTENT_HITS	Number of times an unused extent was found in the pool
MAX_USED_BLOCKS	Maximum number of used blocks
MAX_SORT_BLOCKS	Maximum number of blocks used by an individual sort

Temporary Tablespace Configuration

Default storage parameters for the temporary tablespace apply to sort segments, except that they have unlimited extents (whatever value `MAXEXTENTS` is set to).

Temporary Tablespace Configuration

- **Set appropriate storage values.**
- **Set up different temporary tablespaces based on sorting needs.**

```
SQL> SELECT session_num, tablespace, extents, blocks
  2   FROM v$sort_usage;
SESSION_NUM  TABLESPACE             EXTENTS    BLOCKS
-----------  --------------------- -------- --------
         16  TEMP                         4      200
```

- **Stripe temporary tablespaces.**
- **Use `v$tempfile` and `dba_temp_files` for information on temporary files.**

Guidelines

Storage Parameters

Because sorts are performed in memory if they are smaller than the area of memory allotted for sorting you should consider this value when setting the extent size of the temporary tablespace:

- Select `INITIAL` and `NEXT` values as integer multiples of `SORT_AREA_SIZE`, allowing an extra block for the segment header.
- If you are using `PGA_AGGREGATE_TARGET`, then use the `AUTOALLOCATE` option for the tablespace.
- Set `PCTINCREASE` to 0.

Note: If using locally managed tablespaces `PCTINCREASE` cannot be set, and if the tablespace is `AUTOALLOCATE` then there is no option for either `INITIAL` or `NEXT`.

Different Temporary Tablespaces

To define the sort space needed by the users, and to obtain information on the currently active disk sorts in the instance:

```
SQL> SELECT username, tablespace, contents, extents, blocks
  2> FROM v$sort_usage;
```

Guidelines (continued)

Different Temporary Tablespaces (continued)

```
USERNAME     TABLESPACE    CONTENTS           EXTENTS       BLOCKS
----------------------------------------------------------------------
HR           TEMP          TEMPORARY               20         1000
OE           TEMP          TEMPORARY                2          100
```

The user performing the sort is the username from the V$SESSION view. The user stat requiring a large amount of disk space should be assigned another temporary tablespace with larger extent size.

Striping

The temporary tablespace should be striped over many disks. If the temporary tablespace is striped over only two disks with a maximum of 50 I/Os per second each, then you can only do 100 I/Os per second. This restriction may become a problem, making sort operations take a very long time. You can speed up sorts fivefold by striping the temporary tablespace over ten disks, thus allowing 500 I/Os per second.

Data Files

You also use different views for viewing information about temporary files than you would for data files. The V$TEMPFILE and DBA_TEMP_FILES views are analogous to the V$DATAFILE and DBA_DATA_FILES views.

Summary

In this lesson, you should have learned how to:

- Describe how sorts are performed
- Identify the SQL operations that require sorts
- List ways to reduce total sorts and disk sorts
- Determine the number of sorts performed in memory
- Set old and new sort parameters
- Differentiate between disk and memory sorts

Practice 7

Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as system/manager and query the `V$SYSSTAT` view, and record the value for sorts (memory) and sorts (disk). If the ratio of Disk to Memory sorts is greater than 5% then increase the sort area available.
   Note: The statistics collected from `v$sysstat` are collected from startup. If you need to get accurate statistics per statement, you must record statistics from before the statement has run and again afterwards. Subtracting to two values will give the statistics for the statement.
2. Connect as user sh/sh. In order to ensure that some sorts go to disk run the command "alter session set sort_area_size = 512;". Then execute the SQL script (`$HOME/STUDENT/LABS/lab07_02.sql`) that will force sorts to disk.
   Note: If this script fails due to a lack of free space in the `TEMP` tablespace. Resize the temporary tablespace.
3. Connect as system/manger and query the columns `TABLESPACE_NAME`, `CURRENT_USERS, USED_EXTENTS` and `FREE_EXTENTS` from the `V$SORT_SEGMENT` view. The columns `USED_EXTENTS`, and `FREE_EXTENTS` are useful in monitoring the usage of the `TEMPORARY` tablespace.
   Note: If this statement returns no rows, it means that all sort operations since startup have completed in memory.
4. To decrease the sorts number of sorts going to a temporary tablespace, increase the value of the parameter `SORT_AREA_SIZE` to 512000 using the "alter session" command.
5. Connect as system/manager and configure the new parameters for `PGA` memory allocation using the "alter system" command. Use the values `AUTO` for `WORKAREA_SIZE_POLICY` and 10M for `PGA_AGGREGATE_TARGET`)

8

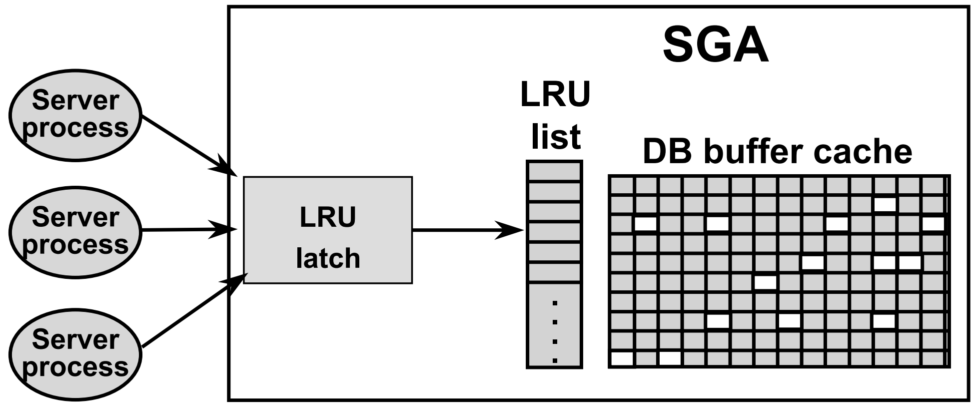
Diagnosing Contention for Latches

Objectives

After completing this lesson, you should be able to:

- Describe the purpose of latches
- Describe the different types of latch requests
- Diagnose contention for latches
- Identify the resources to be tuned to minimize latch contention

Latches: Overview



What Are Latches

Latches are simple, low-level serialization mechanisms to protect shared data structures in the system global area (SGA). For example, latches protect the list of users currently accessing the database, and protect the data structures describing the blocks in the buffer cache. A server or background process must acquire the accompanying latch to start manipulating or looking at a shared data structure and must release the accompanying latch when finished. The implementation of latches is operating system and platform dependent, particularly in regard to whether and how long a process will wait for a latch.

Tuning Latches

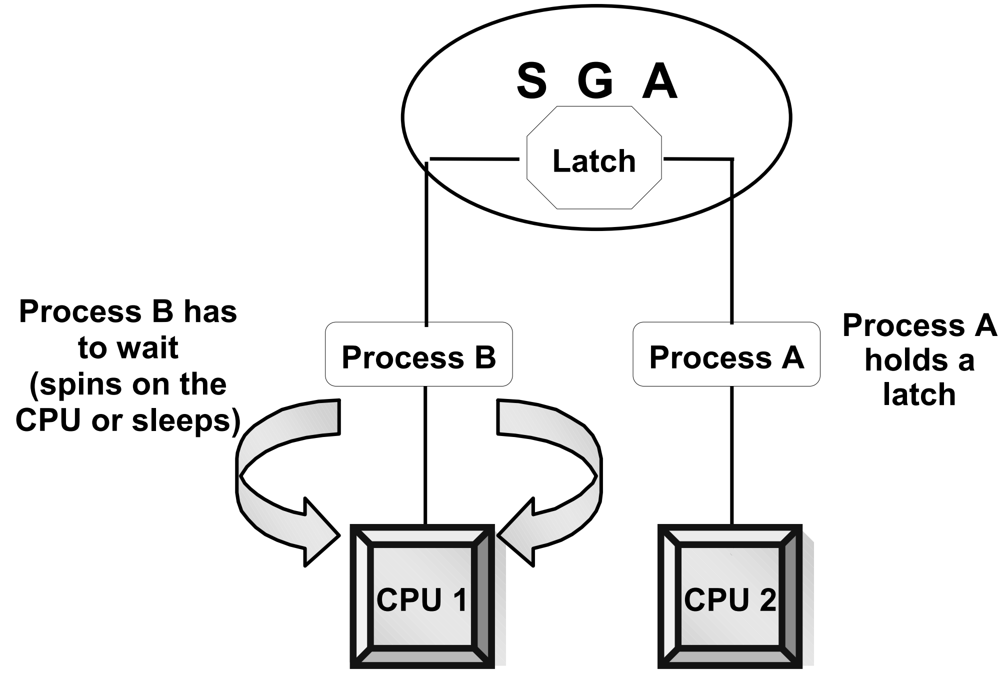
You *do not* tune latches. If you see latch contention, it is a symptom of a part of SGA experiencing abnormal resource usage. Latches control access with certain assumptions, for example, a cursor is parsed once and executed many times. To fix the problem, examine the resource usage for the parts of SGA experiencing contention. Merely looking at V$LATCH does not address the problem.

In summary, latches are light weight locks protecting internal data structures. In Oracle9*i*, latches are inaccessible to users, yet latch contention statistics can serve as a diagnostic tool to identify what resources should be tuned in order to optimize performance.

Purpose of Latches

- To serialize access:
  - Protect data structures within the SGA
  - Protect shared memory allocations
- To serialize execution:
  - Prevent simultaneous execution of certain *critical* pieces of code
  - Prevent corruptions

Waiting for a Latch



8-5

Waiting for a Latch

Although the exact implementation is operating system and platform specific, latches are normally implemented as a single memory location that has a value of zero if the latch is free, or nonzero if it is already acquired.

On single-CPU systems, if a required latch is already held by another process, the process requesting the latch will release the CPU, incurring a costly context switch, and go to sleep for a brief period before trying again. A sleep corresponds to a wait for the latch free wait event.

On a multi-CPU system it is possible that the process holding the latch is running on another CPU and so might potentially release the latch in the next few instructions. Therefore, on multi-CPU systems the requesting process holds the CPU and spins (counts up to a specific number), then tries to acquire the latch again; and if still not available, spins again. The number of spins and the spin time is operating system and platform specific. If after the number of spins the latch is still not available, the process releases the CPU and goes to sleep, like in the single-CPU case. However, since latches are normally held for brief periods of time (order of microseconds), a successful spin in the multi-CPU case avoids a context switch at the expense of holding on to the CPU.

Latch Request Types

Latches are requested in one of two modes:

- **Willing-to-wait**
  - **The requesting process waits a short time and requests the latch again.**
  - **The process continues waiting and requesting until the latch is available.**
- **Immediate**
  - **The requesting process does not wait, but continues processing other instructions.**

The difference is in the way the processes advance when the requested latch is not available.

ORACLE

8-6

Latch Requests

Processes request latches in one of two modes: willing-to-wait or immediate. The essential difference between the two request types is the way the processes proceed when the requested latch is not available.

- Willing-to-wait: If the latch requested with a willing-to-wait request is not available, the requesting process waits a short time and requests the latch again. The process continues waiting and requesting until the latch is available. This is the usual way of processing.
- Immediate: If the latch requested with an immediate request is not available, the requesting process does not wait, but continues processing other instructions. For example, when the PMON process attempts to clean up an abnormally terminated process, it finds that the latch required to access the structure is not available. PMON continues with subsequent instructions rather than waiting for the latch to be freed.

Latch Contention

- **Check if the latch-free wait event is a main wait event.**
- **`V$LATCH` view contains statistics on:**
  - **Columns for the willing-to-wait type requests, such as `GETS`, `MISSES`, `SLEEPS`, `WAIT_TIME`, `CWAIT_TIME`,and `SPIN_GETS`**
  - **Columns for the immediate type requests, such as `IMMEDIATE_GETS`, `IMMEDIATE_MISSES`**
- **You can use STATSPACK report:**
  - **Wait events; check for latch-free**
  - **Latch activity section**

8-7

Latch Contention

If multiple processes are seeking the same latch, any process can acquire the latch, depending on the time of its reseek and the release of the latch. However, latches are held by atomic instructions and hence are obtained and released very quickly. The goal is to minimize the contention for latches among processes. Like all resources, available latches are finite.

Diagnostics

As a first step, check if the wait event `latch free` has a high wait time. If it does, review statistics from the `V$LATCH` view. The following columns in the `V$LATCH` view reflect willing-to-wait requests:

- `gets`: Number of successful willing-to-wait requests for a latch
- `misses`: Number of times an initial willing-to-wait request was unsuccessful
- `sleeps`: Number of times a process waited after an initial willing-to-wait request
- `wait_time`: Number of milliseconds waited after willing-to-wait request
- `cwait_time`: A measure of the cumulative wait time including the time spent spinning and sleeping, the overhead of context switches due to OS time slicing and page faults and interrupts
- `spin_gets`: Gets that missed first try but succeeded after spinning

Diagnostics (continued)

The following columns in the V$LATCH view reflect immediate requests:

- `immediate_gets`: Number of successful immediate requests for each latch.
- `immediate_misses`: Number of unsuccessful immediate requests for each latch.

STATSPACK Report

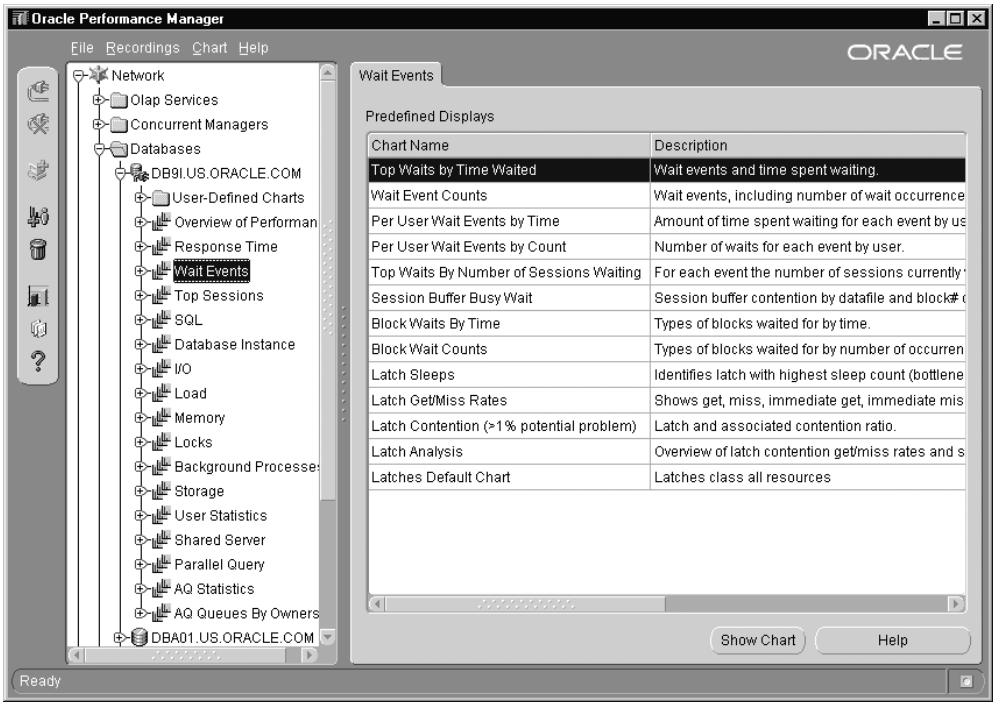
The report produced by STATSPACK contains some sections meant for diagnosing latch contention. First however, you should look at the `Top 5 Wait Events` section. If latch free event is one of them it may be worthwhile to diagnose further to find which latches are involved in contention. Following are sections from a STATSPACK report.

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~
Events                   Waits        Wait Time    % Total
                                      (cs)         Wt Time
---------------------------           ---------    --------- --------
enqueue                    482,210   1,333,260    36.53
latch free               1,000,676     985,646    27.01
buffer busy waits          736,524     745,857    20.44
log file sync              849,791     418,009    11.45
log file parallel write    533,563     132,524     3.63
        -----------------------------------


Latch Activity for DB: ED31  Instance: ed31  Snaps: 1 -2
->"Get Requests", "Pct Get Miss" and "Avg Slps/Miss" are statistics for
  willing-to-wait latch get requests
->"NoWait Requests", "Pct NoWait Miss" are for no-wait latch get requests
->"Pct Misses" for both should be very close to 0.0
-> ordered by Wait Time desc, Avg Slps/Miss, Pct NoWait Miss desc

                                         Pct    Avg           Pct
                             Get         Get   Slps   NoWait NoWait
Latch                      Requests      Miss  /Miss  Requests  Miss
------------------------           ------------ -----  ------ -------
    --------
…
cache buffers chains    142,028,625 0.3     0.4  1,193,834 0.7
cache buffers lru chain   8,379,760 0.1     0.7    414,979 0.1
…
library cache            36,870,207 2.1     0.7          0 0.4
…
Redo allocation          10,478,825 0.9     0.3          0
    …
Row cache objects        27,905,682 0.3     0.1          0
…
```

Performance Manager: Latches



8-9

Performance Manager: Latches

In the Wait Events set of charts you will find the information regarding latches. The charts show information regarding latch contention and latch analysis, which helps to determine where a wait is occurring. This will aid the DBA in resolving the bottleneck.

Reducing Contention for Latches

- Generally, the DBA should not attempt to tune latches. However, the following steps are useful:
  - Investigate further, depending on the latch that is in contention.
  - Consider tuning the application if the contention is mainly for shared pool and library cache.
- If further investigation suggests it, size the shared pool and buffer cache appropriately.

ORACLE

8-10

Reducing Contention

In Oracle9*i*, you should not be attempting to tune the number of latches. Oracle9*i* automatically calculates the number of latches required, based on the environment defined by the initialization parameters and the operating system level parameters.

Contention for latches is a symptom of a performance problem. The type of problem is indicated by the latch contended for.

Frequently, the most effective way to reduce latch contention is to modify the application behavior. If you observe, based on other investigation of cache hit ratios, and so on, that the SGA is inappropriately configured, you may consider changing buffer cache or shared pool sizes.

Important Latches for the DBA

- `shared pool`: **Protects memory allocations in the shared pool**
- `library cache`: **Used to locate matching SQL in the shared pool**
- `cache buffers LRU chain`: **Protects the LRU list of cache buffers**
- `cache buffers chains`: **Needed when searching for data blocks cached in the SGA**
- `redo allocation`: **Manages the allocation of redo space in the log buffer for redo entries**
- `redo copy`: **Used to write redo records into the redo log buffer**

Significant Latches

Contention for the following latches may be significant in your database:

- `shared pool latch`, and `library cache latch`:
  Contention for these latches indicates that SQL, PL/SQL statements are not being reused, possibly because the statements are not using bind variables, or the cursor cache is insufficient. To alleviate the problem, consider:
  - Tuning the shared pool,
  - Tuning the SQL statements
  - Check for "hot blocks"
  - Setting `CURSOR_SHARING`
  - Configure a Large Pool

  Using Oracle Shared Server without a large pool may lead to contention on the Shared Pool Latch. This can be resolved by creating the large pool, or by reverting to dedicated connections.
- `cache buffers lru chain latch`:
  This latch is required when dirty blocks are written to the disk or when a server process is searching for blocks to write to. Contention for this latch indicates excessive buffer cache throughput, such as many cache-based sorts, inefficient SQL that accesses incorrect indexes iteratively (large index range scans), or many full table scans.

Significant Latches (continued)

It is also possible that the database writer is unable to keep pace with the rate of changes to data blocks, forcing the foreground process to wait longer holding the latch while looking for a free buffer. To overcome the problem, consider tuning the buffer cache or the database writer operation.

- `cache buffers chains latch`:
  This latch is needed when user processes try to locate a data block in the buffer cache. The contention for this latch indicates some specific blocks (hot blocks) are accessed repeatedly.
- `Redo Allocation Latch`
  This latch synchronizes the allocation of space in the redo buffer, in irder to prevent two servers writing to the same memory space.
- `Redo Copy Latch`
  Allows server processes to write into the buffer, at the memory location given by the allocation latch.

The following Oracle8*i* `init.ora` parameters related to latches are now obsolete:

`DB_BLOCK_LRU_LATCHES` specifies the maximum number of LRU latch sets, i.e., cache buffers LRU chain and cache buffer chains. The number of latches is the total for all Buffer Cache, Keep Cache and Recycle Cache latches.The buffers of a buffer pool are equally divided among the working LRU latch sets of the buffer pool so that each buffer is protected by one LRU latch. Normally, the more latches you specify, the less contention exists for those latches. However, too many latches may result in small LRU lists, potentially reducing the cache life of a database block.

The maximum of (`CPU_COUNT x 2 x 3`) ensures that the number of latches does not exceed twice the product of the number of CPUs and the number of buffer pools. Typically you should set this parameter to the number of CPUs or a multiple of that number. Each working set is handled entirely by one database writer (DBW*n*) process. Therefore, if multiple DBW*n* processes are running, the number of LRU latches should be greater than or equal to the number of DBW*n* processes. To balance the load evenly between the DBW*n* processes, the number of LRU latches in each buffer pool should be a multiple of the number of DBW*n* processes.

If you do not set this parameter, Oracle Server uses the value `CPU_COUNT/2`. This value is usually adequate. Increase this value only if misses are higher than 3%, as calculated from values in `V$LATCH`. When you increase the value, Oracle Server decides whether to use this value or reduce it based on a number of internal checks.

Shared Pool and Library Cache Latches

Contention for shared pool latch and library cache latch indicates one or more of the following:

- **Unshared SQL**
- **Reparsed sharable SQL**
- **Insufficiently sized library cache**

Shared Pool and Library Cache Latch Contention:

A main cause of shared pool or library cache latch contention is unnecessary parsing. The methods to overcome this problem are also discussed in the "Sizing the Shared Pool" lesson. There are a number of techniques that can be used to identify unnecessary parsing and a number of types of unnecessary parsing:

- Unshared SQL: Identify similar SQL statements that could be shared if literals were replaced with bind variables. You could inspect SQL statements that have only one execution to see if they are similar. If you specify the sort by uppercase, it is easier to notice similar SQL statements.

```
SELECT sql_text
FROM V$SQLAREA
WHERE executions = 1
ORDER BY UPPER(sql_text);
```

- Reparsed Sharable SQL: Check the V$SQLAREA view to find if there are avoidable reparsing.

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS
FROM V$SQLAREA
ORDER BY PARSE_CALLS;.
```

Summary

In this lesson, you should have learned how to:

- Describe latches and their usage
- Diagnose contention for latches
- Identify the resources to be tuned to minimize contention for latches

9

Tuning Undo Segments

Objectives

After completing this lesson, you should be able to do the following:

- Describe the concept of automatic undo management
- Create and maintain the automatic managed undo tablespace
- Use the dynamic performance views to check rollback segment performance
- Reconfigure and monitor rollback segments
- Define the number and sizes of rollback segments
- Appropriately allocate rollback segments to transactions

ORACLE

9-2

Objectives

This lesson helps you understand the automatic undo management feature. You will also learn to configure automatic undo management in an Oracle9*i* database.

Good rollback segment configuration is crucial to a well-tuned Oracle database. This lesson helps you to recognize and solve problems arising from inappropriate numbers or sizes of rollback segments.

Automatic Undo Management in Oracle9*i*

- **The automatic undo management feature simplifies the management of undo segments.**
- **Set the `UNDO_MANAGEMENT` parameter to:**
  - **`AUTO` for automatic undo management**
  - **`MANUAL` for managing rollback segments manually**
- **The `UNDO_RETENTION` parameter specifies the time (in seconds) to retain undo information.**

Automatic Managed Undo

The automatic undo management (AUM) feature, manages undo space in Oracle databases.

You have the choice of using manually managed rollback segments or automatic undo management. Set the `UNDO_MANAGEMENT` initialization parameter to:

- `AUTO` to enable the instance to manage rollback segments automatically (AUM)
- `MANUAL` to create and manage rollback segments manually (RBU)

When the database is set to use auto-managed undo, you need to perform few explicit actions for management of the undo space. The number, and size, of the undo segments is maintained by Oracle9*i* server with no management required from the DBA. When the first DML operation is executed within a transaction, the transaction is assigned to an undo segment in the current undo tablespace.

The DBA must create, and size, the undo tablespace. You can specify the amount of undo information retained in the auto-managed undo segments by using the `UNDO_RETENTION` parameter. The size of the tablespace should be large enough to accommodate the amount of undo generated during the time specified by `UNDO_RETENTION`

Tablespace for Automatic Undo Management

- **Create a tablespace for automatic undo management in one of the following ways:**
  - **Using the `UNDO TABLESPACE` clause in the `CREATE DATABASE` command**
  - **By using the `CREATE UNDO TABLESPACE` command**
- **For UNDO tablespaces the values of `MINIMUM EXTENT` and `DEFAULT STORAGE` are system generated.**
- **Restrictions:**
  - **You cannot create database objects in this tablespace.**
  - **You can specify data file and the `extent_management` clause only.**

Tablespace for Automatic Undo Management

Set the UNDO_MANAGEMENT initialization parameter to AUTO. You can create an auto-managed undo tablespace when creating the database. You can use the UNDO TABLESPACE clause to specify the name, data file, size, and block size of the undo tablespace. If you do not specify the UNDO TABLESPACE clause when creating the database, then:

- An UNDO tablespace with the name SYS_UNDOTBS is created.
- On a UNIX system the data file with name DBU1<ORACLE.SID>.dbf is placed in the $ORACLE_HOME/dbs folder.
- AUTOEXTEND is set to ON.

You can also create an undo tablespace with the CREATE UNDO TABLESPACE command.

You can provide the name of the undo tablespace in UNDO_TABLESPACE initialization parameter.

Altering an Undo Tablespace

- The `ALTER TABLESPACE` command can be used to make changes to undo tablespaces.
- The following example adds another data file to the undo tablespace:

```
ALTER TABLESPACE undotbs1
ADD DATAFILE '/u02/oradata/testdb/undotbs1_02.dbf'
AUTOEXTEND ON;
```

- You cannot take an undo segment in the active undo tablespace offline.

9-5

Altering an Undo Tablespace

The following clauses are supported when altering an undo tablespace:

- ADD DATAFILE
- RENAME
- DATAFILE [ONLINE|OFFLINE]
- BEGIN BACKUP
- ENDBACKUP

This example depicts the addition of a data file to an existing undo tablespace:

```
ALTER TABLESPACE undotbs1
ADD DATAFILE '/u02/oradata/testdb/undotbs1_02.dbf'
AUTOEXTEND ON;
```

Switching Undo Tablespaces

- **A DBA can switch from using one undo tablespace to another.**
- **Only one undo tablespace per instance can be assigned as active.**
- **Switching is performed by using the `ALTER SYSTEM` command:**

```
ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2;
```

9-6

ORACLE

Number of Active Undo Tablespaces

At any given moment of time, there can be only one active undo tablespace. However, an instance may have more than one undo tablespace in use per instance. If an instance has two undo tablespaces (`UNDOTBS1` and `UNDOTBS2`), only one can be active (in this example: `UNDOTBS1`). This means that all new transactions must use this tablespace to store any undo data.

If the DBA switches the undo tablespace using the `ALTER SYSTEM SET UNDO_TABLESPACE=UNDOTBS2` command, all new transactions are directed to the undo tablespace, `UNDOTBS2`; however, all current transactions (that is, those already assigned to `UNDOTBS1`), will continue to use the undo tablespace `UNDOTBS1`, until they are completed.

Dropping an Undo Tablespace

The DROP TABLESPACE command can be used to drop an undo tablespace:

```
DROP TABLESPACE UNDOTBS_2;
```

- **An undo tablespace can be dropped only if it is not the active undo tablespace.**
- **Queries that require a read consistent image of undo data that is stored in an dropped undo tablespace will return an error.**

Dropping an Undo Tablespace

An undo tablespace can be dropped only if:

- It is not currently used by any instance, and
- Its transaction tables do not contain any uncommitted transactions.

The `DROP TABLESPACE` *undo tablespace name* command behaves the same as `DROP TABLESPACE` *tablespace name* `INCLUDING CONTENTS.`

Setting UNDO_RETENTION

UNDO_RETENTION parameter is:

- **Specified in time (seconds)**
- **A target value. If space is required committed data will be overwritten**
- **Controls the amount of undo data to retain after committing**

Setting UNDO_RETENTION

UNDO_RETENTION controls the amount of committed undo information to retain. You can use UNDO_RETENTION to satisfy queries that require old undo information in order to roll back changes to produce older images of data blocks. You can set the value at instance startup. The value should be large enough to cover any long running query that would require a read consistent image of data that was changed since starting the query.

The UNDO_RETENTION parameter works best if the current undo tablespace has enough space for all of the transactions during an UNDO_RETENTION period. If an active transaction needs undo space and the undo tablespace does not have any free space, the database starts reusing undo space that would have been retained, due to UNDO_RETENTION. This may cause long queries to fail due to SNAPSHOT TOO OLD errors. Be sure to allocate enough space in the undo tablespace to satisfy the space requirement for the current setting of the UNDO_RETENTION parameter.

The UNDO_RETENTION parameter value can also be changed dynamically using the ALTER SYSTEM command. The effect of the UNDO_RETENTION parameter is immediate, but it can be honored only if the current undo tablespace has enough space for the active transactions. If an active transaction requires undo space and the undo tablespace does not have available space, the database starts reusing unexpired undo space. Such action can potentially cause some queries to fail with the *snapshot too old* error.

Setting UNDO_RETENTION (continued)

`UNDO_RETENTION` is specified in units of seconds, with default value of 900 seconds. Because undo segments are on disk, they can survive system crashes.

Space Requirement for Undo Tablespace

You can use the following query to set the `UNDO_RETENTION` parameter and size the undo tablespace:

```
SELECT (RD * (UPS * OVERHEAD) + OVERHEAD) AS "Bytes“
FROM   (SELECT value AS RD FROM v$parameter
                WHERE  name = 'undo_retention'),
       (SELECT (SUM(undoblks) / SUM( ((end_time - begin_time)
* 86400)))
                AS UPS  FROM   v$undostat),
       (SELECT value AS Overhead FROM v$parameter
                WHERE  name = 'db_block_size');
```

Other Parameters for Automatic Undo Management

- `UNDO_MANAGEMENT`: **Specifies whether the database uses `AUTO` or `MANUAL` mode**
- `UNDO_TABLESPACE`: **Specifies a particular undo tablespace to be used**
- `UNDO_SUPPRESS_ERRORS`: **Set to `TRUE`, this parameter suppresses errors while attempting to execute manual operations, such as `ALTER ROLLBACK SEGMENT ONLINE`, while in auto mode.**

ORACLE

9-10

Other Parameters for System Managed Undo

The following parameters are used with the System Managed Undo feature:

- `UNDO_MANAGEMENT`: Specifies what mode of undo management to use. The parameter can be reassigned when the database is open.
  - If set to `AUTO`, the system managed undo feature is used. Make sure that you have already created an undo tablespace.
  - A value of `MANUAL` means that the rollback segments are managed by the DBA.
- `UNDO_TABLESPACE`: Specifies the name of the undo tablespace
  - If the database is in SMU mode and the `UNDO_TABLESPACE` parameter is omitted at startup, the first available undo tablespace in the database is chosen.
  - If no undo tablespace is available, the instance starts without an undo tablespace using the `SYSTEM` rollback segment. Make sure that an undo tablespace is available immediately thereafter.
  - To replace one active undo tablespace with another, you can use the `ALTER SYSTEM SET UNDO_TABLESPACE …` command.

Parameters for Automatic Managed Undo (continued)

- `UNDO_SUPPRESS_ERRORS`: Primarily meant for applications and tools that use statements such as `SET TRANSACTION USE ROLLBACK SEGMENT`. Setting this parameter enables users to use the SMU feature before all application programs and scripts are converted to SMU mode. So at the beginning of such sessions, you can add the `ALTER SESSION SET UNDO_SUPPRESS_ERRORS=TRUE` statement to suppress the (OER 30019) error.

Space Requirement for Undo Tablespace

Given a specific `UNDO_RETENTION` parameter setting and some system statistics, the amount of undo space required to satisfy the undo retention requirement can be estimated using the formula:

```
Undo Space = (UNDO_RETENTION * (Undo Blocks Per Second *
                 DB_BLOCK_SIZE) ) + DB_BLOCK_SIZE
```

Monitoring Automatic Undo Management

- **Use `V$UNDOSTAT` view to monitor undo segments.**
- **This view is available in both manual and auto mode.**
- **The `UndoBlks` column displays the number of undo blocks allocated.**

Monitoring Automatic Managed Undo

Use the `V$UNDOSTAT` view to monitor space allocation and usage for automatically managed undo. Each row in the view keeps statistics collected in the instance for a 10-minute interval.

You can use this view to estimate the amount of undo space required for the current workload. This view is available in both SMU and RBU modes.

Using V$UNDOSTAT

```
SQL> select begin_time, end_time, undoblks,
  2* txncount, maxquerylen
  3* from v$undostat;

BEGIN_TIME      END_TIME          UNDOBLKS   TXNCOUNT
--------------- --------------- ---------- ----------
25-oct-01:06:04 25-oct-01:06:14        234         12
25-oct-01:05:44 25-oct-01:05:54        587         21
25-oct-01:05:34 25-oct-01:05:44      1,187         45
25-oct-01:05:24 25-oct-01:05:34        346         15
25-oct-01:05:14 25-oct-01:05:24        642         23
……
```

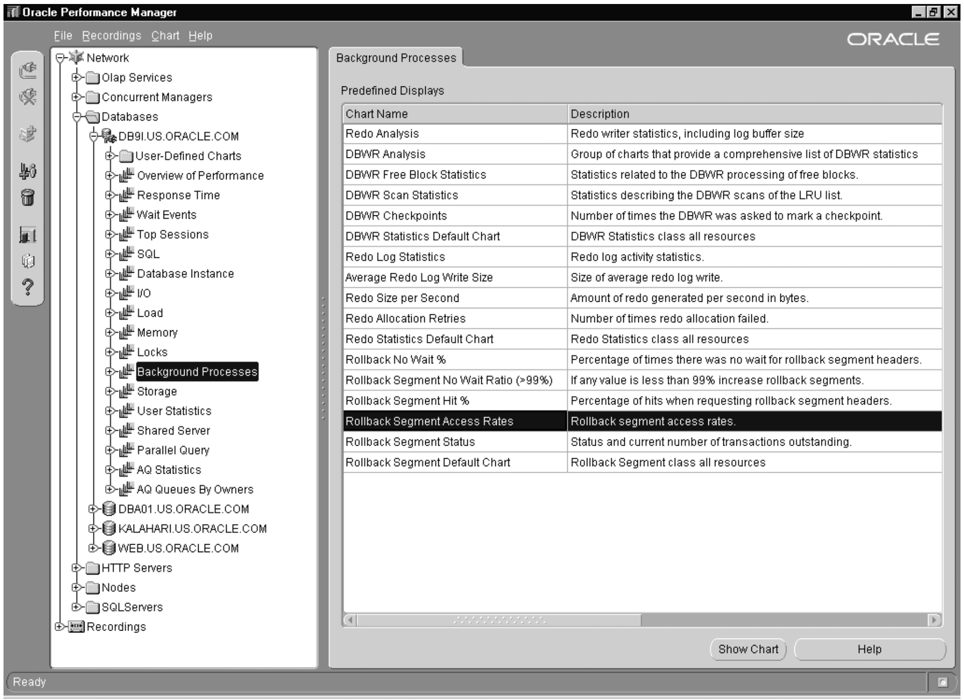

Using V$UNDOSTAT

The example above shows that the peak undo consumption occurred between 05:34 and 15:44; 1,187 undo blocks were consumed in 10 minutes (or about 2 blocks per second). Also, the highest transaction concurrency occurred during that same period, with 45 transactions executing at the same time.

Two aspects can be tuned under automatic undo management:

- The size of the undo tablespace
- The amount of time that undo blocks are retained before being overwritten.

Performance Manager: Rollback/Undo

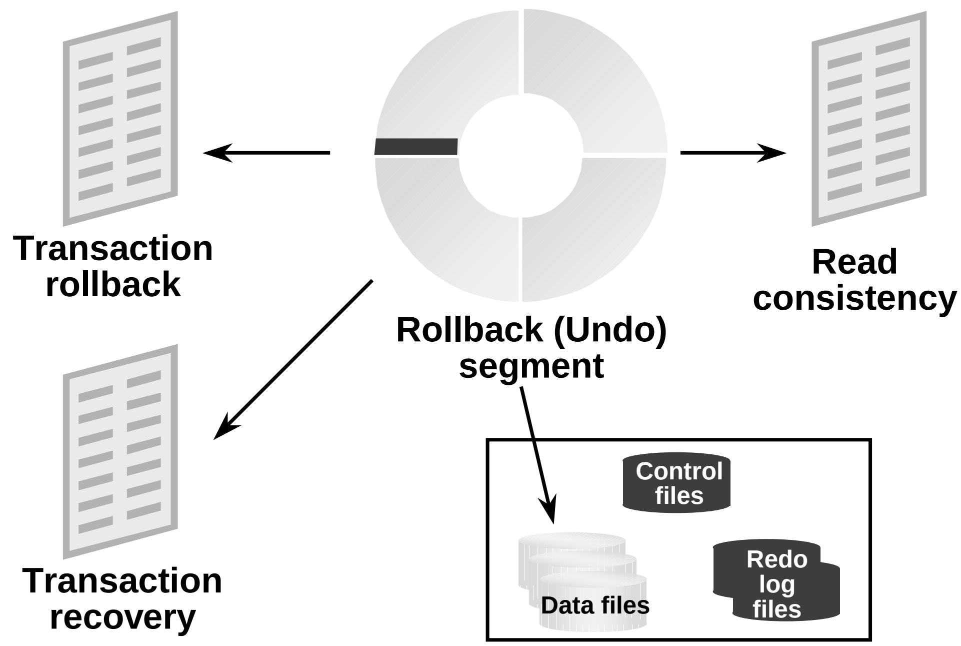


Performance Manager: Rollback/Undo

The Background Processes set of charts will give the DBA information on Rollback, or UNDO, Segments.

If using UNDO segments, you may need to resize the tablespace using the Enterprise Manager Console. The Tablespace Manager will not show information regarding UNDO tablespaces.

Rollback Segments: Usage



Transaction Rollback

When a transaction makes changes to a row in a table, the old image is saved in a rollback segment, also called an *undo segment*. If the transaction is rolled back, the value in the rollback segment is written back to the row, restoring the original value.

Transaction Recovery

If the instance fails when transactions are in progress, the Oracle server rolls the uncommitted changes back when the database is opened again. When Transaction Recovery completes all modified data that was not committed before the instance failure will have been rolled back.

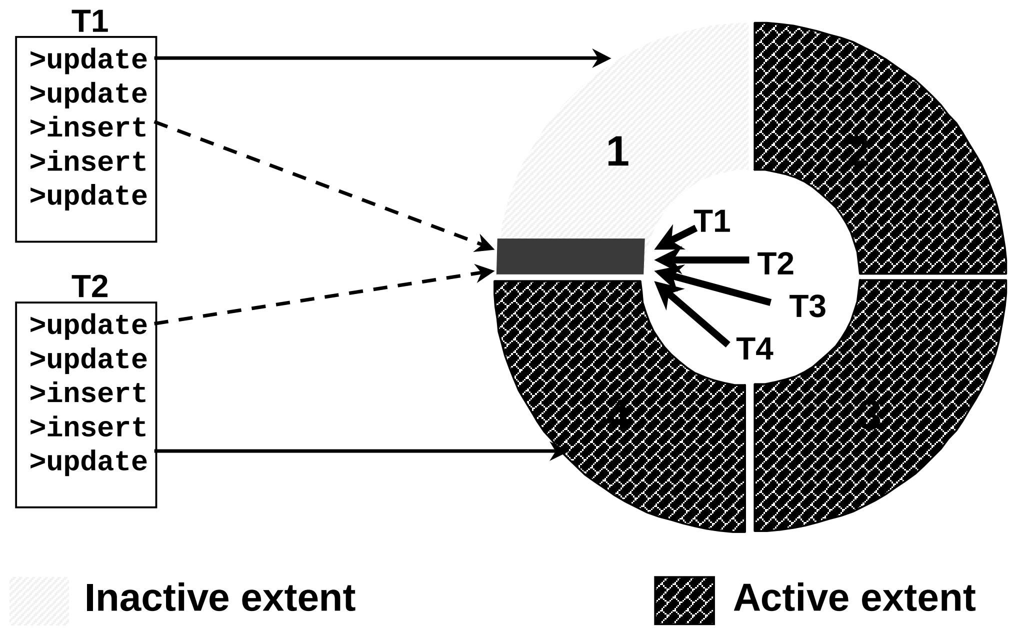
Read Consistency

Read consistency consists of the following conditions:

- When a user's transactions are in progress, other sessions in the database should not see any uncommitted changes.
- A query statement should not see any changes that are made, committed or uncommitted, after the statement commenced execution.
- DML statements will not see any uncommitted changes. However, the changed rows are locked by the transaction making the changes, and the lock will only be released when the transaction ends.

The old values in the rollback segments, also referred to as undo information, are used to provide the read-consistent image.

Rollback Segment Activity



Active and Inactive Extents

Transactions use extents of a rollback segment in an ordered, circular fashion, moving from one extent to the next after the current extent is full. A transaction writes a record to the current location in the rollback segment and advances the current pointer by the size of the record.

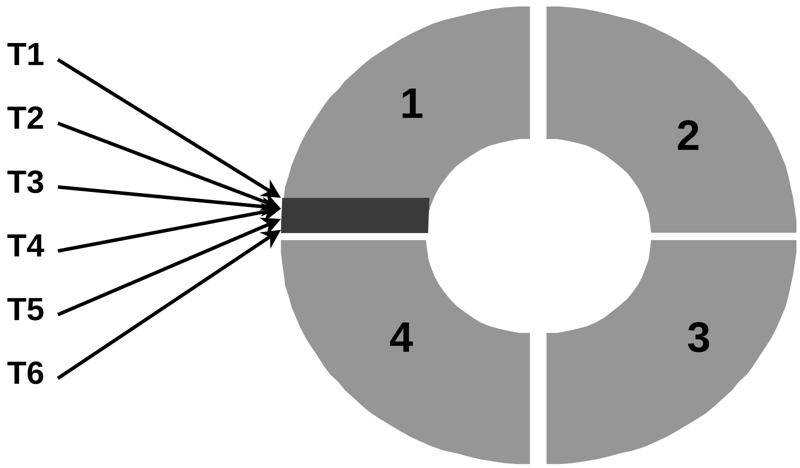
Note: More than one transaction can write to the same extent of a rollback segment. Each rollback segment block contains information from only one active transaction.

Rollback Segment Activity

Writing to rollback segments requires that the corresponding undo data is available in the database buffer cache. To maintain large amounts of undo information, the buffer cache should be quite large, or there is a higher number of physical I/Os.

Rollback Segment Header Activity

- **Rollback segment headers contain entries for their respective transactions.**
- **Every transaction must have update access.**



Rollback Segment Header Activity

The Oracle server keeps a transaction table in the header of every rollback segment.

The rollback segment header activity controls the writing of changed data blocks to the rollback segments. Because the rollback segment header is a data block and it is frequently modified, the rollback segment header block remains in the data block buffer cache for long periods of time. Therefore, accesses to the rollback segment header block increase the hit ratio for the application, even though it is not related to the data blocks.

The Impact of Rollback Segment Header Activity

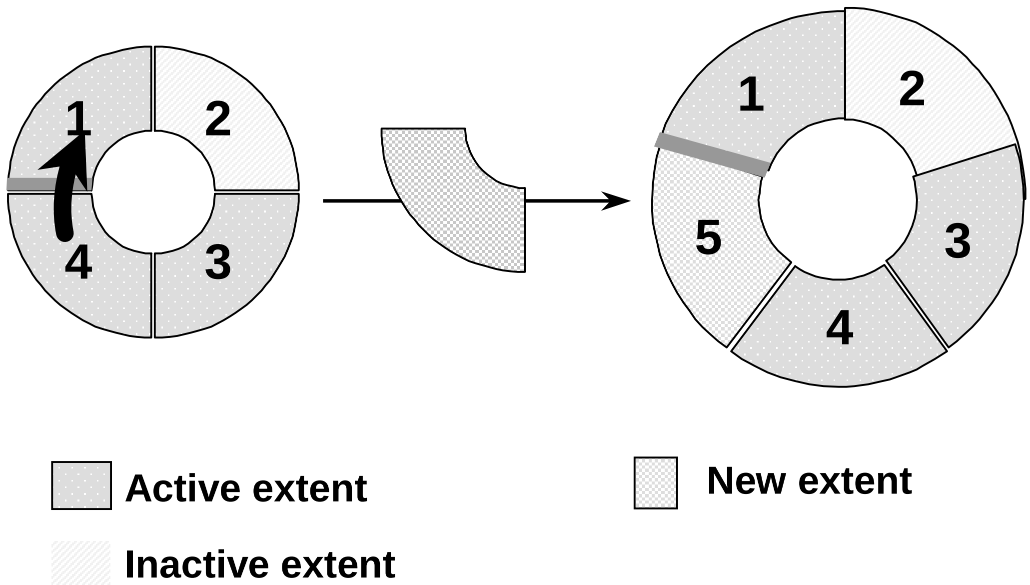
The impact of the rollback segment header activity on the cache hit ratio is important for OLTP systems that feature many small transactions.

Every transaction must have update access to the transaction table for its rollback segment. You need enough rollback segments to prevent transactions from contending for the transaction table.

If you underestimate the number of rollback segments needed, performance is degraded and transactions may generate errors. If you overestimate, you use unnecessary space.

When using the automatic undo management feature, Oracle server automatically manages the number of undo segments, thus relieving the DBA of this burden.

Growth of Rollback Segments



Growth of Rollback Segments

The pointer or the head of the rollback segment moves to the next extent when the current extent is full. When the last extent that is currently available is full, the pointer can move back to the beginning of the first extent only if that extent is free. The pointer cannot skip over an extent and move to the second or any other extent.

If the first extent is being used, the transaction allocates an additional extent for the rollback segment. This is called an *extend*. Similarly, if the head tries to move into an active extent, the rollback segment allocates an additional extent.

The Impact of Rollback Segment Extending

Rollback segments should not be extended during normal running. To prevent this, rollback segments must have enough extents to hold the rollback entries for the transactions.

As with other objects, you should avoid dynamic space management.

If you underestimate the size of rollback segments, performance is degraded and transactions may generate errors. If you overestimate, you use unnecessary space, and some performance issues may arise from having rollback segments that are too large.

Tuning the Manually Managed Rollback Segments

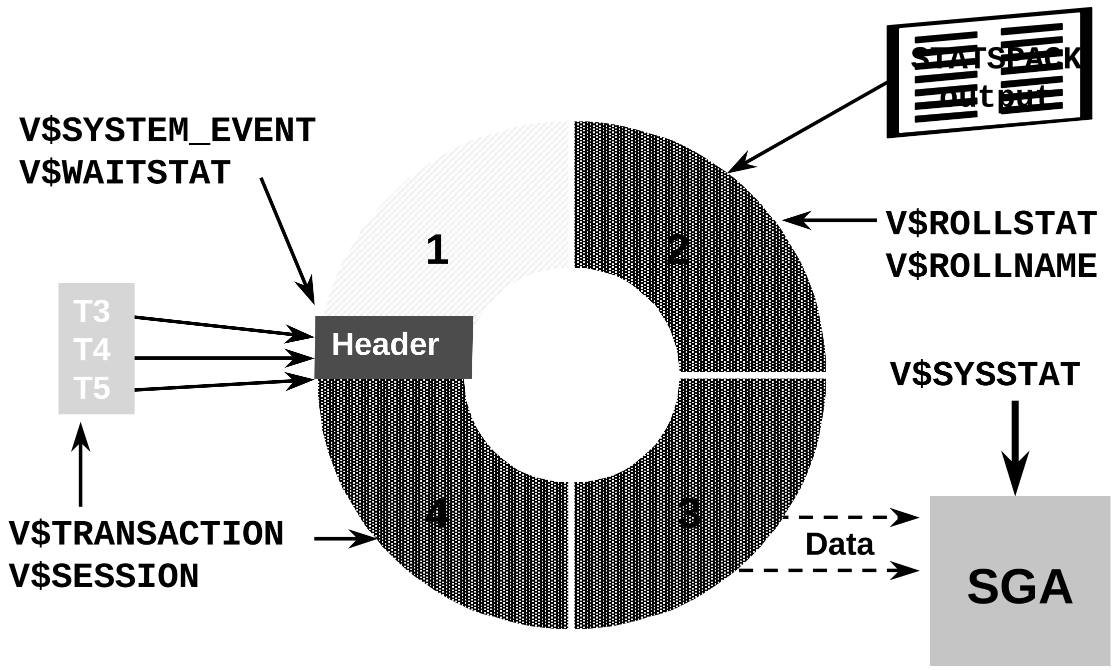
Goals in tuning rollback segments:

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

Tuning Goals

- Transactions should never wait for access to rollback segments. This requires you to have enough rollback segments.
- Rollback segments should not extend during normal running. This requires:
  - An appropriate number of extents per segment
  - The correct sizing of the extents
  - The appropriate number of rollback segments
  - A better use of utilities that can use less rollback per transaction, by committing more frequently.
- No transaction, however large or exceptional, should ever run out of rollback space. This means that:
  - Rollback segments should be sized correctly.
- For large transactions investigate whether these could be split into smaller transactions, by committing more frequently.
- Readers should always be able to see the read-consistent images they need. This requires the appropriate:
  - Number of rollback segments
  - Sizing of rollback segments

Diagnostic Tools



Dynamic Views to Monitor Rollback Activity

- V$ROLLNAME: Displays the name and number of the online rollback segments
- V$ROLLSTAT: Displays statistics of the activity for each online rollback segment:
  - Number of waits on the header transaction table
  - Volume of data written by the transactions
- V$SYSTEM_EVENT: The Undo Segment Tx Slot event shows waits for transaction slots and therefore contention on rollback segment headers.
- V$WAITSTAT: Displays the cumulative statistics of waits on header blocks and data blocks of all rollback segments
- V$SYSSTAT: Displays the number of consistent and data block gets. You can compare the number of waits with the total number of requests for data.
- V$TRANSACTION: Displays the current transactions using rollback segments and therefore the number of rollback segments required.

Except for V$ROLLNAME, all of these views use the undo segment number (USN) as the identifier for rollback. So when you need to get the name of the rollback segment, join the V$ROLLNAME on USN column.

Diagnosing Contention for Manual Rollback Segment Header

```
SQL> SELECT class, count FROM v$waitstat
  2  WHERE class LIKE '%undo%';
or
SQL> SELECT event, total_waits, total_timeouts
  2  FROM v$system_event
  3  WHERE event LIKE 'undo segment tx slot';
or
SQL> SELECT sum(waits)* 100 /sum(gets) "Ratio",
  2  sum(waits) "Waits", sum(gets) "Gets"
  3  FROM   v$rollstat;
```

- The number of waits for any rollback header should be less than 1% of the total number of requests.
- If not, create more rollback segments.

Diagnosing Contention for Rollback Segment Header

A nonzero value in the following indicates contention for rollback segments:

- `waits` column of the `V$ROLLSTAT` view
- `undo header` row of the `V$WAITSTAT` view
- Undo Segment Tx Slot event of the `V$SYSTEM_EVENT` view

The following statement queries the `V$WAITSTAT` view to look for contention on the rollback segment:

```
SQL> select class, count from v$waitstat
  2  where class like '%undo%';
```

Diagnosing Contention for Rollback Segment Headers (continued)

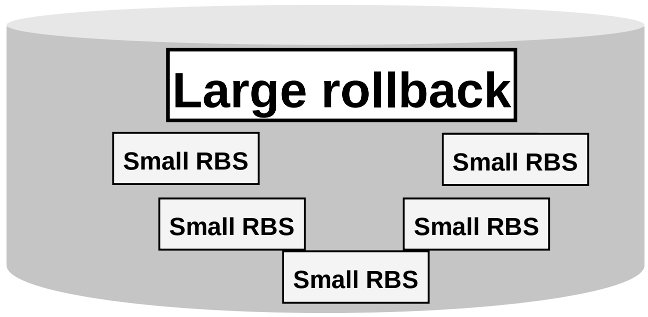
The rollback and undo related information from the STATSPACK are located mainly in the Rollback Segment Stats section. Following is an example of the Rollback Segment Stats section:

```
    Rollback Segment Stats for DB: ED31  Instance: ed31  Snaps: 1 –2
    ->A high value for "Pct Waits" suggests more rollback segments may be
    required
         Trans Table      Pct   Undo Bytes
  RBS No     Gets        Waits    Written        Wraps  Shrinks  Extends
  ------ ------------ ------- --------------- -------- -------- --------
       0          5.0    0.00               0        0        0        0
       1         66.0    0.00           5,636        0        0        0
       2        439.0    0.00         358,772        5        0        0
       3         50.0    0.00           6,314        0        0        0
       4         53.0    0.00           7,004        0        0        0
           -------------------------------------------------------------
```

Guideline

When you observe contention for rollback segments, you should investigate further the cause of contention to determine if the mere addition of rollback segments would alleviate the problem, or if it is necessary to configure the rollback tablespaces to manage the I/O.

Guidelines: Number of Manual Rollback Segments (RBSs)



- **OLTP: One RBS for four transactions**
- **Batch: One rollback segment for each concurrent job**

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT large_rbs;
```

9-23

OLTP Transactions

- OLTP applications are characterized by frequent concurrent transactions, each of which modifies a small amount of data. Assign small rollback segments to OLTP transactions.
- The reasonable rule of thumb is one rollback segment for every four concurrent transactions.

Long Batch Transactions

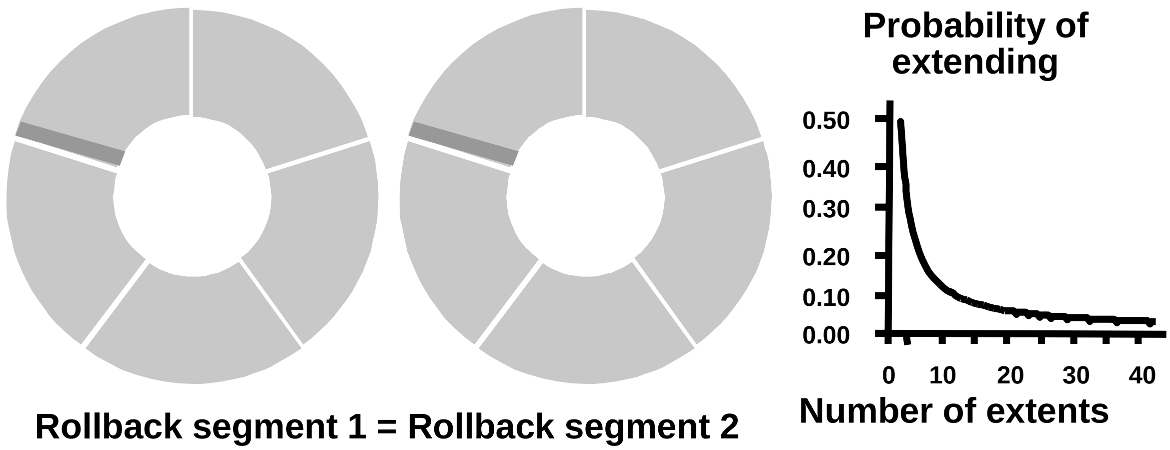
Assign large rollback segments to transactions that modify large amounts of data. Such transactions generate large rollback entries. If a rollback entry does not fit into a rollback segment, the Oracle server extends the segment. Dynamic extension reduces performance and should be avoided whenever possible.

Allow for the growth of the rollback segments by creating them in large or autoextensible tablespaces, with an unlimited value for `MAXEXTENTS`.

Long Batch Transactions (continued)

- For exceptionally long transactions, you may want to assign a large rollback segment using the following syntax:
  ```
  SQL> SET TRANSACTION USE ROLLBACK SEGMENT large_rbs;
  ```
  You can also use a supplied procedure:
  ```
  SQL> EXECUTE
  dbms_transaction.use_rollback_segment('large_rbs');
  ```
  Remember that any commit operation, explicit or implicit, ends the transaction. This means that the command may have to be included repeatedly.
  Note: The `SET TRANSACTION USE rollback segment` command must be the first one in the transaction.

Guidelines: Sizing Manual Rollback Segments



Rollback segment 1 = Rollback segment 2

INITIAL = NEXT = 2^n Mb

MINEXTENTS = 20

OPTIMAL = 20 * INITIAL

9-25

Storage Parameters

Setting the right size for the rollback segments is significant for performance. The aim is to reduce dynamic extension and increase the chances that undo blocks are in the buffer cache when needed.

- Choose a value for the `INITIAL` storage parameter from the list 8 KB, 16 KB, 32 KB, and 64 KB for small transactions, and 128 KB, 256 KB, 512 KB, 1 MB, 2 MB, 4 MB, and so on for larger transactions.
- Use the same value for `NEXT` as for `INITIAL`. Because `PCTINCREASE` is 0, all the other extents will have the same size as the `NEXT`.
- Make all your rollback segments the same size. Take the large rollback segments offline if they are not needed.
- Set `MINEXTENTS` to 20. This makes it unlikely that the rollback segment would need to grab another extent, because the extent that it should move into is still being used by an active transaction.

Tablespace Size

Leave enough free space in the rollback segments tablespace for a larger-than-usual transaction to be able to extend the rollback segment it is using. The `OPTIMAL` setting will later cause the extended rollback segment to shrink.

Sizing Transaction Rollback Data

- **Deletes are expensive for rollback activity.**
- **Inserts use minimal rollback space.**
- **Updates use rollback space, depending on the amount of data changed in the transaction.**
- **Index maintenance adds rollback.**

```
SQL> SELECT s.username, t.used_ublk, t.start_time
  2  FROM v$transaction t, v$session s
  3  WHERE t.addr = s.taddr;
```


Transaction Statements

The number of bytes required to store information that is needed in case of rollback depends on the type of transaction being performed:

- Deletes are expensive for rollback segments; they need to store the actual row itself. If you can use `TRUNCATE` instead, performance is improved.
- Inserts use little rollback space; only the row ID is kept.
- The amount used for updates depends on how many columns are being updated.
- Indexed values generate more rollback because the server process must change values in the index as well as in the table. For updates on indexed columns, the Oracle server records in the rollback segment the old data value, the old index value, and the new index value. Updating rows that change partitions will also generate more rollback.
- Direct path inserts / appends / loads are not likely to use much rollback.

Note: Columns of the `LOB` data type do not use rollback segment space for changes. They use their own segment space defined by the `PCTVERSION` clause setting.

Estimate the size of the rollback segment by running the longest expected transaction and checking the size of the rollback segment. For the current transaction, get the number of blocks used in a rollback segment:

```
SQL> SELECT s.username, t.used_ublk, t.start_time
  2    FROM v$transaction t, v$session s
  3  WHERE t.addr = s.taddr;
```

Sizing Transaction Rollback Data

- The number of bytes in rollback segments before execution of statements:

```
SQL> select usn,writes from v$rollstat;
```

- After execution of statements:

```
SQL> select usn,writes from v$rollstat;
```


Sizing Transaction Rollback Data Volume

Another way to estimate the volume of rollback data for a test transaction is to perform the following steps:

1. Before you execute the test transaction, display the current number of writes in the rollback segments:

```
SQL> select usn,writes from v$rollstat;
      USN     WRITES
--------- ---------
        0      1962
        1   1102686
        2     32538
        3   1226096
```

2. Execute the test transaction:

```
SQL> update employees set salary=1000;
6560 rows updated.
```

Sizing Transaction Rollback Data Volume (continued)

3. Display the new number of writes in the rollback segments:

```
SQL> select usn,writes from v$rollstat;
USN WRITES
--- -------
0      1962
1   2232270
2     32538
3   1226096
```

Calculate the difference between the new and the old number of writes in the USN 1 rollback segment to determine the amount of rollback data used for this test transaction. In order for this to be accurate you would have to ensure that:

- The transaction used the rollback segment USN1.
- No other transaction used the rollback segment USN1

Using Less Rollback Per Transaction

- The design of the application should allow users to commit transactions regularly.
- Developers should not code long transactions.

9-29

Transactions

You may be able to reduce rollback segment wastage by training users and developers to do the following:

- Users should commit work regularly so that their transactions do not lock others out of rollback segment extents.
- Developers should not code unnecessarily long transactions.

Using Less Rollback

- **Export and Import operations**
  - **Import**
    Set `COMMIT = Y`
    Size the set of rows with the `BUFFER` keyword
  - **Export: Set `CONSISTENT = N`**
- **SQL*Loader operations: Set the commit intervals with `ROWS`.**
- **Developers should make sure that the transactions are not unduly long.**

Import

- Set `COMMIT = Y` to make sure that each set of inserted rows is committed as the import goes on.
- Size the set of rows with the `BUFFER_SIZE` keyword.

Export

The `CONSISTENT` option specifies whether or not Export uses the `SET TRANSACTION READ ONLY` statement to ensure that the data seen by Export is consistent to a single point in time and does not change during the execution of the export command.

You should specify `CONSISTENT = Y` when you anticipate that other applications will be updating the target data after an export has started. However, this will mean that any modified data must be kept in the rollback segments until required by the export process. If this data is not available an error will be reported, and the export process will abort.

Setting `CONSISTENT = N` prevents the transaction from being set as read-only.

SQL*Loader

For conventional path loading, set the commit intervals with the `ROWS` keyword.

Possible Problems Caused by Small Rollback Segments

- **The transaction fails for lack of rollback space.**
- **A “snapshot too old” error occurs if:**

  The statement requires data that has been modified, committed, and the rollback data is no longer available.

9-31

ORACLE

Large Transactions

If a transaction is exceptionally large, it may fail because the rollback segment cannot expand:

- The rollback segment has reached its maximum number of extents.
- There is no room left in the tablespace for the rollback segment to expand.

You need bigger rollback segments or more space in the tablespace.

Snapshot Too Old

If a query fails with the following error message, the rollback image needed for read consistency has probably been overwritten by an active transaction:

```
ORA-01555: snapshot too old (rollback segment too small)
```

Occasionally, you may get this error even if there are no other transactions active.

To resolve this error, you need bigger rollback segments. You should also avoid running batch type queries during the day time. If not avoidable, then the long running (queries/load operations) should use a set transaction use rollback segment statement at their beginning.

Summary

In this lesson, you should have learned how to:

- Use automatic managed undo segments
- Managing automatic managed undo segments
- Avoid contention for rollback segment headers
- Work out the appropriate numbers and sizes of rollback segments
- Monitor the rollback space used by transactions
- Identify possible rollback segment problems

Practice 9

The objective of this practice is to use available diagnostic tools to monitor and tune the rollback segments. This would require setting the database to Manual Undo Management mode. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Set the database in Manual Undo Mode.
   a) Connect sys/oracle as SYSDBA and use shutdown immediate to close the database.
   b) Edit the initialization parameter file locate `$HOME/ADMIN/PFILE` and comment out the following lines:
      `undo_management = AUTO`
      `undo_tablespace = UNDOTBS`
   c) Save the modifications and start up the database. `Confirm that the UNDO_MANAGEMENT` is `MANUAL, and UNDO_TABLESPACE is null`
2. Connect sys/oracle as SYSDBA and create a new rollback segment tablespace `RBS_TEST` that is 2 MB in size using the `CREATE TABLESPACE` cammand. Name the datafile rbs_test.dbf and place it in the `$HOME/ORADATA/u03` directory. Create the tablespace as dictionary managed.

   Note: This is *not* to be an UNDO tablespace, and you must specify that it is to be dictionary managed.
3. For the purposes of this practice, create a new rollback segment called RBSX in the `RBS_TEST` tablespace. For the storage parameters, use 64 KB for the `INITIAL` and `NEXT` extent sizes with `MINEXTENTS` value set to 20. Set the `OPTIMAL` value so that the segment shrinks back to 1280 KB automatically.
4. Bring the `rbsx` rollback segment online and ensure that any others (except the SYSTEM rollback segment) are offline. Query the `DBA_ROLLBACK_SEGS` view to get the segment_name and status of the rollback segments to be taken offline using the `ALTER ROLLBACK SEGMENT` command.
5. Before executing a new transaction, find the number of bytes written so far in the RBSX rollback segment, using the writes column of `v$rollstat`.
6. Open two sessions. In session 1 connect as hr/hr, and run the script `$HOME/STUDENT/LABS/ins_temps.sql`. The script inserts 100 new rows into the `TEMP_EMPS` table. Do *not* COMMIT this transaction. In the second session, log in as system/manager, determine how many rollback segment blocks or bytes the transaction is using? To do this query the writes column of `V$ROLLSTAT` to get the number of bytes written in the RBSX rollback segment so far. Record this value.

   Note: The number of writes in the rollback segment between questions 5 and 6 is the difference in the value of the writes column at the respective times.

Practice 9 (continued)

7. Join the V$TRANSACTION and V$SESSION views to find, in the USED_UBLK column, how many blocks the ins_temps transaction is using.
8. Return to the hr session (the first session) and commit the insert. Run the $HOME/STUDENT/LABS/del_temps.sql script. Do *not* COMMIT. The script deletes the hundred rows you have just inserted. As user system (in the second session), check the amount of rollback space used, using the writes column of v$rollstat. Note the difference between the return value, and that found in question 6.
9. Connect as system/manager and find out if you have had any rollback segment contention since startup, using the waits and gets columns in the v$rollstat view. **Note:** In a production environment a better source of information would be "Rollback Segment" section in the STATSPACK report.
10. Does the V$SYSTEM_EVENT view show any waits related to rollback segments? Query in V$SYSTEM_EVENT view for the "undo segment tx slot" entry.
11. Connect as hr/hr and run the $HOME/STUDENT/LABS/ins_temps.sql script again, allocating the transaction to a specific rollback segment RBSX, using the set transaction use rollback segment command. To check that the transaction is using the defined rollback segment join the V$ROLLSTAT, V$SESSION, and V$TRANSACTION views.
12. Set the database in Auto Undo Mode.
    a) Connect sys/oracle as SYSDBA and use shutdown immediate to close the database.
    b) Edit the initialization parameter file locate $HOME/ADMIN/PFILE and uncomment the following lines
       undo_management = AUTO
       undo_tablespace = UNDOTBS
    c) Save the modifications and start up the database. Confirm that the UNDO_MANAGEMENT is AUTO, and UNDO_TABLESPACE is UNDOTBS

10

Monitoring and Detecting Lock Contention

Objectives

After completing this lesson, you should be able to do the following:

- Define levels of locking
- List possible causes of contention
- Use Oracle utilities to detect lock contention
- Resolve contention in an emergency
- Prevent locking problems
- Recognize Oracle errors arising from deadlocks

ORACLE

10-2

Locking Mechanism

- Automatic management
- High level of data concurrency
  - Row-level locks for DML transactions
  - No locks required for queries
- Varying levels of data consistency
- Exclusive and Share lock modes
- Locks held until commit or rollback operations are performed
- Quiesced database

Lock Management

The Oracle server automatically manages locking. The default locking mechanisms lock data at the lowest level of restriction to guarantee data consistency while allowing the highest degree of data concurrency.

Note: The default mechanism can be modified by the `ROW_LOCKING`. The default value is `ALWAYS`, which leads the Oracle server to always lock at the lowest and least restrictive level (the row level, not the table level) during DML statements. The other possibility is to set the value to `INTENT`, which leads the Oracle server to lock at a more constraining level (the table level), except for a `SELECT FOR UPDATE` statement, for which a row-level lock is used.

Quiesced Database

Oracle9*i*, Release 1 (9.0.1), enables a DBA to put the system into quiesced state. The system is in quiesced state if there are no active sessions, other than SYS and SYSTEM. An active session is defined as a session that is currently inside a transaction, a query, a fetch, or a PL/SQL procedure, or a session that is currently holding any shared resources, such as enqueues. DBAs are the only users who can proceed when the system is in quiesced state.

This state is useful for a DBA when performing some housekeeping, such as creating indexes. Since no one else is allowed on, there is no chance of the DBA having to wait for locks to be released. Performance would also improve since there is no load on the database, except what the DBA imposes.

Data Concurrency

Locks are designed to allow a high level of *data concurrency*; that is, many users can safely access the same data at the same time.

- Data Manipulation Language (DML) locking is at row level.

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	`SQL> UPDATE employee`
	`  2  SET salary=salary*1.1`
	`  3  WHERE id= 24878;`
	`1 row updated.`

- A query holds no locks, unless the user specifies that it should.

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary+1200;`
`13120 rows updated.`
	`SQL> SELECT salary`
	` 2   FROM employee`
	` 3   where id = 10;`
	`   SALARY`
	`---------`
	`     1000`

Data Consistency

The Oracle server also provides varying levels of *data consistency*; that is, the user sees a static picture of the data, even if other users are changing it.

Duration

Locks are held until the transaction is committed, rolled back, or terminated. If a transaction terminates abnormally, the `PMON` process cleans up the locks.

Locking Modes

- Exclusive lock mode prevents the associated resource from being shared with other transactions, until the exclusive lock is released.
  Example: Exclusive locks are set at row level for a DML transaction:

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	`SQL> UPDATE employee`
	`  2  SET salary=salary*1.1`
	`  3  WHERE id= 24877;`
	`Transaction 2 waits.`

- In Share lock mode, several transactions can acquire share locks on the same resource.
  Example: Shared locks are set at table level for DML transactions:

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	`SQL> UPDATE employee`
	`  2  SET salary=salary*1.1`
	`  3  WHERE id= 24878;`
	`1 row updated.`

The two transactions update different rows in the same table.

Lock Duration

- Transactions hold locks until the transactions are committed or rolled back:

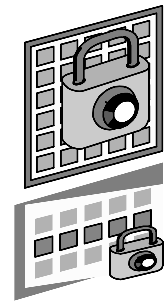
Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
`SQL> commit;`
`Commit complete.`
	`SQL> UPDATE employee`
	`  2  SET salary=salary*1.1`
	`  3  WHERE id= 24877;`
	`Transaction 2 waits until`
	`transaction 1 is committed.`
	`1 row updated.`

As soon as Transaction 1 is committed, Transaction 2 can update the row, because the transaction acquired the requested lock. Transaction 2 must wait because it is wanting to update the same row as Transaction 1.

Two Types of Locks

- **DML or data locks:**
  - **Table-level locks**
  - **Row-level locks**



(TM)

(TX)

- **DDL or dictionary locks**

DML Locks

DML locks guarantee the integrity of data being accessed concurrently by multiple users for incorporating changes. They prevent destructive interference of simultaneous conflicting DML and DDL operations.

DML Levels: A *table-level* lock (TM type) is set for any DML transaction that modifies a table: `INSERT`, `UPDATE`, `DELETE`, `SELECT...FOR UPDATE`, or `LOCK TABLE`. The table lock prevents DDL operations that would conflict with the transaction.

Example

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2   SET salary=salary*1.1;`
`13120 rows updated.`
	`SQL> DROP TABLE employee;`
	`ERROR at line 1:`
	`ORA-00054: resource busy and acquire with NOWAIT specified`

DML Locks (continued)

The *row-level* lock (TX type) is automatically acquired for each row modified by `INSERT`, `UPDATE`, `DELETE`, or `SELECT...FOR UPDATE` statements. The row-level lock ensures that no other user can modify the same row at the same time. Therefore, there is no risk that a user can modify a row that is being modified and not yet committed by another user.

Example

Transaction 1	Transaction 2

`SQL> update employee`
`  2   set salary=salary*1.1`
`  3   where id= 24877;`
`1 row updated.`
	`SQL> update employee`
	`  2   set salary=salary*1.1`
	`  3   where id= 24877;`
	`Transaction 2 waits.`

DDL Locks

A DDL lock protects the definition of a schema object while that object is acted upon or referred to by an ongoing DDL operation. The Oracle server automatically acquires a DDL lock to prevent any destructive interference from other DDL operations that might modify or reference the same schema object.

DML Locks

- A DML transaction gets at least two locks:
  - A shared table lock
  - An exclusive row lock
- The enqueue mechanism keeps track of:
  - Users waiting for locks
  - The requested lock mode
  - The order in which users requested the lock

DML Transactions Acquire at Least Two Locks

Two kinds of lock structures are used for DML statements (`INSERT`, `UPDATE`, `DELETE`, or `SELECT...FOR UPDATE`):

- The transaction gets a shared lock on the table that is referenced as a TM lock, no matter what shared lock mode it is.
- The transaction gets an exclusive lock on the rows it is changing, referenced as a TX lock. Each row gets a lock byte turned on in the row header pointing to the ITL slot used by the transaction. The lock mode at row level can only be exclusive.

Enqueue Mechanism

The Oracle server maintains all locks as *enqueues*. The enqueue mechanism can keep track of:

- Users waiting for locks held by other users
- The lock mode these users require
- The order in which users requested the lock

If three users want to update the same row at the same time, all of them get the shared table lock, but only one (the first) gets the row lock. The table-locking mechanism keeps track of who holds the row lock and who waits for it.

You can increase the overall number of locks available for an instance by increasing the values of the `DML_LOCKS` and `ENQUEUE_RESOURCES` parameters. This may be necessary in a Real Application Cluster configuration.

Table Lock Modes

These table lock modes are automatically assigned by the Oracle server:

- Row Exclusive (RX): `INSERT, UPDATE, DELETE`
- Row Share (RS): `SELECT... FOR UPDATE`

Automatic Table Lock Modes

You often see the two TM table lock modes held by DML transactions: RX and RS.

These are the table lock modes automatically assigned by the Oracle server for DML transactions.

The restrictiveness of a table lock's mode determines the modes in which other table locks on the same table can be obtained and held.

Row Exclusive (RX)

- Permits other transactions to query, insert, update, delete, or lock other rows concurrently in the same table
- Prevents other transactions from manually locking the table for exclusive reading or writing

Example

Transaction 1 (RX table lock held)	Transaction 2 (RX table lock held)

SQL> update employee
  2   set salary=salary*1.1
  3   where id= 24877;
1 row updated.
	SQL> update employee
	  2 set salary=salary*1.1
	  3 where id= 24878;
	1 row updated.

Automatic Table Lock Modes (continued)

Row Share (RS)

You can choose to lock rows during a query by using the `SELECT ... FOR UPDATE` statement.

This prevents other transactions from manually locking the table for exclusive write access.

Example

<table>
<tr><th>Transaction 1 (RS table lock held)</th><th>Transaction 2 (RX table lock held)</th></tr>
<tr><td><pre>SQL> select id,salary
  2 from employee
  3 where id=24877
  4 for update;
       ID   SALARY
--------- ---------
    24877   1100
SQL> commit;
Commit complete.</pre></td><td><pre>SQL> lock table employee
  2   in exclusive mode;
Transaction 2 waits.

Table(s) Locked.</pre></td></tr>
</table>

Table Lock Modes

- **Manually acquired in `LOCK TABLE` statement:**

```
SQL> LOCK TABLE table_name IN mode_name MODE;
```

- **Share (S)**
  - **No DML operations allowed**
  - **Implicitly used for referential integrity**

10-12

ORACLE

Manual Table Lock Modes

The three other table lock modes are assigned manually by an explicit LOCK TABLE command. For example:

```
SQL> LOCK TABLE employee IN exclusive MODE;
Table(s) Locked.
```

Often there are good application reasons for explicit locking, but if you get lock contention you may want to check with the developers.

Non-Oracle developers sometimes use unnecessarily high locking levels.

Share (S) Lock Mode

This lock mode permits other transactions to only query the SELECT ... FOR UPDATE table. It prevents any modification to the table.

The SQL statements that implicitly get a share lock involve referential integrity constraints.

Table Lock Modes

- Share Row Exclusive (SRX)
  - No DML operations or Share mode allowed
  - Implicitly used for referential integrity
  - Requires index on foreign key in child table in pre-Oracle9*i*
- In Oracle9*i*:
  - Implementation of referential integrity constraint has changed.
  - No index is required on the foreign key column in the child table.
- Exclusive (X)

ORACLE

Share Row Exclusive (SRX) Lock Mode

This is an even higher level of table lock, which prevents DML statements and the manual share lock mode from being acquired.The SQL statement that implicitly gets a Share Row Exclusive lock again involves referential integrity.

Note: In pre-Oracle9*i* releases, the recommendation was is to index the foreign key column on the child table. In Oracle9*i*, the implementation of foreign key constraint has been modified and thus the reason for this recommendation falls away.

Exclusive (X) Lock Mode

This is the highest level of table lock, thus the most restrictive mode. Exclusive table lock:

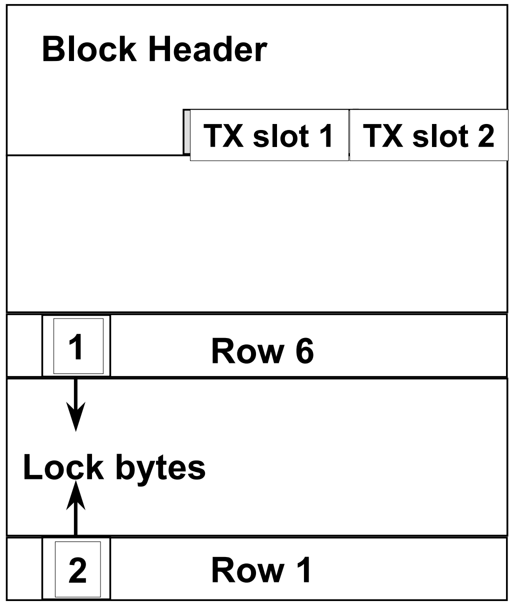
- Permits other transactions only to query the table
- Prevents any type of DML statements and any manual lock mode

Example

Transaction 1 (X table lock held)	Transaction 2 (RX table lock requested)

`SQL> LOCK TABLE department IN EXCLUSIVE MODE;`
`Table(s) Locked.`
	`SQL> SELECT * from department`
	`  2   FOR UPDATE;`
	`Transaction 2 waits.`

DML Locks in Blocks



10-14

Technical Note

This locking information is not cleared out when transactions are committed, but rather when the next query reads the block. This is known as delayed block cleanout.

The query that does the cleaning must check the status of the transaction and the system change number (SCN) in the transaction table held in the rollback segment header.

Within blocks, the Oracle server keeps an identifier for each active transaction in the block header. At row level, the lock byte stores an identifier for the slot containing the transaction.

Example: In the diagram shown on the slide, the transaction using slot 1 is locking row 6, and the transaction in slot 2 is locking row 1.

DDL Locks

- **Exclusive DDL locks are required for:**
  - **DROP TABLE statements**
  - **ALTER TABLE statements**
  - **(The lock is released when the DDL statement completes.)**
- **Shared DDL locks are required for:**
  - **CREATE PROCEDURE statements**
  - **AUDIT statements**
  - **(The lock is released when the DDL parse completes.)**
- **Breakable parse locks are used for invalidating statements in the shared SQL area.**

ORACLE®

DDL Locks

You are unlikely to see contention for DDL locks because they are held only briefly and are requested in NOWAIT mode. There are three types of DDL locks.

Exclusive DDL Locks

Some DDL statements, such as CREATE, ALTER and DROP, must get an exclusive lock on the object they are working on.

Users cannot get an exclusive lock on the table if any other user holds any level of lock, so an ALTER TABLE statement fails if there are users with uncommitted transactions on that table.

Example

Transaction 1	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1;`
`3120 rows updated.`
	`SQL> ALTER TABLE employee`
	`  2  DISABLE PRIMARY KEY;`
	`ORA-00054: resource busy and`
	`acquire with NOWAIT specified`

DDL Locks (continued)

Exclusive DDL Locks (continued)

Note: Use locally managed tablespaces instead of dictionary-managed tablespaces. This will be discussed further in a later lesson.

Shared DDL Locks

Some statements, such as `GRANT` and `CREATE PACKAGE`, need a shared DDL lock on the objects they reference.

This type of lock does not prevent similar DDL statements, or any DML statements, but it prevents another user from altering or dropping the referenced object.

Breakable Parse Locks

A statement or PL/SQL object in the library cache holds one of these locks for every object it references, until the statement is aged out of the shared pool.

The *breakable parse lock* is there to check whether the statement should be invalidated if the object changes.

You could think of this lock as a pointer. It never causes waits or contention. However, this does impact the system in that when a breakable parse lock on an object is broken any objects, such as cursors and procedures, that reference that object will require parsing again. This could cause potential contention on the Shared Pool.

Possible Causes of Lock Contention

- Unnecessarily high locking levels
- Long-running transactions
- Uncommitted changes
- Other products imposing higher-level locks

ORACLE

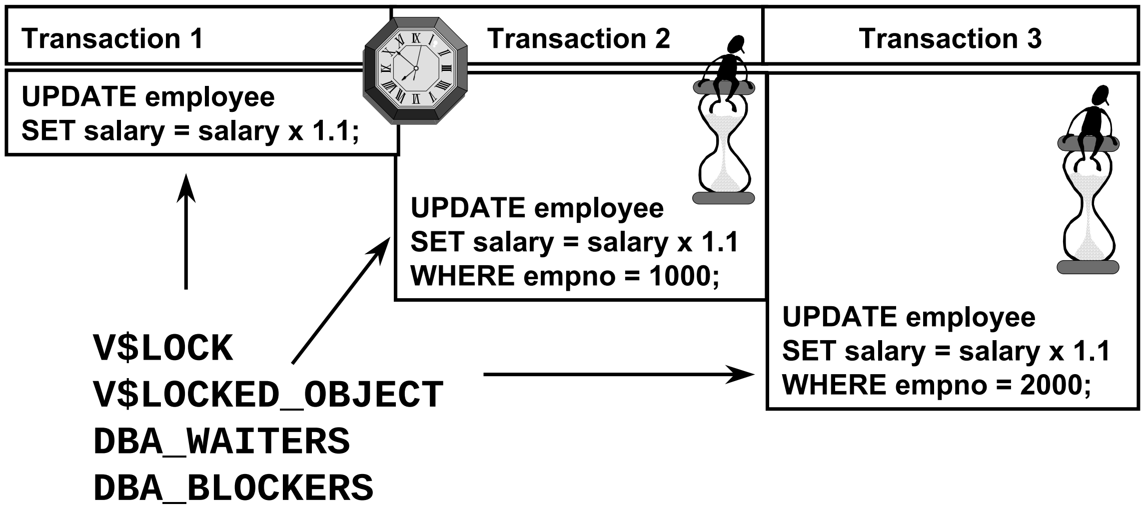
10-17

Development and User Issues

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

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

Diagnostics Tools for Monitoring Locking Activity



DBA_WAITERS and DBA_BLOCKERS

These views give further insight into who is holding or waiting for which tables. In order to create these views the script CATBLOCK.SQL needs to be run. It is found in the $ORACLE_HOME/rdbms/admin directory.

The V$LOCK View

Lock type	ID1
TX	Rollback Segment number and slot number
TM	Object ID of the table being modified

Example

To find the table name that corresponds to a particular resource ID 1 of the V$LOCK view:

```
SQL> select object_name
  2   from dba_objects, v$lock
  3   where object_id=id1 and type='TM';'
```

Any process that is blocking others is likely to be holding a lock obtained by a user application. The locks acquired by user applications are:

- Table locks (TM)
- Row-level locks (TX)

The V$LOCKED_OBJECT View

Lock Type	ID1
XIDUSN	Rollback segment number
OBJECT_ID	ID of the object being modified
SESSION_ID	ID of the session locking the object
ORACLE_USERNAME	
LOCKED_MODE	

Example

To find the table name that corresponds to a particular object ID in the V$LOCKED_OBJECT view:

```
SQL> SELECT xidusn, object_id, session_id, locked_mode
  2  FROM v$locked_object;
   XIDUSN OBJECT_ID SESSION_ID LOCKED_MODE
--------- --------- ---------- -----------
        3      2711          9           3
        0      2711          7           3


SQL> SELECT object_name FROM dba_objects
  2  WHERE object_id = 2711;
OBJECT_NAME
-------------
EMPLOYEE
```

If the value of XIDUSN is 0, then the session with the corresponding session ID is requesting and waiting for the lock being held by the session, for which XIDUSN value is different from 0.

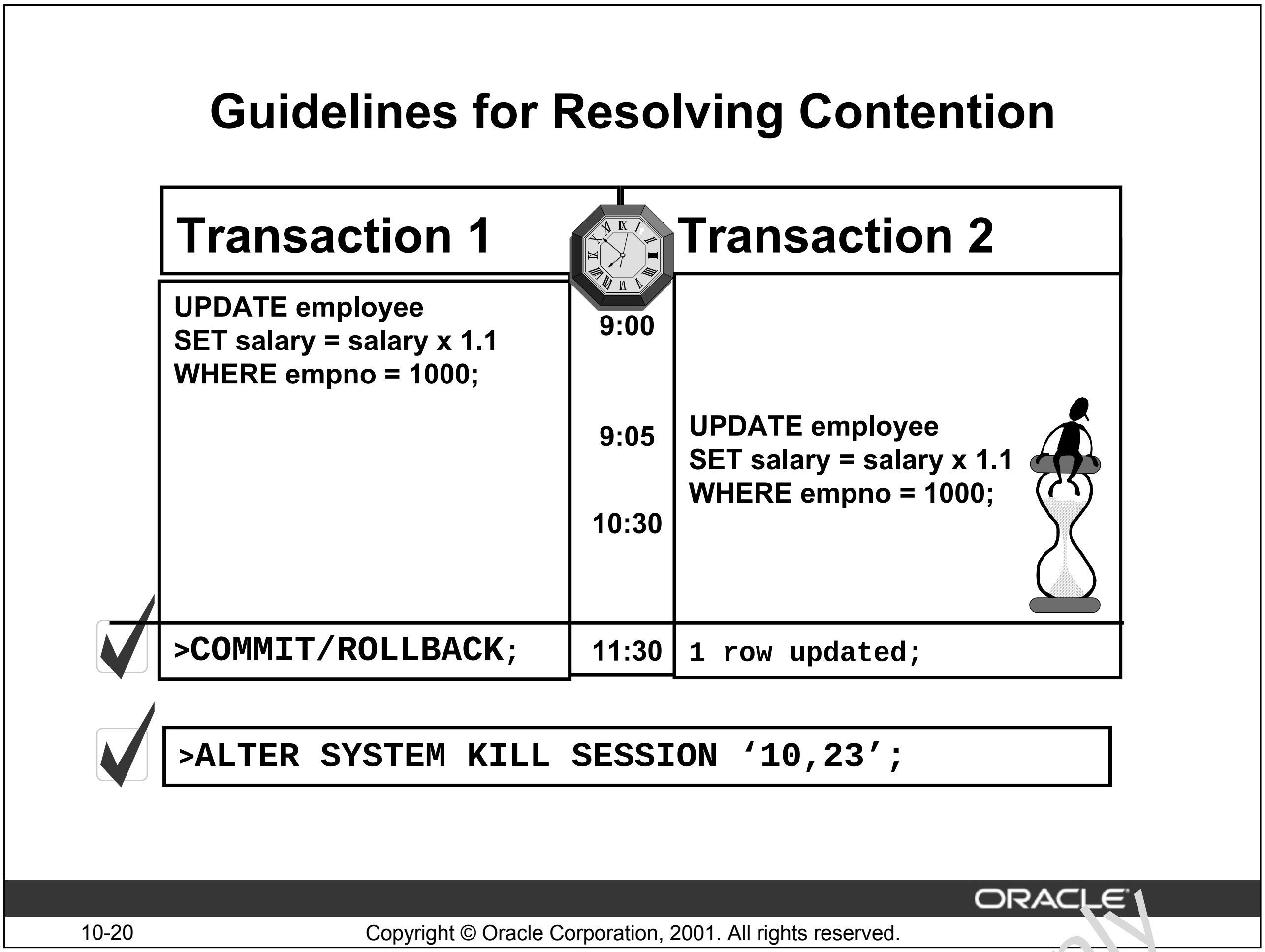
UTLLOCKT Script

You can also use the UTLLOCKT.SQL script to display lock wait-for in a hierarchy. The script prints the sessions that are waiting for locks and the sessions that are blocking.

You must have run the CATBLOCK.SQL script (found in $ORACLE_HOME/rdbms/admin folder) as sysdba user before using UTLLOCKT.SQL. CATBLOCK.SQL creates the views, dba_locks and dba_blockers, along with others that will be used by UTLLOCKT.SQL.

For example, in the following output session 9 is waiting for session 8, Sessions 7 and 10 are waiting for 9.

```
WAITING_SESSION   TYPE MODE REQUESTED    MODE HELD       LOCK ID1 LOCK ID2
----------------- ---- ----------------- --------------- -------- --------
8                 NONE None              None            0        0
    9             TX   Share (S)         Exclusive (X)   65547    16
       7          RW   Exclusive (X)     S/Row-X (SSX)   33554440  2
      10          RW   Exclusive (X)     S/Row-X (SSX)   33554440  2
```



Killing Sessions

If a user is holding a lock required by another user, you can:

- Contact the holder and ask this user to commit or roll back the transaction
- As a last resort, kill the Oracle user session; this rolls back the transaction and releases locks

Any of the monitoring methods detailed above will give you the session identifier for the user.

You can kill user sessions with the `ALTER SYSTEM KILL SESSION SQL` command:

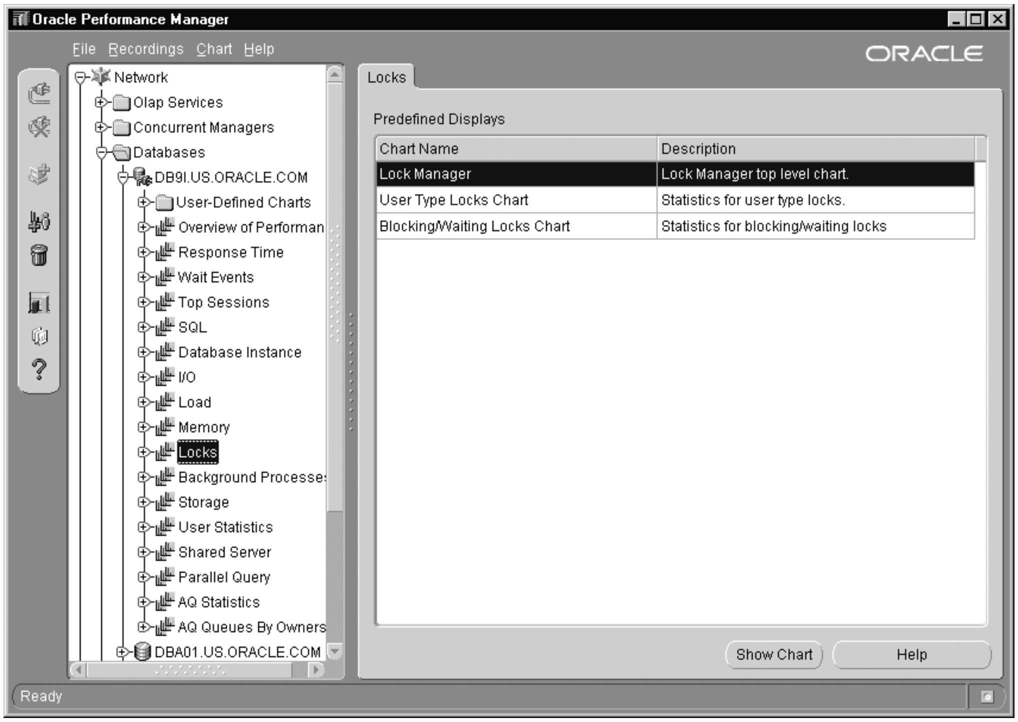
```
SQL> SELECT sid,serial#,username
  2  FROM v$session
  3  WHERE type=’USER’;
       SID   SERIAL#   USERNAME
---------- --------- ------------------------------
         8       122   SYSTEM
        10        23   SCOTT
SQL> ALTER SYSTEM KILL SESSION '10,23';
System altered.
```

Which Row Is Causing Contention?

If you need to know which row is causing contention, the `V$SESSION` view contains the following columns:

- `row_wait_block#`
- `row_wait_row#`
- `row_wait_file#`
- `row_wait_obj#`

Performance Manager: Locks

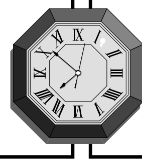


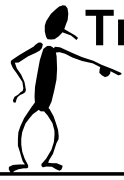
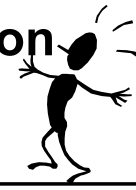
ORACLE

Performance Manager: Locks

The Performance Manager has a set of charts labeled Locks. This set of charts can be used to determine what locks are causing other users to have to wait. These are termed “blocking locks” and must be resolved before the waiting transaction can proceed.

Deadlocks



 Transaction 1		Transaction 2 
UPDATE employee SET salary = salary x 1.1 WHERE empno = = 1000;	9:00	UPDATE employee SET manager = 1342 WHERE empno = 2000;
UPDATE employee SET salary = salary x 1.1 WHERE empno = 2000;	9:15	UPDATE employee SET manager = 1342 WHERE empno = 1000;
ORA-00060: Deadlock detected while waiting for resource	9:16	

10-23

Deadlocks

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

The Oracle server automatically detects and resolves deadlocks by rolling back the statement that detected the deadlock.

Transaction 1	Time	Transaction 2

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24877;`
`1 row updated.`
	1	`SQL> UPDATE employee`
		`  2  SET salary=salary*1.1`
		`  3  WHERE id= 24876;`
		`1 row updated.`

`SQL> UPDATE employee`
`  2  SET salary=salary*1.1`
`  3  WHERE id= 24876;`
Transaction 1 waits.
	2	`SQL> UPDATE employee`
		`  2  SET salary=salary*1.1`
		`  3  WHERE id= 24877;`
		Transaction 2 waits.

`ORA-00060: deadlock detected` `while waiting for resource`	3	

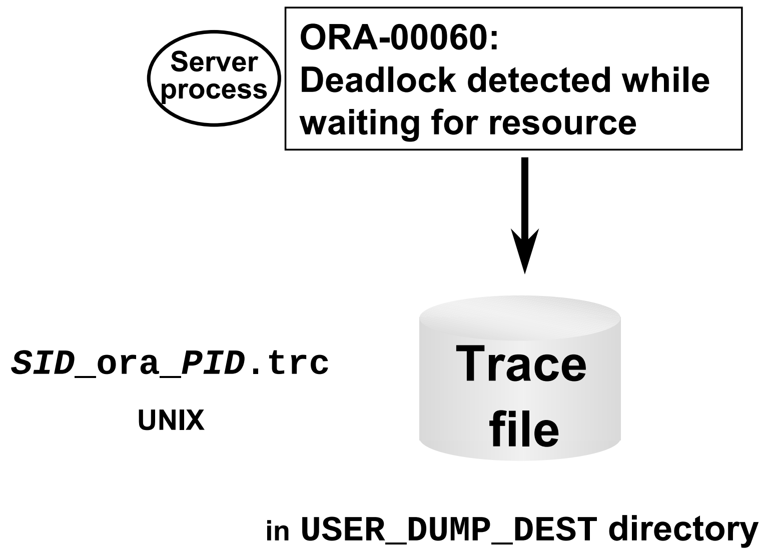
Deadlocks (continued)

If the second update in Transaction 1 detects the deadlock, the Oracle server rolls back that statement and returns the message. Although the statement that caused the deadlock is rolled back, the transaction is not, and you receive an ORA-00060 error. Your next action should be to roll back the remainder of the transaction.

Technical Note

Deadlocks most often occur when transactions explicitly override the default locking of the Oracle server. Distributed deadlocks are handled in the same way as nondistributed deadlocks.

Deadlocks



Trace File

A deadlock situation is recorded in a trace file in the `USER_DUMP_DEST` directory. It is advisable to monitor trace files for deadlock errors to determine whether there are problems with the application. The trace file contains the row IDs of the locking rows.

In distributed transactions, local deadlocks are detected by analyzing a "waits for" graph, and global deadlocks are detected by a time-out.

Once detected, nondistributed and distributed deadlocks are handled by the database and application in the same way.

Summary

In this lesson, you should have learned that:

- Queries do not lock data, except by choice
- DML statements use row-level and table-level locks on tables
- Exclusive locks are rarely used
- You can monitor locks by using the `V$LOCK`, `V$LOCKED_OBJECT`, `DBA_WAITERS`, and `DBA_BLOCKERS` views

Practice 10

The objective of this practice is to use available diagnostic tools to monitor lock contention.

You will need to start three sessions, in separate windows. Log in as hr/hr in two separate sessions (sessions 1 and 2) and as sys/oracle as sysdba in a third session (session 3). Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQL*Plus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. In session 1 (user hr/hr), update the salary by 10% for all employee with a salary < 15000 in the `temp_emps` table. Do *not* COMMIT.
2. In session 3 (sys/oracle as sysdba) check to see if any locks are being held by querying the `V$LOCK`.
3. In session 2 (user hr/hr – the session not yet used) drop the `TEMP_EMPS` table. Does it work?

Note: The DDL statement requires an exclusive table lock. It cannot obtain it, because session 1 already holds a row exclusive table lock on the `TEMP_EMPS` table.

4. In session 2 (hr/hr), update the salary by 5% for all employee with a salary > 15000 in the `temp_emps` table. Do *not* COMMIT.
5. In session 3, check to see what kind of locks are being held on the `TEMP_EMPS` table, using the `V$LOCK` view.
6. Roll back the changes you made in the second session (using hr/hr), and set the `manager_id` column to 10 for all employees who have a salary < 15000.
7. In session 3, check to see what kind of locks are being held on the `TEMP_EMPS` table, using the `V$LOCK` view.
8. In session 3, run the script `$ORACLE_HOME/rdbms/admin/catblock.sql`. The script will create a view `DBA_WAITERS`, that gives information regarding sessions holding or waiting on a lock. Use this view to determine the session id for the session that is holding locks. Use this value to query `v$session` to get the serial number for the session holding the lock. Then issue the alter system kill session command in order to release the session holding the lock.

Note: The second session would now show the blocking message.

Lock Matrix

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Initialization parameters	None	None	No locks on reads
Lock table in Row Share mode	Mode 2	TM(RS) lock on table	Mode 6, so no exclusive DDL (this is the least restrictive lock.)
Lock table partition in Row Share mode	Mode 2 Mode 2	TM (RS)lock on table TM (RS) lock on table partition	Mode 6, so no exclusive DDL (This is the least restrictive lock.
Select for update	Mode 2 Mode 2 Mode 6	TM (RS) lock on table TM (RS) lock on each table partition TX lock on RBX TX slot	Mode 6 and any selects for update or DML on same rows No exclusive DDL
Lock table in Row Exclusive mode	Mode 3	TM (RX) lock on table	Modes 4, 5, 6 (updates allowed, because mode 3 does not conflict with mode 3.) No share locks and no referential integrity locks
Lock table partition in Row Exclusiving mode	Mode 3 Mode 3	TM (RX) lock on table TM (RX) lock on table partition	Modes 4, 5, 6 on the same partition Updates allowed, because mode 3 does not conflict with mode 3 No share locks and no referential integrity locks
DML (up/ins/del)	Mode 3 Mode 6	TM (RX) lock on table TX lock on RBS TX slot	Modes 4, 5, 6 Select for update or DML on same rows No share locks and no referential integrity locks
DML (up/ins/del) on a partioned table	Mode 3 Mode 3 Mode 6	TM (RX) lock on table TM (RX) lock on each table partition owning the updated rows TX lock on RBS TX slot	Modes 4, 5, 6 Select for update or DML on same rows No share locks and no referential integrity locks

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Lock table in Share mode	Mode 4	TM (S) lock on table	Modes 3, 5, 6 Allows Select for Update and other Share Locks No possible ORA 1555 error on locked table
Lock table partition in Share mode	Mode 2 Mode 4	TM(RS) lock on table TM(S) lock on table partition	Mode 3,5,6 on the same partition Allows Select for Update and other Share locks No possible ORA 1555 on locked table
Delete from/update to parent table with referential integrity constraint on child table, *no index on FK column in child table*, and no ON DELETE CASCADE in the FK constraint	Mode 4 Mode 3 Mode 6	TM(S) lock on child TM(RX) lock on parent TX lock on RBS TX slot	Mode 3,5,6 on child Allows Select for Update and Share lock on child No possible ORA 1555 on child Mode 4,5,6 on parent Select for update or DML on same rows against parent
Delete from/update to parent table with referential integrity constraint on child *with index on FK column in child table* and no ON DELECT CASCADE in the FK constraint	Mode 3 Mode 6	TM(RX) lock on parent TX lock on RBS TX slot	Mode 4,5,6 and any selects for update or DML on same rows against parent Updates against rows in child referred to by DML against parent
Lock table in Share Row Exclusive mode	Mode 5	TM(SRX) lock on table	Mode 3,4,5,6 Allows Select for Update only No Share locks No ORA 1555 No cascaded deletes

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Lock table in partition in Share Row Exclusive mode	Mode 2 Mode 4	TM(RS) lock on table TM(S) lock on table partition	Mode 4 on the same partition Mode 3,5,6 on any partition Allows Select for Update only No ORA 1555 No cascaded deletes
Delete from/update to parent table with referential integrity constraint on child table and *no index on FK column in child table and with ON DELETE CASCADE in the FK constraint*	Mode 5 Mode 3 Mode 6	TM(S) lock on child TM(RX) lock on parent TX lock on RBS TX slot	Mode 3,4,5,6 on child Allows Select for Update only No Share locks due to referential integrity No ORA 1555 No cascaded deletes from other parent tables that this child references Mode 4,5,6 Select for update or DML on same rows on parent
Delete from/update to parent table with referential integrity constraint on child table and *with index on FK column in child table and with ON DELETE in the FK constraint*	Mode 3 Mode 3 Mode 6	TM(RX) lock on parent TX lock on RBS TX slot	Mode 4,5,6 and any selects for update or DML on same rows against parent Mode 4,5,6 and any selects for update or DML or other cascaded deletes on same rows in child that are current targets for cascaded deletes.

Type of Request	Lock Mode	Lock Target	Conflicts/Notes
Lock table in Exclusive mode	Mode 6	TM(X) lock on table	Mode 2,3,4,5,6 Selects only; no DDL Most restrictive lock mode
Lock table partition in Exclusive mode	Mode 3 Mode 6	TM(X) lock on table TM(X) lock on table partition	Mode 2,3,4,5,6 on the same partition Mode 5 on any partition No exclusive DDL Most restrictive lock mode on partition
Drop, Truncate, Create Table and Create Index DDL	Mode 6 No wait	TM(X) lock on table	Mode 2,3,4,5,6 Selects only; No DDL DDL fails if any other lock mode on table due to no wait
Drop, Truncate, ADD Partition DDL	Mode 3 Mode 6 No wait	TM(X) lock on table TM(X) lock on table partition	Mode 2,3,4,5,6 on the same partition Mode 5 on any partition DDL fails if any other lock mode on table partition due to no wait

11

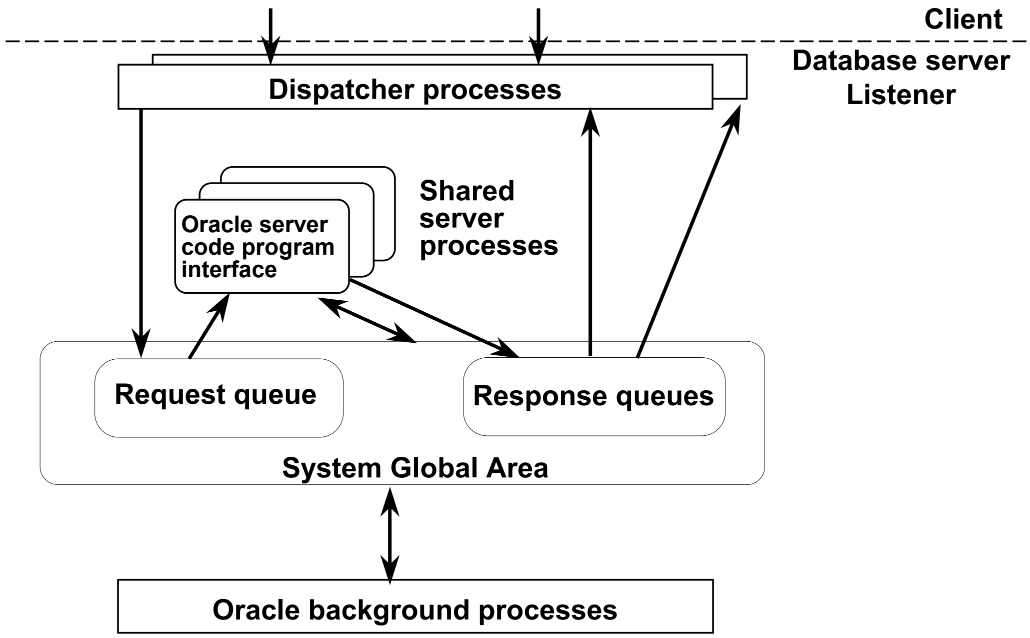
Tuning the Oracle Shared Server

Objectives

After completing this lesson, you should be able to do the following:

- Identify issues associated with managing users in an Oracle Shared Server environment
- Diagnose and resolve performance issues with Oracle Shared Server processes
- Configure the Oracle Shared Server environment to optimize performance

Overview



Oracle Shared Server

Oracle Shared Server is designed to allow multiple user sessions to share a limited number of servers.

In a dedicated server environment, each user process is allocated a server process. This server process, in turn, may not be fully used by a user process, due to idle time and inactivity. However, the allocation of this server process consumes both memory and CPU resources.

When using the Oracle Shared Server, on the other hand, user processes are dynamically allocated to a server process that can be shared across many user processes. The dispatcher process receives a request from a user process and places it in the request queue, so that a shared server can process it and return the results to the response queue for the dispatcher. The dispatcher process returns the results to the user after the results are placed in the response queue.

A useful example is the reservations process for Oracle courses. A customer calls to inquire about a booking. The reservations clerk has a window in which to query course availability, but needs to talk to the customer for a few minutes first. The query is then sent to the database. After a bit more conversation, the customer decides which course to book, and the clerk commits the booking. In a ten-minute conversation, the dedicated server has been idle 99% of the time.

Oracle Shared Server Characteristics

- Enables users to share processes
- Supports Oracle Net functionality
- Increases the number of concurrent users
- Useful on servers with remote clients
- CPU overhead could possibly increase for each individual user request

Characteristics

The Oracle Shared Server provides a way for users to share processes and is meant for scaling up the number of connections that Oracle server can handle concurrently.

The Oracle Shared Server supports both multiplexing and pooling for user connections by means of Connection Manager and Connection Pooling.

Without shared servers, each remote client user needs a dedicated server process to access Oracle server files and memory structures. In an interactive application, where users spend much of their time talking to customers, these dedicated servers are mainly idle. They have work to do only when the user sends a query or a change to the database.

With Oracle Shared Server, multiple users can share dispatcher processes, which access the Oracle Server for them. Oracle uses shared servers to process the SQL statements passed in by the dispatchers.

Oracle Shared Server is useful on:

- Systems that have a high overhead for a dedicated server
- Machines that are approaching limits on resources

There is no advantage in using shared servers for database-intensive work. Heavy or batch users should have dedicated servers.

Monitoring Dispatchers

- Use the following dynamic views:
  - `V$SHARED_SERVER_MONITOR`
  - `V$DISPATCHER`
  - `V$DISPATCHER_RATE`
- Identify contention for dispatchers by checking:
  - Busy rates
  - Dispatcher waiting time
- Check for dispatcher contention
- Add or remove dispatchers while the database is open

Identifying Contention for Dispatcher Usage

In general, dispatchers are not very busy because their task is completed quickly. However, because dispatchers are not started or stopped automatically by the server, you should monitor their usage.

You can find the limit of connections and sessions, and the current usage of sessions from the `V$SHARED_SERVER_MONITOR` view. The value for `MAXIMUM_CONNECTIONS` defaults to the value of the `SESSIONS` parameter, if `SESSIONS` is set lower than the actual limit for a dispatcher.

Query the `V$DISPATCHER` view to determine the usage for selected dispatcher processes. You identify contention for dispatchers by checking:

```
SQL> SELECT network "Protocol", status “Status”,
  2> SUM(OWNED) "Clients“,
  3> SUM(busy)*100/(SUM(busy)+SUM(idle)) "Busy Rate“
  4> FROM V$dispatcher  GROUP BY network;
```

The query returns the percentage of time the dispatcher processes of each protocol are busy. The Clients column indicates the number of clients connected using the protocol.

Guideline: In choosing the number of dispatchers, you should consider the number of clients for a dispatcher as well as the busy rate. If the busy rate of a dispatcher is over 50%, you may consider increasing the number dispatcher. You are also likely to find dispatchers for some protocols being idle. You should consider reducing the number of such idle dispatchers.

Checking Whether Users Wait for Dispatchers

You should check whether users sessions are waiting for dispatchers by executing the following query at frequent intervals:

```
SELECT DECODE(SUM(totalq),0,'No Responses',
       SUM(wait)/SUM(totalq)) "Average wait time"
FROM   v$queue q, v$dispatcher d
WHERE  q.type    = 'DISPATCHER'
AND    q.paddr   = d.paddr;
```

The average wait time is expressed in hundredths of a second. A steadily increasing value indicates a problem. You may want to start up more dispatchers at once if you run the query twice or more and wait times seem to be increasing.

To add or remove dispatchers, use the following command:

```
ALTER SYSTEM SET dispatchers = 'protocol, number';
```

Allocating more dispatchers does not have any immediate effect, because users are bound to the same dispatcher until they log off. Only new connections can make use of the new dispatchers.

You can also query the `V$DISPATCHER_RATE` view to analyze contention. The `V$DISPATCHER_RATE` view contains columns grouped under `CUR`, `AVG`, and `MAX`. Compare `CUR` and `MAX` values.

If the performance of connections using shared servers are not satisfactory, and the `CUR` values are close or equal to the `MAX` values, then you should consider increasing the number of dispatchers for the corresponding protocols.

If, on the other hand, you find that the performance is satisfactory and the CUR values are substantially below the MAX values, you have configured too many dispatchers. Consider reducing the number of dispatchers.

You can use the `ALTER SYSTEM SET DISPATCHERS` command to increase or decrease the number of dispatchers.

Monitoring Shared Servers

- **Oracle Shared Servers are started up dynamically.**
- **However, you should monitor the shared servers by:**
  - **Checking for shared server process contention**
  - **Adding or removing idle shared servers**

Monitoring Shared Servers

Oracle Shared Server processes are started dynamically by the PMON background process when the existing shared servers are busy providing the value of MAX_SHARED_SERVERS is higher than the number of currently available servers. Accordingly, when shared servers are idle, they are removed by PMON till the number reaches SHARED_SERVERS. Thus you do not need to monitor shared servers as closely as you should the dispatchers.

However, you may have started up more shared servers than you need by specifying a higher than required value for the SHARED_SERVERS parameter. Because Oracle9*i* Server does not terminate the unused shared server processes if they number less than that specified in SHARED_SERVERS even if they are idle, such unused shared server processes may overload the system.

You may also want to find out whether the number of servers is approaching the value of MAX_SHARED_SERVERS or (even worse) is bringing the number of processes close to the value of the PROCESSES parameter.

You can add or remove shared servers by using the following command:

```
ALTER SYSTEM SET SHARED_SERVERS = number;
```

Monitoring Shared Servers (continued)

You should query the V$SHARED_SERVER view to obtain information on the current status of shared servers:

```
SELECT name, requests, busy*100/(busy+idle) “BUSY %”, status
FROM v$shared_server
WHERE status != 'QUIT';
```

You can also determine if there is contention for shared server processes by querying the `wait` and `totalq` columns of the V$QUEUE dynamic performance view.

Monitor these statistics occasionally while your application is running:

```
SELECT DECODE( totalq, 0, 'No Requests',
wait/totalq || ' hundredths of seconds')
"Average Wait Time Per Requests"
FROM v$queue
WHERE type = 'COMMON';
```

This query returns the total wait time for all requests and total number of requests for the request queue. The result of this query looks like this:

```
Average Wait Time per Request
-----------------------------
.090909 hundredths of seconds
```

Monitoring Process Usage

The `V$CIRCUIT` view displays:

- **Server address**
- **Dispatcher address**
- **User session address**

Checking Shared Connections

If a user has a problem or a process seems to be doing too much work, you may need to check on current users with shared connections.

Use the `server` column of `V$SESSION` view to ascertain the type of connections that the sessions are using.

```
SELECT SID,SERIAL#, USERNAME, SERVER
FROM   V$SESSION;
```

You can query current dispatcher and server use with the `V$CIRCUIT` view that gives you server and dispatcher addresses, and the session address for the user.

You also need to check with `V$DISPATCHER`, `V$SHARED_SERVER`, and `V$SESSION` views for the corresponding values in the `name` and `username` columns.

Shared Servers and Memory Usage

- Some user information goes into the shared pool.
- To reduce the load on the Shared Pool set a Large Pool
- Overall memory demand is lower when using shared servers.
- Shared servers use the user global area (UGA) for sorts.

Using Oracle Shared Servers as Search Servers

If you decide to use the shared server, some user session information, called the user global area (UGA), is stored in the shared pool, while the data components of the session can be held in the large pool. If the large pool is not configured, then all the information is stored in the shared pool.

It is desirable to set a large pool, if using shared servers, this is done by setting the `LARGE_POOL_SIZE` parameter.

The overall memory demand on the system decreases when using shared servers.

Shared servers use the UGA for sorts, so if you are using a shared server, you should set `SORT_AREA_RETAINED_SIZE` smaller than `SORT_AREA_SIZE.` See the earlier lesson of sorting for details regarding memory and sorting.

Troubleshooting

Possible causes of problems with the shared server include the following:

- **The database listener is not running.**
- **The Oracle Shared Server initialization parameters are set incorrectly.**
- **The dispatcher process has been killed.**
- **The DBA does not have a dedicated connection.**
- **The PROCESSES parameter is too low.**

11-11

Troubleshooting

Troubleshooting the Oracle Shared Server environment is a key DBA function. Some common problems include the following:

- If the Oracle Net listener is not running, all attempts at shared connections fail. You need to bring it up whenever the machine is started up.
- Any Oracle Net configuration error gives a `TNS_error` message when you try to establish a shared connection.
- It is always bad practice to kill a user's server process at the operating system level (use the `ALTER SYSTEM KILL SESSION` command instead). But if a user is connected through a dispatcher, it is even more dangerous, because killing the dispatcher may affect many other users as well.
- You cannot perform privileged operations such as `STARTUP` and `SHUTDOWN` using a shared server. Make sure you have a dedicated connection for yourself as DBA.
- Your servers and dispatchers count as background processes for the instance. Be careful that your setting of `PROCESSES` allows for all possible servers and dispatchers, or new users may not be able to log in. Setting `MAX_SERVERS` and `MAX_DISPATCHERS` can act as a useful ceiling.
- If the parameters (`INSTANCE_NAME`, `SERVICE_NAMES`, or `DB_DOMAIN`) are not set or if they are set to an incorrect value, then the automatic instance registration will not work.

Obtaining Dictionary Information

Dynamic performance views:

V$CIRCUIT

V$DISPATCHER

V$DISPATCHER_RATE

V$QUEUE

V$SHARED_SERVER_MONITOR

V$SESSION

V$SHARED_SERVER

11-12

ORACLE

Using Dynamic Performance Views

You can use different dynamic performance views to obtain information about the Oracle Shared Server environment. Use the Oracle8*i* reference manual to obtain details about each of the following views:

- `V$CIRCUIT`: Contains information about user connections to the database
- `V$DISPATCHER`: Provides information on the dispatcher processes
- `V$DISPATCHER_RATE`: Provides rate statistics for the dispatcher processes
- `V$QUEUE`: Contains information on the multithread message queues
- `V$SHARED_SERVER_MONITOR`: Contains information for tuning the Oracle Shared Server
- `V$SESSION`: Lists session information for each current session
- `V$SHARED_SERVER`: Contains information about the shared server processes

A query to report the dispatcher, session, and process mapping using shared servers:

```
SELECT  d.network network, d.name disp, s.username oracle_user,
         s.sid sid,s.serial# serial#, p.username os_user,
         p.terminal terminal, s.program program
FROM     v$dispatcher d, v$circuit c, v$session s, v$process p
WHERE    d.paddr = c.dispatcher(+)
AND      c.saddr = s.saddr(+)
AND      s.paddr = p.addr (+)
order by d.network, d.name, s.username
```

Summary

In this lesson, you should have learned how to:

- Describe the Oracle Shared Server as a resource-sharing configuration
- List some situations in which it is appropriate to use the Oracle Shared Server
- Monitor dispatcher and server usage
- Troubleshoot Oracle Shared Server configuration

12

SQL Statement Tuning

Objectives

After completing this lesson, you should be able to do the following:

- Describe how the optimizer is used
- Explain the concept of plan stability
- Explain the use of stored outlines
- Describe how hints are used
- Use SQL Trace and TKPROF
- Collect statistics on indexes and tables
- Describe the use of histograms
- Copy statistics between databases

ORACLE

12-2

Optimizer Modes

In Oracle9*i*, two optimizer modes can be chosen:

- **Rule-based:**
  - **Uses a ranking system**
  - **Syntax- and data dictionary-driven**
- **Cost-based:**
  - **Chooses the path with lowest cost**
  - **Statistics-driven**

Optimizer Modes

Rule-Based Optimization

In this mode, the server process chooses its access path to the data by examining the query. This optimizer has a complete set of rules for ranking access paths. Experienced Oracle developers often have a good understanding of these rules, and tune their SQL code accordingly. The rule-based optimizer (RBO) is syntax-driven, in that it uses the statement syntax in combination with data dictionary information about the data structures to determine which execution plan to use. This optimizer mode is supported for backward compatibility with earlier releases of the Oracle server.

Cost-Based Optimization

In this mode, the optimizer examines each statement and identifies all possible access paths to the data. It then calculates the resource cost of each access path and chooses the least expensive one. The costing is based mainly on the number of logical reads. The cost-based optimizer (CBO) is statistics-driven in that it uses statistics generated for the objects involved in the SQL statement to determine the most effective execution plan. The cost-based optimizer is used if any object in the SQL statement has had statistics generated for it. You should use this optimizer mode for new applications, particularly if they use the Parallel Query feature, bitmap indexes or bitmap join indexes.

Note: For additional information about the ranking used by the rule-based optimizer, refer to the *Oracle9i Database Concepts* manual.

Setting the Optimizer Mode

- **At the instance level:**
  - `optimizer_mode = {choose|rule|first_rows|first_rows_n|all_rows}`
- **At the session level:**
  - `alter session set optimizer_mode = {choose|rule|first_rows|first_rows_n|all_rows}`
- **At the statement level:**
  - **Using hints**

12-4

Setting the Optimizer Mode

The optimizer mode can be set at the:

- Instance level by using the `OPTIMIZER_MODE` parameter
- Session level, by using the `ALTER SESSION` command
- Statement level, by using hints

The DBA is responsible for setting the `OPTIMIZER_MODE` parameter at the instance level, because this requires restarting the instance. Typically, application developers can set the `OPTIMIZER_MODE` at the session level, as well as use hints in SQL statements.

The `OPTIMIZER_MODE` Parameter

The default value is `CHOOSE`. This means that the optimizer uses the cost-based mode (`ALL_ROWS`) if statistics are available for at least one of the tables involved. Otherwise, it uses rule-based optimization.

Note: If any table involved has a degree of parallellization greater than 1, or a parallel hint, the default behavior for the statement is cost-based optimization.

The other possible values are `RULE`, `FIRST_ROWS`, `FIRST_ROWS_n`, and `ALL_ROWS`. The first one forces rule-based optimization regardless of the existence of any statistics. The last two represent different ways of using cost-based optimization. `FIRST_ROWS` minimizes immediate response time (possibly at the expense of overall response time), and `FIRST_ROWS_n` minimizes immediate response time for the first *n* rows (possibly at the expense of overall response time). The value of *n* can be 1, 10, 100, or 1000. `ALL_ROWS` minimizes total response time (throughput).

The `OPTIMIZER_MODE` Option at the Session Level

Developers can set this option using the ALTER SESSION command.

```
SQL> ALTER SESSION SET OPTIMIZER_MODE = value
```

Note: For backward compatibility, the `OPTIMIZER_GOAL` option of the ALTER SESSION command is still supported as an alternative for the `OPTIMIZER_MODE` option.

Optimizer Hints

You can code hints into a statement, as shown below:

```
SQL>  SELECT /*+ FIRST_ROWS */
  2          *
  3  FROM  hr.employees;
```

Hints which influence the optimizer mode include PARALLEL, `RULE`, `FIRST_ROWS`, `FIRST_ROWS_n` and `ALL_ROWS`.

Note: Refer to the *Oracle9i Performance Guide and Reference* manual for a listing of all available hints.

Precedence Rules

Hints always override session level-settings, and session-level settings always override instance-level settings.

Optimizer Plan Stability

- Users can stabilize execution plans, in order to force applications to use a desired SQL access path.
- A consistent execution path is thereby maintained through database changes.
- This is done by creating a *stored outline* consisting of hints.
- The `OPTIMIZER_FEATURES_ENABLE` parameter enables the optimizer to keep CBO features of previous versions.

ORACLE

12-6

Optimizer Plan Stability

For every statement, the optimizer prepares a tree of operations called an execution plan that defines the order and methods of operation that the server follows to execute the statement.

Because the optimizer may have incomplete information, sometimes the best possible plan is not chosen. In these cases, you may find it worthwhile to influence the optimizer's plan selection by rewriting the SQL statement, by using hints, or by using other tuning techniques. Once satisfied, you may want to ensure that the same tuned plan is generated whenever the same statement is recompiled, even when factors that affect optimization may have changed.

Oracle9*i* provides the user with a means of stabilizing execution plans across Oracle releases, database changes, or other factors that normally cause an execution plan to change. You can create a stored outline containing a set of hints used by the optimizer to create an execution plan.

`OPTIMIZER_FEATURES_ENABLE`

This parameter allows a version of the Oracle server to run with the **CBO** features of an earlier version. This parameter should be left as its default, which is the current release. However, if the DBA wants to keep the previous CBO features while performing the upgrade then it can be set. Before doing so refer to the *Oracle9i Performance Guide and Reference*.

Plan Equivalence

- **SQL statement text must match the text in a stored outline.**
- **Plans are maintained through:**
  - **New Oracle versions**
  - **New statistics on objects**
  - **Initialization parameter changes**
  - **Database reorganization**
  - **Schema changes**

SQL Statement Equivalence

Plan stability relies on exact textual matching of queries when determining whether a query has a stored outline. This is the same matching criteria used to determine whether an execution plan in the shared pool can be reused.

Stored outlines rely partially on hints that the optimizer uses to achieve stable execution plans. Therefore, the degree to which plans remain equivalent is dependent on the capabilities of the hints the plans use. The execution steps included in a stored outline include row access methods, join order, join methods, distributed accesses, and view/subquery merging. Distributed access does not include the execution plan on the remote node.

Plan Stability

These plans are maintained through many types of database and instance changes. Therefore, if you develop applications for mass distribution, you can use stored outlines to ensure that all your customers access the same execution plans. For example, if a schema is changed by adding an index, the stored outline can prevent the use of the new index.

Creating Stored Outlines

```
SQL> alter session
  2  set CREATE_STORED_OUTLINES = train;
SQL> select … from … ;
SQL> select … from … ;
```

```
SQL> create or replace OUTLINE co_cl_join
  2  FOR CATEGORY train ON
  3  select co.crs_id, ...
  4  from   courses co
  5  ,      classes cl
  6  where  co.crs_id = cl.crs_id;
```


Creating Stored Outlines

The server can create outlines automatically or you can create them for specific SQL statements. Outlines use the cost-based optimizer, because they rely on hints.

Categories

Stored outlines can be grouped by categories. The same SQL statement can have a stored outline in more than one category. For example, you may want to have an OLTP category and a DSS category. If a category name is omitted, outlines are placed in the DEFAULT category.

`CREATE_STORED_OUTLINES` Parameter

Oracle creates stored outlines automatically for all executed SQL statements when you set `CREATE_STORED_OUTLINES` to `TRUE` or to a category name. When set to `TRUE` , the DEFAULT category is used. You can deactivate the process by setting the parameter to `FALSE`. When this parameter is used, the outline names are also generated automatically.

`CREATE OUTLINE` Command

You can also create stored outlines for a specific statement by using the `CREATE OUTLINE` command. One advantage of this approach is that you can specify a name for the stored outline.

Using Stored Outlines

- **Set the `USE_STORED_OUTLINES` parameter to `TRUE` or to a category name:**

```
SQL> alter session
  2  set USE_STORED_OUTLINES = train;
SQL> select … from … ;
```

- **Both `CREATE_STORED_OUTLINES` and `USE_STORED_OUTLINES` can be set at the instance or session level.**

Using Stored Outlines

If `USE_STORED_OUTLINES` is set to `TRUE`, then outlines from the DEFAULT category are used. If `USE_STORED_OUTLINES` is set to a category name, then the outlines from that category are used. If there is no matching outline in that category but there is one in the DEFAULT category, then that outline is used.

The statement must match the text of the statement in the outline. They are compared using the method for comparing cursors in the shared pool. This means that hints in the outline have to be used in the statement text to cause a match. Values in bind variables do not need to match.

To determine a SQL statement's execution plan, Oracle9*i* uses the following logic:

- The statement is compared to statements in the shared pool for matching text and outline category.
- If no matching statement is found, the data dictionary is queried for a matching outline.
- If a matching outline is found, Oracle9*i* integrates the outline into the statement and creates the execution plan.
- If no outline is found, the statement is executed using normal (non-outline) methods.

If an outline specifies the use of an object that cannot be used (for example, it references an index that no longer exists), the statement will simply not use the hint. To verify that a stored outline is being used, the explain plan for a statement needs to be compared when running with and without `USE_STORED_OUTLINES` set.

Using Private Outlines

Private outlines are:

- **Edited without affecting the running system**
- **Copies of current storage outlines**
- **Controlled using the `USE_PRIVATE_OUTLINES` parameter**

Using Private Outlines

The `USE_PRIVATE_OUTLINES` parameter lets you control the use of private outlines. A private outline is an outline seen only in the current session and whose data resides in the current parsing schema. Changes made to such an outline are not seen by any other session on the system, and applying a private outline to the compilation of a statement can only be done in the current session with the `USE_PRIVATE_OUTLINES` parameter. Only when you explicitly choose to save your edits back to the public area are they seen by the rest of the users. The outline is cloned into the user's schema at the onset of the outline editing session. All subsequent editing operations are performed on that clone until the user is satisfied with the edits and chooses to publicize them. Any editing done by the user does not impact the rest of the user community, which would continue to use the public version of the outline until the edits are explicitly saved. When a private outline is created, an error is returned if the prerequisite outline tables to hold the outline data do not exist in the local schema. These tables can be created using the `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES` procedure.

Prerequisites to Using Private Outlines

When the `USE_PRIVATE_OUTLINES` parameter is enabled and an outlined SQL statement is issued, the optimizer retrieves the outline from the session private area rather than the public area used when `USE_STORED_OUTLINES` is enabled. If no outline exists in the session private area, then the optimizer will not use an outline to compile the statement.

Editing Stored Outlines

Editing and using private outlines:

- **Create the outline tables in the current schema**
- **Copy the selected outline to private outline**
- **Edit the outline stored as a private outline**
- **To use the private outline, set the `USE_PRIVATE_OUTLINE` parameter**
- **To allow public access to the new stored outline, overwrite the stored outline**
- **Reset `USE_PRIVATE_OUTLINE to FALSE`**

Editing Stored Outlines

Assume that you want to edit the DEV01 outline. The steps are as follows:

1. Connect to a schema from which the outlined statement can be executed, and make sure that the `CREATE ANY OUTLINE` and `SELECT` privileges have been granted. Create outline editing tables locally with the `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES` procedure.
2. Clone the outline being edited to the private area by using the following:
   ```
   CREATE PRIVATE OUTLINE p_DEV01 FROM dev01;
   ```
3. Edit the outline, either with the Outline Editor in Enterprise Manager or manually by querying the local `OL$HINTS` tables and performing DML against the appropriate hint tables. To change the join order, use the `DBMS_OUTLN_EDIT.CHANGE_JOIN_POS` procedure.
4. If manually editing the outline, then resynchronize the stored outline definition by using the following so-called identity statement:
   ```
   CREATE PRIVATE OUTLINE p_dev01 FROM PRIVATE p_dev01;
   ```

Editing Stored Outlines (continued)

5. You can also use DBMS_OUTLN_EDIT.REFRESH_PRIVATE_OUTLINE or ALTER SYSTEM FLUSH SHARED_POOL to accomplish the same task as in step 4.
6. Test the edits. Set USE_PRIVATE_OUTLINES=TRUE, and issue the outline statement or run EXPLAIN PLAN on the statement.
7. If you want to preserve these edits for public use, then publicize the edits with the following statement. CREATE OR REPLACE OUTLINE dev01 FROM PRIVATE p_dev01;
8. Disable private outline usage by setting the USE_PRIVATE_OUTLINES parameter to FALSE.

Maintaining Stored Outlines

- Use the `OUTLN_PKG` package to:
  - Drop outlines or categories of outlines
  - Rename categories
- Use the `ALTER OUTLINE` command to:
  - Rename an outline
  - Rebuild an outline
  - Change the category of an outline
- Outlines are stored in the `OUTLN` schema.

Maintaining Stored Outlines

Use procedures in the `OUTLN_PKG` package to manage stored outlines and their categories. These procedures are:

- `Drop_unused`: Drops outlines that have not been used since they were created
- `Drop_by_cat`: Drops outlines assigned to the specific category name
- `Update_by_cat`: Reassigns outlines from one category to another

Outlines can also be managed with the `ALTER/DROP OUTLINE` commands.

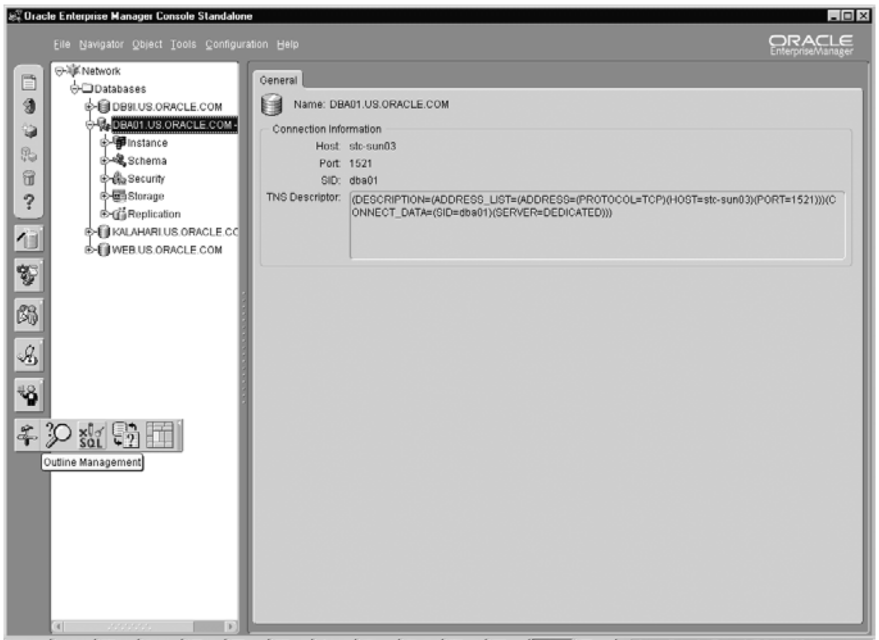
Plans can be exported and imported by exporting the `OUTLN` schema, where all outlines are stored. Outlines can be queried from tables in the schema:

- `OL$`: Outline name, category, creation timestamp, and the text of the statement
- `OL$HINTS`: The hints for the outlines in `OL$`

The equivalent data dictionary views are `DBA_OUTLINES` and `DBA_OUTLINE_HINTS`.

Note: Because the user `OUTLN` is automatically created with the database, its password should be changed.

Enterprise Manager: Maintaining Stored Outlines



ORACLE

12-14

Enterprise Manager: Maintaining Stored Outlines

The Enterprise Manager Console can be used to maintain stored outlines on a database. From this manager stored outlines can be created, dropped, or modified.

Using Hints in a SQL Statement

```
SQL> CREATE index gen_idx on customers
2    (cust_gender);
```

```
SQL> SELECT /*+ INDEX(customers gen_idx)*/
2    cust_last_name, cust_street_address,
3    cust_postal_code
4    FROM sh.customers
5    WHERE UPPER (cust_gender) = 'M';
```

ORACLE

Using Hints in a SQL Statement

You may know information about your data that the optimizer does not know. For example, you may know that a certain index is more selective for certain queries. Based on this information, you may be able to choose a more efficient execution plan than the optimizer. In such a case, use hints to force the optimizer to use the optimal execution plan.

Example

In the above example there is an index on the `cust_gender` column of the customers table. There are very few male customers, therefore the statement runs faster using the `gen_idx` index. In order to force the optimizer to use the index, it is put in a *hint*.

Overview of Diagnostic Tools

- `STATSPACK`
- `EXPLAIN PLAN`
- SQL trace and `TKPROF`
- SQL*Plus autotrace feature
- Oracle SQL Analyze

Overview of Diagnostic Tools

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

- `STATSPACK`: This utility collects information regarding database statistics and SQL statements.
- `EXPLAIN PLAN`: This is executed within a session for a SQL statement.
- `SQL Trace`: This utility provides detailed information regarding the execution of SQL statements.
- `TKPROF`: This is an operating system utility that takes the output from a SQL TRACE session and formats it into a readable format.
- Autotrace: This is a SQL*Plus feature. Autotrace generates an execution plan for a SQL statement and provides statistics relative to the processing of that statement.
- Oracle SQL Analyze: This is part of the Oracle Enterprise Manager Tuning Pack, and it provides a powerful user interface for tuning SQL statements.

SQL Reports in STATSPACK

STATSPACK collects the following regarding SQL statements:

- SQL ordered by gets
- SQL ordered by reads
- SQL ordered by executions
- SQL ordered by parse calls

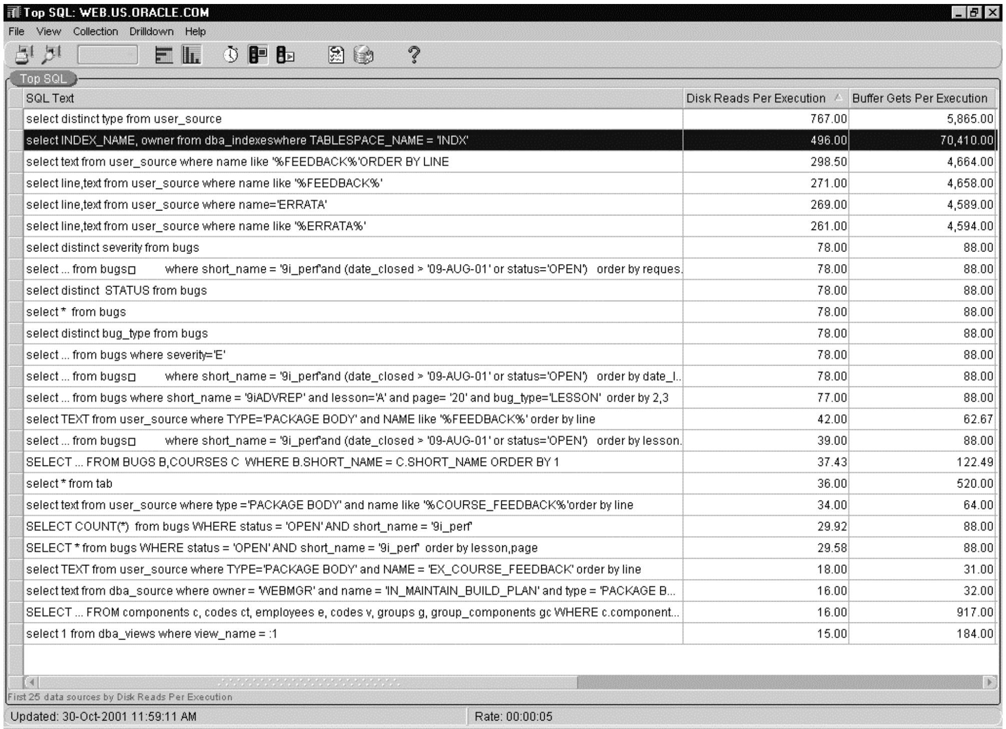
SQL Reports in STATSPACK

STATSPACK gives four different views based on SQL statements stored in the SHARED POOL at the time of either the beginning snapshot or the ending snapshot. The report provides these statements in four different sections. These sections are:

- SQL ordered by gets
- SQL ordered by reads
- SQL ordered by executions
- SQL ordered by parse calls

These views can be examined to see which SQL statements are having the most impact on the performance of the database. These are the best SQL statements to tune because tuning them is most likely to improve performance dramatically.

Performance Manager: Top SQL



Top SQL: WEB.US.ORACLE.COM

File View Collection Drilldown Help

Top SQL

SQL Text	Disk Reads Per Execution	Buffer Gets Per Execution
select distinct type from user_source	767.00	5,865.00
select INDEX_NAME, owner from dba_indexeswhere TABLESPACE_NAME = 'INDX'	496.00	70,410.00
select text from user_source where name like '%FEEDBACK%'ORDER BY LINE	298.50	4,664.00
select line,text from user_source where name like '%FEEDBACK%'	271.00	4,658.00
select line,text from user_source where name='ERRATA'	269.00	4,589.00
select line,text from user_source where name like '%ERRATA%'	261.00	4,594.00
select distinct severity from bugs	78.00	88.00
select ... from bugs□ where short_name = '9i_perf'and (date_closed > '09-AUG-01' or status='OPEN') order by reques.	78.00	88.00
select distinct STATUS from bugs	78.00	88.00
select * from bugs	78.00	88.00
select distinct bug_type from bugs	78.00	88.00
select ... from bugs where severity='E'	78.00	88.00
select ... from bugs□ where short_name = '9i_perf'and (date_closed > '09-AUG-01' or status='OPEN') order by date_l..	78.00	88.00
select ... from bugs where short_name = '9iADVREP' and lesson='A' and page= '20' and bug_type='LESSON' order by 2,3	77.00	88.00
select TEXT from user_source where TYPE='PACKAGE BODY' and NAME like '%FEEDBACK%' order by line	42.00	62.67
select ... from bugs□ where short_name = '9i_perf'and (date_closed > '09-AUG-01' or status='OPEN') order by lesson.	39.00	88.00
SELECT ... FROM BUGS B,COURSES C WHERE B.SHORT_NAME = C.SHORT_NAME ORDER BY 1	37.43	122.49
select * from tab	36.00	520.00
select text from user_source where type ='PACKAGE BODY' and name like '%COURSE_FEEDBACK%'order by line	34.00	64.00
SELECT COUNT(*) from bugs WHERE status = 'OPEN' AND short_name = '9i_perf'	29.92	88.00
SELECT * from bugs WHERE status = 'OPEN' AND short_name = '9i_perf' order by lesson,page	29.58	88.00
select TEXT from user_source where TYPE='PACKAGE BODY' and NAME = 'EX_COURSE_FEEDBACK' order by line	18.00	31.00
select text from dba_source where owner = 'WEBMGR' and name = 'IN_MAINTAIN_BUILD_PLAN' and type = 'PACKAGE B...	16.00	32.00
SELECT ... FROM components c, codes ct, employees e, codes v, groups g, group_components gc WHERE c.component...	16.00	917.00
select 1 from dba_views where view_name = :1	15.00	184.00

First 25 data sources by Disk Reads Per Execution

Updated: 30-Oct-2001 11:59:11 AM | Rate: 00:00:05

Performance Manager: Top SQL

In order to determine which SQL statement to tune, use Top SQL. Top SQL can sort the SQL statements so that the DBA can determine the SQL statement that utilizes the most resources, which is going to be an ideal candidate for tuning because it would produce the highest return.

EXPLAIN PLAN

- Can be used without tracing
- Needs the PLAN_TABLE table utlxplan.sql
- Create the explain plan:

```
SQL> Explain plan for
  2  select last_name from hr.employees;
```

- Query plan_table to display the execution plans:
  - Query PLAN_TABLE directly
  - Use script utlxpls.sql (Hide Parallel Query information)
  - Use script utlxplp.sql (Show parallel Query information)

ORACLE

12-19

The EXPLAIN PLAN Statement

You can use the EXPLAIN PLAN statement in SQL*Plus without using tracing. You need to create a table called plan_table using the supplied utlxplan.sql script. The useful columns for most purposes are operation, options, and object_name.

To explain the plan for a query, use the following syntax:

```
EXPLAIN PLAN [SET STATEMENT_ID = '...'] [INTO
my_plan_table]
FOR SELECT ….
```

Then query plan_table to check the execution plan. Plan_table shows you how the statement would be executed if you chose to run it at that moment. Remember that if you make changes before running the statement (creating an index, for example), the actual execution may be different. Also, if you do not use a STATEMENT_ID in the EXPLAIN PLAN statement, you may want to truncate the PLAN_TABLE prior to generating another execution plan. The STATEMENT_ID provides an easy method of marking a particular statement in the PLAN_TABLE, especially when there could be many versions of the same statement.

Querying PLAN_TABLE

Plan_table can be queried directly, or you can run either utlxpls.sql or utlxplp.sql (depending on whether Parallel Query statistics are required). These scripts show the most commonly selected columns of plan_table.

Note: For additional information on the columns in the plan_table, see the *Oracle9i Performance Guide and Reference* manual.

Using SQL Trace and TKPROF

To use SQL trace and TKPROF:

- **Set the initialization parameters.**
- **Alter session set `SQL_Trace = true`**
- **Run the application.**
- **Alter session set `SQL_Trace = false`**
- **Format the trace file with TKPROF.**
- **Interpret the output.**

Using SQL Trace and TKPROF

A particular sequence of steps is necessary to properly diagnose SQL statement performance with SQL Trace and TKPROF:

- The first step is to ensure appropriate initialization parameters. These can be set at the instance level; some parameters can also be set at the session level.
- SQL Trace must be invoked at either the instance or session level. Generally, it is better if it is invoked at the session level.
- Run the application or SQL statement you want to diagnose.
- Turn off SQL Trace. This is necessary to close the trace file properly at the operating system level.
- Use TKPROF to format the trace file generated during the trace session. Unless the output file is formatted, it is very difficult to interpret the results.
- Use the output from TKPROF to diagnose the performance of the SQL statement.

Initialization Parameters

Two parameters in the `init.ora` file control the size and destination of the output file from the SQL trace facility:

```
max_dump_file_size = n
```

This parameter is measured in bytes if K or M is specified, otherwise the number represents operating system blocks. The default value is 10,000 operating system blocks.

When a trace file exceeds the size defined by the parameter value, the following message appears at the end of the file: `*** Trace file full ***`

The following parameter determines the trace file destination:

```
user_dump_dest = directory
```

You must set a third parameter to get timing information:

```
timed_statistics = TRUE
```

The timing statistics have a resolution of one one-hundredth of a second.

The `TIMED_STATISTICS` parameter can also be set dynamically at the session level, using the ALTER SESSION command.

Enabling and Disabling SQL Trace

- **At the instance level:**
  `SQL_TRACE = {TRUE|FALSE}`
- **At the session level:**

```
SQL> alter session set SQL_TRACE = {true|false};

SQL> execute DBMS_SESSION.SET_SQL_TRACE
  2          ({true|false});
SQL> execute DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION
  2          (session_id, serial_id, {true|false});
```

ORACLE

Enabling and Disabling SQL Trace

SQL Trace can be enabled or disabled using different methods at either the instance or the session level.

Instance Level

Setting the `SQL_TRACE` parameter at the instance level is one way to enable tracing. However, it requires that the instance be shut down, then restarted when tracing is no longer needed. This method also imposes a significant performance cost, because all sessions for the instance are traced.

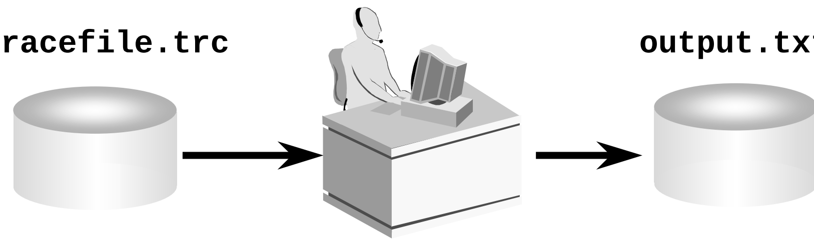
Session Level

Session-level tracing results in less cost to overall performance, because specific sessions can be traced. Three methods for enabling or disabling SQL Trace are:

- Using the `ALTER SESSION` command, which results in tracing for the duration of the session or until the value is set to `FALSE`
- Using the `DBMS_SESSION.SET_SQL_TRACE` procedure for the session
- Using the `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION` procedure to enable tracing in a session, other than the current one.

Formatting the Trace File with TKPROF

```
$ tkprof tracefile.trc output.txt [options]
```



Formatting the Trace File with TKPROF

Use TKPROF to format the trace file into a readable output:

tkprof tracefile outputfile [sort=*option*] [print=*n*]
[explain=*username/password*] [insert=*filename*] [sys=NO]
[record=*filename*] [table=*schema.tablename*]

The trace file is created in the directory specified by the USER_DUMP_DEST parameter and the output is placed in the directory specified by the output file name.

SQL Trace also gathers statistics for recursive SQL statements. You cannot directly affect the amount of recursive SQL that the server executes, so these figures are very useful in themselves. Use the SYS=NO option of TKPROF to suppress the output of these figures.

When you specify the EXPLAIN parameter, TKPROF logs on to the database with the username and password provided. It then works out the access path for each SQL statement traced and includes that in the output. Because TKPROF logs on to the database, it uses the information available at the time that TKPROF is run, not at the time the trace statistics were produced. This could make a difference if, for example, an index has been created or dropped since tracing the statement.

TKPROF also reports library cache misses. This indicates the number of times that the statement was not found in the library cache.

TKPROF Options

Option	Description
TRACEFILE	The name of the trace output file
OUTPUTFILE	The name of the formatted file
SORT=*option*	The order in which to sort the statements
PRINT=*n*	Store the first *n* statements in the output file
EXPLAIN=*user/password*	Run EXPLAIN PLAN in the specified username
INSERT=*filename*	Causes the output to be formatted as a series of SQL INSERT statements in a script file
SYS=NO	Ignore recursive SQL statements run as user `sys`
AGGREGATE=[Y/N]	If you specify AGGREGATE = NO, TKPROF does not aggregate multiple users of the same SQL text
RECORD=*filename*	Record statements found in the trace file
TABLE=*scheme.tablename*	Put execution plan into specified table (rather than the default of plan_table)

You can enter tkprof at the operating system to get a listing of all the available options and output.

Note: The sort options are the following:

Sort Option	Description
Prscnt, execnt, fchcnt	Number of times parse, execute, and fetch were called
prscpu, execpu, fchcpu	CPU time parsing, executing, and fetching
prsela, exela, fchela	Elapsed time parsing, executing, and fetching
prsdsk, exedsk, fchdsk	Number of disk reads during parse, execute, and fetch
prsqry, exeqry, fchqry	Number of buffers for consistent read during parse, execute, fetch
prscu, execu, fchcu	Number of buffers for current read during parse, execute, fetch
prsmis, exemis	Number of misses in library cache during parse, and execute
exerow, fchrow	Number of rows processed during execute, and fetch
userid	User ID of user who parsed the cursor

TKPROF Statistics

- Count: Number of execution calls
- CPU: CPU seconds used
- Elapsed: Total elapsed time
- Disk: Physical reads
- Query: Logical reads for consistent read
- Current: Logical reads in current mode
- Rows: Rows processed

ORACLE

TKPROF Statistics

TKPROF will collect the following statistics:

- Count: The number of times the statement was parsed or executed and the number of fetch calls issued for the statement
- CPU: Processing time for each phase, in seconds. If the statement was found in the shared pool, or if the parse took less than 1/100 of a second, this value will be 0.
- Elapsed: Elapsed time, in seconds (this is not usually very helpful, because other processes affect elapsed time)
- Disk: Physical data blocks read from the database files (usually the statistic is quite low if the data was buffered)
- Query: Logical buffers retrieved for consistent read (usually for SELECT statements)
- Current: Logical buffers retrieved in current mode (usually for DML statements)
- Rows: Rows processed by the outer statement (for SELECT statements, this is shown for the fetch phase; for DML statements, it is shown for the execute phase)

The sum of Query and Current is the total number of logical buffers accessed.

SQL*Plus AUTOTRACE

- **Create the `PLAN_TABLE` table**
- **Create and grant the `Plustrace` role**

```
SQL> @$ORACLE_HOME/sqlplus/admin/plustrce.sql
SQL> grant plustrace to scott;
```

`Autotrace` syntax:

```
set autotrace [ off | on | traceonly ]
              [ explain | statistics ]
```


SQL*Plus AUTOTRACE

SQL*Plus AUTOTRACE can be used instead of SQL Trace. The advantage of using AUTOTRACE is that you do not have to format a trace file and it automatically displays the execution plan for the SQL statement.

However, Autotrace does parse and execute the statement, whereas explain-plan only parses the statement.

The steps for using AUTOTRACE are:

1. Create the `Plan_table` table using the `utlxplan.sql` script.
2. Create the Plustrace role by executing the `plustrce.sql` script. This grants `SELECT` privileges on `V$` views to the role and grants the role to the DBA role. Grant the Plustrace role to users who do not have the DBA role.
3. Set Autotrace to the level desired:
   - OFF : Autotrace turned off. This is the default.
   - ON : Includes the optimizer execution path and the SQL statement execution statistics.
   - ON EXPLAIN : Shows only the optimizer execution path.
   - ON STATISTICS : Shows only the SQL statement execution statistics.
   - TRACEONLY : Same as ON, but suppresses the user's query output.

Managing Statistics

- Use the DBMS_STATS package:
  - GATHER_TABLE_STATS
  - GATHER_INDEX_STATS
  - GATHER_SCHEMA_STATS
  - GATHER_DATABASE_STATS
  - GATHER_STALE_STATS

ORACLE

12-27

Managing Statistics

You collect statistics on an object with the DBMS_STATS package command. Although the cost-based optimizer is not sensitive to minor changes in volume or selectivity, you may want to collect new statistics periodically on frequently modified tables to ensure that the optimizer is using recent, accurate information.

The DBMS_STATS package contains several procedures that allows an index, table, schema or database to be analyzed. This package also enables you to gather most of the statistics with a degree of parallelism. For detailed information, refer to the *Oracle9i Supplied Packages Reference* manual.

Statistics Accuracy

COMPUTE: Calculates exact statistics. It performs a full table scan and several calculations. For large tables this operation may take a long time.

ESTIMATE: You estimate statistics with this option. If you use this option with a suitable sample of the data, it is almost as accurate as the COMPUTE option.

DELETE: You clear out statistics with this option. You do not need to use this option before reanalyzing an object, because existing statistics are overwritten.

Statistics Accuracy (continued)

The FOR Clause

The FOR clause offers the following options:

- FOR TABLE: Restricts the statistics collected to table statistics only rather than table and column statistics
- FOR COLUMNS: Restricts the statistics collected to only column statistics for the specified columns, rather than for all columns and attributes
- FOR ALL COLUMNS: Collects column statistics for all columns.
- FOR ALL INDEXED COLUMNS: Collects column statistics for all indexed columns in the table
- FOR ALL [LOCAL] INDEXES: Specifies that all indexes associated with the table will be analyzed. LOCAL specifies that only local index partitions are analyzed.

The SIZE Clause

The SIZE clause specifies the maximum number of histogram buckets. The default value is 75, and the maximum value is 254. Histograms are discussed in more detail later in this lesson.

Table Statistics

- Number of rows
- Number of blocks and empty blocks
- Average available free space
- Number of chained or migrated rows
- Average row length
- Last ANALYZE date and sample size
- Data dictionary view: DBA_TABLES

Number of Blocks and Empty Blocks

Each table maintains a high-water mark in the segment header block. The high-water mark indicates the last block that was ever used for the table. When the Oracle server performs full table scans, it reads all the blocks up to the high-water mark. Note that the high-water mark is not reset when rows are deleted from the table.

The DBMS_SPACE.UNUSED_SPACE procedure can be used to find the high-water mark and the number of blocks above the high-water mark, if analyzing a table is impossible or undesirable.

Examp7le

```
SQL> exec DBMS_SPACE.UNUSED_SPACE('SH', 'ORDERS', -
  2  'TABLE', :total_blocks, :total_bytes,:unused_blocks,-
  3  :unused_bytes, :lastextf, :last_extb, :lastusedblock);
SQL> select num_rows,blocks,empty_blocks
  2  ,      avg_space,avg_row_len, sample_size
  3  from   dba_tables
  4 where  table_name = 'ORDERS' and owner = 'SH';
NUM_ROW SBLOCKS EMPTY_BLOCKS AVG_SPACE AVG_ROW_LEN  SAMPLE_SIZE
------- ------- ------------ --------- -----------  -----------
  15132     434            5       225          48         1064
```

Index Statistics

- Index level (height)
- Number of leaf blocks and distinct keys
- Average number of leaf blocks per key
- Average number of data blocks per key
- Number of index entries
- Clustering factor
- Data dictionary view: **DBA_INDEXES**

12-30

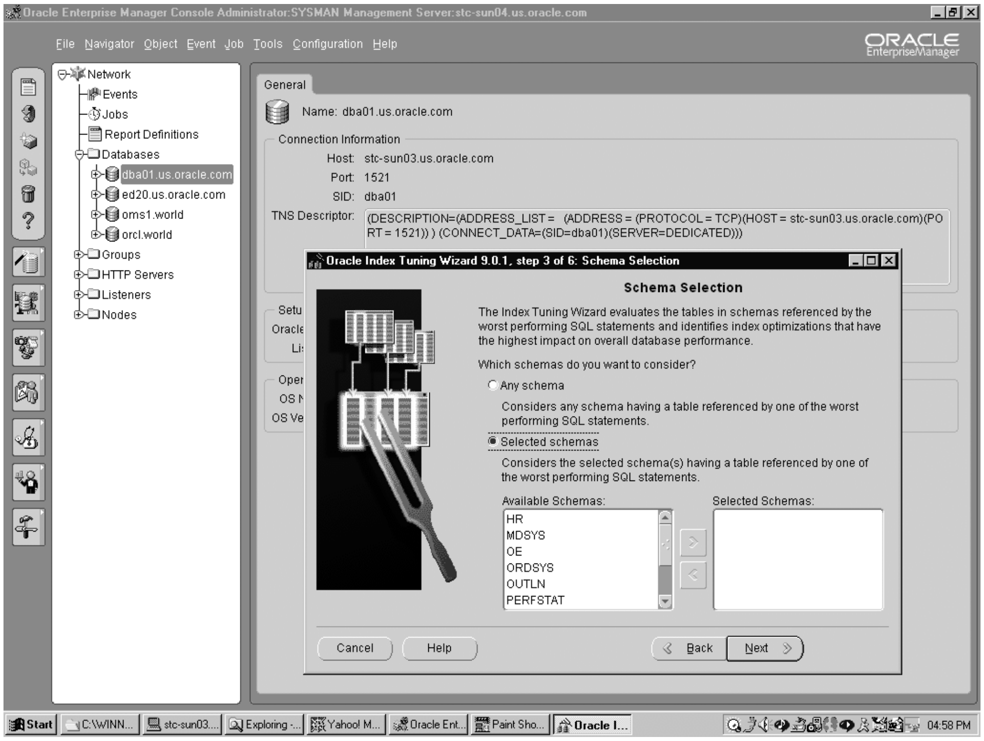
Clustering Factor

The index clustering factor is an important index statistic for the cost-based optimizer to estimate index scan costs. It is an indication of the number of (logical) data block visits needed to retrieve all table rows by means of the index. If the index entries follow the table row order, this value approaches the number of data blocks (each block is visited only once); on the other hand, if the index entries randomly point at different data blocks, the clustering factor could approach the number of rows.

Example

```
SQL> exec DBMS_SPACE.UNUSED_SPACE('SH', 'SALES_PK', 'INDEX', -
  2  :total_blocks, :total_bytes,:unused_blocks, -
  3  :unused_bytes, :lastextf, :last_extb, :lastusedblock);
SQL> select blevel, leaf_blocks, distinct_keys,
  2  clustering_factor
  3  from   dba_indexes
  4  where  index_name = 'SALES_PK' and owner = 'SH';
  BLEVEL  LEAF_BLOCKS   DISTINCT_KEYS     CLUSTERING_FACTOR
 -------  -----------   -------------     -----------------
       2          682           56252                 21349
```

Index Tuning Wizard



ORACLE

12-31

Index Tuning Wizard

From the Enterprise Manager Console start the Index Tuning Wizard in order to get advice on the indexes on your database. You can tune individual schemas or the entire database. Obviously, the more schemas being tuned, the longer the wizard will take in order to make a recommendation.

Column Statistics

- **Number of distinct values**
- **Lowest value, highest value (stored in RAW [binary] format)**
- **Last ANALYZE date and sample size**
- **Data dictionary view: USER_TAB_COL_STATISTICS**

12-32

The USER_TAB_COL_STATISTICS View

The USER_TAB_COL_STATISTICS view offers a relevant subset of the columns displayed by the USER_TAB_COLUMNS view. You can also use the DBA_TAB_COL_STATISTICS view; note however that this view does not contain an owner column, so you get confusing results when your database contains multiple tables with the same name in different schemas.

The num_buckets column shows that regular column statistics are treated as a histogram with one bucket.

Example

```
SQL> select column_name, num_distinct, low_value, high_value
  2 ,     num_nulls, num_buckets
  3 from  user_tab_col_statistics
  4 where table_name = 'EMPLOYEES' and column_name = 'SALARY';
 COLUMN_NAME  NUM_DIST LOW_VALUE  HIGH_VALUE  NUM_NULLS  NUM_BUCKETS
 -----------  -------- ---------  ----------  ---------  -----------
     SALARY       496      C208      C30A62          0            1
```

Histograms

- Histograms describe the data distribution of a particular column in more detail.
- They give better predicate selectivity estimates for unevenly distributed data.
- You create histograms with the `EXECUTE DBMS_STATS.GATHER_TABLE_STATS` procedure.
- Data dictionary views: `DBA_HISTOGRAMS`, `DBA_TAB_HISTOGRAMS`

12-33

Histograms

When using regular column statistics, a minimum value and a maximum value are stored for each column. The cost-based optimizer uses these values to calculate predicate selectivity, assuming an even distribution of the data between those two extreme values. However, if your data is skewed, this assumption may lead to suboptimal plans. You can use histograms to store more detailed information about the data distribution within a column. Statistics are collected for the column and then are stored by partitioning the column values in a number of buckets. Note however that this means additional data dictionary storage requirements. Buckets are height balanced, meaning that each bucket contains approximately the same number of values.

The default number of buckets is 75, and the maximum value is 254.

Generating Histogram Statistics

Histogram statistics are generated by:

```
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS
('HR','EMPLOYEES', METHOD_OPT => 'FOR COLUMNS
SIZE 10 salary');
```


Generate Histogram Statistics

You generate histograms by using the DBMS_STATS package. You can generate histograms for columns of a table or partition. For example, to create a 10-bucket histogram on the SAL column of the emp table, issue the following statement:

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('HR','EMPLOYEES',
  METHOD_OPT => 'FOR COLUMNS SIZE 10 salary');
```

The SIZE keyword declares the maximum number of buckets for the histogram.

Choosing the Number of Buckets for a Histogram

If the number of frequently occurring distinct values in a column is relatively small, then set the number of buckets to be greater than that number. The default number of buckets for a histogram is 75. This value provides an appropriate level of detail for most data distributions. However, because the number of buckets in the histogram and the data distribution both affect a histogram's usefulness, you might need to experiment with different numbers of buckets to obtain optimal results.

Gathering Statistic Estimates

- `DBMS_STATS.AUTO_SAMPLE_SIZE`:
  New `estimate_percent value`
- `Method_opt options`:
  - REPEAT: New histogram with same number of buckets
  - AUTO: New histogram based on data distribution and application workload
  - `SKEWONLY`: New histogram based on data distribution

```
SQL> EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS( -
  2  ownname          => 'OE', -
  3  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, -
  4  method_opt       => 'for all columns size AUTO');
```

ORACLE

12-35

Gathering Statistic Estimations

Because the cost-based approach relies on statistics, users should generate statistics for all tables, clusters, and all indexes accessed by SQL statements before using the cost-based approach. If the size and data distribution of the tables change frequently, then users should regenerate these statistics regularly to make sure that the statistics accurately represent the data in the tables.

Exact statistics computation requires enough space to perform scans and sorts of involved objects. If there is not enough space in memory, then temporary space may be required. Thus, it is also possible to compute only estimations in order to reduce resources needed to gather statistics. The difficulty in computing estimated statistics is to find the best sample size. Some statistics are always computed exactly, such as the number of data blocks currently containing data in a table or the depth of an index from its root block to its leaf blocks. Nevertheless, this is not true for all statistics.

With Oracle9*i*, Oracle Corporation recommends setting the `ESTIMATE_PERCENT` parameter of the `DBMS_STATS` gathering procedures to the new `DBMS_STATS.AUTO_SAMPLE_SIZE` value. This is introduced to maximize performance gains while achieving necessary statistical accuracy avoiding the extremes of collecting inaccurate statistics and wasting valuable time.

Gathering Statistic Estimations (continued)

Some possible values for the `method_opt` parameters of the `dbms_stats` gathering procedures:

- If the size option is set to `REPEAT` and the column currently has a histogram with `b` buckets, the Oracle server attempts to create a new histogram with `b` buckets. If the column has no histogram, no new statistics are gathered. This option is used to maintain the same "class" of statistics (histogram or no-histogram) when looking at new data.
- If the size is set to `AUTO`, the Oracle server decides to create a histogram based on the data distribution AND the way the column is being used by the application. This means that the Oracle server not only looks at non-uniformity in value repetition counts (skew) but also to non-uniformity in range (sparsity). If the application has yet to be run for a sufficient amount of time to capture the workload involving this column, it would be better to use the `SKEWONLY` option temporarily.
- If the size is set to `SKEWONLY`, the Oracle server decides to create a histogram based solely on the data distribution (regardless of how the application uses the column). This option is useful when gathering statistics for the first time (before the workload has had time to run). Using `SKEWONLY` can add quite a bit of overhead to statistics collection, so Oracle recommends that customers use `AUTO` after the application has run for a while.

The example on the previous slide shows you how to collect all table, column, and index statistics for the OE schema where the Oracle server decides what the sampling percentage should be and when histograms are necessary (assuming that the workload has run for a while).

Note: The Oracle server captures workload information for a cursor when it is hard parsed. Information is stored in the SGA and regularly flushed to disk. No access to these memory and disk structures is provided in Oracle9*i*.

Automatic Statistic Collecting

- For the `DBMS_STATS.GATHER_SCHEMA_STATS` procedure set `OPTIONS` to:
  - `GATHER STALE`
  - `GATHER EMPTY`
  - `GATHER AUTO`

```
SQL> EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS( -
  2  ownname  => 'OE', -
  3  options  => 'GATHER AUTO');
```

12-37

Automatic Statistic Collecting

Values for the `OPTIONS` parameter of the `DBMS_STATS.GATHER_SCHEMA_STATISTICS` procedure are:

- GATHER STALE
  Collects statistics only on objects that have the `MONITORING` flag set.
- GATHER EMPTY
  Gathers statistics for all objects that do not have statistics collected.
- GATHER AUTO
  Oracle determines which objects need new statistics, and determines how to gather those statistics.

The goal of this prooption is to simplify statistics gathering at the schema level.

Optimizer Cost Model

- Three columns in PLAN_TABLE are:
  - CPU_COST: Estimated CPU cost of the operation
  - IO_COST: Estimated I/O cost of the operation
  - TEMP_SPACE: Estimated temporary space (in bytes) used by the operation
- Includes CPU usage
- Accounts for the effect of caching
- Accounts for index prefetching

ORACLE

12-38

Optimizer Cost Mode

The role of a query optimizer is to produce the best performing execution plan for a given query. This process includes selecting access paths for single tables, the join order if more than one table is involved in the query, and the join methods.

Currently, you have a choice between using the rule base optimizer (RBO) and the cost-based optimizer (CBO). The CBO uses a cost model to choose between alternative access paths, join order, and join methods, while the RBO uses a set of simple rules.

The CBO compares the cost of several alternatives and selects the one with the lowest cost. In addition to the cost model, the CBO uses a size model in order to derive statistics on intermediate tables; for example, cardinality of the result of a join operation.

The cost model uses statistics on the objects manipulated by the query. Those statistics are produced by using the DBMS_STATS package, and are stored in the database dictionary.

The quality of the execution plan produced by the optimizer is dependent on the accuracy of the cost model. The Oracle8*i* version of this model contains several limitations both in terms of accuracy and completeness. For example, the model assumes independence of columns when computing the selectivity of multiple predicates on different columns and it accounts only for I/O activities.

Optimizer Cost Model Enhancements

The cost model is extended to take into account the following:

- Allow users or developers to convert the cost into more meaningful information. The `PLAN_TABLE` contains three new columns:
  - `CPU_COST`: The CPU cost of the operation as estimated by the optimizer's cost-based approach. For statements that use the rule-based approach, this column is null. The value of this column is proportional to the number of machine cycles required for the operation.
  - `IO_COST`: The I/O cost of the operation as estimated by the optimizer's cost-based approach. For statements that use the rule-based approach, this column is null. The value of this column is proportional to the number of data blocks read by the operation.
  - `TEMP_SPACE`: The temporary space, in bytes, used by the operation as estimated by the optimizer's cost-based approach. For statements that use the rule-based approach, or for operations that don't use any temporary space, this column is null.
- Include CPU usage. CPU usage will be estimated for SQL functions and operators.
- Account for the effect of caching on the performance of nested-loops joins
- Account for index prefetching. Index prefetching consists in fetching multiple leaf blocks in a single IO operation

Gathering System Statistics

- System statistics enable the CBO to use CPU and I/O characteristics.
- System statistics must be gathered on a regular basis; this does not invalidate cached plans.
- Gathering system statistics equals analyzing system activity for a specified period of time.
- New procedures:
  - `DBMS_STATS.GATHER_SYSTEM_STATS`
  - `DBMS_STATS.SET_SYSTEM_STATS`
  - `DBMS_STATS.GET_SYSTEM_STATS`

Gathering System Statistics

System statistics allow the optimizer to consider a system's I/O and CPU performance and utilization. For each candidate plan, the optimizer computes estimates for I/O and CPU costs. It is important to know the system characteristics to pick the most efficient plan with optimal proportion between I/O and CPU cost.

System CPU and I/O characteristics depend on many factors and do not stay constant all the time. Using system statistics management routines, database administrators can capture statistics in the interval of time when the system has the most common workload. For example, database applications can process OLTP transactions during the day and run OLAP reports at night. Administrators can gather statistics for both states and activate appropriate OLTP or OLAP statistics when needed. This allows the optimizer to generate relevant costs with respect to available system resource plans.

When Oracle generates system statistics, it analyzes system activity in a specified period of time. Unlike table, index, or column statistics, Oracle does not invalidate already parsed SQL statements when system statistics get updated. All new SQL statements are parsed using new statistics. Oracle Corporation highly recommends that you gather system statistics.

The `DBMS_STATS.GATHER_SYSTEM_STATS` routine collects system statistics in a user-defined time frame. You can also set system statistics values explicitly using `DBMS_STATS.SET_SYSTEM_STATS`. Use `DBMS_STATS.GET_SYSTEM_STATS` to verify system statistics.

Gathering System Statistics Example

- **First day:**

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLTP');
```

- **First night:**

```
EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS(
interval => 120,
stattab => 'mystats', statid => 'OLAP');
```

- **Subsequent days:**

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLTP');
```

- **Subsequent nights:**

```
EXECUTE DBMS_STATS.IMPORT_SYSTEM_STATS(
stattab => 'mystats', statid => 'OLAP');
```

ORACLE

12-41

Gathering System Statistics Example

The above example shows database applications processing OLTP transactions during the day and running reports at night.

First, system statistics must be collected during the day. In this example, gathering ends after 120 minutes and is stored in the `mystats` table.

Then, system statistics are collected during the night. Gathering ends after 120 minutes and is stored in the `mystats` table.

Generally, you will use the above syntax to gather the system statistics. In that case, you must be sure, before invoking the `GATHER_SYSTEM_STATS` procedure with the `INTERVAL` parameter specified, to activate job processes using a command such as:

```
SQL> alter system set job_queue_processes = 1;
```

Alternatively, you can also invoke the same procedure with different arguments to enable manual gathering instead of using jobs. For syntax information refer to the *Oracle9i Supplied PL/SQL Packages Reference Release 9.0.1*.

If appropriate, you can switch between the statistics gathered. Note that it is possible to automate this process by submitting a job to update the dictionary with appropriate statistics: During the day, a job may import the OLTP statistics for the daytime run, and during the night, another job imports the OLAP statistics for the nighttime run.

Gathering System Statistics Example

- **Start manual system statistics collection in the data dictionary:**

```
SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
  2  gathering_mode => 'START');
```

- **Generate the workload**
- **End system statistics collection:**

```
SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS( -
  2  gathering_mode => 'STOP');
```

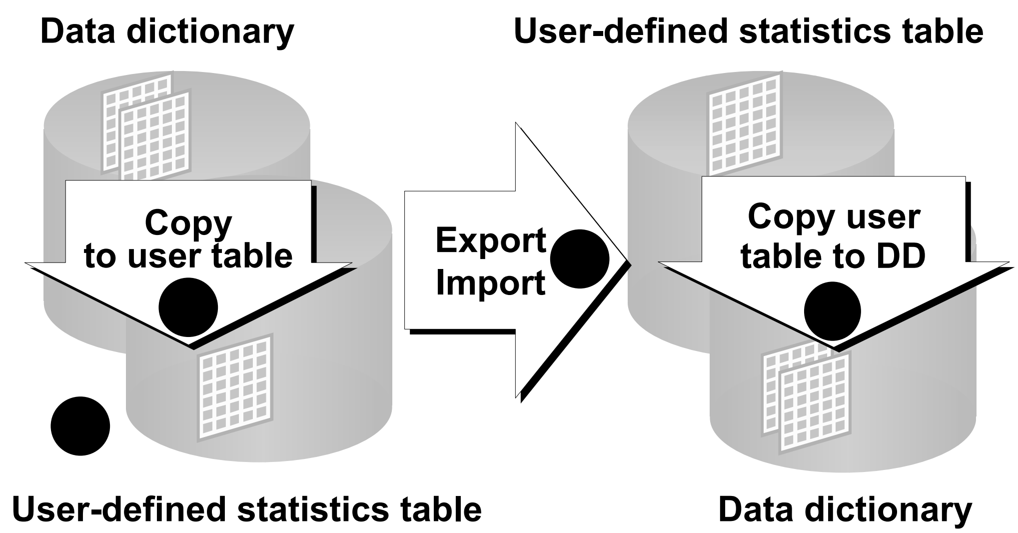

Gathering System Statistics Example (continued)

The previous example shows how to collect system statistics using jobs by using the internal parameter of the DBMS_STATS.GATHER_SYSTEM_STATS procedure. In order to collect system statistics manually , another parameter of this procedure can also be used as shown in the above example.

First, you need to start the system statistics collection, and then you can can end the collecting process at any time after you are sure that a representative workload has been generated on the instance.

The above example collects system statistics directly in the data dictionary.

Copying Statistics Between Databases



12-43

Copying Statistics Between Databases

By using the DBMS_STATS package procedures, you can copy statistics from an Oracle9*i* production to a test database to facilitate tuning. For example, to copy a schema's statistics:

1. Use the DBMS_STATS.CREATE_STAT_TABLE procedure in the production database to create a user-defined statistics table.
2. Use the DBMS_STATS.EXPORT_SCHEMA_STATS procedure in the production database to copy statistics from the data dictionary to the user-defined statistics table from step 1.
3. Use the Export and Import utilities to transfer the statistics to a corresponding user-defined statistics table in the test database.
4. Use the DBMS_STATS.IMPORT_SCHEMA_STATS procedure to import the statistics into the data dictionary in the test database.

The DBMS_STATS package can also be used to back up statistics prior to analyzing objects. The backup can used to:

- Restore old statistics
- Study changes in data characteristics over time

Example: Copying Statistics

Step 1. Create the table to hold the statistics:

```
DBMS_STATS.CREATE_STAT_TABLE
('SH'      /* schema name          */
,'STATS'      /* statistics table name */
,'SAMPLE'      /* tablespace          */
);
```


Example: Copying Statistics

Use the CREATE_STAT_TABLE procedure in the package DBMS_STATS to create the user-defined STATS table that will hold the statistics.

The statistics in this table are not accessible by the Oracle optimizer, and thus cannot be used for generating an explain plan. Likewise, future commands to analyze the data will not update the information held in this table.

Example: Copying Statistics

Step 2. Copy the statistics into the table:

```
DBMS_STATS.EXPORT_TABLE_STATS
('SH'      /* schema name          */
,'SALES'    /* table name           */
, NULL        /* no partitions          */
,'STATS'      /* statistics table name */
,'CRS990601' /* id for statistics     */
, TRUE        /* index statistics      */
);
```

Step 3. Export the STATS table, and then import it into the second database.

Copying Statistics from the Data Dictionary to a User-Defined Table

To copy statistics for a table from the data dictionary to a user-defined statistics table, use one of the following procedures:

- EXPORT_COLUMN_STATS
- EXPORT_INDEX_STATS
- EXPORT_SYSTEM_STATS
- EXPORT_TABLE_STATS
- EXPORT_SCHEMA_STATS
- EXPORT_DATABASE_STATS

The example in the slide is shows an export of the statistics for the COURSES table only.

Example: Copying Statistics

Step 4. Copy the statistics into the data dictionary:

```
DBMS_STATS.IMPORT_TABLE_STATS
('SH'      /* schema name          */
,'SALES'    /* table name           */
, NULL        /* no partitions        */
,'STATS'      /* statistics table name */
,'CRS990601' /* id for statistics     */
, TRUE        /* index statistics     */
);
```


Copying Statistics from a User-Defined Table to the Data Dictionary

To copy statistics for a table from a user-defined statistics table to the data dictionary, use one of the following procedures:

- `IMPORT_COLUMN_STATS`
- `IMPORT_INDEX_STATS`
- `IMPORT_SYSTEM_STATS`
- `IMPORT_TABLE_STATS`
- `IMPORT_SCHEMA_STATS`
- `IMPORT_DATABASE_STATS`

The example the slide is shows an import of the statistics for the `COURSES` table only.

Summary

In this lesson, you should have learned how to:

- Describe how the optimizer is used
- Explain the concept of plan stability
- Explain the use of stored outlines
- Describe how hints are used
- Use SQL Trace and TKPROF
- Collect statistics on indexes and tables
- Describe the use of histograms
- Copy statistics between databases

ORACLE

Practice 12

The objective of this practice is to familiarize you with SQL statement execution plans and to interpret the formatted output of a trace file generated using SQL Trace and the formatted output generated by `TKPROF`. Throughout this practice Enterprise Manager can be used if desired. SQL Worksheet can be used instead of SQ*LPlus, and there are many uses for the Enterprise Manager Console. (Solutions for Enterprise Manager can be found in Appendix B).

1. Connect as hr/hr, and create the `PLAN_TABLE` under the HR schema, if it is not already created, by running `$ORACLE_HOME/rdbms/admin/utlxplan.sql`
   Note: If plan_table already exists and holds rows truncate the table.
2. Set the Optimizer_goal to rule based using the alter session command, and generate the explain plan for the statement `$HOME/STUDENT/LABS/lab12_02.sql`. View the generated plan by querying object_name, operation, optimizer from PLAN_TABLE.
3. Truncate the `PLAN_TABLE`. Change the optimizer_goal to cost based by setting the value to `ALL_ROWS`, and rerun the explain plan for `$HOME/STUDENT/LABS/lab12_02.sql`. Notice that the Optimizer mode, and the explain plan have changed.
   Note: Although exactly the same scripts are being run, due to the different optimizer settings, different explain paths are found. With rule based, one of the rules is to use any index that is on the columns in the where clause. By using cost based optimizer mode, the server has been able to determine that it will be faster to just perform a full table scan, due to the number of rows being returned by the script.
4. Truncate the `PLAN_TABLE`, and set the optimizer goal to rule by using the alter session command. This time generate the explain plan for the script `$HOME/STUDENT/LABS/lab12_04.sql`. Examine the script which is a copy of `$HOME/STUDENT/LABS/lab12_02.sql` except it changes the line "select *" to include a hint /*+ all_rows*/ for the optimizer. View the generated execution plan by querying object_name, operation, optimizer from `PLAN_TABLE`.
5. Exit out of SQLPLUS, change the directory to `$HOME/ADMIN/UDUMP` and delete all the trace files already generated.
6. Connect as sh/sh and enable SQL Trace, using the alter session command, to collect statistics for the script, `$HOME/STUDENT/LABS/lab12_05.sql`. Run the script. After the script has completed, disable the `SQL Trace`. and then format your trace file using TKPROF. Use the options SYS=NO and EXPLAIN= sh/sh. Name the file `myfile.txt.`
7. View the output file `myfile.txt`, and note the CPU, current, and query figures for the fetch phase. Do not spend time analyzing the contents of this file as the only objective here is to become familiar and comfortable with running `TKPROF` and SQL Trace.

8. Connect hr/hr and gather statistics for all objects under the HR schema using the DBMS_STATS package, while saving the current statistics then restore the original statistics.
   a. Connect as HR and create a table to hold statistics in that schema.
   b. Save the current schema statistics into your local statistics table.
   c. Analyze all objects under the HR schema.
   d. Remove all schema statistics from the dictionary and restore the original statistics you saved in step b.