Performance monitoring in SQL*Plus using AWR and analytic functions

Marcin Przepiórowski
Version 1
Dublin, Ireland

Keywords:

AWR, Automatic Workload Repository, performance, SQL, analytic functions

Introduction

The Automatic Workload Repository (AWR) is an Oracle Enterprise Edition feature since version 10g. AWR is working in background and is gathering performance metrics from dynamic view and storing them in persistent tables. This approach allows any user who has access to those tables to learn how database performed in the past. It also allows DBA to check how application behaviour (like plan change) or database workload had been changed over the time.
This paper describes how to create an efficient query to access AWR data and how to display data over a time and find out anomalies.

AWR Internals

The Automatic Workload Repository is configured out of the box but in can be used only if database has a licence for a Diagnostic Pack. AWR persistent tables are kept in SYSAUX table space. There is a background process started by Oracle instance (MMON) which is taking care about gathering data when snapshot time is due. By default there are hourly snapshots and data retention is set to 7 days. This configuration can be changed via OEM GUI or using PL/SQL package DBMS_WORKLOAD_REPOSITORY. Information about the current configuration can be displayed using view called DBA_HIST_WR_CONTROL.

The AWR tables are named using the following pattern WRH$_XXX and most of the tables are partitioned by time and DBID. Oracle database does housekeeping for those tables based on requested retention time.

Users can access all data through DBA_HIST views. There are 111 DBA_HIST views in 11g and 140 in 12g. In case when user wants to keep data for long period and also consolidate AWR information from various Oracle databases there is a possibility to offload data from database into files and import them into the AWR Warehouse database. This operation can be based on user's scripts and AWR scripts to export and import data or in can be automated using Oracle Enterprise Manager Cloud Control 12c (since 12.1.0.4+).

Analytic functions

Analytic functions compute an aggregated value on a group of rows called “window”. In opposite to aggregate functions used with “group by” clause – analytic functions are returning a value for every row in group (window).

Let’s start with simple example:

```
select r, e, sum(r) over ()
from (
              select rownum r, mod(rownum,2) e
              from dba_source where rownum < 11
)

         R          E SUM(R)OVER()
---------- ---------- ------------
         1          1           55
         2          0           55
         3          1           55
         4          0           55
         5          1           55
         6          0           55
         7          1           55
         8          0           55
         9          1           55
        10          0           55
```

Ten rows are returned by sub-query with R column as row number and E column set to 1 for odd and 0 for even numbers. Analytic function sum (R) is using a window (group of rows) defined in over () clause. As this definition is empty all rows will be a part of the group and sum of all rows from column R is equal to 55.

Let’s add some complexity to the group definition

```
select r, e, sum(r) over (partition by e)
from (
              select rownum r, mod(rownum,2) e
              from dba_source where rownum < 11
)

         R          E SUM(R)OVER(PARTITIONBYE)
---------- ---------- ------------------------
         4          0                       30
         8          0                       30
         2          0                       30
         6          0                       30
        10          0                       30
         9          1                       25
         7          1                       25
         3          1                       25
         1          1                       25
         5          1                         25
```

Again ten rows are returned by sub-query with R column as row number and E column set to 1 for odd and 0 for even numbers. This time analytic function sum (R) is going to summarise values from column R using a window (group of rows). Window definition is using a partition clause which split rows into two subgroups based on value of column E – one group for odd and one for even rows. Sum function will aggregate column R against that subgroups like seen on above listening.

List of analytics functions useful for AWR data research.

The documentation of analytic functions can be found in the Oracle Database Data Warehousing Guide book.

lag() – this function is providing an access to rows prior a current one with required offset. In this presentation it will be used to access previous row to calculate delta between current and previous one

lead() - this function is providing an access to rows beyond a current one with required offset. In this presentation it will be used to access previous row to calculate delta between current and the next one

rank() – this function calculates the rank of a particular value in a group of values in defined window. In this presentation it will be used to find top N rows

dense_rank() – this function calculates the rank of a particular value in a group of values in defined window. Opposite to the rank() function a ranks are consecutive integers beginning with 1. In this presentation it will be used to find top N rows

Look at the difference between a rank and a dense_rank in the following example

```
select r, e,
dense_rank() over (partition by e order by r) dense_rank,
rank() over (partition by e order by r) rank
from (
  select mod(rownum,4) r, mod(rownum,2) e
  from dba_source where rownum < 11
)

         R          E DENSE_RANK       RANK
---------- ---------- ---------- ----------
         0          0          1          1
         0          0          1          1
         2          0          2          3
         2          0          2          3
         2          0          2          3
         1          1          1          1
         1          1          1          1
         1          1          1          1
         3          1          2          4
         3          1          2          4
```

min() – this function returns a minimum value in the defined window.

max() – this function returns a maximum value in the defined window.

sum() – this function returns a sum of all values in the defined window.

Using SQL to access AWR data.

Here is a simple example of building a query accessing AWR views using analytic functions to calculate results across time. In this example workload load profile displaying number of Logical IO operations per second and physical reads per second will be displayed for every sample from the AWR repository.

1. In the first step a required statistic will be taken from DBA_HIST_SYSSTAT view for two particular snapshots. This example is going to calculate a number of physical and logical read per second. In this case needed statistics are:
   - physical reads
   - consistent gets
   - db block gets

```
select STAT_NAME, value, …
from dba_hist_sysstat
where stat_name in ….

STAT_NAME                 VALUE    SNAP_ID
-------------------- ---------- ----------

physical reads        439412650      45377
consistent gets      5436891942      45377
db block gets          42173568      45377
physical reads        443619497      45378
consistent gets      5694658872      45378
db block gets          42298564      45378
```

2. In this step values for particular statistics will subtracted from previous ones to calculate delta between two snapshots. Analytic function lag is used for that together with partitioning over statistic name. The query from a previous point has been used as a source of data.

```
select
   snap_id,
   stat_name,
   value -
   lag(value) over (partition by stat_name order by snap_id) delta
from ( query from previous point );

 SNAP_ID STAT_NAME             DELTA
---------- ---------------- ----------
     45378 consistent gets   257766930
     45378 db block gets        124996
     45378 physical reads      4206847
```

3. In this step values statistics are divided into two columns, this will allow to group logical and physical requests into columns and will prepare it for “dummy” pivot operation. Column PR has a physical reads statistics for one row and zero for other rows while column LR will have non-zero numbers for “consistent gets” and “db block gets”. The query from a previous point has been used as a source of data plus there is a join with DBA_HIST_SNAPSHOT to translate snap_id into dates.

```
     select
        s.snap_id,
        cast(begin_interval_time as date),
        decode(stat_name, 'physical reads', delta, 0) pr,
        decode(stat_name, 'physical reads', 0, delta ) lr
     from
        (query from previous point) s,
        dba_hist_snapshot ss
     where …..

SNAP_ID BEGIN_INTERVAL_TIME          PR         LR
--------- ------------------- ---------- ----------
    45378 17-SEP-13 12.00.44          0  257766930
    45378 17-SEP-13 12.00.44          0     124996
    45378 17-SEP-13 12.00.44    4206847          0
```

4. In this step the number of seconds between snapshots is calculated using an analytic function and there is an aggregation operation on “PR” and “LR” column. This is typical aggregated function sum with group by on snapshot time.

```
     select
       begin_interval_time,
      (
       min(begin_interval_time)-(lag(min(begin_interval_time)
      ) over (order by snap_id)) )*24*60*60 sec,
      sum(pr) pr,
      sum(lr) lr
      from (query from previous point)
      group by begin_interval_time

BEGIN_INTERVAL_TIME    SEC        PR          LR
------------------- ------ ---------- ----------
```

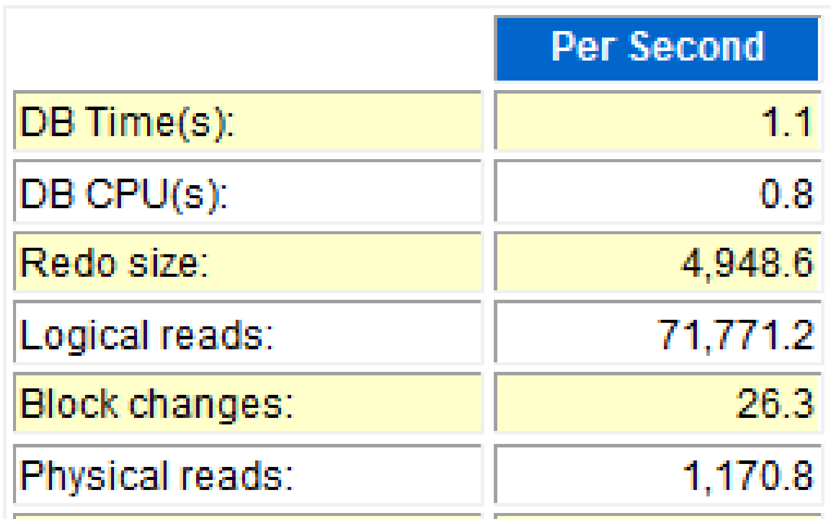
```
17-SEP-13 12.00.44    3596    4206847  257891926
```

5. In the last step there is only simple calculation to divide “PR” and “LR” column over number of second to provide per sec values.

```
select
   begin_interval_time,
   pr/sec "phy read / sec",
   lr/sec "log read / sec"
from (query from previous point)
```

```
BEGIN_INTERVAL_TIME  phy read / sec log read / sec
-------------------- -------------- --------------
17-SEP-13 12.00.44          1169.87       71716.33
```

The results shown above are calculated by SQL and now we can compare it with AWR report header. There are small differences but in general both numbers are very close to each other.



	Per Second
DB Time(s):	1.1
DB CPU(s):	0.8
Redo size:	4,948.6
Logical reads:	71,771.2
Block changes:	26.3
Physical reads:	1,170.8

Illustration. 1: Header of the AWR report

The whole example query and other ones used during presentation can be found in my repository on GitHub using this URL

Example query:
`https://github.com/pioro/ashmasters/load_awr.sql`

Whole repository:
`https://github.com/pioro/ashmasters/`

Your contact details at the end of your article.

Contact address:

Marcin Przepiorowski
Version 1
Dublin, Ireland

Phone:	+353 85 75 85 640
Email	marcin.przepiorowski@gmail.com
Internet:	http://oracleprof.blogspot.ie/