Shared Pool Purging

中国 Oracle 用户组

作者：黄勇

http://www.acoug.org

版本	发布时间
1.0	2013/1/30

Shared pool purging can be done to increase free memory of the shared pool, or to force a re-parse of certain SQL statements. Generally, various methods of purging shared pool objects can fit into one of the following three groups, in order of increasing granularity:

(A) Flush the whole shared pool
(B) Purge all library cache objects referencing a specific object
(C) Purge one single library cache object

Flushing the whole shared pool is done with *alter system flush shared_pool*, excluding objects pinned (being executed) or keep'ed (v$db_object_cache.pin_mode is anything other than 'NONE' or kept is 'YES', respectively; the same exclusion rule applies hereinafter). This is the simplest way to clean up the shared pool, and if the requirement is to have a close to completely clean shared pool, this is the only method. Because of its big impact on the running instance, this should never be done in Production lightly. In the past, flushing shared pool before business peak hours was sometimes suggested to avoid ORA-4031. Nowadays, ORA-4031 does not occur as often as before, so any old nightly cron job to flush shared pool may not be necessary.

A better focused flushing of the shared pool is through invalidation of certain library cache objects. Before 11*g*, common practice is to run a harmless DDL on a table or view referenced in the SQL statements, such as *grant select on emp to dba*, which removes all SQLs that operate on emp from the shared pool. (But be aware that not all DDLs invalidate cursors, *alter table shrink space, alter index coalesce*, etc.) This is usually done when the DBA finds that an application uses literal values, piling up thousands of similar SQLs in the shared pool. Or the SQLs of the application lists column names in different permutations in the select-list or where-clause. Or a developer runs a large number of insert statements with literal values (instead of using SQL*Loader), or uses a tool such as Microsoft Visio to create an ER diagram. The solutions include using bind variables in the application (technically the best), setting cursor_sharing to force, and as a last resort, just purging those cursors with a simple DDL.

Beginning with 11*g*, a DDL on an object (table, view, etc) no longer purges the cursors referencing this object, although invalidation still occurs. That is, the status column of v$sql(area) or v$db_object_cache changes to INVALID_UNAUTH and invalidations increments by 1, but sharable_mem of these views stays the same. It's not clear why the new version Oracle keeps the invalid library cache objects in memory. One theory is that perhaps it makes the next parsing faster, but a small test shows no parsing speed advantage. The disadvantage, however, is obvious: a large number of such cursors, although invalidated, still take precious memory, which can be purged either by flushing the entire shared pool or using dbms_shared_pool.purge, discussed below.

The third method of shared pool purging targets one library cache object at a time (but multiple child cursors belonging to one parent cursor). This is achieved by the purge procedure of dbms_shared_pool package, which may need to be installed with $ORACLE_HOME/rdbms/admin/dbmspool.sql. Although this procedure is available in 10.2.0.4, that version needs to set event 5614566 (*alter session set events '5614566 trace name context forever*'). The common usage is to purge a cursor using the first of the multiple overloaded syntax formats (two other overloaded formats mostly throw ORA-06570: shared pool object does not exist, cannot be pinned/purged).

```
PROCEDURE PURGE

 Argument Name                  Type                    In/Out Default?
```

```
------------------------------ ---------------------- ------ --------

 NAME                           VARCHAR2               IN
 FLAG                           CHAR                   IN     DEFAULT
 HEAPS                          NUMBER                 IN     DEFAULT
```

For example,

```
SQL> select address, hash_value, executions, loads, version_count, invalidations, parse_calls
  2  from v$sqlarea
  3  where sql_text = 'select ename from emp where empno=7900';

ADDRESS          HASH_VALUE EXECUTIONS      LOADS VERSION_COUNT INVALIDATIONS PARSE_CALLS
---------------- ---------- ---------- ---------- ------------- ------------- -----------
000000007A6CF430 1052545619          1          1             1             0           1

SQL> exec dbms_shared_pool.purge('000000007A6CF430,1052545619','C', 65)
```

To purge a large number of SQLs having a specific attribute, such as parsing schema being SCOTT, you have to generate the SQLs yourself:

```
select 'exec sys.dbms_shared_pool.purge(''' || address || ',' || hash_value || ''',''c'',65)'
from v$sqlarea where parsing_schema_name = 'SCOTT';
```

But you don't want to do that directly because the dynamic SQLs generated that way in turn pollute the shared pool, so you have to use bind variables when calling dbms_shared_pool, or set cursor_sharing.

Calling this procedure to purge a cursor requires the (parent) cursor address concatenated with its hash value separated with comma, without any space (but spaces after the hash value are tolerated). The second argument is any letter other than 'p', 'q', 'r', 't' or their uppercase (they represent package/procedure/function, sequence, trigger, type, respectively, according to "Oracle Database PL/SQL Packages and Types Reference"). People like to use 'c' for easy reading ('c' for 'cursor'). The optional third argument is either 1 (by default) for heap 0, 64 for heap 6, or 65 for both (a cursor only has these two heaps; a PL/SQL object has other heaps). Because you pass the parent cursor address to the procedure, it's understandable that all child cursors are purged; there's no way to purge a specific child cursor, which, if Oracle were to allow us, might be useful in eliminating a specific bad execution plan. Additionally, due to bug 14127231, obsolete child cursors in 11gR2 may not be purged as of 11.2.0.3.0.

Of course the shared pool contains not only cursors, but also various other objects. For example, in an 11.2.0.3 database,

```
SQL> select namespace,type,count(*),sum(sharable_mem) from v$db_object_cache group by namespace, type order by 3;
```

NAMESPACE	TYPE	COUNT(*)	SUM(SHARABLE_MEM)
...			
BODY	TYPE BODY	10	166544
TABLE/PROCEDURE	SCHEDULER JOB	11	47536
SCHEMA	NONE	20	0
TABLE/PROCEDURE	FUNCTION	23	128504
TABLE/PROCEDURE	LIBRARY	24	14208
TRIGGER	TRIGGER	25	214808
TABLE/PROCEDURE	SEQUENCE	28	56976
TABLE/PROCEDURE	PROCEDURE	62	1563168
MULTI-VERSION OBJECT FOR TABLE	MULTI-VERSIONED OBJECT	102	891632
MULTI-VERSION OBJECT FOR INDEX	MULTI-VERSIONED OBJECT	107	930640
TEMPORARY TABLE	TABLE	107	0
BODY	PACKAGE BODY	111	10270096
TEMPORARY INDEX	INDEX	124	0
TABLE/PROCEDURE	SYNONYM	147	285392
TABLE/PROCEDURE	CURSOR	152	0
INDEX	INDEX	152	459928
TABLE/PROCEDURE	PACKAGE	359	4965352
TABLE/PROCEDURE	TYPE	362	1428352
TABLE/PROCEDURE	VIEW	493	1589408
TABLE/PROCEDURE	TABLE	1171	3346024
SQL AREA BUILD	CURSOR	2020	0
SQL AREA STATS	CURSOR STATS	10440	48692032
SQL AREA	CURSOR	21697	357631253

where namespace is the name of the type of objects while they're in the library cache, and type is (supposedly) the type as in the data dictionary, which often differs from the type name in memory. As you can see, although 'SQL AREA' as well as its STATS dominate the total count and also the summed memory, other library cache object types are not negligible. If there's a need to purge those other objects, either flush the whole shared pool, or if the object type is package/procedure/function, type (including type body), trigger, or sequence, dbms_shared_pool.purge can do the job fine. Just pass <owner>.<name> as the first argument, 'p' or 't' or 'r' or 'q' respectively as the second, to the purge procedure. For example, to purge this trigger from library cache,

exec dbms_shared_pool.purge('MDSYS.SDO_ST_SYN_CREATE', 'r', 65)

Only those four flags (second argument) are directly documented. The undocumented ones are Java related, 'jc' (java class), 'jr' (java resource), 'js' (java source), 'jd' (java data). Because of these additional flags, we can accurately state that for a cursor to be purged, the flag must be any character string other than 'p', 't', 'r', 'q', 'jc', 'jr', 'js', 'jd' or their uppercase.

Obviously there're a large number of object types that can't be purged with this purge procedure. Without extensive testing, I find that flag 'q', supposedly only for sequence, can be used to purge a TABLE or VIEW object, in spite of the statement in documentation "Currently, TABLE and VIEW objects may not be purged". Here's a test in 11.2.0.3:

```
SQL> select owner, name, namespace, sharable_mem, status from v$db_object_cache where type = 'TABLE' and name =
'WRI$_OPTSTAT_HISTGRM_HISTORY';
OWNER      NAME                           NAMESPACE       SHARABLE_MEM STATUS
---------- ------------------------------ --------------- ------------ -------------------
SYS        WRI$_OPTSTAT_HISTGRM_HISTORY   TABLE/PROCEDURE         4736 VALID

SQL> exec sys.dbms_shared_pool.purge('SYS.WRI$_OPTSTAT_HISTGRM_HISTORY', 'q')  <-- 'q' for sequence
PL/SQL procedure successfully completed.
SQL> select owner, name, namespace, sharable_mem, status from v$db_object_cache where type = 'TABLE' and name =
'WRI$_OPTSTAT_HISTGRM_HISTORY';
OWNER      NAME                           NAMESPACE       SHARABLE_MEM STATUS
---------- ------------------------------ --------------- ------------ -------------------
SYS        WRI$_OPTSTAT_HISTGRM_HISTORY   TABLE/PROCEDURE            0 UNKOWN  <-- 0 memory, unknown status

SQL> select owner, object_type, namespace from dba_objects where object_name = 'WRI$_OPTSTAT_HISTGRM_HISTORY';
OWNER      OBJECT_TYPE          NAMESPACE
---------- ------------------- ----------
SYS        TABLE                        1  <-- 1 means TABLE/PROCEDURE/TYPE according to dcore.bsq for obj$
```

Now let's take a look at the third form of the purge procedure, new in 11.2.0.2 and above.

```
PROCEDURE PURGE
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 HASH                           VARCHAR2                IN
 NAMESPACE                      NUMBER                  IN
 HEAPS                          NUMBER                  IN
```

Note the namespace argument, as a number. For example, an index type object can be purged with namespace number 4:

```
SQL> select owner, name, namespace, type, sharable_mem, kept, hash_value from v$db_object_cache where full_hash_value
= '41954f10923bf6eca09a193f7804402f';
OWNER       NAME                    NAMESPACE TYPE        SHARABLE_MEM KEP HASH_VALUE
----------- ---------------------- --------- ----------- ------------ --- ----------
SYS         WRH$_MVPARAMETER_PK   INDEX     INDEX             13872 NO  2013544495

SQL> exec sys.dbms_shared_pool.purge('2013544495', 4, 65)
BEGIN sys.dbms_shared_pool.purge('2013544495', 4, 65); END;
*
ERROR at line 1:
ORA-06570: shared pool object does not exist, cannot be pinned/purged
ORA-06512: at "SYS.DBMS_SHARED_POOL", line 126
ORA-06512: at line 1

SQL> exec sys.dbms_shared_pool.purge('41954f10923bf6eca09a193f7804402f', 4, 65)
PL/SQL procedure successfully completed.
```

The purge reduced sharable_mem to 0.

As we can see, the hash value must be full_hash_value of v$db_object_cache, a column new in 11.2.0.3 but exists as kglnahsv of x$kglob regardless of Oracle version. The namespace column is "a number indicating the library cache namespace in which the object is to be searched", but that statement is not accompanied with name-value pairs matching up number with namespace. Using Fuyuncat's unwrapper tool, we can see that this number in this new form of dbms_shared_pool.purge is passed to the second argument, context, of dbms_utility.name_resolve, which is documented. But according to that documentation, 4 is supposed to be "Java Source" and yet works for an index object; on the contrary, 9 for "index" does not work. I suspect the documentation for context of dbms_utility.name_resolve is wrong or severely outdated. Consistent with our test, however, is the column kglstidn (perhaps kernel generic library cache statistics ID number or identifier) of x$kglst in 11gR2 or simply indx in older versions (see the definition of v$librarycache in v$fixed_view_definition).

```
SQL> select kglsttyp, kglstdsc, kglstidn from x$kglst order by 3, 1;
KGLSTTYP   KGLSTDSC                 KGLSTIDN
---------- ------------------------ --------
NAMESPACE  SQL AREA                        0
```

```
TYPE        CURSOR                          0
NAMESPACE   TABLE/PROCEDURE                 1
TYPE        INDEX                           1
NAMESPACE   BODY                            2
TYPE        TABLE                           2
...
```

Since each number has a pair, one for namespace one for type, we can format the output better as follows:

```
SQL> select a.kglstidn, a.kglstdsc as_namespace, b.kglstdsc as_type from
  2  (select kglstdsc, kglstidn from x$kglst where kglsttyp = 'NAMESPACE') a,
  3  (select kglstdsc, kglstidn from x$kglst where kglsttyp = 'TYPE') b
  4  where a.kglstidn = b.kglstidn order by 1;


KGLSTIDN AS_NAMESPACE                 AS_TYPE
-------- ---------------------------- ---------------
       0 SQL AREA                     CURSOR
       1 TABLE/PROCEDURE              INDEX
       2 BODY                         TABLE
       3 TRIGGER                      CLUSTER
       4 INDEX                        VIEW
       5 CLUSTER                      SYNONYM
       6 KGL TESTING                  SEQUENCE
       7 PIPE                         PROCEDURE
       8 LOB                          FUNCTION
       9 DIRECTORY                    PACKAGE
      10 QUEUE                        NON-EXISTENT
      11 REPLICATION OBJECT GROUP     PACKAGE BODY
      12 REPLICATION PROPAGATOR       TRIGGER
      13 JAVA SOURCE                  TYPE
      14 JAVA RESOURCE                TYPE BODY
      15 REPLICATED TABLE OBJECT      OBJECT
      16 REPLICATION INTERNAL PACKAGE USER
      17 CONTEXT POLICY               DBLINK
      18 PUB SUB INTERNAL INFORMATION PIPE
```

```
      19 SUMMARY                    TABLE PARTITION
      20 DIMENSION                  INDEX PARTITION
...
```

So we can see the number 4 as namespace is for index; 9 rather is for directory objects. And there's no good relationship between library cache object namespace and data dictionary object type.

This new form of the purge procedure can purge objects of more namespaces; for example, passing 1 as namespace can purge a table object, which obviates the need to use the first form passing the awkward 'q' flag for table. In addition, we can also purge cursors using namespace 0 representing 'SQL AREA' or a cursor.

```
SQL> select owner, namespace, type, sharable_mem, kept, full_hash_value from v$db_object_cache where name = 'select
count(*) from testpurge';
OWNER      NAMESPACE TYPE        SHARABLE_MEM KEP FULL_HASH_VALUE
---------- --------- ----------- ------------ --- --------------------------------
           SQL AREA  CURSOR             12560 NO  5bf0f1ff0fdcedfd460c8d1f422fea98
           SQL AREA  CURSOR              4704 NO  5bf0f1ff0fdcedfd460c8d1f422fea98

SQL> exec sys.dbms_shared_pool.purge('5bf0f1ff0fdcedfd460c8d1f422fea98', 0, 65)  <-- namespace 0
PL/SQL procedure successfully completed.
SQL> select owner, namespace, type, sharable_mem, kept, full_hash_value from v$db_object_cache where name = 'select
count(*) from testpurge';
OWNER      NAMESPACE TYPE        SHARABLE_MEM KEP FULL_HASH_VALUE
---------- --------- ----------- ------------ --- --------------------------------
           SQL AREA  CURSOR                0 NO  5bf0f1ff0fdcedfd460c8d1f422fea98  <-- sharable_mem becomes 0
           SQL AREA  CURSOR             4704 NO  5bf0f1ff0fdcedfd460c8d1f422fea98
```

Note that the second row shown above represents the parent cursor. The only way to purge a parent cursor is flushing the shared pool. It doesn't take much memory anyway, except in case of bugs such as Bug 10082277.

Now you may ask, Is there any practical use in purging non-cursor objects? I tried to find the pattern as to when a SQL trace will generate recursive, data dictionary, queries. Unfortunately, no definitive pattern is found; regardless purging the cursor or the table object as shown above, a recursive SQL may or may not appear in the SQL trace, although immediately after a previous execution, another same execution will not trigger recursive SQL's and flushing the shared pool will of course guarantee all necessary recursive SQLs to be re-run. Additionally, purging these objects does not remove the entry in v$rowcache_parent (*select * from v$rowcache_parent where utl_raw.cast_to_varchar2(key) like '%THE_PURGED_TABLE%'*). On the other hand, as we have seen, the bulk of library cache is taken by SQL areas and their stats. Thus, purging objects other than cursors remains a curious academic exercise, at least for now.

Summary

Flushing the whole shared pool or purging out certain shared pool objects is needed in certain circumstances. The alter system flush shared_pool command has not gone through any change over the years. Invalidating library cache objects through a DDL slightly changes its behavior in Oracle 11*g*, leaving invalid objects in the shared pool, and therefore no longer serves the purpose of literally purging the objects. Dbms_shared_pool.purge, the most precisely targeting tool, works nicely on cursors using the traditional form. But the new forms, although potentially more powerful, are poorly documented. Hopefully Oracle will explain in full how they work and what they can be used for beyond anything obvious.

Acknowledgement

* My former coworker Du Shenglin first brought to my attention that v$sql* views in 11*g* retain the cursor after a DDL.

* Fuyuncat's excellent unwrapper tool helps find other flags the purge procedure may take.

黄勇的个人简介

黄勇

现在在美国德克萨斯州休斯顿城 M.D.安德森癌症中心工作，是该中心的一位 DBA。在加入 M.D.安德森之前，他曾在斯伦贝谢公司（全球最大的油田技术服务公司）、Unocal 石油、全美互惠保险公司、艺电（EA）和 eBay 担任 Oracle DBA 或咨询师。他在 ITPUB 上十分活跃，在个人博客上也撰写了大量技术文章。想了解更多关于他的资料，请查看：http://yong321.freeshell.org/computer.html