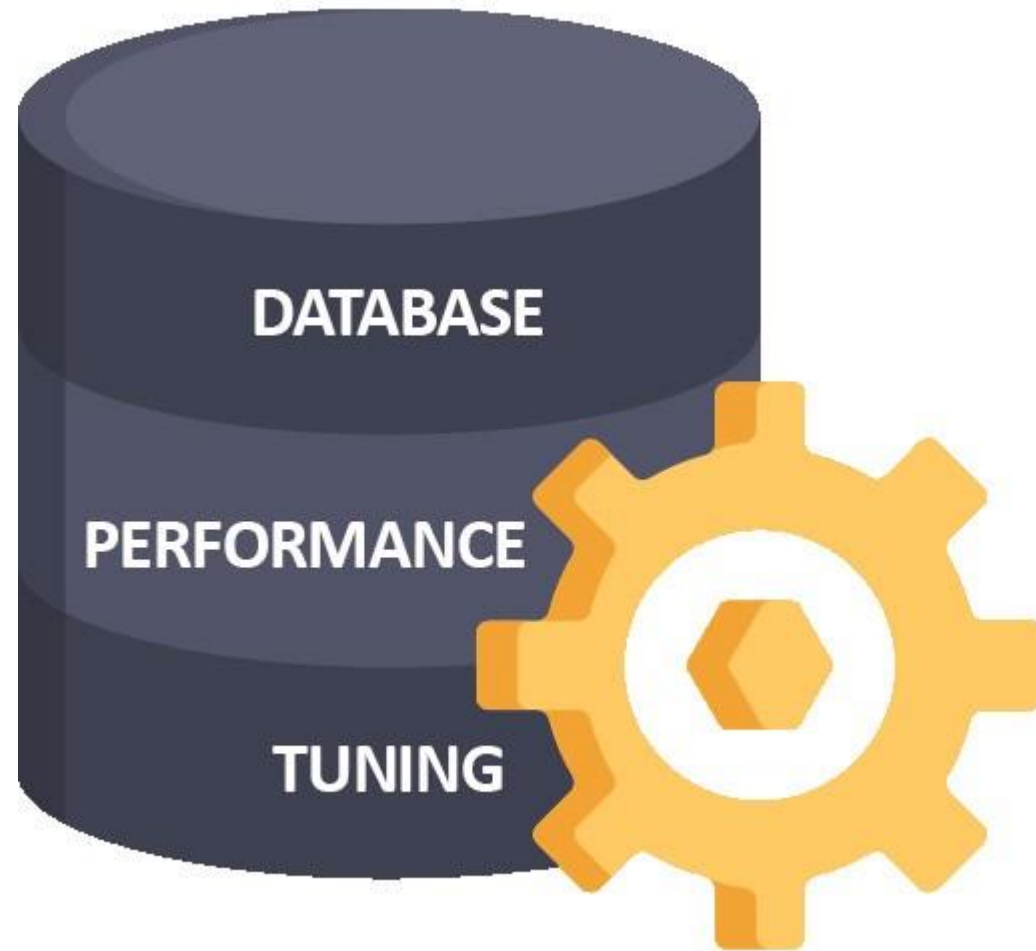


Performance Tuning...



Who Tunes?

The people who are involved with tuning:

- Database administrators
- Application architects
- Application designers
- Application developers
- System administrators
- Storage administrators
- Network administrators



What Does the DBA Tune?

Performance tuning areas:

- Application:
 - SQL statement performance **Shared with developers**
 - Change management
- Instance tuning:
 - Memory
 - Database structure
 - Instance configuration
- Operating system interactions:
 - I/O **Shared with SA**
 - Swap
 - Parameters



Introduction to Performance Tuning

- Monitoring and Diagnostics
 - Monitoring using available tools
 - Identifying the problem
 - Using AWR-based tools
- SQL Tuning
 - Identifying and tuning SQL statements by influencing the optimizer
 - Managing change
 - SQL Performance Management
 - Real Application Testing
- Instance Tuning
 - Tuning memory components
 - Tuning space usage and I/O

How to Tune

Available tools:

- Basic diagnostics:
 - Dynamic performance views
 - Statistics
 - Metrics
 - Enterprise Manager pages
- AWR or Statspack
- Automatic Database Diagnostic Monitor (ADDM)
- DBA scripts

Tuning Methodology

Tuning steps:

- Identify the scope of the problem (OS, database, and so on).
- Tune the following from the top down:
 - The design before tuning the application code
 - The code before tuning the instance
- Tune the area with the greatest potential benefit:
 - Identify the performance problem (AWR, Statspack).
 - Analyze the problem, looking for skewed and tunable components.
 - Use appropriate tools to tune the components implicated.
- Stop tuning when the goal is met.

Performance Tuning Tools

Available tools:

- Basic:
 - Time model
 - Top wait events
 - Dynamic performance views and tables
 - Alert log
 - Trace files
 - Enterprise Manager pages
- Add-in: Statspack
- Options:
 - Diagnostics Pack
 - Tuning Pack



Tuning Objectives

The objectives of tuning are:

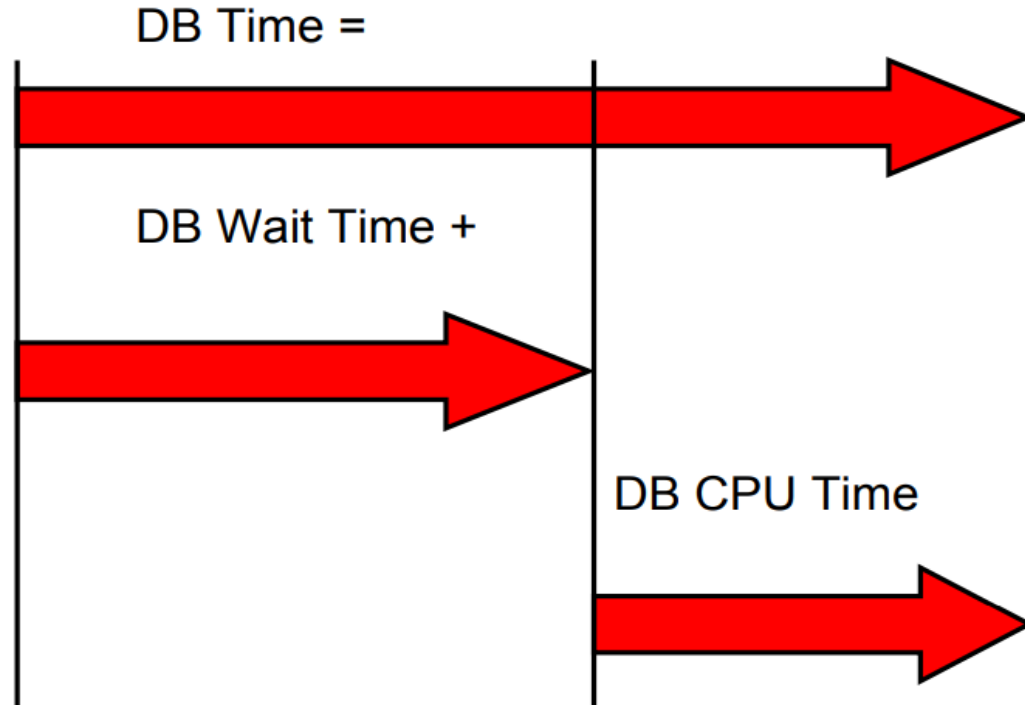
- Minimizing response time
- Increasing throughput
- Increasing load capabilities
- Reducing recovery time

Top Timed Events

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
free buffer waits	688,778	8,239	12	52.80	Configuration
buffer busy waits	67,632	3,817	56	24.46	Concurrency
enq: TX - index contention	18,705	1,475	79	9.45	Concurrency
log file sync	11,306	860	76	5.51	Commit
db file sequential read	180,952	552	3	3.53	User I/O

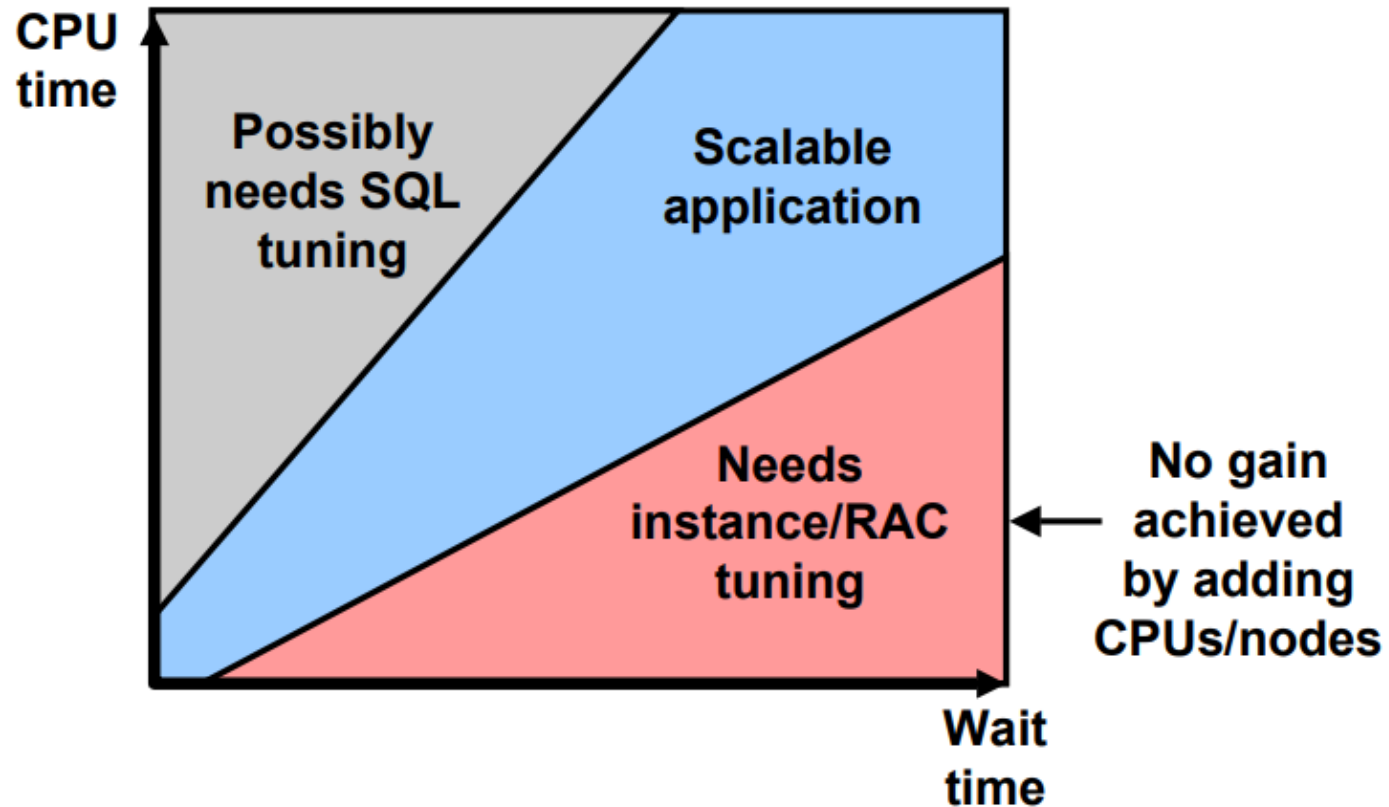
DB Time



Time model statistics use time to identify quantitative effects about specific actions performed on the database, such as logon operations and parsing. The most important time model statistic is database time, or DB time. This statistic represents the total time spent in database calls for foreground sessions and is an indicator of the total instance workload.

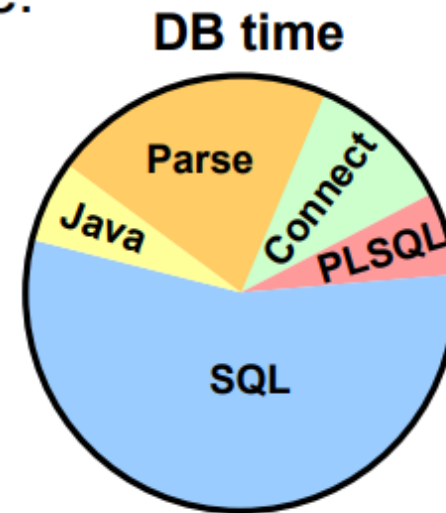
CPU and Wait Time Tuning Dimensions

$$\text{DB time} = \text{DB CPU time} + \text{DB wait time}$$



Time Model: Overview

- The time model is a set of statistics that give an overview of where time is spent inside the Oracle database.
- All statistics use the same dimension: time.
- The statistics are accessible through:
 - `V$SYS_TIME_MODEL`
 - `V$SESS_TIME_MODEL`
- DB time represents the total time spent in database calls by user sessions.
- A tuning goal is to reduce DB time.
- Using DB time, you can gauge the performance impact of any entity of the database.



Time Model Statistics Hierarchy

The relationships between the time model statistics are listed in the slide. They form two trees: background elapsed time and DB time. The time reported by a child in the tree is contained within the parent in the tree.

- **DB time:** Amount of elapsed time (in microseconds) spent performing database user-level calls. This does not include the time spent on instance background processes such as PMON. DB time is measured cumulatively from the time that the instance was started. Because DB time is calculated by combining the times from all non-idle user sessions, it is possible for the DB time to exceed the actual time elapsed since the instance started. For example, an instance that has been running for 30 minutes could have four active user sessions whose cumulative DB time is approximately 120 minutes.
- **DB CPU:** Amount of CPU time (in microseconds) spent on database user-level calls. This time include processes on the runqueue.

- **Parse time elapsed:** Amount of elapsed time spent parsing SQL statements. It includes both soft and hard parse time.
- **Hard parse elapsed time:** Amount of elapsed time spent hard-parsing SQL statements
- **SQL execute elapsed time:** Amount of elapsed time SQL statements are executing. Note that for SELECT statements, this also includes the amount of time spent performing fetches of query results.
- **Connection management call elapsed time:** Amount of elapsed time spent performing session connect and disconnect calls.
- **Failed parse elapsed time:** Amount of time spent performing SQL parses that ultimately fail with some parse error.
- **Failed parse (out of shared memory) elapsed time:** Amount of time spent performing SQL parses that fail with out of shared memory error.
- **Hard parse (sharing criteria) elapsed time:** Amount of elapsed time spent performing SQL hard parses when the hard parse resulted from not being able to share an existing cursor in the SQL cache.
- **Hard parse (bind mismatch) elapsed time:** Amount of elapsed time spent performing SQL hard parses when the hard parse resulted from bind type or bind size mismatch with an existing cursor in the SQL cache.

Dynamic Performance Views

The Oracle database server maintains a dynamic set of data about the operation and performance of the instance. These dynamic performance views are based on virtual tables that are built from memory structures inside the database server. That is, they are not conventional tables that reside in a database. V\$ views externalize metadata contained in memory structures of an Oracle instance. Some V\$ views can show data before a database is mounted or open. The V\$FIXED_TABLE view lists all the dynamic views.

Dynamic performance views include the raw information used by AWR and Statspack and detail information about but not limited to:

- Sessions
- Wait events
- Locks
- Backup status
- Memory usage and allocation
- System and session parameters
- SQL execution
- Statistics and metrics

Note: The DICT and DICT_COLUMNS views also contain the names of these dynamic

Dynamic Performance Views: Usage Examples

a

```
SQL> SELECT sql_text, executions  
2 FROM v$sqlstats  
3 WHERE cpu_time > 200000;
```

b

```
SQL> SELECT * FROM v$session  
2 WHERE machine = 'EDRSR9P1' and  
3 logon_time > SYSDATE - 1;
```

c

```
SQL> SELECT sid, ctime  
2 FROM v$lock WHERE block > 0;
```


Dynamic Performance Views: Considerations

- These views are owned by `SYS`.
- Different views are available at different times:
 - The instance has been started.
 - The database is mounted.
 - The database is open.
- You can query `V$FIXED_TABLE` to see all the view names.
- These views are often referred to as “v-dollar views.”
- All reads on these views are current reads.

Displaying Statistics

Instance Activity Statistics are collected for:

- **Sessions**
 - All sessions `V$SESSTAT`
 - Current session `V$MYSTAT`
- **Services** `V$SERVICE_STATS`
- **System** `V$SYSSTAT`

Displaying Statistics

The server displays a summary of all calculated instance activity statistics at the system level in the V\$SYSSTAT view. You can query this view to find cumulative totals since the instance started. At all levels there is a statistics identifier that can be joined to the V\$STATNAME table.

System-level statistics

```
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
redo entries	2	1126226
redo size	2	816992940

Service-Level Statistics

Service data is cumulative from the instance startup. The service name allows collection of statistics by a connection service name. This is very useful for performance monitoring by application. Every user that connects uses a specific service name per application.

Example

There are always two services defined: SYS\$BACKGROUND and SYS\$USERS. Up to 116 additional services may be created based on the SERVICE_NAMES parameter or set with the DBMS_SERVICE package. Service data is cumulative from the instance startup.

The Oracle database server displays all calculated service statistics in the V\$SERVICE_STAT view. You can query this view to find service cumulative totals since the instance started.

```
SQL> select service_name, stat_name, value
       2  from v$service_stats;
```

SERVICE_NAME	STAT_NAME	VALUE
-----	-----	-----
SYS\$USERS	user calls	6977
SERV1	user calls	532
SYS\$BACKGROUND	user calls	0
orcl.oracle.com	user calls	18948
orclXDB	user calls	0
SYS\$USERS	DB time	84608280
SERV1	DB time	222965588
SYS\$BACKGROUND	DB time	0
orcl.oracle.com	DB time	55877745

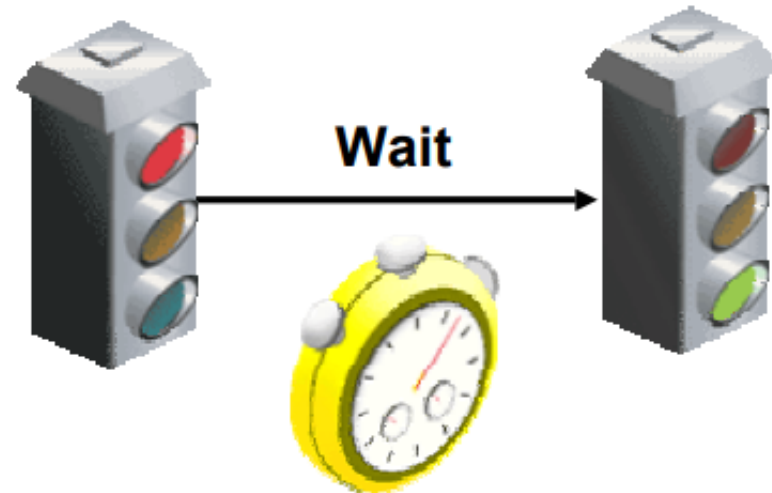
Displaying SGA Statistics

```
SQL> SELECT * FROM V$SGAINFO;
```

NAME	BYTES	RES
-----	-----	---
Fixed SGA Size	1303132	No
Redo Buffers	17780736	No
Buffer Cache Size	50331648	Yes
Shared Pool Size	142606336	Yes
Large Pool Size	4194304	Yes
Java Pool Size	12582912	Yes
Streams Pool Size	0	Yes
Shared IO Pool Size	0	Yes
Granule Size	4194304	No
Maximum SGA Size	836976640	No
Startup overhead in Shared Pool	41943040	No
Free SGA Memory Available	608174080	

Wait Events

- A collection of wait events provides information about 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



Wait Events

All wait events are named in the `V$EVENT_NAME` view, including:

- Free buffer waits
- Latch free
- Buffer busy waits
- Db file sequential read
- Db file scattered read
- Db file parallel write
- Undo segment tx slot
- Undo segment extension

Each event is assigned to a wait class. This assignment is shown in the `V$EVENT_NAME` view. Each event can have additional parameters returned with the event, columns `PARAMETER1` through `PARAMETER3` show the meaning of these parameters.

Note: Time information columns for wait events are populated only if the `TIMED_STATISTICS` initialization parameter is set to true.

Commonly Observed Wait Events

Wait Event	Area
Buffer busy waits	Buffer cache, DBWR
Free buffer waits	Buffer cache, DBWR, I/O
Db file scattered read, Db file sequential read	I/O, SQL Tuning
Enqueue waits (enq:)	Locks
Library cache waits	Mutexes/Latches
Log buffer space	Log buffer I/O
Log file sync	Over-commit, I/O

Using Features of the Packs

Monitoring and tuning with packs

Database Diagnostics Pack

- Automatic Workload Repository
- Automatic Database Diagnostic Monitor (ADDM)
- Active Session History (ASH)
- Performance monitoring (database and host)
- Event notifications: notification methods, rules, and schedules
- Event history and metric history (database and host)
- Blackouts
- Dynamic metric baselines
- Monitoring templates

Database Tuning Pack

- SQL Access Advisor
- SQL Tuning Advisor
- Automatic SQL Tuning
- SQL Tuning Sets
- Automatic Plan Evolution of SQL Plan Management
- SQL Monitoring
- Reorganize objects

Database Configuration Management Pack

- Database and Host Configuration
- Deployments
- Patch Database and View Patch Cache
- Patch staging
- Clone Database
- Clone Oracle Home
- Search configuration
- Compare configuration
- Policies

Monitoring and tuning without packs

- SQL traces
- Statspack
- System statistics
- Wait model
- Time model
- OS statistics
- Metrics
- Service statistics
- Histograms
- Optimizer statistics
- SQL statistics

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:
 - Performing a `BACKUP CONTROL FILE TO TRACE`
 - Process errors

User Trace Files

Server processes can generate user trace files at the request of the user or DBA.

Instance-Level Tracing

Instance-level tracing should only be enabled when absolutely necessary. Tracing all sessions will create an I/O load and can fill the file system quickly. This trace logging is enabled or disabled by the `EXEC DBMS_MONITOR.DATABASE_TRACE_ENABLE()`.

Session-Level Tracing

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

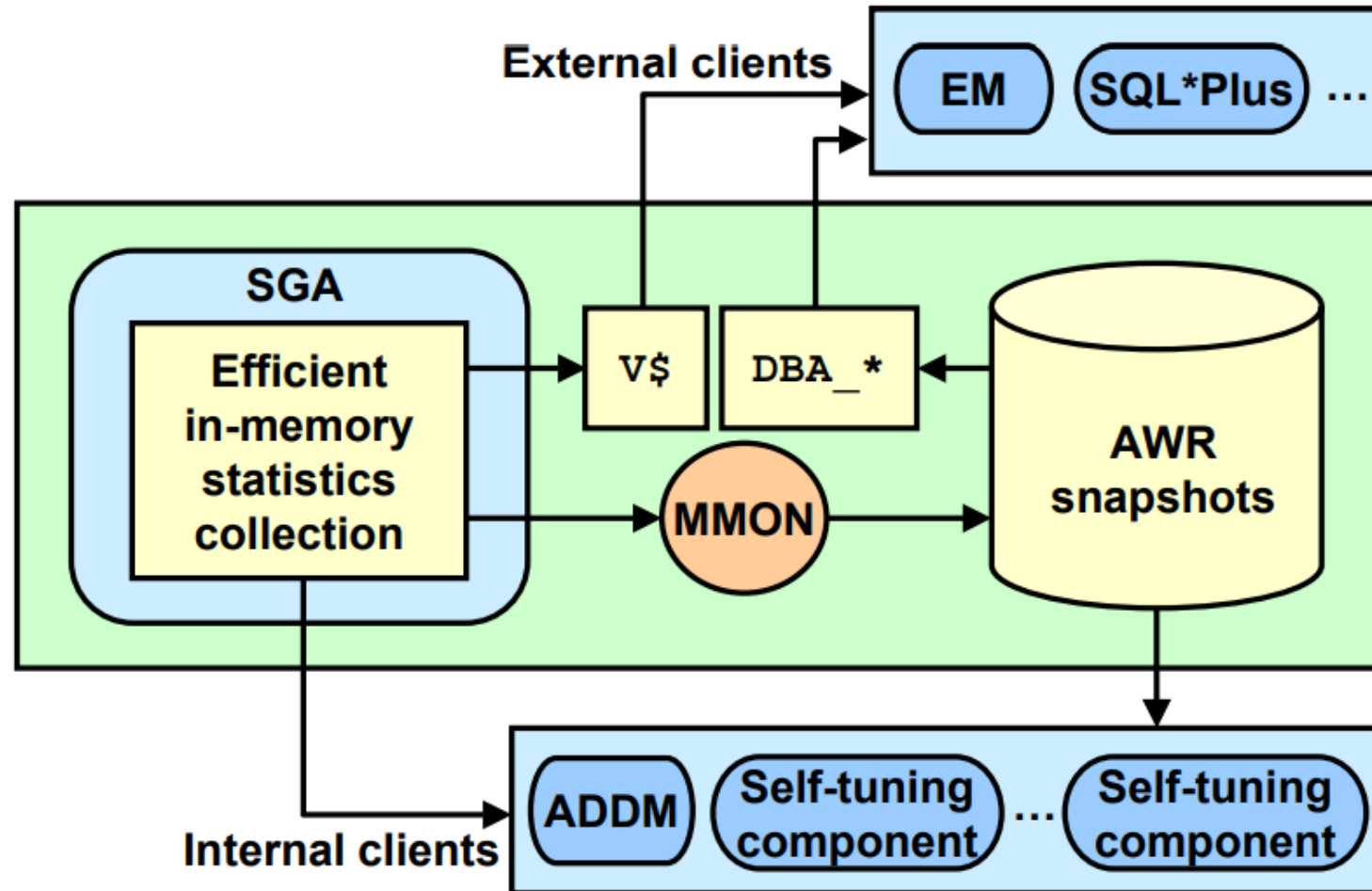
```
EXECUTE DBMS_MONITOR.SESSION_TRACE_ENABLE (8,12,  
      waits=>TRUE, binds=>TRUE);
```

where 8 and 12 are the system identifier and serial number of the connected user. Typically only a DBA has the permissions required to enable tracing on any session.

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

- **On UNIX:** `$ORACLE_HOME/rdbms/admin`
- **On Windows:** `%ORACLE_HOME%\rdbms\admin`

Automatic Workload Repository: Overview



The AWR infrastructure consists of two major parts:

- An in-memory statistics collection facility that is used by various components to collect statistics. These statistics are stored in memory for performance reasons. Statistics stored in memory are accessible through dynamic performance (V\$) views.
- AWR snapshots represent the persistent portion of the facility. The AWR snapshots are accessible through data dictionary (DBA) views and Database Control.

Statistics are stored in persistent storage for several reasons:

- The statistics need to survive instance crashes.
- Historical data for baseline comparisons is needed for certain types of analysis.
- Memory overflow: When old statistics are replaced by new ones due to memory shortage, the replaced data can be stored for later use.

The memory version of the statistics is transferred to disk on a regular basis by a background process called MMON (Manageability Monitor).

Automatic Workload Repository Data



**Base
statistics**



**SQL
statistics**



**Advisor
results**



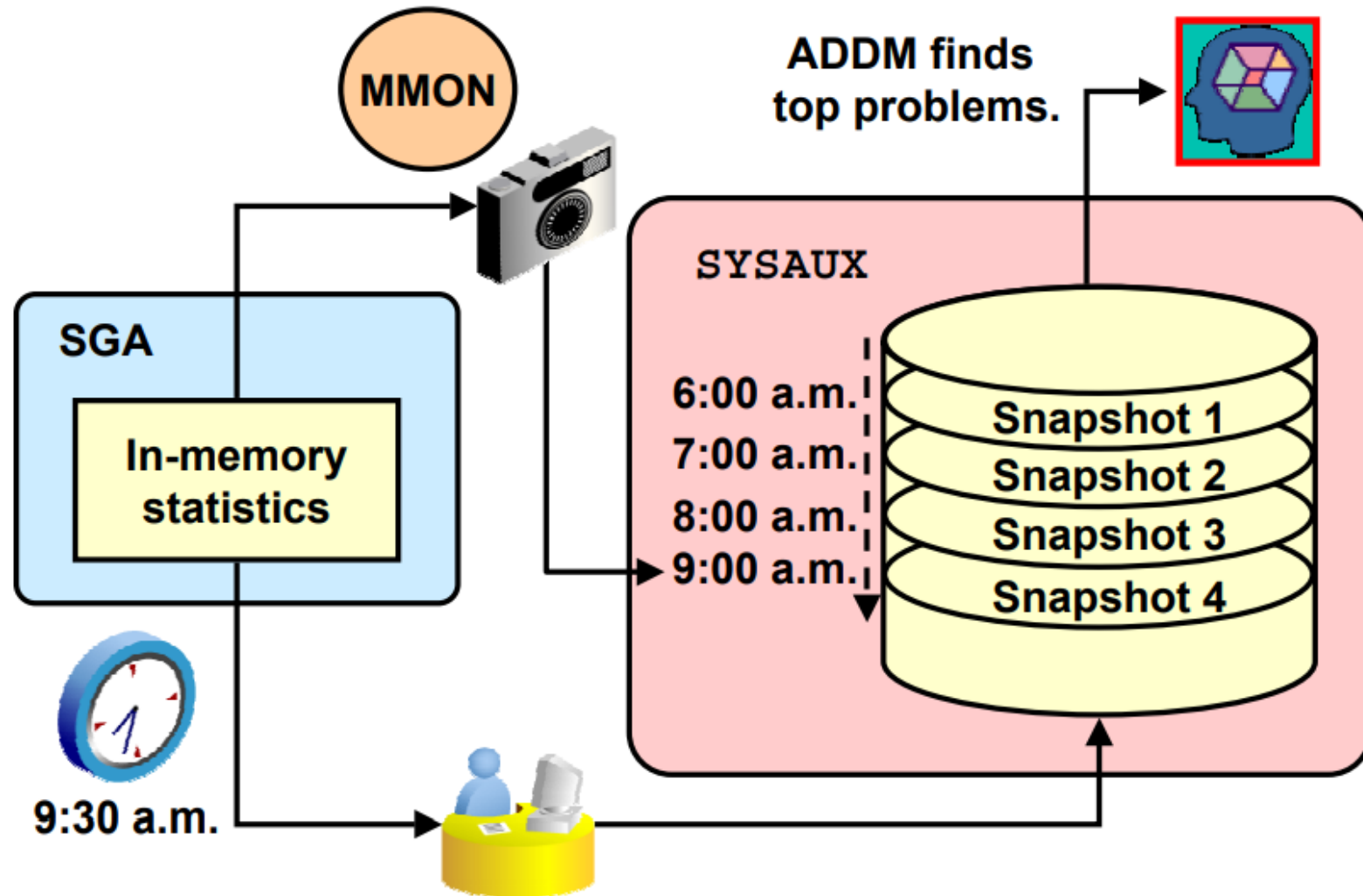
Metrics



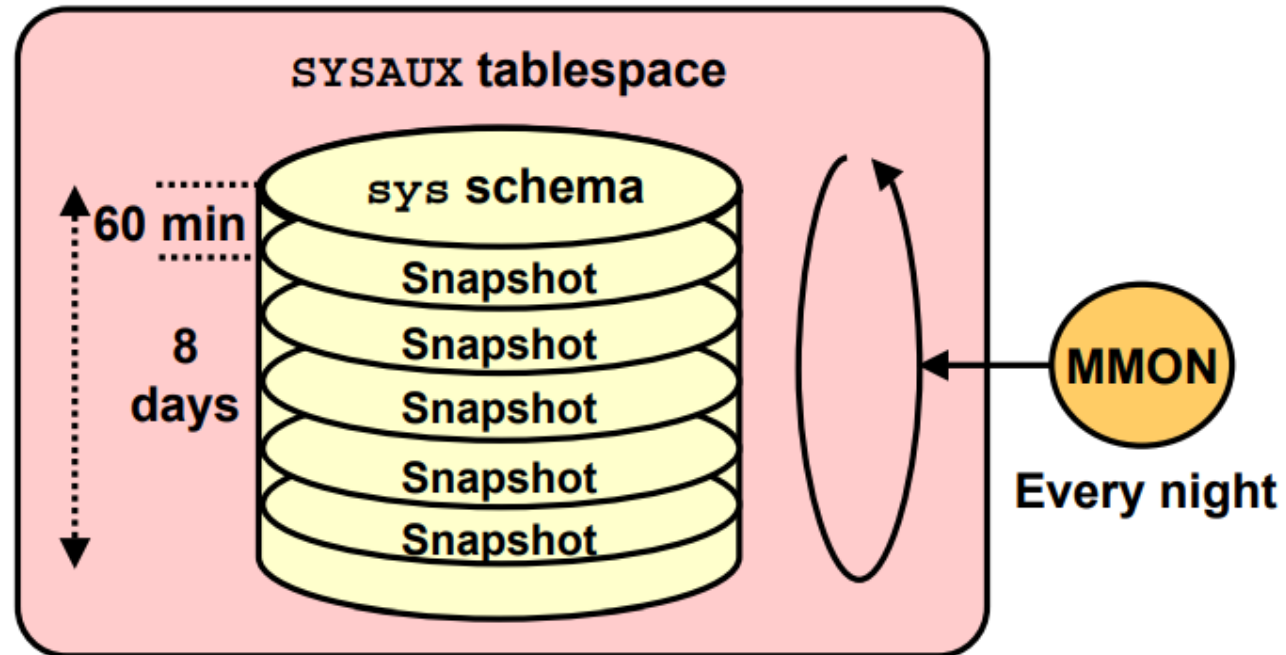
ASH

AWR

Workload Repository



AWR Snapshot Purging Policy



Generating AWR Reports in SQL*Plus

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
ORCL	1237161768	orcl	1	03-Feb-10 17:02	11.2.0.1.0	NO

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
edrsr10p1.us.oracle.com	Linux IA (32-bit)	1	1	1	1.97

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	268	03-Feb-10 17:48:15	37	2.4
End Snap:	270	03-Feb-10 19:22:03	37	2.7
Elapsed:		93.79 (mins)		
DB Time:		25.76 (mins)		

Reading the AWR Report

- The first section provides
 - Overview
 - Most significant diagnostics
- Additional pages
 - Detailed statistical information for specific areas

Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
buffer busy waits	55,295	691	13	91.85	Concurrency
log file sync	114	26	230	3.48	Commit
DB CPU		25		3.31	
db file sequential read	6,671	6	1	0.75	User I/O
enq: HW - contention	428	5	11	0.60	Configuration

Compare Periods: Load Profile

	1st per sec	2nd per sec	%Diff	1st per txn	2nd per txn	%Diff
DB time:	5.21	2.12	-59.31	12.55	2.71	-78.41
CPU time:	0.21	0.24	14.29	0.50	0.30	-40.00
Redo size:	1,339,092.39	1,336,024.21	-0.23	3,226,176.75	1,708,280.64	-47.05
Logical reads:	10,193.42	5,883.66	-42.28	24,558.26	7,523.02	-69.37
Block changes:	10,784.82	10,763.85	-0.19	25,983.08	13,762.98	-47.03
Physical reads:	89.16	89.36	0.22	214.81	114.26	-46.81
Physical writes:	50.32	59.85	18.94	121.23	76.52	-36.88
User calls:	16.55	23.25	40.48	39.87	29.73	-25.43
Parses:	29.02	34.28	18.13	69.92	43.83	-37.31
Hard parses:	3.34	2.76	-17.37	8.06	3.53	-56.20
Sorts:	17.01	13.23	-22.22	40.98	16.92	-58.71
Logons:	0.34	0.34	0.00	0.81	0.43	-46.91
Executes:	5,389.25	5,377.51	-0.22	12,983.92	6,875.84	-47.04
Transactions:	0.42	0.78	85.71			

Using AWR-Based Tools

This practice covers the following topics:

- Creating and managing Automatic Workload Repository (AWR) snapshots
- Generating and viewing the sections of an AWR report
- Generating and viewing the sections of a Compare Periods report

Top SQL Reports

SQL by Elapsed Time

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
1,006	453	1,757	0.57	95.22	<u>fu02q80b2kva1</u>	DEMO	select time_id, QUANTITY_SOLD...
31	10	0		2.94	<u>710ydg8q1fj3r</u>	SQL*Plus	DECLARE pid NUMBER := 37; ...

SQL by CPU Time

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total	% Total DB Time	SQL Id	SQL Module	SQL Text
453	1,006	1,757	0.26	95.58	95.22	<u>fu02q80b2kva1</u>	DEMO	select time_id, QUANTITY_SOLD...
11	21	0		2.29	1.99	<u>710ydg8q1fj3r</u>	SQL*Plus	DECLARE pid NUMBER := 39; ...

SQL by Executions

Executions	Rows Processed	Rows per Exec	CPU per Exec (s)	Elap per Exec (s)	SQL Id	SQL Module	SQL Text
1,757	5,271	3.00	0.26	0.57	<u>fu02q80b2kva1</u>	DEMO	select time_id, QUANTITY_SOLD...
415	284	0.68	0.00	0.00	<u>96q93hntzjtr</u>		select /*+ rule */ bucket_cnt,...

SQL by Buffer Gets

Buffer Gets	Executions	Gets per Exec	%Total	CPU Time (s)	Elapsed Time (s)	SQL Id	SQL Module	SQL Text
5,579,956	1,757	3,175.84	93.92	453.08	1006.49	<u>fu02q80b2kva1</u>	DEMO	select time_id, QUANTITY_SOLD...
134,514	0		2.26	10.84	21.04	<u>710ydg8q1fj3r</u>	SQL*Plus	DECLARE pid NUMBER := 39; ...

Common Tuning Problems

The most common tuning problems:

- Inefficient or high-load SQL statements
- Suboptimal use of Oracle Database by the application
- Undersized memory structures
- Concurrency issues
- I/O issues
- Database configuration issues
- Short-lived performance problems
- Degradation of database performance over time
- Unexpected performance regression after environment changes
- Locking issues

ADDM Tuning Session

An ADDM tuning session follows the same procedure as a manual tuning session, but combines steps.

ADDM Tuning Session	Manual Tuning Session
Generate the ADDM report.	Collect current statistics. Compare current statistics with a previous set; look up in a performance-issues knowledge base. Define the problem and make recommendations.
Review the recommendations.	Build a trial solution.
Implement the recommendations.	Implement and measure the change.
Review the next ADDM report.	Decide: "Did the solution meet the goal?"

Active Session History: Overview

- Stores the history of database time
- Samples session activity in the system including:
 - SQL identifier of a SQL statement
 - Object number, file number, and block number
 - Wait event identifier and parameters
 - Session identifier and session serial number
 - Module and action name
 - Client identifier of the session
 - Service hash identifier
 - Blocking session
- Is always on for first fault analysis
- No need to replay the workload

Accessing ASH Data

- V\$ACTIVE_SESSION_HISTORY
- DBA_HIST_ACTIVE_SESS_HISTORY
- ASH report
- EM Diagnostic Pack performance pages

ASH Report: General Section

ASH Report For ORCL/orcl

DB Name	DB Id	Instance	Inst num	Release	RAC	Host
ORCL	1236987013	orcl	1	11.2.0.1.0	NO	edtdr35p1.us.oracle.com

CPUs	SGA Size	Buffer Cache	Shared Pool	ASH Buffer Size
2	459M (100%)	108M (23.5%)	172M (37.5%)	4.0M (0.9%)

	Sample Time	Data Source
Analysis Begin Time:	08-Feb-10 14:28:02	V\$ACTIVE_SESSION_HISTORY
Analysis End Time:	08-Feb-10 14:33:02	V\$ACTIVE_SESSION_HISTORY
Elapsed Time:	5.0 (mins)	
Sample Count:	0	
Average Active Sessions:	0.00	
Avg. Active Session per CPU:	0.00	
Report Target:	None specified	

V\$ACTIVE_SESSION_HISTORY

DBA_HIST_ACTIVE_SESSION_HISTORY

ASH Report Structure

The slide shows you the various section of the ASH Report. The ASH Report follows the pattern of the AWR report. Starting from the upper right of the slide, the report sections are as follows:

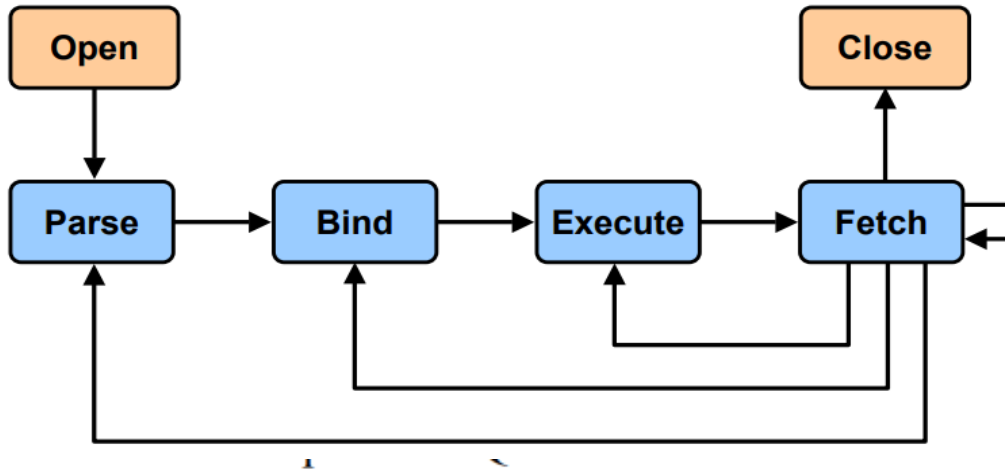
- **Top Events:** Reports the user events, background events, and the event parameter values
- **Load Profile:** Reports the top service/module, top clients, identifies the type of SQL commands and top phases of execution
- **Top SQL:** Reports top SQL statements associated with the top events, SQL associated with the top rowsources, top SQL using literals, and the SQL text for these SQL statements.
- **Top PL/SQL Procedures:** Lists the PL/SQL procedures that accounted for the highest percentages of sampled session activity
- **Top Java Workload:** Describes the top Java programs in the sampled session activity
- **Top Sessions:** Reports the top sessions found waiting, top blocking sessions, and aggregates for PQ sessions
- **Top Objects/Files/Latches:** Reports the top objects, files, or latches that were involved in a wait
- **Activity Over Time:** Reports the top three wait events for 10 equally sized time periods during the report period

Conclusion

- Do not forget to use ASH where needed
- It is important to check ADDM report
- Consider using SQL Plan Management
- Compare the numbers over time
- Reduce long full table scans in OLTP
- Minimize locking usage
- Do not forget to do maintenance
- Do not forget to use Real time ADDM

SQL Statement Processing Phases

Parse Phase



- Parse phase:
 - Always:
 - Checks syntax
 - Checks semantics and privileges
 - Soft parse:
 - Searches for the statement in the shared pool
 - Hard parse:
 - Merges view definitions and subqueries
 - Determines execution plan

There are two types of parse operations:

- **Soft parsing:** A SQL statement is submitted, and a match *is* found in the shared pool. The match can be the result of a previous execution by another user. The SQL statement is shared, which is good for performance. However, soft parses still require syntax and security checking, which consume system resources.
- **Hard parsing:** A SQL statement is submitted for the first time, and no shareable match is found in the shared pool. Hard parses are the most resource-intensive and unscalable, because they perform all the operations involved in a parse.

When bind variables are used properly, more soft parses are possible, thereby reducing hard parses and keeping parsed statements in the library cache for a longer period.

SQL Statement Processing Phases: Bind

- Bind phase:
 - Checks the statement for bind variables
 - Assigns or reassigns a value to the bind variable
- Bind variables impact performance when:
 - Parsing is reduced by using a shared cursor.
 - A different execution plan might benefit performance with different bind values.

SQL Statement Processing Phases: Bind

During the bind phase:

- The Oracle Database checks the statement for references to bind variables.
- The Oracle Database assigns or reassigns a value to each variable.

When bind variables are used in a statement, the optimizer assumes that cursor sharing is intended and that different invocations should use the same execution plan. This helps performance by reducing hard parses.

Role of the Oracle Optimizer

- The Oracle query optimizer determines the most efficient execution plan and is the most important step in the processing of any SQL statement.
- The optimizer:
 - Evaluates expressions and conditions
 - Uses object and system statistics
 - Decides how to access the data
 - Decides how to join tables
 - Decides which access path is most efficient

Role of the Oracle Optimizer

The optimizer is the part of the Oracle Database that creates the execution plan for a SQL statement. The determination of the execution plan is an important step in the processing of any SQL statement and can greatly affect execution time.

The information needed by the optimizer includes:

- Statistics gathered for the system (I/O, CPU, and so on) as well as schema objects (number of rows, index, and so on)
- Information in the dictionary
- WHERE clause qualifiers
- Hints supplied by the developer

When you use diagnostic tools such as Enterprise Manager, EXPLAIN PLAN, and SQL*Plus AUTOTRACE, you can see the execution plan that the optimizer chooses.

TOP SQL Reports

SQL Ordered by CPU Time

CPU Time (s)	Executions	CPU per Exec (s)	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
5.28	120	0.04	16.92	6.24	84.59	2.16	6gvch1xu9ca3g		DECLARE job BINARY_INTEGER := ...
2.25	2,880	0.00	7.20	2.15	104.77	0.00	6v7n0y2bq89n8	OEM.SystemPool	BEGIN EMDW_LOG.set_context(MGM...
1.45	2	0.73	4.65	1.46	99.47	0.00	dayq182sk41ks		insert into wrh\$_memory_target...
1.45	2	0.73	4.65	1.46	99.50	0.00	bm2pwrpcr8ru6		select sga_size s, sga_size_fa...

SQL Ordered by Gets

Buffer Gets	Executions	Gets per Exec	%Total	Elapsed Time (s)	%CPU	%IO	SQL Id	SQL Module	SQL Text
93,294	120	777.45	28.77	6.24	84.59	2.16	6gvch1xu9ca3g		DECLARE job BINARY_INTEGER := ...
29,912	5,971	5.01	9.22	0.48	92.58	0.02	8vww6hx92ymmm	OEM.SystemPool	UPDATE MGMT_CURRENT_METRICS SE...
27,551	19	1,450.05	8.50	0.77	89.80	0.00	5k5v1ah25fb2c	OEM.SystemPool	BEGIN EMD_LOADER.UPDATE_CURREN...
23,256	6,597	3.53	7.17	0.45	103.56	0.00	cm5vu20fhtng1		select /*+ connect_by_filterin...
18,748	671	27.94	5.78	0.27	100.13	0.05	6amvqb1yqg2y7	OEM.SystemPool	INSERT INTO MGMT_METRICS_RAW(C...

The SQL Monitoring Report

Database Instance: orcl.example.com > Monitored SQL Executions >

Logged in As SYS

Monitored SQL Execution Details



SQL Monitoring Report

SQL Text

```
SELECT count(*) FROM moni_test t1, moni_test t2 WHERE t1.c=t2.c AND t1.c=1
```

Global Information

Status : EXECUTING
Instance ID : 1
Session ID : 125
SQL ID : 0j1tb5nmgm437
SQL Execution ID : 16777217
Plan Hash Value : 183808681
Execution Started : 03/09/2009 08:37:45
First Refresh Time : 03/09/2009 08:37:59
Last Refresh Time : 03/09/2009 08:54:13

Elapsed Time(s)	Cpu Time(s)	IO Waits(s)	Other Waits(s)	Buffer Gets	Reads	read_reqs	read_bytes
989	961	0.01	27	536	18	18	3162K

SQL Plan Monitoring Details

Id	Operation	Name	Rows (Estim)	Cost	Time Active(s)	Start Active	Starts	Rows (Actual)	Memory	Activity (percent)	Activity Detail (sample #)
0	SELECT STATEMENT						1				
-> 1	SORT AGGREGATE		1		992	+1	1	0			
2	HASH JOIN		40305M	181K	990	+2	1	4210M	9044K		
3	INDEX FAST FULL SCAN	MONI_TEST_C_INDX	201K	114	1	+14	1	200K			
-> 4	INDEX FAST FULL SCAN	MONI_TEST_C_INDX	201K	114	975	+14	1	67066			

What Is an Execution Plan?

An execution plan is a set of steps that the optimizer performs when executing a SQL statement and performing an operation.

What Is an Execution Plan?

When a statement is executed, the server executes steps of the plan created by the optimizer. Each step either retrieves rows of data physically from the database or prepares them in some way for the user issuing the statement. The combination of steps that are used to run a statement is called an “execution plan.”

An execution plan includes an access method for each table that the statement accesses and an ordering of the tables (the join order). The optimizer also uses different methods to combine the rows from multiple tables (the join method). The steps of the execution plan are not performed in the order in which they are numbered.

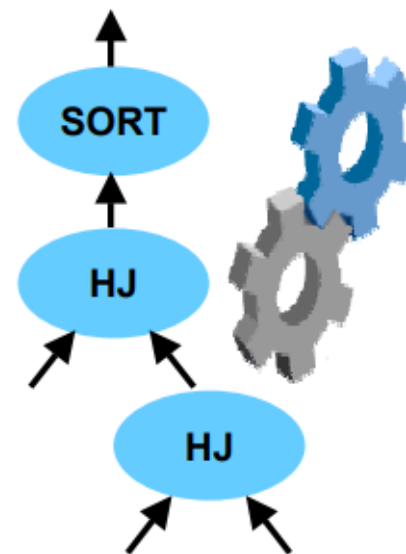
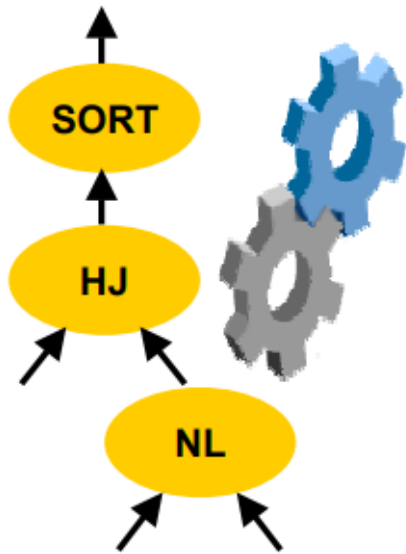
Methods for Viewing Execution Plans

To view execution plans, use:

- Enterprise Manager SQL pages
- DBMS_XPLAN methods to view plans from:
 - Automatic Workload Repository
 - V\$SQL_PLAN
 - SQL Tuning Sets
 - Plan table
- SQL Trace (event 10046) with tkprof
- SQL*Plus AUTOTRACE
- EXPLAIN PLAN

Uses of Execution Plans

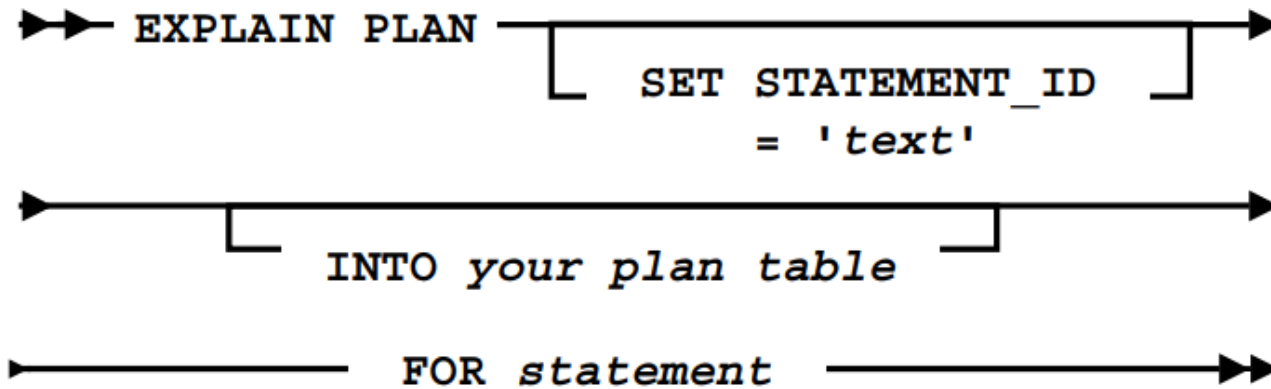
- Determining the current execution plan
- Identifying the effect of indexes
- Determining access paths
- Verifying the use of indexes
- Verifying which execution plan may be used



DBMS_XPLAN Package: Overview

- The DBMS_XPLAN package provides an easy way to display the output from the:
 - EXPLAIN PLAN command
 - Automatic Workload Repository (AWR)
 - V\$SQL_PLAN and V\$SQL_PLAN_STATISTICS_ALL fixed views
- The DBMS_XPLAN package supplies three table functions that can be used to retrieve and display the execution plan:
 - DISPLAY
 - DISPLAY_AWR
 - DISPLAY_CURSOR

EXPLAIN PLAN Command: Example



```
EXPLAIN PLAN
SET STATEMENT_ID = 'demo01' FOR
SELECT e.last_name, d.department_name
FROM hr.employees e, hr.departments d
WHERE e.department_id = d.department_id;
Explained.
```

Note: The `EXPLAIN PLAN` command does not actually execute the statement.

EXPLAIN PLAN Command: Output

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Plan hash value: 1343509718

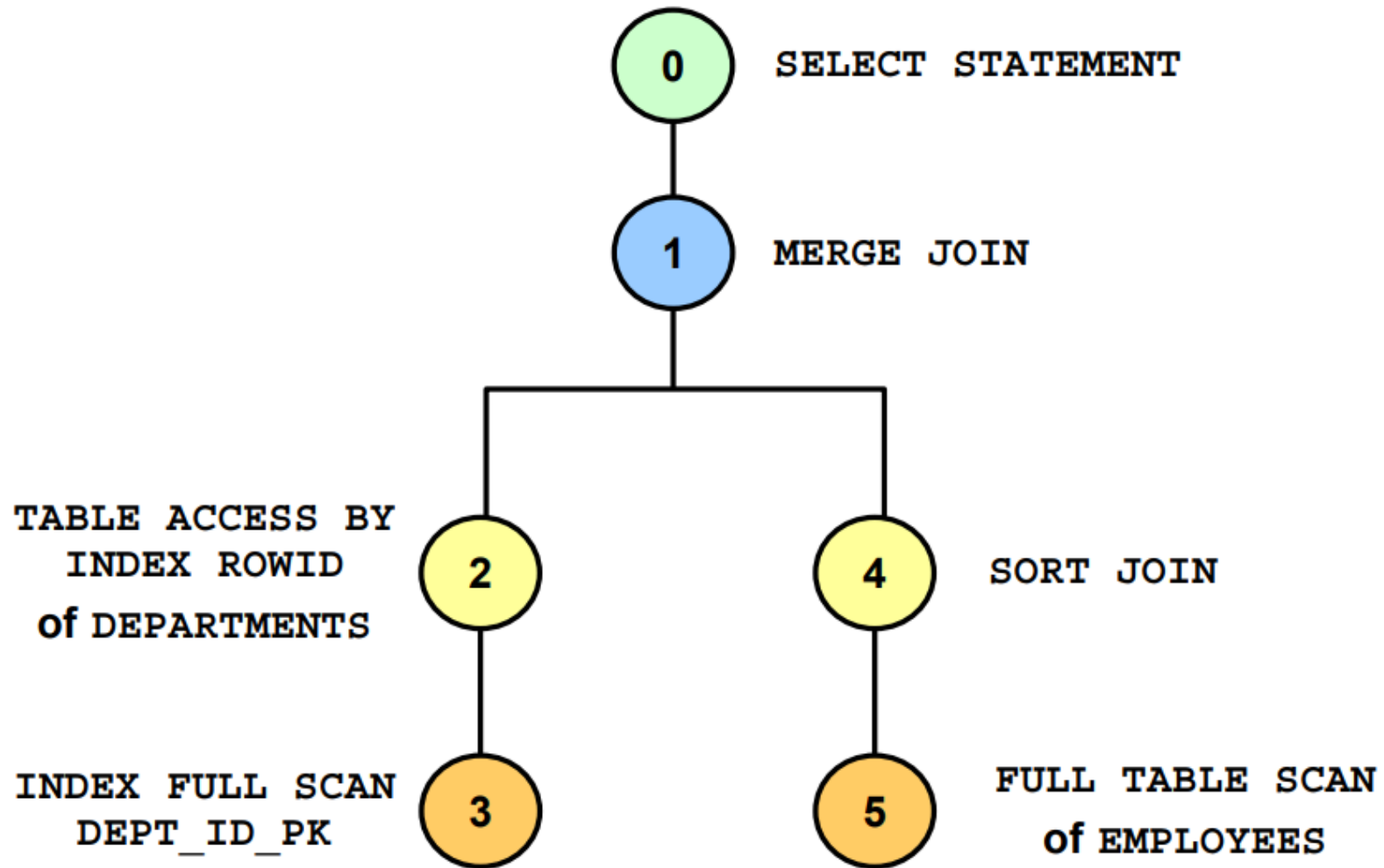
Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT		106	2862	6 (17)
1	MERGE JOIN		106	2862	6 (17)
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	27	432	2 (0)
3	INDEX FULL SCAN	DEPT_ID_PK	27		1 (0)
* 4	SORT JOIN		107	1177	4 (25)
5	TABLE ACCESS FULL	EMPLOYEES	107	1177	3 (0)

Predicate Information (identified by operation id):

```
4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
    filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

18 rows selected.

Reading an Execution Plan



Querying V\$SQL_PLAN

```
SELECT PLAN_TABLE_OUTPUT FROM  
TABLE(DBMS_XPLAN.DISPLAY_CURSOR('cfz0cdukrfdnu'));
```

```
SQL_ID cfz0cdukrfdnu, child number 0  
-----  
SELECT e.last_name, d.department_name  
FROM hr.employees e, hr.departments d WHERE  
e.department_id =d.department_id
```

Plan hash value: 1343509718

Id	Operation	Name	Rows	Bytes	Cost (%CPU)
0	SELECT STATEMENT				6 (100)
1	MERGE JOIN		106	2862	6 (17)
2	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	27	432	2 (0)
3	INDEX FULL SCAN	DEPT_ID_PK	27		1 (0)
* 4	SORT JOIN		107	1177	4 (25)
5	TABLE ACCESS FULL	EMPLOYEES	107	1177	3 (0)

Predicate Information (identified by operation id):

```
-----  
4 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")  
filter("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

24 rows selected.

Querying the AWR

```
SELECT PLAN_TABLE_OUTPUT FROM TABLE  
(DBMS_XPLAN.DISPLAY_AWR('454rug2yva18w'));
```

PLAN_TABLE_OUTPUT

SQL_ID 454rug2yva18w

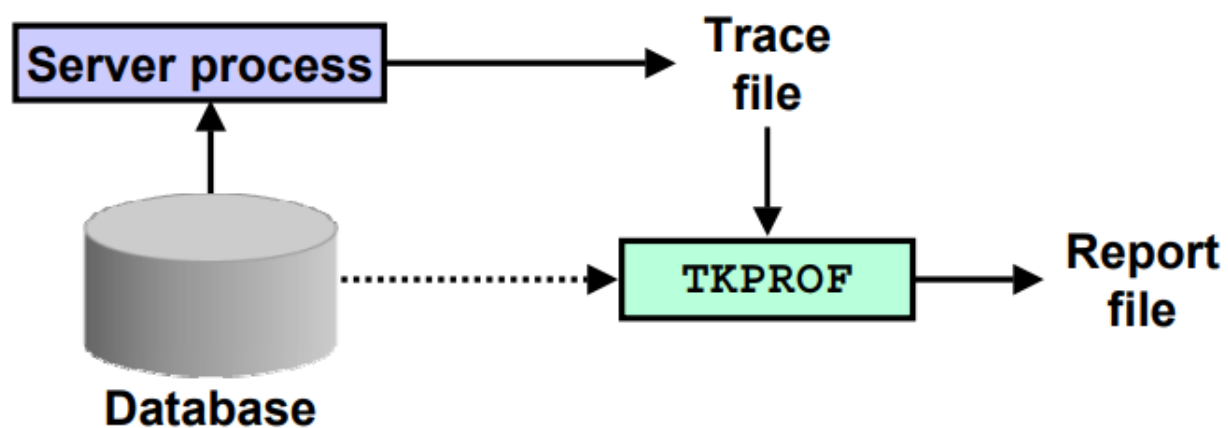
select /* example */ * from hr.employees natural join hr.departments

Plan hash value: 2052257371

Id	Operation	Name	Rows	Bytes	Cost(%CPU)	Time
0	SELECT STATEMENT				6 (100)	
1	HASH JOIN		11	968	6 (17)	00:00:01
2	TABLE ACCESS FULL	DEPARTMENTS	11	220	2 (0)	00:00:01
3	TABLE ACCESS FULL	EMPLOYEES	107	7276	3 (0)	00:00:01

SQL Trace Facility

- Usually enabled at the session level
- Gathers session statistics for SQL statements grouped by session
- Produces output that can be formatted by TKPROF



SQL Trace Facility

If you are using Standard Edition or do not have the Diagnostics Pack, the SQL Trace facility and TKPROF let you collect the statistics for SQL execution plans to compare performance. A good way to compare two execution plans is to execute the statements and compare the statistics to see which one performs better. SQL Trace writes its session statistics output to a file, and you use TKPROF to format it. You can use these tools along with EXPLAIN PLAN to get the best results.

SQL Trace facility:

- Can be enabled for a session or for an instance
- Reports on volume and time statistics for the parse, execute, and fetch phases
- Produces output that can be formatted by TKPROF

When the SQL Trace facility is enabled for a session, the Oracle Database generates a trace file containing session statistics for traced SQL statements for that session. When the SQL Trace facility is enabled for an instance, the Oracle Database creates trace files for all sessions.

Note: SQL Trace involves some overhead, so you usually do not want to enable SQL Trace at the instance level.

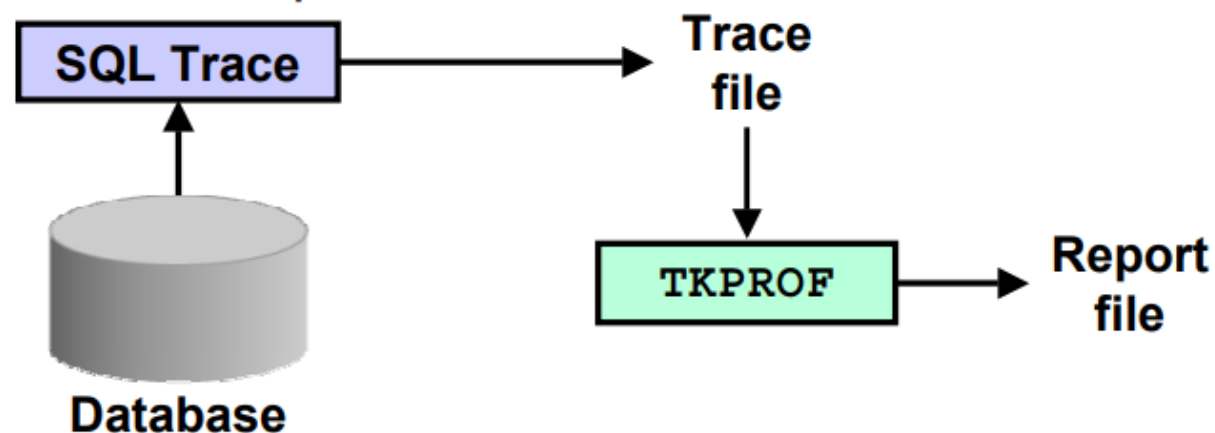
The SQL Trace facility provides performance information on individual SQL statements. SQL Trace provides the following, including row source information:

- Parse, execute, and fetch counts
- CPU and elapsed times
- Physical reads and logical reads
- Number of rows processed
- Misses on the library cache
- Username under which each parse occurred
- Each commit and rollback
- Row operations showing the actual execution plan of each SQL statement
- Number of rows, number of consistent reads, number of physical reads, number of physical writes, and time elapsed for each operation on a row

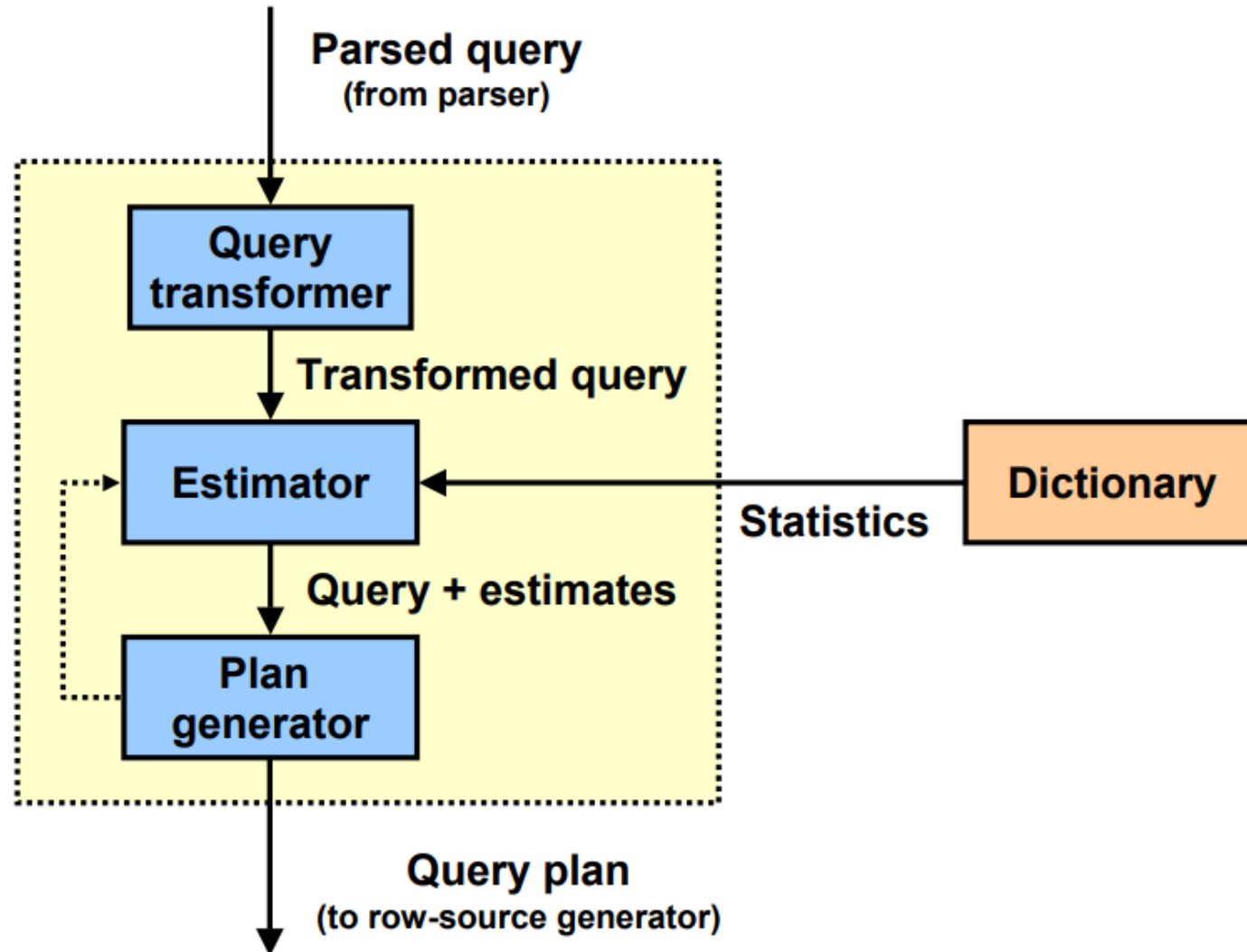
Note: A summary for each trace file can be obtained using the TKPROF utility.

How to Use the SQL Trace Facility

1. Set the initialization parameters.
2. Enable tracing.
3. Run the application.
4. Disable Trace.
5. Close the session.
6. Format the trace file.
7. Interpret the output.



Functions of the Query Optimizer



Selectivity

- Selectivity represents a fraction of rows from a row source.
 - Selectivity affects the estimates of I/O cost.
 - Selectivity affects the sort cost.
- Selectivity lies in a value range from 0.0 to 1.0.
- When statistics are available, the estimator uses them to estimate selectivity.
- When statistics are not available the estimator uses default values or dynamic sampling.
- With histograms on columns that contain skewed data, the results are good selectivity estimates.

Cardinality and Cost

- Cardinality represents the number of rows in a row source.
- Cost represents the units of work or resource that are used.

Cardinality and Cost

Cardinality: Represents the number of rows in a row source. Here, the row source can be a base table, a view, or the result of a join or GROUP BY operator. If a select from a table is performed, the table is the row source and the cardinality is the number of rows in that table.

Note: Not all of the possible row sources are considered here.

Cost: Represents the number of units of work (or resource) that are used. The query optimizer uses disk I/O, CPU usage, and memory usage as units of work. So the cost used by the query optimizer represents an estimate of the number of disk I/Os and the amount of CPU and memory used in performing an operation. The operation can be scanning a table, accessing rows from a table by using an index, joining two tables together, or sorting a row source. The cost of a query plan is the number of work units that are expected to be incurred when the query is executed and its result is produced.

Optimizer Statistics

The optimizer depends on various statistics to determine an optimal execution plan for a SQL statement.

- Most statistics are gathered automatically and are controlled by statistic preferences.
 - Object statistics
 - Dictionary statistics
- Other statistics are only gathered manually.
 - Statistics on fixed objects
 - Operating system statistics

Optimizer Parameters

Optimizer parameter can be set at:

- The system level
- The session level

You can display the optimizer parameter settings for:

- The System with `V$SYS_OPTIMIZER_ENV`
- The sessions with `V$SESS_OPTIMIZER_ENV`
- A specific plan using `V$SQL_OPTIMIZER_ENV` joined with `V$SQL` or `V$SQLAREA`

Using Hints

Hints

- Are directives (suggestions) to the optimizer
- Require changing the code
- Are useful to test specific access paths
- Must be maintained through upgrades

```
SELECT /*+ INDEX (tablename indexname) */ ...
```

Full Table Scans

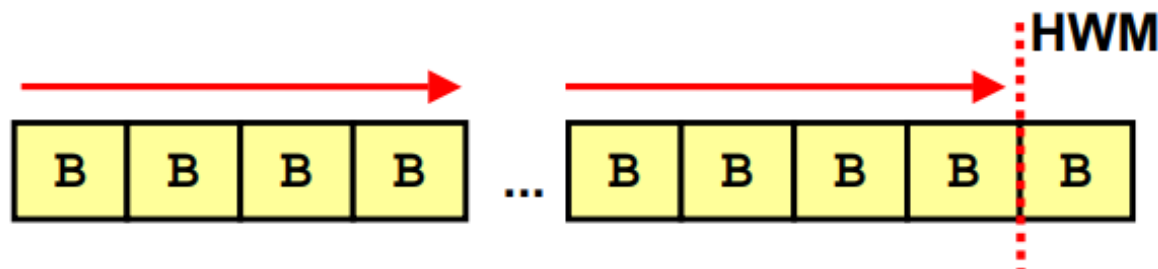
- Lack of index
- Large amount of data
- Small table
- Multiblock I/O calls
- All rows below high-water mark

```
select * from emp where ename = 'KING';
```

Id	Operation	Name	Rows	Bytes	Cost
0	SELECT STATEMENT		1	22	2
* 1	TABLE ACCESS FULL	EMP	1	22	2

Predicate Information (identified by operation id):

```
1 - filter("EMP"."ENAME"='KING')
```



The optimizer uses a full table scan in each of the following cases:

- **Lack of index:** If the query is unable to use any existing indexes, then it uses a full table scan. For example, if there is a function used on the indexed column in the query, the optimizer is unable to use the index and instead uses a full table scan.
- **Large amount of data:** If the optimizer thinks that the query will access most of the blocks in the table, then it uses a full table scan, even though indexes might be available.
- **Small table:** If a table contains blocks fewer than the value of `DB_FILE_MULTIBLOCK_READ_COUNT` under the high-water mark, then a full table scan might be less costly because this can be read in a single I/O call.

Row ID Scans

The row ID specifies the data file and data block containing the row as well as the location of the row in that block.

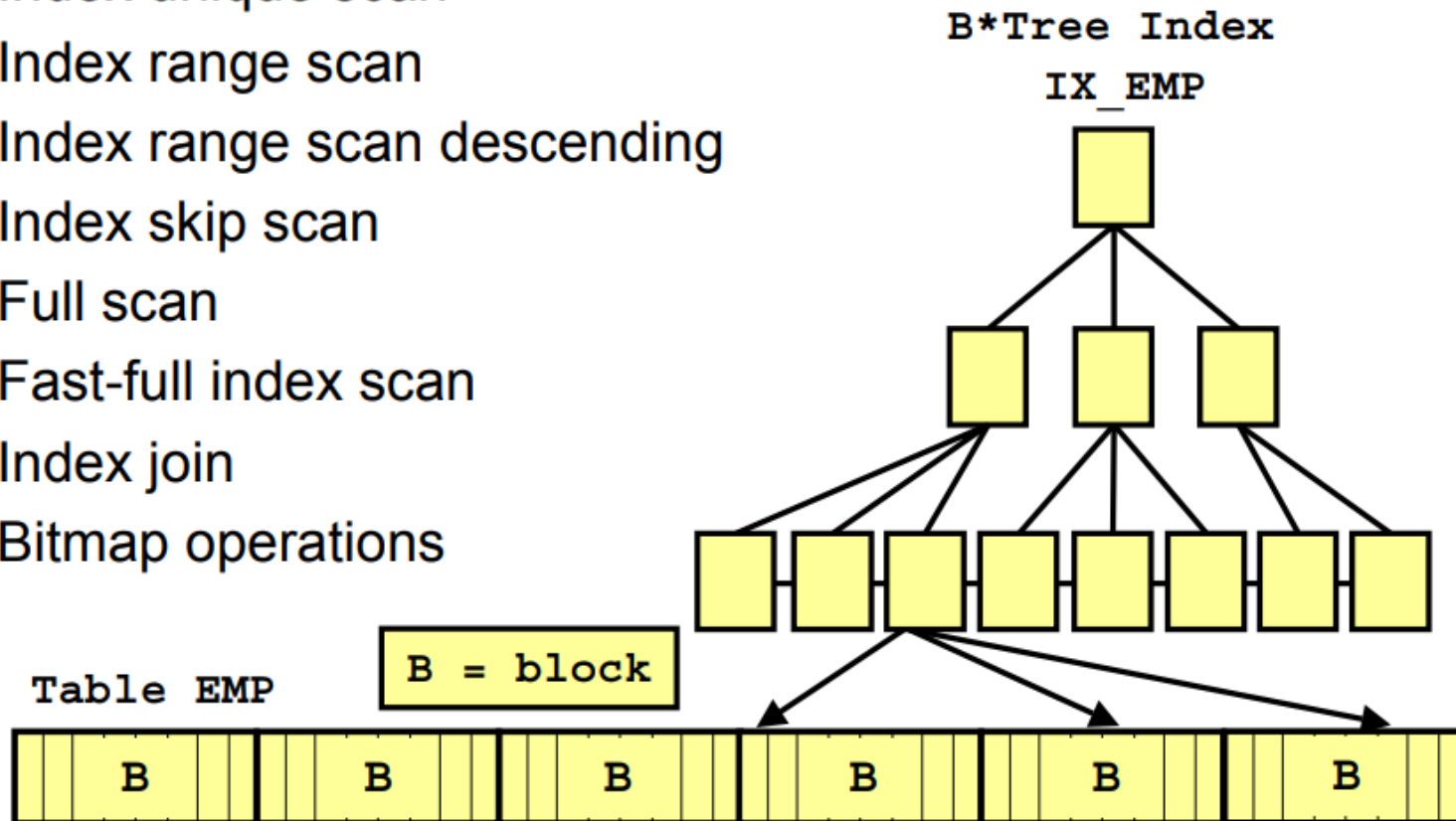
```
explain plan for  
select * from emp where rowid='AAAJ5QAAFABsvAAC';
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		1	1
1	TABLE ACCESS BY USER ROWID	EMP	1	1

Index Operations

Types of index scans:

- Index unique scan
- Index range scan
- Index range scan descending
- Index skip scan
- Full scan
- Fast-full index scan
- Index join
- Bitmap operations



Bitmap Indexes

Key values ↓

BE	1	0	0	0	0	0	0	0	0	0	0	0
DE	0	1	0	0	0	0	0	0	0	0	0	0
FR	0	0	1	1	1	1	0	0	0	0	0	0
IL	0	0	0	0	0	0	1	0	1	0	0	0
NL	0	0	0	0	0	0	1	0	0	0	0	0
UK	0	0	0	0	0	0	0	0	1	1	1	0

← Bitmap for key value FR

The bit for the eighth record is set because the row has the value NL in the COUNTRY column.

Key values ↓

M	1	1	1	0	1	1	0	1	1	1	1	1
F	0	0	0	1	0	0	1	0	0	0	0	0

← Bitmap for key value F

Bitmap Index Access

```
create bitmap index CUST_COUNTRY on CUST(country_iso)
```

```
SELECT CUST_LAST_NAME  
FROM CUST  
WHERE country_iso = 'FR';
```

Id	Operation	Name	Rows	Cost
0	SELECT STATEMENT		2921	368
1	TABLE ACCESS BY INDEX ROWID	CUST	2921	368
2	BITMAP CONVERSION TO ROWIDS			
* 3	BITMAP INDEX SINGLE VALUE	CUST_COUNTRY		

Predicate Information (identified by operation id):

3 - access(COUNTRY_ISO<'FR')

I/O Architecture

Oracle Database 11g includes three standard storage options:

- File system
 - Network attached storage (NAS)
 - Storage area network (SAN)
 - Direct attached storage
- Automatic Storage Management (ASM)

Database File Location for Oracle Database

A database includes several files that store the user data, database metadata, and information required to recover from failures. As an administrator, you decide what kind of storage subsystem to use for these files.

You can select from the following options:

- **File System**—This default option creates database files that are managed by the file system of your operating system. You can specify the directory path where database files are to be stored. Oracle Database can create and manage the actual files.

If you are not certain about which option to use, then select File System (the default).

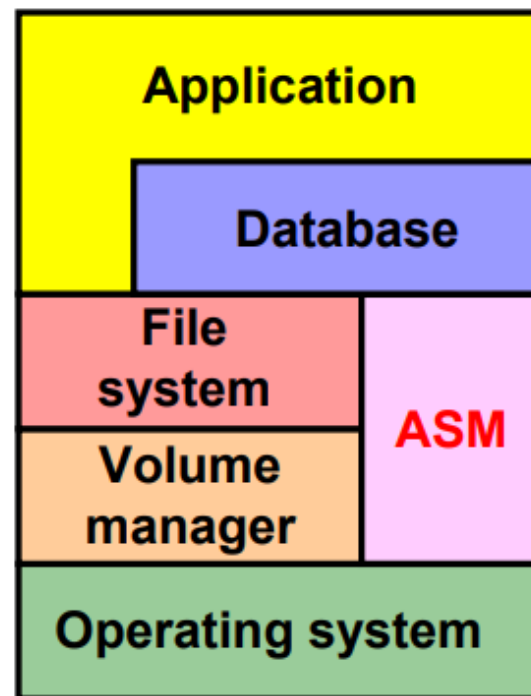
- **Automatic Storage Management**—This option enables you to place your data files in Oracle Automatic Storage Management (Oracle ASM) disk groups. If you choose Oracle ASM, then Oracle Database automatically manages database file placement and naming. For environments with a large number of disks, this option simplifies database administration and maximizes performance. Oracle ASM performs software striping and mirroring at the file level for maximum storage flexibility, performance, and availability.

Oracle ASM uses an Oracle ASM instance, which is distinct from the database instance, to configure and manage disk groups. A single Oracle ASM instance can provide storage for multiple databases on the same server.

What Is Automatic Storage Management?





ASM:

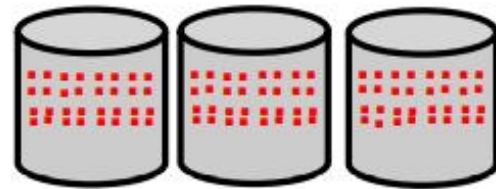
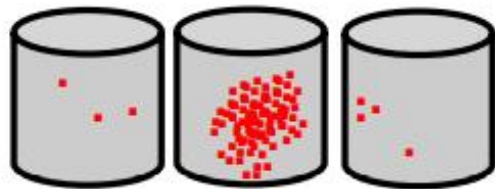
- Is a portable and high-performance cluster file system
- Manages Oracle database files
- Distributes data across disks to balance load
- Provides integrated mirroring across disks
- Solves many storage management challenges
- Encapsulates the SAME methodology



Tuning ASM

- Adjust rebalancing speed
- Set redundancy on a file basis
- Set `DISK_ASYNC_IO` to `TRUE`

Select	Disk Δ	Failure Group	Path	Read/Write Errors	State	Mode	Size (GB)	Used (GB)	Used (%)
<input type="checkbox"/>	DATA_0000	DATA_0000	/dev/xvdc	0	NORMAL	ONLINE	2.00	0.46	 23.14
<input type="checkbox"/>	DATA_0001	DATA_0001	/dev/xvdd	0	NORMAL	ONLINE	2.00	0.45	 22.71
<input type="checkbox"/>	DATA_0002	DATA_0002	/dev/xvde	0	NORMAL	ONLINE	2.00	0.46	 22.80
<input type="checkbox"/>	DATA_0003	DATA_0003	/dev/xvdf	0	NORMAL	ONLINE	2.00	0.46	 22.75



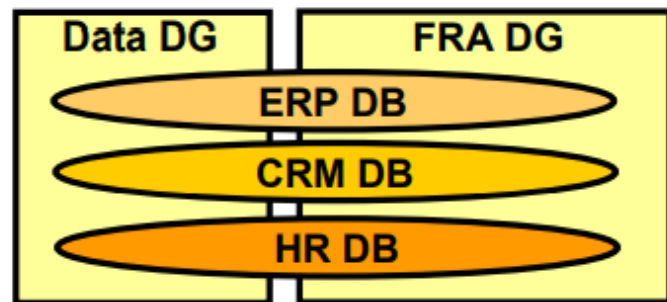
How Many Disk Groups per Database

- Two disk groups are recommended:

- Leverage maximum of LUNs
- Backup for each other
- Lower performance may be used for FRA (or inner tracks)

- Exceptions:

- Additional disk groups for different capacity or performance characteristics
- Different ILM storage tiers



ASM Guidelines

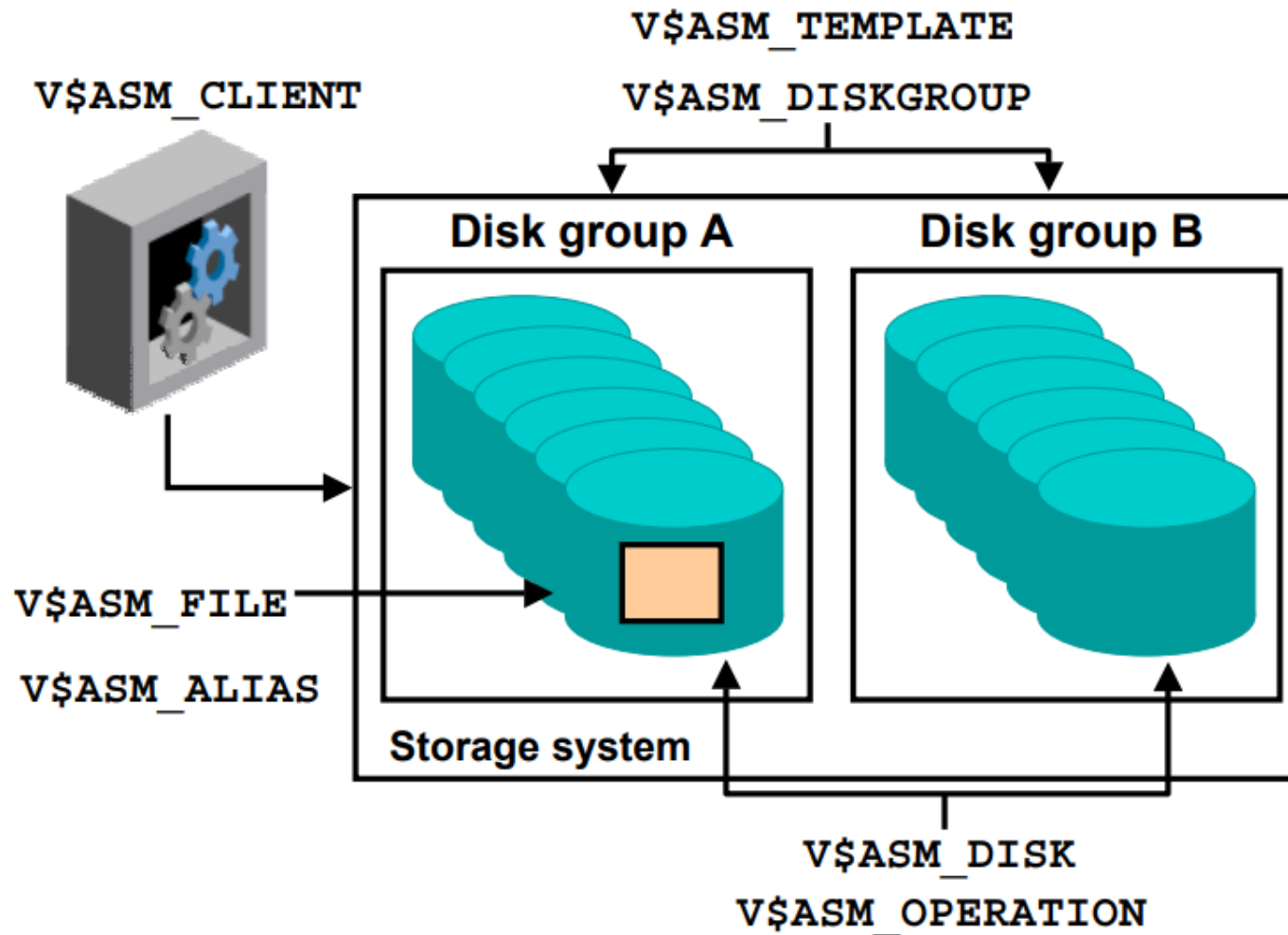
- Use external RAID protection when possible.
- Create logical units (LUNs) using:
 - Outside half of disk drives for highest performance
 - Small disk, high RPM (for example, 73 GB/15k RPM)
- Use LUNs with the same performance characteristics.
- Use LUNs with the same capacity.
- Maximize the number of spindles in your disk group.

ASM Instance Initialization Parameters

- ASM instances have static memory needs.
- Use Automatic Memory Management.
- Using default SGA sizing parameters should be enough for most configurations:

```
INSTANCE_TYPE = ASM
DB_UNIQUE_NAME = +ASM
ASM_POWER_LIMIT = 1
ASM_DISKSTRING = '/dev/rdisk/*s2', '/dev/rdisk/c1*'
ASM_DISKGROUPS = dgroupA, dgroupB
MEMORY_TARGET = 256M /* Default value */
```

Dynamic Performance Views



Dynamic Performance Views

The dynamic performance views that display the ASM information behave differently depending on the type of instance they are accessed from. The following gives the view name and the behavior on an ASM instance and a database instance.

V\$ASM_CLIENT

- ASM—One row for every database instance using a disk group in the ASM instance
- Database—One row for each disk group with the database and ASM instance name

V\$ASM_DISKGROUP

- ASM—One row for every discovered disk group
- Database—A row for all disk groups mounted or dismounted

V\$ASM_TEMPLATE

- ASM—One row for every template present in every mounted disk group
- Database—Rows for all templates in mounted disk groups

V\$ASM_DISK

- ASM—One row for every discovered disk, including disks that are not part of any disk group
- Database—Rows for disks in the disk groups in use by the database instance

Monitoring Long-Running Operations by Using V\$ASM_OPERATION

Column	Description
GROUP_NUMBER	Disk group
OPERATION	Type of operation: REBAL
STATE	State of operation: WAIT or RUN
POWER	Power requested for this operation
ACTUAL	Power allocated to this operation
SOFAR	Number of allocation units moved so far
EST_WORK	Estimated number of remaining allocation units
EST_RATE	Estimated number of allocation units moved per minute
EST_MINUTES	Estimated amount of time (in minutes) for operation termination

ASM Scalability

ASM imposes the following limits:

- 63 disk groups
- 10,000 ASM disks
- 4 petabytes (PB) per ASM disk
- 40 exabytes (EB) of storage
- 1 million files per disk group
- Maximum file size:
 - External redundancy: 140 PB
 - Normal redundancy: 42 PB
 - High redundancy: 15 PB

Important Initialization Parameters with Performance Impact

Parameter	Description
COMPATIBLE	To take advantage of the latest improvements of a new release
DB_BLOCK_SIZE	8192 for OLTP and higher for DSS
MEMORY_TARGET	Automatically sized memory components
SGA_TARGET	Automatic SGA component management
PGA_AGGREGATE_TARGET	Automatic PGA management
PROCESSES	Maximum number of processes that can be started by that instance
SESSIONS	To be used essentially with shared server
UNDO_MANAGEMENT	AUTO mode recommended
UNDO_TABLESPACE	Undo tablespace to be used by instance

Sizing Memory Initially

As an initial guess for memory allocation:

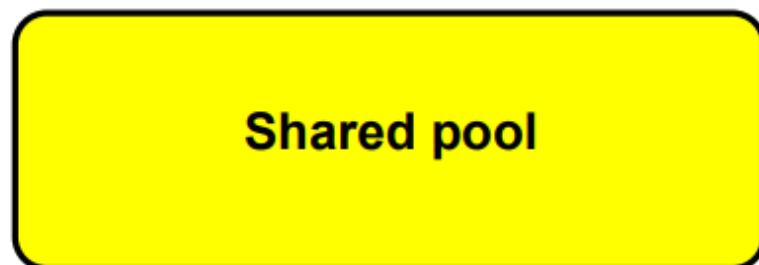
- Leave 20% of available memory to other applications.
- Give 80% of memory to the Oracle instance.

```
MEMORY_MAX_TARGET=(total_mem*80%)
```

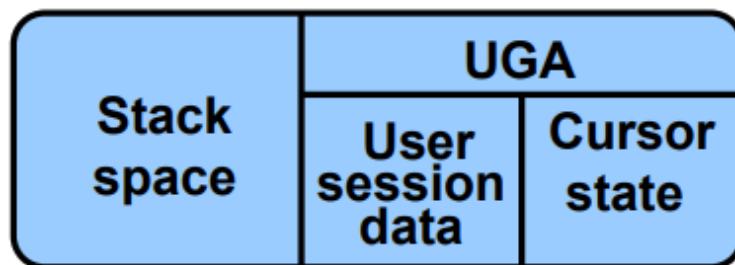
- By default the memory distribution starts at 60% SGA and 40% PGA.
- Memory distribution will adjust to workload.

UGA and Oracle Shared Server

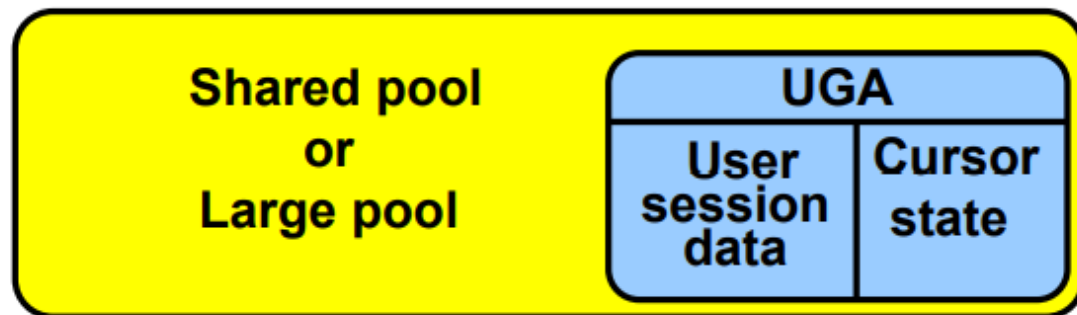
Dedicated server configuration



PGA



Shared server configuration



PGA

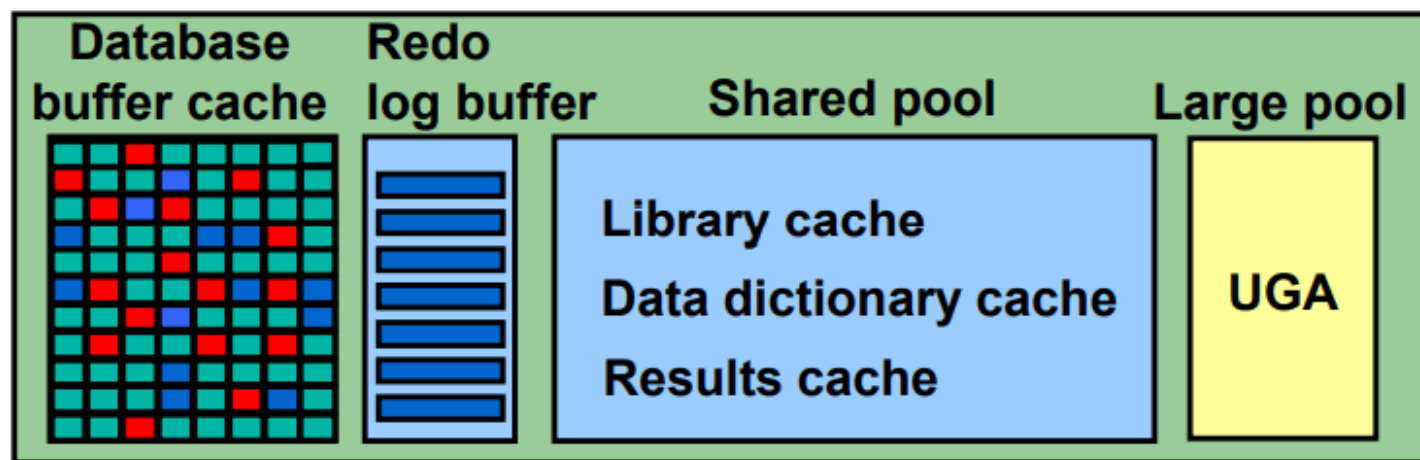


V\$STATNAME
V\$SESSTAT
V\$MYSTAT

OPEN_CURSORS
SESSION_CACHED_CURSORS

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 execution messaging
- Is used to avoid performance overhead caused by shrinking the shared SQL cache
- Is sized by the `LARGE_POOL_SIZE` parameter



Tuning the Large Pool

- The large pool has one parameter, `LARGE_POOL_SIZE`.
- `V$SGASTAT` shows used and free memory.

```
SELECT * FROM V$SGASTAT
WHERE pool = 'large pool';
```

POOL	NAME	BYTES
large pool	free memory	2814112
large pool	session heap	1380192

Undo Tablespace: Best Practices

- Use Automatic Undo Management.
- UNDO_RETENTION:
 - Is automatically adjusted when the UNDO tablespace has AUTOEXTEND enabled.
- Undo tablespace size:
 - Initial size: Small with AUTOEXTEND enabled
 - Steady state: Fix size using the Undo Advisor and add a 20% safe margin.

Temporary Tablespace: Best Practices

Locally managed temporary tablespaces use a uniform extent. Extent size should be:

- 1 MB to 10 MB extent size:
 - For DSS, OLAP applications involving huge work areas
 - Where large temporary LOBs are predominant.
- 64 KB or multiples, less than 1 MB:
 - Small global temporary tables are predominant.
 - OLTP

Temporary tablespace group increases addressability from terabytes to petabytes.

Automatic Statistics Gathering

- `STATISTICS_LEVEL = TYPICAL | ALL`
- Statistics are gathered by the Optimizer Statistics Gathering automatic maintenance task.
- This task implicitly determines the following:
 - Database objects with missing or stale statistics
 - Appropriate sampling percentage necessary to gather good statistics on those objects
 - Appropriate columns that require histograms and the size of those histograms
 - Degree of parallelism for statistics gathering
 - Prioritization of objects on which to collect statistics

Automatic Statistics Collection: Considerations

- You should still manually gather statistics in the following cases:
 - After bulk operations
 - When using external tables
 - To collect system statistics
 - To collect statistics on fixed objects
- Prevent automatic gathering for volatile tables:
 - Lock with statistics for representative values
 - Lock without statistics implies dynamic sampling.
- Set Optimizer Statistic Preferences for objects that need special handling.