[admin](http://feed.askmaclean.com/archives/author/admin)  [2013/07/10](http://feed.askmaclean.com/archives/oracle-12c-ilm-management-part-3-heat-map.html" \o "19:18) [12c](http://feed.askmaclean.com/archives/category/12c), [ILM](http://feed.askmaclean.com/archives/category/ilm), [genernal](http://feed.askmaclean.com/archives/category/genernal)

[加入dbDao.com Oracle技术学习QQ群：171092051](http://jq.qq.com/?_wv=1027&k=ZmWvtL" \t "_blank)

[如果自己搞不定可以找诗檀软件专业ORACLE数据库修复团队成员帮您恢复!](http://www.parnassusdata.com/zh-hans/emergency-services" \t "_blank)

诗檀软件专业数据库修复团队

服务热线 ： 400-690-3643 备用电话： 13764045638 邮箱：service@parnassusdata.com

In the last two parts, we saw two new approaches available in 12c for the better management of the data that’s considered to be stale or inactive in a database-In-Database Archiving and Temporal Validity. The key point in both the techniques is to make it possible that the data which is not relevant to the business shouldn’t impact the regular queries and the DML’s. Both the techniques are indeed good but there is still room for more options when it comes to manage multi-gigabytes(or terabytes) of data which probably is not required in the daily usage of the business and some action needs to be taken for it. This post would extend the discussion further for the better ways of doing ILM in 12c.

**Pre 12.1 way of managing things!**

Before we would discuss what is new in 12c, let’s try to understand how things are done till now when it comes to manage large sized data sets which are considered stale as per the business policy and to illustrate, let’s take an example of a telecom company. In telecom business, the data which is related to the most current month is of the highest importance, obviously because that’s going to bring the rental money. The data that’s preceding the current month is important but not that much and moreover, its not the whole set of the old data that’s going to be useful. For example, as the current month is July so the data belonging to this month would be used in order to generate the bills for the subscribers next month and thus, this month’s data is extremely important and not just for the reason that it’s going to bring money but in terms of performance as well because using it only, the itemized bills, reports etc. would be generated so the access of this must be fast enough. The data of the month of June, May can be considered relevant because of reporting, for example if we consider that in the quarterly report would be generated from the cumulative access of all the three months, May, June and July. But what about January’s data or even of months before that? Is that is having the same importance as that of June or rather the better question would be, should that data have the same importance? The answer would be a NO because it’s very old data and though we shouldn’t say the data is completely useless now, after 6 months later but rather that its very less important now. So what we can do to manage such kind of data distributions which become obsolete after some time on the object level? Probably the most effective technique which comes to mind is *Partitioning* and since we have taken an example of telecom sector, it would be a perfect fit for such kind of data as well!For each month, a separate partition needs to be created and the partition of the most current month would also be the most active one. Now, for the older partitions, since they are not going to be of much in use, you have couple of choices like moving them to a slower storage or along with it compressing the data so that it won’t consume much of the space and after some time, compress it using some aggressive compression algorithm, make it read-only and move it to some other facility where it would be kept till the time what your company decides as per its policy. And why to do all this, to make a better use of the storage that you have with you.

So what’s the issue in all this when this is what DBA’s have been doing already from many years? Well, there isn’t any other issue except one, a DBA has to do it all with a proper strategy and planning and when the time comes to take actions like making the segment compressed or moving it to some less performance centric storage, do those tasks manually! So to make your life easier, 12c has made this process automatic, well almost automatic to be precise! How? By using two new techniques where one is going to monitor your segments i.e. activities happening over them like queries and DML’s, to even the fine level of data blocks and extents and the second technique, based on the results of the first would decide what action(s) to take, for example whether to compress the data or compress it and move it to some alternative storage etc.

In this post, we shall look at the first technique, **[Heat Map](http://docs.oracle.com/cd/E16655_01/server.121/e17906/chapter1.htm" \l "NEWFT232)** which would be used to monitor the activities of the segments.

**Heat Map, getting started!**

A very interesting title isn’t it but it is indeed a very appropriate one as well! The meaning of Heat Map is to track what’s the *temperature* of your object like a table or a partition of a table, in terms of “accessibility”, for both selects and DML’s or in other words, for both reads and writes done over the object. To put simply, it’s the tracking of the “hotness” of your object! There can be three kinds of temperatures possible,

* **Hot**
* **Warm**
* **Cold**

The Hot temperature would mean that the object is actively participating in both Read as well as Write operations. Warms stands for that object which is accessed but only via queries i.e. for reads only and the Cold means the object is not participating in any kind of activity and thus can be safely considered as a suitable candidate for being moved to a facility(for example to a tablespace sitting on a slow storage) which would hold such kind of “stale data”. Using such kind of heat tracking for objects, it would be very easy to take such ILM decisions!

**Enabling Heat Map using HEAT\_MAP parameter**  
Now, this shouldn’t be any surprise that since the feature’s name is Heat Map, the parameter which would enable it would also be called **HEAT\_MAP** only!This parameter is possible to be modified at either session or system level and would enable or disable the segment-activity tracking. The statistics which are collected over a segment are stored first in the memory in the view V**$HEAT\_MAP\_SEGMENT**(X$HEATMAPSEGMENT) before being pushed over to disk every hour using the DBMS\_SCHEDULER interface.The statistics collection is in real-time and is categorized as

* **Segment Access(either by Full Table Scan or a Lookup Scan)**
* **Segment Modification**
* **Row Modification**

So let’s see what’s the value of the parameter HEAT\_MAP is in our non-container database ORCL12. This is very important to remember because at this moment, with 12.1, the feature of heat map is **NOT** supported for multitenant databases.

SQL> SHOW PARAMETER HEAT

NAME TYPE VALUE

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

heat\_map string OFF

SQL>

So the parameter is set to OFF which means no tracking of any segment is going to be done by default. Since there is no tracking going on, there must not be any thing recorded in the view V$HEAT\_MAP\_SEGMENT as well.

SQL> select \* from V$heat\_map\_segment;

no rows selected

So now, let’s modify the parameter to value ON and see the effect of it by accessing tables of Scott schema in both the queries and in DML’s.

SQL> alter session set heat\_map=on;

Session altered.

SQL> select \* from scott.emp;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 7902 17-DEC-80 800 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7566 JONES MANAGER 7839 02-APR-81 2975 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 1100 20

7900 JAMES CLERK 7698 03-DEC-81 950 30

7902 FORD ANALYST 7566 03-DEC-81 3000 20

7934 MILLER CLERK 7782 23-JAN-82 1300 10

14 rows selected.

SQL> select \* from v$heat\_map\_segment;

OBJECT\_NAME SUBOBJECT\_NAME OBJ# DATAOBJ# TRACK\_TIM SEG SEG FUL LOO CON\_ID

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

EMP 91902 91902 10-JUL-13 NO NO YES NO 0

SQL> select \* from scott.salgrade;

GRADE LOSAL HISAL

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

1 700 1200

2 1201 1400

3 1401 2000

4 2001 3000

5 3001 9999

SQL> select \* from v$heat\_map\_segment;

OBJECT\_NAME SUBOBJECT\_NAME OBJ# DATAOBJ# TRACK\_TIM SEG SEG FUL LOO CON\_ID

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

SALGRADE 91953 91953 10-JUL-13 NO NO YES NO 0

EMP 91902 91902 10-JUL-13 NO NO YES NO 0

SQL> create table scott.e as select \* from scott.emp;

Table created.

SQL> select \* from scott.e;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 7902 17-DEC-80 800 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7566 JONES MANAGER 7839 02-APR-81 2975 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 1100 20

7900 JAMES CLERK 7698 03-DEC-81 950 30

7902 FORD ANALYST 7566 03-DEC-81 3000 20

7934 MILLER CLERK 7782 23-JAN-82 1300 10

14 rows selected.

SQL> select \* from v$heat\_map\_segment;

OBJECT\_NAME SUBOBJECT\_NAME OBJ# DATAOBJ# TRACK\_TIM SEG SEG FUL LOO CON\_ID

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

E 92277 92277 10-JUL-13 NO NO YES NO 0

SALGRADE 91953 91953 10-JUL-13 NO NO YES NO 0

EMP 91902 91902 10-JUL-13 NO NO YES NO 0

SQL> delete scott.e where empno=7369;

1 row deleted.

SQL> commit;

Commit complete.

SQL> select \* from v$heat\_map\_segment;

OBJECT\_NAME SUBOBJECT\_NAME OBJ# DATAOBJ# TRACK\_TIM SEG SEG FUL LOO CON\_ID

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

E 92277 92277 10-JUL-13 YES NO YES NO 0

SALGRADE 91953 91953 10-JUL-13 NO NO YES NO 0

EMP 91902 91902 10-JUL-13 NO NO YES NO 0

SQL>

SQL> select \* from scott.e where empno=7499;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

SQL> select \* from V$heat\_map\_segment;

OBJECT\_NAME SUBOBJECT\_NAME OBJ# DATAOBJ# TRACK\_TIM SEG SEG FUL LOO CON\_ID

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

E 92277 92277 10-JUL-13 YES NO YES NO 0

SALGRADE 91953 91953 10-JUL-13 NO NO YES NO 0

EMP 91902 91902 10-JUL-13 NO NO YES NO 0

The view V$HEAT\_MAP\_SEGMENT is an in-memory view and that means, the statistics collected in it won’t be persistent. So these statistics are flushed to a more persistent table HEAT\_MAP\_STAT$ table. As mentioned before as well, this flushing is scheduled using the scheduler interface. After being flushed, you can view the same information from the views **DBA\_HEAT\_MAP\_SEG\_HISTOGRAM**and **DBA\_HEAT\_MAP\_SEG**. So let’s do a database bounce which would flush the view and we shall re-populate it by accessing the EMP table of Scott schema.

SQL> startup force

ORACLE instance started.

Total System Global Area 417546240 bytes

Fixed Size 2289064 bytes

Variable Size 339739224 bytes

Database Buffers 71303168 bytes

Redo Buffers 4214784 bytes

Database mounted.

Database opened.

SQL> @heatmap

no rows selected

So we can see that the view is empty after the database restart. So let’s enable the heat-tracking by modifying the parameter again.

SQL> alter session set heat\_map=on;

Session altered.

SQL> select \* from scott.emp;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7369 SMITH CLERK 7902 17-DEC-80 800 20

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

7521 WARD SALESMAN 7698 22-FEB-81 1250 500 30

7566 JONES MANAGER 7839 02-APR-81 2975 20

7654 MARTIN SALESMAN 7698 28-SEP-81 1250 1400 30

7698 BLAKE MANAGER 7839 01-MAY-81 2850 30

7782 CLARK MANAGER 7839 09-JUN-81 2450 10

7788 SCOTT ANALYST 7566 19-APR-87 3000 20

7839 KING PRESIDENT 17-NOV-81 5000 10

7844 TURNER SALESMAN 7698 08-SEP-81 1500 0 30

7876 ADAMS CLERK 7788 23-MAY-87 1100 20

7900 JAMES CLERK 7698 03-DEC-81 950 30

7902 FORD ANALYST 7566 03-DEC-81 3000 20

7934 MILLER CLERK 7782 23-JAN-82 1300 10

14 rows selected.

SQL> @heatmap

OBJECT\_NAME TRACK\_TIM SEG SEG FUL LOO

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

EMP 10-JUL-13 NO NO YES NO

So

SQL> select \* from scott.emp where empno=7499;

EMPNO ENAME JOB MGR HIREDATE SAL COMM DEPTNO

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

7499 ALLEN SALESMAN 7698 20-FEB-81 1600 300 30

SQL> @heatmap

OBJECT\_NAME TRACK\_TIM SEG SEG FUL LOO

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

PK\_EMP 10-JUL-13 NO NO NO YES

EMP 10-JUL-13 NO NO YES NO

So we have got the view populated for both the Full table scan and also for a Lookup i.e. index based scan. Please note that this access is tracked for even partitions as well. Let’s write into our table by doing a simple insert into it.

SQL> insert into scott.emp values (2222,'Aman','DBA',null,sysdate,1000,null,40);

1 row created.

SQL> commit;

Commit complete.

SQL> @heatmap

OBJECT\_NAME TRACK\_TIM SEG SEG FUL LOO

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

PK\_EMP 10-JUL-13 YES NO NO YES

EMP 10-JUL-13 YES NO YES NO

So indeed that’s also is tracked in the view! Now, let’s check the outputs from the views DBA\_HEAT\_MAP\_SEG\_HISTOGRAM and DBA\_HEAT\_MAP\_SEGMENT.

SQL> select object\_name, track\_time, segment\_write, full\_scan, lookup\_scan

2 from dba\_heat\_map\_seg\_histogram;

OBJECT\_NAME TRACK\_TIM SEG FUL LOO

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

EMP 10-JUL-13 YES YES NO

PK\_EMP 10-JUL-13 YES NO YES

SQL>

SQL> select object\_name, segment\_write\_time, segment\_read\_time, full\_scan, lookup\_scan

2 from dba\_heat\_map\_segment;

OBJECT\_NAME SEGMENT\_W SEGMENT\_R FULL\_SCAN LOOKUP\_SC

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

EMP 10-JUL-13 10-JUL-13

PK\_EMP 10-JUL-13 10-JUL-13

SQL>

**Using DBMS\_HEAT\_MAP package to go even deeper**

The default access is only going to let you know at segment level the details of the statistics. But if you want to go more deeper, for example to check which blocks were accessed and at what time, you would need to use the new package**[DBMS\_HEAT\_MAP](http://docs.oracle.com/cd/E16655_01/appdev.121/e17602/d_heat_map.htm" \l "BEHDHBCB)**. Using it, you can query the heat mapping at different levels like,

1. **Object level**
2. **Segment level**
3. **Tablespace level**
4. **Extent level**
5. **Block level**

Here is an example of using the package to access the information for the blocks which were accessed for the EMP table.

SQL> select segment\_name, block\_id

from table(dbms\_heat\_map.block\_heat\_map('SCOTT','EMP')); 2

SEGMENT\_NAME BLOCK\_ID

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

EMP 195

EMP 196

EMP 197

EMP 198

EMP 199

**Heat Map, old wine in a new bottle? Not really!**

If you are wondering that what’s so new about this Heat Map feature and isn’t it the same or kind-of-same to Auditing the table or like monitoring of the read and write activities on the table using the views like V$SEGMENT\_STATISTICS, V$OBJECT\_USAGE etc. which are there in the prior releases well than the answer is a NO. What you do with the Auditing or with the statistics like how many reads, writes done on a table in the previous versions is very basic and that information can’t be used in any intelligent way by the database itself. Yes, being a DBA you make take some actions manually but that’s about it! Heat Map, on the other hand, though appears to work in the same fashion but is meant for an entirely different requirement and that is to decide what to do intelligently and automatically for the data based on the access patterns of its usage. How to do it? Well, we shall see that in the next post where we shall use this heat map data in creating policies over our objects which would be applied based on the conditions that we shall mention. Sounds cool isn’t it? Well it is indeed a very cool stuff so stay tuned for the next part of this series!

**References**

<http://docs.oracle.com/cd/E16655_01/server.121/e17613/part_lifecycle.htm#CHDCEGFE>

[关注dbDao.com的新浪微博](http://weibo.com/liumaclean)

扫码关注dbDao.com 微信公众号：

[关注dbDao.com的新浪微博](http://weibo.com/liumaclean)

[扫码加入微信Oracle小密圈，了解Oracle最新技术下载分享资源](http://t.xiaomiquan.com/qjIMrf2)