Shared Pool Management

Shared pool is used to cache different types of data such as textual and executable forms of PL/SQL blocks and SQL statements, dictionary cache data, and other data. If you use shared pool effectively you can reduce resource consumption in at least four ways

1. Parse overhead is avoided if the SQL statement is already in the shared pool. This saves CPU resources on the host and elapsed time for the end user.
2. Latching resource usage is significantly reduced, which results in greater scalability.
3. Shared pool memory requirements are reduced, because all applications use the same pool of SQL statements and dictionary resources.
4. I/O resources are saved, because dictionary elements that are in the shared pool do not require disk access.

Main components of shared pool are **library cache** (executable forms of SQL cursors, PL/SQL programs, and Java classes.) and the **dictionary cache** (usernames, segment information, profile data, tablespace information, and sequence numbers.).The library cache stores the executable (parsed or compiled) form of recently referenced SQL and PL/SQL code. The dictionary cache stores data referenced from the data dictionary. This caches are managed by LRU algorithm to “age out” memory structures that have not been reused over time. Allocation of memory from the shared pool is performed in chunks. This allows large objects (over 5k) to be loaded into the cache without requiring a single contiguous area, hence reducing the possibility of running out of enough contiguous memory due to fragmentation. Starting with 9i The Shared Pool divide its shared memory areas into subpools. Each subpool will have Free List Buckets (containing pointers to memory chunks within the subpool) and , memory structure entries, and LRU list. This architecture is designed to to increase the throughput of shared pool in that now each subpool is protected by a Pool child latch. This means there is no longer contention in the Shared Pool for a single latch as in earlier versions.

Infrequently, Java, PL/SQL, or SQL cursors may make allocations out of the shared pool that are larger than 5k then Oracle must search for and free enough memory to satisfy this request. This operation could conceivably hold the latch resource for detectable periods of time, causing minor disruption to other concurrent attempts at memory allocation. To allow these allocations to occur most efficiently, Oracle segregates a small amount of the shared pool. This memory is used if the shared pool does not have enough space. The segregated area of the shared pool is called the **reserved pool** which is also divided into subpools. Smaller objects will not fragment the reserved list, helping to ensure the reserved list will have large contiguous chunks of memory. Once the memory allocated from the reserved list is freed, it returns to the reserved list.

By using automatic shared memory management (ASMM) option available with 10G, which is activated by setting SGA_TARGET parameter with a value greater than 0 and STATISTICS_LEVEL to TYPICAL or ALL, shared pool size is started to be managed by Oracle, under the limits of SGA_TARGET and other SGA components.

After these explanations lets start to explain **how to manage shared pool with ASMM**.

1- Using Shared Pool Effectively

avoid hard parsing by

- using bind variables instead of literal values in your queries The script below can be used to find sqls which use literals

```
SELECT substr(sql_text,1,40) “SQL”, count(*) , sum(executions) “TotExecs”;
  FROM v$sqlarea
  WHERE executions < 5
  GROUP BY substr(sql_text,1,40)
  HAVING count(*) > 30
  ORDER BY 2;
```

- Standardizing naming conventions for bind variables and spacing conventions for SQL statements and PL/SQL blocks.

- Because they are stored, Consider using stored procedures whenever possible
- Avoiding users from change the optimization approach and goal for their individual sessions.
- Reducing the number of entries in the dictionary cache by explicitly qualifying the segment owner, rather than using public synonyms or try to connect to the database through a single user ID, rather than individual user IDs because Reducing the number of distinct userIDs also reduces the load on the dictionary cache.SELECT employee_id FROM **hr.**employees WHERE department_id = :dept_id;
- Using PL/SQL packages when your system has thousands of users, each with individual user sign-on and public synonyms because a package is executed as the owner, rather than the caller, which reduces the dictionary cache load considerably.
- Avoid performing DDLs in peak hours because ddl operations invalidate the dependent SQLs and cause hard parsing when the statement called again.
- Cache the sequence numbers by using CACHE keyword of CREATE/ALTER SEQUENCE clause to reduce the frequency of dictionary cache locks,
- Try to avoid closing of rapidly executed cursors
- Check for hash values which maps different literals. The query below should return no rows otherwise there is possibility for a bug

```
SELECT hash_value, count(*)
  FROM v$sqlarea
  GROUP BY hash_value
  HAVING count(*) > 5;
```

3- Identify which sqls are using lots of shared memory

```
SELECT substr(sql_text,1,20) "Stmt", count(*),
 sum(sharable_mem) "Mem",
 sum(users_opening) "Open",
 sum(executions) "Exec"
 FROM v$sql
 GROUP BY substr(sql_text,1,20)
 HAVING sum(sharable_mem) > 1426063 –%10 of Shared Pool Size;
```

Stmt	COUNT(*)	Mem	Open	Exec
/* OracleOEM */ SEL	18	1445971	2	54

This should show if there are similar literal statements, or multiple versions of a statements which account for a large portion of the memory in the shared pool.

4- Identify which allocations causing shared pool to be aged out

```
spool ageout.txt
SELECT *FROM x$ksmlru
where  ksmlrnum>0;
spool off
```

This select returns no more than 10 rows and then erases the contents of the X$KSMLRU table so be sure to SPOOL the output. The X$KSMLRU table shows which memory allocations have caused the MOST memory chunks to be thrown out of the shared pool since it was last queried. This is sometimes useful to help identify sessions or statements which are continually causing space to be requested.

5- Why are there multiple child cursors.

V$SQL_SHARED_CURSOR explains why a particular child cursor is not shared with existing child cursors. Each column identifies a specific reason why the cursor cannot be shared.

```
SELECT SA.SQL_TEXT, SA.VERSION_COUNT,SS.*
FROM V$SQLAREA SA, V$SQL_SHARED_CURSOR SS
WHERE SA.ADDRESS=SS.ADDRESS
AND SA.VERSION_COUNT > 5
ORDER BY SA.VERSION_COUNT ;
```

6- Monitor Shared Pool sizing operations

You can see the shrinking and growing operations from V$SGA_RESIZE_OPS dynamic view and you can guess

why there is need for this operations by focusing the sql at the sizing operation times.

```
select to_char(end_time, 'dd-Mon-yyyy hh24:mi') end, oper_type, initial_size,
  target_size, final_size from V$SGA_RESIZE_OPS
  where component='shared pool
  order by end;
```

END	OPER_TYPE	INITIAL_SIZE	TARGET_SIZE	FINAL_SIZE
12-Sep-2007 19:05	STATIC	0	134217728	134217728
12-Sep-2007 22:01	SHRINK	134217728	130023424	130023424
13-Sep-2007 11:35	SHRINK	130023424	125829120	125829120
13-Sep-2007 11:36	SHRINK	125829120	121634816	121634816
13-Sep-2007 22:08	GROW	121634816	125829120	125829120
13-Sep-2007 22:09	GROW	125829120	130023424	130023424
13-Sep-2007 22:10	GROW	130023424	134217728	134217728
13-Sep-2007 22:12	GROW	134217728	138412032	138412032
14-Sep-2007 09:49	GROW	138412032	142606336	142606336
14-Sep-2007 16:13	GROW	142606336	146800640	146800640

7- Minimum Size of Shared Pool

Current size of the shared pool;

```
select bytes
from v$sgainfo
where name='Shared Pool Size';

BYTES
-------------
138412032

138412032
```

You can use the sizig advices from the view v$shared_pool_advice. This view displays information about estimated parse time in the shared pool for different pool sizes and the sizes range from %10 to %200 of current shared pool size. This can give you idea for sizing SGA and obliquely shared pool by the help of ASMM.

select * from V$SHARED_POOL_ADVICE;

Suggested minimum Shared Pool Size:

```
set numwidth 20
column cr_shared_pool_size format 999,999,999,999
column sum_obj_size format 999,999,999,999
column sum_sql_size format 999,999,999,999
column sum_user_size format 999,999,999,999
column min_shared_pool format 999,999,999,999
select cr_shared_pool_size, sum_obj_size, sum_sql_size, sum_user_size,
       (sum_obj_size + sum_sql_size+sum_user_size)* 1.3 min_shared_pool
  from (select sum(sharable_mem) sum_obj_size from v$db_object_cache where type<> 'CURSOR'),
       (select sum(sharable_mem) sum_sql_size from v$sqlarea),
       (select sum(250*users_opening) sum_user_size from v$sqlarea),
       (select to_Number(b.ksppstvl) cr_shared_pool_size from x$ksppi a, x$ksppcv b, x$ksppsv c
  where a.indx = b.indx and a.indx = c.indx
    and a.ksppinm ='__shared_pool_size' );
```

CR_SHARED_POOL_SIZE	SUM_OBJ_SIZE	SUM_SQL_SIZE	SUM_USER_SIZE	MIN_SHARED_POOL
146800640	9520659	25660770	11750	45751132,7

You should set the suggested minimum shared pool size to avoid shrinking operation of ASMM

```
alter system set shared_pool_size=73M;
```

8- How much free memory in SGA is available for shared pool and how to interpret the free memory

First of all find the free memory in shared pool. If you have free memory you should relax but if you don't have go to the step below

```
SELECT * FROM V$SGASTAT
  WHERE NAME = 'FREE MEMORY'
  AND POOL = 'SHARED POOL';no rows selected
```

The **X$KSMSP** view shows the breakdown of memory in the SGA. You can run this query to build trend information on memory usage in the SGA. Remember, the 'free' class in this query is not specific to the Shared Pool, but is across the SGA. **Dont use the script below when db is under load. Check out Jonathan Lewis's experiences on this view from here**

```
SELECT KSMCHCLS CLASS, COUNT(KSMCHCLS) NUM, SUM(KSMCHSIZ) SIZ,
  To_char( ((SUM(KSMCHSIZ)/COUNT(KSMCHCLS)/1024)),'999,999.00')||'k' "AVG SIZE"
  FROM X$KSMSP GROUP BY KSMCHCLS;
```

CLASS	NUM	SIZ	AVG SIZE
freeabl	19010	34519404	1.77k
recr	23581	24967956	1.03k
R-freea	68	1632	.02k
perm	22	39801268	1,766.75k
R-free	34	7238192	207.90k
free	2389	36075980	14.75k

Watch for trends using these guidelines:

a) if 'free' memory is low (less than 5mb or so) you may need to increase the shared_pool_size and shared_pool_reserved_size. You should expect 'free' memory to increase and decrease over time. Seeing trends where 'free' memory decreases consistently is not necessarily a problem, but seeing consistent spikes up and down could be a problem.
b) if 'freeable' or 'perm' memory continually grows then it is possible you are seeing a memory bug.
c) if 'freeabl' and 'recr' memory classes are always huge, this indicates that you have a lot of cursor info stored that is not releasing.
d) if 'free' memory is huge but you are still getting 4031 errors, the problem is likely reloads and invalids in the library cache causing fragmentation.
!!!!!!!!!! Note says that this query can hang database on HP platforms

To see the free memory chunks detailed use the script below

```
select KSMCHIDX "SubPool", 'sga heap('||KSMCHIDX||',0)'sga_heap,ksmchcom ChunkComment,
        decode(round(ksmchsiz/1000),0,'0-1K', 1,'1-2K', 2,'2-3K',3,'3-4K', 4,'4-5K',5,'5-6k',6,'6-7k',7,'7-8k',8,'8-9k', 9,'9-10k','> 10K') "size",
       count(*),ksmchcls Status, sum(ksmchsiz) Bytes
  from x$ksmsp
  where KSMCHCOM = 'free memory'
  group by ksmchidx, ksmchcls,
  'sga heap('||KSMCHIDX||',0)',ksmchcom, ksmchcls,decode(round(ksmchsiz/1000),0,'0-1K', 1,'1-2K', 2,'2-3K', 3,'3-4K',4,'4-5K',5,'5-6k',6, '6-7k',7,'7-8k',8,'8-9k', 9,'9-10k','> 10K');
```

SubPool	SGA_HEAP	CHUNKCOMMENT	size	COUNT(*)	STATUS	BYTES
1	sga heap(1,0)	free memory	> 10K	34	R-free	7238192
1	sga heap(1,0)	free memory	3-4K	2	free	6284
1	sga heap(1,0)	free memory	> 10K	241	free	35707400
1	sga heap(1,0)	free memory	8-9k	1	free	7712

1	sga heap(1,0)	free memory	2-3K	4	free	6752
1	sga heap(1,0)	free memory	0-1K	2090	free	133288
1	sga heap(1,0)	free memory	9-10k	21	free	188676
1	sga heap(1,0)	free memory	1-2K	30	free	25868

If you see lack of large chunks it is possible that you can face with ORA-04031 in near future.
9- Is library_cache or dictionary_cache utilization satisfactory ?

The statistics below is based since the start of the instance. You should take interval statistics to interpret these values for performance issues .

- **Library Cache Stats**

```
SELECT NAMESPACE, PINS, PINHITS, RELOADS, INVALIDATIONS
  FROM V$LIBRARYCACHE
  ORDER BY NAMESPACE;
```

NAMESPACE	PINS	PINHITS	RELOADS	INVALIDATIONS
BODY	72782	72582	49	0
CLUSTER	1175	1161	3	0
INDEX	2800	2023	42	0
JAVA DATA	0	0	0	0
JAVA RESOURCE	0	0	0	0
JAVA SOURCE	0	0	0	0
OBJECT	0	0	0	0
PIPE	0	0	0	0
SQL AREA	563349	541678	2069	342
TABLE/PROCEDURE	175850	165318	2005	0
TRIGGER	6923	6802	34	0

High invalidations indicates that there is parsing problem with the namespace and high reloads indicates that there is a sizing problem which causes aging out.

- **Library cache hit ratio;**

```
 SELECT SUM(PINHITS)/SUM(PINS) FROM V$LIBRARYCACHE;

SUM(PINHITS)/SUM(PINS)
----------------------
95558088
```

low hit ratio is an indication of a sizing or caching problem

- **Dictionary cache stats**

```
SELECT PARAMETER, SUM(GETS) , SUM(GETMISSES), 100*SUM(GETS - GETMISSES) / SUM(GETS) PCT_SUCC_GETS, SUM(MODIFICATIONS) UPDATES
  FROM V$ROWCACHE
  WHERE GETS > 0
  GROUP BY PARAMETER;
```

PARAMETER	SUM(GETS)	SUM(GETMISSES)	PCT_SUCC_GETS	UPDATES
dc_constraints	99	35	64,6464646	99
dc_tablespaces	90104	14	99,9844624	0
dc_tablespace_quotas	13	3	76,9230769	0
dc_awr_control	1351	2	99,8519615	121

dc_object_grants	867	174	79,9307958	0
dc_histogram_data	52053	6181	88,1255643	3047
dc_rollback_segments	55098	92	99,8330248	263
dc_sequences	100	27	73	100
dc_usernames	6632	33	99,5024125	0
dc_segments	23404	2466	89,4633396	331
dc_objects	37434	3776	89,9129134	358
dc_histogram_defs	65987	16796	74,5465016	3280
dc_table_scns	8	8	0	0
dc_users	171638	105	99,9388247	0
outstanding_alerts	1674	58	96,5352449	66
dc_files	80	10	87,5	0
dc_object_ids	134005	2646	98,0254468	123
dc_global_oids	52337	185	99,6465216	0
dc_profiles	1962	4	99,7961264	0

High updates with low pct_succ_gets can be a clue of performance problems when accessing that dictionary object. For frequently accessed dictionary caches, the ratio of total `GETMISSES` to total `GETS` should be less than 10% or 15%, depending on the application. If this ratio is higher and every previous control is OK then you should consider to increase the shared pool size

- **Dictionary cache hit ratio;**

```
SELECT (SUM(GETS - GETMISSES - FIXED)) / SUM(GETS) “ROW CACHE” FROM V$ROWCACHE;

ROW CACHE
----------
9516921886454345524
```

Low hit ratio is an indication of a sizing problem.

10- Are there any objects candidate for library cache pinning ?

Having objects pinned will reduce fragmentation and changes of encountering the ORA-04031 error. Objects causing a large number of other objects been flushed out from the shared pool are candidates to be pinned into the shared pool using dbms_shared_pool.keep procedure. You can check the x$ksmlru fixed table to see the candidates. This table keeps track of the objects and the corresponding number of objects flushed out of the shared pool to allocate space for the load. These objects are stored and flushed out based on the Least Recently Used (LRU) algorithm. Because this is a fixed table, once you query the table, Oracle will automatically reset the table so first insert the contents to temporary table like below,

```
CREATE TABLE LRU_TMP AS SELECT * FROM X$KSMLRU;
```

and on regular intervals issue

```
INSERT INTO LRU_TMP SELECT * FROM X$KSMLRU;
```

Use the LRU_TMP table for analysis. You can use a query below to see more information on candidate code in the library cache.

```
SELECT USERNAME, KSMLRCOM, KSMLRHON, KSMLRNUM, KSMLRSIZ, SQL_TEXT
FROM V$SQLAREA A, LRU_TMP K, V$SESSION S
WHERE KSMLRSIZ > 3000
AND A.ADDRESS=S.SQL_ADDRESS AND A.HASH_VALUE = S.SQL_HASH_VALUE
AND SADDR=KSMLRSES;
```

You can see the candidates to pin from the query below

```
COL STORED_OBJECT FORMAT A40;
COL SQ_EXECUTIONS FORMAT 999,999;
SELECT /*+ ORDERED USE_HASH(D) USE_HASH(C) */ O.KGLNAOWN||'.'||O.KGLNAOBJ STORED_OBJECT, SUM(C.KGLHDEXC) SQL_EXECUTIONS
FROM SYS.X$KGLOB O, SYS.X$KGLRD D, SYS.X$KGLCURSOR C
WHERE O.INST_ID = USERENV('INSTANCE') AND
        D.INST_ID = USERENV('INSTANCE') AND
        C.INST_ID = USERENV('INSTANCE') AND
        O.KGLOBTYP IN (7, 8, 9, 11, 12) AND
        D.KGLHDCDR = O.KGLHDADR AND
        C.KGLHDPAR = D.KGLRDHDL
GROUP BY O.KGLNAOWN, O.KGLNAOBJ
HAVING SUM(C.KGLHDEXC) > 0
ORDER BY 2 DESC;
```

You should pin objects you find immediatelly after the each restart of instance. You can pin the object by DBMS_SHARED_POOL package like below

```
EXECUTE DBMS_SHARED_POOL.KEEP(OWNER.TRIGGER, 'R')
```

11- Is my Reserved Area sized properly?

An ORA-04031 error referencing a large failed requests indicates the Reserved Area is too fragmented.

```
col free_space for 999,999,999,999 head "TOTAL FREE"
col avg_free_size for 999,999,999,999 head "AVERAGE|CHUNK SIZE
col free_count for 999,999,999,999 head "COUNT"
col request_misses for 999,999,999,999 head "REQUEST|MISSES
col request_failures for 999,999,999,999 head "REQUEST|FAILURES"
col max_free_size for 999,999,999,999 head "LARGEST CHUNK"
select free_space, avg_free_size, free_count, max_free_size, request_misses, request_failures
from v$shared_pool_reserved;
```

TOTAL FREE	AVERAGE CHUNK SIZE	COUNT	LARGEST CHUNK	REQUEST MISSES	REQUEST FAILURES
7,238,192	212,888	34	212,888	0	0

The reserved pool is small when:

REQUEST_FAILURES > 0 (and increasing)

The DBA should Increase shared_pool_reserved_size and shared_pool_size together.

It is possible that too much memory has been allocated to the reserved list.
If:

REQUEST_MISS = 0 or not increasing
FREE_MEMORY = > 50% of shared_pool_reserved_size minimum

The DBA should Decrease shared_pool_reserved_size

You should also use hidden and unsupported parameter "_shared_pool_reserved_pct" to control reserved pool. This parameter controls the allocated percentage of shared pool for reserved pool. By default it is %5 of the shared pool and if you use ASMM for memory management you can set this value higher like 10 to allocate reserved pool dynamically. When you set the parameter you will see the shared_pool_reserved_size parameter will be adjusted to the new setting.

The parameter can not be modified when instance is started. You can use the query below to see the current value

```
select a.ksppinm "Parameter", b.ksppstvl "Session Value", c.ksppstvl "Instance Value"
from sys.x$ksppi a, sys.x$ksppcv b, sys.x$ksppsv c
where a.indx = b.indx and a.indx = c.indx
  and a.ksppinm = '_shared_pool_reserved_pct';
```

Parameter	Session Value	Instance Value

_shared_pool_reserved_pct	10	10

12-Is there any fragmentation in shared pool?

The primary problem that occurs is that free memory in the shared pool becomes fragmented into small pieces over time. Any attempt to allocate a large piece of memory in the shared pool will cause large amount of objects in the library cache to be flushed out and may result in an ORA-04031 out of shared memory error. But how to understand the fragmentation ?

- Occurrence of ORA-04031 error. Before this error signalled, memory is freed from unnecessary objects and merged. This error only occurs when there is still not a large enough contiguous piece of free memory after this cleaning process. There may be very large amounts of total free memory in the shared pool, but just not enough contiguous memory.
- Using X$KSMLRU internal fixed table. We told about this view before about its usage for tracking age out operations, it also can be used to identify what is causing the large allocations. KSMLRSIZ column of this table shows the amount of contiguous memory being allocated. Values over around 5K start to be a problem, values over 10K are a serious problem, and values over 20K are very serious problems. Anything less then 5K should not be a problem. Again be careful to save spool the result when you query this table

```
select * from x$ksmlru where ksmlrsiz > 5000;
```

After finding the result you should do the followings to correct fragmentation

- Keep object by pinning them as we discussed above
- Use bind variables as we discussed before
- Eliminate large anonymous PL/SQL block. Large anonymous PL/SQL blocks should be turned into small anonymous PL/SQL blocks that call packaged functions. The packages should be 'kept' in memory. To view candidates

```
select sql_text from v$sqlarea
where command_type=47 — command type for anonymous block
and length(sql_text) > 500;
```

Fallacies about solving shared pool fragmentation

- **Free memory in shared pool prevents fragmentation.** This is not true because Free memory is more properly thought of as 'wasted memory'. You would rather see this value be low than very high. In fact, a high value of free memory is sometimes a symptom that a lot of objects have been aged out of the shared pool and therefore the system is experiencing fragmentation problems.
- **Flushing shared pool frequently solves fragmentation and improves performance.** This is also incorrect because Executing this statement causes a big spike in performance and does nothing to improve fragmentation. You lost your cached cursors when you flush and they will hard parsed next time with high CPU consumption.

13- Using related database parameters

- CURSOR_SHARING: Setting this parameter to smilar can solve your hard parse problems caused by using literals but can have side effects mostly on DSS environments and systems which uses stored outlines.
- CURSOR_SPACE_FOR_TIME: This parameter specifies whether a cursor can be deallocated from the library cache to make room for a new SQL statement. CURSOR_SPACE_FOR_TIME has the following values meanings:
  - If CURSOR_SPACE_FOR_TIME is set to false (the default), then a cursor can be deallocated from the library cache regardless of whether application cursors associated with its SQL statement are open. In this case, Oracle must verify that the cursor containing the SQL statement is in the library cache.
  - If CURSOR_SPACE_FOR_TIME is set to true, then a cursor can be deallocated only when all application cursors associated with its statement are closed. In this case, Oracle need not verify that a

cursor is in the cache, because it cannot be deallocated while an application cursor associated with it is open.

You **must** be sure that the shared pool is large enough for the work load otherwise performance will be badly affected and ORA-4031 eventually signalled.

- OPEN_CURSORS: This parameter sets the upper bound for the number of cursor that a session can have open and if you size it correctly, cached cursors can be stay opened and won't have to be closed to let new cursor open
- PROCESSES / SESSIONS: You can review the high water mark for Sessions and Processes in the V$RESOURCE_LIMIT view. If the hard-coded values for these parameters are much higher than the high water mark information, consider decreasing the parameter settings to free up some memory in the Shared Pool for other uses.
- SESSION_CACHED_CURSORS: When a cursor is closed, Oracle divorces all association between the session and the library cache state. If no other session has the same cursor opened, the library cache object and its heaps are unpinned and available for an LRU operation. The parameter SESSION_CACHED_CURSORS controls the number of cursors "soft" closed, much like the cached PL/SQL cursors. Oracle checks the library cache to determine whether more than three parse requests have been issued on a given statement. If so, then Oracle assumes that the session cursor associated with the statement should be cached and moves the cursor into the session cursor cache. Subsequent requests to parse that SQL statement by the same session then find the cursor in the session cursor cache.To determine whether the session cursor cache is sufficiently large for your instance, you can examine the session statistic `session cursor cache hits` in the `V$SYSSTAT` view. This statistic counts the number of times a parse call found a cursor in the session cursor cache. If this statistic is a relatively low percentage of the total parse call count for the session, then consider setting `SESSION_CACHED_CURSORS` to a larger value. Steve Adams also wrote usefully queries to find the usage and the maximum cacheable cursors.session_cursor_cache.sql

This was a long article and if you see anything wrong or suspicious please feel free to comment for correction

All of the queries are tested on Oracle 10.2.0.3 for Windows

Code Depot of The Queries (All scripts are taken from metalink notes and official documentation)
References :

Oracle® Database Performance Tuning Guide 10*g* Release 2 (10.2)

Metalink Notes

Note:396940.1 Troubleshooting and Diagnosing ORA-4031 Error

Note:146599.1 Diagnosing and Resolving Error ORA-04031

Note:61623.1 Resolving Shared Pool Fragmentation In Oracle7

Note:62143.1 Understanding and Tuning the Shared Pool

Note:1012047.6 How To Pin Objects in Your Shared Pool

Note:274496.1 ora-7445 and ora-4031 in 9.2.0.5 and 10g if SESSION_CACHED_CURSORS is used

www.ixora.com Oracle Advanced Performance Tuning Scripts

Mailing list threads from Oracle-l

http://www.freelists.org/archives/oracle-l/08-2007/msg00975.html