**实例调优**

云和恩墨(北京)信息技术有限公司

技术顾问 燕鑫

http://www.enmotech.com

**文档控制：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序** | **版本号** | **更改人** | **日期** | **备注** |
| 1 | 1.0版 | 燕鑫 | 2019-01-16 | 初始版本 |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. 调优方法论 - 6 -](#_Toc2850131)

[2. Enqueues - 6 -](#_Toc2850132)

[2.1 锁类型 - 6 -](#_Toc2850133)

[2.2 Enqueues - 6 -](#_Toc2850134)

[2.3 Enqueue的定义 - 7 -](#_Toc2850135)

[2.4 Enqueue的模式 - 7 -](#_Toc2850136)

[2.5 Enqueue的兼容性 - 7 -](#_Toc2850137)

[2.6 资源和锁 - 8 -](#_Toc2850138)

[2.7 Client and Managed Enqueues - 9 -](#_Toc2850139)

[2.8 State Objects - 9 -](#_Toc2850140)

[2.9 Hashing and Latching - 9 -](#_Toc2850141)

[2.10 Resource Free List - 11 -](#_Toc2850142)

[2.11 初始化参数（都是老参数，有待进一步验证） - 11 -](#_Toc2850143)

[2.12 锁获取 - 11 -](#_Toc2850144)

[2.13 锁的转换 - 12 -](#_Toc2850145)

[2.14 锁的释放 - 12 -](#_Toc2850146)

[2.15 Enqueue Operation - 12 -](#_Toc2850147)

[2.16 Enqueue等待事件 - 13 -](#_Toc2850148)

[2.17 Deadlock Detection（死锁探测） - 16 -](#_Toc2850149)

[2.18 Diagnostic - 16 -](#_Toc2850150)

[2.18.1 State Object Dump - 16 -](#_Toc2850151)

[2.18.2 Enqueues Dump - 17 -](#_Toc2850152)

[2.18.3 相关视图 - 17 -](#_Toc2850153)

[2.18.4 脚本 - 24 -](#_Toc2850154)

[3. Lock Contention - 24 -](#_Toc2850155)

[3.1 Data Dictionary Locks - 24 -](#_Toc2850156)

[3.1.1 Row Cache Locks - 25 -](#_Toc2850157)

[3.1.2 Wait Event:Row Cache Lock - 25 -](#_Toc2850158)

[3.1.3 Library Cache Locks - 28 -](#_Toc2850159)

[3.1.4 Wait Event: Library Cache Lock - 29 -](#_Toc2850160)

[3.1.5 Library Cache Pins - 30 -](#_Toc2850161)

[3.1.6 Wait Event: Library Cache Pin - 31 -](#_Toc2850162)

[3.2 DML Locks - 31 -](#_Toc2850163)

[3.2.1 DML Table Locks: Implementation - 31 -](#_Toc2850164)

[3.2.2 Disabling DML Table Locks - 32 -](#_Toc2850165)

[3.2.3 DML Row Locks - 32 -](#_Toc2850166)

[3.2.4 Buffer Locks - 35 -](#_Toc2850167)

[3.2.5 Sort Locks - 38 -](#_Toc2850168)

[3.2.6 ORA-1575 - 39 -](#_Toc2850169)

[3.2.7 SMON Functions - 39 -](#_Toc2850170)

[4. Latch Internals - 42 -](#_Toc2850171)

[4.1 Latch的作用 - 42 -](#_Toc2850172)

[5. Memory Management - 42 -](#_Toc2850173)

[5.1 System Global Area - 42 -](#_Toc2850174)

[5.1.1 Dynamic SGA Memory Allocation - 42 -](#_Toc2850175)

[5.1.2 Granule Size - 43 -](#_Toc2850176)

[5.1.3 SGA与大页 - 44 -](#_Toc2850177)

[5.1.4 Fixed Area - 44 -](#_Toc2850178)

[5.1.5 Buffer Cache - 45 -](#_Toc2850179)

[5.1.6 Redo Buffers - 45 -](#_Toc2850180)

[5.1.7 Heaps and Subheaps - 45 -](#_Toc2850181)

[5.1.8 Shared Pool - 46 -](#_Toc2850182)

[5.1.9 Large Pool - 49 -](#_Toc2850183)

[5.1.10 Java Pool - 49 -](#_Toc2850184)

[5.2 Process Global Area - 50 -](#_Toc2850185)

[5.3 User Global Area - 50 -](#_Toc2850186)

[5.4 Call Global Area - 51 -](#_Toc2850187)

[5.5 Heap和Subheap的DUMP - 51 -](#_Toc2850188)

# 调优方法论

# Enqueues

## 锁类型

Oracle的锁分为如下几类：

1. Data Dictionary Locks (DDL)  
   – Row cache locks  
   – Library cache locks and pins
2. Data Manipulation Locks (DML)  
   – Row locks  
   – Table locks
3. Internal locks and latches：内部锁和latch
4. Distributed locks：分布式事务锁
5. Parallel Cache Management (PCM) locks：RAC中的PCM锁

## Enqueues

Enqueues是KSQ（Kernel Service Enqueues）layer提供的服务。

其他利用该服务的layer叫做client layer。

Enqueues是一种用于管理共享资源访问（access shared resource）的锁机制。

Enqueues涉及的范围可以是一个实例中，也可以是跨实例的。

KSQ提供的服务接口有：acquire、convert和release enqueues。

如果enqueue无法获取，client layer可以选择一直等，也可以选择timeout超时不等。

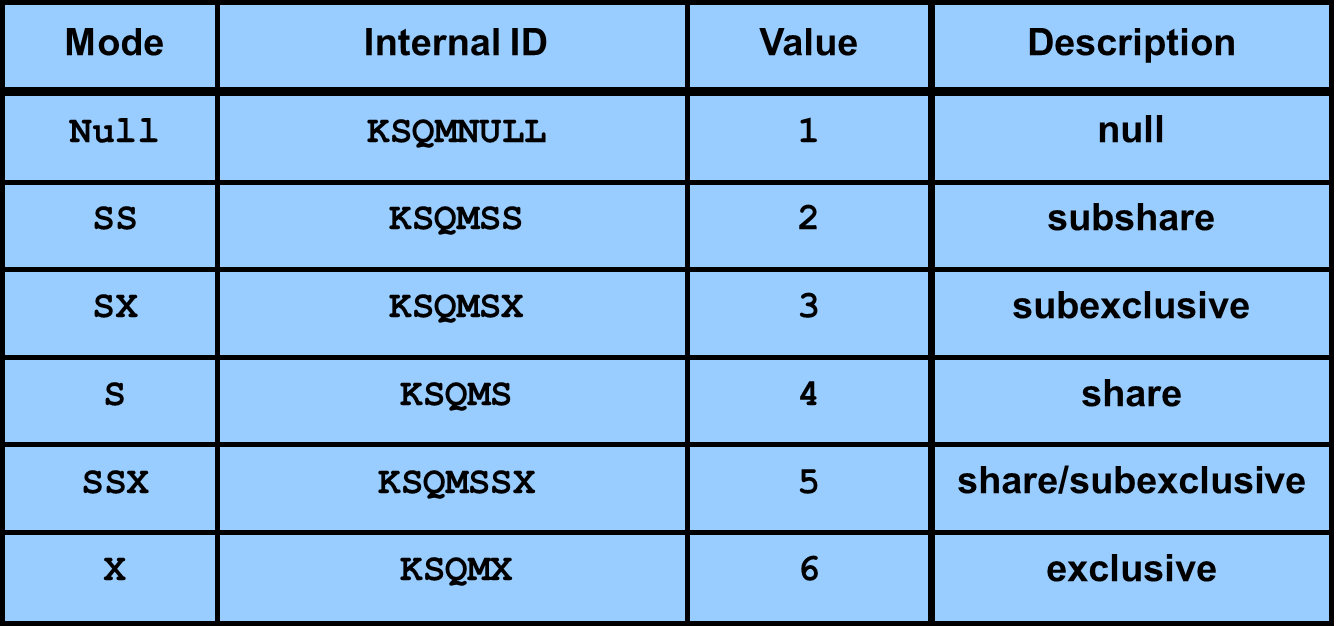
## Enqueue的定义

每一个Enqueue都有自己唯一的名字，就是我们常说的锁或者队列资源的名字。

Enqueue的定义格式为：<Type-ID1-ID2>:

1. Type is a two-character string, for example：“TM.”
2. ID1 and ID2 are positive numbers (4 bytes).

## Enqueue的模式



## Enqueue的兼容性

说白了，一个会话对一个资源acquire了一个锁，另一个想要在这个资源上acqurie另一个锁，如果后者与前者兼容，那么后者就可以acquire成功，否则就是要等待。



## 资源和锁

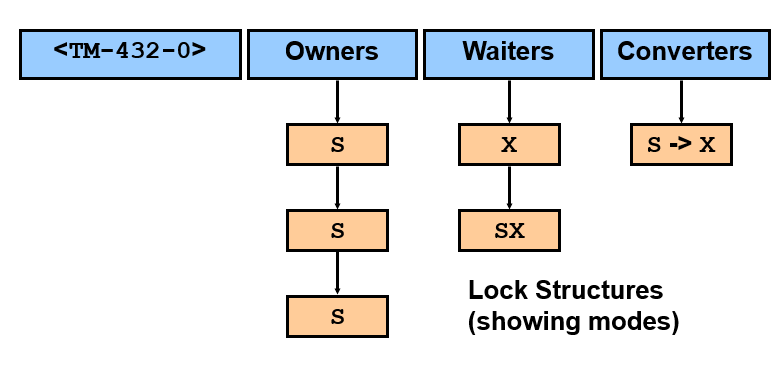
Enqueues只能是被会话获取、转换和释放！

Enqueues are acquired, converted, and released by sessions (not processes).

任何时刻都有可能有一些会话持有一个锁，而另一些会话需要等待acquire或者convert一个锁。

对于每一个Enqueue都有一个资源结构（resource structure）--- ksqrs，用来维护一个记录owners、waiters和converters的列表。

每个owner、waiter和converter都有一个锁结构（lock structure）--- ksqlk。



## Client and Managed Enqueues

enqueue可以分为俩种类型：

1. Client Enqueues：  
   Client layer分配的锁结构，比如DML锁（TM）
2. Managed Enqueues：  
   KSQ自己分配的锁结构，比如MR锁-- media recovery enqueues

## State Objects

在一个进程失败时，为了让PMON可以对锁进行恢复，所有的锁结构都被嵌在一个state ojbect中。

Client lock structures被嵌在自定义的状态对象（customized state objects）中，比如：

1. DML lock (TM) structures are embedded in a DML lock state object.  
   TM锁结构被嵌在DML lock SO中。
2. Transaction (TX) lock structures are embedded in a transaction state object.  
   TX锁结构被嵌在transaction SO中

Managed lock structures被嵌在enqueue state objects中。

## Hashing and Latching

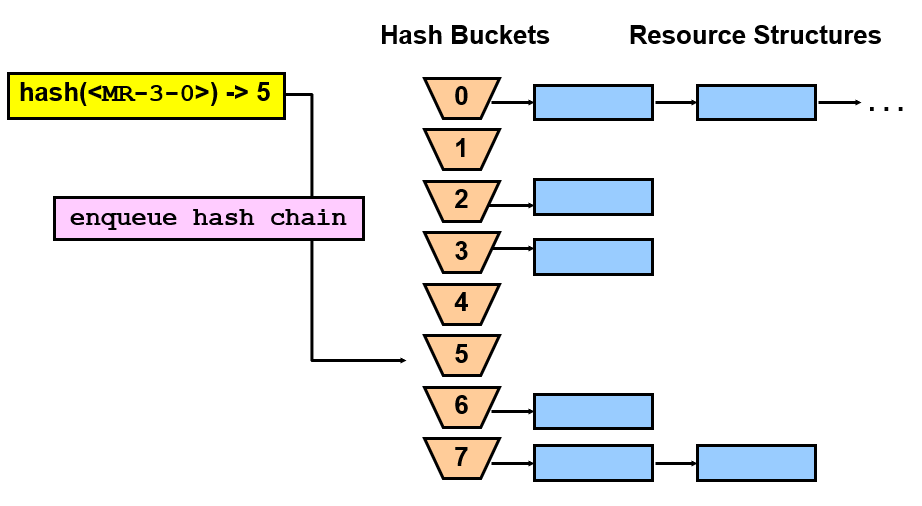
所有resource structures构成了resource table的一部分。

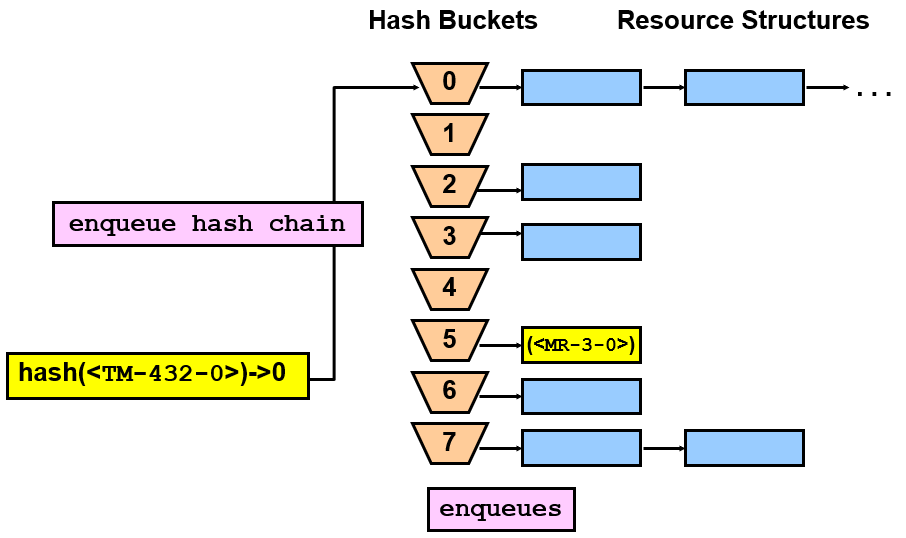
在resource table中找resource用的是hash算法。

使用resource name（<Type-ID1-ID2>）进行hash算法，来产生一个由hash buckets构成的hash table（ksqht）。

每个hash bucket包含一个resource组成的链表，就是我们熟知的hash chain。

hash chain被一个或多个以round-robin方式分配的latches（enqueue hash chain latch的子latch）保护着。





我们看到一个MR锁资源，在enqueue hash chain latch的保护下，放到了5号bucket的hash chain中。

## Resource Free List

resource free list是一个专门放unused resource的链表，为的就是资源的重新利用。

这个链表上的操作都是由enqueue latch来保护的。

## 初始化参数（都是老参数，有待进一步验证）

ENQUEUE\_RESOURCES: resource structures（10.2以后被弃用）

\_ENQUEUE\_LOCKS: enqueue lock structures

\_ENQUEUE\_HASH: size of hash table for resource structures

\_ENQUEUE\_HASH\_CHAIN\_LATCHES: number of latches protecting the hash chains of the resource structure

## 锁获取

Lock acquisition就是一个会话要获取一个自己没有持有的锁。

一个会话可以获得一个锁的条件是：

1. There are no converters or waiters ahead of it.
2. 要获取的锁的模式和现在其他会话持有该锁的模式兼容。

获取步骤：

1. 定位一个资源结构（Locate the resource structure）
2. 获得一个free lock structure
3. 将该锁加于请求的资源之上（Populate it for the resource that is requested）
4. 将锁结构link到第一个定位的资源结构上（Link the lock structure to the resource structure）

## 锁的转换

Lock Conversion就是一个会话要将当前持有的锁的模式进行转换。

一个会话转换自己持有的锁的模式的条件是：

1. 被转换的模式是当前持有模式的子集。
2. 转换后的模式必须与当前其他会话持有的锁的模式兼容。

## 锁的释放

Lock Release不需要条件，想啥时候释放就啥时候释放。

Release操作步骤如下：

1. 定位对应的resource structure
2. 把对应的lock structure从resource上unlink下来。
3. 然后再把这个lock structure link到lock free list。
4. 如果此时有waiter或者converter等待，就通知它们，锁已经释放了（Post the next converter or waiter (if any).）

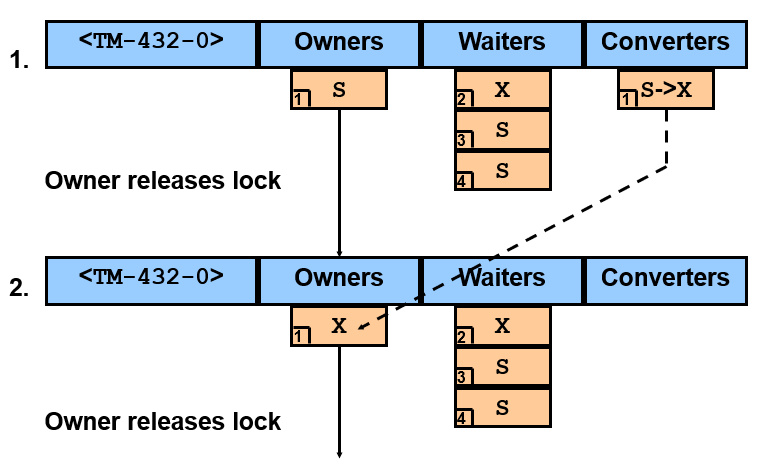
## Enqueue Operation

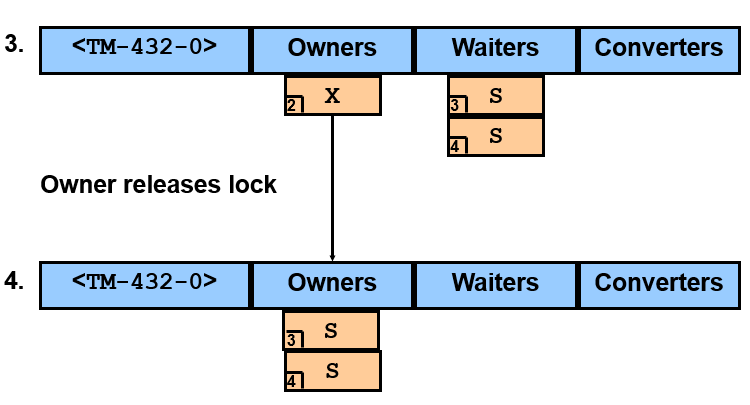
当一个新锁被请求时，会检查队列来进行确认：请求是否可以被授权（grant）。

如果一个锁被释放，那么oracle会再次检查converters和waiters，看看它们能不能获得锁。

先处理转换队列，再处理等待队列（The converters queue is processed first,followed by the waiters queue.）

示例如下：





## Enqueue等待事件

The session is waiting for a local enqueue.

The wait depends on the name of the enqueue (see [Oracle Enqueue Names](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/oracle-enqueue-names.html#GUID-94B7C247-F152-43EB-9482-35B35EDBA934)).

**Wait Time:**Depends on the enqueue name

| **Parameter** | **Description** |
| --- | --- |
| *name* | See "[name and type](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/descriptions-of-common-wait-event-parameters.html#GUID-82606140-F4F2-4912-823F-8E92370BCC63__I970937)" |
| *mode* | See "[mode](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/descriptions-of-common-wait-event-parameters.html#GUID-82606140-F4F2-4912-823F-8E92370BCC63__I970894)" |

我们在v$session、v$session\_wait中可以查看会话的enqueue等待。

下面我们把oracle enqueue names列出来：

BL, Buffer Cache Management

BR, Backup/Restore

CF, Controlfile Transaction

CI, Cross-instance Call Invocation

CU, Bind Enqueue

DF, Datafile

DL, Direct Loader Index Creation

DM, Database Mount

DR, Distributed Recovery Process

DW, SecureFiles

DX, Distributed Transaction

FP, File Object

FS, File Set

HW, High-Water Lock

IN, Instance Number

IR, Instance Recovery

IS, Instance State

IV, Library Cache Invalidation

JI, Enqueue used during AJV snapshot refresh

JQ, Job Queue

KK, Redo Log "Kick"

KP, contention in Oracle Data Pump startup and shutdown processes

KO, Multiple Object Checkpoint

L[A-P], Library Cache Lock

LS, Log Start or Switch

MM, Mount Definition

MR, Media Recovery

N[A-Z], Library Cache Pin

PE, ALTER SYSTEM SET PARAMETER = VALUE

PF, Password File

PI, Parallel Slaves

PR, Process Startup

PS, Parallel Slave Synchronization

Q[A-Z], Row Cache

RO, Object Reuse

RT, Redo Thread

RW, Row Wait

SC, System Change Number

SM, SMON

SN, Sequence Number

SQ, Sequence Number Enqueue

SR, Synchronized Replication

SS, Sort Segment

ST, Space Management Transaction

SV, Sequence Number Value

TA, Transaction Recovery

TC, Thread Checkpoint

TE, Extend Table

TM, DML Enqueue

TO, Temporary Table Object Enqueue

TS, Temporary Segment (also TableSpace)

TT, Temporary Table

TX, Transaction

UL, User-defined Locks

UN, User Name

US, Undo Segment, Serialization

WL, Being Written Redo Log

XA, Instance Attribute Lock

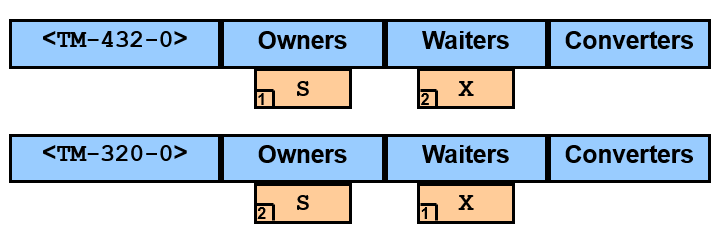
XI, Instance Registration Lock

ZA, Exclusive Lock When Moving Audit Table

## Deadlock Detection（死锁探测）

如果经过一段时间等待，一个锁仍没有获取到或者进行转换，那么Oracle就会调用ksqded()来进行死锁检查。

该函数从请求锁的会话A开始，扫描所有的队列来找到阻塞该会话的阻塞源会话B，然后再从这个阻塞源会话B递归调用ksqded()，说白了就是在接着调用该函数继续扫描B会话所在的队列，查找B会话的阻塞源，如果这时候又扫描到了A，那就说明发生了死锁。



## Diagnostic

### State Object Dump

我们前面说过，锁结构是嵌在state object中的，所以lock structure可以在system或者process的state dump中找到。

enqueque的名称会同持有的模式（mode）以及等待的模式（req）一同显示。

这样就可以指示出它在到底在哪一个队列中：

1. 如果是Owner，就只有mode
2. 如果是Waiter，就只有req
3. 如果是Converter，就同时又mode和req。

Information in a State Object Dump offers significant insight on enqueues:

Lock structures are embedded within state objects and can be found in system or process state dumps.

The enqueue’s name is shown along with the mode that is held (mode) and the mode that is requested (req).

This indicates which queue it resides in:

1. Owner: mode only
2. Waiter: req only
3. Converter: mode and req

### Enqueues Dump

可以使用oradebug或者如下的命令进行enqueue dump：

alter session set events 'immediate trace name enqueues level 2';

This trace allows the following levels：

1. Level 1: dump a summary of active resources and locks, the resource free list, and the hash table
2. Level >= 2: include a dump of resource structs
3. Level >= 3: include a dump of lock structs

### 相关视图

#### V$LOCK

V$LOCK保存的是当前系统中正在被持有的lock structure。

可以使用TYPE，ID1，ID2可以关联V$RESOURCE

LMODE和REQUEST列可以用来找出会话正在等待的队列：

1. LMODE > 0, REQUEST = 0 owner
2. LMODE = 0, REQUEST > 0 waiter (acquirer)
3. LMODE > 0, REQUEST > 0 converter

V$LOCK contains a list of all the lock structures that are currently held in the system.

TYPE, ID1, and ID2, can be used to join with V$RESOURCE.

LMODE and REQUEST indicate which queue the session is waiting for, as follows:

LMODE > 0, REQUEST = 0 owner

LMODE = 0, REQUEST > 0 waiter (acquirer)

LMODE > 0, REQUEST > 0 converter

| **Column** | **Datatype** | **Description** |
| --- | --- | --- |
| ADDR | RAW(4 | 8) | Address of lock state object |
| KADDR | RAW(4 | 8) | Address of lock |
| SID | NUMBER | Identifier for session holding or acquiring the lock |
| TYPE | VARCHAR2(2) | Type of user or system lock  The locks on the user types are obtained by user applications. Any process that is blocking others is likely to be holding one of these locks. The user type locks are:  TM - DML enqueue  TX - Transaction enqueue  UL - User supplied  The system type locks are listed in [Table 8-1](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/V-LOCK.html#GUID-87D76889-832C-4BFC-B8B0-154A22721781__G1427852). Be aware that not all types of locks are documented. To find a complete list of locks for the current release, query the V$LOCK\_TYPE data dictionary view, described on "[V$LOCK\_TYPE](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/V-LOCK_TYPE.html#GUID-2DCB0E6F-0BA7-4E15-A52B-F8AFB8DF5F90)". |
| ID1 | NUMBER | Lock identifier #1 (depends on type) |
| ID2 | NUMBER | Lock identifier #2 (depends on type) |
| LMODE | NUMBER | Lock mode in which the session holds the lock:   * 0 - none * 1 - null (NULL) * 2 - row-S (SS) * 3 - row-X (SX) * 4 - share (S) * 5 - S/Row-X (SSX) * 6 - exclusive (X) |
| REQUEST | NUMBER | Lock mode in which the process requests the lock:   * 0 - none * 1 - null (NULL) * 2 - row-S (SS) * 3 - row-X (SX) * 4 - share (S) * 5 - S/Row-X (SSX) * 6 - exclusive (X) |
| CTIME | NUMBER | Time since current mode was granted |
| BLOCK | NUMBER | Indicates whether the lock in question is blocking other processes. The possible values are:   * 0 - The lock is not blocking any other processes * 1 - The lock is blocking other processes * 2 - The lock is not blocking any blocked processes on the local node, but it may or may not be blocking processes on remote nodes. This value is used only in Oracle Real Application Clusters (Oracle RAC) configurations (not in single instance configurations). |
| CON\_ID | NUMBER | The ID of the container to which the data pertains. Possible values include:   * 0: This value is used for rows containing data that pertain to the entire CDB. This value is also used for rows in non-CDBs. * 1: This value is used for rows containing data that pertain to only the root * *n*: Where *n* is the applicable container ID for the rows containing data |

#### V$RESOURCE

V$RESOURCE保存着所有当前系统被锁住的资源。

每一行记录都与V$LOCK中的一行或者多行有关联。

V$RESOURCE的基表是X$KSQRS。

V$RESOURCE gives a list of all the resource structures that are currently locked in the system.

Each row is associated with one or more rows in V$LOCK.

V$RESOURCE is based on X$KSQRS.

| **Column** | **Datatype** | **Description** |
| --- | --- | --- |
| ADDR | RAW(4 | 8) | Address of the resource object |
| TYPE | VARCHAR2(2) | Resource type; the resource types are listed in [Table 8-1](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/refrn/V-LOCK.html#GUID-87D76889-832C-4BFC-B8B0-154A22721781__G1427852) |
| ID1 | NUMBER | Resource identifier #1 |
| ID2 | NUMBER | Resource identifier #2 |
| CON\_ID | NUMBER | The ID of the container to which the data pertains. Possible values include:   * 0: This value is used for rows containing data that pertain to the entire CDB. This value is also used for rows in non-CDBs. * 1: This value is used for rows containing data that pertain to only the root * *n*: Where *n* is the applicable container ID for the rows containing data |

#### V$LOCKED\_OBJECTS

V$LOCKED\_OBJECT展示当前系统中所有对象上的锁，说白了就是TM锁。

可以提供如下信息：

1. 跟事务相关的undo段。
2. 操作系统对应的用户名

V$LOCKED\_OBJECT shows all the locks on objects (TM locks) in the system.

Provides the following cross-reference information:

1. Rollback segment associated with the transaction that is run by the locking session (if any)
2. Username at the operating system level
3. Process ID

#### 实验

我们分别在俩个会话对t1表的不同行进行修改：

会话1：

SQL> update t1 set name='a' where id=1;

1 row updated.

SQL> select sid,spid from v$session s,v$process p where s.paddr=p.addr and sid=(select sid from v$mystat where rownum=1);

SID SPID

---------- ------------------------

378 48662

会话2：

SQL> update t1 set name='b' where id=2;

1 row updated.

SQL> select sid,spid from v$session s,v$process p where s.paddr=p.addr and sid=(select sid from v$mystat where rownum=1) ;

SID SPID

---------- ------------------------

254 48761

然后我们来查v$locked\_object和v$lock：

SQL> select \* from v$locked\_object;

XIDUSN XIDSLOT XIDSQN OBJECT\_ID SESSION\_ID ORACLE\_USERNAME OS\_USER\_NAME PROCESS LOCKED\_MODE CON\_ID

---------- ---------- ---------- ---------- ---------- --------------- ------------- -------- ----------- ----------

5 24 2013 78156 254 YX oracle 48760 3 0

2 12 1991 78156 378 YX oracle 48661 3 0

SQL> select \* from v$lock where sid in(254,378);

ADDR KADDR SID TY ID1 ID2 LMODE REQUEST CTIME BLOCK CON\_ID

---------------- ---------------- ---------- -- ---------- ---------- ---------- ---------- ---------- ---------- ----------

00007F8BE6DA1350 00007F8BE6DA13C0 378 TM 78156 0 3 0 1887 0 0

00007F8BE6DA1350 00007F8BE6DA13C0 254 TM 78156 0 3 0 1797 0 0

000000007EB1BCB0 000000007EB1BD30 378 AE 133 0 4 0 1923 0 0

000000007EB28C90 000000007EB28D10 254 AE 133 0 4 0 1838 0 0

0000000078DEC2F8 0000000078DEC380 254 TX 327704 2013 6 0 1797 0 0

0000000078DA1D88 0000000078DA1E10 378 TX 131084 1991 6 0 1887 0 0

6 rows selected.

SQL> select r.\* from v$resource r,v$lock l where r.type=l.type and r.id1=l.id1 and r.id2=l.id2 and l.sid in(254,378);

ADDR TY ID1 ID2 CON\_ID

---------------- -- ---------- ---------- ----------

000000007EE29530 TM 78156 0 0

000000007EE29530 TM 78156 0 0

000000007EE30F88 AE 133 0 0

000000007EE30F88 AE 133 0 0

000000007EE84EA8 TX 327704 2013 0

000000007EE86860 TX 131084 1991 0

6 rows selected.

我们可以看到，v$lock\_object里面显示的是TM锁（DML锁）的相关信息，其中回滚段的信息，从v$lock中对应的TX锁的ID1和ID2也能求得。

并且我们看到，通过type，ID1，ID2可以在v$resouce找到lock structure对应的resource structure。

### 脚本

The script catblock.sql creates the following views:

DBA\_LOCKS like V$LOCK

DBA\_DML\_LOCKS DML (TM) locks only

DBA\_DDL\_LOCKS DDL locks (not enqueues!)

DBA\_WAITERS waiters and converters

DBA\_BLOCKERS blockers

The script utllockt.sql gives a tree of locks. It requires catblock.sql to have been run firs

# Lock Contention

## Data Dictionary Locks

作用就是：

1. 保护数据字典的完整性。
2. 为事务提供一致性的数据字典中的对象定义。（Provide transactions with a consistent view of object definitions in the data dictionary）

data dictionary lock分3类：

1. row cache locks
2. library cache locks（breakable parse locks）
3. library cache pins

### Row Cache Locks

Row Cache：

1. 存储数据字典的行
2. shared pool的一部分
3. 为了减少对system表空间访问的物理I/O
4. 使用Row Cache Locks来达到锁住数据字典行的细粒度

锁的实现：

1. 被缓存的行就如同前面所说的resource structure（The cached row acts as the resource structure.）
2. Lock structures从shared pool中动态分配
3. Types为QA…QZ

### Wait Event:Row Cache Lock

***文档34609.1对这一部分说的很详细，所以直接摘录下来代替DSI中这一段的描述***。

This event is used to wait for a lock on a data dictionary cache specified by "cache id" (P1).

该等待事件就是等待获取一个P1对应的数据字典缓存的锁。

Parameters:

P1 = cache - ID of the dictionary cache

P2 = mode - Mode held

P3 = request - Mode requested

下面分别来说明各个值得意义：

cache - ID of the dictionary cache

Row cache lock we are waiting for. Note that the actual CACHE# values differ between Oracle versions. The cache can be found using this select - "PARAMETER" is the cache name:

对应的cache可以使用如下的方式查找：

SELECT DISTINCT cache#, type, parameter

FROM v$rowcache

WHERE cache# = &P1

With the release of 11g, v$rowcache may not report all valid cache# values.

If the query above does not return any rows, you can also query:

但是在11g中，v$rowcache中可能查不到所有的cache#，所以还要结合下面的查询：

SELECT DISTINCT cache#, cache\_name

FROM v$rowcache\_parent

WHERE cache# = &P1;

In a RAC environment the row cache locks use global enqueues of type "Q[A-Z]" with the lock id being the hashed object name.

mode - Mode held

The mode the lock is currently held in:

KQRMNULL 0 null mode - not locked

KQRMS 3 share mode

KQRMX 5 exclusive mode

KQRMFAIL 10 fail to acquire instance lock

request - Mode requested

The mode the lock is requested in:

KQRMNULL 0 null mode - not locked

KQRMS 3 share mode

KQRMX 5 exclusive mode

KQRMFAIL 10 fail to acquire instance lock

Wait Time:

In exclusive mode any process other than PMON will timeout after 8 hours (10000 waits of 3 seconds)

In RAC the foreground will wait 60 seconds for the LCK0 to get the lock, the foreground will wait in infinite loop until the lock has been granted (LCK0 will notify foreground).

In either case PMON will wait for only 5 seconds.

If a session times out when waiting for a row cache lock then it will report this to the alert log and tracefile with a message like:

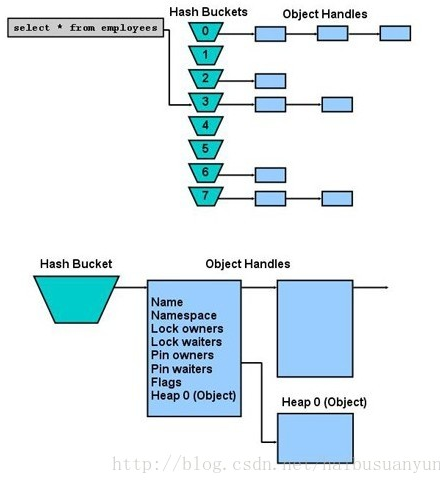
WAITED TOO LONG FOR A ROW CACHE ENQUEUE LOCK

如果是申请模式为exclusive mode，除了PMON以外任何进程都会在8小时后超时（3秒一次，尝试等待10000次---这里DSI中说的是1000次，可能版本不同原因，这个也不好测）。

在RAC中，前台进程会等LCK0去获取对应的row cache lock，等60秒超时，会再进入一次新的等待，就这样无限循环，直到获得申请的锁为止。

无论是哪种情况，PMON如果要获取改锁，5秒钟获取不到就超时。

### Library Cache Locks



This event controls the concurrency between clients of the library cache. It acquires a lock on the object handle so that either:

* 1. One client can prevent other clients from accessing the same object
  2. The client can maintain a dependency for a long time (for example, no other client can change the object)

This lock is also obtained to locate an object in the library cache.

改等待事件是控制对library cache的并发访问的。在一个object handle上上锁是为了：

1. 一个client可以阻止其他clients同时访问相同的库缓存对象
2. client可以持续维护一个依赖一段时间（比如此时其他client就不能修改这个object）

在将库缓存中添加一个库缓存对象时也要获取改lock。

在解析的过程中，该锁以如下方式被获取：

1. parent和child对象句柄上是exclusive mode
2. 所有依赖关系上的该锁是share mode

说白了，就是解析的过程中，要给父子LCO handle都加上X模式的library cache lock。而它们所依赖的，比如表啊、用户啊这些LCO handle，都要加上S模式的library cache lock。

该lock会保持NULL模式，以便于：

1. 发现失效对象（Detect invalidations）
2. Avoid having to locate the handle again

该lock的实现：

1. resource structure就是object handle
2. lock structure从shared pool中动态分配
3. 锁类型是LA…LP。

### Wait Event: Library Cache Lock

该等待事件就是等待获取一个library cache lock。

等待3秒就超时，如果是PMON的话，等待1秒就超时。

The wait parameters are:

P1: object handle address

P2: lock structure address

P3: 100 \* mode requested + namespace#

namespace可以在V$DB\_OBJECT\_CACHE中查到。

该等待事件应该很少才对。

如下解释P3值：

event: 'library cache lock'

time waited: 43 min 12 sec

wait id: 9

p1: 'handle address'=0x7000003117dfca0

p2: 'lock address'=0x700000310866c80

p3: '100\*mode+namespace'=0x4f0003

\* time between wait #1 and #2: 0.000164 sec

<=================p3: '100\*mode+namespace'=0x4f0003

mode=3

namespace=4f

HEX: 4f =>DEC: 79

SQL> select distinct KGLHDNSP,KGLHDNSD from x$kglob where KGLHDNSP=79;

KGLHDNSP KGLHDNSD

---------- --------------------

79 ACCOUNT\_STATUS

### Library Cache Pins

当访问library data heap时会获得Library cache pin。

但是因为要先访问object handle，所以要先获得object handle上的library cache lock。

library cache pin有俩种模式：

1. shared：  
   读取data heap  
   防止依赖的对象被修改
2. exclusive：  
   修改data heap

library cache pin的实现：

1. ojbect handle时source structure
2. pins是从shared pool中分配的
3. types从NA到NZ

### Wait Event: Library Cache Pin

p1、p2、p3跟library cache lock差不多。

该等待事件就是等待获取一个library cache pin。

等待3秒就超时，如果是PMON的话，等待1秒就超时。

The wait parameters are:

P1: object handle address

P2: lock structure address

P3: 100 \* mode requested + namespace#

这种等待比较罕见。

## DML Locks

DML lock主要为了保障如下俩点：

1. Table Lock（表锁）：主要为了在整个事务期间，保证对象的一致性。
2. Row Lock（行锁）：主要为了在整个事务期间，保证数据的一致性。

### DML Table Locks: Implementation

DML Table Locks是通过TM enqueue实现的。

TM enqueue是一个有额外的conversion history table（x$ktadm）的enqueue。

The wait parameters for TM enqueue waits are:

1. P1: name | mode
2. P2: object ID
3. P3: 0

注意P2值是object\_id就可以了，也就是v$lock中的ID1。

### Disabling DML Table Locks

手动取消DML表锁有俩种方式：

1. Setting DML\_LOCKS to zero
2. Using the ALTER TABLE command to disable table locks

减少DML锁有一点需要特别注意就是减少外键索引的维护。

### DML Row Locks

该锁的实现是通过行锁和事务锁共同实现的：

1. Row-level locks
2. Transaction locks

#### Row-level locks

行锁的实现是通过：

1. Lock bytes in each row header（lb那个字节）
2. Interested transaction lists (ITLs) in each data or index block.

也就是说DML Row Locks在数据块级别可能存在俩种争用等待：

1. ITL的争用
2. 行锁的争用

#### Transaction Locks

Transaction Locks与活动事务相关，这些活动事务被列在undo段头块的TX table中，而ITL槽中的XID就是指向TX table中对应的事务的。

事务锁是通过TX enqueue实现的。

The wait parameters for TX enqueue waits are:

1. P1: name | mode
2. P2: rbs# | wrap#
3. P3: slot#

这个我们也可以从v$lock中TX enqueue的ID1和ID2得到，ID1和ID2组起来就是个XID。

ID1以十进制数值表示该事务所占用的回滚段号和事务槽slot number号,其组形式：

0xRRRRSSSS,RRRR=RBS/UNDO NUMBER,SSSS=SLOT NUMBER

ID2 以十进制数值表示环绕wrap的次数，即事务槽被重用的次数

#### 找到哪一行被锁了

TX enqueue的等待并不会告诉你哪一行被锁了，V$SESSION的如下列可以找到被锁的行：

1. ROW\_WAIT\_OBJ#
2. ROW\_WAIT\_FILE#
3. ROW\_WAIT\_BLOCK#
4. ROW\_WAIT\_ROW#

实验如下：

第一个会话：

SQL> select sid from v$mystat where rownum=1;

SID

----------

619

SQL> @yx/rowid

Enter value for owner: yx

Enter value for tname: t1

Enter value for predicate:

OBJ# FILE# BLOCK# ROW#

---------- ---------- ---------- ----------

78347 5 94 0

78347 5 95 0

78347 5 95 1

SQL> update t1 set id=6 where id in(1,3);

2 rows updated.

第二个会话：

SQL> @mysp

SID SPID

---------- ------------------------

255 61196

SQL> update t1 set id=8 where id=3;

观察会话：

SQL> select sid,ROW\_WAIT\_OBJ#,ROW\_WAIT\_FILE#,ROW\_WAIT\_BLOCK#,ROW\_WAIT\_ROW# from v$session where sid in (619,255);

SID ROW\_WAIT\_OBJ# ROW\_WAIT\_FILE# ROW\_WAIT\_BLOCK# ROW\_WAIT\_ROW#

---------- ------------- -------------- --------------- -------------

255 78347 5 95 1

619 -1 0 0 0

### Buffer Locks

buffer locks的作用是保护block在buffer cache中的完整性。

锁的实现：

1. buffer header作为resource structure。
2. buffer handles作为lock structure
3. 会话通过buffer handles访问buffer
4. buffer handles是从shared pool中自动分配的

#### Wait Event: Buffer Busy Waits

Wait until a buffer becomes available.

There are four reasons that a session cannot pin a buffer in the buffer cache, and a separate wait event exists for each reason：

1. "buffer busy waits": A session cannot pin the buffer in the buffer cache because another session has the buffer pinned.
2. "read by other session": A session cannot pin the buffer in the buffer cache because another session is reading the buffer from disk.
3. "gc buffer busy acquire": A session cannot pin the buffer in the buffer cache because another session is reading the buffer from the cache of another instance.
4. "gc buffer busy release": A session cannot pin the buffer in the buffer cache because another session on another instance is taking the buffer from this cache into its own cache so it can pin it.

Wait Time: Normal wait time is 1 second. If the session was waiting for a buffer during the last wait, then the next wait will be 3 seconds.

第一次等待是1秒钟，之后是3秒钟。

三个参数：

1. P1：绝对文件号
2. P2：块号
3. P3：class#：就是块的内容是啥类型，1代表data block，2代表sort block。

如下查询可以得到所有的class#，就是查GV$WAITSTAT的视图定义：

SQL> SELECT view\_definition FROM V\_$FIXED\_VIEW\_DEFINITION WHERE view\_name='GV$WAITSTAT';

VIEW\_DEFINITION

--------------------------------------------------------------------------------

select inst\_id,

decode(mod(indx,19),

1,'data block',

2,'sort block',

3,'save undo block',

4,'segment header',

5,'save undo header',

6,'free list',

7,'extent map',   
8,'1st level bmb',

9,'2nd level bmb',

10,'3rd level bmb',

11,'bitmap block',

12,'bitmap index block',

13,'file header block',

14,'unused',

15,'system undo header',

16,'system undo block',

17,'undo header',

18,'undo block'

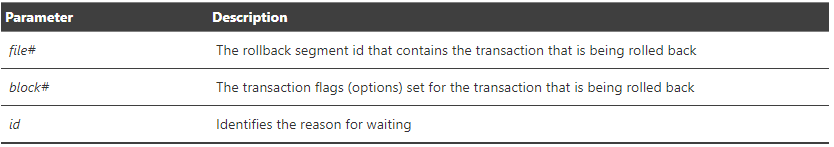
),

count,time, con\_id from x$kcbwait where mod(indx,19)!=0

#### Wait Event: Write Complete Waits

The session waits for a buffer to be written. The write is caused by normal aging or by a cross-instance call.

等待时间为1秒。



#### Buffer Lock Contention

这个主要就查V$WAITSTAT，类型就是上面说的那18种类型。

#### Data Block Contention

如果是读等待，那就建立合适的索引。

如果是写等待：

1. 减少行密度（row density）:  
   加大PCTFREE  
   通过减小RECORDS\_PER\_BLOCK来减少每个block中的行数  
   减小db\_block\_size，言外之意，16k block比8k block更容易发生写等待
2. 避免“right-hand” indexes，就是避免单调递增索引列。

#### Undo Segment Contention

undo段的争用就分为undo段头和undo block的争用。

undo段头争用说明需要增加undo段了，其实就是加undo表空间。

system undo header和system undo block的争用比较罕见，说明system RBU紧张了。

如果发现undo段头争用，使用下面查询确定ratio，如果ratio大于5%，那说明该加undo表空间了。

SQL> SELECT sum(waits)\* 100 /sum(gets) "Ratio",sum(waits) "Waits", sum(gets) "Gets" FROM v$rollstat;

Ratio Waits Gets

---------- ---------- ----------

.018033452 4 22181

#### Index Block Contention

跟data block contention差不多，额外需要关注的是：

1. 索引块分裂（split）
2. 位图索引更新

#### Free List Contention

现在都用ASSM，这个不多说了。

### Sort Locks

有俩种类型：

1. 为永久表空间服务的Temporary table locks (TS)
2. 为临时表空间服务的Sort segment locks (SS)

对sort space usage保持良好的监控。

### ORA-1575

[oracle@host01 yx]$ oerr ora 1575

01575, 00000, "timeout waiting for space management resource"

// \*Cause: failed to acquire necessary resource to do space management.

// \*Action: Retry the operation.

就是等待space management enqueue，为了减少该enqueue等待：

1. 使用locally managed tablespaces
2. 使用临时表空间进行排序操作
3. 增加SORT\_AREA\_SIZE来减少disk sort。
4. 合理规划extent的尺寸，设置PCTINCREASE为0。  
   注意，pctincrease对于dictionary-managed tablespaces意义更大一些，如果设置为0，则接下来的extent大小都跟之前的extent大小一致，默认是50，这是一个百分比。  
   而对于locally managed tablespaces，它只在segment建立时起作用，oracle会根据该值联合initial、next等，综合确定段的初始extents的分配，一旦段建立，这些值就在没作用了。

### SMON Functions

SMON的功能是通过调用ktmmon()实现的。

SMON的主要功能有：

1. 合并空闲的extent
2. 清除临时段
3. 清除obj$中不存在的对象
4. 当创建索引的行为异常终止，清理ind$
5. shrinking undo segments
6. 启动时做实例恢复
7. 被PMON唤醒进行事务回滚

#### Coalescing Free Space

SMON每5分钟就进行一次tablespace空闲空间的合并。

SMON会调用ktsclsb()来合并5组extents。

该函数会合并PCTINCREASE>0的dictionary managed tablespace的extents。

当该行为发生时，会产生ST enqueue。

为了确定SMON是否进行了coalesce的行为，可以使用如下查询：

SQL> select count(1) from dba\_free\_space\_coalesced;

COUNT(1)

----------

5

##### 禁用Background Coalescing

SMON的空间自动合并一般不会造成问题，为了减少这种行为，可以：

1. 使用locally managed tablespaces
2. 把PCTINCREASE设为0

想要彻底禁用，可以设置事件10269。

注意alter tablespace xx shrink space只能对locally managed的temp表空间使用：

SQL> alter tablespace test shrink space;

alter tablespace test shrink space

\*

ERROR at line 1:

ORA-12916: cannot shrink permanent or dictionary managed tablespace

SQL> alter tablespace temp shrink space;

Tablespace altered.

[oracle@host01 yx]$ oerr ora 12916

12916, 00000, "cannot shrink permanent or dictionary managed tablespace"

// \*Cause: An attempt was made to shrink a permanent tablespace or a

// dictionary managed tablespace.

// \*Action: Check the tablespace type and issue the statement only on

// locally managed temporary tablespaces.

#### Temporary Segment Cleanup

SMON每隔2小时进行一次该工作。

SMON调用ktssdt\_segs()来drop掉所有零散的（stray）temporary segments。

如果很多正在使用temporary segment的进程被强行终止，那么该工作就非常有必要了。

每当SMON找到可以被drop的temporary segment，就会产生TS (temporary segment) enqueue。

##### 禁用临时段自动清理

临时段自动清理正常不会构成问题。

想要减少该工作，需要注意：

1. 使用临时表空间的sort segment
2. 不要随意终止进程。  
   这一点可以实验，kill一个正在进行排序操作的会话，被占用临时段并不会立即释放。

该行为可以通过设置10061事件完全禁用。

# Latch Internals

## Latch的作用

序列化访问：

1. 保护SGA中的数据结构
2. 保护共享内存的分配

序列化执行：

1. 防止对一些关键代码的同时执行。
2. 防止损坏

# Memory Management

## System Global Area

The SGA consists of:

1. Fixed area
2. Variable area
3. Database buffer cache
4. Redo log buffer
5. Shared pool

### Dynamic SGA Memory Allocation

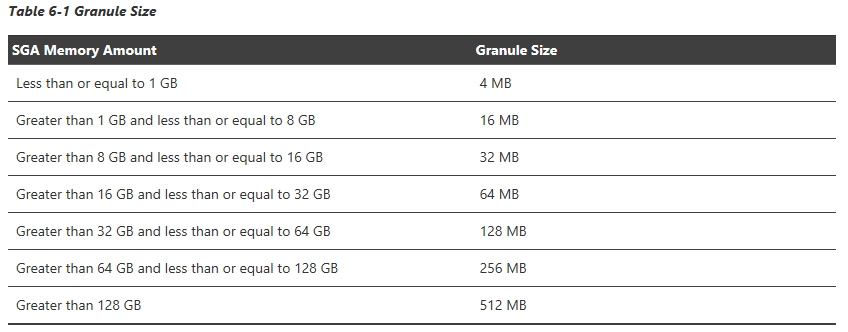
SGA动态分配最大分配到sga\_max\_size，并且SGA中的每一个组件的大小都是memory page的整数倍。

可以调整的系统参数有：DB\_CACHE\_SIZE、LOG\_BUFFER、SHARED\_POOL\_SIZE、LARGE\_POOL\_SIZE。

### Granule Size

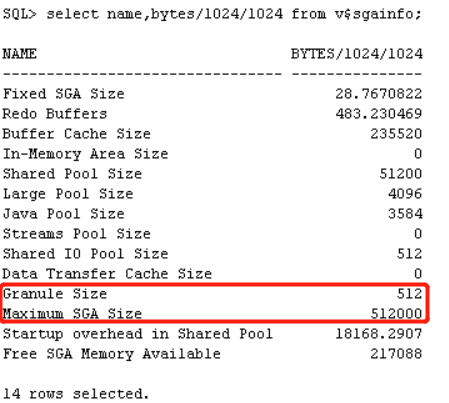
SGA中为各个组件分配空间的最小单位是granule，说白了就是如果granule size为16m，那么我们设置某一个组件为20m，则oracle会给该组件分配32m。

granule的大小取决于sga\_max\_size的设置，对应关系如下：



如果sga\_max\_size没有设置，那么granule size将取决于各个组件的初始值的总和。

可以从v$sga\_info中查看到granule size：



### SGA与大页

其实是SGA和shared memory，linux中就是大页。因为在linux中，强烈建议配置大页。所以这里就只说SGA和大页的关系。

配置大页的好处如下：

1. 可以让数据库运行起来更高效（function more efficiently）。
2. 进程使用大页访问数据的时候可以大大减少换页（without excessive paging）。
3. 只有oracle进程可以访问这块内存。

注意如果，使用shared memory，那么SGA的大小必须小于等于（fit into）设置的shared memory的大小。

### Fixed Area

网上说：可以把这个区想成是 SGA中的“自启”区，Oracle在内部要使用这个区来找到SGA的其他区。换一句话，就是在这个内存里面存有其他区的地址，我们可以通过访问这个区来查找到其他区的位置！

DSI的说法：

The fixed area contains:

1. Many small variables of fixed size
2. Pointers to large or variable size variables

fixed area包括许多固定大小的小变量，以及指向大的或可变的变量的指针。

所以结合网上的说，网上的说法没毛病。

The table X$KSMFSV provides the name, type, size, and memory address for the variables in this area.

表X$KSMFSV可以查到fixed area中的变量的内存地址、名字、类型、大小等。

Use oradebug to obtain the variable’s value.

使用oradebug可以获取这些变量的值。

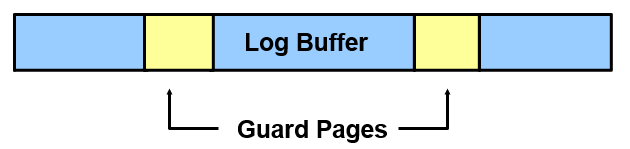
### Buffer Cache

### Redo Buffers

该组件的大小可以通过LOG\_BUFFER来设置。

该区域的目的就是为了防止redo record的遭遇意外损坏。

该区域由俩个guard page进行保护：



### Heaps and Subheaps

内存区域是由heap构成的。

物理上，一个memory heap包括：

1. 一个header或者一个heap描述符（heap descriptor）
2. 一个或多个extent

逻辑上，一个heap由许多子heap构成：

1. subheap的extent由父heap分配
2. 父heap和subheap中都是通过free lists和LRU lists来管理空闲或以使用的空间。

subheap的一些相关细节如下：

1. 减轻父heap的碎片化
2. 经常需要从父heap中临时分配的内存，可以从subheap中直接分配
3. subheap可以附属于某个进程的内存在被访问时，不需要使用latch进行保护
4. subheap由pin和unpin俩种状态。

### Shared Pool

共享池由heap组成，分为俩个部分：

1. permanent memory area：存放data structure，这部分区域的大小取决于很多参数。
2. dynamically managed memory area：存放library和row cache，这部分区域的大小取决于shared\_pool\_size。

shared pool中的内存空间被分为很多chunks：

1. 每个chunk有一个header
2. chunk可以是任何大小（any size）
3. 每一个chunk都可以在X$KSMSP中找到。

#### chunk的allocation class

每个chunk都有如下分类中的一个：

1. Freeable
2. Re-creatable
3. Freeable with a mark
4. Permanent

被使用过的chunk也可以被释放（unallocated）。

SQL> select distinct KSMCHCLS from X$KSMSP where KSMCHCLS not like 'R%';

KSMCHCLS

--------

recr

freeabl

perm

free

#### chunk的分配：free lists

free类型的chunk根据大小，被排列在free lists上。

当有空间请求时，heap manager搜索free lists来找到对应大下的chunk，比如说需求8k的chunk，那就找有没有8k的free类型的chunk。

要是没找到，那么就用现有的跟需求大小最接近的free chunk进行切分。比如找到一个9k的chunk，那么就用这个切出来一个8k。

如果所有的free lists都是空的，那么heap manager就找一个合适的re-creatable chunk来使用。

Re-creatable chunk是被组织在LRU lists上的。

#### chunk的分配：LRU lists

对于inactive的chunk，有俩条LRU list：

1. Transient（瞬时的）：可能不再被需要了
2. Recurrent（周期的）：可能不久后还会被需要

如果没有可用的free chunks，那么就从这俩条链上找chunk来释放。

先从transient list找，找unpinned re-creatable chunks进行释放。

一次要释放8个chunk，才会到另一条list上去找。

当找到足够大小的chunk后，就停止查找。

#### chunk的分配：Hidden Free Memory（9i）

如果之前的策略失败了，还是没找到可用的chunk，那么就会释放Hidden Free Memory，一半的shared pool memory在实例启动的时候被作为Hidden Free Memory。

这部分内存本来是不存在于free list上的，当现有的内存中找不到可用的chunk时，才会使用这部分内存，否则永远不会使用。

这部分内存一旦被使用了，就与shared pool中的其他的内存一样了。不会再变回成hidden free memory。

#### Reserved Pool

保留池就是shared pool的一部分。默认是百分之5的shared pool的大小，但是可以通过参数SHARED\_POOL\_RESERVED\_SIZE进行指定。

保留池是用来保留大chunk的，具体多大以上的chunk会放在保留池，取决于参数\_SHARED\_POOL\_RESERVED\_MIN\_ALLOC。

SQL> show parameter shared\_pool\_reserved

NAME TYPE VALUE

------------------------------------ ----------- ------------------------------

\_shared\_pool\_reserved\_min\_alloc big integer 4400（单位是字节）

\_shared\_pool\_reserved\_pct integer 5

shared\_pool\_reserved\_size big integer 23488102

这部分内存一般用不到。

#### Keeping Objects

如果有些对象就暂时用一下，可能很快就会被刷出共享池。一些对象的重建过程是非常昂贵的（找free chunk，持有各种latch、lock、mutex、pin）。所以如果想让一个对象一直保持在共享池中，可以使用DBMS\_SHARED\_POOL.KEEP。

#### Flushing the Shared Pool

flush共享池是唯一coalesce共享池的方法。

该操作在如下俩种情况下，可以考虑执行：

1. 解决很长时间的shared pool latch contention
2. 减少ORA-4031报错的风险

但是如果执行了该操作，会有如下风险：

1. sequence可能产生gap
2. 如果是没有被keep的object（注意，打开的cursor和被keep的object是不会被flush出共享池的）则需要重建，会造成短暂的shared pool latch contention。

### Large Pool

可以设置LARGE\_POOL\_SIZE来定制large pool的大小。

大池的分配要受shared pool latch的保护。

如果有如下需求或配置，需要使用配置大池：

1. Oracle Shared Server
2. Recovery Manager
3. Parallel Query

网上对应的说法如下：large pool的作用:

1. 为I/O服务进程分配内存
2. 为备份与恢复操作分配内存
3. 为Oracle共享服务器模式与多个数据库间的联机事务分配内存。

通过从large pool中为共享服务器模式分配会话内存，可以减少share pool因频繁为大对象分配和回收内存而产生的碎片。将大的对象从share pool中分离出来，可以提高shared pool的使用效率，使其可以为新的请求提供服务或者根据需要保留现有的数据。

通过视图v$sgastat可以看到large pool当前的使用情况，通过v$sgainfo可以查看大池总的大小。

### Java Pool

如果应用需要使用Java Virtual Machine（JVM），需要配置java池。

可以通过JAVA\_POOL\_SIZE来配置java池的大小。

该区域内存分配需要shared Java pool latch的保护。

网上的说法：

1. 通过JAVA\_POOL\_SIZE参数指定java pool大小。
2. 保存了jvm中特定会话的java code和数据。
3. 在编译数据库中的java代码和使用数据库中的java资源对象时，都会用到java pool。java的类加载程序对每个加载的类会使用大约8K的空间。
4. 系统跟踪运行过程中，动态加载的java类，也会使用到java pool。

## Process Global Area

每个进程都有自己的PGA。

PGA分俩部分：

1. Fixed PGA (data and control files)
2. Variable PGA (UGA and CGA subheaps)

PGA的heap的情况可以在X$KSMPP中进行查询。

## User Global Area

UGA是放session信息的，也分俩部分：

1. Fixed UGA
2. Variable UGA (private SQL and PL/SQL areas, open cursors and links, enabled roles and events, and so on)

当使用dedicated servers时，UGA从PGA中分配。

当使用shared servers时，UGA从SGA中分配。

UGA的heap信息可以在X$KSMUP中查到。

## Call Global Area

## Heap和Subheap的DUMP

Heap的DUMP：

SQL> alter session set events 'immediate trace name heapdump level n';

The level determines which heap/s to dump:

Level 1: PGA

Level 2: SGA

Level 4: UGA

Level 8: Current call (CGA)

Level 16: User call (CGA)

Level 32: Large pool

Subheap的DUMP：

SQL> alter session set events 'immediate trace name heapdump\_addr level n';

level用来指定导出基于哪个地址的subheap，而subheap的地址可以从parent heap的DUMP中得到。