PARSING IN ORACLE

Frits Hoogland
This is a deep dive into the internal working of the Oracle database.

This session provides nothing that has to do with normal day-to-day operational database tasks.

The target audience is Oracle database specialists with a profound understanding of the inner working of the Oracle database and C program internals.

This session can also be used to simply get more familiar with the database inner workings.

If this is not what you are looking for, this session is not for you.
Why?

Parsing in most cases does not cause any issues.

In some (exceptional) cases parsing can cause issues.

The intention of this presentation is to detail the parsing process, so issues can be investigated and analysed when they occur.
This is research based on profiling Oracle database execution.

There is no guarantee this will be the same in any other version, and can change at any time without notice, like with a newer PSU applied.

Although experience learns that the basic principle almost always remains the same (but can change of course!).

Investigations in this presentation are done using Oracle database version 18.4 on Oracle Linux 7.5.
WHAT IS PARSING?

SQL tuning guide (docs.oracle.com):

SQL processing
The stages of parsing, optimization, row source generation, and execution of a SQL statement.

parse call
A call to Oracle to prepare a SQL statement for execution. The call includes syntactically checking the SQL statement, optimizing it, and then building or locating an executable form of that statement.

parsing
The stage of SQL processing that involves separating the pieces of a SQL statement into a data structure that can be processed by other routines.
A hard parse occurs when the SQL statement to be executed is either not in the shared pool, or it is in the shared pool but it cannot be shared. A soft parse occurs when a session attempts to execute a SQL statement, and the statement is already in the shared pool, and it can be used.

hard parse
The steps performed by the database to build a new executable version of application code. The database must perform a hard parse instead of a soft parse if the parsed representation of a submitted statement does not exist in the shared SQL area.

soft parse
Any parse that is not a hard parse. If a submitted SQL statement is the same as a reusable SQL statement in the shared pool, then Oracle Database reuses the existing code. This reuse of code is also called a library cache hit.
WHAT IS PARSING?

SQL tuning guide (docs.oracle.com):

library cache
An area of memory in the shared pool. This cache includes the shared SQL areas, private SQL areas (in a shared server configuration), PL/SQL procedures and packages, and control structures such as locks and library cache handles.

shared SQL area
An area in the shared pool that contains the parse tree and execution plan for a SQL statement. Only one shared SQL area exists for a unique statement. The shared SQL area is sometimes referred to as the cursor cache.
WHAT IS PARSING?

The following diagram is included in the concepts guide and the SQL Tuning guide:

This seems to suggest syntax and semantic checking takes place during a soft parse, after which the shared pool is checked.

All these statements seem confusing and sometimes conflicting to me.
WHAT IS PARSING?

Let’s look how this really works!
LIBRARY CACHE

Everything that is parsed and subsequently executed, is put in the library cache first.

This includes invalid SQL.

Please mind UI’s like sqlplus and sqlci validate SQL, and might not send it to the database at all:

SQL> xxxxx;
SP2-0042: unknown command "xxxxx" - rest of line ignored.
LIBRARY CACHE

But java allows you to submit anything to the database you like!

$ $ORACLE_HOME/jdk/bin/javac -classpath $ORACLE_HOME/jdbc/lib/ojdbc8.jar OracleSample.java
$ $ORACLE_HOME/jdk/bin/java -classpath "$ORACLE_HOME/jdbc/lib/ojdbc8.jar:." OracleSample

Exception in thread "main" java.sql.SQLSyntaxErrorException: ORA-00900: invalid SQL statement
...
Caused by: Error : 900, Position : 0, Sql = XXXXXXXX, OriginalSql = XXXXXXXX, Error Msg = ORA-00900: invalid SQL statement
...
import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class OracleSample {

    public static final String DBURL = "jdbc:oracle:thin:@localhost:1521:o184";
    public static final String DBUSER = "system";
    public static final String DBPASS = "oracle";

    public static void main(String[] args) throws SQLException {
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection con = DriverManager.getConnection(DBURL, DBUSER, DBPASS);
        Statement statement = con.createStatement();
        ResultSet rs = statement.executeQuery("XXXXXXXX");
        if (rs.next()) {
            Date currentDate = rs.getDate(1); // get first column returned
            System.out.println("Current Date from Oracle is : "+currentDate);
        }
        rs.close();
        statement.close();
        con.close();
    }
}
LIBRARY CACHE

Wait a minute…

SQL> select sql_text from v$sqlarea where sql_text like 'XXX%';
no rows selected

It seems V$SQLAREA only shows valid cursors, use X$KGLCURSOR:

SQL> select kglhdadr, kglhdpar, kglnaobj from x$kglcursor
    where kglnaobj like 'XXXX%';

<table>
<thead>
<tr>
<th>KGLHDADR</th>
<th>KGLHDPAR</th>
<th>KGLENABJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000791E9A30</td>
<td>000000007C6BF4E8</td>
<td>XXXXXXXX</td>
</tr>
<tr>
<td>000000007C6BF4E8</td>
<td>000000007C6BF4E8</td>
<td>XXXXXXXX</td>
</tr>
</tbody>
</table>
Hash buckets
Hash buckets (parent) handles

Namespace=TABLE/PROCEDURE(01) Type=TABLE(02)

Namespace=SQL AREA BUILD(82) Type=CURSOR(00)

Namespace=SQL AREA STATS(75) Type=CURSOR STATS(102)

Namespace=SQL AREA(00) Type=CURSOR(00)
LIBRARY CACHE

The library cache buckets and objects can be seen for yourself using:

```
alter session set events 'immediate trace name library_cache level 16';
```

Level is a bitmap:

- 1  statistics
- 2  hash table bucket statistics
- 4  library cache contents
- 8  library cache contents including dependencies
- 16 library cache contents including dependencies and children
- 32 heap contents
LIBRARY CACHE

This is how that looks like:

Bucket: `#=80464 Mutex=0x61b3c1d8(863288426496, 199, 0, 6)
LibraryHandle: Address=0x7e875d90 Hash=b6213a50 LockMode=N PinMode=0 LoadLockMode=0 Status=VALD
ObjectName: Name=select count(*) from test
  FullHashValue=e9e1b41ddd730d39bc6e4cafb6213a50 Namespace=SQL AREA(00)
ContainerId=0 ContainerUid=0 Identifier=3055630928 OwnerIdn=86
Statistics: InvalidationCount=0 ExecutionCount=2 LoadCount=2 ActiveLocks=1 TotalLockCount=3
  TotalPinCount=1
Counters: BrokenCount=1 RevocablePointer=1 KeepDependency=1 Version=0
HandleInUse=1 HandleReferenceCount=0
Concurrency: DependencyMutex=0x7e875e40(0, 25, 0, 0) Mutex=0x7e875ee0(201, 259, 0, 6)
  Flags=RON/PIN/TIM/PN0/DBN/[10012841] Flags2=[0000]
WaitersLists:
  Lock=0x7e875e20[0x7e875e20,0x7e875e20]
  Pin=0x7e875e00[0x7e875e00,0x7e875e00]
  LoadLock=0x7e875e78[0x7e875e78,0x7e875e78]
Timestamp: Current=03-19-2019 14:21:54
LIBRARY CACHE

Let’s look at what happens when SQL is parsed, alias a new SQL cursor is introduced?

The next slides tell a simplified version of what happens.

Can I see this for myself?
Yes, but not using a convenient Oracle trace like the library cache dump.

It can be seen using Intel PinTools, using the tool ‘debugtrace’.
At least, that is what I used.
Important!

The SQL statement that is executed is: ‘select count(*) from test’.

You’ll see why this matters later on!
PARSING STEPS 1/3

• Generate hash value from SQL text (kgscComputeHash).
• Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
• Lookup parent handle via hash bucket; create handle.
• KGL lock parent handle.
• KGL pin parent handle, creates heap 0, unpin.
• Search compatible child in child list (list in heap 0, currently no children exist).
• Create a $BUILD$.last 16 characters of SQL text hash value independent object (lookup, create, KGL lock).
• Create child handle, KGL lock child handle.
• Obtain a CU enqueue (ksqgelctx).
• KGL pin child handle, creates heap 0.
• Create SQL AREA STATS independent object (lookup, create, KGL lock, KGL pin, allocate, unpin, unlock)
• Unpin child handle.
• Child cursor initialization (kksParseChildCursor).
• Create/allocate heap 6 (qksshBeginCompile).
PARSING STEPS 2/3

• Syntax check (prscmd).
• Semantic checks (opiSem).
  • Validate all dependent objects (lookup via KGL hash bucket, KGL get, KGL pin), add as reference in the dependent object, add to dependency table in the child heap 0.
  • Validate authorization (KGL pin of SQL cursor that looks up authorization via objauth$).
  • Start creating child cursor object information for execution (heap 6).
• Access path analysis (apadrv).
  • Create OPTIMIZER DIRECTIVE OWNER independent object (lookup, allocate, KGL lock, KGL pin, allocate, unpin, unlock).
  • Costing of access paths and transformations.
• Create query structures (in heap 6; qkadrv, qkadrv2).
  • Explain plan generation.
  • Explain plan annotations.
• Audit (not sure what is audited and where to information is written; audpre).
• End code generation (kksEndOfCompile).
PARSING STEPS 3/3

• All KGL locked and KGL pinned objects (dependencies) are unpinned and unlocked (kksPopCompileCall).
• The CU enqueue is freed (ksqdel).
• The child cursor is added to the child list in the parent’s heap 0.
• The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.
LIBRARY CACHE

Yes, that’s a lot of work.

But parsing it again should take advantage of all this work, right?

And that is what is considered a “soft parse” according to documentation.
LIBRARY CACHE

1st parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket; create handle.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
- Create a $BUILD$.last 16 characters of SQL text hash value independent object.
- Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (kksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
  - Validate all dependent objects, add to dependency table.
  - Validate authorization.
  - Start creating child cursor execution information (heap 6).
- Access path analysis (apadrv).
  - Create OPTIMIZER DIRECTIVE OWNER independent object.
  - Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
  - Explain plan generation.
  - Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (kksEndOfCompile).
- All KGL locked and KGL pinned objects (cursor and dependencies) are unpinned and unlocked (kksPopCompileCall).
- The CU enqueue is freed (ksqdel).
- The child cursor is added to the child list in the parent’s heap 0.
- The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.

2nd/3rd parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).

4th parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Validate authorization (3rd time uses cached data).
LIBRARY CACHE

What is parsing and what is hard parsing exactly?

Luckily, V$ACTIVE_SESSION_HISTORY has a few columns that can help us:

SQL> describe v$active_session_history

...  
IN_CONNECTION_MGMT VARCHAR2 (1)
IN_PARSE VARCHAR2 (1)
IN_HARD_PARSE VARCHAR2 (1)
IN_SQL_EXECUTION VARCHAR2 (1)
IN_PLSQL_EXECUTION VARCHAR2 (1)
IN_PLSQL_RPC VARCHAR2 (1)
IN_PLSQL_COMPILATION VARCHAR2 (1)
IN_JAVA_EXECUTION VARCHAR2 (1)
IN_BIND VARCHAR2 (1)
IN_CURSOR_CLOSE VARCHAR2 (1)
LIBRARY CACHE

I created a little script that lists the flags in v$active_session_history:

```sql
select sample_time,
in_connection_mgmt,
in_parse,
in_hard_parse,
in_sql_execution,
in_bind,
in_cursor_close,
in_sequence_load
from v$active_session_history
where session_id=&1
   and sample_time > sysdate-0.5/60/24 --last 30 seconds
order by 1
/
```
And looked into a PinTools debugtrace trace, and made a list of functions that I thought were part of parsing, hard parsing and execution, and which are close but not part of it, and simply created a gdb script to break on these functions:

break kpoal8
break kpooprx
break opiosq0
break kksParseCursor
break kkspsc0
break kgscComputeHash
break kgscFindCursor
break ksqgelctx
break kksParseChildCursor
break qksshBeginCompile
...etc...
LIBRARY CACHE

1. Log on to oracle, execute “warmup” SQL, setup unique SQL and hold.
   $ sqlplus ts/ts
   ...
   SQL> select 1 from dual;
       1
       -------
           1
   SQL> select 8 from dual;

2. Log on to oracle as sysdba, obtain OS PID and SID.
   $ sqlplus / as sysdba
   ...
   SQL> @who
   
   SID_SERIAL  OSPID  USERNAME  C    SERVER       PROGRAM
   ----------  -------  --------  ----  ------------  ---------------
   14,64999    9647    TS      DEDICATED sqlplus@o184.local
   211,10621    9902    SYS     * DEDICATED sqlplus@o184.local
LIBRARY CACHE

3. Attach to the session with gdb, execute the breaks, and continue the session.

```
$ gdb -p 9647
...
(gdb) source parse.gdb
...
(gdb) c
Continuing.
```

4. The sysdba session will show nothing yet (ASH does not store idle waits)

```
SQL> @ash_state 14

no rows selected
```
LIBRARY CACHE

5. Execute the statement that will hard parse
SQL> select 8 from dual;
This will hang, and gdb will show it broke execution.
Breakpoint 1, 0x00000000002e25f20 in kpoal8 ()
(gdb)

6. Execute the ASH query in the sysdba session:
SQL> @ash_state 14
SAMPLE_TIME               IN_CON IN_PAR IN_HAR IN_SQL IN_BIN IN_CUR IN_SEQ
------------------------- ------ ------ ------ ------ ------ ------ ------
28-MAR-19 11.05.24.387 AM N      N      N      N      N      N      N
28-MAR-19 11.05.53.605 AM N      N      N      N      N      N      N
...

The bottom row shows the latest state. Not in parse, hard parse and execution.
LIBRARY CACHE

7. Continue until kksParseCursor:
Breakpoint 4, 0x0000000000337b000 in kksParseCursor ()
(gdb)

8. Execute the ASH query in the sysdba session:
SQL> @ash_state 14

<table>
<thead>
<tr>
<th>SAMPLE_TIME</th>
<th>IN_CON</th>
<th>IN_PAR</th>
<th>IN_HAR</th>
<th>IN_SQL</th>
<th>IN_BIN</th>
<th>IN_CUR</th>
<th>IN_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>28-MAR-19 11.51.42.536 AM N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>28-MAR-19 11.58.02.560 AM N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

So, with kksParseCursor Oracle considers to be in the state ‘in parse’, which means
the function that calls kksParseCursor (opiosq0) probably is where the parse phase is
entered.
### LIBRARY CACHE

#### 1st parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket; create handle.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
- Create a $BUILD$.last 16 characters of SQL text hash value independent object.
- Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (kksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
  - Validate all dependent objects, add to dependency table.
  - Validate authorization.
  - Start creating child cursor execution information (heap 6).
- Access path analysis (apadrv).
  - Create OPTIMIZER DIRECTIVE OWNER independent object.
  - Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
  - Explain plan generation.
  - Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (kksEndOfCompile).
- All KGL locked and KGL pinned objects (cursor and dependencies) are unpinned and unlocked (kksPopCompileCall).
- The CU enqueue is freed (ksqdel).
- The child cursor is added to the child list in the parent’s heap 0.
- The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.

#### 2nd/3rd parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).

#### 4th parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).
LIBRARY CACHE

1st parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket; create handle.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
- Create a $BUILD$.last 16 characters of SQL text hash value independent object.
- Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (kksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
  • Validate all dependent objects, add to dependency table.
  • Validate authorization.
  • Start creating child cursor execution information (heap 6).
- Access path analysis (apadr).
  • Create OPTIMIZER DIRECTIVE OWNER independent object.
  • Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
  • Explain plan generation.
  • Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (kksEndOfCompile).
- All KGL locked and KGL pinned objects (cursor and dependencies) are unpinned and unlocked (kksPopCompileCall).
- The CU enqueue is freed (ksqdel).
- The child cursor is added to the child list in the parent’s heap 0.
- The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.

2nd/3rd parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).

4th parse
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).
## LIBRARY CACHE

**1st parse**
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket; create handle.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
- Create a $BUILD$.last 16 characters of SQL text hash value independent object.
- Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (kksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
  - Validate all dependent objects, add to dependency table.
  - Validate authorization.
  - Start creating child cursor execution information (heap 6).
- Access path analysis (apadrv).
- Create OPTIMIZER DIRECTIVE OWNER independent object.
- Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
  - Explain plan generation.
  - Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (kksEndOfCompile).
- All KGL locked and KGL pinned objects (cursor and dependencies) are unpinned and unlocked (kksPopCompileCall).
- The CU enqueue is freed (ksqdel).
- The child cursor is added to the child list in the parent’s heap 0.
- The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.

**2nd/3rd parse**
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).

**4th parse**
- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Look up parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
- Validate authorization (3rd time uses cached data).
LIBRARY CACHE

1st parse

- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
- Create a $BUILD$.last 16 characters of SQL text hash value independent object.
- Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (kksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
- Access path analysis (apadrv).
- Create OPTIMIZER DIRECTIVE OWNER independent object.
- Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
- Explain plan generation.
- Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (kksEndOfCompile).

2nd/3rd parse

- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).

3rd/4th parse

- Validate authorization (3rd time uses cached data).

in parse

in hard parse

- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached_cursors; kgscFindCursor).
- Lookup parent handle via hash bucket.
- KGL lock parent handle.
- Search compatible child in child list (list in heap 0, compatible child is found).
Now that we are aware of the general principle, a word of warning. Things, sadly, are not as simple and straightforward as has been described…
LIBRARY CACHE

SQL> @ash_state 14

<table>
<thead>
<tr>
<th>SAMPLE_TIME</th>
<th>IN_CON</th>
<th>IN_PAR</th>
<th>IN_HAR</th>
<th>IN_SQL</th>
<th>IN_BIN</th>
<th>IN_CUR</th>
<th>IN_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>28-MAR-19 11.51.42.536 AM</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28-MAR-19 11.58.02.560 AM</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

So: this indicates being in hard parse AND in execution at the same time??

Let’s look at a backtrace (in gdb) to see what is being executed:
LIBRARY CACHE

#0 0x00000000012a46a0 in kksfbc ()
#1 0x000000000120f8d1 in opiexe ()
#2 0x000000000120f2db9 in opial10 ()
#3 0x00000000012106119 in opikpr ()
#4 0x000000000120f44fc in opiodr ()
#5 0x000000000121df7d in rpidrus ()
#6 0x00000000012424ab1 in skgmstack ()
#7 0x0000000001221e3e8 in rpidru ()
#8 0x0000000001221ce17 in rpiswu2 ()
#9 0x000000000122223ed in kprball ()
#10 0x0000000003642d9b in kqdoebf_new ()
#11 0x0000000003642425 in kqdoebf ()
#12 0x00000000012211c19 in kqrfReadFromDB ()
#13 0x0000000001220d6ba in kqrpre2 ()
#14 0x000000000122141a8 in kqrpre1 ()
#15 0x00000000031f3de7 in kqlhdlod ()
#16 0x00000000031e6538 in kqlCallback ()
#17 0x00000000031e614b in kqlldod ()
#18 0x00000000012458b93 in klglLoadOnLock ()
#19 0x000000000124573cd in klglkal ()
#20 0x00000000012452a05 in klgllock ()
#21 0x0000000001244cd2e in klglget ()
#22 0x0000000001244b0c8 in kglgob ()
#23 0x0000000004c25a11 in qcdlgb0 ()
#24 0x0000000004c24ff6 in qcdlgb ()
#25 0x0000000004a62aa6 in qcsfgob ()
#26 0x0000000004a61f69 in qcsprfro ()
#27 0x000000000121e38bc in qcsprfro_tree ()
#28 0x000000000121e38bc in qcsprfro_tree ()
#29 0x000000000121e38bc in qcsprfro_tree ()
#30 0x000000000121e38bc in qcsprfro_tree ()
#31 0x000000000121e38bc in qcsprfro_tree ()
#32 0x000000000121e38bc in qcsprfro_tree ()
#33 0x000000000121e38bc in qcsprfro_tree ()
#34 0x000000000121e38bc in qcsprfro_tree ()
#35 0x000000000121e38bc in qcsprfro_tree ()
#36 0x000000000121e38bc in qcsprfro_tree ()
#37 0x000000000121e38bc in qcsprfro_tree ()
#38 0x000000000121e38bc in qcsprfro_tree ()
#39 0x000000000121e38bc in qcsprfro_tree ()
#40 0x000000000121e38bc in qcsprfro_tree ()
#41 0x000000000121e38bc in qcsprfro_tree ()
#42 0x000000000121e38bc in qcsprfro_tree ()
#43 0x000000000121e38bc in qcsprfro_tree ()
#44 0x000000000121e38bc in qcsprfro_tree ()
#45 0x000000000121e38bc in qcsprfro_tree ()
#46 0x000000000121e38bc in qcsprfro_tree ()
#47 0x000000000121e38bc in qcsprfro_tree ()
#48 0x000000000121e38bc in qcsprfro_tree ()
#49 0x000000000121e38bc in qcsprfro_tree ()
#50 0x000000000121e38bc in qcsprfro_tree ()
#51 0x000000000121e38bc in qcsprfro_tree ()
#52 0x000000000121e38bc in qcsprfro_tree ()
#53 0x000000000121e38bc in qcsprfro_tree ()
#54 0x000000000121e38bc in qcsprfro_tree ()
#55 0x000000000121e38bc in qcsprfro_tree ()
Now that we are aware of the general principle, a word of warning. Things, sadly, are not as simple and straightforward as has been described…
LIBRARY CACHE

Based on the findings, I decided to perform the following actions:

1. Execute a SQL that introduces a new cursor.
   Stop execution of that session right AFTER parse, but BEFORE execution.
   This can be accomplished by using gdb and breaking on the function kpoal8Check.

2. I just established a child cursor is created as part of parsing.
   That must mean that at the stopped position I can query the execution plan, right?
## LIBRARY CACHE

SQL> @who sql

<table>
<thead>
<tr>
<th>SID_SERIAL</th>
<th>OSPID</th>
<th>USERNAME</th>
<th>C</th>
<th>EXTRA</th>
</tr>
</thead>
<tbody>
<tr>
<td>201,41789</td>
<td>18417</td>
<td>SYS</td>
<td>*</td>
<td>sql_id:dytmgjf2ca809,child#:0</td>
</tr>
<tr>
<td>595,45886</td>
<td>18753</td>
<td>TS</td>
<td></td>
<td>sql_id:43jhwsvkv1rt,child#:0</td>
</tr>
</tbody>
</table>

LIBRARY CACHE

SQL> @dplan
Enter value for sql_id: 43jhwswsvkv1rt
Enter value for child_no: 0

SQL_ID 43jhwswsvkv1rt, child number 0
-------------------------------------
select 8 from dual

-----------------------------------------------------------------
<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td>2 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>FAST DUAL</td>
<td></td>
<td>1</td>
<td>2 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
LIBRARY CACHE

Now the test is repeated with a different SQL.
## LIBRARY CACHE

SQL> @who sql

<table>
<thead>
<tr>
<th>SID_SERIAL</th>
<th>OSPID</th>
<th>USERNAME</th>
<th>C</th>
<th>EXTRA</th>
<th>sql_id</th>
<th>child#</th>
</tr>
</thead>
<tbody>
<tr>
<td>201,41789</td>
<td>18417</td>
<td>SYS</td>
<td>*</td>
<td></td>
<td>sql_id:dytmgjf2ca809,child#:0</td>
<td></td>
</tr>
<tr>
<td>595,45886</td>
<td>18753</td>
<td>TS</td>
<td></td>
<td></td>
<td>sql_id:4rkqppzd4wvh,child#:0</td>
<td></td>
</tr>
</tbody>
</table>
LIBRARY CACHE

SQL> @dplan
Enter value for sql_id: 4rkqpzhdv4whh
Enter value for child_no: 0

SQL_ID  4rkqpzhdv4whh, child number 0
-------------------------------------
select * from test where f1 = :T

NOTE: cannot fetch plan for SQL_ID: 4rkqpzhdv4whh, CHILD_NUMBER: 0
Please verify value of SQL_ID and CHILD_NUMBER;
It could also be that the plan is no longer in cursor cache (check v$sql_plan)
LIBRARY CACHE

How come I can’t pick up the explain plan right after parsing just before I execute??

Is something wrong? Let’s continue the sqlplus session in gdb:
Breakpoint 5, 0x0000000002e289b0 in kpoal8Check ()
(gdb) c
Continuing.

The sqlplus session peacefully finishes and shows nothing weird:
SQL> select * from test where f1 = :T;

D
-
X
LIBRARY CACHE

And now the plan is available!

SQL> @dplan
Enter value for sql_id: 4rkqpzhdv4whh
Enter value for child_no: 0

SQL_ID 4rkqpzhdv4whh, child number 0

select * from test where f1 = :T

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>Name</th>
<th>Rows</th>
<th>Bytes</th>
<th>Cost (%CPU)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td>3 (100)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>SORT AGGREGATE</td>
<td></td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* 2</td>
<td>TABLE ACCESS FULL</td>
<td>TEST</td>
<td>1</td>
<td>2</td>
<td>3 (0)</td>
<td>00:00:01</td>
</tr>
</tbody>
</table>
LIBRARY CACHE

Let’s break at the primary functions for the generation of the child cursor:

• prscmd (syntax check)
• opiSem (semantic check)
• apadrv (costing)
• qkadrv (plan generation)

And look at the backtrace.
This is prscmd:

(gdb) bt
#0 0x00000000002dc9a90 in prscmd ()
#1 0x00000000002dc5147 in prsdrv ()
#2 0x000000000040fd2fe in opiParse ()
#3 0x000000000040fcf19 in opiprs ()
#4 0x03364d1a in kksParseChildCursor ()
#5 0x00000000001221ce17 in rpiswu2 ()
#6 0x00000000033711b0 in kksLoadChild ()
#7 0x0012209f51 in kxsGetRuntimeLock ()
#8 0x00000000121a890f in kksfbc ()
#9 0x000000000337bc0e in kkspsc0 ()
#10 0x00000000337b07b in kksParseCursor ()
#11 0x0000000012354f9a in opiosq0 ()
#12 0x0000000002e288d4 in kpooprx ()
#13 0x0000000002e26266 in kpoal8 ()
#14 0x00000000120f44fc in opiadr ()
#15 0x00000000123be757 in ttcpip ()
#16 0x000000002741357 in opitsk ()
#17 0x000000002745f6d in opiino ()
#18 0x00000000120f44fc in opiadr ()
#19 0x00000000273cd93 in opidrv ()
#20 0x00000000334337f in sou2o ()
#21 0x00000000d6d107 in opimai_real ()
#22 0x0000000033501d7 in ssthrdmain ()
#23 0x00000000d6cf13 in main ()
This is opiSem:

(gdb) bt

#0 0x0000000000040f5ed0 in opiSem ()
#1 0x00000040f4edf in opiDeferredSem ()
#2 0x00000000040ef714 in opitca ()
#3 0x000003364926 in kksFullTypeCheck ()
#4 0x0000000001221ce17 in rpiswu2 ()
#5 0x0000000336c565 in kksSetBindType ()
#6 0x00000000121a7666 in kksfbc ()
#7 0x00000000120f8dal in opiexe ()
#8 0x00000000002e26893 in kpoal8 ()
#9 0x0000000000120f44fc in opiodr ()
#10 0x000000000123be757 in ttcpip ()
#11 0x0000000002741357 in opitsk ()
#12 0x0000000002745f6d in opiino ()
#13 0x00000000120f44fc in opiodr ()
#14 0x000000000273cd93 in opidrv ()
#15 0x00000000334337f in sou2o ()
#16 0x00000000d6d107 in opimai_real ()
#17 0x0000000033501d7 in ssthrdmain ()
#18 0x00000000d6cf13 in main ()
LIBRARY CACHE

This is apadrv:

(gdb) bt
#0 0x000000000036b1060 in apadrv ()
#1 0x000000000040efde0 in opitca ()
#2 0x000003364926 in kksFullTypeCheck ()
#3 0x0000000001221ce17 in rpiswu2 ()
#4 0x0000000336c565 in kksSetBindType ()
#5 0x000000000121a7666 in kksfbc ()
#6 0x000000000120f8dal in opiexe ()
#7 0x0000000002e26893 in kpoal8 ()
#8 0x000000000120f44fc in opiodr ()
#9 0x000000000123be757 in ttcpip ()
#10 0x0000000002741357 in opitsk ()
#11 0x0000000002745f6d in opiino ()
#12 0x000000000120f44fc in opiodr ()
#13 0x000000000273cd93 in opidrv ()
#14 0x000000000334337f in sou2o ()
#15 0x00000000000d6d107 in opimai_real ()
#16 0x00000000033501d7 in ssthrdmain ()
#17 0x00000000000d6cf13 in main ()
LIBRARY CACHE

This is how it looks like when you look at the flags in ASH:

SQL> @ash_state 14

<table>
<thead>
<tr>
<th>SAMPLE_TIME</th>
<th>IN_CON</th>
<th>IN_PAR</th>
<th>IN_HAR</th>
<th>IN_SQL</th>
<th>IN_BIN</th>
<th>IN_CUR</th>
<th>IN_SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>28-MAR-19 11.51.42.536 AM</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28-MAR-19 11.58.02.560 AM</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

This means that it’s showing being in parse, in hard parse and in execution.

There is no way to tell if a session was originally in parse or execution phase, or if it’s because of recursive SQL.

I do think the makers of ASH assumed a hard parse was always in the parse phase.
LIBRARY CACHE

I decided to create another test: break on opiSem, when it’s known to be deferred until execution time, and let the process sleep for 1 sec.

This way I can see how time gets accounted in the data dictionary (v$sess_time_model):

<table>
<thead>
<tr>
<th>TYPE</th>
<th>Name</th>
<th>diff</th>
<th>diff/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>13,STAT</td>
<td>parse count (total)</td>
<td>1</td>
<td>.037</td>
</tr>
<tr>
<td>13,STAT</td>
<td>parse count (hard)</td>
<td>1</td>
<td>.037</td>
</tr>
<tr>
<td></td>
<td>hard parse elapsed t</td>
<td>1.010</td>
<td>.037</td>
</tr>
<tr>
<td></td>
<td>repeated bind ela t</td>
<td>.000</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>parse time elapsed</td>
<td>1.010</td>
<td>.037</td>
</tr>
<tr>
<td></td>
<td>DB CPU</td>
<td>.004</td>
<td>.000</td>
</tr>
<tr>
<td></td>
<td>sql execute elapsed t</td>
<td>1.010</td>
<td>.037</td>
</tr>
<tr>
<td></td>
<td>DB time</td>
<td>1.010</td>
<td>.037</td>
</tr>
<tr>
<td></td>
<td>parse time elapsed</td>
<td>1</td>
<td>.000</td>
</tr>
</tbody>
</table>
LIBRARY CACHE

I wondered how this looks like in a 10046/sql trace; the accounting is done per trace occasion, so it can’t update parse time for this cursor once it has moved to execution.

This is what I found:

PARSING IN CURSOR #140643331798432 len=32 dep=0 uid=86 oct=3 lid= select 8 from test where f3 = :t
END OF STMT

PARSE #140643331798432:c=336,e=336,p=0,cr=0,cu=0,mis=1,r=0,dep=0, ... other stuff ...

EXEC #140643331798432:c=1116,e=1007793,p=0,cr=6,cu=0,mis=1,r=0,dep=0, ... other stuff ...

WAIT #140643331798432: nam='SQL*Net message to client' ela= 5 ... other stuff ...

FETCH #140643331798432:c=0,e=39,p=0,cr=6,cu=0,mis=0,r=0,dep=0,og= STAT #140643331798432 id=1 cnt=0 pid=0 pos=1 obj=32192 op='TABLE
LIBRARY CACHE

Library cache contention scenario’s.
MUTEXES, OBJECT CHANGE SERIALISATION

The library cache (and dictionary cache) objects are protected by mutexes.

Older documentation talks about library cache latches. These do not exist anymore.

Mutexes in the database are an Oracle implementation of spinlocks, like latches.

The aim for mutexes is to have a simpler spin lock construction created with and inside the object to protect, thus being much more granular than latches.

Despite being a simpler construction, mutexes can have more states than latches:
MUTEXES, OBJECT CHANGE SERIALISATION

(gdb) set $op=&kgxOpcodeName
(gdb) x/s *$op++
0x128e3f18: "NONE" 0
0x1475b514: "GET_SHRD" 1
0x146b5b80: "SHRD" 2
0x1475b520: "SHRD_EXAM" 3
0x1475b52c: "REL_SHRD" 4
0x1475b538: "GET_EXCL" 5
0x12d3f754: "EXCL" 6
0x1475b544: "REL_EXCL" 7
0x1475b550: "GET_INCR" 8
0x1475b55c: "INCR_EXAM" 9
0x1475b568: "GET_DECR" 10
0x1475b574: "DECR_EXAM" 11
0x1475b580: "RELEASED" 12
0x1475b58c: "EXCL_SHRD" 13
0x1475b598: "GET_EXAM" 14
0x1475b5a4: "EXAM" 15
0x1475b5ac: "GET_LONG_EXCL" 16
0x1475b5bc: "REL_LONG_EXCL" 17
0x1475b5c0: "LONG_EXCL" 18
Hash buckets

Namespace=TABLE/PROCEDURE(01) Type=TABLE(02)

Namespace=SQL AREA BUILD(82) Type=CURSOR(00)

Namespace=SQL AREA STATS(75) Type=CURSOR STATS(102)

Namespace=SQL AREA(00) Type=CURSOR(00)

Namespace=TABLE/PROCEDURE(01) Type=TABLE(02)

Namespace=SQL AREA BUILD(82) Type=CURSOR(00)

Namespace=SQL AREA STATS(75) Type=CURSOR STATS(102)

Namespace=SQL AREA(00) Type=CURSOR(00)

cursor: mutex *
cursor: pin *

library cache: mutex X / library cache: dependency mutex X

library cache: bucket mutex X / library cache: mutex X
MUTEXES, OBJECT CHANGE SERIALISATION

Most library cache data structure changes are protected by a mutex.

- Access to a hash bucket (usage counter)
- Creation, locking, unlocking, pinning and unpinning of parent and child handles.
- Any manipulation of the child hash table (X) or reading (S) it.
- Manipulation of reference and dependency tables.
- Manipulation of data blocks/heaps.

This is for every individual library cache object!

It is fundamentally important that a process can do its work while holding a mutex so it can release it as fast as possible. If not, it’s possible the mutex waits escalate instance wide. This is why CPU oversubscription can harm performance.
CURSOR CONCURRENCY WAIT EVENTS

During parsing alias cursor creation other sessions trying to use the same cursor should wait until it’s ready.

Outside of mutexes limiting access when state indicators and counters that are part of a cursor are in an in-flux state, there are other waits.
**CURSOR CONCURRENCY WAIT EVENTS**

- Generate hash value from SQL text (kgscComputeHash).
- Inspect session side cursor cache (session_cached.Cursors; kgscFindCursor).
- Lookup parent handle via hash bucket; create handle.
- KGL lock parent handle.
- KGL pin parent handle, creates heap 0, unpin.
- Search compatible child in child list (list in heap 0, currently no children exist).
  - Create a $BUILD$.last 16 characters of SQL text hash value independent object.
  - Create child handle, KGL lock child handle.
- Obtain a CU enqueue (ksqgelctx).
- KGL pin child handle, creates heap 0.
- Create SQL AREA STATS independent object.
- Unpin child handle.
- Child cursor initialization (ksParseChildCursor).
- Create/allocate heap 6 (qksshBeginCompile).
- Syntax check (prscmd).
- Semantic checks (opiSem).
  - Validate all dependent objects, add to dependency table.
  - Validate authorization.
  - Start creating child cursor execution information (heap 6).
- Access path analysis (apadrv).
  - Create OPTIMIZER DIRECTIVE OWNER independent object.
  - Costing.
- Create query structures (in heap 6; qkadrv, qkadrv2).
  - Explain plan generation.
  - Explain plan annotations.
- Audit (not sure what is audited and where to information is written; audpre).
- End code generation (ksEndOfCompile).
- All KGL locked and KGL pinned objects (cursor and dependencies) are unpinned and unlocked (kksPopCompileCall).
- The CU enqueue is freed (ksqdel).
- The child cursor is added to the child list in the parent’s heap 0.
- The $BUILD$.last 16 characters of SQL text hash value independent object is unlocked.

**no wait event / not blocking**

library cache lock

ksfbc child completion
cursor: pin S wait on X
SUMMARY

• Any database request parses before it can execute, unless it’s explicitly been programmed to parse once and execute multiple times.
• Parsing means organising an executable cursor.
• A hard parse means (re)creating the child cursor.
• Part of the parsing might be deferred to the execution phase.
• Any database object has a library cache object to represent it.
• In order to validate the dependencies for any cursor, the library cache keeps meticulously track of dependencies back and forth.
• The library cache consists of a hash table with pointers to handles, the handles can have pointers to additional memory areas.
• Concurrency of the library cache is protected by mutexes, a mutex protects a single memory area, which means for a cursor with a single child that is has 6 mutexes.
If time permits :-)

Questions and (hopefully) answers

…but wait! There is more info beyond this slide!
CURSOR CLOSE

Actually, parsing is **not** the first thing that happens when a foreground session gets active.

The first thing that happens as part of executing a normal SQL by a foreground is closing the previous cursor.

- Even if it’s the exact same SQL.
- Even if the cursor is in the session cursor cache.

The previous cursor is actually kept in open state until a new SQL execution occurs. This is visible in V$OPEN_CURSOR.

You might see freeing resources that were taken as part of execution of the previous cursor.
CURSOR CLOSE

`v$active_session_history` has a flag that indicates it’s in the codepath for closing a cursor: `in_cursor_close`.

A cursor is not closed and can be repeatedly executed after parsing only if it’s explicitly programmed, examples for Java en PL/SQL on the next slides.
PreparedStatement pstatement =
    con.prepareStatement("select /* prep test */ name from sys.obj$ where obj# = ?");
System.out.println("2. prepared statement created");
try { System.in.read(); } catch (IOException e) {}; 

ResultSet rs = null;
for (int i=1; i<100000; i++) {
    pstatement.setInt(1, i);
    rs=pstatement.executeQuery();

    while (rs.next()) {
        System.out.println("Current row from oracle is : "+rs.getString("NAME"));
    }
    System.out.println("prepared statement executed nr:"+i);
    try { System.in.read(); } catch (IOException e) {}}; 
}
PARSE ONCE, EXECUTE MANY: PL/SQL

declare
    cursor_name integer;
    cursor_number number;
    v_name varchar2(128) ;
begin
    cursor_name := dbms_sql.open_cursor;
    dbms_sql.parse(cursor_name,
        'select /* sql prep test */ name from sys.obj$ where obj# = :ob_nr', dbms_sql.native);
    for nr in 1..10 loop
        dbms_sql.bind_variable(cursor_name, ':ob_nr', nr);
        dbms_sql.define_column(cursor_name, 1, v_name, 128);
        cursor_number := dbms_sql.execute(cursor_name);
        cursor_number := dbms_sql.fetch_rows(cursor_name);
        dbms_sql.column_value(cursor_name, 1, v_name);
        dbms_output.put_line('loop nr: '||nr||' name: '||v_name);
    end loop;
end;
/

LIBRARY CACHE REPRESENTATION OF A CURSOR

After execution, state of the cursor in V$OPEN_CURSOR=‘OPEN’

LibraryHandle:  Address=0x78e4c670 Hash=89988315 LockMode=N PinMode=0 LoadLockMode=0 Status=VALD
ObjectName:  Name=select /* cursor test */ * from dual
Block:  #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x78f51398 Pointer=0x79676af8 Extent=0x79676980 Flags=I/-/P/A/-/-/

Child:  childNum='0'
LibraryHandle:  Address=0x7c5d7878 Hash=0 LockMode=N PinMode=0 LoadLockMode=0 Status=VALD
Block:  #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x797a8f78 Pointer=0x7923bd60 Extent=0x7923bbe8 Flags=I/-/P/A/-/-/

Block:  #='6' name=SQLA^89988315 pins=0 Change=NONE
Heap=0x796777138 Pointer=0x7f3c5af0 Extent=0x7f3c4f18 Flags=I/-/A/-/-/E
LIBRARY CACHE REPRESENTATION OF A CURSOR

Session executed another SQL, cursor is completely idle.

LibraryHandle:  Address=0x78e4c670 Hash=89988315 LockMode=0 PinMode=0 LoadLockMode=0 Status=VALID
ObjectName:  Name=select /* cursor test */ * from dual
Block:  #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x78f51398 Pointer=0x79676af8 Extent=0x79676980 Flags=I/-/P/A/-/-/

Child:  childNum='0'
LibraryHandle:  Address=0x7c5d7878 Hash=0 LockMode=0 PinMode=0 LoadLockMode=0 Status=VALID
Block:  #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x797a8f78 Pointer=0x7923bd60 Extent=0x7923bbe8 Flags=I/-/-/A/-/-/-

Block:  #='6' name=SQLA^89988315 pins=0 Change=NONE
Heap=0x79677138 Pointer=0x7f3c4f18 Extent=0x7f3c4f18 Flags=I/-/-/A/-/-/-/E
LIBRARY CACHE REPRESENTATION OF A CURSOR

When the SQL became cached in the session cursor cache, not opened.

LibraryHandle: Address=0x78e4c670 Hash=89988315 LockMode=\(N\) PinMode=0 LoadLockMode=0 Status=VALD
ObjectName: Name=select /* cursor test */ * from dual
Block: #='0' name=KGLH0^89988315 pins=0 Change=None
Heap=0x78f51398 Pointer=0x79676af8 Extent=0x79676980 Flags=I/-/P/A/-/-/
Child: childNum='0'
LibraryHandle: Address=0x7c5d7878 Hash=0 LockMode=\(N\) PinMode=0 LoadLockMode=0 Status=VALD
Block: #='0' name=KGLH0^89988315 pins=0 Change=None
Heap=0x797a8f78 Pointer=0x7923bd60 Extent=0x7923bbe8 Flags=I/-/P/A/-/-/
Block: #='6' name=SQLA^89988315 pins=0 Change=None
Heap=0x79677138 Pointer=0x7f3c5af0 Extent=0x7f3c4f18 Flags=I/-/A/-/-/E
LIBRARY CACHE REPRESENTATION OF A CURSOR

When the SQL is being executed between parse and execution phase (“active”).

```
LibraryHandle: Address=0x78e4c670 Hash=89988315 LockMode=N PinMode=0 LoadLockMode=0
Status=VALID
ObjectName: Name=select /* cursor test */ * from dual
Block: #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x78f51398 Pointer=0x79676af8 Extent=0x79676980 Flags=I/-/P/A/-/-/

Child: childNum='0'
LibraryHandle: Address=0x7c5d7878 Hash=0 LockMode=N PinMode=S LoadLockMode=0
Status=VALID
Block: #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x797a8f78 Pointer=0x7923bd60 Extent=0x7923bbe8 Flags=I/-/P/A/-/-/
Block: #='6' name=SQLA^89988315 pins=0 Change=NONE
Heap=0x79677138 Pointer=0x7f3c5af0 Extent=0x7f3c4f18 Flags=I/-/P/A/-/-/E
```
LIBRARY CACHE REPRESENTATION OF A CURSOR

When the SQL is being executed; hard parse (=child cursor creation), break at apaddrv.

LibraryHandle: Address=0x78e4c670 Hash=89988315 LockMode=N PinMode=0 LoadLockMode=0 Status=VALD
ObjectName: Name=select /* cursor test */ */ * from dual
Block: #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x78f51398 Pointer=0x79676af8 Extent=0x79676980 Flags=I/-/P/A/-/-/

Child: childNum='0'
LibraryHandle: Address=0x7c5d7878 Hash=0 LockMode=N PinMode=X LoadLockMode=0 Status=VALD
Block: #='0' name=KGLH0^89988315 pins=0 Change=NONE
Heap=0x797a8f78 Pointer=0x7923bd60 Extent=0x7923bbe8 Flags=I/-/P/A/-/-/
Block: #='6' name=SQLA^89988315 pins=0 Change=NONE
Heap=0x79677138 Pointer=0x7f3c5af0 Extent=0x7f3c4f18 Flags=I/-/P/A/-/-/E
EXTRA SLIDES
A resolution to avoid too ‘hot’ objects in the library cache is DBMS_SHARED_POOL.MARKHOT.

“Hot” means too much concurrency on the object, resulting in:
- cursor: pin S wait on X
- library cache lock
- cursor: mutex X

according to Oracle in DocID: 2369968.1

To use ‘markhot’:
- set “_kgl_hot_object_copies” (half of cpu_count)
- obtain X$KGLOB.KGLNAHSV (V$SQL.ADDRESS=X$KGLOB.KGLHDADR)
- exec DBMS_SHARED_POOL.MARKHOT(‘< X$KGLOB.KGLNAHSV>’,0);
- or “_kgl_debug”=“hash=‘< X$KGLOB.KGLNAHSV>’ namespace=0 debug=33554432
What is interesting, and important to know, is what it does.

1. The parent handle gets a flag ‘HOT’
2. “_kgl_hot_object_copies” number of additional parent handles are created with flag ‘HOC’ (my guess: HOT object handle Copy)

Let’s look at a schematic overview using the earlier library cache model, where:
• “_kgl_hot_object_copies” is set to 2.
• A cursor is marked as ‘HOT’
1. Regular Cursor

Hash buckets (parent) handles

KGLNA

Namespace=SQL AREA(00) Type=CURSOR(00)

KGLH0^XXXXXXXX

child handle

heap 0

SQL text

child handle

KGLH0^XXXXXXXX

heap 0

KGLH0^XXXXXXXX

heap 0

SQLA^XXXXXXXX

heap 6

KGLH0^XXXXXXXX

heap 0

SQLA^XXXXXXXX

heap 6
Hash buckets (parent) handles

```
2. DBMS_SHARED_POOL.MARKHOT is applied

Namespace=SQL AREA(00) Type=CURSOR(00)

Namespace=SQL AREA(00) Type=CURSOR(00)

Namespace=SQL AREA(00) Type=CURSOR(00)
```

Copyright © 2019 Accenture All rights reserved.
1. SQL text is hashed

2. Hash bucket is calculated (original hash value)

3. HOT flag is found, indicating it needs to use a HOC

4. New hash is created based on ?

5. Hash bucket is calculated (new hash value)

6. Handle is used normally from here
KKSSP MEMORY

Mutexes are allocated inside the object they protect. There is no “central registration” for mutexes, which does exist for latches; You can query the database to see which latches exist and who holds them.

In order for the database to be able to free mutexes for a session that has died, among other things, metadata is kept per session in the shared pool in memory that is annotated as KKSSP (Kernel Kompile Shared (objects) Session (state) Pointers). The mutex metadata is known as ‘AOL’: Atomic Operation Log.

KKSSP memory is visible in X$KSMSP.KSMCHCOM as ‘KKSSP^V$SESSION.SID’

The AOL for a mutex is stored in KKSSP in memory/subheap annotated as ‘kglss’.
Outside of Mutex metadata, other allocations in KKSSP that are known to me are:
- kgllk (library cache locks)
- kglpn (library cache pins)
- kglll (library cache load lock)
- kglss (library cache session state)
- KQR ENQ (kernel query rowcache enqueues)
- kglsesht (library cache session hash table)
- kglseshtSegs (~ segments)
- kglseshtTable (~ table)

At least for mutexes, it is documented PMON can use the KKSSP session data to recover mutex locks.
@who
SID_SERIAL_INST OSPID USERNAME C_SERVER PROGRAM EXTRA
--------------- -------- ------------- - --------- ---------------------------------
17,23683,@1 11472 TS DEDICATED sqlplus@o184.local (TNS V1-V3)
26,36116,@1 12512 SYS * DEDICATED sqlplus@o184.local (TNS V1-V3)
gdb -p 11472
break kgxRelease
c
>>> sid 17 executes SQL, breaks on kgxRelease
p/x $rsi --$rsi is second argument, which is a pointer to AOL
$s1 = 0x7ff91610
select distinct ksmchpar from x$ksmsp where ksmchcom = 'KKSSP^17';

KSMCHPAR
----------------
000000007F1926B8
oradebug setmypid
oradebug dump heapdump_addr 2 0x000000007F1926B8
vi tracefile
KKSSP MEMORY - FIND AOL

Chunk 07ff915b0 sz= 1096 freeable "kglss"

Dump of memory from 0x000000007FF915B0 to 0x000000007FF91F8

07FF915B0 00000449 10B38F00 7FF91580 00000000 [I.................]
07FF915C0 146C8334 00000000 00000000 00000000 [4.l.............]
07FF915D0 00000000 00000000 00000000 00000000 [................]

Repeat 1 times

07FF915F0 7F1926B8 00000000 7FDCE930 00000000 [.&.......0........]
07FF91600 7F064DE0 00000000 00000084 00000011 [.M..............]
07FF91610 61451860 00000000 003E0506 00040011 [\`Ea........>.....]
07FF91620 00FF0000 00000000 00000000 00000000 [................]
07FF91630 00000000 00000000 00000000 00000000 [................]
LIBRARY CACHE MEMORY AREAS

The library cache consists of several types of memory as described in X$KSMSP.KSMCHCOM.

• The hash table buckets are stored in memory marked as ‘permanent memor’.
• Only the namespace SQL AREA, type CURSOR seems to have children.
• Many different namespace/type’s have a heap 0, visible as KGL0^XXXXXXXXX.
• The memory comments for the different heaps is not always consistent.
• Heap 0 seems to be a memory area that can hold different types of data.
• Heap 6 seems to be holding runtime data for execution.
• The column X$KSMSP.KSMCHPAR can be used to dump the contents of the heap the chunk is in:

  oradebug dump heapdump_addr (1|2) 0x<X$KSMSP.KSMCHPAR>
  alter session set events ‘immediate trace name heapdump_addr address 0x<X$KSMSP.KSMCHPAR>’;
<table>
<thead>
<tr>
<th>Namespace</th>
<th>Type</th>
<th>C0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0. SQL AREA</strong></td>
<td>0. cursor (SQL)</td>
<td>Y</td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SQLA</td>
</tr>
<tr>
<td></td>
<td>0. cursor (PLSQL)</td>
<td>Y</td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PL/SQL MPCODE PL/Scope SQLP parameters repository</td>
</tr>
<tr>
<td></td>
<td>0. cursor (table_%)</td>
<td>Y</td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SQLK</td>
</tr>
<tr>
<td><strong>1. TABLE/PROCEDURE</strong></td>
<td>0. cursor</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. table</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. view</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. synonym</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. sequence</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9. package</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>13. type</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22. library</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>33. operator</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>66. scheduler job</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>68. scheduler class</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>2. BODY</strong></td>
<td>0. cursor</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11. package body</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PLMCD</td>
</tr>
<tr>
<td></td>
<td>14. type body</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>4. INDEX</strong></td>
<td>1. index</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. cluster</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>5. CLUSTER</strong></td>
<td>24. queue</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>39. pub sub internal info</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>KKTIN</td>
</tr>
<tr>
<td><strong>10. QUEUE</strong></td>
<td>46. ruleset</td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KGLH</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namespace</td>
<td>Type</td>
<td>C0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-----------------------------------------</td>
<td>----</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>45. MULTI-VERSION OBJECT FOR TABLE</td>
<td>70. multi versioned object</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48. MULTI-VERSION OBJECT FOR INDEX</td>
<td>70. multi versioned object</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51. SCHEDULER GLOBAL ATTRIBUTE</td>
<td>0. cursor</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64. EDITION</td>
<td>57. edition</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69. DLINK</td>
<td>0. cursor</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73. SCHEMA</td>
<td>255. none</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74. DBINSTANCE</td>
<td>0. cursor</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75. SQL AREA STATS</td>
<td>102. cursor stats</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82. SQL AREA BUILD</td>
<td>0. cursor</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93. AUDIT POLICY</td>
<td>115. audit policy</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>103. OPTIMIZER FINDING</td>
<td>126. optimizer finding</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>104. OPTIMIZER DIRECTIVE OWNER</td>
<td>0. cursor</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>114. RUNTIME STATISTICS</td>
<td>158. runtime</td>
<td>KGLH0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PINNING OBJECTS IN THE SHARED POOL

Memory allocations in the shared pool are as volatile as possible on purpose. Outside of X$KSMSP.KSMCHCLS ‘perm’, other memory types can be freed.

Of course when metadata is stored in non-permanent memory, you need a mechanism to temporarily disallow deallocation when it is in use. This mechanism is called ‘pinning’. Please mind the distinction between pinning an object handle of any type and pinning a chunk of memory.

The heap 0 of a SQL AREA CURSOR parent handle contains a table of pointers to child handles. As soon as a child handle is allocated, the parent heap 0 is pinned, because the child handle (and memory allocations of the child) is dependent on the parent heap 0. During execution, all heaps of all used handles are pinned so they can’t be deallocated during execution/usage.
PINNING OBJECTS IN THE SHARED POOL

Oracle allows you to manually keep the heaps of a given cursor or package. This can be accomplished using DBMS_SHARED_POOL.KEEP. I haven’t seen the need for it for a long time, Oracle shared pool memory administration has been enhanced significantly since Oracle 7 (single shared pool), and ‘newer’ 64 bit systems are not as physically limited in memory size like 32 bit once systems were.

A typical need would be higher parse times and/or shared pool latch wait time due to re-parsing of a library cache object. Outside of higher parse times, this should also visible as low GETHITRATIO in V$LIBRARYCACHE, and if it’s due to memory pressure, accompanied by RELOADS and not INVALIDATIONS for the namespace(s) that are used.
NOT MANUALLY PINNED OBJECT

LibraryHandle:  Address=0x78eee118 Hash=845e1263 LockMode=0 PinMode=0 LoadLockMode=0 Status=VALID
ObjectName:  Name=SYS.DBMS_OUTPUT
FullHashValue=88725469f5ab02458cbb736f845e1263 Namespace=BODY(02) Type=PACKAGE BODY(11)
Flags=TIM/[00002801] Flags2=[0000]
Block:  #='0' name=KGLH0^845e1263 pins=0 Change=NONE
Heap=0x791b2a78 Pointer=0x7ea85e60 Extent=0x7ea85c98 Flags=I/-/-/A/-/-/-

Block:  #='4' name=PLMCD^845e1263 pins=0 Change=NONE
Heap=0x7ea86138 Pointer=0x7e45ec08 Extent=0x7e45eb60 Flags=I/-/-/A/-/-/-

---

LibraryHandle:  Address=0x78e8f160 Hash=9b903123 LockMode=0 PinMode=0 LoadLockMode=0 Status=VALID
ObjectName:  Name=SYS.DBMS_OUTPUT
FullHashValue=025fb8a134ee50fb6fbcdal39b903123 Namespace=TABLE/PROCEDURE(01) Type=PACKAGE
Flags=PIN/TIM/[00002801] Flags2=[0000]
Block:  #='0' name=KGLH0^9b903123 pins=0 Change=NONE
Heap=0x7e2c67d8 Pointer=0x7eb16950 Extent=0x7eb16788 Flags=I/-/-/A/-/-/-

Block:  #='2' name=PLDIA^9b903123 pins=0 Change=NONE
Heap=0x7eb16b50 Pointer=(nil) Extent=(nil) Flags=I/-/-/-/-/-/-

Block:  #='4' name=PLMCD^9b903123 pins=0 Change=NONE
Heap=0x7eb16c28 Pointer=0x7e536750 Extent=0x7e5366a8 Flags=I/-/-/-/-/-/-
EXPLICITLY MANUALLY PINNED OBJECT

LibraryHandle:  Address=0x78eee118 Hash=845e1263 LockMode=0 PinMode= LoadLockMode=0 Status=VALD
ObjectName:  Name=SYS.DBMS_OUTPUT FullHashValue=88725469f5ab02458cbb736f845e1263 Namespace=BODY(02) Type=PACKAGE BODY(11) Flags=PIN/TIM/KEP/[00802801] Flags2=[0000]
Block:  #='0' name=KGLH0^845e1263 pins=0 Change=NONE Heap=0x791b2a78 Pointer=0x7ea85e60 Extent=0x7ea85c98 Flags=I/-/P/A/-/-/-
Block:  #='4' name=PLMCD^845e1263 pins=0 Change=NONE Heap=0x7ea86138 Pointer=0x7e45ec08 Extent=0x7e45eb60 Flags=I/-/P/A/-/-/-
--
LibraryHandle:  Address=0x78e8f160 Hash=9b903123 LockMode=0 PinMode= LoadLockMode=0 Status=VALD
ObjectName:  Name=SYS.DBMS_OUTPUT FullHashValue=025fb8a134ee50fb6fbcda139b903123 Namespace=TABLE/PROCEDURE(01) Type=PACKAGE BODY(11) Flags=PIN/TIM/KEP/[00802801] Flags2=[0000]
Block:  #='0' name=KGLH0^9b903123 pins=0 Change=NONE Heap=0x7e2c67d8 Pointer=0x7eb16950 Extent=0x7eb16788 Flags=I/-/P/A/-/-/-
Block:  #='2' name=PLDIA^9b903123 pins=0 Change=NONE Heap=0x7eb16b50 Pointer=0x7ec67000 Extent=0x7ec66f58 Flags=I/-/P/A/-/-/-
Block:  #='4' name=PLMCD^9b903123 pins=0 Change=NONE Heap=0x7eb16c28 Pointer=0x7e536750 Extent=0x7e5366a8 Flags=I/-/P/A/-/-/-
PINNING OBJECTS IN THE SHARED POOL

By explicitly pinning and thus fixing allocations, the shared pool memory manager is crippled/limited.

Only pin packages or cursors yourself if you are 100% sure it is needed.

There are some other objects that have the KEP flag set. Limited investigation leads me toward the following conclusion:

• CURSOR STATS objects can have the KEP flag set. It means the cursor child has at least has the KGLH0 memory area allocated.
• Certain data dictionary objects of type CLUSTER, INDEX, TABLE, MULTI-VERSIONED OBJECT(?) have the KEP flag set, sometimes the handle seems to be freed. (?)
EXTRA SLIDES
SMOKE AND MIRRORS: CURSOR CACHE

A SQL cursor is cached in the client cursor cache after 4 executions or 3 executions of the cursor already existed. That should be clear after this presentation.

However, a SQL cursor is cached in the client cursor cache immediately if it’s executed from PL/SQL:

```
declare
t  varchar2(1);
begin
select dummy into t from dual;
end;
/
```

```
@open_cursors 600
600  PL/SQL CURSOR CACHED       SELECT DUMMY FROM DUAL
```
SMOKE AND MIRRORS: CURSOR CACHE

Some documentation talks about SQL executed from PL/SQL being a different cache, which seems to be true, if you take the SQL and execute it directly, it shows up as a separate row:

```sql
SELECT DUMMY FROM DUAL;
```

```
@open_cursors 600
600 OPEN  SELECT DUMMY FROM DUAL
600 PL/SQL CURSOR CACHED  SELECT DUMMY FROM DUAL
```

If you execute it a few times, it becomes ‘SESSION CURSOR CACHED’, alongside the ‘PL/SQL CURSOR CACHED’ entry.

What actually happens becomes clear if you add a few columns from the V$OPEN_CURSOR view:
SMOKE AND MIRRORS: CURSOR CACHE

@open_cursors 600

<table>
<thead>
<tr>
<th>SID</th>
<th>CURSOR_TYPE</th>
<th>SQL_TEXT</th>
<th>SQL_ID</th>
<th>ADDRESS</th>
<th>CHILD_ADDRESS</th>
<th>HASH_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>SESSION CURSOR CACHED</td>
<td>SELECT DUMMY FROM DUAL</td>
<td>47pmdpk15c80p</td>
<td>0000000079721D50</td>
<td>000000007C571290</td>
<td>2186682389</td>
</tr>
<tr>
<td>600</td>
<td>PL/SQL CURSOR CACHED</td>
<td>SELECT DUMMY FROM DUAL</td>
<td>47pmdpk15c80p</td>
<td>0000000079721D50</td>
<td>000000007E1390D0</td>
<td>2186682389</td>
</tr>
</tbody>
</table>

Both have the same SQL_ID, because it’s the same SQL. Both have the the same ‘ADDRESS’, which is the parent handle. This means they are in exactly the same ‘cache’. Both have a different ‘CHILD_ADDRESS’, which means a different child handle and heaps.

The reason for the inability to share is: Top Level RPI Cursor. RPI=Recursive Program Interface; the cursors have been executed at a different recursive levels. => This means SQL executed directly and executed from PL/SQL always will need a different child.