**Oracle 12c Database Performance Tuning (Ahmed Barakha) – Notes**

Broadly, performance tuning of Oracle Database can be divided into three major catagories:

1. **Performance Tuning Tools** (Tools used by DBAs in performance tuning such as V$ views, utilities executed at OS level, facilities called from Database or separate products used to assist in performance tuning.
2. **SQL Tuning** (not covered here).
3. **Instance Tuning** (It involves performance tuning activities of the database structure components such as SGA, DISK I/O Background processes and Client sessions. It also involves tuning the Operating System).

**Performance Issue Attributes**:

**Scope**: Entire System or part of it (specific functionality, session, user, batch, etc.). The entire Database can become sluggish when external processes consumes most of the CPU power or there is issue in Disk I/O system. An example of partial low performance is an order confirmation that takes two minutes to return to the user. Another example could be an ETL process is a datawarehousing database. In the past it was taking 2 hours to finish but now it takes 12 hours to finish.

**Slow or Hang**: If the issue is slow response time or hanging of the sessions.

**Permanent or Interminent**: If the issue is permanent or interminent. Interminent issue is more diffucult to diagnose.

**Reproducible or Not**: Reproducible issues are easier to trace and diagnose.

Specific Errors: Most errors are not considered as performance issue. However, some of them are. For example, ORA-00257 (Archive Error) indicates that the Archive log destination is full. The database hangs and new connections are refused. Another example is ORA-04031 which indicates that the Shared Pool memory of SGA is full.

**Conclusion**: Before we diagnose performance issue we must determine the attribute of the performance issue.

**Performance Tuning Methodology**:

Performance tuning covers every aspect of the IT stack hence there is no single or hard-line procedure to implement. Performance tuning procedures are not one-size-fit-all. For this reason, to diagnose a performance tuning issue we should a **Methodology**, not a procedure. A Methodology is a general strategy that oulines the way we follow to diagnose performance issue. A DBA can follow a different steps or procedures to diagnose and troubleshoot and resolve a performance issue from another DBA yet they follow the same methodology and both of them are able to resolve the issue successfully.

A DBA should implement the methodology only after gathering information about the performance issue attribute.

**Oracle recommended Peformance Tuning Methodology**:

1. **Set a measurable target**: For example, if the issue is about a slow report, ask how it used to take before and how long it takes now. The time that the report used to take normally is your target.
2. **Discover the Symptoms**: The second step is to gather information about the symptoms. Here the symptoms do not refer to those experienced by the end user. We mean the technical symptoms gathered by the DBA. To perform this task, you may have to generate an AWR report, a comparison AWR report to point out the symptoms. This is true when the issue is database system wide and impacts significant number of users. The other tools used may be the V$ views, ADDM, database Alert logs, SQL tracing and OS utilities. This is the most important step as everything that comes next relies on the information gathered in this step.
3. **Determine possible causes**: Make a list of possible causes. For uncommon causes you may have to take help from Oracle support.
4. **Develop a Trial solution**: Develop a trial solution. This can be as simple as creating an index or following an ADDM recommendation.
5. **Testing the Solution**: Testing the solution in a test environment before implementing it into a Production environment in order to avoid a worse implication in the Production environment. Testing environment must be same as the production environment and the workload should be should be same or closest possible to the production workload causing the issue. To reproduce the same workload can be performed using **Oracle Replay** database tool.
6. **Check the outcome (Pass)**: If the target is hit, the solution can be implemented. Do not try to tune the process further. Enhancing a process performance is a proactive developmental process and should be done seperately.
7. **Check the outcome (Fail)**: Go back to step number three and repeat the steps in order to come up with a different solution.

**Please note** that this methodology does not require you to stop the database service. As per Oracle support analogy, this methodology is applicable for issues with severity 4 or less. But if the issue makes the database unavailable then it’s considered an emergency issue and you don’t have time to test the solution in a testing environment before implementing. For example, in case Archive log is full (ORA-00257) you need to take action on the Production system by freeing up the disk space.

**Typical System Workload Example of an OLTP system**:

It’s important for a DBA to be able to describe the workload of a system.

|  |  |
| --- | --- |
| **Dimension** | **Value** |
| Number of Users | 1000 |
| Transaction rate | 1 transaction per 5 minutes per user. |
| Business hour | 8 hours in 5 working days (5\*8) |
| Calculated rate | 20 transaction per minute, 9600 transaction daily |
| Peak rate | 120 transaction per minute |

**Oracle Performance Tuning Tools**:

1. **V$ views and Database Dictionary views**:. V$ views are maintained in memory (SGA) whereas Database dictionary views are stored on the disk. Consequently, V$ views figures are reset at every instance restart.
2. Automatic Workload Repository or AWR
3. Statspack reports
4. Automatic Database Diagnostic Monitor (ADDM)
5. Performance Tuning Advisor
6. Alert log files
7. Trace files
8. Enterprise Cloud Control and EM Database Express

**Types of Performance Tuning Statistics**:

Oracle provides three kinds of performance statistics:

1. **Cumulative Statistics**: They have no meaning without the time factor.
2. **Metrics**: Statistic rate.
3. **Sampled statistics**: ADDM, AWR, Active Session History (ASH).

**Cumulative Statistics**: These statistics are gathered for a long list of events in the database. Oracle starts counting the cummulative statistics since the last instance startup. When the instance restarts startiscts are reset. Therefore they have little value until they’re compared over time. For example, if we know the instance has spent one hundred minutes performing I/O operations, does that help on anything? Is it too little or too much? We cannot tell without the time factor. However, some of the Cummulative statistics are useful in performance diagnostics. For example, transaction in queue waits should be barely seen in a healthy system. Example of Cumulative statistics are **Time Model** and **Instance Wait Events**.

A **cumulative statistic** is a count such as the number of block reads. Oracle Database generates many types of cumulative statistics for the system, sessions, and individual SQL statements. Oracle Database also tracks cumulative statistics about segments and services. Automatic Workload Repository (AWR) automates database statistics gathering by collecting, processing, and maintaining performance statistics for database problem detection and self-tuning purposes.

**Metrics**: Metrics is the rate of change in accumulates statistics. You can measure these rates in a variety of units including time, transactions or database calls, for example, number of database calls per second is a metric. Metrics are the basic DBA tools to discover performance issue before they occur in the database. A DBA can set a threshold on a metric so that alerts are issued when the threshold are exceeded, for example, a threshold for when an Archive log is 90% full. The main use of Metrics is they’re used by various Oracle tools to provide detailed information about specific performance area over a given time period.

**Sampled Statistics**: Sampled statistics are saved as snapshots. Saving the statistics allows you to look back in time into statistics and diagnose what happened in tha past. ADDM, AWR, Active Session History (ASH) are called Sampled Statistics.

For example, by default, the database gathers statistics every hour and creates an **AWR snapshot**, which is a set of data for a specific time that is used for performance comparisons. The delta values captured by the snapshot represent the changes for each statistic over the time period. Statistics gathered by AWR are queried from memory. The gathered data can be displayed in both reports and views.

**Setting Initialization Parameter and Statistics Level**:

Database initialization paremeters **STATISTICS\_LEVEL** and **CONTROL\_MANAGEMENT\_PACK\_ACCESS** are relevant for AWR. STATISTICS\_LEVEL must be set to TYPICAL or ALL to enable statistics gathering by AWR.

Setting CONTROL\_MANAGEMENT\_PACK\_ACCESS to DIAGNOSTIC+TUNING or DIAGNOSTIC enables automatic database diagnostic monitoring or ADDM. The STATISTICS\_LEVEL can be verified in V$STATISTICS\_LEVEL dynamic performance view.

**Time Model**:

**Idle Wait**: When Oracle server is up and running but not receiving any requests from clients, the database is practically in an idle state. The server engine must be busy in running some background processes but from user point of view it’s idle. During idle state, Oracle server registers in its data dictionary that it’s waiting for events of a class named **Idle Wait**. This event means database server is waiting for client requests.

**DB CPU and Wait Time**: When database server serves client requests it consumes system resources such as CPU, memory and disk storage. The time database spends on executing client requests is called DB CPU time or DB CPU. Since system resources are shared across many processes and client sessions, the database may find the resource busy and wait for it to become free. The time database spends in waiting for database resource is called **DB Wait**. The actions of waiting for specific resources are called wait events.

**DB Time**: The time the database spends on executing and waiting for system resources is collectively called DB Time. DB Time = DB CPU + Total DB Waits. This effectively means the time database spends on serving the client requests. DB Time is calculated by aggregating the CPU time and wait time of all **active sessions** (sessions that are not idle).

The set of statistics that describe where the database time (DB Time) has been spent inside the Oracle database is called **Time Model**. Oracle classifies DB Time into **statistics group** with each group representing the time spent on specific operation type. In Oracle 12c the Time Model has nearly **20 groups** such as SQL execution elapsed time, Soft and Hard parse time, PL/SQL execution elapsed time, etc.

Time Model statistics are retrieved from V$ views: v$sys\_time\_model and v$sess\_time\_model. The first one is a **system time model statistics** and the second one is **session time model**.

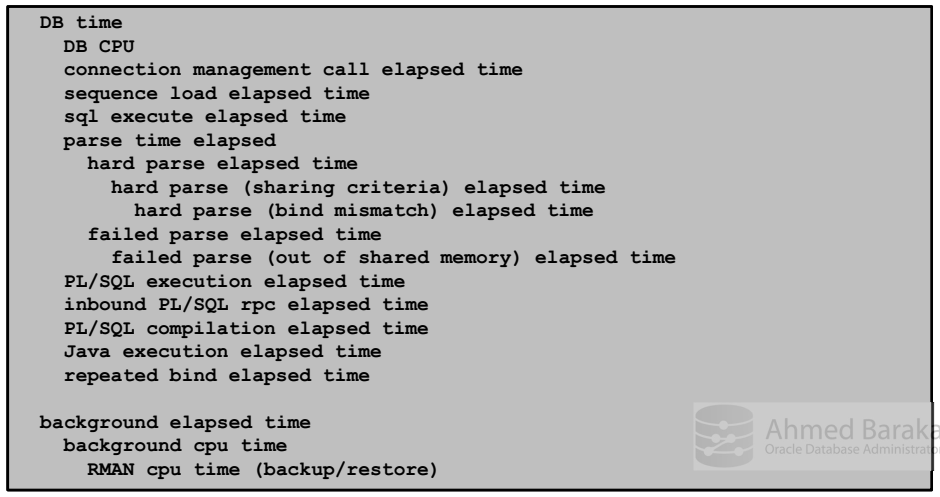
System Time model view displays the system-wide accumulated time for various operations. The statistics are accmulated since the instance startup and reset at instance restart. Session time model view displays the session level accumulated time for various operations.

**Note that** DB time in a specific time period can be longer than that specific time period. This is because DB time is the aggregation of all CPUs and wait time of all active sessions. Therefore, having multiple CPUs and multiple users waiting for the same resources will all be counted in the DB Time. For example, if we one CPU, for any 1 minute given in real time, we have 8 minutes of CPU time available.

Remember the following points about Time Model:

* If you want a single indicator which tells how much a Database is busy then DB time is the indicator.
* If the toal wait time represents high percentage of of the total DB Time (like 50%) then that is this is an indication of a poorly performing system because most of the DB Time was spent on waiting rather than doing processing data at the CPU.
* DB Time and Time Model helps you to measure the impact of a change on the entire database performance. For example, you increase the database server memory and want to check whether that really helps the database performance or not. This also helps you to measure the impact of Database upgrade on Database performance.
* Time Model tells you how much time is spent on each Operation type. You may notice that a specific operation type takes unnecessarily high percentage of the total DB Time. This may help in examining the system and zero in on a bottleneck of the system. For example, you may notice that high percentage of time is spent on hard parses. This is not a good sign for a healthy database system.

**Time Model Statistics Hierarchy**:



The hierarchy has two trees. At the top of the one is the DB Time. At the top of the other is the Background Elapsed time. **It’s important to know** that Background elapsed time is not counted in DB Time because the DB Time counts only the Foreground processes. Background time is included in the model for only one reason, that is to report the time consumed by the RMAN utility. RMAN is a background process and cosumes lot of resources.

The reported by a child in the tree is contained within its parent in the tree but the Parent’s figure does not necessarily be the total of its children.

The total Wait Time can be obtained from the Time Model by substracting the DB CPU value from the DB Time value.

**Historical System Time Model Statistics**:

By defualt, Oracle database server takes a snapshot every hour and saves into a view called **DBA\_HIST\_SYS\_TIME\_MODEL**. The best way to view the historical Time Model statistics is to generate a AWR report.

**Instance Activity and Wait Events**:

Time model provides statistics about Total Database Operations and Wait time. There are two types of views which provides more details about the Total Database Operations and Wait time. They are the Instance Activity views and Wait events views.

**Instance Activity views**:

The view that provides instance activity statistics at the instance level is called V$SYSSTAT. The view structure is simple. It has Statistic names and their values. Each statistic name can belong to one or more from 8 statistic classes. A system statistic class may have more than one system statistics. Their names are described in V$STATNAME.

Instance activity statistics can be obtained at the Service, Session and Segment level.

**Service** level: V$SERVICE\_STATS

**Session** level: V$SESSTAT and V$MYSTAT: V$MYSTAT lists the instance activity statistics of the current session.

**Segment** level: V$SEGMENT\_STATISTICS: This view not only provides instance activity statistics but also wait time on database segments.

These views, except Segment statistics, has only one drawback. They don’t have the statistic names. We have to join these view with V$STATNAME.

**System statistic Class**:

There are 8 system statistic **additive** classes. Each class is assigned a number and a name. For example, 1 – User, 2 - Redo, 4 – Enqueue, etc. A statistic can belong to more than one class. If a statistic class ID is 72, it means it belongs to two classes, one class is Cach class (8) and other one is SQL class (64). Statistic class is present only in V$SYSSTAT and V$STATNAME views. It’s not present in V$SESSTAT or V$MYSTAT view.

We need to understand that when we view statistics from system statistic view (V$SYSSTAT), the statistics does not make much sense unless they are compared with the previous baseline statistics. That’s the reason Oracle takes a snapshot of this view with the AWR. Using AWR report we can compare the statistics taken in two difference time and that makes performance tuning job easier.

**Wait Events**:

Wait event statistics are incremented when the process should wait for a resource to be available before it can continue its processing. Wait event statistics can be obtained from V$SYSTEM\_EVENT at system level and from V$SESSION\_EVENT at session level and V$SERVICE\_EVENT at service level. A full list of wait events can be obtained from V$EVENT\_NAME. There are more than 1500 wait events distributed across 12 classes (explained in Oracle Database reference guide). Wait events are logged when a process is waiting for lock, latch or I/O operations to complete. Wait events are symptoms, not the actual cause of the problem.

Some Wait events are accompanied by the parameters, for example, buffer busy wait wait event has three parameters (that’s the maximum number of parameters an wait event can have): file number, block number and class number. The parameters are listend in V$EVENT\_NAME.

**Wait Class**:

Wait events are classified by Wait classes. But Wait classe numbers are not cumulative as Statistic classes. Each Wait event belongs only one and one Wait class.

**Some of the most important Wait Classes**:

**Application**: Includes Lock waits caused by row-level locking or explicit LOCK command.

**Administration**: Includes the DB administration commands that cause other users to wait, such as INDEX REBUILD.

**Cluster**: Waits related to Oracle RAC resources (for example, global cache resources such as 'gc cr block busy').

**Commmit**: Includes for redo log confirmation after a commit. There is only one wait event, 'log file sync'.

**Concurrency**: Includes a set of Waits resulted from concurrent parsing, buffer cache lock and lock contention.

**Configuration**: Includes waits resulted from the undersized log buffer space, log file sizes, buffer cache size, shared pool size or High-water mark enqueue contention.

**User I/O**: Includes waits for blocks to be read off the disk by foreground processes.

**System I/O:** Waits for background process I/O (for example, DBWR wait for 'db file parallel write').

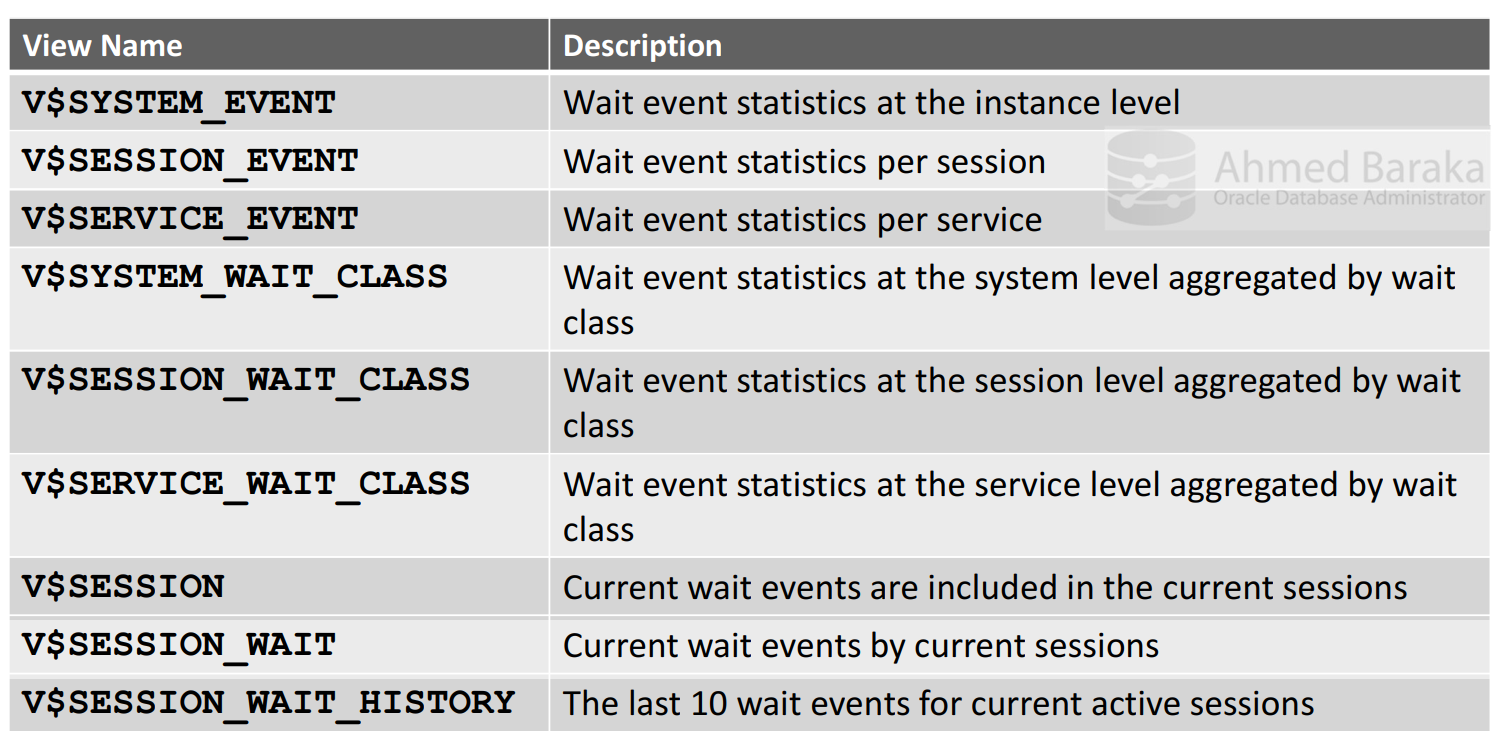
**Network**: Includes wait for data to be sent over the Network.

**Scheduler**: Resource Manager related waits (for example, 'resmgr: become active').

**Idle**: Includes waits for inactive sessions such as “SQL\*Net message from client”. It means the DB session is nothing for the Application users. It’s just waiting for the client to send their calls.

**Queue**: Contains events that signify delays in obtaining additional data in a pipelined environment. The time spent on these wait events indicates inefficiency or other problems in the pipeline. It affects features such as parallel queries or DBMS\_PIPE PL/SQL packages.

**Displaying wait even statistics**:

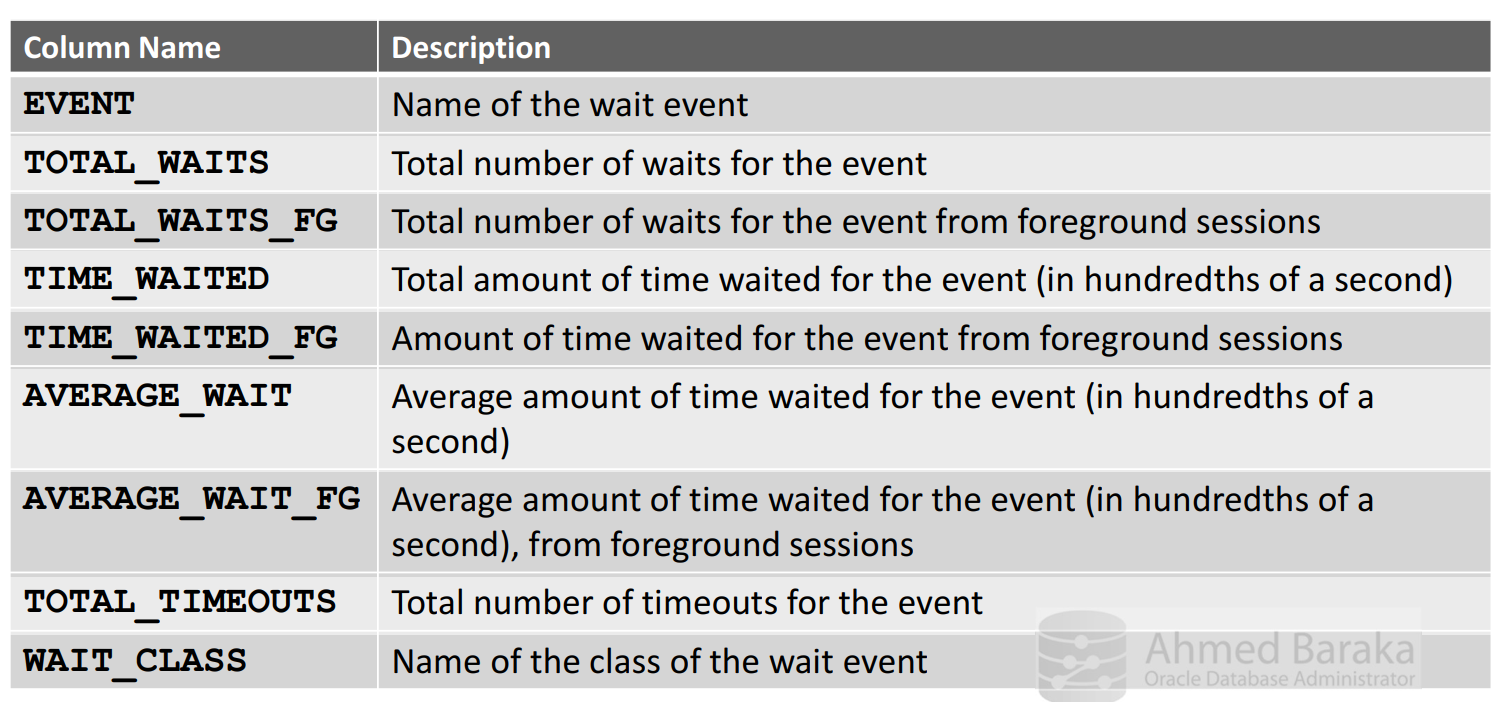


V$SESSION view displays wait events held by the current view, if there is any. It displays the name of the wait events together with their parameters. Remember that it shows the current wait events, not the accumulated wait events for the entire life of the current session. Total wait time per session is obtained from the V$SESSION\_EVENT.

V$SESSION\_WAIT displays the the current or last wait for each session.

V$SYSTEM\_EVENT, V$SESSION\_EVENT, and V$SERVICE\_EVENT displays cumulative and aggregate wait event statistics such as Total waits, Total timeouts, Time waited, etc.

**Common Columns in V$\*\_EVENT Views**:

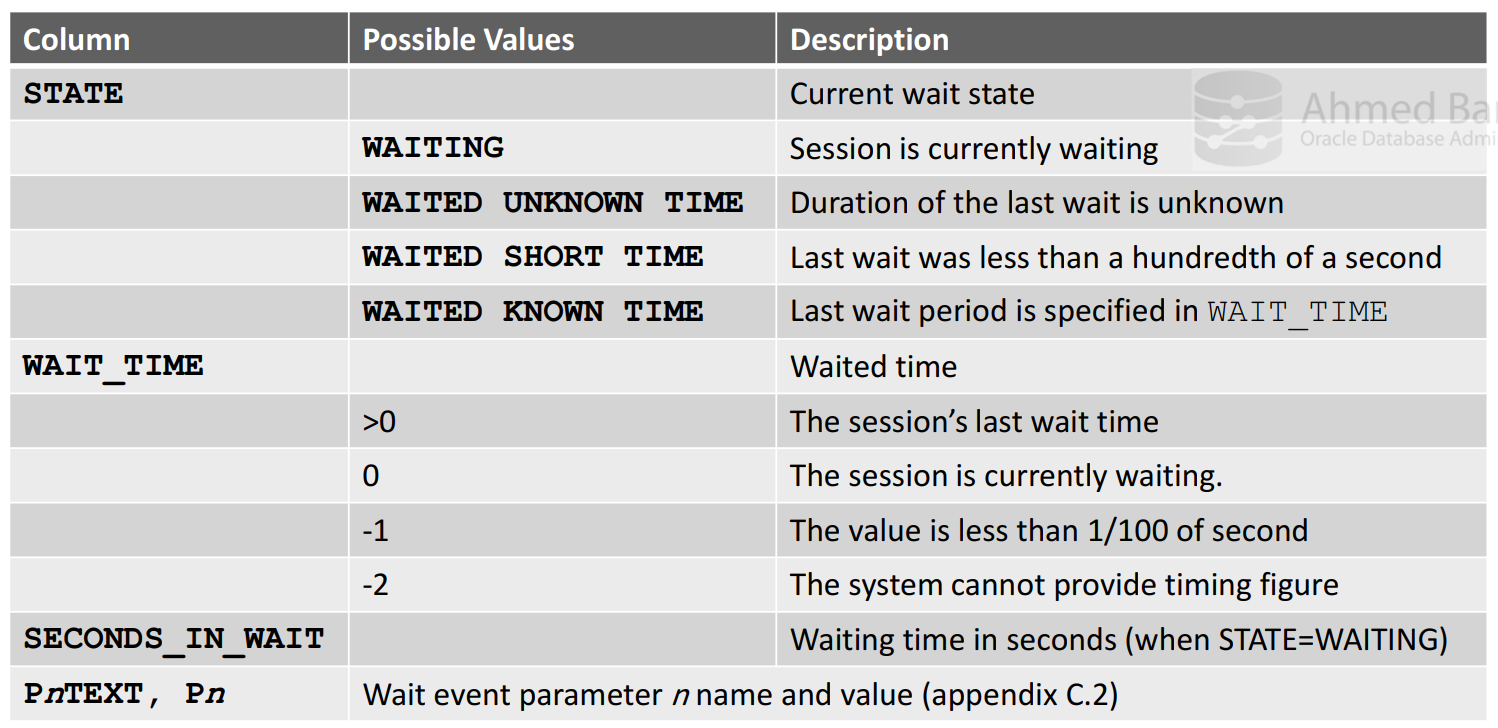


**TOTAL\_WAITS**: It’s the total number of waits for the event, not the Total waiting time.

**TIME\_WAITED**: Total amount of time waited for the event (in 100th of a second).

**AVERAGE\_WAIT**: Average amount of time waited for the event (in 100th of a second).

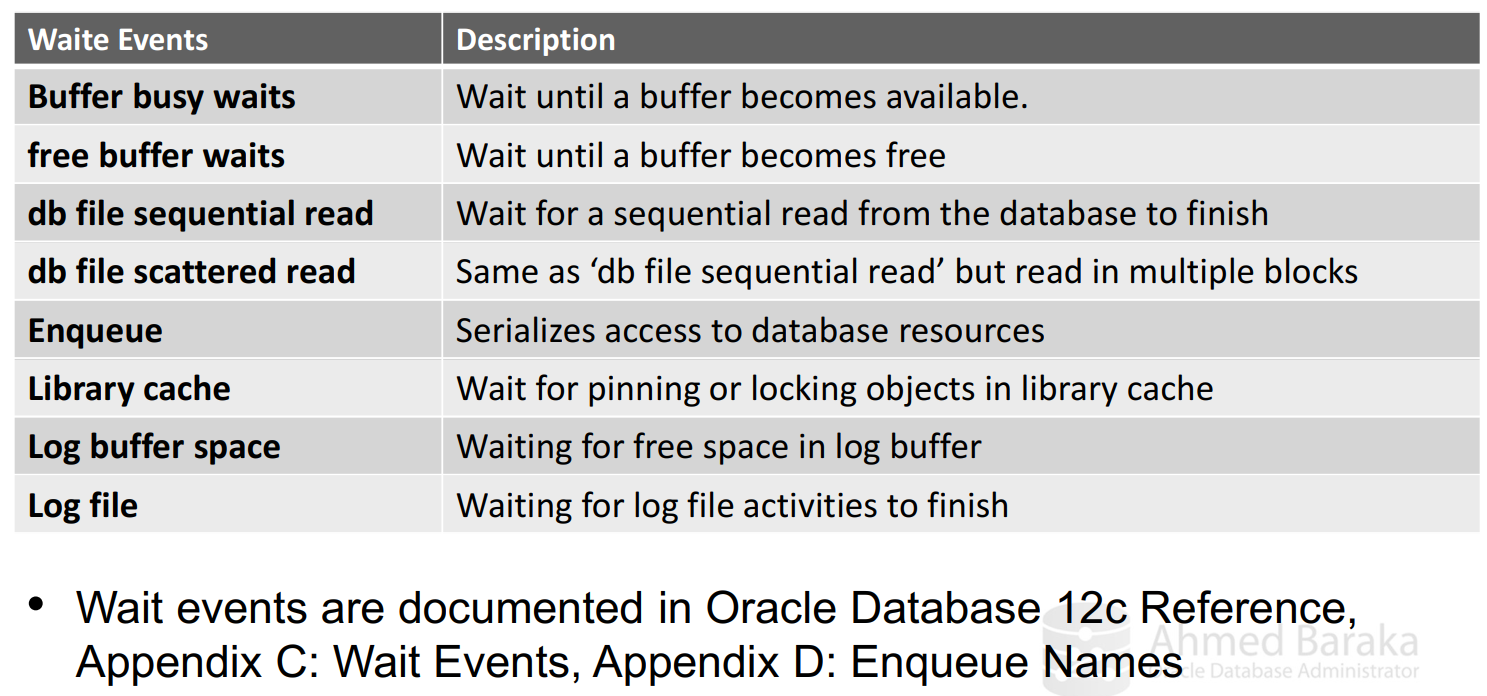
We can check the V$SESSION\_WAIT and V$SESSION view to know the waiting status of sessions. We need to check the STATE, WAIT\_TIME (deprecated in 19c), SECOND\_IN\_WAIT, PnTEXT, Pn and WAIT\_TIME\_MICRO or SECONDS\_IN\_WAIT. TIMED\_STATISTICS parameter should be set to TRUE or the STATE column value would show WAITED UNKNOWN TIME and WAIT\_TIME -2.



**WAIT\_TIME\_MICRO** column shows amount of time waited (in microseconds). If the session is currently waiting (STATE = WAITING), then the value is the time spent in the current wait. If the session is currently not in a wait, then the value is the amount of time waited in the last wait.

**PnTEXT and Pn** will show the Wait event parameter and their description.

**Most Common Wait Events (Non-RAC Database)**:



Wait Events should be looked into AWR reports or V$SESSION\_EVENT view joined with V$SESSION view (if more details about session specific wait events).

Among the top events in **normal systems** are that log file sync, DB file sequential scattered or parallel read, control file parallel write and library cache wait events.

Wait events are the symptoms, not the cause of the issue. Some Wait events appear at the top but they indicate to an issue if their value way exceed their normal figures. Some other wait events shouldn’t be in the top event at all. If they’re it’s a strong indication that there is a problem in the performance of the system.

**Section – 9: Using AWR (Automatic Workload Repository)**

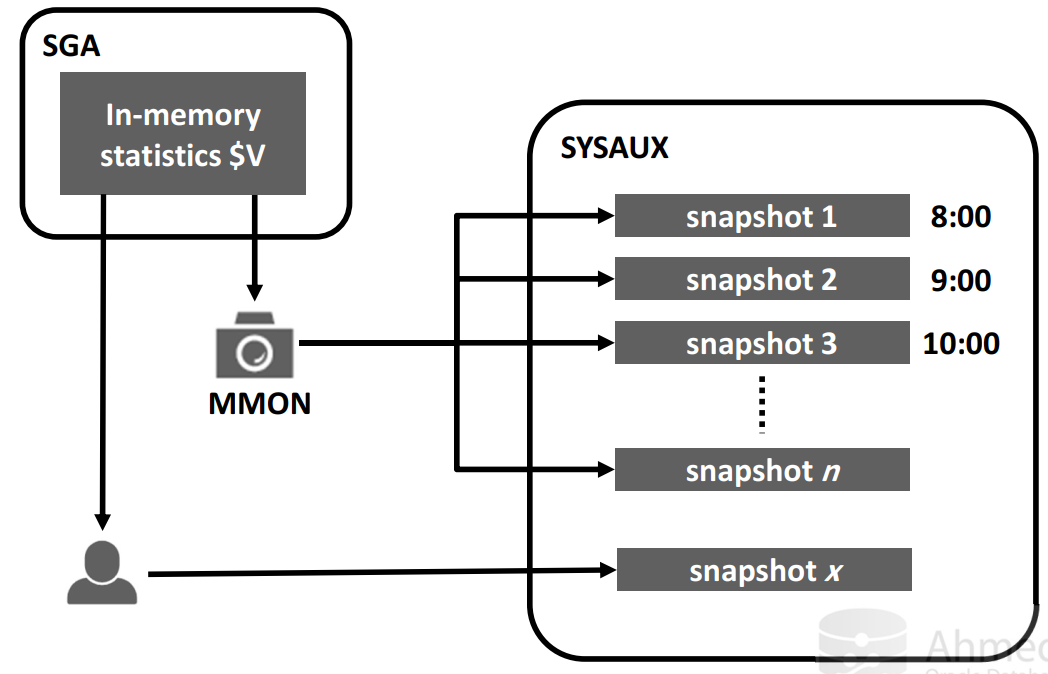
Overview of AWR:

AWR is a mechanism which automatically captues in-memory performance statistics and save them into **AWR snapshots** (in SYSAUX tablespace). These snapshots are used for automatic performance tunig (ADDM) and manual tuning). It has two goals:

1. To assist the database in automatically detecting performance issues.
2. To provide the DBA a sophiscated too to detect the cause of the issues.

AWR collects performance statistics at pre-defined interval and store them in its repository. The database engine keeps updating performance statistics of its current operations in the memory nearly every second. These statistics can be accessed through V$ views. In the database, there is a background process called MMON (Managebility Monitor process). By default MMON takes a snapshot from the performance statistics (V$) views every hour and saves them in the SYSAUX tablespace. These snapshots are not reset at instance startup as they are, unlike the V$ views which reside in memory, saved in a tablespace. Besides, these default snapshots a DBA can manually create a snapshot. The manually created snapshots are assigned an ID, just as system created snapshots.The snapshots are accessed using the DBA\_ views.

**AWR Snapshot creation Diagram**:



**AWR ­reports and Comparison reports**:

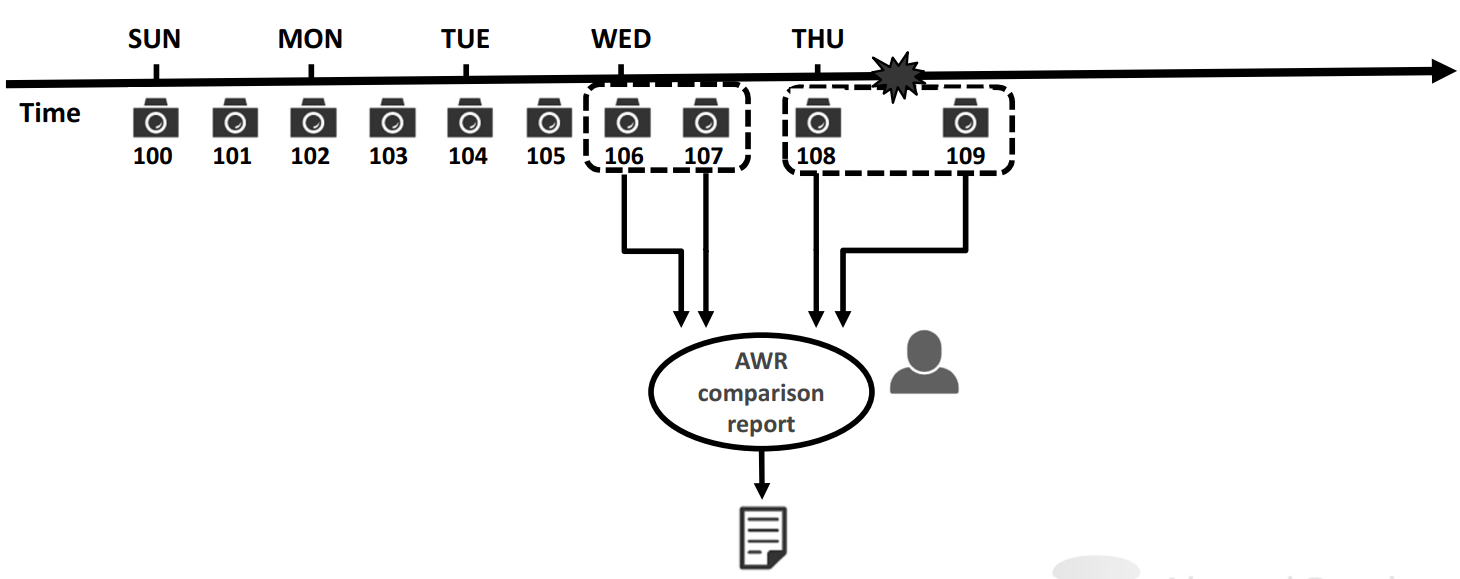
The diagram below depicts a database with a AWR snapshots. The last AWR snapshot was taken on Thursday (SNAP ID: 108). A performance issue has occurred after the most recent snapshot.



We will create a new AWR snapshot (SNAP ID 109) and identify the most recent pair of AWR snapshots. Create an AWR reports based on the comparison of these two snapshots (SNAP ID: 108 and 109). AWR comapres the performance statistics in these two snapshots and create the **AWR report**. Thus it does what a DBA does manually. The DBA still has to manually analyze the AWR reports to find our the bottenecks.



It's possible to compare two pairs of AWR snapshots. In the below diagram, SNAP ID 106 and 107 represents the system AWR snapshots when the system was running fine with similar workload. To get most benefit of these comparison, the pairs should be too close to each other. We can then generate a comaprison report between the two defined pairs of AWR snapshots. This report is called **AWR Comparison report**.



**Automatic Database Diagnostic Monitor (ADDM)**:

ADDM is an automatic mechanism developed by Oracle which automatically comapres AWR snapshots, identify the top SQL statements consuming DB time and propose solution to tune them. ADDM applies same mechanism as we do in manual comaprison.

**Managing AWR snapshots**:

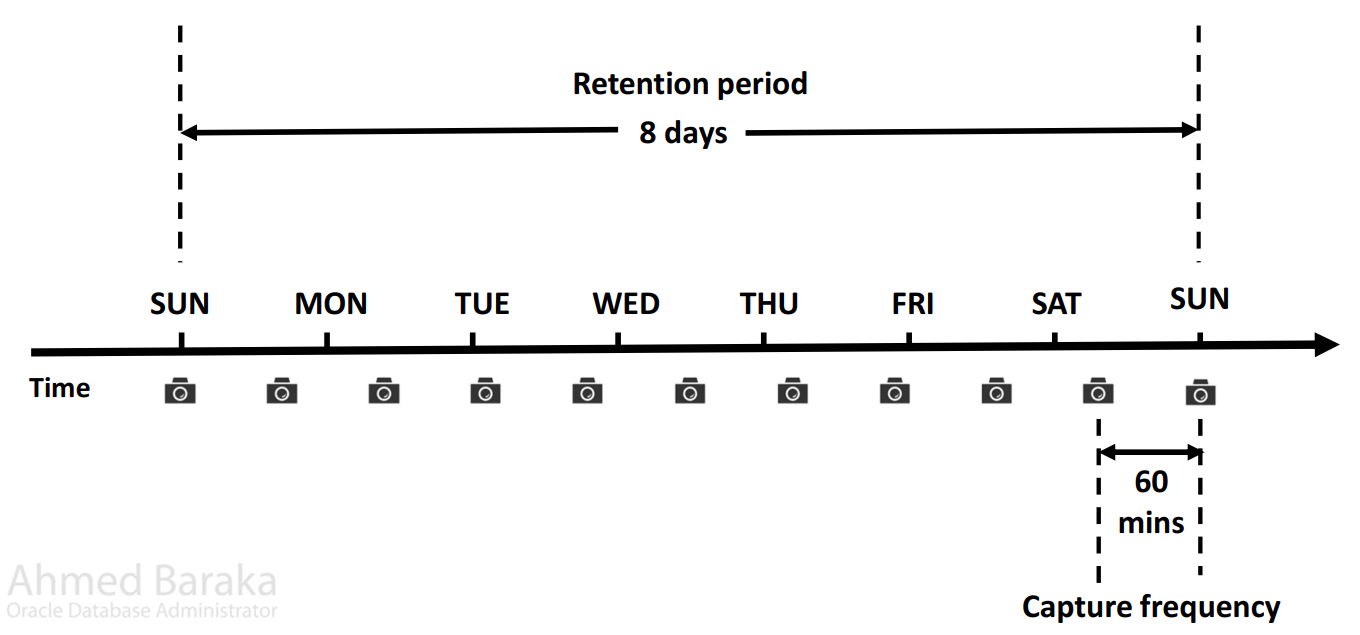
AWR captured statistics include:

1. Base statistics: Counters and value statistics like logfile switches and allocated memory components
2. SQL statistics
3. Metrics: Metrics are different than statistics. They’re the rate at which statistics have changed, e.g., physical reads per minutes or redo size per transaction
4. Active Session History (ASH)
5. ADDM reports

In order to use AWR, the organization must have **Diagnostic Pack** license.

**AWR Default Purging Policy:**

AWR snapshots cannot remain in the database forever or the SYAUX tablespace will run out of space. AWR maintains a purging policy which keeps the AWR snapshots for 8 days. Every night MMON database process purge the AWR snapshots older than 8 days. AWR snapshots are captured every one hour (60 minutes). These settings can be changed by the DBA.



To enable/disable automatic AWR snapshot capture, STATISTICS\_LEVEL must be set to TYPICAL or ALL. If it’s set to BASIC many Oracle performance features including automatic AWR snapshot capture is disabled.

**Managing Snapshots with PL/SQL**:

There are three procedures of DBMS\_WORKLOAD\_REPOSITORY package which are used to manage snapshots: MODIFY\_SNAPSHOT\_SETTINGS, CREATE\_SNAPSHOT and DROP\_SNAPSHOT\_RANGE.

MODIFY\_SNAPSHOT\_SETTINGS: Four parameters: RETENTION, INTERVAL, TOPNSQL and DBID.

Retention (in minutes): How long AWR snapshots would be retained. Default value is 8 days. If set to 0, snapshots uatomatic purging would be disabled and snapshots would be retained forever.

Interval: It controls the interval period between snapshots. Minimum value is 10 minutes and maximum is 100 years. By default it’s 60 minutes. If it’s set to 0, snapshot capture would be stopped. It’s a temporary solution to resolve issues such as SYSAUX tablespace running out of space.

Topsql: DEFAULT, MAXIMUM and integer n. The number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, Shareable Memory, Version Count). The value for this setting will not be affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. he setting will have a minimum value of 30 and a maximum value of 50,000. Specifying NULL will keep the current setting. DEFAULT value of topsql is 30. MAXIMUM will cause the system to capture the complete set of SQL in the cursor cache.

Dbid: generally applicable for Dataguard environment. In non-Dataguard environment it’s applied to local instance (default = NULL).

To retrieve applied AWR snapshot settings query DBA\_HIST\_WR\_CONTROL data dictioanry table. SNAP\_INTERVAL and RETENTION are displayed in interval date format.

**Setting AWR Snapshots Guidelines**:

1. **Assess the workload cycle**: Some applications process OLTP workload during the day time and batch workload during the night. In such cases work cycle is one day. Some applications produce normal workload (OLTP) and in weekends they do batch processing. In such cases work cycle is one week. Some applications produce OLTP workload along the month and do batch processing only at the month end. Work cycle is one month in such cases.
2. For daily workload, keep the retention to the default value or higher. The higher the better.
3. For longer workload cycles (weekly or monthly), retain the snapshots of at least a couple

of workload cycles. The more is better.

1. Study the disk space requirements because more snapshot you save the more space it would consume.
2. Decrease the interval to 30 minutes or 15 minutes, if you can accept the consumed disk space.

BEGIN

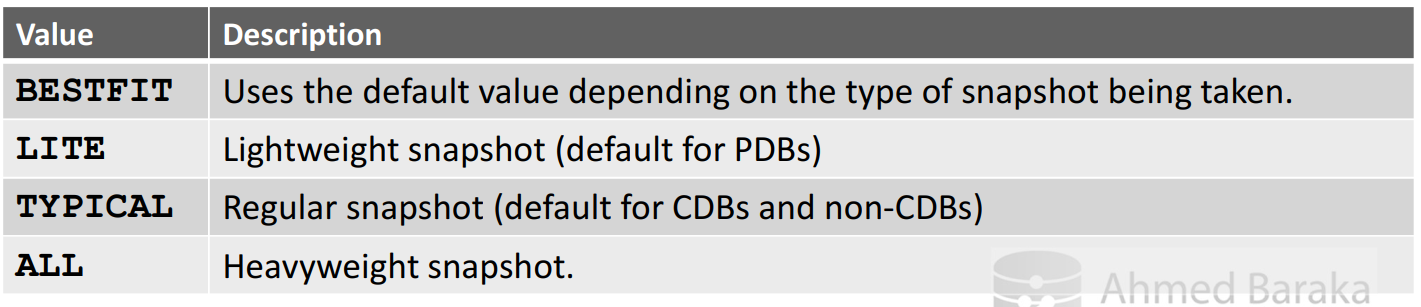
DBMS\_WORKLOAD\_REPOSITORY.MODIFY\_SNAPSHOT\_SETTINGS (RETENTION => 43200, INTERVAL=> 30, TOPNSQL => 100);

END;/

Retention period is 43200 minutes (equivalent to 30 days). Interval is 30 minutes or every 30 minutes AWR snapshot would be captured and number of top SQL to flush for each SQL criteria is 100.

**Managing AWR Snapshots: Creation and Deletion**:

To create AWR snapshot manually, execute DBMS\_WORKLOAD.CREATE\_SNAPSHOT. This procedure has a parameter called snapshot type which can be set using FLASH\_LEVEL. Following are the options which we can set for FLASH\_LEVEL:



To delete a range of AWR snapshots: DROP\_SNAPSHOT\_RANGE.

**begin**

**DBMS\_WORKLOAD\_REPOSITORY.DROP\_SNAPSHOT\_RANGE**

**(LOW SNAP ID => 100, HIGH\_SNAP\_ID => 120);**

**end;**

Snapshots related to baseline are not deleted. Please bear in mind that this procedure delete entries from the internal tables (data dictionary) but does not reclaim the space. Also, the deletion generates redo and undo so if you intend to delete plenty of snapshots, do it in batches.

To display information about available snapshots:

SELECT

SNAP\_ID, BEGIN\_INTERVAL\_TIME, END\_INTERVAL\_TIME, SNAP\_LEVEL

FROM

DBA\_HIST\_SNAPSHOT

ORDER BY SNAP\_ID;

**Manage Space cosumed by AWR snapshots**:

Storage consumed by AWR snapshots affected by:

* AWR snapshots retention period.
* AWR snapshots interval
* Number of active sessions

For example, an Oracle database captures snapshots once every hour, returns them in the database for 8 days, and if the concurrent active sessions is 10 then AWR can easily reach up to 200 – 30000 MB.

To obtain the size occupied by AWR snapshots:

**SELECT OCCUPANT\_NAME, SPACE\_USAGE\_KBYTES/1024 MB, MOVE\_PROCEDURE**

**FROM V$SYSAUX\_OCCUPANTS**

**WHERE OCCUPANT\_NAME = 'SM/AWR';**

A script to provide more detailed report:

**@?/rdbms/admin/awrinfo.sql**

**Using AWR Reports**:

So far we have learned about using the performance Tuning tools, Time model views, Operation Statistics views, Wait Event views.

**Time Model Views**: We use them to measure the entire database instance work load and performance. We do not normally start with looking at time model views.

**Operation Statistics views**: We have three views: Instance level, Client sessions and for the current sessions.

Wait events are useful to to troubleshoot the current sessions; it tells us what the session is waiting for. But it doesn’t tell us anything more. System events provides statistics since last instance startup and hence not useful in performance troubleshooting.

To start with high level performance troubleshooting we start with ASH (Active Session History) or AWR report. If performance issue is included in the AWR report, we use them. If it happens a short time ago we use the ASH.

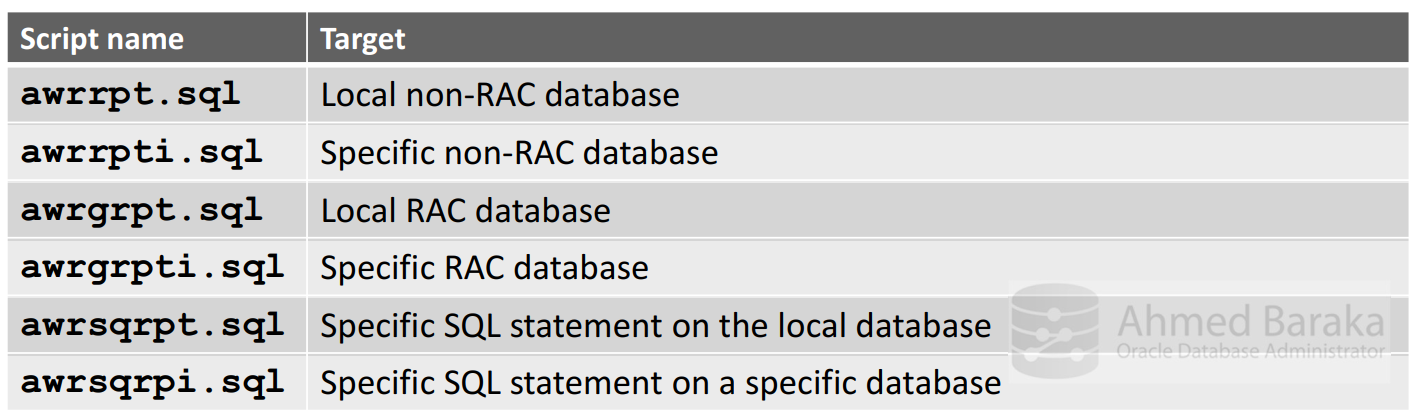
**Generating AWR Reports**:

An AWR report contains the performance statistics over a period of time defined by two snapshots. The report is actually a delta statistics report between the selected snapshots. A delta statistics report means the statistics in the second snapshot is substracted from the first snapshot.

AWR reports can be generated by running one of the 6 scripts supplied by Oracle. Which script you should run depends upon if you want to run it for Local non-RAC database, specific non-RAC database, Local RAC database, specific RAC database, specific SQL statement on the local database or specific SQL statement on a specific database. SELECT CATALOG role privilege is needed for the script executor.

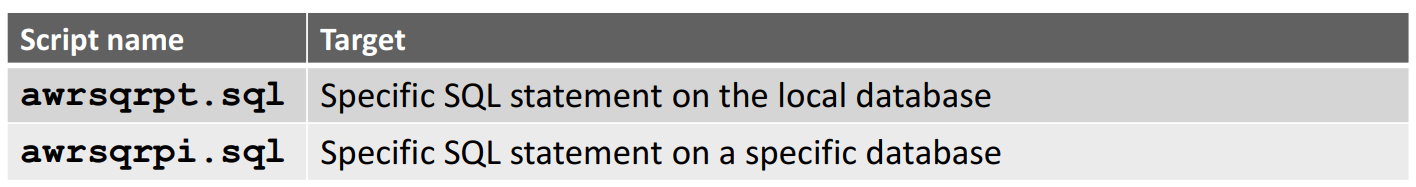
Depending on the script you run it prompts you to enter the starting and ending snap ID or database or SQL statement ID.

Script files saved in **ORACLE\_HOME/rdbms/admin/**



**Generating AWR SQL report**:

AWR SQL report is a report for a specific SQL statement in a specific database or local database. You are asked to enter start, end snap ID and SQL statement.



Remember that only Top n SQLs are captured in the AWR repository. Which SQL statements figure in TOP n SQL depends on how frequently the SQL is exceuted and CPU time, etc.

**Using AWR for Diagnostic Performance Issues**:

Reactive procedure:

1. Read the AWR reports to find the worst bottleneck in the database.
2. Once the bottleneck is identified, relieve it.
3. Repeat until the performance target is hit.

Proactive procedure:

When the system is running in healthy condition, AWR report can help you familirize with the performance profiles in a healthy state. We need to perform this process for various workloads such as OLTP, Batch Processing, Datewarehousing. Every system has performance finger print. By reading the AWR report when system is in good condition, you will know the normal level of Wait events, I/O reads, number and types of sorts, PGA activity levels and so on.

**Strategy of Reading AWR Reports**:

AWR report is organized into so many sections. Each section displayes statistics for for a particular area. The summary section provides high level statistics of many areas. The remaining sections provides further details. First read the report under the Summary title, find the area which has the problem and depending upon the area read the detail section. For example, while reading the top wait events (in summary section), you observe an enqueue lock at the top of the list. Therefore you went to the enqueue activity section in the detail to verify the wait event which has longest wait time. Then you can go to Wait Event Histogram to know if that wait time was spent on one session or on a few session. Then you want to know which SQL statement is responsible for that enqueue so you go to the SQL by elapsed time section to see if the culprit SQL is in the list.

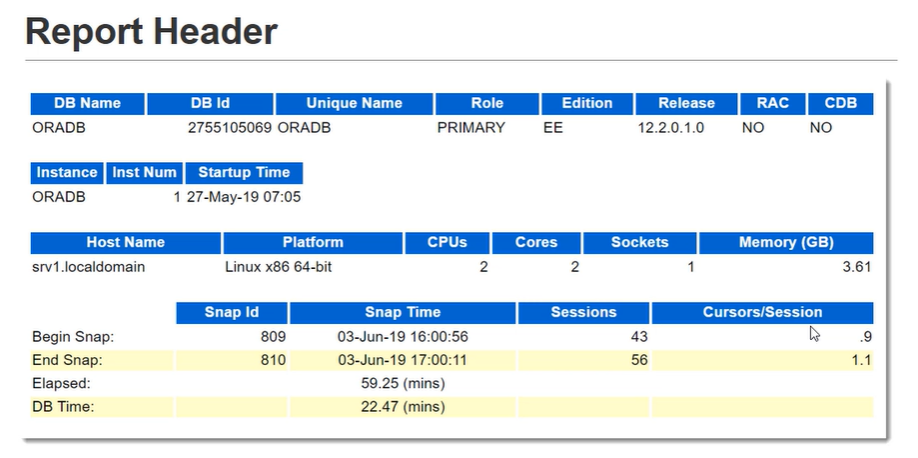
**AWR Report Summary section**:

Here are the most important sections under the Summary section of the AWR Report:

1. **ADDM findings**: If you’ve significant performance bottleneck, they will be caught in ADDM. ADDM analyze the issue based on the amount of time spent on the operation or wait events compared to the AWR period DB time.
2. **Host CPU and Instance CPU**: This section tells us if we have a CPU bound sitiation. CPU drain could be caused by either external processes or internal database sessions or both. If they are caused by external processes there is no point in investigating further in database performance statistics.
3. **Load profile**: From these sections tells us which opeation type the database was loaded with the most.
4. **Instance Efficiency percentage**: This tells us for each instance which operation types the database is loaded with.
5. **Top 10 Foreground Events by Total Wait Time**.
6. **Top Wait Events and Background Wait Events**.
7. **Memory Statistics, Cach size, and Shared Pool Statistics**.

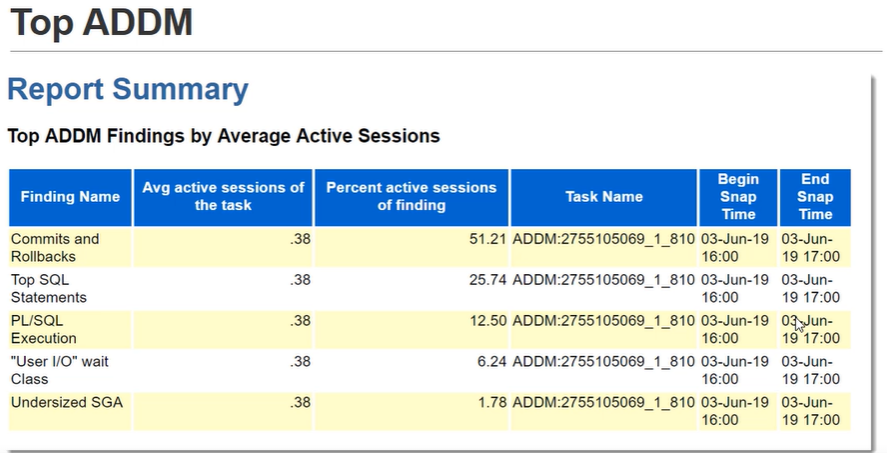
**AWR Report Header Section**:

In the below sample, the DB has 2 logical CPUs and 2 Cores (hardware threads or physical CPUs). It has top DB worload indicator, that is the DB time. DB time is ~22 minutes of around 1 hour of elapsed time. The number of sessions include background and foreground sessions. When system is under stress the **DB time can be greater than the CPU time**, for example, if we have four CPUs within 1 hour the CPUs could give 4 hours of CPU time (DB Time = DB CPU + Total DB Waits).



**AWR Report Top ADDM Section**:

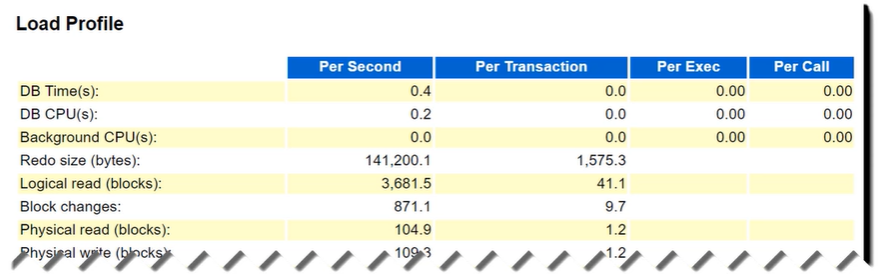
ADDM findings are sorted by Percent Active sessions of findings. Active sessions means number of Active session per second. A session is considered active, if it’s being served by the DB (consuming CPU or waiting). **Average active sessions** of an ADDM task is computed by divinding the DB Time by Elapsed time, in other words how much time the database was busy. We will see this term in many places.



**Load Profile**:

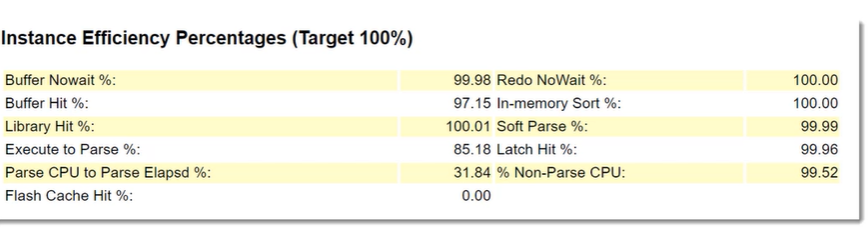
This section tells us what are the operation types the DB is loaded with. The section has metrics i.e., Per Second, Per Transaction, Per Exec, and Per Call.

For **DSS environments**, large number of physical reads and logical reads with few block changes, SQL Work area, lower user calls and parses and for OLTP environments, more logical reads, user calls, parses, executes, rollbacks, and transactions but fewer physical reads.



**Instance Efficiency and Percentage**:

Generally, the statistics in this section should be as close to 100%. We will describe what is implication and recommendation if these statistics are lower than their normal limit.



We will now briefly describe (elaborate description later) the important statistics reported in Instance Efficiency Percentages:

1. **Buffer Nowait%**: When the database processes want to access data blocks in DB buffer cache in SGA, what was the percentage that the processes did not need to wait for the database blocks to be accessible. If concurrent processes try to access data blocks some or the other process has to wait. It’s always desirable that the processes gets to access data blocks as soon as they place a request, which is Buffer Nowait% is 100%. If this statistic is less than 95% we should investigate the issue.
2. **Buffer Hit%**: It represents percentage of the finding of the required datablocks in the buffer cache. If the required data blocks is not present in the buffer cache, **SMON (not the server process) has to fetch them from the disk storage**. Since accessing data blocks from memory is always faster than disk storage, it’s desirable that Buffer Hit% should always be 100%. If it’s less than 95%, investigate the data buffer usage.
3. **Library Hit%**: It represents the percentage of the finding of the SQL or PL/SQL in the Library cache in Shared pool. If this statistic is less than 95%, the Shared pool might be too small or application may have cusrsor sharing issue. Cursor sharing issue is caused by application using literals instead of bind variables.
4. **Execute to Parse%**: It represents the number of SQL executions to (divided by) the number of parses. The Load profile displays the number of Execute and Number of parses in the Load Profile. If this percentage is low it means lot of parses have been performed in the system. This is expected in a reporting system where adhoc queries are executed by the users. In such systems users are firing different statements. However, in OLTP system users fires similar statements so we expect this percentage to be high.
5. **Parse CPU to Parse Elapsed%**: It represents how long it took for the CPU to parse the statements compared to the total parsing time. In theory, we want the total parsing time equals to the CPU parsing time which means ideally we want this statistic to be 100. If this statistic is low it means the process had to wait before it completes its parsing which is, usually, a sign of the contention on the Shared Pool. The referenced objects may have been compiled or locked by somehow. We need to investigate further to trace the issue.
6. **Redo Nowait%**: It represents the process does not have to wait for the Redo log buffer when running DML statements. It’s available immeditaely. In a healthy OLTP system, it’s very close to the 100%. In a batch processing system, this statistic can go low because of large amount of changes that take place in a very short period. This statistic could go low if the Redo log buffer is too small or as an indirect result of inefficient archives. The storage system that hosts the Redo log archive file may be congested or nearly full.
7. **In-memory Sort%**: It represents the percentage on which the sorting is performed in the memory. If this percentage is less than 100%, it means some sorting is performed on the disk (sorting in a disk is much slower than sorting in memory).
8. **Soft Parse%**: It represents how often the SQL statements submitted by users are found found in cusrsor cache in Shared SQL area. If it’s less than 95%, you most likely have cusrsor sharing issue.
9. **Latch Hit%**: It represents how often we are not waiting for latches. If this percentage is low, look for CPU bound processes and issues with latching.

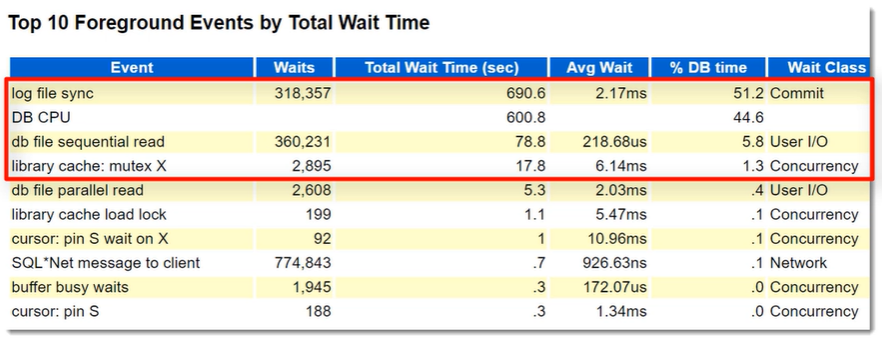
**Top 10 Foreground Events by Total Wait Time**:

The next section in AWR report is Top 10 Foreground Events by Total Wait Time which lists Top wait events in the reporting period. It gets information from history view V$SYSTEM\_EVENT however, AWR adds DB CPU operational statistics to that list. It’s added to the list so that you can compare any wait event value to the DB CPU value. Also, this list only includes Foreground processes.

**One important thing to remember about DB CPU**. Its presence in this list is in fact counterintuitive because it’s not a Wait Event. The process is on CPU.

First, wait time means something special to Oracle DBAs. To an Oracle DBA anything associate with a "wait" should have a wait event name, a wait occurance, the time should be instrumented (i.e., measured) and should be recorded in the many wait interface related views, such as V$SYSTEM\_EVENT or V$SESSION. Secondly, from an Oracle perspective the process is truly "on cpu" because the process is not "waiting." Remember, an Oracle session is either in one of two states; CPU or WAIT. There is no third choice. So the words "CPU Wait" are really confusing. Thirdly, rom an OS perspective or simply a non-Oracle perspective, the Oracle process is sitting in the CPU run queue.

**DB CPU in Top 10 Foreground Events** does not represent how much time DB foreground processes are waiting for the CPU resource. It rather shows how much CPU time DB foreground processes are consuming. This does not include the CPU-run-queue time.

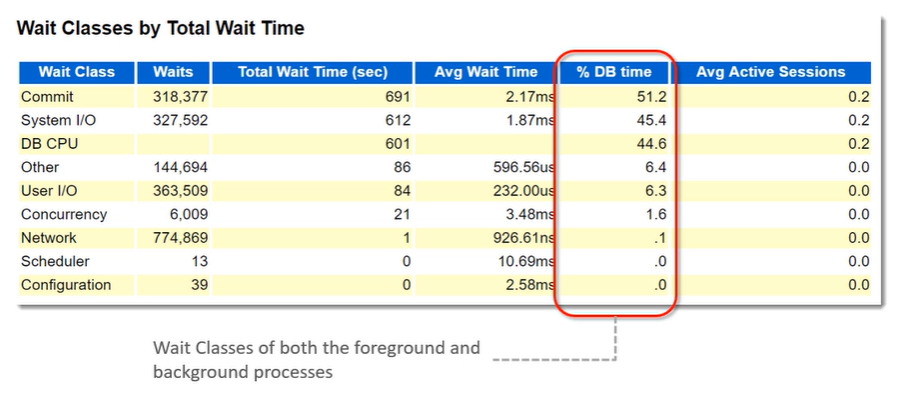


Usually we care only the Top events (marked in Red). They represent high pecentage of total wait time. Usually, there is a link between the peformance issue and at least one top wait event in the list. For example, if the top events include **DB File read** event and they represent 50% or more of the DB time then most likely you have performance issue. The issue could high number of full table scan operations. As a remedy, we should look into the Top SQL statements and tune them. The issue could also be with the storage system – it could be congested, could have issue in its firmware or connections.

Another Foreground wait event is “**log file sync**”. This wait event occurs when a session issues a COMMIT. The Redo log writer then writes the changes from Redo log buffer into Redo log file and send ackonowledgement to the user. In this example, it accounts for 51.2% of DB time; it happened nearly 320000 times and each time the user has to wait nearly 2 milliseconds. While this is not a long time for this event so the real issue here is we have plenty of transactions committing after each data change which is not the correct way of desisgning a batch applications. The applications should issue commit only after a batch of data changes to reduce the number of COMMITs and “**log file sync**” wait event.

**Wait Class by Total Wait Time**:

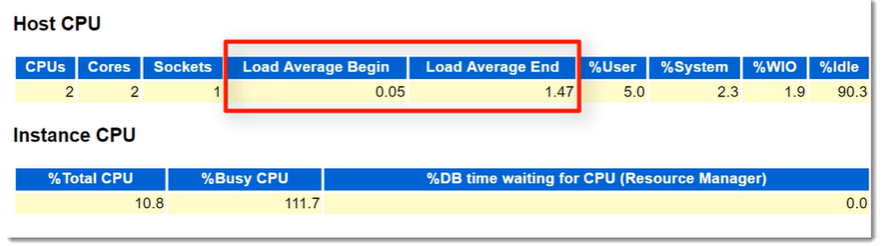
It lists the Top wait classes in the AWR report reporing period time. In the belwo reports %DB Time for Commit is 51.2% (same as “log file sync” in the previous section). Note that Wait class list includes both background and foreground processes while DB time includes only foreground processes.



**Host CPU and Instance CPU**:

The reason for looking at this section is to know if we have a **CPU bound** situation or not. Host CPU statistics directly come from the Operating system and reflect the CPU usage of the whole server machine. Instance CPU reflects the CPU usage (allocated to the instance) as used by the database.

Keep in mind that OS statistics and CPU statistics reported in AWR report are not very accurate. OS utilities provides more accurate and precise statistics. However, Host CPU statistics reported inAWR are fair enough to construct a fair image about the load on the CPU.



**Load Average Begin** and **Low Average End** represent average load on the CPU. In normal state they should not exceed the **twice the number of cores** or they may indicate a **CPU bound state**. **%User** represents the percentage of what the applications in the machine including the Oracle database instances are consuming from the total CPU computing time. In other words, it represents how much of the total CPU is used by the applications. **%System** represent the percentage of using the CPU by the OS. **%WIO** is the time percent of CPU spent on waiting for Input-Ouput (I/O). **%Idle** represents the percentage of CPUs were idle, doing nothing (because no process is using them).

The total of **%User, %System, %WIO, %Idle** would be close to 100%. The total of **%User** and **%System** is called **Busy Time**.

**Instance CPU**: **%Total CPU** represents the ratio of the total CPU time consumed by the database including the foreground and background processes to the **Total CPU time**, which is the Busy Time and Idle Time. The exact formula is **%Total CPU** = Total DB CPU Time (can be obtained from Time Model Statistics table in the AWR report) / (“BUSY\_TIME + “IDLE\_TIME”). BUSY\_TIME and IDLE\_TIME can be obtained from OS Statistics table. The **%Busy CPU** represents, in theory, represents ratio between the Total CPU time by the Database to the Total CPU Busy time. In other words, it represents database portion of the machine CPU Busy time. **The formula to represent this**: Total Database CPU time / CPU Busy Time. While it should never exceed 100%, due to the inaccurate OS statistics in AWR report this is often reported more than 100%. However, it should not be a problem for troubleshooting. If this figue is close 100%, it suggests the that machine CPU power is mostly used to serve the database.

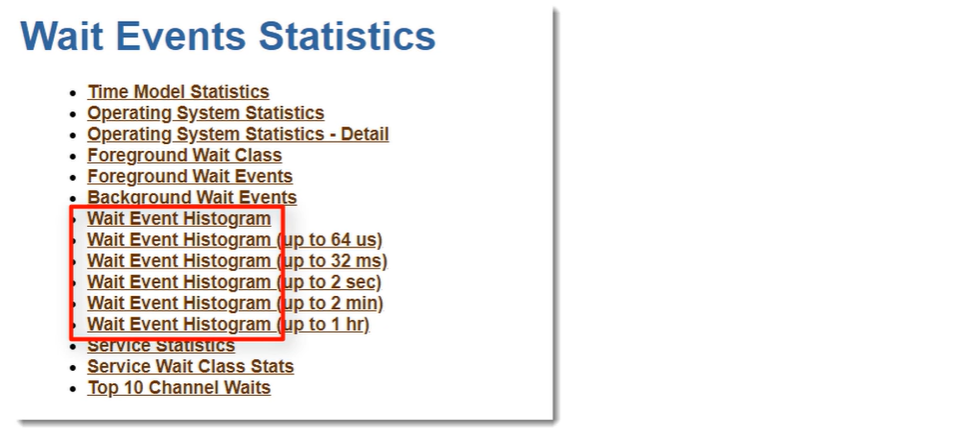
**Main Report Section**:

We have discussed most of the sections under the Summary title of the AWR report. The sections that come after the Summary sections provide further details on the performance statistics. Based on our findings in the Summary sections we check related details sections.

The important sections in AWR report come under Main Report section title and each section is linked in the Main Report section:

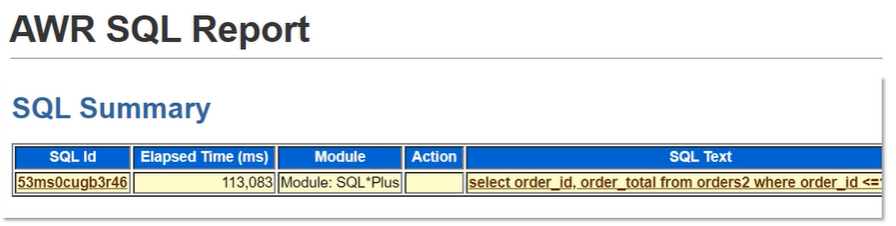
**SQL Statistics**: This section lists different SQL statement executed during the AWR period ordered by different dimesnions. This is a great of way of presenting the SQL statements. We refer to those sections to obtain the code that is linked to the area that we are investigating.

**Wait Evens Statistics**: It has links to further Statistics tables. It has links to Statistics just other othan Wait events, e.g., time model statistics, OS statistics. In the beginning of the AWR report, we have seen the Foreground wait events which provides total statistics details on the Wait events. But we cannot determine if the wait events were for all the sessions or a few sessions. That’s what the Wait Event Histograms are helpful for. Wait Event Histogram table describe how Wait events are distributed among time buckets in percentages. This way we can know if the specific wait event type was spent on a few sessions or high number of multiple sessions. AWR report does not provide session level statistics but ASH report does which is already included in the AWR report.

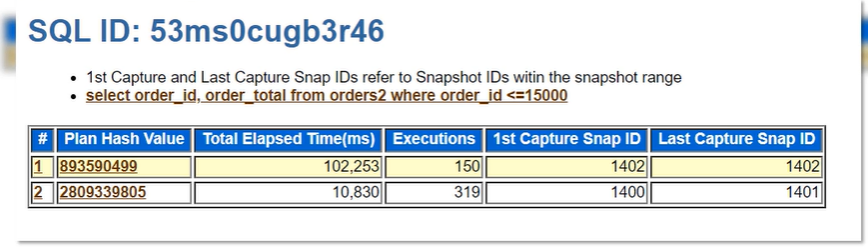


**AWR SQL Report**: In this report header, we will see SQL summary section.

**SQL Summary**: In this section we see the total elapsed time of the SQL statement and full text of the statement.



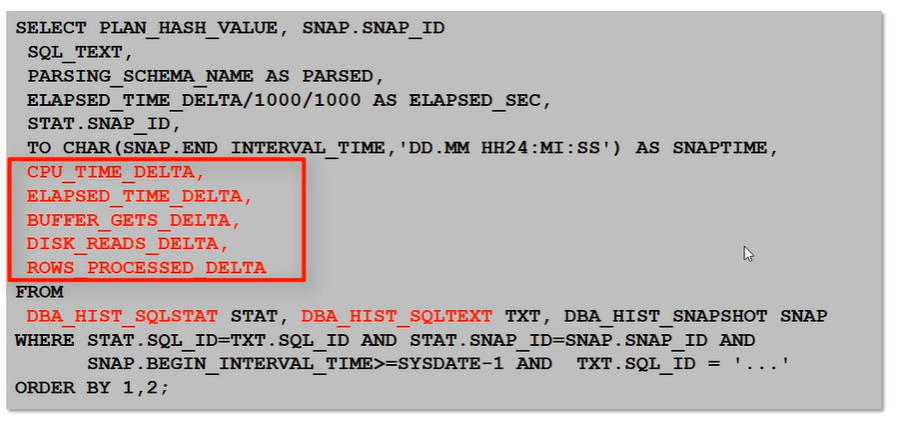
**SQL ID**: After that comes the most important part of the report, that’s the execution plan of the statement. The section header name is SQL ID value. Here we see all the execution plans used to execute the SQL statements as per AWR snapshot. From this table, we can know the difference in the performance between different plans. In the below table the second execution plan is much more efficient than the first one.



**Plan Statistics**: For each execution plan we see its statistics in the below table format.



While AWR SQL report is sufficient to obtain performance history SQL statements, some DBAs prefer to get the information from data dictionary views. SQL statement history statistics are saved in DBA\_HIST\_SQLSTAT. By linking it to the DBA\_HIST\_SNAPSHOT we can get details of performance history of SQL statements.



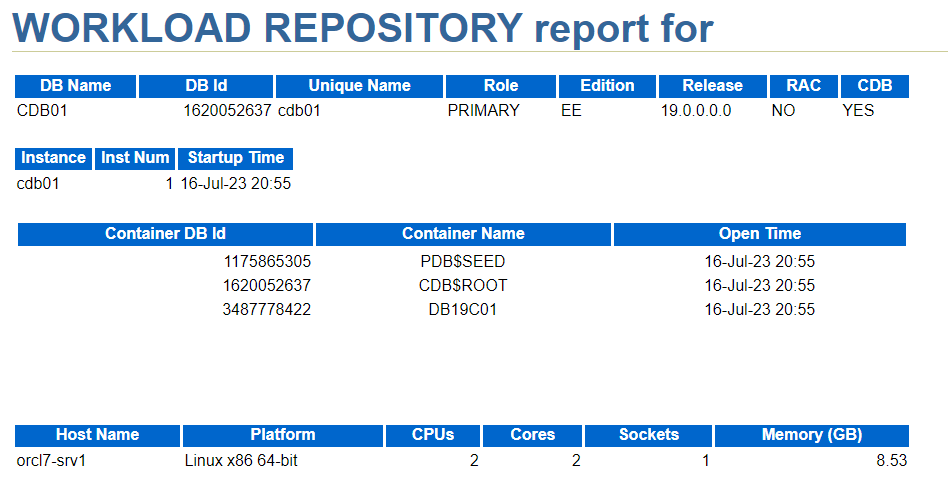
The delta columns represent the value of the statistics from the BEGIN\_INTERVAL\_TIME to the END\_INTERVAL\_TIME in the DBA\_HIST\_SNAPSHOT view. Remember that we refer to these views only if we analysing specific SQL query for long period in the past. If we analyze queries currently executed by current session, we should refer to the V$SQL, V$SQLAREA, and V$SESSION. If we want to analyze the SQL statements in past hours, we should use the ASH views.

**Tips on Using AWR Report**:

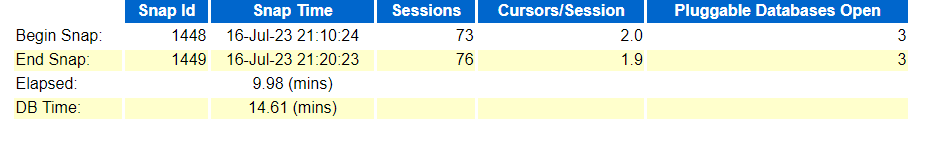
1. Study the summary section and then set your path in the detail section.
2. When the system runs fine, be familiar with the AWR report. This lets you identify irregular statistics in the AWR reports.
3. Not all performance issue could be caught by the AWR reprt. AWR report relies on the performance statistics within the AWR elapsed time. If a performance issue does not impact the AWR statistics, it will not be reported. Generally, AWR report is useful for Database wide performance troubleshooting.

**Understanding Sample AWR report - OLTP:**

We have created a sample AWR report called normal\_oltp.html. We will now explain the report. In the Report header we see the Database name, number of CPUs (2) and cores 92), etc.



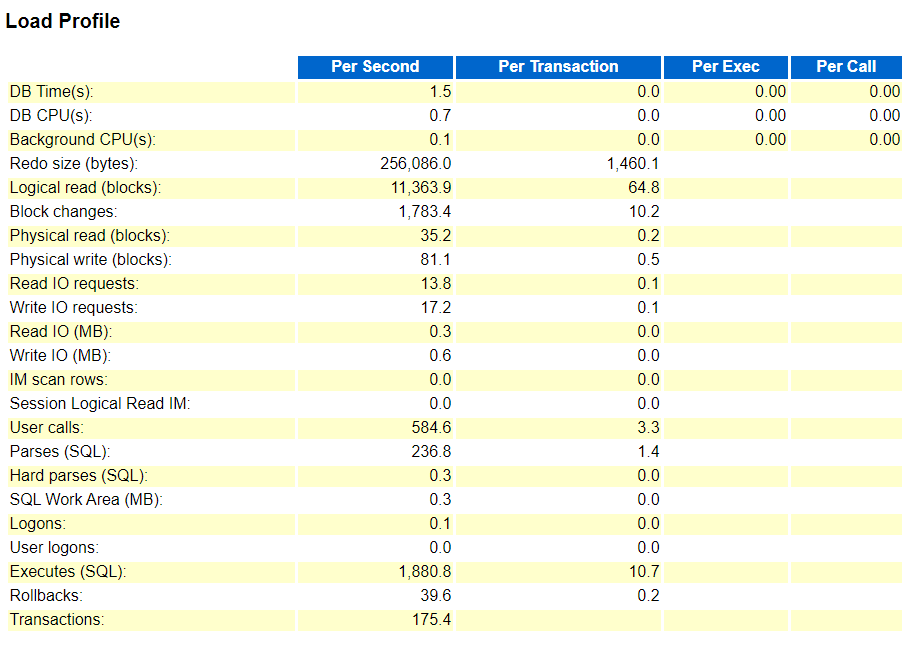
Next we see that the Elapsed time is ~10 minutes (difference between two snapshots) and DB Time is 14.61 minutes. The DB Time is greater than Elapsed Time because of more than one CPUs and parallel processing. Remember **DB Time = DB CPU Time + Total Wait Time** (Foreground processes).



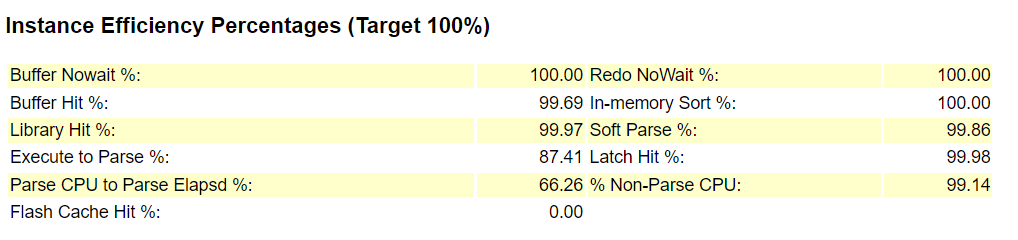
**Load Profile Section**:

Load Profile Section describes to what level database was loaded. It displays metrics rather than statistics such as Per Second, Per Transaction, Per Exec, Per Call. DB Time (Per Second) is DB Time / Total Elapsed Time. We have two CPU cores which means we have 2 seconds of CPU time per second (1 second per CPU). DB CPU time is reported as 0.7 per second meaning DB system is using 0.7 second of CPU of the potential 2 seconds/second CPU Time that it can use.

It means Wait Time / Second = DB Time/Second – DB CPU / Second = 0.8 per second. This means that the total wait time for Foreground processes is around 8 minutes. **Redo Size** is quite high, 256,086.0/Second which is expected in a OLTP workload.



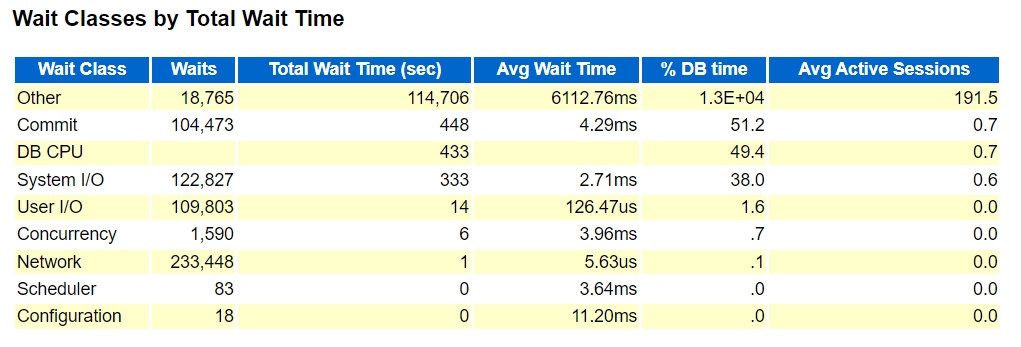
**Parse CPU to Parse Elapsed%** is 66.26%. The parse time CPU statistic shows the total CPU time used for parsing. This ratio indicates how much of the time spent parsing was due to the parse operation itself (by CPU), rather than waiting for resources, such as latches. A ratio of one or 100% is good, indicating that the elapsed time was not spent waiting for highly contended resources. If it’s lower than 100% then check **Number of Hard Parses (SQL)** in **Load Profile** section. If it is low (in our case 0.3) it’s good news then check Parse Time Elapsed In Time Model Statistics, if it’s low then the low ratio is not an issue.



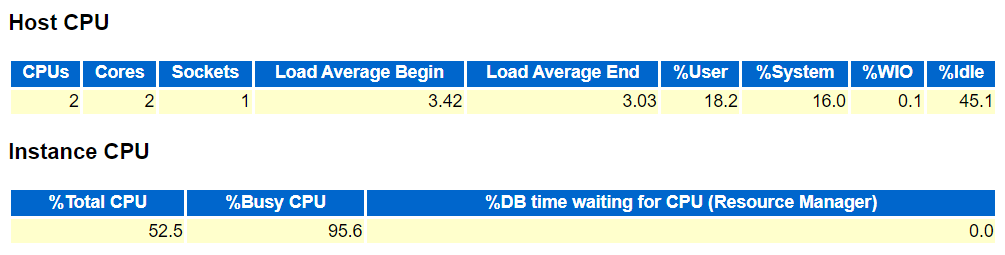
In the Top 10 Foreground events by Wait Time, log file sync is the topmost. It’s due to the multiple COMMIT statements. Total Wait Time (sec) is 448.5 seconds which around 8 mins. Remember we already calculated this while discussing **Load Profile** section.



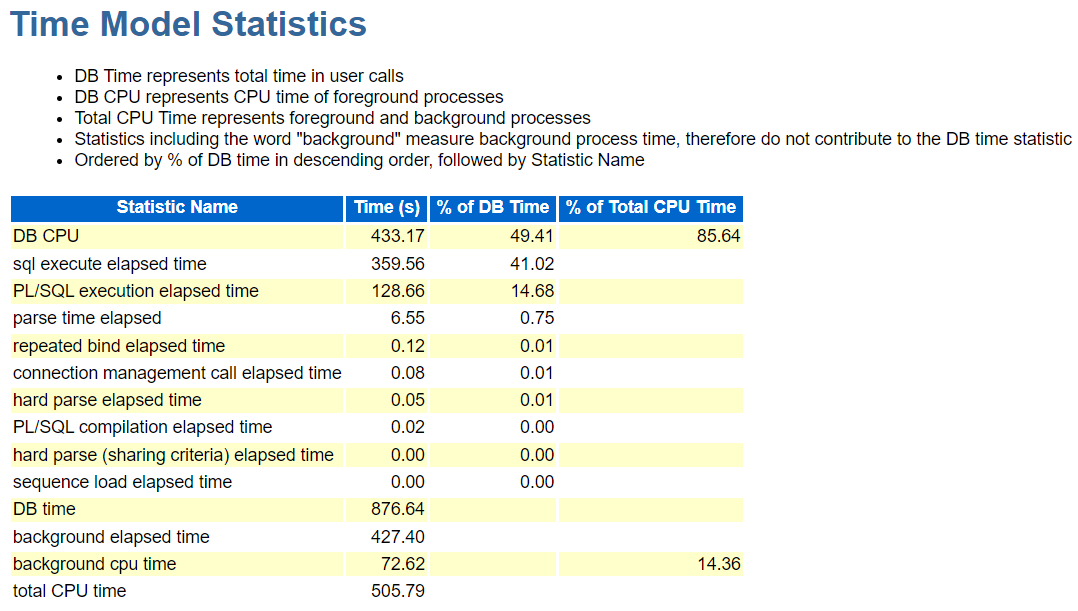
Among Wait Classes by Total Wait Time, Commit Class is a dominant one because log file sync Foreground event is of Commit class.



Load Average Begin and End are not more than twice the number of cores and User% is merely 18% so we are not CPU bound. %Total CPU time is 52.5% which means 52.5% of machine CPU time is spent for database. %Busy CPU is more than 95% so CPU is not sufficiently busy.



**Time Model Statistics**:

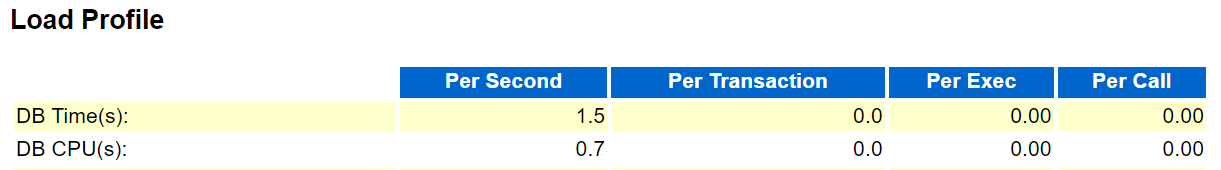


**Understanding Sample AWR report – DWH**:

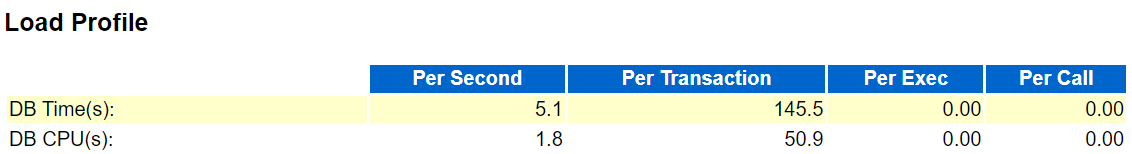
Let us open the OLTP and DWH AWR report side by side.

**Load Profile - DB Time and DB CPU Time**:

**OLTP workload**: DB time (DB CPU + Total Wait) and DB CPU Time are 1.5 and .07 per second respectively.



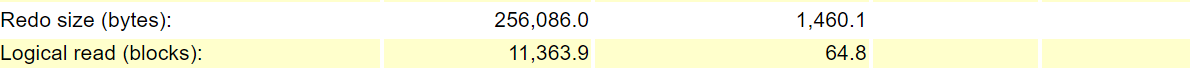
**Warehouse workload**: DB time and DB CPU Time are 5.1 and 1.8 per second respectively.



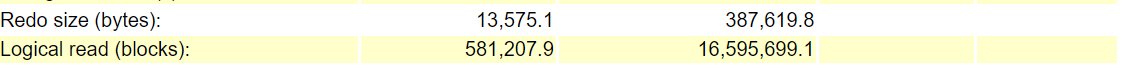
**Conclusion**: DB Time and DB CPU Time were much higher under Datawarehouse workload than those of OLTP workload.

**Load Profile - Redo generation and Logical Reads**:

**OLTP workload**: Redo generation rate and Logical reads are 256,086.0 and 11,363.9 respectively.



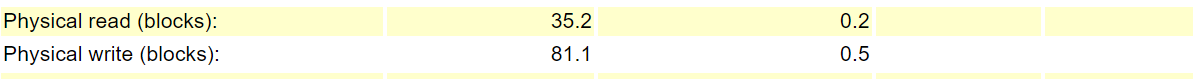
**Warehouse workload**: Redo generation rate and Logical reads are 13,575.1 and 581,207.9 per second respectively.



**Conclusion**: The Redo generation rate and Logical reads for Warehouse workload are far higher than those of OLTP workload and that’s quite expected.

**Load Profile - Physical Read and Write (Blocks):**

**OLTP workload**: Physical read (blocks) and Physical write (blocks) are 35.2 and 81.1 Per second respectively.



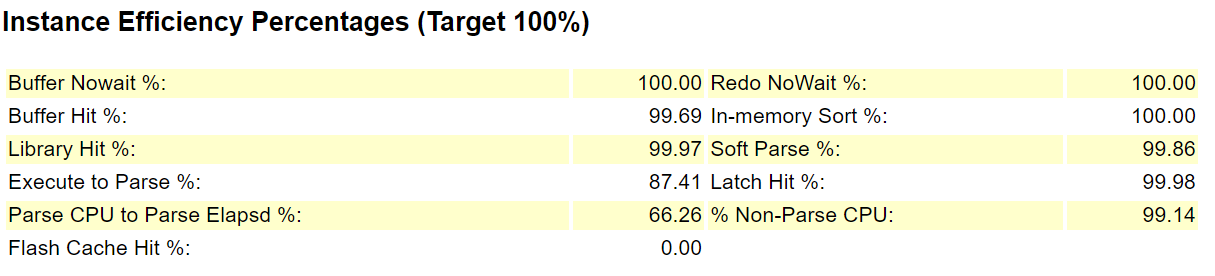
**Warehouse workload**: Physical read (blocks) and Physical write (blocks) are 0.2 and 1.8 Per second respectively.



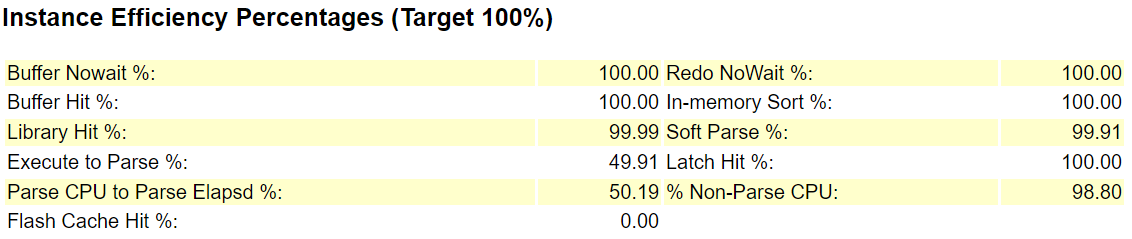
**Conclusion**: The Physical reads and writes are far higher in OLTP workload than those of Warehouse workload since in Data warehouse deal primarily deal with reports (read only SQL) whereas OLTP deals with heavy volume of write operations.

**Instance Efficiency Percentages**:

**OLTP Workload**: Execute to parse % shows how often parsed SQL statements are reused without re-parsing. A value of 100% is most desirable. If this number is low, it suggests parsing is consuming CPU and shared pool latching. In OLTP workload this should be high and close to 100%.

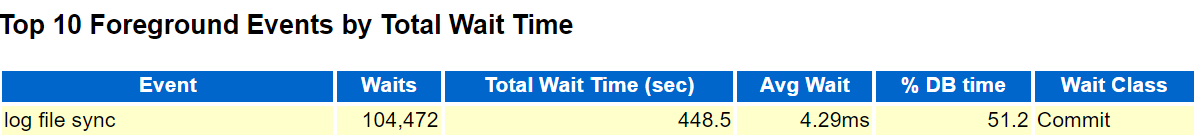


**Warehouse workload**: This ratio is low in Warehouse workload as the number of SQL statements executed is fewer.

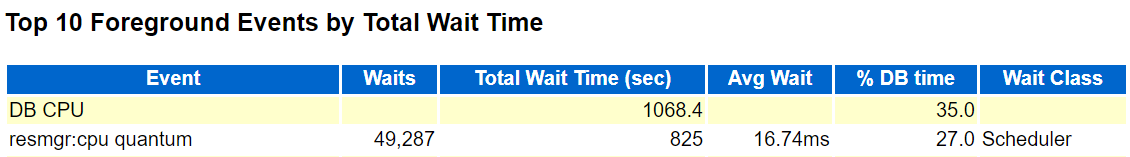


**Top 10 Foreground Events by Total Wait Time:**

**OLTP workload:** In OLTP worload log file sync has the highest wait time (total wait time 448.5 second) and Average wait time 4.29 ms.

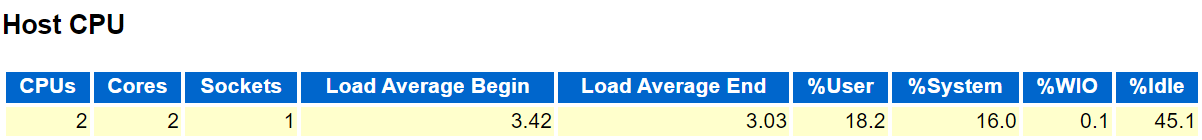


**Warehouse workload**: resource manager CPU quantum wait time is quite high. This wait even is triggered when the resource manager has to distribute resource across sessions due to high demand of resource for the sessions. It’s possible that the workload ran at the same time when the maintenance window was open and maintenance jobs competed with the maintenance jobs for resource. The possible solution is to change the maintenance window time. As we can see that the Wait Class of resource manager CPU quantum is **Scheduler**. This is wait class of the maintenance jobs.



**Host CPU**:

**OLTP workload**: In OLTP workload %User is 18.2% and Load Average Begin and End is less than the twice of the cores (2).



**Warehouse workload**: In OLTP workload %User is 24.2% and Load Average Begin and End is far higher than the twice of the cores (2).



This means under Warehouse workload, the Host CPU is under stress.

**For Instance CPU of Warehouse workload**, we see %Total CPU is far higher than 100% which means the Instance CPU is under stress in Warehouse workload.

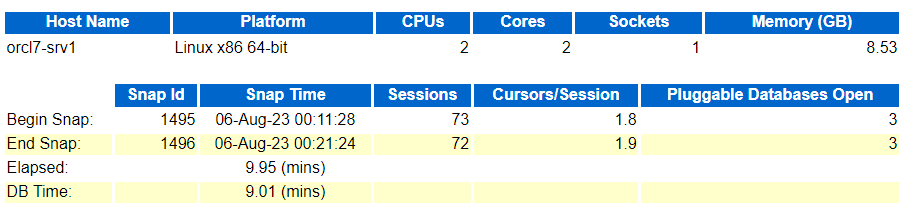


**Conclusion**: The analysis suggests that the system is under stress by the Database processes.

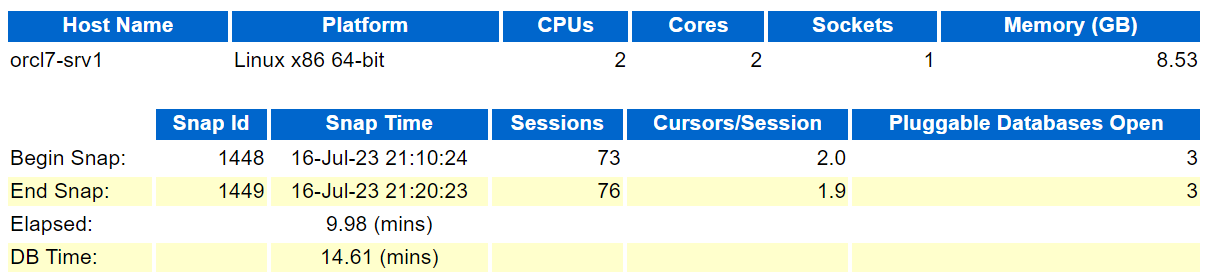
**Using AWR Reports to Troubleshoot a Performance Issue**

**Scenario-1**:

**Issue Report**:

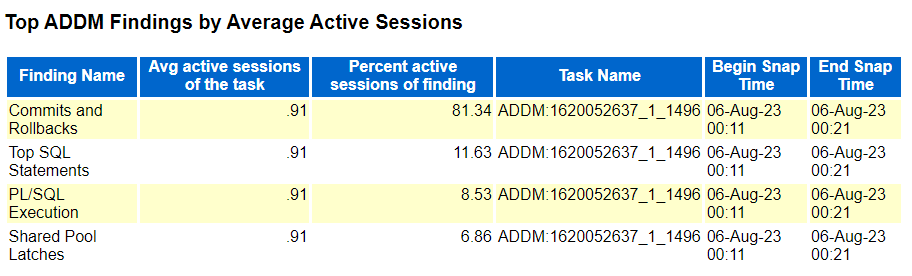


**OLTP normal**:

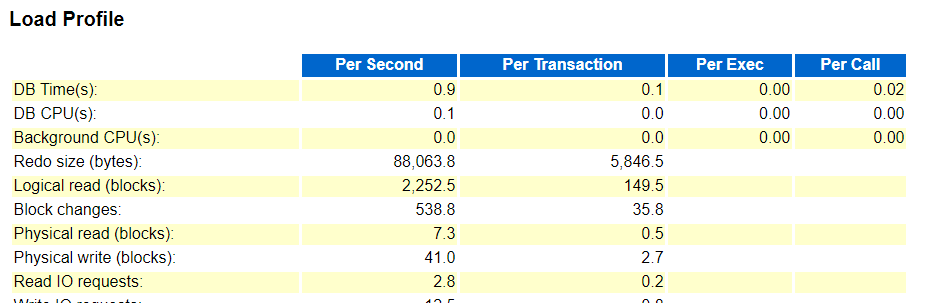


**Conclusion**: Issue Report’s DB time is less than that of OLTP normal. This means the CPU stress did not impact the performace of the system. User did not even notice and might not even have complained.

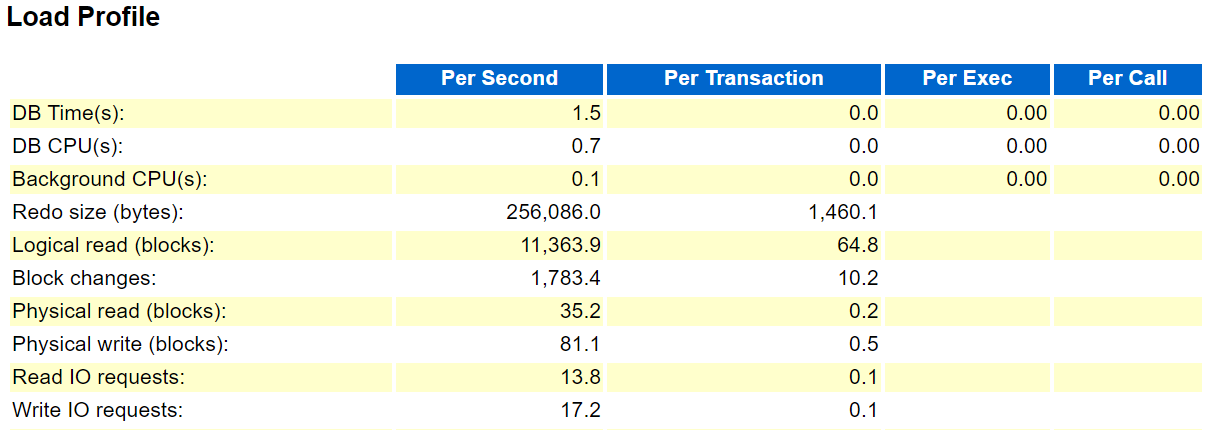
**Issue Report**: In the ADDM there is no finding of the CPU stress.



**Issue Report**:

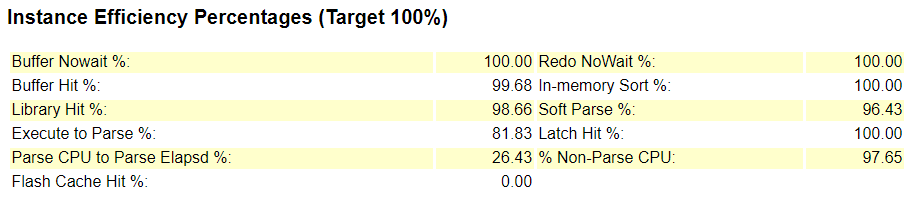


**OLTP**:

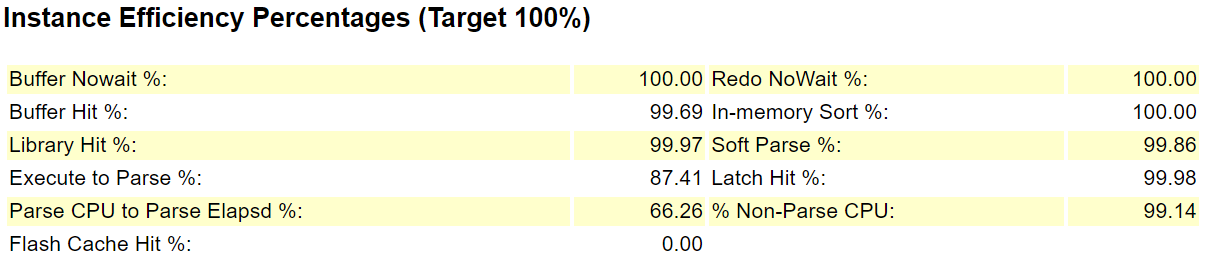


**Conclusion**: In Load profile section too we don’t see any stress on the system due to CPU stress.

**Issue Report**:



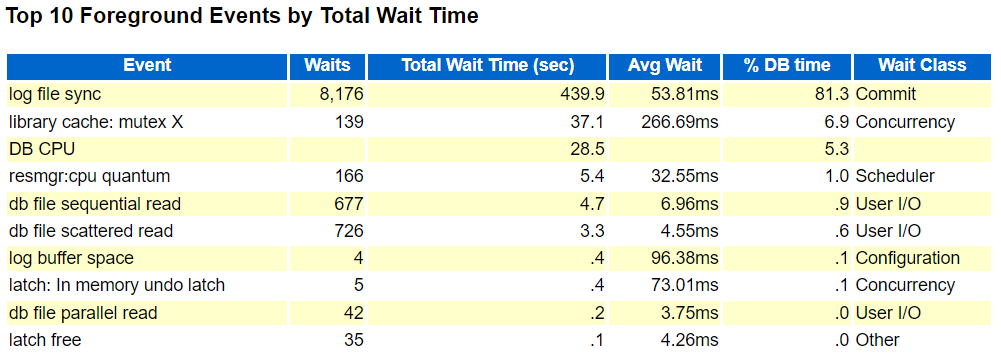
**OLTP report**:



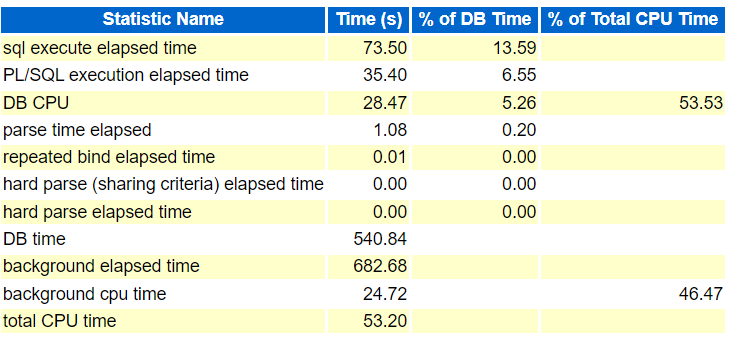
**Conclusion**: Again there is no difference.

**Issue Report**:

DB CPU is 5.3% of DB time.



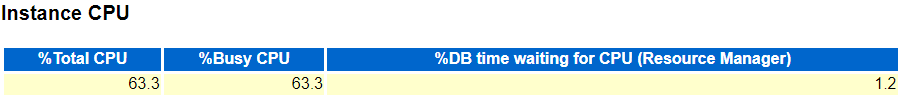
**Time Model Statistics**: It shows Total CPU Time is 53.20 seconds (DB CPU foreground + DB CPU background).



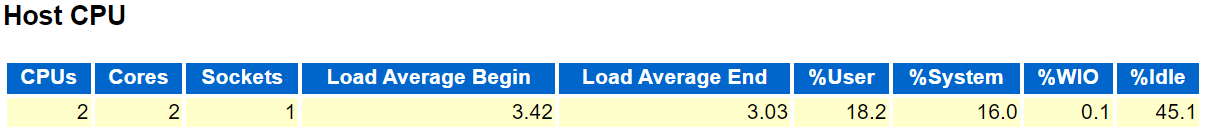
**Host CPU**: %User shows 52.% of CPU time is consumed by applications and DB. CPU is 50% loaded.



Instance CPU: %Total CPU is less than 100%. This implies the Host CPU is not quite loaded by DB applications.



**OLTP report**:



**Conclusion**: