为什么要学习原理呢？

在遇到问题时，理解原理能带来精准的直觉和深刻的洞察。有助于事半功倍的解决问题。

学习原理是DBA的必经之路

先知其然，然后之气所以然。更有助于理解原理。

纸上得来终觉浅，绝知此事要躬行！

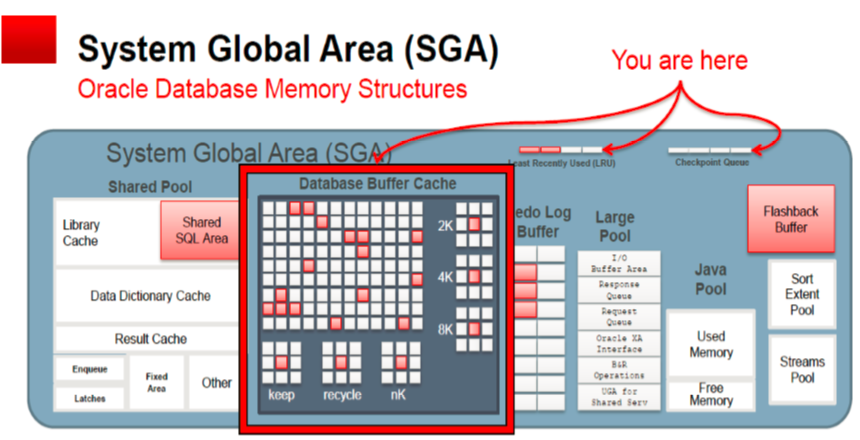
Event waited on                             Times

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

  buffer busy waits                           889

  latch: cache buffers chains                   151

**》1 buffer cache的内存结构！！**



buffer cache 是sga中最大子内存，约占60%-70%

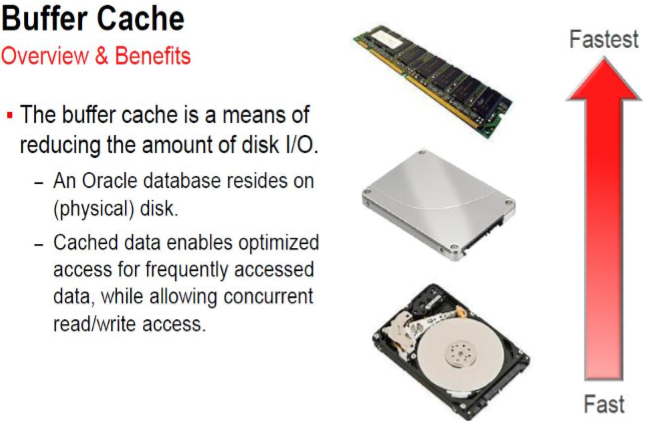
database buffer cache，

least recently used（LRU）--存储在buffer header中

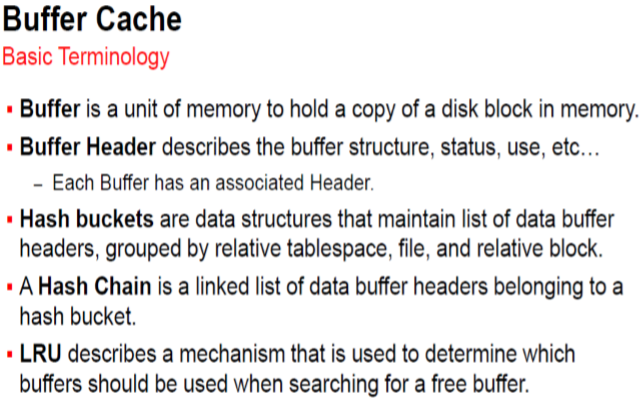
checkpoint queue（CKPTQ）--存储在buffer header中

为什么要有buffer cache？

把block放在内存中操作（i/u/d），解决性能问题！



**》 2 buffer cache technical terms ！！**



理解：

buffer    --类似一个人

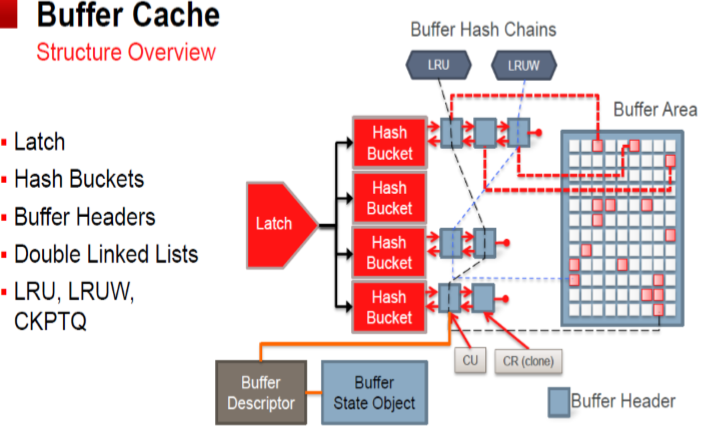
buffer header 对应x$bh   --就是人的身份证

hash buckets  一个数据结构，管理bh的    --类似管理身份证的机构，公安厅户政处

hash chain：一个双向链表（指针）链接buffer header，相邻的bh可以相互指向

LRU：最近最少使用列表， 一个机制，查找空闲buffer时，决定哪些buffer可被使用

**》3 buffer cache structure overview！！**



latch    =》内存锁（lock），保护内存结构

            =》也是一块内存(200byte) ，在shared pool中

            =》获取latch

V$LATCH displays aggregate latch statistics for both parent and child latches, grouped by latch name. Individual parent and child latch statistics are broken down in the views V$LATCH\_PARENT

V$LATCH\_CHILDREN displays statistics about child latches. This view includes all columns of V$LATCH plus the CHILD# column. Note that child latches have the same parent if their LATCH# columns match each other.

latch的数量是一定的，固定的！通过隐含参数查看

SQL> @?/rdbms/admin/show\_para

Enter value for p: hash\_latch

P\_NAME                         P\_DESCRIPTION                             P\_VALUE                  ISDEFAULT ISMODIFIED ISADJ

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

\_cp\_num\_hash\_latches               connection pool number of hash latches          1                       TRUE      FALSE     FALSE

\_db\_block\_hash\_latches               Number of database block hash latches              1024                  TRUE      FALSE     FALSE

查看有关buffer cache的latch类型种类

select addr,latch#,name from v$latch where lower(name) like '%cache buffer%';

cache buffer chain latch （hash chain latch）保护buffer header的latch

对一种latch详细查看，以地址排序

select addr,latch#,name from v$latch\_children where lower(name) like '%cache buffers chains%' order by 1;

查看一个latch占用多少字节！！从上面的排序看，找相邻的两个，后一个减去前一个。所以一个latch大约200byte

select to\_number('77BFFEB8','xxxxxxxxxxxx')-to\_number('77BFFDF0','xxxxxxxxxx') from dual;

('77BFFDF0','XXXXXXXXXX')

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

                      200

查看buffer address的最大最小值。所有的buffer都在这个范围内！对比latch，latch不在此范围内，则latch在share pool or 其他内存中！

select min(ba),max(ba) from x$bh;

buffer header 大约20byte 指向buffer

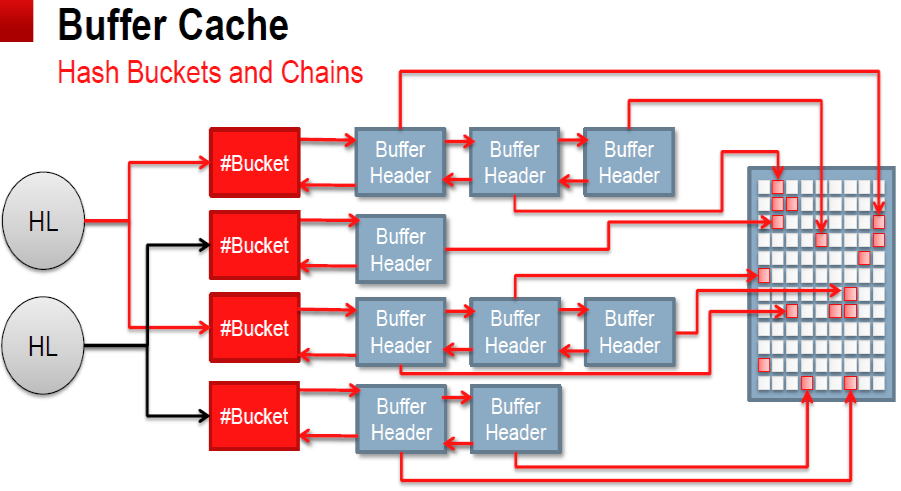
buffer header通过double linked lists=hash chain 指针来相互查找（可以找到上一个和下一个）

LRU,LRUW,CHPTQ，CR（consisttent read）,CU（ cucrrent）,SO（state object）都放在了buffer header中

hash chain latch （概念名）

cache buffers chains（对应数据库中latch 类型名）

\_db\_block\_hash\_latches（对应的隐含参数名）



bucket是buffer header的入口！数值是固定的。oracle 通过隐含参数控制hash bucket的数量。

SQL> @?/rdbms/admin/show\_para

Enter value for p: hash\_bucket

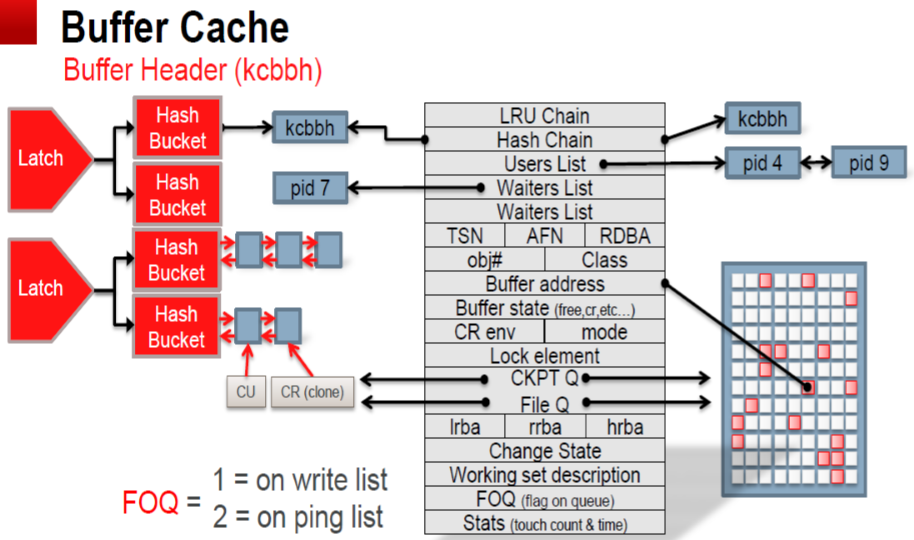
P\_NAME                         P\_DESCRIPTION                             P\_VALUE                  ISDEFAULT ISMODIFIED ISADJ

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

\_db\_block\_hash\_buckets               Number of database block hash buckets              32768                  TRUE      FALSE     FALSE

\_lm\_res\_hash\_bucket               number of resource hash buckets              0                       TRUE      FALSE     FALSE

hash chain： 由一些double linked lists组成！



buffer header=kcbbh（kernel cache buffer block header）所有的元素对应x$bh

working set description： 描述LRU，如何查找free buffer！

SQL> desc x$bh

Name                                                  Null?    Type

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

ADDR                                                           RAW(8)   --buffer header地址

INDX                                                           NUMBER

INST\_ID                                                        NUMBER

HLADDR                                                         RAW(8)    -->latch address

BLSIZ                                                          NUMBER

NXT\_HASH                                                       RAW(8)    -->HASH CHAIN

PRV\_HASH                                                       RAW(8)    -->HASH CHAIN

NXT\_REPL                                                       RAW(8)

PRV\_REPL                                                       RAW(8)    --LRU

FLAG                                                           NUMBER  --buffer标记

FLAG2                                                          NUMBER

LOBID                                                          NUMBER

RFLAG                                                          NUMBER

SFLAG                                                          NUMBER

LRU\_FLAG                                                       NUMBER     --LRU 标记

TS#                                                            NUMBER   -表空间

FILE#                                                          NUMBER     --文件号

DBARFIL                                                        NUMBER    --相对文件号

DBABLK                                                         NUMBER   --块号

CLASS                                                          NUMBER   --类别

STATE                                                          NUMBER

MODE\_HELD                                                      NUMBER

CHANGES                                                        NUMBER

CSTATE                                                         NUMBER

LE\_ADDR                                                        RAW(8)

DIRTY\_QUEUE                                                    NUMBER

SET\_DS                                                         RAW(8)

OBJ                                                            NUMBER   --包含快的段号=dba\_objects.data\_object\_id

BA                                                             RAW(8)    ---buffer address

CR\_SCN\_BAS                                                     NUMBER

CR\_SCN\_WRP                                                     NUMBER

CR\_XID\_USN                                                     NUMBER

CR\_XID\_SLT                                                     NUMBER

CR\_XID\_SQN                                                     NUMBER

CR\_UBA\_FIL                                                     NUMBER

CR\_UBA\_BLK                                                     NUMBER

CR\_UBA\_SEQ                                                     NUMBER

CR\_UBA\_REC                                                     NUMBER

CR\_SFL                                                         NUMBER

CR\_CLS\_BAS                                                     NUMBER

CR\_CLS\_WRP                                                     NUMBER

LRBA\_SEQ                                                       NUMBER

LRBA\_BNO                                                       NUMBER

HSCN\_BAS                                                       NUMBER

HSCN\_WRP                                                       NUMBER

HSUB\_SCN                                                       NUMBER

US\_NXT                                                         RAW(8)

US\_PRV                                                         RAW(8)

WA\_NXT                                                         RAW(8)

WA\_PRV                                                         RAW(8)

OQ\_NXT                                                         RAW(8)

OQ\_PRV                                                         RAW(8)

AQ\_NXT                                                         RAW(8)

AQ\_PRV                                                         RAW(8)

OBJ\_FLAG                                                       NUMBER

TCH                                                            NUMBER    --TCH（touch） 块被读了几次》热块

TIM                                                            NUMBER

CR\_RFCNT                                                       NUMBER

SHR\_RFCNT                                                      NUMBER

SQL> select addr,hladdr,ts#,file#,DBARFIL,DBABLK,obj,ba,tch from x$bh where rownum<10;

ADDR          HLADDR                                TS#       FILE#    DBARFIL     DBABLK          OBJ BA               TCH

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

00007FBB5DE25190 000000007178CF98        0           1      1     73280          250 00000000630CA000        1

00007FBB5DE25190 000000007178CF98        0           1      1     34005           37 0000000062E9E000        1

00007FBB5DE25190 000000007178CF98        0           1      1     90865          421 0000000065F7A000        1

00007FBB5DE25190 000000007178CF98        0           1      1     16653          495 0000000062784000        1

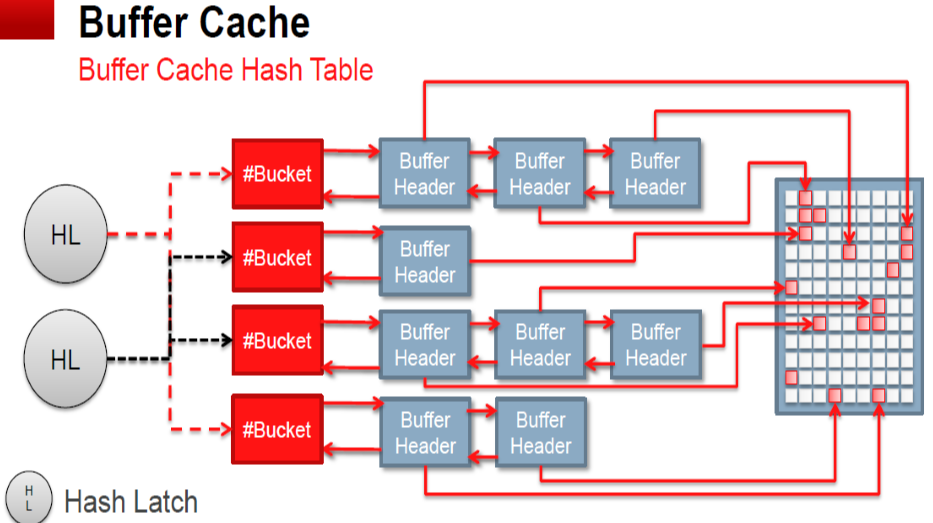
00007FBB5DE25190 000000007178CF98        0           1      1     47252           18 0000000064840000        0

00007FBB5DE25190 000000007178CF98        0           1      1     3872          568 0000000065022000        1

00007FBB5DE25190 000000007178CF98        1           2      2     68476         6339 0000000060D24000        2

00007FBB5DE25048 000000007178CF98        0           1      1     21457          495 00000000622D2000        1

00007FBB5DE25190 000000007178CF98        0           1      1     78317          225 0000000064368000        1



hash table： 通过此表维护hash bucket，hash table里有数组标记的hash bucket

初始值+偏移量=数组

0+1    1

0+2    2

0+3    3

......

=========================================

问题思考：

SQL> select EMPLOYEE\_ID,salary from hr.employees where rownum<3;

EMPLOYEE\_ID    SALARY

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

    198      2600

    199      2600

 这两行数据在物理位置的那个地方呢？？对应在buffer cache的那个位置呢？？

=》

ROWID 查找物理地址！ --参考体系结构

X$BH 查找内存地址！

EG：

select rowid,EMPLOYEE\_ID,salary from hr.employees where rownum<3;

SQL>

ROWID             EMPLOYEE\_ID        SALARY

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

AAASbIAAFAAAADLAAA        198          2600

AAASbIAAFAAAADLAAB        199          2600

SELECT

    ROWID,

    dbms\_rowid.rowid\_object(ROWID) obj,

    dbms\_rowid.rowid\_relative\_fno(ROWID) fno,

    dbms\_rowid.rowid\_block\_number(ROWID) bno,

    dbms\_rowid.rowid\_row\_number(ROWID),

    employee\_id,

    salary

FROM

    hr.employees

WHERE

    employee\_id = 198;

ROWID              OBJ         FNO    BNO DBMS\_ROWID.ROWID\_ROW\_NUMBER(ROWID) EMPLOYEE\_ID     SALARY

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

AAASbIAAFAAAADLAAA    75464           5    203                     0           198     2600

select addr,ba,obj,HLADDR,PRV\_HASH,NXT\_HASH from x$bh where FILE#=5 and DBABLK=203;

ADDR          BA               OBJ HLADDR           PRV\_HASH            NXT\_HASH

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

00007FBB5DD8FED0 00000000625D0000      75464 00000000717DF2B0 00000000717DFAB8 00000000717DFAB8

PRV\_HASH,NXT\_HASH 就是hash chain！！！PRV\_HASH  = NXT\_HASH: 说明这个hash bucket上就只有一个bh

ADDR: BH的地址

BA： buffer的地址

HLADDR：hash latch地址==》CBC LATCH：

hash latch对应的latch名字：

select name from v$latch\_children where addr=(select HLADDR from x$bh where DBARFIL=5 and DBABLK=203);

NAME

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

cache buffers chains

==================================

》buffer cache（pool）

Keep pool

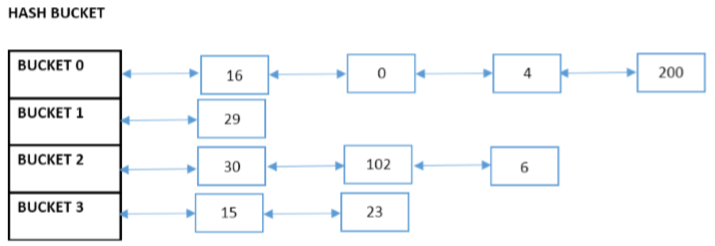
Recycle pool

nK buffer pool

--参考体系结构

====================================

**》 4 逻辑读**



，

在0-2000个数据中，找到1988，怎么找？

一般情况下，则需要全部扫描一遍（全表扫描），从0开始，扫描到1988，找到这个数据

使用索引的情况下，根据范围索引，扫描索引，直接找到1988

Oracle在内存中怎么找到需要的buffer呢？

使用hash算法--类似索引

使用hash算法先把buffer进行hash函数计算（除法散列法），然后将buffer散列（hash）存放到不同的bucket中

理解：

假设hash算法是除余法：对7取余

buffer7/7 余0 将buffer7 放到 bucket 0中

buffer8/7 余1 将buffer8 放到 bucket 1中

buffer9/7 余2 将buffer9 放到 bucket 2中

......

理解：

buffer 就类似球场的座位，根据球场的大小有多少座位都已经提前设计好了

bucket 就类似球场不同看台的入口，找到座位首先要找到看台区域

hash算法，就类似球票，会生成每个座位的位置

这样，去球场看球找座位，不是全场乱找，而是根据球票，按照规则找到座位！

注意：放在bucket上的是bh，为什么不是buffer呢？=》因为一个8k buffer太大，影响读取数据的速度

如果hash计算有冲突（余数一样）则通过buffer chain链表，把数据关联排队挂载到同一bucket下

chain链表不会太长，否则也会影响查询速度 ，因为oracle会定义一个合适的bucket数量（\_db\_block\_hash\_buckets）

==================================

那么，下面这个语句，怎么找到数据的呢？？？--逻辑读

SELECT

    employee\_id,

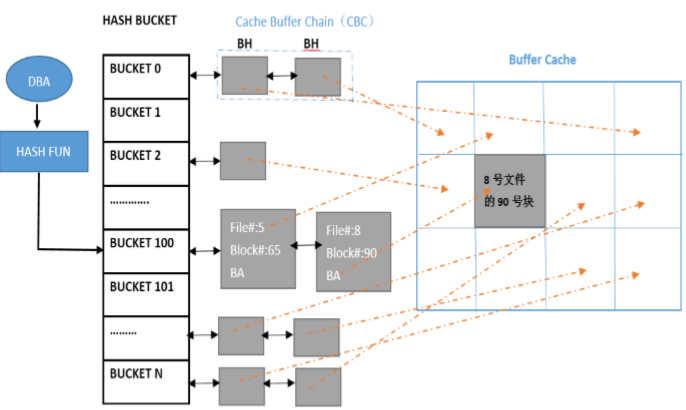
    salary

FROM

    hr.employees

WHERE

    employee\_id = 199;



前提

Oracle会根据DBA计算出buffer放在bucket 100上，如果前面已经有buffer了，则以chain的形式依次挂载

199这行数据有一个ROWID，根据ROWID计算出DBA（data block address ）

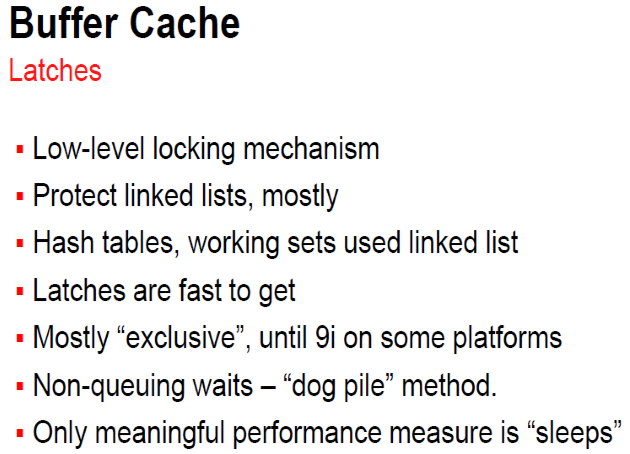
DBA：8,90    =》对block做hash运算 ，得到hash值，到相应的Hash Bucket中搜寻相应Buffer Header

                     =》查看bucket 100上的第一个BH，对比其存储的dba，正确，就查看buffer状态，不正确，则通过chain 找到下一个

                     =》在chain链表上找到需要的buffer，这是逻辑读

                     =》在chain链表上没有找到需要的buffer，需要做物理读

物理读/逻辑读 涉及到latch！！



LATCH: 主要是排他，独占锁。

select 查询的时候chain不能变，否则就找不到对应的数据。LATCH主要作用就是用来保护chain列表的

latch是内存资源，是有限的（如hash latch有隐含参数\_db\_block\_hash\_latches 限制）获取latch的时候会消耗cpu资源，所以oracle 获取latch时，如果不成功会尝试1次，1次之后还是无法获取就sleep

SQL> @?/rdbms/admin/show\_para

Enter value for p: spin

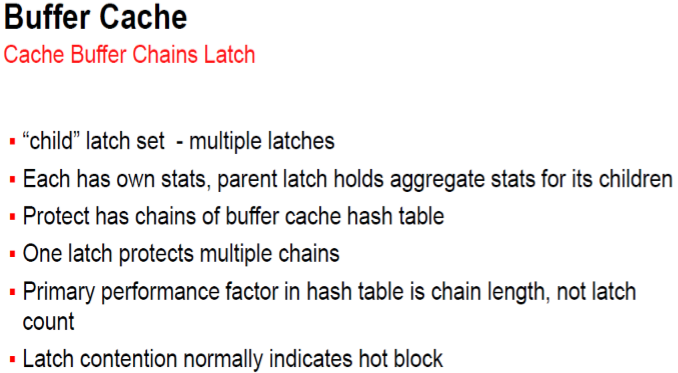
P\_NAME                         P\_DESCRIPTION                             P\_VALUE                  ISDEFAULT ISMODIFIED ISADJ

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

\_mutex\_spin\_count               Mutex spin count                        255                   TRUE      FALSE     FALSE

\_spin\_count                    Amount to spin waiting for a latch              1                       TRUE      FALSE     FALSE

**CBC latch/buffer pin latch**



为什么会有cbc latch？

有这样一种情况：select 的时候定位了bucket和chains，当实际在chain上找BH的时候，因为物理读或buffer被覆盖重用了，这个时候就找不到需要的buffer 了！

目的1：保护bucket 100 上的chain链表的稳定。因为chain上BH可能动态增加（物理读）减少（buffer覆盖重用）

目的2：保护bh，因为要在bh上添加buffer pin

为什么要有buffer pin？

假设没有buffer pin，Oracle访问数据会有如下步骤

1）依据数据块的地址计算出数据块所在的bucket

2）获得保护这个bucket的cbc latch

3）在这个链表上找寻我们需要的数据块

4）读取/修改数据块的内容

5）释放cbc latch

latch自身的获得和释放时间一般都是极短的，但第四步消耗的时间会比较长，latch的释放需要等待

在大量并发环境下，由于cbc latch持有时间过长，导致latch争用容易导致系统的cpu资源出现瓶颈

如何解决这个问题呢？

ORACLE通过让每次访问buffer block的会话获取两次cbc latch，再配合在内存块上加buffer pin来解决这个问题

逻辑读的步骤！

1）依据数据块的地址计算出数据块所在的bucket

2）获得保护这个bucket的cbc latch

3）根据HASH值,搜索CBC链表。找到需要的buffer后，pin这个buffer（读取s，修改x）

4）释放cbc latch

5）读取/修改数据块的内容

6）获取cbc latch

7）unpin这个buffer

8）释放cbc latch

通过这种实现方式，我们看到cbc latch的持有时间大大降低了，因为cbc latch的持有，只做了很少的事情，这样就大大降低了cbc latch的争用。

虽然cbc latch的争用会大大减轻，可是ORACLE只不过是转移了竞争点，现在变成了buffer lock之间的竞争。

这种转移将代价降至最低

如果数据库里读极多，写极少，由于各个读之间的buffer pin是兼容的，都是s模式，因此不会产生任何的争用。

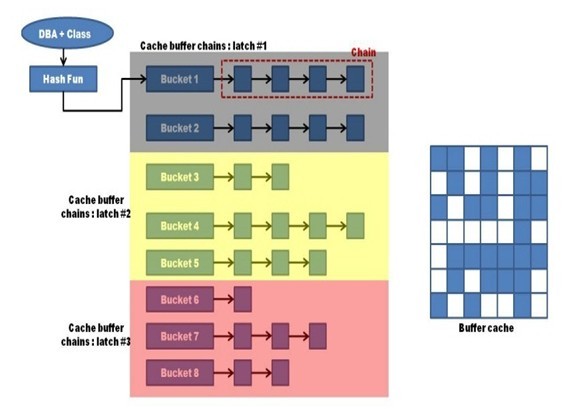
如果数据库里写极多，读极小，就会产生buffer busy waits等待，但是这种等待的代价比cbc latch的等待代价要小的多，latch的spin机制是非常耗cpu的，而buffer pin的管理本质上类似于enq 锁的机制，没有spin机制，不需要自旋耗费大量的cpu。

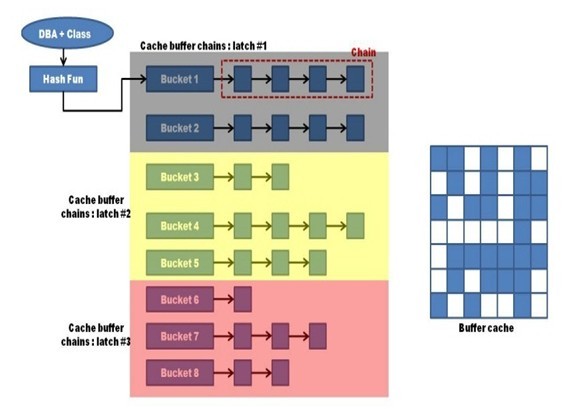
目的：buffer pin latch保护buffer上的数据，防止多个用户同时修改

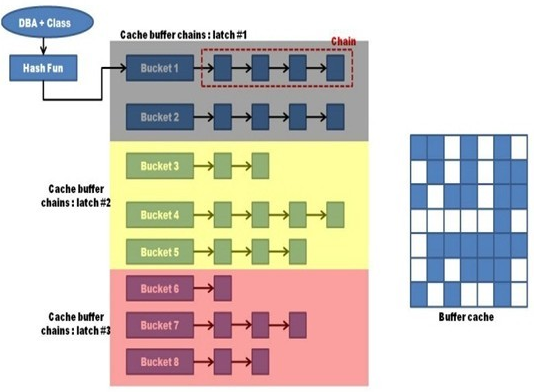
数据库是读写混合的场景，那么写会阻塞读，产生buffer busy waits，但是读不会阻塞写，不会产生这个等待。

===============================================

**》6 CBC LATCH(cache buffer chain latch)等待事件**







在逻辑读（select）的过程中，可能会遇到 cbc latch等待事件（oracle等待事件是ms级别）

如上图所示

一个latch对应多个buket，cbc latch是独占模式

同一时间并发查询

A：读取block（8，90）

B：去读block（10，200）

通过hash，逻辑读需要到不同的bucket上找BH，此时，这两个bucket都是由同一个latch保护，就会产生争用。如果争用较严重，就会产生cbc latch等待事件

可能的原因：

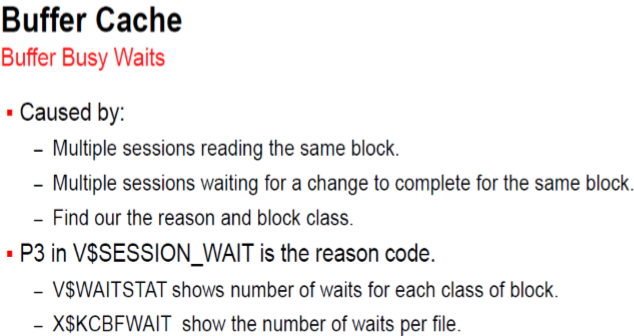
select语句导致！

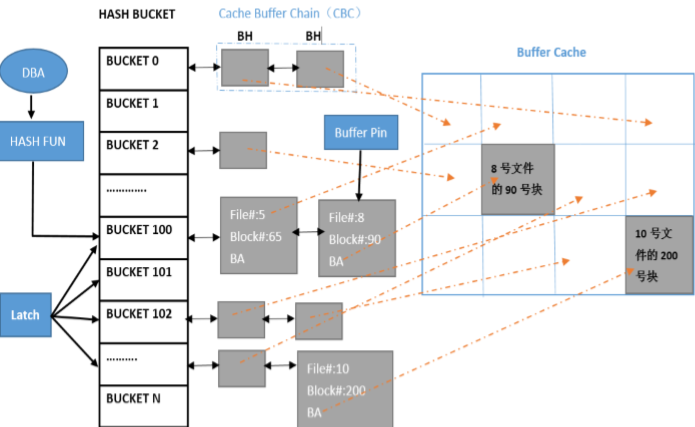
1 全表扫描非常容易引起CBC latch。此时很多用户正在用bucket上的bh，你全表扫描会读取所有的bh，涉及到多个bucket，会有多个hash latch。容易造成cbc latch

2 一致性读CR也会导致CBC LATCH

就是热链（一个latch对应多个bucket），和热快（一个bucket对应多个BH）容易导致cbc latch

**》7 buffer busy wait等待事件**





在逻辑读的过程中，可能会遇到 buffer busy wait等待事件（oracle等待事件是ms级别）

buffer pin 类型 ：0（空闲）、S1（共享）、X2（排他独占）

如上图所示

同一时间并发操作

A：读取block（8，90）    --buffer pin S1

B：读取block（8，90）    --buffer pin S1

A-B读不会阻塞读

C：修改block（8，90）    --构造CR块（clone一个新的buffer，此时，一个block对应多个buffer，将原来的CUR变为CR，将新的块变为XCUR，在新XCUR块上更改，持有buffer pin X2）

B-C读也不会阻塞写

D：读取block（8，90）    --读CUR块，修改动作没有完成》buffer busy wait；修改动作完成了但没提交，依据事务的ACID，继续构造CR读；修改完成且提交，读CUR

C-D写会阻塞读

当然，写会阻塞写

1）当读取的进程发现内存块正在被修改的时候（如果有x模式的buffer pin，就说明正在被修改），它只能等待，它不能clone块，因为这个时候内存块正在变化过程中ing，这个时候clone是不安全的。很多人说，oracle里读写是互相不阻塞的，oracle可以clone内存块，把读写的竞争分开。其实要看情况，在读的时候发现内存块正在被写，是不能够clone的，因为是不安全的。这个时候读的进程只能等待buffer busy waits。

2）当写的进程发现内存块正在被读，这个时候，读是不阻塞写的，因为ORACLE可以很容易的clone出一个CR数据块，然后在clone的块上进行写，这个时候clone是安全的，因为读内存块的进程不会去修改数据块，保证了clone的安全性。

每个buffer header上都有2个列表：users list和waiter list。

users list用来记录，当前有哪些会话获得了此buffer block上的buffer pin,并记录下buffer pin的模式。

waiter list用来记录，当前有哪些会话在等待buffer block 上的buffer pin，并记录下申请buffer pin的模式。

总结：

1）buffer busy waits是产生在buffer block上的等待，由于n个进程想以不兼容的模式获得buffer block的buffer pin，进而引起buffer busy waits等待。

2）buffer lock的管理模式非常类似enq锁的管理模式，先进先出，有队列去记录锁的拥有者和等待着。

3）写写，读写都会产生buffer busy wiats等待。写写的两个会话，都会产生buffer busy wiaits等待，而读写的两个会话，只有读的session会产生，因为它不能去简单的clone一个内存块，正在发生写的内存块发生克隆是不安全的

4）oracle为了解决cbc latch持有时间过长的问题，以每次访问buffer block的会话获取两次cbc latch，再配合在内存块上加buffer pin来解决这个问题。

可能的原因及对策

多个session同时读写相同的数据块，尽量减少update修改操作

可加大 log\_buffer有助于减少buffer busy wait

所有软件都不会完美，在内存和争用间做一个权衡，牺牲争用保留内存是最好的选择

======================================

Eg：buffer busy wait

SELECT

    ROWID,

    dbms\_rowid.rowid\_relative\_fno(ROWID) fno,

    dbms\_rowid.rowid\_block\_number(ROWID) bno,

    employee\_id

FROM

    hr.employees

WHERE

    employee\_id > 190;

ROWID              FNO         BNO EMPLOYEE\_ID

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

AAASbIAAFAAAADPABb        5         207     191

AAASbIAAFAAAADPABc        5         207     192

AAASbIAAFAAAADPABd        5         207     193

AAASbIAAFAAAADPABe        5         207     194

AAASbIAAFAAAADPABf        5         207     195

AAASbIAAFAAAADPABg        5         207     196

AAASbIAAFAAAADPABh        5         207     197

AAASbIAAFAAAADLAAA        5         203     198

AAASbIAAFAAAADLAAB        5         203     199

AAASbIAAFAAAADLAAC        5         203     200

AAASbIAAFAAAADLAAD        5         203     201

AAASbIAAFAAAADLAAE        5         203     202

AAASbIAAFAAAADLAAF        5         203     203

AAASbIAAFAAAADLAAG        5         203     204

AAASbIAAFAAAADLAAH        5         203     205

AAASbIAAFAAAADLAAI        5         203     206

16 rows selected.

session1: hr 不断读取

select userenv('sid') from dual;

35

declare

vid number;

begin

    for i in 1..5000000 loop

    select employee\_id into vid from hr.employees where rowid='AAASbIAAFAAAADLAAC';

end loop;

end;

/

session2: system 不断更改500000次，大事务

select userenv('sid') from dual;

30

declare

vid number;

begin

    for i in 1..500000 loop

    update hr.employees set first\_name='jason'  where rowid='AAASbIAAFAAAADLAAC';

end loop;

commit;

end;

/

declare

vid number;

begin

    for i in 1..5000000 loop

    update hr.employees set first\_name='jason'  where employee\_id=200;

end loop;

commit;

end;

/

session3：sys查看等待事件

select event,count(1) from v$session where wait\_class#<>6 group by event;

select sid,event,total\_waits,TIME\_WAITED from v$session\_event where wait\_class<>'idle' and event like '%busy%';

如果session2 执行是每50次提交一次，小事物有可能不会阻塞select

declare

vid number;

begin

    for i in 1..5000 loop

    update test\_bbwait set first\_name='jason'  where id=469;

    if mod(i,50)=0 then

       commit;

    end if;

end loop;

commit;

end;

/

Eg：CBC latch

declare

vid number;

begin

    for i in 1..5000000 loop

    select count(\*) into vid from hr.employees;

end loop;

end;

/

declare

vid number;

begin

    for i in 1..5000000 loop

    select count(\*) into vid from hr.employees;

end loop;

end;

/

select event,count(1) from v$session where wait\_class#<>6 group by event;

select sid,event,total\_waits,TIME\_WAITED from v$session\_event where wait\_class<>'idle' and event like '%chain%';

###############################################

其他

\_db\_block\_hash\_latches定义latch数量

\_db\_block\_hash\_buckets 定义bucket数量

\_db\_block\_buffers 定义buffers的数量（通常buffer多余buckets，如果少于，则是因为内存太小）

select count(\*) from x$bh; ==》实际使用的buffer数量（一个buffer对应一个bh）

以上可以看出latch，bucket，buffer header，buffer的数量！！数据库的 latch和bucket都是通过隐含参数确定好的

一个latch对应多个bucket，一个bucket上有多个BH，即有很多的object

为什么不是hash latch=hash bucket？（这样就不会有cbc latch 等待了）==》latch是资源（cpu+mem），资源消耗大。

为什么BH<BUFFER  =>只有被使用的buffer才会有BH   ==》buffer的数量是隐含参数设置的上限值！

select inst\_id,event,p2text,p2,p1text,p1,p3text,p3,count(1) from gv$session where wait\_class#<>6 group by inst\_id,event,p2text,p2,p1text,p1,p3text,p3;

创建环境数据

conn jason/jason@localhost:1521/pdbtest

create table test\_bbw( id int,name varchar2(22));

begin

   for i in 1..100

   loop

      insert into test\_bbw values (i,'xsj'||i);

   end loop;

   commit;

end;

/

CR-CONSISTENT READ

CUR-CURRENT

|  |
| --- |
| > 为啥不直接构造新的CR块，而是被阻塞，也就是里边描述的Buffer busy waits。    The first CR is immediately created without being blocked (because the original buffer only has a shared pin). The page you're reading is talking about this: right after creating the CR, the buffer pin is still in exclusive mode in this CR. Maybe the CR hasn't been applied with necessary undo. (After all, there must be a reason for a CR to be created.) If another CR is needed, the new session can wait till the necessary work on the first CR to complete and then can change its state from XCUR to S. In short, if the buffer pin is in X mode, the reader of the buffer must wait.    > 这里的CR块不是通过undo段来构造的，而是通过buffer clone构造的。那么，一般来说通过undo段来构造的CR块是不是也是clone一个buffer，在根据undo里边的前镜像对这个新clone回滚构成一个新的CR块，还是说根据undo里边的前镜像和当前不满足一致性读的这个buffer一起构成的新的CR块。    It's the former. A CR is cloned first, followed by rolling back changes according to the information recorded in UNDO. The UNDO records row IDs for inserts, column before-values for updates, and whole row data for deletes. |