
Mutexes
And Changes in Library Cache
Jože Senegačnik
joze.senegacnik@dbprof.com

About the Speaker

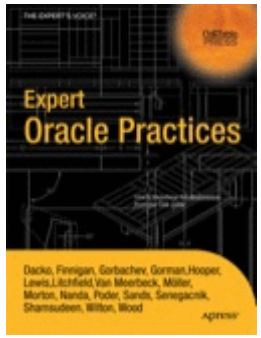
Jože Senegačnik

- Owner of Dbprof d.o.o.
- First experience with Oracle Version 4.1 in 1988
- 24+ years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com

OakTable.net

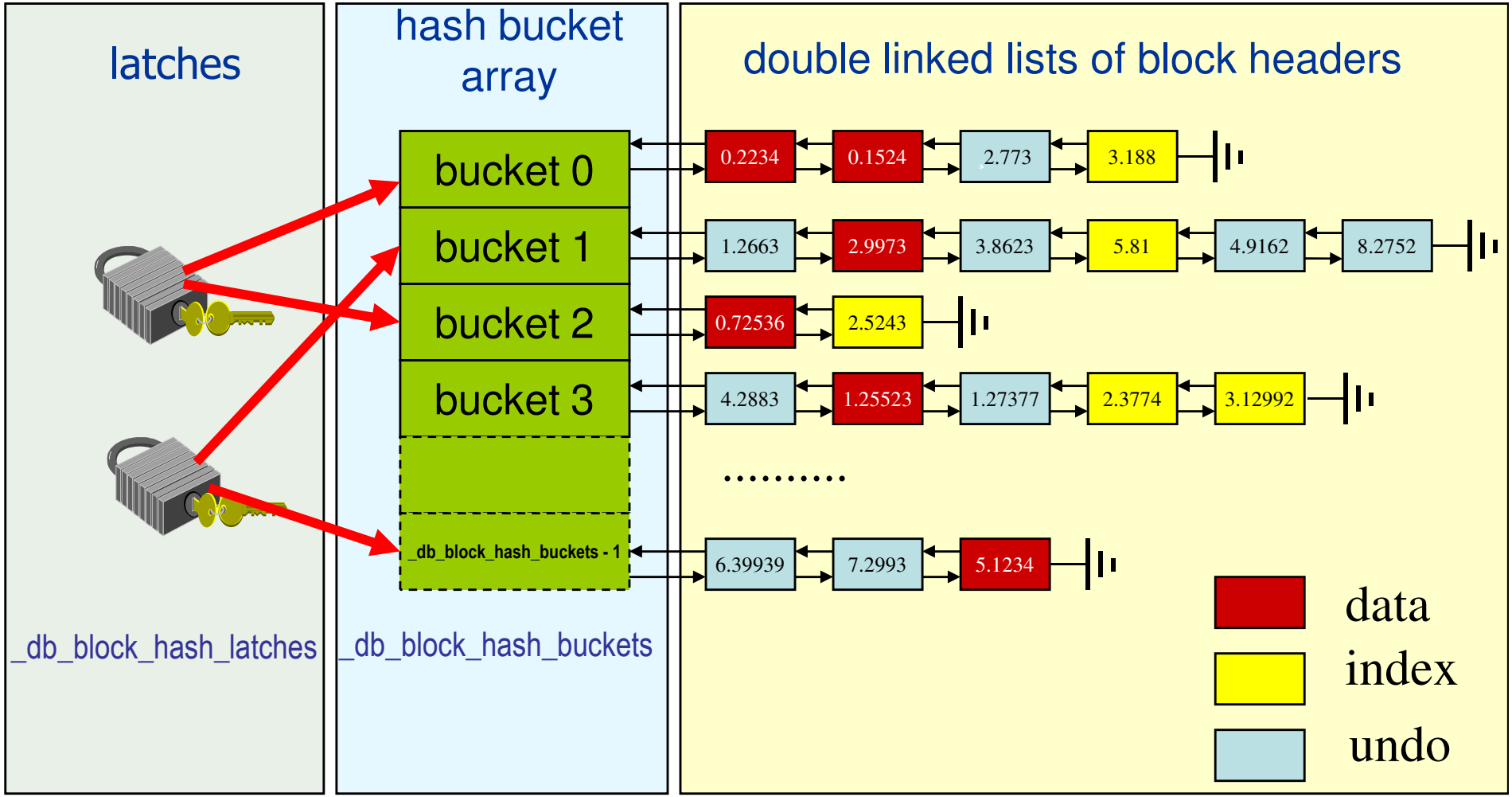
ORACLE ACE Director

Oracle ACE



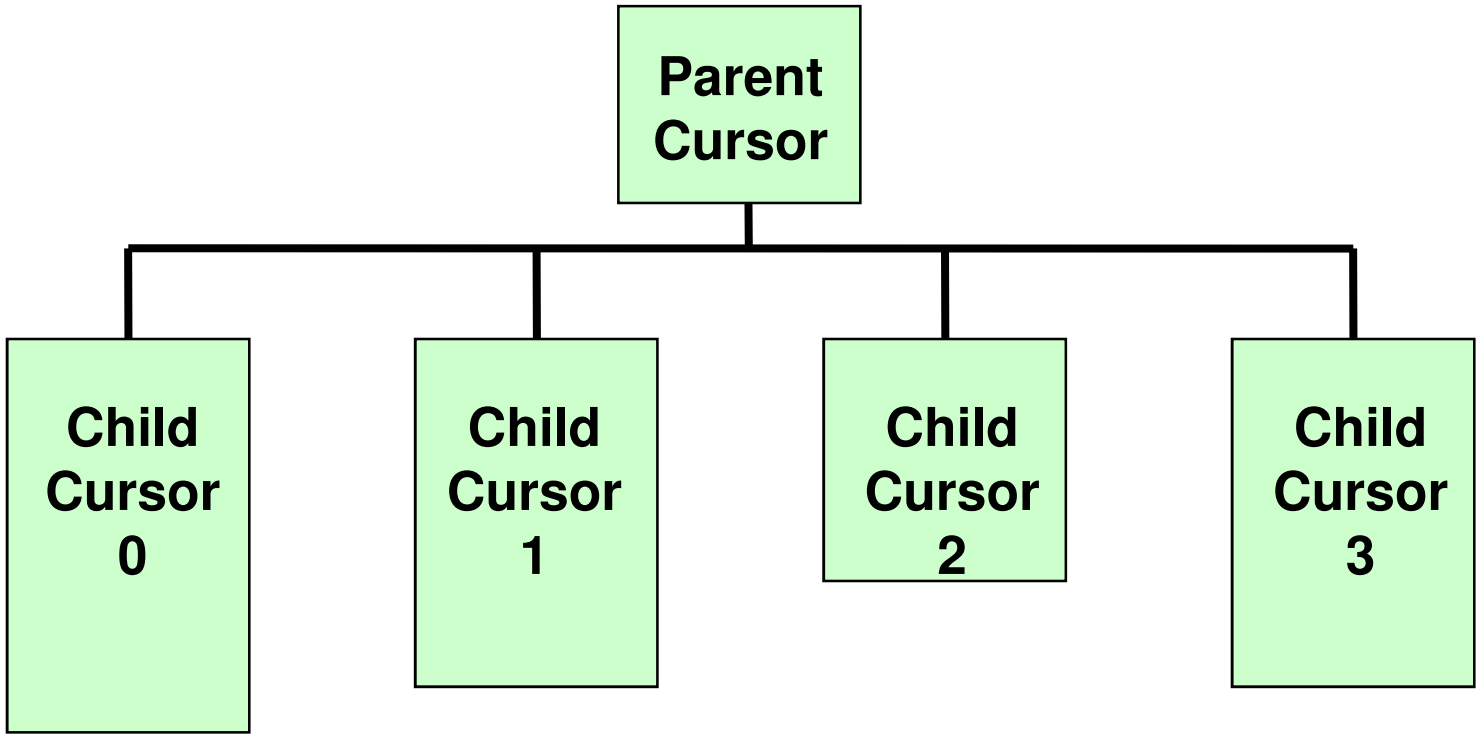
- PPL(A) – private pilot license, instrument rated
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

Latches, Hash Buckets & Data Blocks

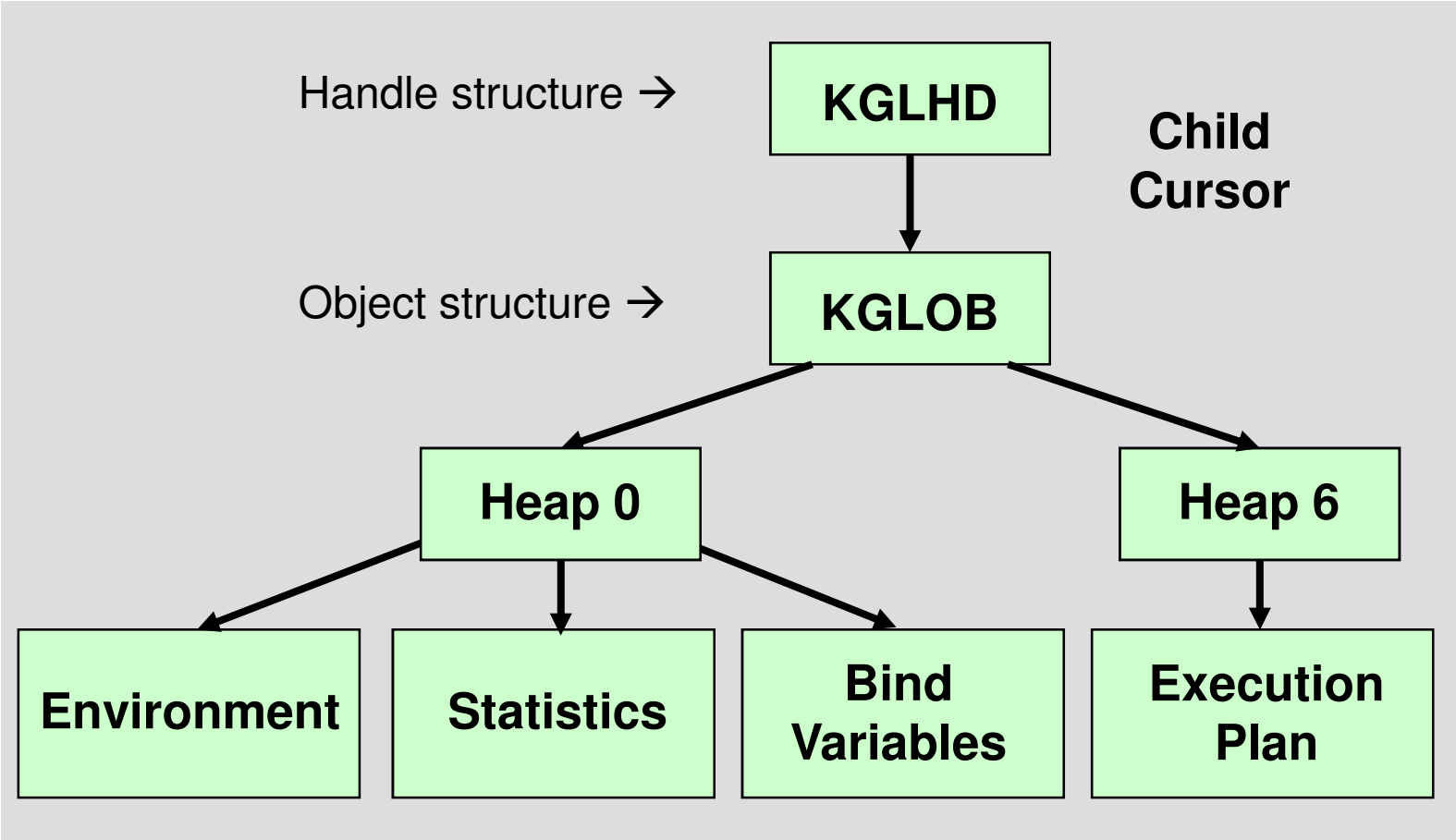


Parent / Child Cursors

- Each SQL statement has
  - Parent cursor
  - One or more child cursors
  - Each parent requires at least one child cursor



Structure of the Child Cursor



Object Heaps

- Every object can have optional sub heaps

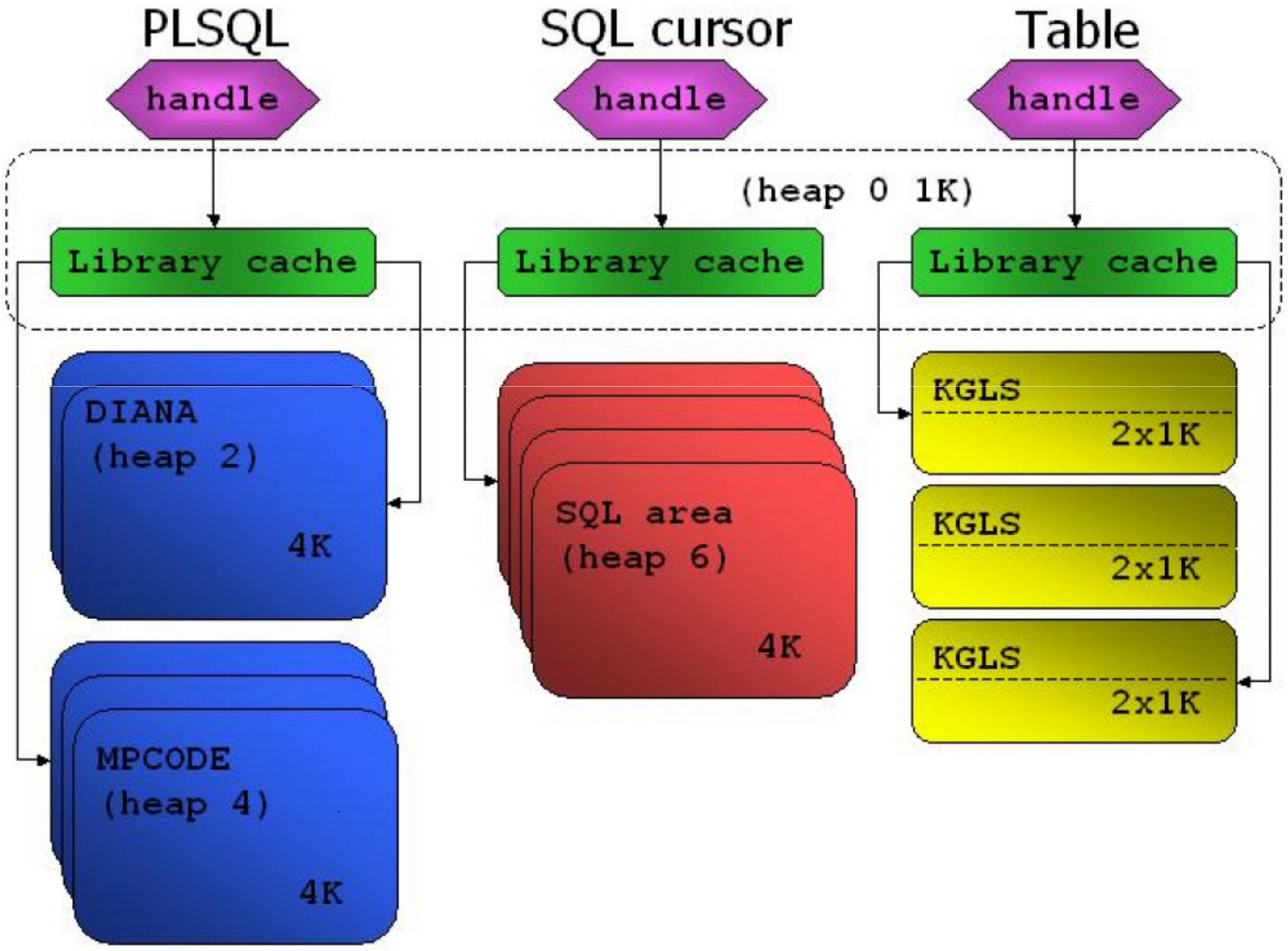
Heap #	Description
0	Object
1	Source
2	DIANA
3	PCODE
4	MCODE
5	Errors
6	SQL Context
7	Free
8	Subordinate Heap
9	Subordinate Heap
10	Subordinate Heap
11	Subordinate Heap

- Both parent and child cursors have sub heap 0
- In addition child cursors have sub heap 6 (execution plan)

Memory Allocations

- Memory for objects is comprised from several partitions which are independently allocated – so called heaps.
- Object can have different number of heaps – object type defines the number of heaps.
- SQL Cursor: has two heaps:
  - heap 0 – library cache metadata
  - heap 6 – executable representation of the cursor (sqlarea)
- Each heap is itself comprised of one or more chunks of memory (usually 4kB but can be also less), which are contiguous memory.

Shared Pool Heaps



Multiple Child Cursors

- Can be created for a number of reasons including differences in:
  - System / Session parameters
  - Object translation
  - Bind variables
  - NLS parameters
  - CBO environment
  - Tracing
  - ...
- Check V$SQL_SHARED_CURSOR
  - Heap 0 must still exist
  - Reports only for child numbers > 0

```
SELECT TO_CHAR (bitvector,'XXXXXXXXXXXXXXX'),
COUNT(*) FROM x$kkscs
GROUP BY  TO_CHAR (bitvector,'XXXXXXXXXXXXXXX')
ORDER BY 1;
```

Benefits of SESSION_CACHED_CURSORS

- When cursor is in the session cursor cache then the cost of finding cursor in SGA is minimal.
  - less latching
  - less CPU
  - Concurrency improvement will apply only to already closed cursors which are afterwards reparsed several times.
- Negative effect: cursors are partly pinned (heap 0) and therefore we should check the size of shared pool.
- More cached cursors less candidates for aging out – for freeing memory!!!

Efficiency of setting SESSION_CACHED_CURSORS

- We can check the efficiency of setting session_cached_cursors parameters with event 10270 ("Debug shared cursors") at level 10.
- SQL> alter session set events '10270 trace name context forever, level 10';
- **session_cached_cursors = 0**

```
#1: checking for already pinned child
#1: no valid child pinned
#1: pinning parent in shared mode to search 68509d68 694dcce0
#1: ksfbc: calling kksSearchChildList  outside while loop
#1: kkslce: next child is #1
#1: kkslce: pinning child #1 in shared mode 6832a12c 69f44490
#1: kksCheckCriteria: calling kksauc
#1: kksCheckCriteria: unpinning the parent
```

- **session_cached_cursors > 0**

```
#1: checking for already pinned child
#1: no valid child pinned
#1: child already locked 68509d68, checking for validity
```

- Each line means one step in performing **soft parse**. When this parameter is set to values bigger than 0 then the soft parse is even "**softer**".

V$SYS_TIME_MODEL

```
SQL> select inst_id,stat_name, round(value/1000000,0) as seconds
     from gv$sys_time_model order by 2;

INST_ID STAT_NAME                                                SECONDS
------- ---------------------------------------------------- --------
      2 PL/SQL compilation elapsed time                          6847
      1 PL/SQL compilation elapsed time                          2548
      2 PL/SQL execution elapsed time                           64182
      1 PL/SQL execution elapsed time                           42829
      2 failed parse (out of shared memory) elapsed time            0
      1 failed parse (out of shared memory) elapsed time            0
      2 failed parse elapsed time                                1827
      1 failed parse elapsed time                                 441
      2 hard parse (bind mismatch) elapsed time                   259
      1 hard parse (bind mismatch) elapsed time                    86
      2 hard parse (sharing criteria) elapsed time               2112
      1 hard parse (sharing criteria) elapsed time                881
      1 hard parse elapsed time                                  9280
      2 hard parse elapsed time                                 29582
      2 parse time elapsed                                      33952
      1 parse time elapsed                                      10805
      1 repeated bind elapsed time                                 59
      2 repeated bind elapsed time                                142
      1 sql execute elapsed time                               288100
      2 sql execute elapsed time                              1017934
...
```

Parsing Latches (Prior Mutexes)

- Hard parse requires:
  - library cache latch, shared pool latch, library cache pin, execute
  - These latches are required for memory allocation (shared pool latch) and object creation (library cache latch)
- Soft Parse
  - library cache latch, library cache pin, execution
  - Latches are required for object lookup and execution
- Session cached cursor
  - library cache pin, execute
  - Cache lookup, object execution
- Re-executing open cursor
  - library cache pin, execution
  - Object execution

Library Cache Misses (1)

- 2 types of “library cache misses”
  - **Misses in library cache during parse** -> Misses while looking for execution plan in library cache – this means “Hard parse” - normal situation
  - **Misses in library cache during execute** -> Misses while about to execute the plan and found it missing/invalid in library cache

- Reasons for library cache miss during execution:
  - Statement invalidation due to some significant change has occurred to the referenced objects
  - Execution plan was flushed from the library cache (and “reloaded”) fairly frequently due to the demand for free memory
    - Execution plans are re-creatable, so they can be flushed from memory even when the cursor is being held by the client.
- Investigation to determine the right cause:
  - library cache misses reported in SQL trace (10046 trace)
  - Examining V$SQL area

Mutexes

Purpose of Mutexes

- **Mutex** - mutual exclusion algorithms are used in so called critical section of the computer code to avoid the concurrent use of un-shareable resources.
- **Mutex** is used within Oracle as a low-level serialization mechanism to control access to shared data structures in the SGA. (Similarly like latches)
- Serialization is required to avoid an object being:
  - read while someone is modifying it
  - modified while someone is modifying it
  - modified while someone is reading it
  - de-allocated while someone is accessing it

- Since Oracle 10gR2 mutexes are used to protect the shared cursor component of the library cache.

Mutex Holding Modes

- S - Shared mode. The reference (ref) count is used to store
  - the total number of sessions referencing a mutex in S (shared) mode
  - Used as a replacement for the library cache pin to protect aging out whilst the object is used. Only a ref count is incremented / decremented.
- X –exclusive mode – when held only by one session

Mutex Benefits (1)

- Smaller and faster
  - faster to get in comparison with latches and use less memory
- Less potential for false collision
  - Latches usually protect multiple objects and are therefore candidates for false contention – it is contention for the latch rather than the shared resource they protect.
  - Mutexes are smaller and are usually part of the structure they protect or there may be several mutexes to protect one structure. Therefore false contention is less likely.
- Use more flexible wait strategy
  - wait can be: blocking wait, sleep or CPU yield
- They can be held in shared or exclusive mode.

Mutex Benefits (2)

- Re-executing SQL statements already loaded in the library cache.
- When Library cache pin is used:
  - Cursor must be pinned by each session prior execution.
  - Library cache pin must be acquired when cursor is left opened for re-execution and CURSOR_SPACE_FOR_TIME=FALSE.
  - Library cache pin is not required when cursor is left opened for re-execution and can be rebound and CURSOR_SPACE_FOR_TIME=TRUE.
- Mutexes instead of a “library cache pin”
  - When mutexes are used instead of library cache pin replacements, CURSOR_SPACE_FOR_TIME does not need to be set. The session executing shared cursor will only need a mutex pin – it will increment the ref count when re-executing the cursor.
  - Parent examination: The parent cursor is examined when finding a child cursor to execute. Mutex is used for the parent examination.
  - Building a new cursor under a parent – is cheaper, however building many cursors under a parent cursor is not recommended.

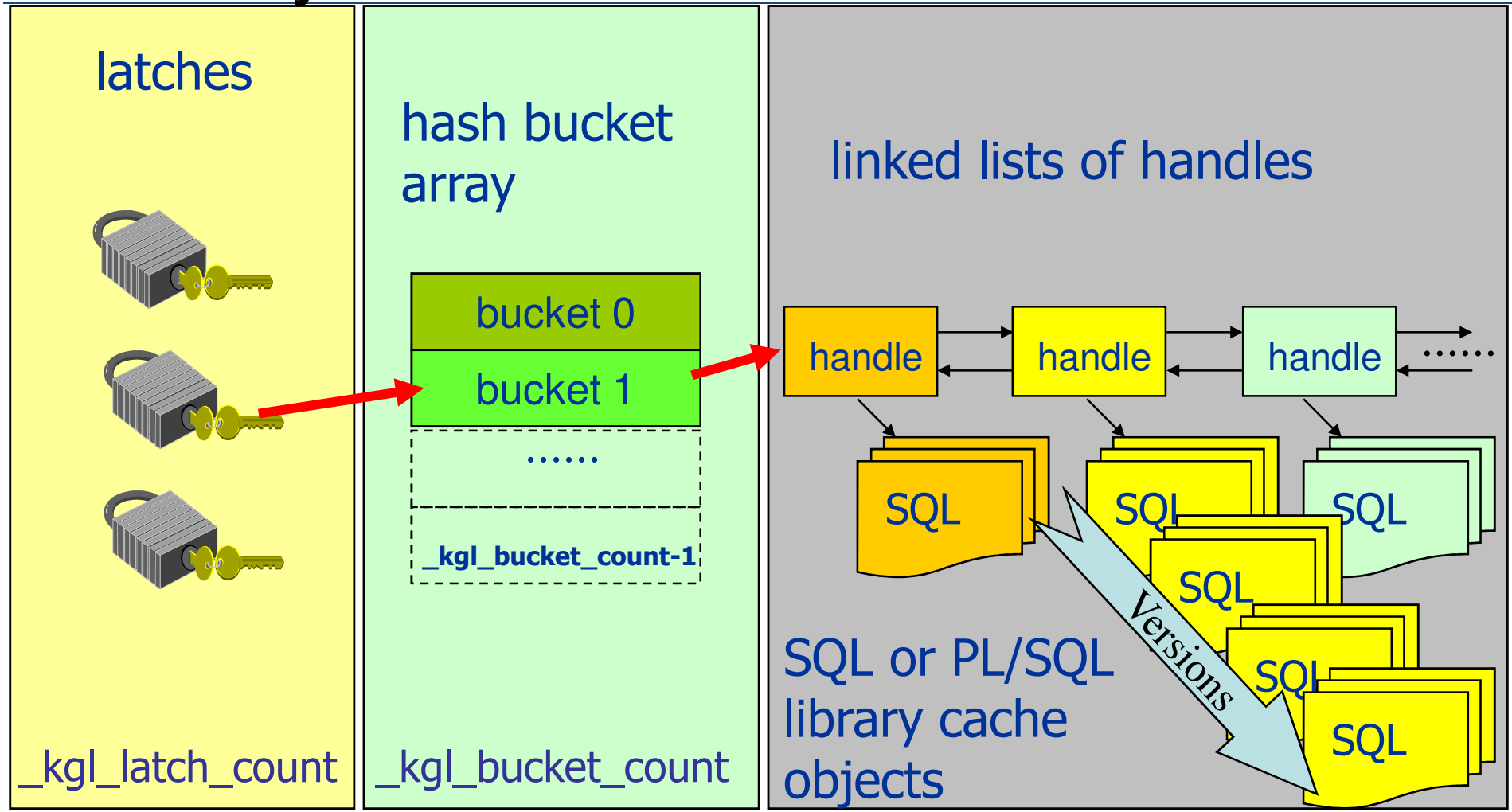
Mutex Benefits (3)

- Querying SQL statistics via V$SQLSTATS
- V$SQLSTATS uses mutexes and therefore performs better than accessing statistics via v$sqlarea or v$sql
  - less CPU
  - significantly fewer latch gets

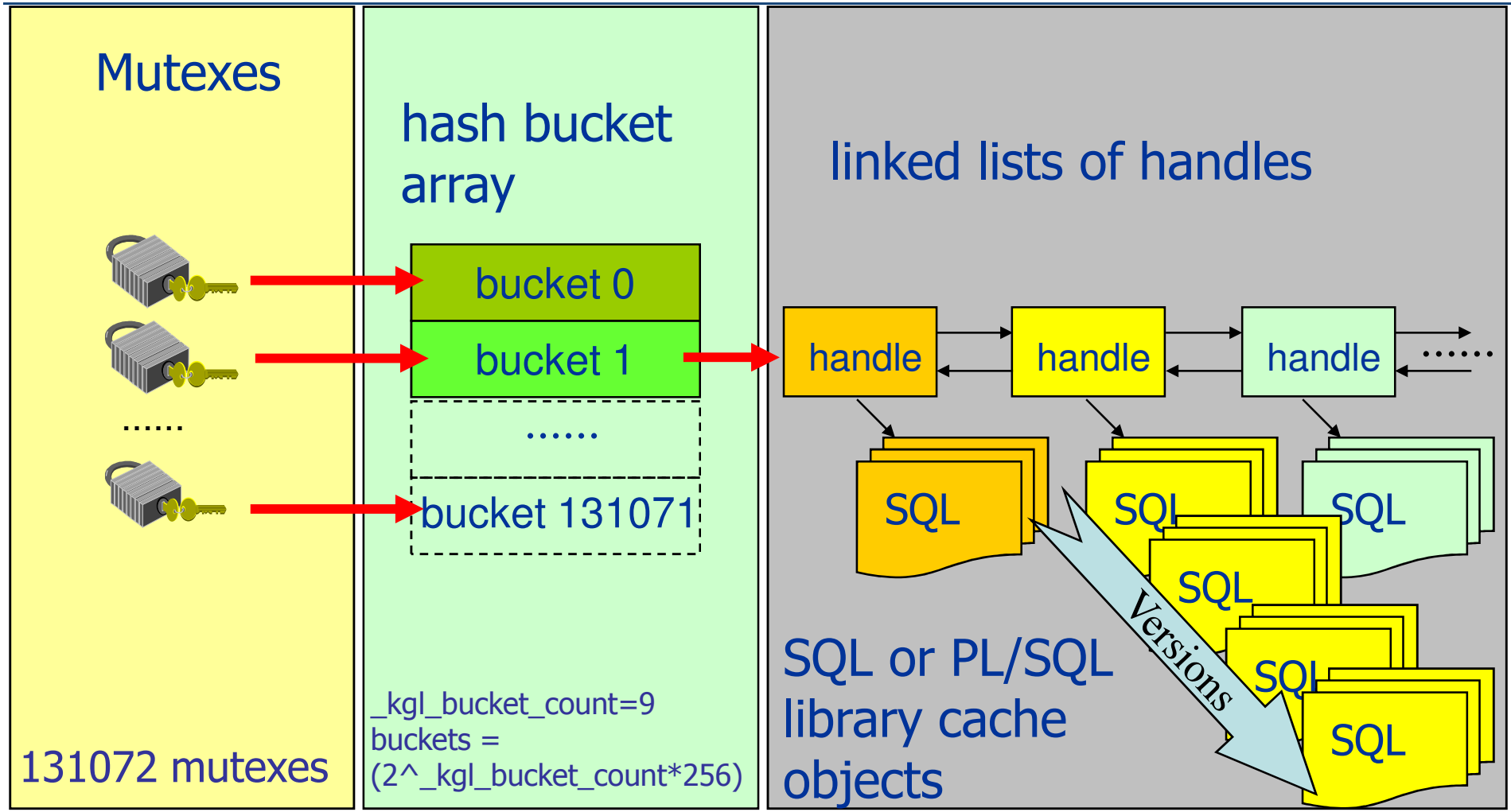
Mutex Creation and Usage

- The mutex can either be stored within the structure it protects, or elsewhere.
- They are usually dynamically created at the same time as the structure they protect.
- When embedded within the structure it protects, then they will share the destiny of the structure - they will be destroyed when the owning structure is destroyed (freed).
- Mutexes can be defined and used in many different ways:
  - Each structure being protected by a mutex can have its own mutex (e.g. a parent cursor has its own mutex, and each child cursor has its own mutex)
  - Each structure can be protected by more than one mutex, with each mutex protecting a different part of the structure
  - A mutex can protect more than one structure.

Library Cache Latch



Library Cache Mutexes



Mutex Types and Sleeps

```
select mutex_type,count(*)
from sys.v$mutex_sleep_history
group by mutex_type order by 2 desc;

MUTEX_TYPE                       COUNT(*)
-------------------------------- ----------
Library Cache                           602
Cursor Pin                               42
Cursor Stat                               5
Cursor Parent                             1

4 rows selected.
```

(from a production database 11.2.0.2 with about 1000 sessions)

Mutex Locations

```
select mutex_type,location,count(*)
from v$mutex_sleep_history
group by mutex_type,location order by 3 desc
```

```
MUTEX_TYPE     LOCATION                                    COUNT(*)
-------------- ------------------------------------------ ----------
Library Cache  kglpin1   4                                       171
Library Cache  kglget1   1                                       105
Library Cache  kglpndl1  95                                      104
Library Cache  kglpnal1  90                                       84
Library Cache  kglhdgn2 106                                       44
Library Cache  kglhdgn1  62                                       25
Library Cache  kgllkc1   57                                       25
Cursor Pin     kksfbc [KKSCHLFSP2]                                18
Library Cache  kgllkdl1  85                                       16
Library Cache  kglget2   2                                        16
Cursor Pin     kksLockDelete [KKSCHLPIN6]                         12
Cursor Pin     kkslce [KKSCHLPIN2]                                11
Library Cache  kglobpn1  71                                        9
Cursor Stat    kkocsStoreBindAwareStats [KKSSTALOC8]               5
Cursor Parent  kkscsAddChildNode [KKSPRTLOC34]                     1
Cursor Pin     kksfbc [KKSCHLPIN1]                                 1
Library Cache  kglldl2 112                                         1
Library Cache  kglhdgh1  64                                        1
Library Cache  kglhbh1   63                                        1

19 rows selected.
```

Hight Version Count Statements

```
select sql_id,version_count,substr(sql_text,1,30)
from v$sqlarea where version_count > 100
order by 2 desc;

SQL_ID        VERSION_COUNT SUBSTR(SQL_TEXT,1,30)
------------- ------------- ------------------------------
0v3dvmc22qnam          2799 insert into sys.col_usage$ (ob
14n9gy4jw9nuh           620 SELECT effect_date, cnt_dr, su
2bpp4r8ajsuz3           328 select ID,SERVICEID,ACCT_ID,
f711myt0q6cma           291 insert into sys.aud$( sessioni
7a85zvjn43j6r           278 INSERT INTO XMS_PAYMENT (
a9s5xz5v4qw95           166 select ID,SERVICEID,ACCT_ID,
dyfyjuyfpn8rm           148 select /*+ RULE */ L.LDATE,L.E
83cq1aqjw5gmg           110 select ownername,classname,met

8 rows selected.
```

The Mystery of Numbers

```
select count(*) from v$sql_shared_cursor where sql_id='0v3dvmc22qnam';

  COUNT(*)
----------
         1

select count(*) from v$sql where sql_id='0v3dvmc22qnam';

  COUNT(*)
----------
         1

select count(*) from sys.x$kglob where kglobt03='0v3dvmc22qnam';

  COUNT(*)
----------
      2800
```

Mutex Usage - “Cursor Parent”

- Mutex type: ‘Cursor Parent’
- Used in parent-cursor operations:
  - building a new cursor under a parent
    - wait event ‘cursor: mutex X’
  - examining a parent
    - wait event ‘cursor: mutex S’
  - bind capture
    - wait event ‘cursor: mutex X’

Mutex Usage – “Cursor Statistics”

- Mutex type ‘Cursor Stat’
- Used for cursor statistics operations:
  - building and updating cursor-related statistics
    - wait event ‘cursor: mutex X’
  - examining cursor-related statistics (usually querying v$sqlstats)
    - wait event ‘cursor: mutex S’

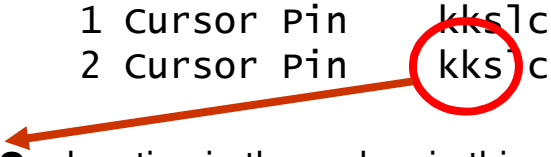
Mutex Usage – "Cursor Pin"

- 'library cache pin' is replaced by a mutex and a ref count. Used for pinning (e.g. for execution), or compiling a child cursor.
- Used in 10.2.0.2+ and library cache pins are not used any more (no 'library cache pin' wait events)
- Used for:
  - Pinning a cursor for execute
    - wait event: 'cursor: pin S wait on X'
  - When pinning a cursor for execute, and the cursor is currently being examined by another S
    - wait event 'cursor: pin S'
  - Cursor rebuild
    - wait event ('cursor: pin X').
    - This event should not be seen typically, because if a cursor is currently being used, and it needs to be rebuilt, another cursor will be created

V$MUTEX_SLEEP

```
SQL> select * from GV$MUTEX_SLEEP order by 2,3;

INST_ID MUTEX_TYPE    LOCATION                          SLEEPS  WAIT_TIME
------- ------------- ------------------------------ ------ ----------
      2 Cursor Parent kksfbc [KKSPRTLOC1]                 2          12
      1 Cursor Parent kksfbc [KKSPRTLOC2]                 1           3
      2 Cursor Parent kksfbc [KKSPRTLOC2]                 9          38
      2 Cursor Parent kksfbc [KKSPRTLOC3]                 1           7
      2 Cursor Parent kkspsc0 [KKSPRTLOC26]              57         297
      1 Cursor Parent kkspsc0 [KKSPRTLOC26]              10          44
      2 Cursor Parent kkspsc0 [KKSPRTLOC27]              61         321
      1 Cursor Parent kkspsc0 [KKSPRTLOC27]              22          96
      2 Cursor Pin    kksLockDelete [KKSCHLPIN6]          6          23
      1 Cursor Pin    kksfbc [KKSCHLFSP2]               253     3091383
      2 Cursor Pin    kksfbc [KKSCHLFSP2]               494     5864297
      1 Cursor Pin    kksfbc [KKSCHLPIN1]                 2       27080
      1 Cursor Pin    kkslce [KKSCHLPIN2]             24852   384452146
      2 Cursor Pin    kkslce [KKSCHLPIN2]             92401  1433220667
```



kks – location in the code – in this case this part of code is support for managing shared cursors/ shared sql (see Metalink Note:175982.1 - ORA-600 lookup Error Categories)

Data from 2-node RAC 10.2.0.1 Standard Edition – the instance was running for 9 days

"Cursor: pin S wait on X" waits

- **MOS ID 786507.1**
- A session waits on this event when requesting a mutex for shareable operations related to pins (such as executing a cursor), but the mutex cannot be granted because it is being held exclusively by another session (which is most likely parsing the cursor).

- Causes:
  - Frequent Hard Parses
    - If the frequency of Hard Parsing is extremely high, then contention can occur on this pin.
  - High Version Counts
    - When Version counts become excessive, a long chain of versions needs to be examined and this can lead to contention on this event

- One of the most likely causes of cursor: pin S wait on X is high parsing time.

Blocker Session Id

- select p2raw from v$session where event = 'cursor: pin S wait on X';

```
P2RAW
----------------
0000001F00000000
 <SID>  <RefCnt>
```

- Blocker session_id is in top 8 bytes.
- 0000001F (hex) = 31 (decimal).

Cursor: pin S

- A session waits for "cursor: pin S" when it wants a specific mutex in S (share) mode on a specific cursor and there is no concurrent X holder but it could not acquire that mutex immediately.
- A wait on "cursor: pin S" thus occurs if a session cannot make that atomic change in ref count immediately due to other concurrent requests.
- Wait event parameters:
  - **P1 = idn**
    - Mutex identifier which matches to the HASH_VALUE of the SQL statement that we are waiting to get the mutex on.
  - **P2 = value**
    - The value is made up of 2 parts:
    - High 8 order bits contain the session id of the session holding the mutex
    - Low order bits contain a reference count *(like the number of other S mode holders)*
  - **P3 = where** (where | sleeps)

High Version Count and Cursortrace

- `alter system set events 'immediate trace name cursortrace level 577, address hash_value';`

- High SQL Version Counts - Script to determine reason(s) [ID 438755.1]
- Miladin Modrakovic:Cursor high_version count Part 1
- http://oraclue.com/2008/12/03/cursor-high_version-count-1/

Cardinality Feedback Tuning

```
select sql_id,count(*)
from v$sql_shared_cursor
where use_feedback_stats='Y'
group by sql_id order by 2 desc;

SQL_ID          COUNT(*)
------------- ----------
b7xhjjbmqdms4           5
gu7my5yzjm1ap           3
7s909d9xhykp9           3
2dfqzhstwt3hx           2
9j2zwk9b9zbwg           2
4hnwhsp5p4wwb           2
05uzz5j7qtf77           2
ff5spha9pjuyu           2
4rvk1jwvkam8a           2
```

Cardinality Feedback Tuning

```
select sql_text from v$sql where sql_id = 'b7xhjjbmqdms4';

SQL_TEXT
--------------------------------------------------------------------------------
SELECT axs_profid as Id, name as Name    FROM  axs_profile  WHERE is_named > 0   ORDER BY Name
SELECT axs_profid as Id, name as Name    FROM  axs_profile  WHERE is_named > 0   ORDER BY Name
SELECT axs_profid as Id, name as Name    FROM  axs_profile  WHERE is_named > 0   ORDER BY Name
SELECT axs_profid as Id, name as Name    FROM  axs_profile  WHERE is_named > 0   ORDER BY Name
SELECT axs_profid as Id, name as Name    FROM  axs_profile  WHERE is_named > 0   ORDER BY Name

5 rows selected.
```

References

- Julian Dyke, Finding the Trash in the Library Cache, UKOUG 2006
- Understanding Shared Pool Memory Structures, Oracle White Paper, September 2005

Thank you for your interest!

Q&A