Oracle Performance

# Managing Optimizer Statistics

<http://docs.oracle.com/cd/E11882_01/server.112/e16638/stats.htm#PFGRF94712>

## Understanding Stats

Optimizer stats include:

* table stats
  + number of rows
  + number of blocks
* column stats
  + distinct values
  + nulls in column
  + data distribution
* index stats
  + leaf blocks
  + levels

Stored in data dictionary, viewable through:

* DBA\_TABLES / DBA\_INDEXES / DBA\_TAB\_COLS
* DBA\_TAB\_STATISTICS
* DBA\_TAB\_HISTOGRAMS
* DBA\_IND\_STATISTICS

## Automatic Optimizer Stats Collection

AutoTask – automated maintenance task infrastructure

* schedules tasks to run in Oracle Scheduler window (maintenance window)
* by default, one window scheduled every day

Calls DBMS\_STATS.GATHER\_DATABASE\_STATS\_JOB\_PROC

* gather stats on objects
  + with no stats
  + stale stats – more than 10% of rows have changed since stat collection
* similar to DBMS\_STATS.GATHER\_DATABASE\_STATS with GATHER AUTO option
* primary difference – this one prioritizes the DB objects that require stats, so objects which most need updated stats are processed first

### Enabling Automatic Optimizer Stats Collection

Enabled by default to run in all predefined maintenance windows

If disabled, can be enabled with the following:

BEGIN

DBMS\_AUTO\_TASK\_ADMIN.ENABLE(

client\_name => 'auto optimizer stats collection',

operation => NULL,

window\_name => NULL);

END;

/

Disabled in a similar manner:

BEGIN

DBMS\_AUTO\_TASK\_ADMIN.DISABLE(

client\_name => 'auto optimizer stats collection',

operation => NULL,

window\_name => NULL);

END;

/

Relies on modification monitoring feature

* if disabled, job will not be able to detect stale stats
* enabled when the STATISTICS\_LEVEL parameter is set to TYPICAL or ALL

### Considerations When Gathering Statistics

#### When to Use Manual Stats

Since the job runs once a day, stats may become stale on tables that are significantly modified throughout the day. Typically two types of such objects:

* Volatile tables that are being deleted or truncated and rebuilt during the day
* Objects which are the target of large bulk loads which add 10% or more to the object’s total size

Possible approaches for volatile tables

* Set the stats to NULL
  + forces the DB to gather stats as part of query optimization
  + controlled by OPTIMZER\_DYNAMIC\_SAMPLING parameter (set to 2 or higher)
  + stats can be set to NULL by deleting and then locking the stats

BEGIN

DBMS\_STATS.DELETE\_TABLE\_STATS('OE','ORDERS');

DBMS\_STATS.LOCK\_TABLE\_STATS('OE','ORDERS');

END;

/

* Set the stats to values that represent the typical state of it. Gather stats when it has a representative number of rows, then lock the stats.

For tables that are bulk-loaded, stats gathering should be run immediately following the load.

Other scenarios that call for manual stats gathering

* if STATISTICS\_LEVEL is set to BASIC, AOS can’t detect stale stats
* system stats are never gathered automatically
* stats on fixed objects, such as dynamic performance tables, must be collected manually with GATHER\_FIXED\_OBJECTS\_STATS

## Manual Stats Gathering

### Using DBMS\_STATS Procedures

The PL/SQL package DBMS\_STATS lets you generate and manage statistics for cost-based optimization. You can use this package to gather, modify, view, export, import, and delete statistics.

In 10g the COMPUTE STATISTICS clause of the ALTER INDEX statement was deprecated. The DB now automatically collects stats during index creation and rebuild. (<http://docs.oracle.com/cd/E11882_01/server.112/e25494/tables.htm#ADMIN11656>)

* when stats are gathered for a table, column or index, older stats are saved and can be restored if necessary
* use DBMS\_STATS.GATHER\_DICTIONARY\_STATS to gather stats on system schemas – includes SYS and SYSTEM, as well as optional ones like CTXSYS and DRSYS
* when stats are updated, currently parsed SQL statements are invalidated
* the exception is distributed statements

DBMS\_STATS procedures

* GATHER\_INDEX\_STATS
* GATHER\_TABLE\_STATS – table, column and index stats
* GATHER\_SCHEMA\_STATS – all objects in schema
* GATHER\_DICTIONARY\_STATS
* GATHER\_DATABASE\_STATS – all objects in DB

#### Sampling

Used in stats-gathering to estimate stats. If not used, gathering stats requires full table scans Specified using ESTIMATE\_PERCENT argument. Oracle recommends using DBMS\_STATS.AUTO\_SAMPLE\_SIZE for this parameter.

#### Parallel Stats Gathering

The degree of parallelism is specified with the DEGREE argument. Oracle recommends using DBMS\_STATS.AUTO\_DEGREE.

#### Partitioned Objects

For partitioned tables and indexes, DBMS\_STATS can gather stats for each (sub-) partition and global stats for the whole object. This is specified in the GRANULARITY argument. Oracle recommends settings this argument to AUTO.

If the INCREMENTAL value for a partitioned table is set to TRUE, and set GRANULARITY to AUTO, Oracle will only scan those partitions that have been modified. If INCREMENTAL is set to FALSE, then a full table scan is required to maintain global stats. (Setting INCREMENTAL to TRUE consumes additional space in SYSAUX to maintain the global stats.)

#### Column Stats and Histograms

DBMS\_STATS also gathers stats about the data distribution of the columns within the table. The most basic info is the max and min values of a column. This may be insufficient if the data is skewed. Histograms can be created to describe the data distribution.

Use the METHOD\_OPT argument of the gather procedures. Oracle recommends setting it to FOR ALL COLUMNS SIZE AUTO. Then Oracle automatically determines which columns require histograms and the number of buckets of each. You can also manually specify these details.

#### Stale Stats

Oracle monitors activity on tables when the STATISTICS\_LEVEL parameter is set to TYPICAL or ALL. Info about changes to tables can be viewed in DBA\_TAB\_MODIFICATIONS, but there can be a delay in updating the view. Use DBMS\_STATS.FLUSH\_DATABASE\_MONITORING\_INFO to push this info to the view immediately.

### When to Gather Stats

For an application in which tables are incrementally modified, you may only need to gather new statistics every week or every month. Use a script or job scheduling tool to regularly run the GATHER\_SCHEMA\_STATS and GATHER\_DATABASE\_STATS procedures.

## System Statistics

System statistics describe the system's hardware characteristics, such as I/O and CPU performance and utilization. These enable the query optimizer to choose a better execution plan. The stats are collected using DBMS\_STATS.GATHER\_SYSTEM\_STATS. Oracle highly recommends that you gather system stats.

### Workload Stats

Workload stats include:

* single and multiblock read times
* multiblock count (avg multiblock read count sequentially)
* CPU speed
* max system throughput
* avg slave throughput

They depend on the activity the system had during the workload window. If a system is I/O bound, then optimizer promotes a less I/O-intensive plan. Workload stats gathering does not generate additional overhead.

#### Gathering Workload Stats

Perform one of the following:

* Run DBMS\_STATS.GATHER\_SYSTEM\_STATS('start') followed by DBMS\_STATS.GATHER\_SYSTEM\_STATS('stop')
* Run DBMS\_STATS.GATHER\_SYSTEM\_STATS('interval', interval=>N), where N is number of minutes to run

# Oracle Advisors

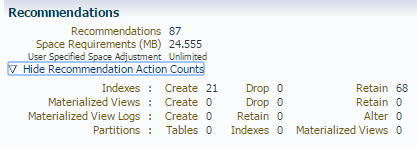
## SQL Advisors

### SQL Access Advisor

Evaluates an entire workload of SQL and recommends indexes, partitioning, and materialized views that will improve the collective performance of the SQL workload. (Doesn't offer suggestions for index rebuilds.)

Source:

* SQL currently in the cache
* SQL Tuning Set
* Hypothetical workload based on schemas and tables



### SQL Tuning Advisor

Analyzes individual SQL statements and recommends SQL profiles, stats, indexes and restructured SQL for improved performance.

### SQL Repair Advisor

Analyzes and potentially patches failing SQL statements.

### SQL Performance Analyzer

SQL Performance Analyzer allows you to test and to analyze the effects of changes on the execution performance of SQL contained in a SQL Tuning Set. **(OEM page is throwing error message in DEV)**

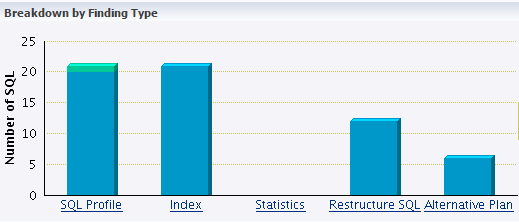
## Automatic SQL Tuning

### Automatic Tuning Optimizer (ATO)

Oracle uses the optimizer to generate execution plans for SQL statements. The optimizer operates in the following modes:

* Normal mode – it compiles SQL and generates an execution plan; generates a reasonable plan for most statements; operates within very strict time constraints
* Tuning mode – it performs additional analysis with an eye toward improved performance; output is not an execution plan but a series of actions; in this mode it's known as Automatic Tuning Optimizer

ADDM identifies high-load SQL statements that are good candidates for tuning. Automatic SQL tuning can also identify problematic statements during system maintenance windows.



#### Statistics Analysis

The optimizer relies on object stats to generate execution plans. If these are stale or missing, the optimizer can generate poor execution plans. The ATO checks each query object and produces two types of output:

* Recommendations to gather stats (only occurs when automatic optimizer stats collection is disabled)
* Auxiliary stats for objects with no stats, and stat adjustment factor for those with stale stats

#### SQL Profiling

A SQL profile is a set of auxiliary info specifici to a SQL statement. Conceptually, it is to a statement what stats are to a table.

#### Access Path Analysis

An access path is the means by which data is retrieved from the DB. Indexes can tremendously enhance performance by reducing the need for full scans of large tables. Effective indexing is a common tuning technique. ATO explores whether a new index can significantly enhance query performance. Because it does not analyze how its index recommendations can affect the entire workload, it advisable to run the SQL Access Advisor as well.

#### Alternative Plan Analysis

While tuning, SQL Tuning Advisor searches for alternative execution plans for the statement. It validates the alt plans and notes any that are not reproducible. You can use one of the alt plans to create a baseline to instruct the optimizer to choose it in the future. The most important statistic is **elapsed time**.

## Managing the Automatic SQL Tuning Advisor

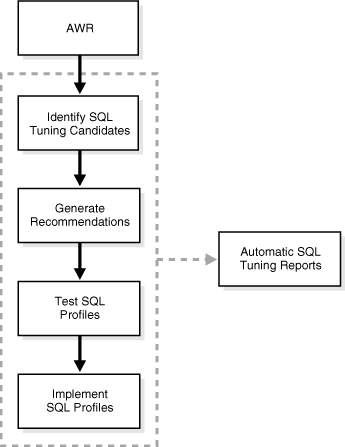
SQL Tuning Advisor takes one or more SQL statements and invokes the ATO to tune them. Output is generated in the form of advice or recommendations, along with a rationale for each and its expected benefit. Recommendations can include the collection of stats on objects, creationg of new indexes, restructuring the statement or creating a new profile. When run automatically, this is know as **Automatic SQL Tuning Advisor**.

### How It Works

By default the DB runs the Automatic SQL Tuning Advisor on selected high-load statements from the AWR. It runs in the default maintenance windows on a nightly basis for at most one hour. These attributes can be changed, including start and end times, frequency and days of week.

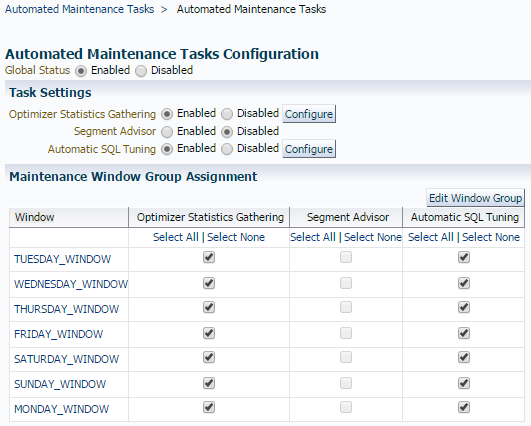
Steps:

1. Identify SQL candidates – The DB analyzes stats in AWR and generates a list of potential statements for tuning. It ignotes recursive SQL and statements that have been tuned in the past month, as well as parallel queries, DML, DDL and statements with performance problems caused by concurrency.
2. Tune each statement by calling SQL Tuning Advisor – The DB considers and reports all recommendation types, but can only implement profiles automatically.
3. Test profiles by executing the statement – If a profile is recommended, the DB tests the statement both with and w/o the profile and reports on its findings.
4. (Optional) Implement SQL profiles – If the recommended profile allows the statement to perform three times better and ACCEPT\_SQL\_PROFILES is set to TRUE, the DB can automatically implement the profile. Other factors may cause this not to happen, such as stale stats.



### Enabling/Disabling Automatic SQL Tuning

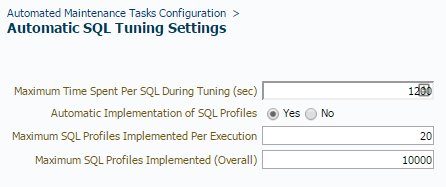
This can be done through the OEM or by using the DBMS\_AUTO\_TASK\_ADMIN package. If the STATISTICS\_LEVEL parameter is set to BASIC, this disables stats gathering by the AWR and SQL tuning.



### Configuring Automatic SQL Tuning

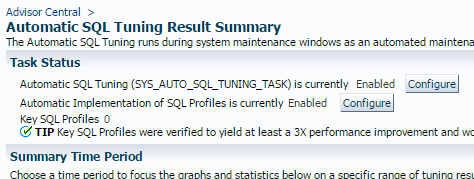
This can be done using either the OEM or the DBMS\_AUTO\_SQLTUNE package.

|  |  |
| --- | --- |
| Parameter | Description |
| **ACCEPT\_SQL\_PROFILE** | Specifies whether to accept SQL profiles automatically. |
| **EXECUTION\_DAYS\_TO\_EXPIRE** | Specifies the number of days for which to save the task history in the advisor framework schema. By default, the task history is saved for 30 days before it expires. |
| **MAX\_SQL\_PROFILES\_PER\_EXEC** | Specifies the limit of SQL profiles that are accepted for each automatic SQL tuning task. Consider setting the limit of SQL profiles that are accepted for each automatic SQL tuning task based on the acceptable level of changes that can be made to the system on a daily basis. |
| **MAX\_AUTO\_SQL\_PROFILES** | Specifies the limit of SQL profiles that are accepted in total. |



### Viewing Tuning Reports

Use the DBMS\_AUTO\_SQLTUNE.REPORT\_AUTO\_TUNING\_TASK function to view reports or check the OEM.



(continued...)

# AWR

## Performance Degradation Over Time

[*http://docs.oracle.com/cd/B28359\_01/server.111/b28275/tdppt\_degrade.htm#TDPPT334*](http://docs.oracle.com/cd/B28359_01/server.111/b28275/tdppt_degrade.htm#TDPPT334)

While an AWR report shows data between 2 snapshots, the Compare Periods report shows the difference between 2 periods (or 2 AWR reports). The report helps identify performance attributes and config settings that differ between 2 time periods.

### Managing Baselines

AWR supports the capture of baseline data by enabling you to specify and preserve a pair or range of snapshots as a baseline. These snapshots are excluded from the automatic purging process. A moving window baseline corresponds to all AWR data that exists within the AWR retention period. Oracle automatically maintains a system-defined moving window baseline. The default size of the window is the AWR retention period.

#### Creating a Baseline

When creating a baseline, carefully consider the time period you choose. It should represent the DB operating at an optimal level. In the future you can compare these baseline with other baselines or snapshots captured during periods of poor performance.

##### Creating a Single Baseline

A single baseline is captured at a single, fixed time interval. For example, March 5 from 5pm to 8pm. These times can be in the future to capture future activity.

##### Creating a Repeating Baseline

A repeating baseline repeats during a time interval over a specified period. For example, a repeating baseline can be taken every Monday from 5pm to 8pm.

# ADDM

[*http://docs.oracle.com/cd/E11882\_01/server.112/e41573/diag.htm#PFGRF026*](http://docs.oracle.com/cd/E11882_01/server.112/e41573/diag.htm#PFGRF026)

The data needed for accurate diagnosis of a problem is stored in the Automatic Workload Repository (AWR). The Automatic Database Diagnostic Monitor (ADDM):

* Analyzes the AWR data regularly
* Diagnoses the root cause of perf problems
* Provides recommendations for correcting problem areas
* Identifies non-problem areas of the system

## ADDM Analysis

An ADDM analysis is performed each time a new snapshot is taken and the results are saved in the DB. This analysis uses the last 2 snapshots as the period of analysis (the last hour by default). One can also be performed on any pair of AWR snapshots for a custom time period. The results can be viewed in OEM or you can run a report from SQL\*Plus.

Analysis is performed top down, first identifying symptoms, then refining them to reach the root cause of the problem. The goal is to reduce a single throughput metric called DB time. This is the cumulative time spent by the DB in processing user requests. It includes wait time and CPU time of all non-idle user sessions. It's displayed in the V$SESS\_TIME\_MODEL and V$SYS\_TIME\_MODEL views.

By reducing DB time, the DB is able to support more user requests, increasing throughput. Problems are sorted by the amount of time they are responsible for. The types of problems that ADDM considers include:

* CPU bottlenecks
* Undersized memory structures
* I/O capacity
* High load SQL statements
* High load PL/SQL execution and compliation
* Sub-optimal use of the DB by the application (problems with poor connection management, excessive parsing or app level lock contention)
* DB config issues
* Concurrency issues

## ADDM Analysis Results

In addition to problem diagnostics, ADDM recommends possible solutions. Each finding can belong to one of the following types:

* Problem findings describe the root cause of a performance problem
* Symptom findings contain info that often lead to one or more problem findings
* Information findings are relevant to understanding the DB's performance, but do not constitute a problem
* Warning findings contain info about problems that may affect the completeness or accuracy of the analysis

Each finding is quantified by an impact that is an estimate of the portion of DB time caused by finding's issue. The types of recommendations include:

* Hardware changes – adding CPUs or changing the I/O subsystem
* DB config – initialization parameters
* Schema changes –partitioning or using ASSM
* Application changes – using the cache option for sequences or using bind variables
* Using other advisors – running SQL Tuning Advisor on high-load statements or Segment Advisor on hot objects

The list can contain various alternatives for solving the same problem. Recommendations are composed of actions and rationales. You must apply all the actions of a recommendation to gain the estimated benefit, but this is not required.

## Tables

DBA\_ADDM\_FINDINGS

DBA\_ADDM\_INSTANCES

DBA\_ADDM\_SYSTEM\_DIRECTIVES

DBA\_ADDM\_TASK\_DIRECTIVES

DBA\_ADDM\_TASKS

# Parallel Execution

http://docs.oracle.com/cd/E11882\_01/server.112/e25523/parallel002.htm

## Parallel Execution of SQL Statements

Each SQL statement undergoes an optimization and parallelization process when it is parsed. If parallel execution is chosen, the following steps occur:

1. The user session takes on the role of the query coordinator
2. The query coordinator obtains the number of necessary parallel servers
3. The SQL statement is executed as a sequence of operations. The parallel servers perform each operation in parallel if possible.
4. When the parallel servers are finished, the query coordinator performs any portion of the work that can't be done in parallel. (e.g. a SUM() operation requires adding the individual subtotals calculated by each parallel server)
5. Finally, the query coordinator returns any results to the user

### Dividing Work Among Parallel Execution Servers

The coordinator examines each operation in an execution plan then determines the way in which the rows operated on must be divided amoung the execution servers. For example, it might perform a full table scan by block range or an index range scan by partition.

SELECT /\*+ PARALLEL(4) \*/ customers.cust\_first\_name, customers.cust\_last\_name,

MAX(QUANTITY\_SOLD), AVG(QUANTITY\_SOLD)

FROM sales, customers

WHERE sales.cust\_id=customers.cust\_id

GROUP BY customers.cust\_first\_name, customers.cust\_last\_name;

--------------------------------------------------------------------------------------------

| Id | Operation | Name | Rows | Bytes | TQ |IN-OUT| PQ Distrib |

---------------------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 925 | 25900 | | | |

| 1 | PX COORDINATOR | | | | | | |

| 2 | PX SEND QC (RANDOM) | :TQ10003 | 925 | 25900 | Q1,03 | P->S | QC (RAND) |

| 3 | HASH GROUP BY | | 925 | 25900 | Q1,03 | PCWP | |

| 4 | PX RECEIVE | | 925 | 25900 | Q1,03 | PCWP | |

| 5 | PX SEND HASH | :TQ10002 | 925 | 25900 | Q1,02 | P->P | HASH |

|\* 6 | HASH JOIN BUFFERED | | 925 | 25900 | Q1,02 | PCWP | |

| 7 | PX RECEIVE | | 630 | 12600 | Q1,02 | PCWP | |

| 8 | PX SEND HASH | :TQ10000 | 630 | 12600 | Q1,00 | P->P | HASH |

| 9 | PX BLOCK ITERATOR | | 630 | 12600 | Q1,00 | PCWC | |

| 10 | TABLE ACCESS FULL| CUSTOMERS | 630 | 12600 | Q1,00 | PCWP | |

| 11 | PX RECEIVE | | 960 | 7680 | Q1,02 | PCWP | |

| 12 | PX SEND HASH | :TQ10001 | 960 | 7680 | Q1,01 | P->P | HASH |

| 13 | PX BLOCK ITERATOR | | 960 | 7680 | Q1,01 | PCWC | |

| 14 | TABLE ACCESS FULL| SALES | 960 | 7680 | Q1,01 | PCWP | |

------------------------------------------------------------------------------------------------

### Parallelism Between Operations

Given 2 sets of execution servers SS1 and SS2 for the above query, each set has 4 execution processes because of the hint. Set SS1 first scans the table CUSTOMERS in parallel and sends rows to SS2, which builds a hash table on the rows. These can work concurrently. After SS1 finishes with the CUSTOMERS table, it scans the SALES table in parallel. It then sends rows to SS2, which performs the probes to finish the hash-join in parallel. SS1 then moves perform the GROUP BY operation.

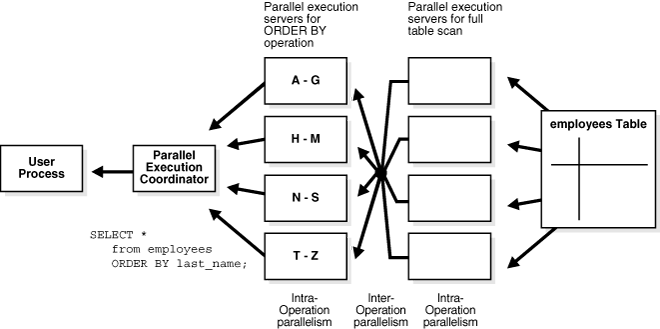
### Producer and Consumer Operations

Operations that require the output of other operations are known as consumer operations. In the above plan the (3) HASH GROUP BY operation is the consumer of (6) HASH JOIN BUFFERED because it requires that output. Consumers can begin working as soon as the producer op starts sending rows.

Each of the 2 operations performed concurrently is given its own set of parallel execution servers. Therefore, both query operations and the data flow tree itself have parallelism. Due to the producer-consumer nature of Oracle ops, only 2 ops in a given tree must be performed simultaneously to minimize execution time.

SELECT \* FROM employees ORDER BY last\_name;

For example, assume the above is run with a DOP of 4, and the LAST\_NAME column is not indexed. The plan needs to perform a full scan of the EMPLOYEES table followed by a sorting of the retrieved rows. The below shows how producer and consumer operations can perform concurrently, thus involving 8 parallel execution servers, even though the DOP is 4.



(continued...)

## Why Doesn't My Query Run in Parallel? (Doc ID 233754.1)

There are 2 parts to running a query in parallel.

1. The optimizer evaluates plans and selects the cheaper one.
   1. A parallel plan is generated if the following conditions are met:
      1. the chosen plan can be run in parallel
      2. it's cheaper when done so
      3. requirements for parallelism are met
   2. Then the DOP is set according to:
      1. multi adaptive user algorithm
      2. restrictions from Resource Manager
      3. restrictions from a User Profile
      4. restrictions from a parallel hint
   3. If the cheapest plan can't be executed in parallel, it will be run in serial (even with a hint)
2. Execution by the runtime engine

## Best Practices: Proactively Avoiding Database and Query Performance Issues (Doc ID 1482811.1)

### Managing Change

#### Collecting and Maintaining Baselines

With regards to proactive avoidance, there are 2 types of baselines you should collect.

1. System-wide "performance" AWR baselines recording the resource usage of the system running normally

If your system is performing well, this is a good time to record that info to form a baseline. The AWR allows you to capture baseline data and enables you to specify and preserve a pair or range of snapshots as a baseline. Those snapshots are excluded from the purging process and are retained indefinitely. Also see *Best Practices: Proactive Data Collection for Performance Issues (Doc ID 1477599.1)*.

1. SQL Plan baselines to maintain plan stability and allow the restoration of a recorded plan in the event of a change

With any change to a system, you to be able to quickly and easily restore the previous performance. The best way to ensure that stability is to use SQL Plan Management (SPM) to ensure operations use known plans and perform the same before and after the change. Also see *Plan Stability Features (Including SPM) Start Point (Doc ID 1359841.1)* and <https://blogs.oracle.com/optimizer/entry/sql_plan_management_part_1_of_4_creating_sql_plan_baselines>

#### Upgrades

Upgrades can introduce significant changes in terms of software and hardware usage. Prior to this ensure that provisions for performance baselines are in place so the new system performs at least as well as the previous iteration.

### Recommended Setup

It is important to use recommended settings as much as possible. The following are part of that:

* Patches – support recommends the latest version and patchset available
* Collect Optimizer Stats – good performance is dependent on the presence of accurate stats on the objects being queried
  + See also *How To: Gather Statistics for the Cost Based Optimizer (Doc ID 1226841.1)*
* Prepare to collect diagnostics – if an issue occurs, collection info about it after the event may prove insufficient for many issues
  + See *Best Practices: Proactive Data Collection for Performance Issues (Doc ID 1477599.1)*

### Monitoring for Potential Problems

#### Use ADDM findings to drive investigation

By default, the DB collects various performance info that can be used to determine the cause of and resolve performance issues. ADDM reports can provide info about potential problems and advice for dealing with them. The DB home page in OEM has a diagnostics section on the left-hand side.

For a good example of the capabilities of the OEM diagnostics and tuning pack, see "[Tuning Toolkit for Advanced DBAs](https://support.oracle.com/epmos/main/downloadattachmentprocessor?parent=DOCUMENT&sourceId=1482811.1&attachid=1380043.1:TUNINGTOOLKIT&clickstream=yes)".

#### Use Real Time SQL Monitoring to Your Advantage

The real-time SQL monitoring feature enables you to monitor the performance of SQL statements while they are executing. By default it automatically starts when a statement runs in parallel, or when it has consumed at least 5 seconds of CPU or I/O time in a single execution.

# Wait Events

| **Wait Event** | **General Area** | **Possible Causes** | **Look for / Examine** |
| --- | --- | --- | --- |
| buffer busy waits | Buffer cache, DBWR | Depends on buffer type. For example, waits for an index block may be caused by a primary key that is based on an ascending sequence. | Examine V$SESSION while the problem is occurring to determine the type of block in contention. |
| free buffer waits | Buffer cache, DBWR, I/O | Slow DBWR (possibly due to I/O?)  Cache too small | Examine write time using operating system statistics. Check buffer cache statistics for evidence of too small cache. |
| db file scattered read | I/O, SQL statement tuning | Poorly tuned SQL  Slow I/O system | Investigate V$SQLAREA to see whether there are SQL statements performing many disk reads. Cross-check I/O system and V$FILESTAT for poor read time. |
| db file sequential read | I/O, SQL statement tuning | Poorly tuned SQL  Slow I/O system | Investigate V$SQLAREA to see whether there are SQL statements performing many disk reads. Cross-check I/O system and V$FILESTAT for poor read time. |
| enqueue waits (waits starting with enq:) | Locks | Depends on type of enqueue | Look at V$ENQUEUE\_STAT. |
| library cache latch waits: library cache, library cache pin, and library cache lock | Latch contention | SQL parsing or sharing | Check V$SQLAREA to see whether there are SQL statements with a relatively high number of parse calls or a high number of child cursors (column VERSION\_COUNT). Check parse statistics in V$SYSSTAT and their corresponding rate for each second. |
| log buffer space | Log buffer, I/O | Log buffer small  Slow I/O system | Check the statistic redo buffer allocation retries in V$SYSSTAT. Check configuring log buffer section in configuring memory chapter. Check the disks that house the online redo logs for resource contention. |
| log file sync | I/O, over- committing | Slow disks that store the online logs  Un-batched commits | Check the disks that house the online redo logs for resource contention. Check the number of transactions (commits + rollbacks) each second, from V$SYSSTAT. |

# Using AWR for Database Tuning: Tips for Expert DBAs

<http://www.oracle.com/technetwork/database/manageability/diag-pack-ow09-133950.pdf>

## Automatic Workload Repository Infrastructure

DB Time

* Total time in database calls by foreground sessions
* Includes CPU, IO and non-idle wait time
* DB Time is NOT response time
* Total DB time = sum of DB time for all active sessions

**Goal: Reduce Total DB Time**

Active Session – Session currently spending time in a database call

Average Active Sessions = (DB Time) / (Wall-clock time)

### Active Session History (ASH)

* Session data sampled at 1 second intervals
* ASH is many-dimensional FACT table
  + Dimensions are V$SESSION columns
  + Fact is that DB time was accumulating over these dimensions

## AWR Reports

|  |  |
| --- | --- |
| Report Name | SQL Script |
| AWR Report | awrrpt.sql |
| ADDM Report | addmrpt.sql |
| ASH Report | ashrpt.sql |
| AWR Diff Periods Report | awrddrpt.sql |
| AWR Single SQL Statement Report | awrsqrpt.sql |

### Reading an AWR Report

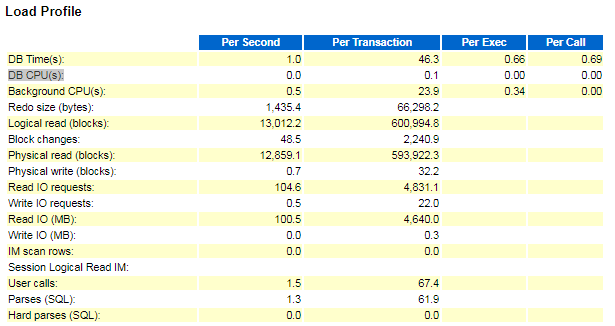
* Check corresponding ADDM report for actionable recommendations
* ADDM analyzes data after an AWR snapshot
* ADDM makes specific performance recommendations
* ADDM also tells you what is NOT a problem

# 10 Steps to Analyze AWR Report

<http://www.dbas-oracle.com/2013/05/10-steps-to-analyze-awr-report-in-oracle.html>

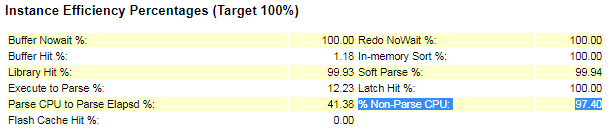
## Load Profile

* If the **Per Second** column for **DB CPU(s)** is greater than the number of cores, the DB is CPU bound.
* If the ratio of **Hard parses** to **Parses** is high, look at parameters like cursor\_sharing and application level for bind variables etc.



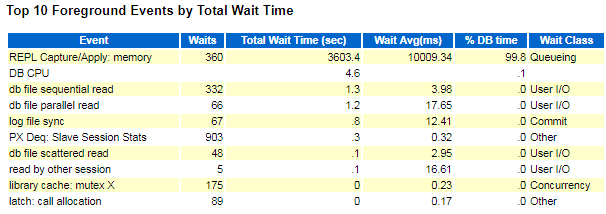
## Instance Efficiency Percentages

* If **% Non-Parse CPU** is close to 100%, most CPU resources are used for operations other than parsing. This is good for DB health.



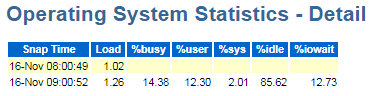
## Top *x* Foreground Events

* If the **Wait Class** column shows **Concurrency**, there could be a problem.
* If both **Time(s)** and **Avg wait (ms)** are high for a row, you should investigate further. If **Avg wait (ms)** is low, you can ignore it.



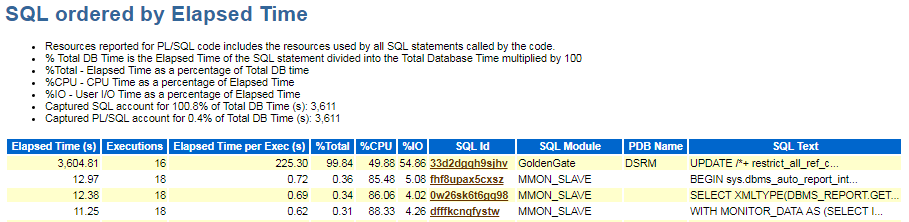
## Operating System Statistics – Detail

* If the %idle column is very low, this may indicate a resource crunch on the system.



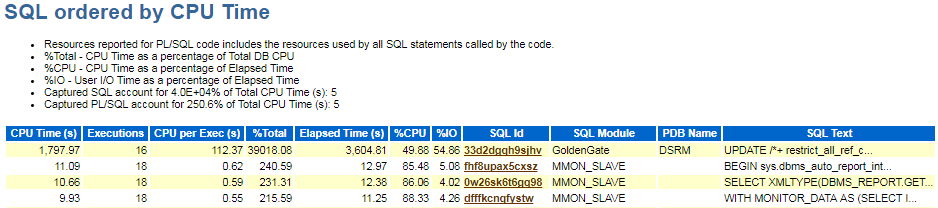
## SQL Ordered by Elapsed Time

* Look for statements with low Executions but high Elapsed Time per Exec (s) and check for tuning opportunities.
* If Executions is 0, the query might still be running when the report was taken.



## SQL ordered by CPU Time

* The above also apply for this section.



# Troubleshooting DB Performance issues with AWR

<https://support.oracle.com/epmos/main/downloadattachmentprocessor?parent=DOCUMENT&sourceId=1597373.1&attachid=1456176.1:94&clickstream=yes>

AWR is an interval sampling tool that averages data for all sessions. ASH reports timed events, by session, at one second intervals. It can identify session bottlenecks immediately.

## DB Time

AWR and ASH construct DB time to help identify if the bottleneck is in the DB or elsewhere. For each session Oracle measures how much time it spends in the DB, either working (consuming CPU) or waiting for a resource. Thus, if the response time of a query suddenly increases but the DB time remains the same, we know the problem exists outside the database.

DB Time is the sum of CPU and wait times for all foreground sessions. The time outside the database is referred to as idle time. Rather than meaning nothing is happening, “idle” means there is no active request to be processed by the DB server. The most common idle event is “sql\*net message from client”.

Cores are physical CPUs and determine the absolute limit to CPU capacity. Although literature for multithreading suggests in increases CPU capacity, it does not. Unfortunately, AWR and ASH report thread count as CPU count. The core count is used in this analysis.

## Load Profile

DB CPUs Per Second is the amount of time spent by all foreground sessions for each wall clock second. Dividing this by the number of cores tells you how busy each core is on average. If that value is approaching one, that represents high CPU usage.

A parse is the effort of looking up a cursor or other executable-related object in the cursor (library) cache. As long as a cursor is in an executable state, a new session can simply “pin” it, then execute it without recompilation. This is a “soft parse”. These are less time consuming than hard parses but are still not free.

## Instance Efficiency Percentages

Even a DB with a near 100% buffer hit ratio can have bad performance. For instance, a low Execute to Parse % can indicate a poorly written application that has to parse a cursor each time it wants to execute it. Ideally, a cursor will be parsed one time by the application then executed many times before it is closed.

High Parse CPU to Parse Elapsed % can indicate optimizer problems. % Non-Parse CPU means all other uses of CPU other than parsing. This includes PL/SQL execution, buffer reading and manipulation and looking up database block buffers in the buffer cache.