Purpose and Scope:
This article is intended to provide a inside view and working of Shared Pool and its solution since Oracle 7 version thru Oracle 10g Release 2.

Introduction to Shared Pool
It is fundamentally to serve the metadata cache. Shared pool helps execute SQL and PLSQL. Efficient metadata caching required to support creation of objects and to be a repository of many sharable components for efficient running of RDBMS engine and PL/SQL engine

Contents of Shared Pool

SQL and PLSQL Objects

SQL Area

V$SQLAREA table contains all the data of this sql area and also provides a list of statistics on shared SQL area and contains one row per SQL string. It provides statistics on SQL statements that are in memory, parsed, and ready for execution.

PL/SQL MPCODE

The "compiled" form a PL/SQL is called PCODE, which stands for pseudo code, and is interpreted at runtime by the PL/SQL engine. That is, it is not really compiled; it is just pre-compiled. There is also a machine dependent form of pseudo code used in some cases called MPCODE. (Source Steve Adams – 1999 – Questions and Answers)

PL/SQL DIANA

Source: Steve Adams - 1999 Q & A

IDL stands for Interface Definition Language. It is an intermediate language in which the structure of database tables and the logic of PL/SQL program units can be consistently represented as attributed trees. Oracle uses the DIANA IDL, which comes from compilers for the Ada programming language. DIANA stands for Descriptive Intermediate Attributed Notation for Ada. Anyway, this is one of four tables in the data dictionary used to store the DIANA for PL/SQL program units, and the database objects that they reference.

To convert from a human-readable source language (such as PL/SQL) to a machine-readable language (such as m-code) that is efficiently executed. In Oracle, compiling converts PL/SQL into two internal forms, m-code and DIANA, to assist with both efficient execution and dependency management.

Library cache:

It contains the objects, grants on those objects, dependencies, sequences and synonyms. It also contains the complex metadata that helps to compile the shared SQL and PLSQL
Please query v$librarycache and under namespace column all the following are listed.

BODY
CLUSTER
INDEX
JAVA DATA
JAVA RESOURCE
JAVA SOURCE
OBJECT
PIPE
SQL AREA
TABLE/PROCEDURE
TRIGGER

Row cache:

Please query ‘parameter’ from v$rowcache and the following is listed.

dc_awr_control
dc_constraints
dc_database_links
dc_files
dc_free_extents
dc_global_oids
dc_hintsets
dc_histogram_data
dc_histogram_defs
dc_object_grants
dc_object_ids
dc_objects
dc_outlines
dc_partition_scns
dc_profiles
dc_qmc_cache_entries
dc_qmc_ldap_cache_entries
dc_rollback_segments
dc_segments
dc_sequences
dc_table_scns
dc_tablespace_quotas
dc_tablespaces
dc_used_extents
dc_usernames
dc_users
global database name
kqlsubheap_object
outstanding_alerts
qmtmrcin_cache_entries
qmtmrcip_cache_entries
qmtmrciq_cache_entries
qmtmrcin_cache_entries
qmtmrcn_cache_entries
qmtmrcpq_cache_entries
rule_fast_operators
rule_info
rule_or_piece
Active Session History (ASH) buffers

There are new background processes introduced by Oracle in Oracle 10g in R1 and R2. They are

The Memory Monitor Light (MMNL) process is a new process in 10g which works with the Automatic Workload Repository new feature (AWR) to write out full statistics buffers to disk as needed.

The memory monitor (MMON) process was introduced in 10g and is associated with the Automatic Workload Repository new features used for automatic problem detection and self-tuning. MMON writes out the required statistics for AWR on a scheduled basis.

MMON background slave (m000) processes.

MMAN - SGA Background Process: The Automatic Shared Memory Management feature uses a new background process named Memory Manager (MMAN). MMAN serves as the SGA Memory Broker and coordinates the sizing of the memory components. The SGA Memory Broker keeps track of the sizes of the components and pending resize operations

ASH is a circular buffer and is an integral part of shared pool. Oracle Metalink Doc ID Note: 243132.1

Size of ASH Circular Buffer = Max [Min [ #CPUs * 2 MB, 5% of Shared Pool Size, 30MB ], 1MB ]

An ACTIVE session means

01. Present inside a user call
02. Not a recursive session
03. Not waiting for the 'IDLE' wait-event
04. If it is a background process, not waiting for its usual timer-event
05. If it is a parallel slave, not waiting for the PX_IDLE wait event.

So will not see any info if a process is waiting for "SQL*Net message from client".

Get that SQL with this SQL

```sql
select SESSION_ID,
NAME,
P1,
P2,
P3,
WAIT_TIME,
CURRENT_OBJ#, 
CURRENT_FILE#,
CURRENT_BLOCK#
from v$active_session_history ash,
v$event_name enm
where ash.event#=enm.event#
and SESSION_ID=&SID and SAMPLE_TIME>=(sysdate-&minute/(24*60));
```

Oracle has provided a utility under $ORACLE_HOME/rdbms/demo (Location may change), by which you can upload the ASH trace dump to a database table and do the analysis.
And many more stuff is stored in the Shared Pool of every Oracle Instance.

**Memory Allocation and Release**

The SHARED_POOL has at the highest level has 2 kinds of structures. The first structure is PERMANENT structure which is not alterable as has been spawned by Oracle Instance Startup and its stay in the SHARED_POOL is not negotiable.

The second structure is re-creatable and hence is negotiable. The negotiating mechanism is Least Recently Used (LRU) and thus the contents can be aged out and re-loadable. But every re-load has its price to pay.

01. When new objects are referenced they need to be brought into memory and they need memory allocation
02. So re-creatable objects are aged out and pushed out of memory
03. Objects are made up of chunks of memory and when they are created the process checks for contiguous required space
04. The chunks of memory are in 1 and 4 K
05. If the available free memory is not sufficient to create a contiguous required chunk, Oracle throws up error ORA-04031

**SQL and PL/SQL objects and cursor usage and parsing**

There are two kinds of cursors.

01. One is explicit cursor – created by PL/SQL explicitly in the declarative section and then managed and closed in the executable/run section of the PL/SQL object. For queries that return more than one row, one can explicitly declare a cursor to process the rows individually.
02. PL/SQL implicitly declares a cursor for all SQL data manipulation statements, including queries that return only one row.
03. All the SELECT statements issued are managed by the implicit cursors.

When a cursor is closed, the cursor information is moved into session’s closed cursor cache in User Global Area (UGA). The Shared pool maintains the handles in hashed chains. SESSION_CACHED_CURSORS is the parameter that controls the number. The default value for this in Oracle 10.2.0.2.0 is 0 (zero) and in the later versions, it was said that the same is set to 50.

**How it works?**

When a cursor is opened, the session process hashes the SQL statement and performs a hash lookup in the closed cursor cache in the session memory (UGA) and if found the same is moved to open cursors and then no parsing is required.

If the cursor is not found in the session, the hash value is used to search the hash chains in the shared pool for the cursor handle. This search is registered as hard parse.

If the cursor handle is found and the cursor has not aged out, the cursor is executed. This is a soft parse.

If some part of the cursor has aged out of the shared pool
Or
If the cursor does not exist in the shared pool, then the cursor is reconstructed.

This is called hard parse. The cursor reconstruction requires a lookup of the metadata for the dependent objects such as tables, indexes, extents and sequences.
If the metadata for these objects does not already cached in the shared pool, recursive SQL is generated to fetch the information from the data dictionary.

**How the cache in the shared pool is protected from being over written?**

The following listing is to identify the latches listed from v$latchname.

<table>
<thead>
<tr>
<th>LATCH#</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>213</td>
<td>shared pool</td>
</tr>
<tr>
<td>214</td>
<td>library cache</td>
</tr>
<tr>
<td>215</td>
<td>library cache lock</td>
</tr>
<tr>
<td>216</td>
<td>library cache pin</td>
</tr>
<tr>
<td></td>
<td>library cache pin allocation</td>
</tr>
<tr>
<td>217</td>
<td>library cache lock</td>
</tr>
<tr>
<td>218</td>
<td>allocation</td>
</tr>
<tr>
<td>219</td>
<td>library cache load lock</td>
</tr>
<tr>
<td>220</td>
<td>library cache hash chains</td>
</tr>
<tr>
<td>240</td>
<td>shared pool simulator</td>
</tr>
<tr>
<td>241</td>
<td>shared pool sim alloc</td>
</tr>
</tbody>
</table>

Query v$system_event to get the events

```sql
select event
from v$system_event
where (event like '%shared%' or event like '%library%') /

EVENT

latch: shared pool
latch: library cache
library cache pin
library cache lock
library cache load lock
```

**latch: shared pool**

protects memory allocations in the shared pool for various categories.

**latch: library cache**

locates the matching SQL in the shared pool.

**Library cache pin**

A session is in need of a row cache cursor from the LRU and it could pick any dictionary cursor to map which it has previously used.

Please note that from 8.1.7 to 10.2 versions are impacted by this latch. Metalink Doc Id Note: 287059.1 reads as under:

A session is in need of a row cache cursor from the LRU and it could pick any dictionary cursor to map which it has previously used. Now consider a second process (or session) concurrently doing some different operation parsing a different row cache cursor but needing to close one from its LRU could end up deadlocked with a similar stack segment.
The X mode when closing the cursor is needed when updating the statistics of the cursor generated while having some tracing enabled like (but not limited to) TIMED_STATISTICS=TRUE, or SQL_TRACE.

The deadlock and the pile up will not happen until there is a request in X mode.

Some of the reasons to close a row cache cursor are

- Lots of Recursive operations
- Shared Pool too small
- runs out of cache

There is currently a limitation to detect deadlock at the row cache level.

```
SELECT
  s.sid,
  kglpnmmod "Mode",
  kglpnreq "Req",
  SPID "OS Process"
FROM  v$session_wait w,
       x$kglpn p,
       v$session s ,
       v$process o
WHERE
  p.kglpnuse=s.saddr
AND   kglpnhdl=w.p1raw
and w.event like '%library cache pin%' 
and s.paddr=o.addr
/
```

To free up the instance kill the blocking session at the OS level.

To prevent it from happening do any, some or all of the following :

- set TIMED_STATISTICS=FALSE
- set _row_cache_cursors=20 or more (10 default)
- don’t do any kind of tracing

**Latch cache lock**

There may be shadow processes holding locks on database objects that need to be released. The session that is locking an object and is not releasing the lock and is blocking the users run the catblock.sql that is stored in $ORACLE_HOME/rdbms/admin and identify the blocking sessions. There are multiple reasons for this

01. the application design which demands serialization is trying to do this concurrently
02. the ITL slots on the object may not have been set properly thru INITRANS parameter
03. the chosen Oracle block size is too small and can not allow the concurrency as wanted/demanded by the application
04. the referential integrity constraints (foreign key constraints) are not indexed
05. many more user induced application related issues like not committing and closing sessions
06. and bugs

**Library cache load lock**
This lock is waiting for a reload by another session. These locks are possible in various scenarios.

01. if the table is being re-organized online
02. if the move command is used in your DDL
03. if the application is badly built
04. and many more
05. bugs too

What to happen if Shared Pool is fragmented and latch contention persists?

ORA-04031

Identify the latch contentions:

```sql
select name,gets,misses,sleeps
from v$latch
where name like 'library%';
```

To get the OS process id:

```sql
select a.name,pid from v$latch a, V$latchholder b
where a.addr=b.laddr
and a.name = 'library cache%';
```

Identify if sessions are waiting for the latches

By selecting from v$session_wait during a slowdown period you can usually determine very
accurately whether you have a problem with latching and which latch is causing the problem. If
you see a large number (more then 3 or 4) of processes waiting for the library cache or library
cache pin latch, then there may be a problem. Run the following query to determine this:

```sql
select count(*) number_of_waiters
from v$session_wait w,
     v$latch l
where  w.wait_time = 0
and w.event     = 'latch free'
and w.p2        = l.latch#
and l.name      like 'library%';
```

It is also very useful to just select from v$session_wait to determine what else is causing a
slowdown:

```sql
select *
from v$session_wait
where event != 'client message'
and event not like '%NET%'
and wait_time = 0
;
```

Issue this SQL to the average used memory

```sql
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;

From the output

a) if free memory (SIZ) is low (less than 5mb or so) you may need to increase the shared_pool_size and shared_pool_reserved_size.
b) if perm continually grows then it is possible you are seeing system memory leak.
c) if freeabl and recr are always huge, this indicates that you have lots of cursor info stored that is not releasing.
d) if free is huge but you are still getting 4031 errors, (you can correlate that with the reloads and invalids causing fragmentation)

Oracle 10g R1 onwards a trace file is generated for ORA-04031 to user dump destinations. This error should not appear in any application logs, the alert log file or any trace file. ORA-04031 is not an internal error like ORA-00600 or ORA-07445. The immediate solution is ADD memory to shared pool.

Library Cache Reloads

The library cache should avoid constant reload of the re-creatable objects. The shared pool should be big enough to keep re-creatable objects.

If the number of invalidations is more, it means that the reload of the re-creatable objects is more or the DDL changes are more in the application management and this is a high over head.

DBA is to investigate and fix the problem. If invalidations are about 20% of loads, please consider that as a problem and resolve that.

This SQL helps:

```sql
select namespace, pins, pins-pinhits loads, reloads, invalidations, round(100*(reloads-invalidations)/(pins-pinhits)) "%RELOADS"
from v$librarycache
where pins > 0
order by namespace;
```

In the reloads % is greater than 20, then increase the shared pool by at least 30% of the existing size.

Row Cache Misses

Let us consider the following query.

```sql
select parameter, sum(gets) gets, sum(getmisses) getmisses, sum(usage) usage
from v$rowcache
where getmisses > 0
group by parameter
order by parameter;
```

Assuming that there are no hard parses, each row cache miss will result in a single row fetch from the data dictionary.
Latch Contentions

If reloads have not caused any ORA-04031 errors, there many not be any latch contentions related to the shared pool size but some other unrelated issue is there lurking in the dark.

The latch contention can be caused by

01. high hard parse rates
02. exceptionally high soft parses and execution rates
03. excessive shared pool monitoring activity
04. non-standard initialization parameter (under score) parameter

The latch contention can be reduced by setting cursor_space_for_time may reduce latch contention for SQL execution but increase latch contention due to reloads.

Oversized Pool

Select 
  count (1) num_sql,
  sum (decode(executions, 1,1,0)) num_1_use_sql,
  sum(sharable_mem)/1024/1024 mb_sql_mem,
  sum(decode(executions, 1,sharable_mem, 0)) / 1024/1024 mb_1_use_sql_mem
from v$sqlarea
where sharable_mem > 0;

This tells how much of memory is used for one time executions. Now it is to be investigated to identify if the SQL are really ad-hoc or because of the use of literals, they have become ad-hoc like queries. If they are same SQL with changing literals, please suggest the developers/ or users to use bind variables.

Parameters that can help:

SESSION_CASHED_CURSORS helps avoid the overhead of a session locating the SQL statement in the shared pool when it next attempts to re-parse the same SQL.

CURSOR_SPACE_FOR_TIME is not frequently used. It is good for OLTP systems. But this is necessary in Oracle 10gR2 to reduce contention as a new serialization mechanism which is smaller and faster than latches is available. This has the additional benefits of not locking down the cursor memory in the shared pool and not requiring the cursor to remain open. (Refer to the white paper of Oracle on internals of shared pool memory structures).

Query V$DB_OBJECT_CACHE to identify the objects in the shared pool with the predicate kept='YES'.

Mutexes and their increased role in Oracle 10gR2

_USE_KKS_MUTEX=TRUE is to be set to enable the use of mutexes.

Data Dictionary Support for Mutex

V$MUTEX_SLEEP
V$MUTEX_SLEEP_HISTORY
GV$MUTEX_SLEEP
GV$MUTEX_SLEEP_HISTORY
X$TABLES

X$MUTEX_SLEEP_HISTORY
X$MUTEX_SLEEP

References:

01. Understanding and Tuning the Shared Pool Doc ID: Note:62143.1
02. Metalink Doc Id Note: 287059.1
03. Oracle Metalink Doc ID Note: 243132.1
04. Oracle Concepts 10g R2 document
05. Oracle presentation on Understanding Shared Pool Structures
06. Steve Adams – Oracle Internals on his
07. Tom Kyte – Many of his docs
08. Many Friends and many websites – I can not count but I acknowledge
09. Oracle Trace Files and
10. Hundreds of Metalink many other docs