查询碎片程度高（实际使用率小于30%）的表

可以收缩的表条件为什么block>100，因为一些很小的表，只有几行数据实际大小很小，但是block一次性分配就是5个（11g开始默认一次性分配1M的block大小了，见create table storged的NEXT参数），5个block相对于几行小表数据来说就相差太大了。

算法中/0.9是因为块的pfree一般为10%，所以一个块最多只用了90%，而且一行数据大于8KB时容易产生行链接，把一行分片存储，一样的一个块连90%都用不满，AVG_ROW_LEN还是比较准的，比如个人实验情况一表6个字段，一个number，其他5个都是char(100)但是实际数据都是'1111111'7位，AVG_ROW_LEN显示依然为513 。

```
SELECT TABLE_NAME,(BLOCKS*8192/1024/1024)"理论大小M",
(NUM_ROWS*AVG_ROW_LEN/1024/1024/0.9)"实际大小M",
round((NUM_ROWS*AVG_ROW_LEN/1024/1024/0.9)/(BLOCKS*8192/1024/1024),3)*100||'%'
"实际使用率%"
FROM USER_TABLES where blocks>100 and
(NUM_ROWS*AVG_ROW_LEN/1024/1024/0.9)/(BLOCKS*8192/1024/1024)<0.3
order by (NUM_ROWS*AVG_ROW_LEN/1024/1024/0.9)/(BLOCKS*8192/1024/1024) desc
```

查询索引碎片的比例

索引删除行数除以索引总行数的百分比>30%即认为索引碎片大，也就是需要重建的索引

```
select name,del_lf_rows,lf_rows,
round(del_lf_rows/decode(lf_rows,0,1,lf_rows)*100,0)||'%' frag_pct
from index_stats where round(del_lf_rows/decode(lf_rows,0,1,lf_rows)*100,0)>30;
```

集群因子clustering_factor高的表

集群因子越接近块数越好，接近行数则说明索引列的列值相等的行分布极度散列，可能不走索引扫描而走全表扫描：

方法一

```
select
tab.table_name,tab.blocks,tab.num_rows,ind.index_name,ind.clustering_factor,
round(nvl(ind.clustering_factor,1)/decode(tab.num_rows,0,1,tab.num_rows),3)*100|
|'%' "集群因子接近行数"
from user_tables tab, user_indexes ind where tab.table_name=ind.table_name
and tab.blocks>100
and nvl(ind.clustering_factor,1)/decode(tab.num_rows,0,1,tab.num_rows) between
0.35 and 3
```

方法二

```
select
tab.owner,tab.table_name,tab.blocks,tab.num_rows,ind.index_name,ind.clustering_f
actor,
round(nvl(ind.clustering_factor,1)/decode(tab.num_rows,0,1,tab.num_rows),3)*100|
|'%' "集群因子接近行数"
from dba_tables tab, dba_indexes ind where tab.table_name=ind.table_name and
tab.owner
not in
('SYS','SYSTEM','WMSYS','DBSNMP','CTXSYS','XDB','ORDDATA','SYSMAN','CATALOG','AP
EX_030200','MDSYS','OLAPSYS','EXFSYS')
and tab.blocks>100
and nvl(ind.clustering_factor,1)/decode(tab.num_rows,0,1,tab.num_rows) between
0.35 and 3
```

根据sid查spid或根据spid查sid

```
select
s.sid,s.serial#,p.spid,s.terminal,s.LOGON_TIME,s.status,s.PROGRAM,s.CLIENT_IDENT
IFIER,s.machine,s.action,s.MODULE,s.PROCESS "客户端机器进程号",s.osuser from
v$session s,v$process p
where  s.paddr=p.addr and s.sid=XX or p.spid=YY
```

根据sid查看具体的sql语句，不要加条件v$session.status=' ACTIVE'，比如toad对同一数据库开两个连接会话，都执行了一些语句，其中一个窗口查询select * from v$session时会发现另一个窗口在v$session.status是INACTIVE，并不代表另一个窗口没有执行过sql语句，而当前窗口是active状态，对应的sql_id对应的语句就是select * from v$session而不是之前执行过的sql语句，ACTIVE表示当前正在执行sql。

一个sid可能执行过很多个sql，所以有时需要的sql通过如下查不到是正常的，比如查询到某死锁源sid，通过如下查询可能只是个select语句，而真正引起死锁的sql却查不到，是因为可能这个sid持续了很长时间，这个sid之前执行的一些sql在v$sql可能已经被清除了。

方法一

```
select
username,sid,SERIAL#,LOGON_TIME,status,PROGRAM,CLIENT_IDENTIFIER,machine,action,
PROCESS "客户端机器进程号",osuser,sql_text from v$session a,v$sqltext_with_newlines
b
where DECODE(a.sql_hash_value, 0, prev_hash_value, sql_hash_value)=b.hash_value
and a.sid=&sid order by piece;
```

方法二

```
select
username,sid,SERIAL#,LOGON_TIME,status,sql_fulltext,PROGRAM,CLIENT_IDENTIFIER,ma
chine,a.action,PROCESS "客户端机器进程号",osuser from v$session a,v$sql b
where  DECODE(a.sql_hash_value, 0, prev_hash_value, sql_hash_value)=b.hash_value
and a.sid=&sid
```

如果上面语句执行太慢，则按如下两步

```
select  sql_hash_value, prev_hash_value, username,sid,SERIAL#,LOGON_TIME,status,
PROGRAM,CLIENT_IDENTIFIER,
machine,action,PROCESS "客户端机器进程号",osuser from v$session where  sid=&sid
select  sql_fulltext from v$sql where  hash_value=XX
```

--XX为上面 sql_hash_value，如果 sql_hash_value为0，则XX为上面 prev_hash_value

根据spid查询具体的sql语句(不要加条件v$session.status=' ACTIVE',比如toad对同一数据库开两个连接会话，都执行了一些语句，其中一个窗口查询select * from v$session时会发现另一个窗口在v$session.status是INACTIVE，并不代表另一个窗口没有执行过sql语句，而当前窗口是active状态，对应的sql_id对应的语句就是select * from v$session而不是之前执行过的sql语句，ACTIVE表示当前正在执行sql。)

```
Select ss.SID,ss.SERIAL#,ss.LOGON_TIME,pr.SPID,sa.SQL_FULLTEXT,ss.machine,
ss.TERMINAL,ss.PROGRAM,ss.USERNAME,ss.CLIENT_IDENTIFIER,ss.action,ss.PROCESS "客
户端机器进程号", ss.STATUS, ss.OSUSER,ss.status,ss.last_call_et,sa.sql_text
from v$process pr, v$session ss, v$sql sa
where pr.ADDR = ss.PADDR
and  DECODE(ss.sql_hash_value, 0, prev_hash_value, sql_hash_value)=sa.hash_value
and pr.spid=&spid
```

查看历史session_id的SQL来自哪个IP

查看trace文件名就可以知道spid,trace文件里面有sid和具体sql，如果trace存在incident，那trace就看不到具体sql，但是可以在incident文件中看到具体的sql，如DW_ora_17751.trc中17751就是spid，里面有这样的内容Incident 115 created, dump file: /XX/incident/incdir_115/DW_ora_17751_i115.trc，那么在DW_ora_17751_i115.trc就可以看到具体的sql语句)

DB_ora_29349.trc中出现

```
*** SESSION ID:(5057.12807) 2016-10-26 14:45:52.726
```

通过表V$ACTIVE_SESSION_HISTORY来查

```
select a.sql_id,a.machine,a.* from V$ACTIVE_SESSION_HISTORY a
where a.session_id=5057 and a.SESSION_SERIAL#=12807
```

查询上面的machine的IP

```
select s.sid,s.serial#,s.LOGON_TIME,s.machine,p.spid,p.terminal from v$session
s,v$process p
where  s.paddr=p.addr and s.machine='localhost'
```

通过上面的spid在oracle服务器上执行netstat -anp |grep spid即可

```
[oracle@dwdb trace]$ netstat -anp |grep 17630
tcp      210      0 192.168.64.228:11095        192.168.21.16:1521
ESTABLISHED 17630/oracleDB
tcp        0      0 ::ffff:192.168.64.228:1521  ::ffff:192.168.64.220:59848
ESTABLISHED 17630/oracleDB
```

出现两个，说明来自220，连接了228数据库服务器，但是又通过228服务器的dblink去连接了16服务器

查询死锁堵塞的会话sid

最简单的一个SQL

```
select * from V$SESSION_BLOCKERS
select * from dba_waiters
```

最常用的一个SQL

```
select sid,status,LOGON_TIME,sql_id,blocking_session "死锁直接
源",FINAL_BLOCKING_SESSION "死锁最终源",event,seconds_in_wait "会话锁住时间
_S",LAST_CALL_ET "会话STATUS持续时间_S"
from v$session where state='WAITING' and BLOCKING_SESSION_STATUS='VALID' and
FINAL_BLOCKING_SESSION_STATUS='VALID'
```

可以把两者SID放入v$session，发现LOGON_TIME字段FINAL_BLOCKING_SESSION比SID要早

> BLOCKING_SESSION:Session identifier of the blocking session. This column is valid only if BLOCKING_SESSION_STATUS has the value VALID.
> FINAL_BLOCKING_SESSION:Session identifier of the blocking session. This column is valid only if FINAL_BLOCKING_SESSION_STATUS has the value VALID.

如果遇到RAC环境，一定要用gv$来查，并且执行alter system kill session 'sid,serial#'要到RAC对应的实例上去执行

把上面被堵塞会话的sid代入如下语句，可以发现锁住的对象和对象的哪一行(如果sid是堵塞源的会话，则 row_wait_obj#=-1，表示锁持有者，就是死锁源了)

```
select
s.sid,s.username,d.owner,d.object_name,s.row_wait_obj#,s.row_wait_row#,s.row_wai
t_file#,s.row_wait_block#
from v$session s,dba_objects d where s.row_wait_obj#=d.object_id and s.sid
 in(XX,XX)
```

查询锁住的DDL对象

```
select d.session_id,s.SERIAL#,d.name
from dba_ddl_locks d,v$session s where d.owner='MKLMIGEM' and d.SESSION_ID=s.sid
```

查询超过两个小时的不活动会话

```
select
s.sid,s.serial#,p.spid,s.LOGON_TIME,s.LAST_CALL_ET,s.status,s.PROGRAM,s.CLIENT_I
DENTIFIER,s.machine,s.terminal,s.action,s.PROCESS "客户端机器进程号",s.osuser from
v$session s,v$process p
where  s.paddr=p.addr and s.sid in (select sid from v$session where machine<>&DB
服务器名称 and status='INACTIVE' and sql_id is null and LAST_CALL_ET>7200)
```

查询堵塞别的会话超过30分钟且自身是不活动的会话

```
select username,sid,serial#,status,seconds_in_wait,LAST_CALL_ET from v$session
 where sid in (select FINAL_BLOCKING_SESSION from v$session
  where state='WAITING' and BLOCKING_SESSION_STATUS='VALID' and
FINAL_BLOCKING_SESSION_STATUS='VALID') and status='INACTIVE' and sql_id is null
and seconds_in_wait>1800
```

查询可能存在连接池空闲初始配置过大的连接

（来自同一台机器的同一个程序的状态为INACTIVE的连接非常多）

```
select
count(ss.SID),ss.machine,ss.status,ss.TERMINAL,ss.PROGRAM,ss.USERNAME,ss.CLIENT_
IDENTIFIER
from v$session ss group by
ss.machine,ss.status,ss.TERMINAL,ss.PROGRAM,ss.USERNAME,ss.CLIENT_IDENTIFIER
having count(ss.SID)>10
```

查询当前正在执行的sql

```
SELECT
s.sid,s.serial#,s.username,spid,v$sql.sql_id,machine,s.terminal,s.program,sql_te
xt
FROM v$process,v$session s,v$sql
WHERE addr=paddr and s.sql_id=v$sql.sql_id AND sql_hash_value=hash_value and
s.STATUS='ACTIVE'
```

查询正在执行的SCHEDULER_JOB

```
select owner,job_name,sid,b.SERIAL#,b.username,spid from
ALL_SCHEDULER_RUNNING_JOBS,v$session b,v$process
 where session_id=sid and paddr=addr
```

查询正在执行的dbms_job

```
select job,b.sid,b.SERIAL#,b.username,spid from DBA_JOBS_RUNNING a ,v$session
b,v$process
 where a.sid=b.sid and paddr=addr
```

查询一个会话session、process平均消耗多少PGA内存，查看下面avg_used_M值

```
select round(sum(pga_used_mem)/1024/1024,0) total_used_M,
round(sum(pga_used_mem)/count(1)/1024/1024,0) avg_used_M,
round(sum(pga_alloc_mem)/1024/1024,0) total_alloc_M,
round(sum(pga_alloc_mem)/count(1)/1024/1024,0) avg_alloc_M from v$process;
```

TOP 10 执行次数排序

```
select *
from (select executions,username,PARSING_USER_ID,sql_id,sql_text
   from v$sql,dba_users where user_id=PARSING_USER_ID order by executions desc)
where rownum <=5;
```

TOP 10 物理读排序（消耗IO排序，即最差性能SQL、低效SQL排序）

```
select *
from (select
DISK_READS,username,PARSING_USER_ID,sql_id,ELAPSED_TIME/1000000,sql_text
   from v$sql,dba_users where user_id=PARSING_USER_ID order by DISK_READS desc)
where rownum <=5;
```

注意：不要使用DISK_READS/ EXECUTIONS来排序，因为任何一条语句不管执行几次都会耗逻辑读和cpu，可能不会耗物理读（遇到LRU还会耗物理读，LRU规则是执行最不频繁的且最后一次执行时间距离现在最久远的就会被交互出buffer cache），是因为buffer cache存放的是数据块，去数据块里找行一定会消耗cpu和逻辑读的。Shared pool执行存放sql的解析结果，sql执行的时候只是去share pool中找hash value，如果有匹配的就是软解析。所以物理读逻辑读是在buffer cache中，软解析硬解析是在shared pool。

TOP 10 逻辑读排序（消耗内存排序）

```
select *
from (select
BUFFER_GETS,username,PARSING_USER_ID,sql_id,ELAPSED_TIME/1000000,sql_text
   from v$sql,dba_users where user_id=PARSING_USER_ID order by BUFFER_GETS desc)
where rownum <=5;
```

注意：不要使用BUFFER_GETS/ EXECUTIONS来排序，因为任何一条语句不管执行几次都会耗逻辑读和cpu，可能不会耗物理读（遇到LRU还会耗物理读，LRU规则是执行最不频繁的且最后一次执行时间距离现在最久远的就会被交互出buffer cache），是因为buffer cache存放的是数据块，去数据块里找行一定会消耗cpu和逻辑读的。Shared pool执行存放sql的解析结果，sql执行的时候只是去share pool中找hash value，如果有匹配的就是软解析。所以物理读逻辑读是在buffer cache中，软解析硬解析是在shared pool)

TOP 10 CPU排序(单位秒=cpu_time/1000000)

```
select *
from (select
CPU_TIME/1000000,username,PARSING_USER_ID,sql_id,ELAPSED_TIME/1000000,sql_text

   from v$sql,dba_users where user_id=PARSING_USER_ID order by CPU_TIME/1000000
desc)
where rownum <=5;
```

注意：不要使用CPU_TIME/ EXECUTIONS来排序，因为任何一条语句不管执行几次都会耗逻辑读和cpu，可能不会耗物理读（遇到LRU还会耗物理读，LRU规则是执行最不频繁的且最后一次执行时间距离现在最久远的就会被交互出buffer cache），是因为buffer cache存放的是数据块，去数据块里找行一定会消耗cpu和逻辑读的。Shared pool执行存放sql的解析结果，sql执行的时候只是去share pool中找hash value，如果有匹配的就是软解析。所以物理读逻辑读是在buffer cache中，软解析硬解析是在shared pool。

查询等待事件

```
select event,sum(decode(wait_time,0,0,1)) "之前等待次数",
sum(decode(wait_time,0,1,0))  "正在等待次数",count(*)
 from v$session_wait  group by event order by 4 desc
```

查询当前等待事件对应的对象

```
select distinct wait_class#,wait_class from v$session_wait_class order by 1
```

以上sql发现wait_class#=6的是空闲等待

```
select * from
(select
sid,event,p1text,p1,p2text,p2,p3text,p3,WAIT_TIME,SECONDS_IN_WAIT,wait_class#
from v$session_wait where wait_class# <> 6 order by wait_time desc)
where rownum <=10;
```

能查出等待的对象是否来自数据文件（如果以上查到p1text是file#或file number）

```
select * from
(select owner,segment_name,segment_type,block_id,bytes from dba_extents where
file_id=p1 and block_id<p2 order="" by="" block_id="" desc)
where rownum<2
```

把上面第二个sql结果的p1、p2值代入上述sql的file_id、block_id

通过AWR的top sql或v$sql.sql_text查看是否有该对象的语句，检查该语句的执行计划就可以查出问题所在。

查询当前正在消耗临时空间的sql语句

方法一：

```
Select distinct se.username,
         se.sid,
         su.blocks * to_number(rtrim(p.value))/1024/1024 as space_G,
         su.tablespace,
         sql_text
    from V$TEMPSEG_USAGE su, v$parameter p, v$session se, v$sql s
   where p.name = 'db_block_size'
     and su.session_addr=se.saddr
     and su.sqlhash=s.hash_value
     and su.sqladdr=s.address
     and se.STATUS='ACTIVE'
```

方法二：

```
select v$sql.sql_id,v$sql.sql_fulltext,swa.TEMPSEG_SIZE/1024/1024 TEMPSEG_M,
swa.*
 from v$sql_workarea_active swa,v$sql where swa.sql_id=v$sql.sql_id and
swa.NUMBER_PASSES>0
```

查询因PGA不足而使用临时表空间的最频繁的10条SQL语句

```
select * from
(
select OPERATION_TYPE,ESTIMATED_OPTIMAL_SIZE,ESTIMATED_ONEPASS_SIZE,
sum(OPTIMAL_EXECUTIONS) optimal_cnt,sum(ONEPASS_EXECUTIONS) as onepass_cnt,
sum(MULTIPASSES_EXECUTIONS) as mpass_cnt,s.sql_text
from V$SQL_WORKAREA swa, v$sql s
where swa.sql_id=s.sql_id
group by OPERATION_TYPE,ESTIMATED_OPTIMAL_SIZE,ESTIMATED_ONEPASS_SIZE,sql_text
having sum(ONEPASS_EXECUTIONS+MULTIPASSES_EXECUTIONS)>0
order by sum(ONEPASS_EXECUTIONS) desc
)
where rownum<10
```

查询正在消耗PGA的SQL

```
select s.sql_text, sw.EXPECTED_SIZE, sw.ACTUAL_MEM_USED,sw.NUMBER_PASSES,
sw.TEMPSEG_SIZE
from v$sql_workarea_active sw, v$sql s
 where sw.sql_id=s.sql_id;
```

查询需要使用绑定变量的sql

10G以后推荐第二种

注意：任何一条执行过的语句不管执行了几次在V$SQL中都只有一条记录，V$SQL中会记录执行了几次。两条一模一样的语句但是在不同的schema下执行的两种结果，如select * from t1.test在sye、system下执行则V$SQL只有一条记录(谁先执行则PARSING_SCHEMA_NAME显示谁)。如在sys和system都执行select * from test则V$SQL中有两条记录，两条记录的CHILD_NUMBER和PARSING_SCHEMA_NAME不一样。

同一个用户下执行一样的语句如果大小写不一样或加了hint的话则会出现多个V$SQL记录，说明V$SQL对应的sql语句必须一模一样，如果alter system flush shared_pool（主站慎用）后再执行一样的语句，发现语句在V$SQL中的SQL_ID和HASH_VALUE与之前的一样，说明SQL_ID和HASH_VALUE应该是oracle自己的一套算法来的，只是根据sql语句内容来进行转换，sql语句不变则SQL_ID和HASH_VALUE也不变。

第一种

```
select * from (
select count(*),sql_id, substr(sql_text,1,40)
from v$sql
group by sql_id, substr(sql_text,1,40) having count(*) > 10 order by count(*)
desc) where rownum<10
```

第二种

```
count(1)>10表示类语句运行了10次以上
select sql_id, FORCE_MATCHING_SIGNATURE, sql_text
from v$SQL
where FORCE_MATCHING_SIGNATURE in
(select /*+ unnest */
FORCE_MATCHING_SIGNATURE
from v$sql
where FORCE_MATCHING_SIGNATURE > 0
and FORCE_MATCHING_SIGNATURE != EXACT_MATCHING_SIGNATURE
group by FORCE_MATCHING_SIGNATURE
having count(1) > 10)
```

查看数据文件可用百分比

dba_free_space并不会包含所有file_id，如果该数据文件满了，则 dba_free_space.file_id没有该数据文件，所以以下sql中 a.file_id=b.file_id的条件过滤后是不会有所有file_id的

```
select b.file_id,b.tablespace_name,b.file_name,b.AUTOEXTENSIBLE,
ROUND(b.MAXBYTES/1024/1024/1024,2) ||'G'  "文件最大可用总容量",
ROUND(b.bytes/1024/1024/1024,2) ||'G'  "文件总容量",
ROUND((b.bytes-sum(nvl(a.bytes,0)))/1024/1024/1024,2)||'G' "文件已用容量",
ROUND(sum(nvl(a.bytes,0))/1024/1024/1024,2)||'G' "文件可用容量",
ROUND(sum(nvl(a.bytes,0))/(b.bytes),2)*100||'%' "文件可用百分比"
from dba_free_space a,dba_data_files b
where a.file_id=b.file_id
group by
b.tablespace_name,b.file_name,b.file_id,b.bytes,b.AUTOEXTENSIBLE,b.MAXBYTES
order by b.tablespace_name;
```

--如下为标准版

```
select b.file_id,b.tablespace_name,b.file_name,b.AUTOEXTENSIBLE,
ROUND(b.MAXBYTES/1024/1024/1024,2) ||'G'  "文件最大可用总容量",
ROUND(b.bytes/1024/1024/1024,2) ||'G'  "文件当前总容量",
ROUND((b.bytes-sum(nvl(a.bytes,0)))/1024/1024/1024,2)||'G' "文件当前已用容量",
ROUND((decode(AUTOEXTENSIBLE,'NO',b.BYTES,b.MAXBYTES)+sum(nvl(a.bytes,0))-
b.bytes)/1024/1024/1024,2)||'G' "文件可用容量",
ROUND((decode(AUTOEXTENSIBLE,'NO',b.BYTES,b.MAXBYTES)+sum(nvl(a.bytes,0))-
b.bytes)/(decode(AUTOEXTENSIBLE,'NO',b.BYTES,b.MAXBYTES)),2)*100||'%' "文件可用百分
比"
from dba_free_space a,dba_data_files b
where a.file_id=b.file_id
group by
b.tablespace_name,b.file_name,b.file_id,b.bytes,b.AUTOEXTENSIBLE,b.MAXBYTES
order by decode(AUTOEXTENSIBLE,'NO',b.BYTES,b.MAXBYTES)+sum(nvl(a.bytes,0))-
b.bytes;
```

查看数据库文件的实际总量，单位G

```
select a.datafile_size+b.tempfile_size-c.free_size from
(select sum(bytes/1024/1024/1024) datafile_size from dba_data_files ) a,
(select sum(bytes/1024/1024/1024) tempfile_size from dba_temp_files ) b,
(select sum(bytes/1024/1024/1024) free_size from dba_free_space ) c
```

查看表空间可用百分比（dba_free_space不会包含所有tablespace，如果一个表空间的数据文件都满了，则这个表空间不会出现在dba_free_space中）

```
select b.tablespace_name,a.maxsize max_M,a.total total_M,b.free
free_M,round((b.free/a.total)*100) "% Free" from
(select tablespace_name, sum(bytes/(1024*1024)) total ,sum(MAXBYTES/(1024*1024))
maxsize from dba_data_files group by tablespace_name) a,
(select tablespace_name, round(sum(bytes/(1024*1024))) free from dba_free_space
group by tablespace_name) b
WHERE a.tablespace_name = b.tablespace_name order by "% Free";
```

--如下为标准版

```
select b.tablespace_name,a.maxsize max_M,a.total total_M,b.free
free_M,round(((a.maxsize+b.free-a.total)/a.maxsize)*100) "% Free" from
(select tablespace_name, sum(bytes/(1024*1024)) total
,sum((decode(AUTOEXTENSIBLE,'NO',BYTES,MAXBYTES))/(1024*1024)) maxsize from
dba_data_files group by tablespace_name) a,
(select tablespace_name, round(sum(bytes/(1024*1024))) free from dba_free_space
group by tablespace_name) b
WHERE a.tablespace_name = b.tablespace_name order by "% Free";
```

查看临时表空间使用率

方法一

```
SELECT temp_used.tablespace_name,round(total),used,
           round(total - used) as "Free",
           round(nvl(total-used, 0) * 100/total,1) "Free percent"
      FROM (SELECT tablespace_name, SUM(bytes_used)/1024/1024 used
              FROM GV$TEMP_SPACE_HEADER
             GROUP BY tablespace_name) temp_used,
           (SELECT tablespace_name,
SUM(decode(autoextensible,'YES',MAXBYTES,bytes))/1024/1024 total
              FROM dba_temp_files
             GROUP BY tablespace_name) temp_total
     WHERE temp_used.tablespace_name = temp_total.tablespace_name
```

方法二

```
SELECT a.tablespace_name, round(a.BYTES/1024/1024) total_M,
round(a.bytes/1024/1024 - nvl(b.bytes/1024/1024, 0)) free_M,
round(b.bytes/1024/1024) used,round(b.using/1024/1024) using
  FROM (SELECT   tablespace_name, SUM
(decode(autoextensible,'YES',MAXBYTES,bytes)) bytes FROM dba_temp_files GROUP BY
tablespace_name) a,
       (SELECT   tablespace_name, SUM (bytes_cached) bytes,sum(bytes_used) using
FROM v$temp_extent_pool GROUP BY tablespace_name) b
WHERE a.tablespace_name = b.tablespace_name(+)
```

查询undo表空间使用情况

```
select tablespace_name,status,sum(bytes)/1024/1024 M
 from dba_undo_extents group by tablespace_name,status
```

查询使用undo比较多的SQL

```
select *from (
  select maxqueryid,
 round(sum(undoblks )*8/1024) consumed_size_MB
 from v$undostat    group by maxqueryid order by  consumed_size_MB desc
 ) where rownum<10;
```

估计undo需要多大

```
SELECT (UR * (UPS * DBS)) AS "Bytes"
FROM (select max(tuned_undoretention) AS UR from v$undostat),
(SELECT undoblks/((end_time-begin_time)*86400) AS UPS
FROM v$undostat
WHERE undoblks = (SELECT MAX(undoblks) FROM v$undostat)),
(SELECT block_size AS DBS
FROM dba_tablespaces
WHERE tablespace_name = (SELECT UPPER(value) FROM v$parameter WHERE name =
'undo_tablespace'));
```

产生undo的当前活动会话是哪些

方法一

```
SELECT a.inst_id, a.sid, c.username, c.osuser, c.program, b.name,
a.value, d.used_urec, d.used_ublk
FROM gv$sesstat a, v$statname b, gv$session c, gv$transaction d
WHERE a.statistic# = b.statistic#
AND a.inst_id = c.inst_id
AND a.sid = c.sid
AND c.inst_id = d.inst_id
AND c.saddr = d.ses_addr
AND b.name = 'undo change vector size'
AND a.value>0
ORDER BY a.value DESC
```

方法二

```
select s.sid,s.serial#,s.sql_id,v.usn,r.status, v.rssize/1024/1024 mb
from dba_rollback_segs r, v$rollstat v,v$transaction t,v$session s
where r.segment_id = v.usn and v.usn=t.xidusn and t.addr=s.taddr
order by 6 desc;
```

查看ASM磁盘组使用率

```
select name,round(total_mb/1024) "总容量",round(free_mb/1024) "空闲空
间",round((free_mb/total_mb)*100) "可用空间比例"
 from gv$asm_diskgroup
```

统计每个用户使用表空间率

```
SELECT c.owner                                  "用户",
       a.tablespace_name                        "表空间名",
       total/1024/1024                          "表空间大小M",
       free/1024/1024                           "表空间剩余大小M",
       ( total - free )/1024/1024               "表空间使用大小M",
       Round(( total - free ) / total, 4) * 100 "表空间总计使用率    %",
       c.schemas_use/1024/1024                  "用户使用表空间大小M",
       round((schemas_use)/total,4)*100         "用户使用表空间率   %"

FROM   (SELECT tablespace_name,
               Sum(bytes) free
        FROM   DBA_FREE_SPACE
        GROUP  BY tablespace_name) a,
       (SELECT tablespace_name,
               Sum(bytes) total
        FROM   DBA_DATA_FILES
        GROUP  BY tablespace_name) b,
       (Select owner ,Tablespace_Name,
                Sum(bytes) schemas_use
        From Dba_Segments
        Group By owner,Tablespace_Name) c
WHERE  a.tablespace_name = b.tablespace_name
and a.tablespace_name =c.Tablespace_Name
order by "用户","表空间名"
```

查看闪回区\快速恢复区空间使用率

```
select sum(percent_space_used)||'%' "已使用空间比例"
from V$RECOVERY_AREA_USAGE
或
select round(100*(a.space_used/space_limit),2)||'%' "已使用空间比例",a.*
from v$recovery_file_dest a;
```

查看僵死进程

分两种（一种是会话不在的，另一种是会话标记为killed的但是会话还在的）
alter system kill session一执行则session即标记为KILLED，但是如果会话产生的数据量大则这个kill可能会比较久，在这个过程中session标记为KILLED但是这个会话还在V$session中，则V$session.paddr还在，所以可以匹配到V$process.addr，所以process进程还在；当kill过程执行完毕，则这个会话即不在V$session中

会话不在的

```
select * from v$process where addr not in (select paddr
from v$session) and pid not in (1,17,18)
```

会话还在的，但是会话标记为killed

```
select * from v$process where addr in (select paddr from v$session where
status='KILLED')
```

再根据上述结果中的SPID通过如下命令可以查看到process的启动时间

```
ps auxw|head -1;ps auxw|grep SPID
```

查看行迁移或行链接的表

```
select * From dba_tables where nvl(chain_cnt,0)<>0
chain_cnt : Number of rows in the table that are chained
from one data block to another or that have migrated to a new block, requiring a
link to preserve the old rowid. This column is updated only after you analyze
the table.
```

数据缓冲区命中率（百分比小于90就要加大db_cache_size）

```
SELECT a.VALUE+b.VALUE logical_reads, c.VALUE phys_reads,
round(100*(1-c.value/(a.value+b.value)),2)||'%' hit_ratio
FROM v$sysstat a,v$sysstat b,v$sysstat c
WHERE a.NAME='db block gets'
AND b.NAME='consistent gets'
AND c.NAME='physical reads';
```

方法二

```
SELECT DB_BLOCK_GETS+CONSISTENT_GETS Logical_reads,PHYSICAL_READS phys_reads,
round(100*(1-(PHYSICAL_READS/(DB_BLOCK_GETS+CONSISTENT_GETS))),2)||'%' "Hit
Ratio"
FROM V$BUFFER_POOL_STATISTICS WHERE NAME='DEFAULT';
```

共享池命中率（百分比小于90就要加大shared_pool_size）

以下两者可以根据个人理解运用

```
select sum(pinhits)/sum(pins)*100 from v$librarycache;
select sum(pinhits-reloads)/sum(pins)*100 from v$librarycache;
```

查询归档日志切换频率

```
select sequence#,to_char(first_time,'yyyymmdd_hh24:mi:ss')
firsttime,round((first_time-lag(first_time) over(order by first_time))*24*60,2)
minutes from
v$log_history where first_time > sysdate - 3 order by first_time,minutes;
或
select sequence#,to_char(first_time,'yyyy-mm-dd hh24:mi:ss')
First_time,First_change#,switch_change# from
v$loghist where first_time>sysdate-3 order by 1;
或
SELECT TO_CHAR(first_time, 'MM/DD') DAY,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '00', 1, 0)) H00,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '01', 1, 0)) H01,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '02', 1, 0)) H02,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '03', 1, 0)) H03,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '04', 1, 0)) H04,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '05', 1, 0)) H05,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '06', 1, 0)) H06,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '07', 1, 0)) H07,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '08', 1, 0)) H08,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '09', 1, 0)) H09,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '10', 1, 0)) H10,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '11', 1, 0)) H11,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '12', 1, 0)) H12,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '13', 1, 0)) H13,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '14', 1, 0)) H14,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '15', 1, 0)) H15,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '16', 1, 0)) H16,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '17', 1, 0)) H17,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '18', 1, 0)) H18,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '19', 1, 0)) H19,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '20', 1, 0)) H20,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '21', 1, 0)) H21,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '22', 1, 0)) H22,
SUM(DECODE(TO_CHAR(first_time, 'HH24'), '23', 1, 0)) H23,
COUNT(*) TOTAL
FROM (SELECT ROWNUM RN, FIRST_TIME FROM V$LOG_HISTORY WHERE first_time>sysdate-
18
and FIRST_TIME>ADD_MONTHS(SYSDATE,-1) ORDER BY FIRST_TIME)
GROUP BY TO_CHAR(first_time, 'MM/DD')
ORDER BY MIN(RN);
```

查询lgwr进程写日志时每执行一次lgwr需要多少秒，在state是waiting的情况下，某个等待编号seq#下，seconds_in_wait达多少秒，就是lgwr进程写一次IO需要多少秒

```
select event,state,seq#,seconds_in_wait,program from v$session where program
like '%LGWR%'  and state='WAITING'
```

查询没有索引的表

```
Select table_name from user_tables where table_name not in (select table_name
from user_indexes)
Select table_name from user_tables where table_name not in (select table_name
from user_ind_columns)
```

查询一个AWR周期内的平均session数、OS平均负载、平均db time、平均每秒多少事务

```
select to_char(max(BEGIN_TIME),'yyyy-mm-dd hh24:mi')||to_char(max(end_time),'--
hh24:mi') time,
snap_id,
trunc(sum(case metric_name when 'Session Count' then average end),2) sessions,
trunc(sum(case metric_name when 'Current OS Load' then average end),2) OS_LOAD,
(trunc(sum(case metric_name when 'Database Time Per Sec' then average
end),2)/100)*(ceil((max(end_time)-max(BEGIN_TIME))*24*60*60))
Database_Time_second,
trunc(sum(case metric_name when 'User Transaction Per Sec' then average end),2)
User_Transaction_Per_Sec
from dba_hist_sysmetric_summary
group by snap_id
order by snap_id;
```

--Database Time Per Sec对应值的单位是百分一秒/每秒
--(/100)*(ceil((max(end_time)-max(BEGIN_TIME))*24*60*60))*是代表每个snap周期内的总秒数，oracle 两个时间相减默认的是天数,*24*60*60 为相差的秒数
--这个SQL查到的DB TIME比较准确，和awr上面的db time比较一致

查询产生热块较多的对象

x$bh .tch(Touch)表示访问次数越高，热点快竞争问题就存在

```
SELECT e.owner, e.segment_name, e.segment_type
FROM dba_extents e,
(SELECT *
FROM (SELECT addr,ts#,file#,dbarfil,dbablk,tch
FROM x$bh
ORDER BY tch DESC)
WHERE ROWNUM < 11) b
WHERE e.relative_fno = b.dbarfil
AND e.block_id <= b.dbablk
AND e.block_id + e.blocks > b.dbablk;
```

手工创建快照的语句

```
exec dbms_workload_repository.create_snapshot;
```

AWR设置每隔30分钟收集一次报告，保留14天的报告

```
exec DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(retention=>14*24*60,
interval=>30);
select * from dba_hist_wr_control;
```

AWR基线查看和创建

```
select * from dba_hist_baseline;
exec
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE(start_snap_id=>7550,end_snap_id=>7660,b
aseline_name=>'am_baseline');
```

导出AWR报告的SQL语句

```
select * from dba_hist_snapshot
select * from table(dbms_workload_repository.awr_report_html(DBID,
INSTANCE_NUMBER, startsnapid,endsnapid))
select * from TABLE(DBMS_WORKLOAD_REPOSITORY.awr_diff_report_html(DBID,
INSTANCE_NUMBER, startsnapid,endsnapid, DBID, INSTANCE_NUMBER,
startsnapid,endsnapid));
```

导出最新ADDM的报告

（需要sys用户）

```
select dbms_advisor.get_task_report(task_name) from dba_advisor_tasks
where task_id =(
select max(t.task_id) from dba_advisor_tasks t, dba_advisor_log l where
t.task_id=l.task_id and t.advisor_name='ADDM' and l.status='COMPLETED' );
select task_id,task_name,description from dba_advisor_tasks order by 1 desc
select dbms_advisor.get_task_report(task_name) from dba_advisor_tasks where
task_id =XX
```

查询某个SQL的执行计划

```
select * from table(dbms_xplan.display_cursor('sql_id',0,' advanced '));
```

上面的0表示v$sql.child_number为0，如果一个sql_id在v$sql中有多行说明有多个child_number，要看哪儿child_number的执行计划，就写哪个的值，比如要看child_number为2的执行计划，就把上面sql的0改为2 。

官方文档对display_cursor这个函数的说明里面没有advanced这个参数值，只有BASIC、TYPICAL、ALL这几个，不过实践中发现advanced这个参数值显示的内容比这几个参数值显示的都多。

```
select * from table(xplan.display_cursor('v$sql.sql_id',0,'advanced'));
```

创建xplan包，再执行

```
SQL> CREATE PUBLIC SYNONYM XPLAN FOR SYS.XPLAN;
SQL> grant execute on sys.xplan to public;
```

查询Rman的配置信息

```
SELECT NAME,VALUE FROM V$RMAN_CONFIGURATION;
```

查询Rman备份集详细信息

（未过期的，过期并已删除的查不到）

```
SELECT B.RECID BackupSet_ID,
       A.SET_STAMP,
        DECODE (B.INCREMENTAL_LEVEL,
                '', DECODE (BACKUP_TYPE, 'L', 'Archivelog', 'Full'),
                1, 'Incr-1级',
                0, 'Incr-0级',
                B.INCREMENTAL_LEVEL)
           "Type LV",
        B.CONTROLFILE_INCLUDED "包含CTL",
        DECODE (A.STATUS,
                'A', 'AVAILABLE',
                'D', 'DELETED',
                'X', 'EXPIRED',
                'ERROR')
           "STATUS",
        A.DEVICE_TYPE "Device Type",
        A.START_TIME "Start Time",
        A.COMPLETION_TIME "Completion Time",
        A.ELAPSED_SECONDS "Elapsed Seconds",
        A.BYTES/1024/1024/1024 "Size(G)",
        A.COMPRESSED,
        A.TAG "Tag",
        A.HANDLE "Path"
   FROM GV$BACKUP_PIECE A, GV$BACKUP_SET B
  WHERE A.SET_STAMP = B.SET_STAMP AND A.DELETED = 'NO'
ORDER BY A.COMPLETION_TIME DESC;
```

查询Rman备份进度

```
SELECT SID, SERIAL#, opname,ROUND(SOFAR/TOTALWORK*100)||'%' "%_COMPLETE",
TRUNC(elapsed_seconds/60) || ':' || MOD(elapsed_seconds,60) elapsed,
TRUNC(time_remaining/60) || ':' || MOD(time_remaining,60) remaining,
CONTEXT,target,SOFAR, TOTALWORK
FROM V$SESSION_LONGOPS
WHERE OPNAME LIKE 'RMAN%'
AND OPNAME NOT LIKE '%aggregate%'
AND TOTALWORK != 0
AND SOFAR <> TOTALWORK;
```

查询执行过全表扫描的sql语句的SQL_ID和sql_fulltext

```
select
s.sid,s.serial#,s.inst_id,s.sql_id,s.username,s.target,s.ELAPSED_SECONDS,s.START
_TIME,s.LAST_UPDATE_TIME,v.sql_fulltext
  from gv$session_longops s,gv$sql v
where s.OPNAME = 'Table Scan'
   and s.SQL_PLAN_OPERATION = 'TABLE ACCESS'
   and s.SQL_PLAN_OPTIONS = 'FULL'
   and s.sql_id=v.sql_id
   order by s.LAST_UPDATE_TIME desc
```

查询死事务需要多长的回滚时间

```
X$KTUXE: [K]ernel [T]ransaction [U]ndo Transa[x]tion [E]ntry (table)
```

X$KTUXE表的一个重要功能是，可以获得无法通过v$transaction来观察的死事务信息，当一个数据库发生异常中断，或者进行延迟事务恢复时，数据库启动后，无法通过V$TRANSACTION来观察事务信息，但是X$KTUXE可以帮助我们获得这些信息。该表中的KTUXECFL代表了事务的Flag标记，通过这个标记可以找到那些Dead事务：

```
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;
    KTUXECFL                  COUNT(*)
    ------------------------ ----------
    DEAD                              1
NONE                          2393
```

KTUXESIZ用来记录事务使用的回滚段块数，可以通过观察这个字段来评估恢复进度,例如如下事务回滚经过测算需要大约3小时：

```
SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ from x$ktuxe where
KTUXECFL ='DEAD';
    ADDR             KTUXEUSN  KTUXESLT  KTUXESQN  KTUXESIZ
    ---------------- ---------- ---------- ---------- ----------
FFFFFFFF7D07B91C         10        39    2567412    1086075

SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ from x$ktuxe where
KTUXECFL ='DEAD';
    ADDR             KTUXEUSN  KTUXESLT  KTUXESQN  KTUXESIZ
    ---------------- ---------- ---------- ---------- ----------
    FFFFFFFF7D07B91C         10        39    2567412    1086067
```

```
SQL> declare
   l_start number;
   l_end    number;
   begin
    select ktuxesiz into l_start from x$ktuxe where  KTUXEUSN=10 and
KTUXESLT=39;
    dbms_lock.sleep(60);
    select ktuxesiz into l_end from x$ktuxe where  KTUXEUSN=10 and KTUXESLT=39;
    dbms_output.put_line('time_H:'|| round(l_end/(l_start -l_end)/60,2));
  end;
  /

time_H:3
```

把XXX用户下面的某些YYY表赋权给user,XXX\YYY要大写

```
set serveroutput on
--XXX要大写
declare tablename varchar2(200);
    begin
    for x IN (SELECT * FROM dba_tables where owner='XXX' and table_name like
'%YYY%') loop
    tablename:=x.table_name;
    dbms_output.put_line('GRANT SELECT ON XXX.'||tablename||' to user');
    EXECUTE IMMEDIATE 'GRANT SELECT ON XXX.'||tablename||' TO user';
    end loop;
end;
```

Oracle查出一个用户具有的所有系统权限和对象权限

系统权限（和用户自己查询select * from session_privs的结果一致）

```
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE = '用户名'
UNION ALL
SELECT * FROM DBA_SYS_PRIVS WHERE GRANTEE IN
(SELECT GRANTED_ROLE FROM DBA_ROLE_PRIVS WHERE GRANTEE = '用户名');
```

对象权限（和用户自己查询select * FROM TABLE_PRIVILEGES where GRANTEE='当前用户'的结果一致）

```
SELECT * FROM DBA_TAB_PRIVS WHERE GRANTEE = '用户名'
UNION ALL
SELECT * FROM DBA_TAB_PRIVS WHERE GRANTEE IN
(SELECT GRANTED_ROLE FROM DBA_ROLE_PRIVS WHERE GRANTEE = '用户名');
```

查询某个用户拥有的角色

```
select * from dba_role_privs where GRANTEE='用户名';
```

查询拥有DBA角色权限的用户

```
select * from dba_role_privs where GRANTED_ROLE='DBA';
```

查询某个角色拥有的系统权限

```
select * from ROLE_SYS_PRIVS where role='角色名'
```

清除某个SQL的执行计划

```
Exec DBMS_SHARED_POOL.PURGE('v$sqlarea.ADDRESS,v$sqlarea.HASH_VALUE','c')
```

查询密码是否有过期限制

默认是180天，一般修改为unlimited

```
select * from dba_profiles where profile='DEFAULT' and RESOURCE_NAME like
'PASSWORD%';
ALTER PROFILE DEFAULT LIMIT PASSWORD_LIFE_TIME UNLIMITED
```

查询和修改隐含参数

（必须在sysdba权限下操作）

```
select a.ksppinm name, b.ksppstvl value, a.ksppdesc description
  from x$ksppi a, x$ksppcv b
 where a.indx = b.indx and a.ksppinm like '%_small_table_threshold%'
alter system set "_small_table_threshold"=value scope=both sid='*';
```

不加sid则说明在默认在RAC的所有实例中修改
需要注意的是一定要加上双引号, 另外引号内不能有空格, 只能包含参数的名字

评估PGA该设置多少

```
select PGA_TARGET_FOR_ESTIMATE from (select  * from V$PGA_TARGET_ADVICE
 where ESTD_OVERALLOC_COUNT=0 order by 1) where rownum=1;
```

评估SGA该设置多少

```
select SGA_SIZE from (select * from V$SGA_TARGET_ADVICE
where ESTD_DB_TIME_FACTOR=1 order by 1) where rownum=1;
```

查看shared pool还剩多少

```
select * from v$sgastat where name='free memory' and pool='shared pool';
```

统计所有表的容量大小(含分区字段、LOB字段)

一般先执行select distinct SEGMENT_TYPE from dba_segments where owner<>'SYS' and tablespace_name<>'SYSAUX'查看到所有的segment_type

如下SQL就足够了

```
SELECT
   owner,table_name, TRUNC(sum(bytes)/1024/1024) Meg
FROM
(SELECT segment_name table_name, owner, bytes
 FROM dba_segments
 WHERE segment_type = 'TABLE'
 UNION ALL
SELECT s.segment_name table_name, pt.owner, s.bytes
 FROM dba_segments s, dba_part_tables pt
 WHERE s.segment_name = pt.table_name
 AND   s.owner = pt.owner
 AND   s.segment_type = 'TABLE PARTITION'
 UNION ALL
 SELECT i.table_name, i.owner, s.bytes
 FROM dba_indexes i, dba_segments s
 WHERE s.segment_name = i.index_name
 AND   s.owner = i.owner
 AND   s.segment_type = 'INDEX'
 UNION ALL
 SELECT pi.table_name, pi.owner, s.bytes
 FROM dba_part_indexes pi, dba_segments s
 WHERE s.segment_name = pi.index_name
 AND   s.owner = pi.owner
 AND   s.segment_type = 'INDEX PARTITION'
 UNION ALL
 SELECT l.table_name, l.owner, s.bytes
 FROM dba_lobs l, dba_segments s
 WHERE s.segment_name = l.segment_name
 AND   s.owner = l.owner
 AND   s.segment_type = 'LOBSEGMENT'
 UNION ALL
 SELECT l.table_name, l.owner, s.bytes
 FROM dba_lobs l, dba_segments s
 WHERE s.segment_name = l.index_name
 AND   s.owner = l.owner
 AND   s.segment_type = 'LOBINDEX'
 union all
 SELECT l.table_name, l.owner, s.bytes
 FROM dba_lobs l, dba_segments s
 WHERE s.segment_name = l.segment_name
 AND   s.owner = l.owner
 AND   s.segment_type = 'LOB PARTITION'
)
GROUP BY  owner,table_name
```

```
HAVING SUM(bytes)/1024/1024 > 10
ORDER BY SUM(bytes) desc
```

查看当前会话的SID

```
select * from V$MYSTAT where rownum<2
```

查询某个SID的某个统计信息

比如consistent gets一致性读

```
select A.SID,A.STATISTIC#,A.VALUE SID_VALUE,B.NAME,B.VALUE ALL_SID_VALUE from
V$SESSTAT A ,V$SYSSTAT B where A.STATISTIC#=B.STATISTIC#
and A.SID=1187 and B.NAME='consistent gets'
V$SYSSTAT统计整个DB的统计信息，V$SYSSTAT已经取代了V$STATNAME，并且多了VALUE这一列
V$SESSTAT统计每个用户的统计信息
```

查询某个SID的某个等待事件的信息

比如log file sync

```
select A.SID,A.EVENT,C.NAME,C.PARAMETER1,C.PARAMETER2,C.PARAMETER3,
A.TIME_WAITED SID_TIMEWAITED,B.TIME_WAITED ALL_SID_TIMEWAITED,A.TOTAL_WAITS
SID_TOTALWAITS,B.TOTAL_WAITS ALL_SID_TOTALWAITS
from V$SESSION_EVENT A ,V$SYSTEM_EVENT B,V$EVENT_NAME C where A.EVENT=B.EVENT
and A.EVENT=C.NAME and A.SID=1 and C.NAME='log file sync'
V$SESSION_EVENT描述每个用户的等待事件信息
V$SYSTEM_EVENT描述整个DB等待事件信息
V$EVENT_NAME描述等待事件信本身的信息(比如V$ACTIVE_SESSION_HISTORY的P1TEXT、P2TEXT、
P2TEXT匹配V$EVENT_NAME的PARAMETER1、PARAMETER2、PARAMETER3)
```

RAC跨节点杀会话

```
alter system kill session 'SID,serial#,@1'  --杀掉1节点的进程
alter system kill session 'SID,serial#,@2'  --杀掉2节点的进程
```

Truncate 分区的SQL

```
ALTER TABLE table_name TRUNCATE PARTITION p1 DROP STORAGE UPDATE GLOBAL INDEXES;
```

Drop分区的SQL

```
ALTER TABLE table_name DROP PARTITION p1 UPDATE GLOBAL INDEXES;
```

DATAGUARD主备延迟多少时间的查询方法

```
备 库sqlplus>select value from v$dataguard_stats where name='apply lag'
或
备库sqlplus>select ceil((sysdate-next_time)*24*60) "M" from v$archived_log where
applied='YES' AND SEQUENCE#=(SELECT MAX(SEQUENCE#)  FROM V$ARCHIVED_LOG WHERE
applied='YES');
```

查看某个包或存储过程是否正在被调用,如果如下有结果，则此时不能编译，否则会锁住

```
select * from V$DB_OBJECT_CACHE where pin>0 and name=upper('XX')
```

查询数据库打补丁的记录

```
select * from dba_registry_history;
```

查询某表的索引字段的distinct行数和CLUSTERING_FACTOR信息

```
select
a.table_name,a.index_name,b.COLUMN_NAME,a.blevel,a.distinct_keys,A.CLUSTERING_FA
CTOR,A.NUM_ROWS,trunc((a.distinct_keys/A.NUM_ROWS),2)*100||'%'
"distinct%",trunc((a.CLUSTERING_FACTOR/A.NUM_ROWS),2)*100||'%'
"CLUSTERING_FACTOR%"
 from DBA_IND_STATISTICS a,DBA_IND_COLUMNS b
where a.table_name='XX' and a.INDEX_NAME=b.index_name order by 5 desc
```

查询某表的所有字段的distinct行数

```
select
a.table_name,b.num_rows,a.column_name,a.data_type,a.data_length,a.num_distinct,t
runc((a.num_distinct/b.num_rows),2)*100||'%'
from dba_TAB_COLS a,dba_tables b where a.table_name='XX' and
a.table_name=b.table_name order by 6 desc
```

查询5G以上空闲空间可以进行收缩的数据文件

```
select 'alter database datafile ''' || a.file_name || ''' resize ' ||
round(a.filesize - (a.filesize - c.hwmsize) * 0.8) || 'M;',
a.filesize || 'M' as "数据文件的总大小",
c.hwmsize || 'M' as "数据文件的实用大小"
from (select file_id, file_name, round(bytes / 1024 / 1024) as filesize
from dba_data_files) a,
(select file_id, round(max(block_id) * 8 / 1024) as HWMsize
from dba_extents group by file_id) c
where a.file_id = c.file_id
and a.filesize - c.hwmsize > 5000;
```