Practice 7

Monitoring and Tuning Oracle RAC Database

Practice Overview

The target of this practice is practicing various ways to monitor and tune Oracle RAC database. This is **not** about tuning Oracle database in general. This is about understanding Oracle RAC-related performance monitoring and troubleshooting. Specifically, you will perform the following:

- Monitor the status of the clusterware resource, RAC database, and GRD memory usage.
- Use EM Express to narrow down a performance bottleneck and tune slow sessions
- Proof of concept study cases:
  - The impact of using sequences to generate unique values versus using tables.
  - The impact of enabling the cache option in the sequences. You will use the v$views as tuning tools in this study case.
  - Catching Transaction Enqueue locks in application sessions. You will use AWR reports as tuning tools in this study case.

Note: this practice is not to cover all the performance tuning techniques in Oracle. The targets of this practice are mentioned in the list above.

Practice Assumptions

- The practice assumes that you have the Oracle RAC database up and running in the virtual machines `srv1` and `srv2`.
- You have downloaded the scripts files from the downloadable section of this practice.

  The practice examples assume that the script files have been extracted in the directory `D:\scripts`. If you have extracted the files somewhere else, change the code accordingly when you run it.

Note: Not all the scripts will be used in this practice. Some script files will be used in future lectures.

A. Monitoring the RAC database

In this section of the practice, you will execute commands to check on the status of the cluster resources, the status of the RAC database, and the memory usage.

General checking on the nodes, ASM, applications

The following are very common commands used in RAC databases to check the status of their components.

1. Open a Putty terminal window and connect to `srv1` as `grid`.

2. Check the status of the Grid Infrastructure services.

```
crsctl check crs
```

3. Display the status of the resources registered in the Clusterware.

```
crs_stat
crs_stat -t
```

4. Switch user to `oracle` and check the status of the applications running in the nodes.

   Application as a term in the Clusterware context is different that the meaning of "application" in the database. Any resource managed by the Clusterware is called "application", including the database itself.

```
srvctl status nodeapps

# -n switch displays the status of the applications in specific node
srvctl status nodeapps -n srv1
```

5. Check on the status of the database and the ASM

```
srvctl status database -d rac
srvctl status asm
```

Checking on the RAC memory usage

GRD is a new locking layer added to Oracle RAC and does not exist in the single-instance database. In this sub-section of the practice, you will study how the impact of the GRD on the memory consumption in Oracle RAC database.

6. Login to `rac` in SQL*Plus as `sys` user.

```
sqlplus / as sysdba
```

7. Run the following query to retrieve the amount of memory allocated to GRD in the shared pool area.

```
SELECT NAME, ROUND(BYTES/1024/1024, 2) SIZE_MB
FROM V$SGASTAT
WHERE ROUND(BYTES/1024/1024, 2)<>0
  AND (name LIKE 'ges %' OR name LIKE 'gcs %')
ORDER BY 2;
```

8. Run the following query to monitor how the GRD memory allocation has risen or dropped over the last three days.

Sharp increase or drop should be investigated.

```
SELECT
TO_CHAR(BEGIN_INTERVAL_TIME, 'DD-MON-YYYY HH12:MI AM')TIME, INSTANCE_NUMBER,
TRUNC(MAX(BYTES/1024/1024),2) SZ_MB
FROM
 (SELECT BEGIN_INTERVAL_TIME, S.INSTANCE_NUMBER, SUM(BYTES) BYTES
  FROM
    DBA_HIST_SGASTAT G, DBA_HIST_SNAPSHOT S
  WHERE (NAME LIKE '%ges%' OR NAME LIKE '%gcs%')
    AND S.SNAP_ID = G.SNAP_ID
    AND S.INSTANCE_NUMBER = G.INSTANCE_NUMBER
    AND BEGIN_INTERVAL_TIME BETWEEN SYSDATE-3 AND SYSDATE
    GROUP BY BEGIN_INTERVAL_TIME, S.INSTANCE_NUMBER
  )
GROUP BY TO_CHAR(BEGIN_INTERVAL_TIME, 'DD-MON-YYYY HH12:MI AM'),INSTANCE_NUMBER
ORDER BY 1
/
```

B. Using EM Express for Tuning

In this section of the practice, you will use the EM Express to tune a slow session.

The performance scenario issue that you are trying to tune is as follows:

- The application users running as normal and they do not complain from any performance issue.
- One or more reporting users are complaining that a report is taking very long time to execute, although the amount of data retrieved by the report is small.

9. Just to save disk space, I recommend deleting archive logs before proceeding with the practice.

10. Start Swingbench and invoke the Benchmark run in it.

Sessions created by the Swingbench represent the users who are running their OLTP processing as normal.

11. Open a command prompt window in your hosting PC and login to `rac` in SQL*Plus as `soe` user.

```
sqlplus soe/soe@rac
```

12. Execute the following PL/SQL block. The block simply executes a select statement ten times. **Do not wait for the block execution to finish**. Go to the next step straight away after running the code block.

This session represents the user complaining from a slow report.

```
set timing on
DECLARE
  V_FNAME VARCHAR2(40);
BEGIN
DBMS_APPLICATION_INFO.SET_MODULE (module_name => 'REPORTING', action_name =>
'Customer Report');
 FOR I IN 1..10 LOOP
  SELECT MAX(CUST_FIRST_NAME) INTO V_FNAME
  FROM SOE.CUSTOMERS
  WHERE CUST_FIRST_NAME LIKE DBMS_RANDOM.STRING('A', 2)||'%';
 END LOOP;
END;
/
```

In this scenario, you communicate with the user and try to obtain a session identifier. The identifier could be anything like the SID, username, module name, program name, or any data the distinguish the complaint from the rest of sessions.

In this practice, we assume you will use the module name/action name to get access to the suffering session.

13. Open the browser and login to EM Express as `sys` user.

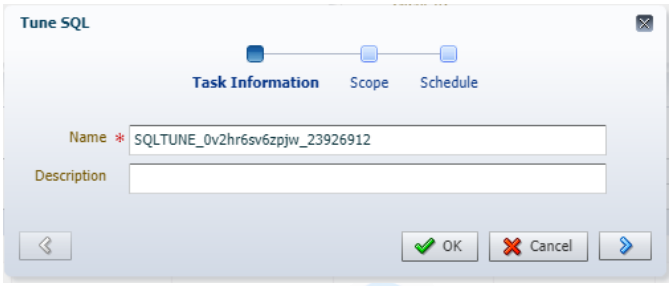
14. Open the **Performance Hub** | click on **Activity** tab | click on the **Wait Class** drop list | **Top Dimensions** | **Action**

15. Click on the **Customer Report** in the legend list of the chart.

16. Click on the most top **SQL ID** link. It should take you to the SQL Details of the selected statement. This statement is most likely the culprit statement.

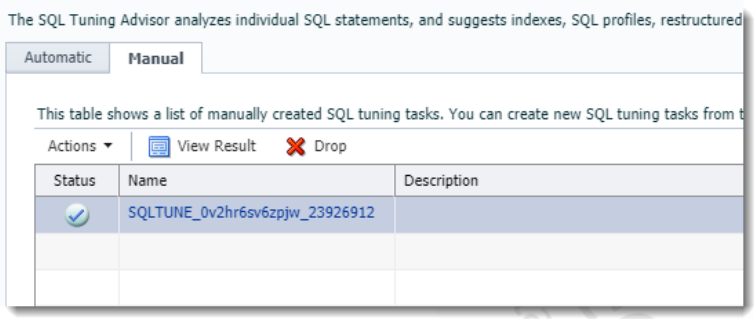
17. Click on **Tune SQL** button in the top of the window.

18. **Tune SQL** window similar to the following screenshot will appear to you.



19. Click on **Ok** button. The ADDM will invoke and process the selected statement.

20. You should see the tuning result appearing in a list as follows. If the **status** of the result is **not completed**, wait for it to finish and become completed.



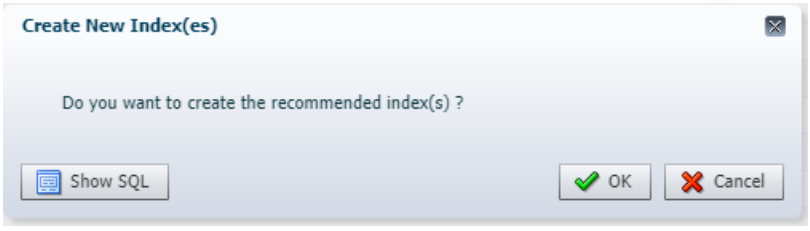
21. When the result is **completed**, click on it.

    It should display to you the SQL statement code and the ADDM recommendation to improve its efficiency.

    In our case, the advisor is recommending to create an index and it shows that the benefit from implementing this action would increase its efficiency by more than 99%.

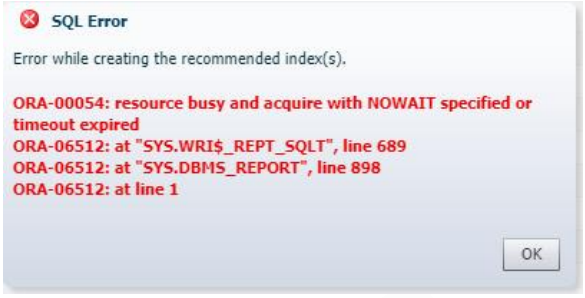
22. Select the recommendation then click on **View Details**. It displays to you the explain plans before and after the proposed implementation. Click on **Implement** button.

23. The following message should appear to you. If you click on **Show SQL**, it shows to you the script that the advisor will execute to implement the solution.



24. Click on **OK** button.

25. You may receive the following error message. This message is displayed because some sessions are locking the `CUSTOMERS` table.



26. Close the Swingbench sessions.

27. Go back to EM Express, and click on **OK** button to try re-implementing the solution.

28. You should receive a message informing you that the creating the index was successful. Click on **OK** button.

29. Start Swingbench and kick off its sessions again.

30. Wait for the Swingbench sessions to saturate.

31. Execute the reporting block again in the SQL*Plus client session.

32. Compare the execution time between executing the block before and after implementing the solution.

33. You should see a signification improvement in the execution time after implementing the solution. As this stage you consider the issue being resolved.

34. Stop the Swingbench sessions and close it.

35. Close all the sessions that you created.

C. Handling Sequences in Oracle RAC database

Sequences are usually used in Oracle database to generate unique values for primary key columns. It has always been a recommendation to increase the default cache size of the sequences in Oracle RAC database.

In this section of the practice, you will practically study:

(1) the difference in performance point of view between using the sequences to generate unique value for primary key columns and using a common table that stores the generated numbers.

(2) how increasing the sequence cache would contribute in increasing their performance.

For every following performance case study, you will follow the following procedure:

- Run `setupscript<n>.sql` to create the initialized state components (tables, sequences, and others)
- Run `script<n>.bat` script to execute the load. The script in most cases runs 10 SQL*Plus sessions in parallel. Each session is executing specific load against the components created in the previous step.
- Run `showstats<n>.sql` script to display the performance statistics of all the sessions executed in the previous step. It displays the top wait events for the sessions of the previous step.

Case Study 1: performance difference between using tables and using sequences for generating unique values

36. In the hosting PC, open a command prompt window and change the directory to the folder where you downloaded the script files.

```
cd D:\scripts
```

37. Examine the code in `setupscript1.sql`. It creates simply two tables. One of them (`SN` table) will be used to generate the primary key values and the other one (`TLOGS` table) will be used to insert some data in it.

38. Examine the code in `script1.sql`. It inserts 2000 rows into `TLOGS` table and rely on `VSN` table to increment the primary key values.

39. Examine the code in `showstats1.sql`. The script performs the following:

- retrieve and calculate how long the sessions took to load the table
- display the number of rows inserted by every instance
- retrieve the top wait events of the sessions and the average wait time for each session on the event. It prompts the user to enter the begin and end date and time. The sessions that were there during this period will be analyzed. This is to narrow down the sessions run in the previous step.

40. Run the following script.

```
sqlplus -silent soe/soe@rac @setupscript1.sql
```

41. Run the following script.

The script first displays the current date and time. It then runs `script1.sql` ten times in parallel. It waits for the last run to finish and prints the current date and time after the execution.

```
script1.bat
```

42. After all the sessions in the previous step finish their executions, run the following script to display the statistics about the parallel execution run performed in the previous step.

When it prompts to enter the begin time, copy/paste the required value from the output of the previous step.

When it prompts to enter the end time, copy/paste the required value from the output of the previous step.

Observe that the rows were equally inserted by the RAC instances. There could be some difference but it should be small.

```
sqlplus -silent "sys/oracle@rac as sysdba" @showstats1.sql
```

43. Copy the output of this script and paste it into a new text file. You will use those figures to compare the performance of this script with the performance of the other scripts that you will execute later.

In the following steps, you will insert the same number of rows using the same number of parallel executions but this time you will use a sequence to generate the primary key values.

44. Examine the code in `setupscript2.sql`. It simply creates a sequence and a table (`TLOGS` table) to insert some data in it.

45. Examine the code in `script2.sql`. It inserts 2000 rows into `TLOGS` table and rely on a sequence to generate the primary key values.

46. Examine the code in `showstats2.sql`. The script performs exactly the same code as in `showstats1.sql`.

47. Run the following script.

```
sqlplus -silent soe/soe@rac @setupscript2.sql
```

48. Run the following script. Again it will kick off ten sessions to run in parallel.

```
script2.bat
```

49. After all the sessions in the previous step finish their executions, run the following script to display their top wait statistics.

```
sqlplus -silent "sys/oracle@rac as sysdba" @showstats2.sql
```

50. Copy the output of this script and paste it into the text file. Compare the figures with the statistics obtained from the previous run. You will observe the following:

- The second run finished faster than the first run (%85 faster in my case).
- Some of the wait events in the first run were gone in the second run
- The wait time in the second run is much less than it in the first run.

Note: do not delete the text file. You will still use it for the next practice section.

Case Study 2: performance impact when increasing cache option in sequences

In the following steps, you will insert the same number of rows using the same number of parallel executions but this time you will increase the cache value of the sequence.

51. Examine the code in `setupscript3.sql`. The script creates the sequence with cache option equals to 1000.

52. The code in `script3.sql` and `showstats3.sql` scripts are typically the same as in the code in `script2.sql`. and `showstats2.sql` respectively.

53. Similar to what you have done in the previous case study, run the following scripts

```
sqlplus -silent soe/soe@rac @setupscript3.sql
script3.bat
sqlplus -silent "sys/oracle@rac as sysdba" @showstats3.sql
```

54. Copy the output of this script and paste it into the text file. Compare the figures with the statistics obtained from the previous run. You will observe performance improvement in terms of the execution time and wait events.

Case Study 3: monitoring "enq: TX" waits

"enq: TX " waits are observed in some badly designed applications. This wait event appears mostly when a transaction updates a table that is locked by some other transactions. This could occur in applications that allow transactions to be open for long time before commit or rollback.

The performance scenario issue that you are trying to tune is as follows:

- Most application users running as normal and they do not complain from any performance issue.
- One or more users complain that a specific form or process is taking long time to execute.

You learnt in the previous sections in this practice how to use EM Express and the V$VIEWS to monitor the performance statistics. In this section, you will use the AWR reports to analyze this performance issue.

55. Start Swingbench and invoke the Benchmark run in it. Sessions created by the Swingbench represent the normal workload.

56. Keep the Swingbench sessions running for 2 minutes.

57. Open a Putty session and login to `srv1` as `oracle` user.

58. Login to `rac` as `sysdba` and generate an AWR snapshot.

By default, every one hour, Oracle automatically creates a new AWR snapshot.

```
sqlplus sys/oracle@rac as sysdba

EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
```

59. Keep the sessions running for another 4 minutes or more then go to next step.

60. Generate another AWR snapshot.

```
EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
```

61. Display list of the AWR snapshot stored in the database.

Observe that in our case there are two AWR snapshots created for every SNAP_ID: one snapshot for every instance.

```
col begin_interval_time format a20
col end_interval_time format a20

SELECT INSTANCE_NUMBER, SNAP_ID,
 TO_CHAR(BEGIN_INTERVAL_TIME, 'DD-MM-YY HH24:MI:SS') AS BEGIN_INTERVAL_TIME,
 TO_CHAR(END_INTERVAL_TIME, 'DD-MM-YY HH24:MI:SS') AS END_INTERVAL_TIME
FROM   DBA_HIST_SNAPSHOT
WHERE  TRUNC(BEGIN_INTERVAL_TIME)=TRUNC(SYSDATE)
ORDER BY SNAP_ID;
```

62. Create a new AWR baseline. The START_SNAP_ID and the END_SNAP_ID are the SNAP_ID of the snapshots that you created earlier. They would be the last two snapshots retrieved by the SELECT statement that you run in the previous step.

The pair of snapshots associated with a baseline are retained until the baseline is explicitly deleted.

Note: in real life scenario, you would create a baseline of a full workload cycle. That could be an eight-hour work in a business day for a 5 days x 8 hours (5x8) applications.

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (
    START_SNAP_ID => <your start snap id>,
    END_SNAP_ID   => <your end snap id>,
    BASELINE_NAME => 'NORMAL WOKRLOAD');
END;
/
```

63. Examine the code in script4.sql. Its code updates a randomly selected row in TLOGS, waits for four seconds, then commits. It repeats this process four times.

64. Run the following script. Again it will kick off ten sessions to run in parallel. Those sessions represent the sessions who are complaining from slow process.

You can have a look at the script code before you run it.

```
script4.bat
```

At this stage, you are reported about the performance issue and you, therefore, decide to take an AWR snapshot in the period when the issue occurred to compare it with the normal baseline period.

Note: I understand that instead of using the AWR, you could monitor the wait events of the complaining sessions. However, you are using the AWR in this section of the practice just to have some practical experience on using the AWR to analyze performance issues.

65. Generate an AWR snapshot.

```
EXEC DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT;
```

66. Run the script `awrdgrpt.sql` (AWR Compare Period Report) located in the `$ORACLE_HOME/rdbms/admin` to generate a comparison report between the normal workload baseline and the period when the issue has occurred.

Note: in a single-instance database, you would run the script `awrddrpt.sql`

Note: Compare period report is different from the AWR report. Compare period report displays the difference between two pairs of AWR snapshots. AWR report displays the information about the events that took place between two AWR snapshots.

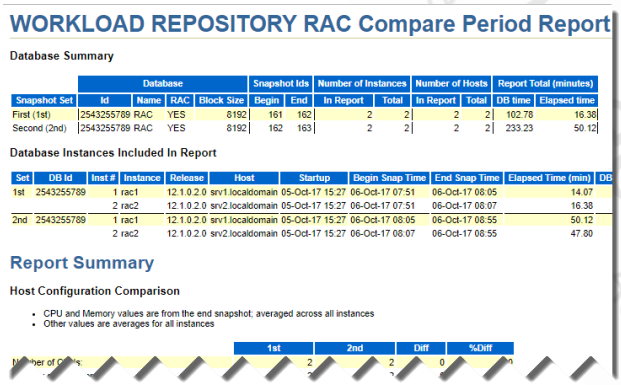
The script will ask you to enter the following:

- Database Id and Instance Number for the First Pair of Snapshots: enter the snap ids of the normal baseline. If you do not know them, query the view `DBA_HIST_BASELINE`.
- The Second Pair of Begin and End Snapshot Ids: enter the snap ids of the period when the issue took place.

```
cd $ORACLE_HOME/rdbms/admin/
sqlplus sys/oracle@rac as sysdba
@awrgdrpt.sql
```

67. Open the generated html file in a browser. It should look like the following screenshot:

In my case, I copied the generated file from the virtual machine to my hosting PC using WinScp utility and opened the file in a browser from the hosting PC.



WORKLOAD REPOSITORY RAC Compare Period Report

Database Summary

	Database				Snapshot Ids		Number of Instances		Number of Hosts		Report Total (minutes)	
Snapshot Set	Id	Name	RAC	Block Size	Begin	End	In Report	Total	In Report	Total	DB time	Elapsed time
First (1st)	2543255789	RAC	YES	8192	161	162	2	2	2	2	102.78	16.38
Second (2nd)	2543255789	RAC	YES	8192	162	163	2	2	2	2	233.23	50.12

Database Instances Included In Report

Set	DB Id	Inst #	Instance	Release	Host	Startup	Begin Snap Time	End Snap Time	Elapsed Time (min)	DB
1st	2543255789	1	rac1	12.1.0.2.0	srv1.localdomain	05-Oct-17 15:27	06-Oct-17 07:51	06-Oct-17 08:05	14.07	
		2	rac2	12.1.0.2.0	srv2.localdomain	05-Oct-17 15:27	06-Oct-17 07:51	06-Oct-17 08:07	16.38	
2nd	2543255789	1	rac1	12.1.0.2.0	srv1.localdomain	05-Oct-17 15:27	06-Oct-17 08:05	06-Oct-17 08:55	50.12	
		2	rac2	12.1.0.2.0	srv2.localdomain	05-Oct-17 15:27	06-Oct-17 08:07	06-Oct-17 08:55	47.80	

Report Summary

Host Configuration Comparison

- CPU and Memory values are from the end snapshot; averaged across all instances
- Other values are averages for all instances

	1st	2nd	Diff	%Diff
Number of CPUs:	2	2	0	0

68. Go to the **Reports Details** section | click on **Wait Stats** | click on **Enqueue Activity**

For "TX-Transaction" wait, compare the **Wait Time** and the **Requests** between the snapshots. Observe that there was hardly any time spent on waiting for this wait type in the first snapshot set whereas there are plenty of them in the second snapshot set. This is a clear indication that you are facing an enqueue lock situation.

You have seen a jump in the enqueue wait event in the difference analysis report. Now, you want to know which statements caused those wait events.

The report itself does not link the wait figures with the statements that caused them but you can get the link yourself.

69. In a SQL*Plus session, login as `sys` to `rac` and run the following query. Replace the <begin snap_id> and <end snap_id> with the start snap id and end snap id of the troubling period.

The query retrieves the `SQL_ID` of all the SQL statements that were run in the second snapshot set and waited for enqueue wait event.

```
sqlplus / as sysdba

col snap_id format a3
col sql_opname format a10
col sql_plan_operation format a10
col event format a30
col avg_time format a9

SELECT to_char(A.SNAP_ID) SNAP_ID, A.SQL_OPNAME, A.SQL_PLAN_OPERATION, A.EVENT,
A.SQL_ID, to_char(AVG(TIME_WAITED)) AVG_TIME
FROM   DBA_HIST_ACTIVE_SESS_HISTORY A, DBA_HIST_SNAPSHOT X
WHERE  A.SNAP_ID = X.SNAP_ID AND TIME_WAITED <> 0
  AND  UPPER(A.EVENT) LIKE UPPER('ENQ: TX%')
  AND  A.SNAP_ID BETWEEN <begin snap_id> AND <end snap_id>
GROUP BY A.SNAP_ID, A.SQL_OPNAME, A.SQL_PLAN_OPERATION, A.EVENT,  A.SQL_ID;
```

70. Go back to the Compare Period Report and search for the SQL_ID returned from the query above.

71. You should find the `UPDATE` statement to the `TLOGS` table. You should then go to the developer team and check out from which application module that update statement executes. When you have a look at its code, you will notice that the code does a lot of time processing before it commits the update statement.

Note on the Compare Report

The AWR report has the following three tables that display information about RAC-specific latencies:

- **Global Cache Load Profile**: displays statistics about the number of cache blocks and GCS/GES messages.
- **Global Cache and Enqueue Services - Workload Characteristics**: displays statistics about Global Cache Current/CR blocks. The time to process a Current/CR block is the total of pin/build + send + flush.
- **Global Cache and Enqueue Services - Messaging Statistics**: displays statistics about messages queue.

In a RAC database, those tables should regularly be monitored and compare their contents over the workloads. Constant increase in their figures is usually a symptom of a performance issue.

72. Close the report page.

73. Close and exit Swingbench.

74. Delete archive log files.

75. Perform the following cleanup steps.

```
sqlplus sys/oracle@rac as sysdba

DROP TABLE SOE.TLOGS;
```

Summary

- Monitoring the entire RAC database involves the following:
  - Monitoring the Clusterware stack
  - Monitoring the Database
- The most powerful Oracle EM Express usage is managing Oracle database performance. You can monitor the database performance, identify bottlenecks, and resolve performance issues using ADDM.
- In an Oracle RAC environment, it is highly advisable to enable the cache option in sequences and set it to a high value. Just remember, this solution leads to have the rows inserted from different instances to be not in sequence. The application should accept this side effect.
- Transaction enqueue should be as least as possible in any application. AWR reports can be used to identify transaction enqueue.
- The performance tools that you used in this practice are:
  - performance views (v$ views)
  - EM Express
  - AWR Reports