# RDA工具

## 原始：

## 可直接部署



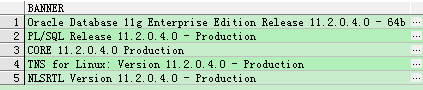
# RDA报告



# 版本和平台信息

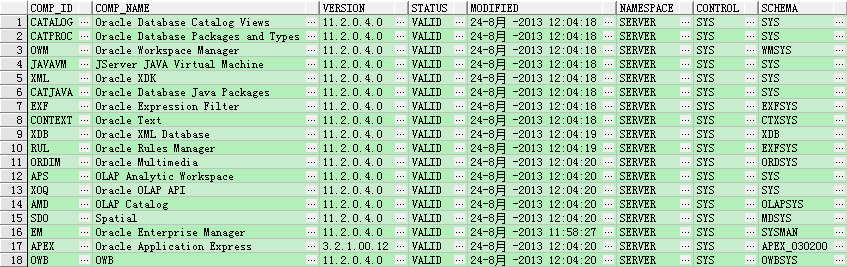
## 版本

|  |
| --- |
| SELECT \* FROM V$VERSION; |



## 安装了哪些组件

|  |
| --- |
| SELECT \* FROM DBA\_REGISTRY; |



## 是否是数据库集群

|  |
| --- |
| DECLARE  INST\_TAB DBMS\_UTILITY.INSTANCE\_TABLE;  INST\_CNT NUMBER;  BEGIN  IF DBMS\_UTILITY.IS\_CLUSTER\_DATABASE THEN  DBMS\_UTILITY.ACTIVE\_INSTANCES(INST\_TAB, INST\_CNT);  DBMS\_OUTPUT.PUT\_LINE('-' || INST\_TAB.FIRST);  DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(INST\_CNT));  ELSE  DBMS\_OUTPUT.PUT\_LINE('NOT A CLUSTERED DATABASE');  END IF;  END;  / |

## 是否为DataGuard

|  |
| --- |
| SELECT  PROTECTION\_MODE,  PROTECTION\_LEVEL,  REMOTE\_ARCHIVE,  DATABASE\_ROLE,  DATAGUARD\_BROKER,  GUARD\_STATUS  FROM  V$DATABASE; |

C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\K{UYU87{JM8XEJNUTP4LDD7.png

# 安全策略

## 有哪些人可以访问数据库？

## 有哪些主机可以访问数据库？

## 哪些人有哪些系统权限和对象权限？

## DBA有几个，超级用户密码如何管理？

## 审计有没有开，审计的粒度如何？

## 创建数据库用户并进行基本的赋权

|  |
| --- |
| CREATE USER DB133\_HIS IDENTIFIED BY db133\_his;  GRANT CONNECT TO DB133\_HIS;  GRANT RESOURCE TO DB133\_HIS;  GRANT UNLIMITED TABLESPACE TO DB133\_HIS; |

## 创建只能访问某个OWNER的表和在USER表空间只有10G配额的用户

|  |
| --- |
| CREATE ROLE BUSINESS\_SUPPORT\_ROLE;  GRANT CONNECT, RESOURCE TO BUSINESS\_SUPPORT\_ROLE;  CREATE ROLE ABOSS\_SUPPORT\_ROLE;  GRANT BUSINESS\_SUPPORT\_ROLE TO ABOSS\_SUPPORT\_ROLE;  CREATE USER HTGX\_RENYINGDA  IDENTIFIED BY renyingda\_1234  DEFAULT TABLESPACE USERS  TEMPORARY TABLESPACE TEMP  PROFILE DEFAULT  QUOTA 10G ON USERS;  GRANT ABOSS\_SUPPORT\_ROLE TO HTGX\_RENYINGDA;  SELECT 'GRANT SELECT ON ' || OWNER || '.' || TABLE\_NAME || ' TO HTGX\_RENYINGDA;' FROM DBA\_TABLES WHERE OWNER = 'DGABOSS';  SELECT 'GRANT INSERT ON ' || OWNER || '.' || TABLE\_NAME || ' TO HTGX\_RENYINGDA;' FROM DBA\_TABLES WHERE OWNER = 'DGABOSS';  SELECT 'GRANT DELETE ON ' || OWNER || '.' || TABLE\_NAME || ' TO HTGX\_RENYINGDA;' FROM DBA\_TABLES WHERE OWNER = 'DGABOSS';  SELECT 'DELETE DELETE ON ' || OWNER || '.' || TABLE\_NAME || ' TO HTGX\_RENYINGDA;' FROM DBA\_TABLES WHERE OWNER = 'DGABOSS'; |

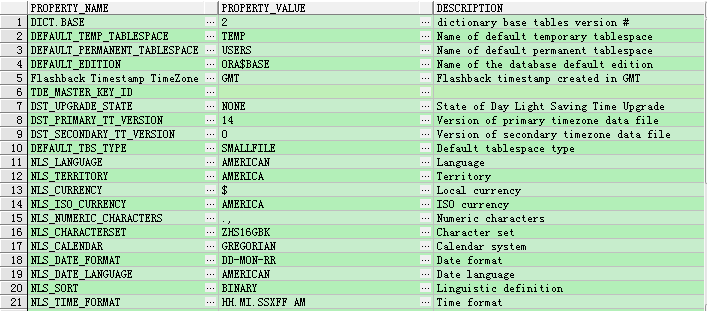
# 数据库参数

## 是否打开了Block Change Tracking

|  |
| --- |
| SELECT FILENAME, STATUS, BYTES FROM V$BLOCK\_CHANGE\_TRACKING; |

## 字符集等环境变量

|  |
| --- |
| SELECT \* FROM DATABASE\_PROPERTIES; |



## 强制归档和补充日志

|  |
| --- |
| SELECT  FORCE\_LOGGING,  SUPPLEMENTAL\_LOG\_DATA\_MIN,  SUPPLEMENTAL\_LOG\_DATA\_PK,  SUPPLEMENTAL\_LOG\_DATA\_UI,  SUPPLEMENTAL\_LOG\_DATA\_FK,  SUPPLEMENTAL\_LOG\_DATA\_ALL  FROM  V$DATABASE; |

## Flashback Database特性

|  |
| --- |
| SELECT FLASHBACK\_ON FROM V$DATABASE; |

## Parameters所有参数

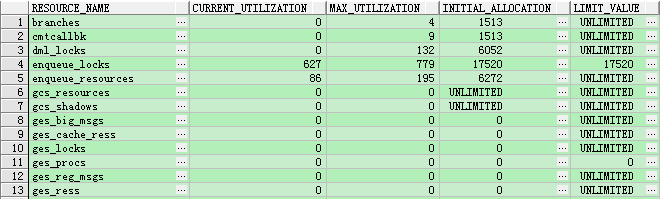
|  |
| --- |
| SELECT \* FROM V$PARAMETER ORDER BY NAME; |

## Option的打开情况

|  |
| --- |
| SELECT \* FROM V$OPTION; |

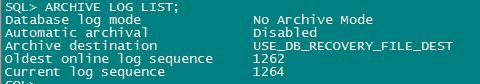
## 资源限制

|  |
| --- |
| SELECT \* FROM V$RESOURCE\_LIMIT ORDER BY RESOURCE\_NAME; |



## 打开归档模式

|  |
| --- |
| ARCHIVE LOG LIST; |

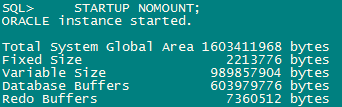


|  |
| --- |
| ALTER SYSTEM SET DB\_RECOVERY\_FILE\_DEST\_SIZE=20G;  ALTER SYSTEM SET DB\_RECOVERY\_FILE\_DEST='/data/archivedlog'; |

|  |
| --- |
| SHUTDOWN IMMEDIATE; |

C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\TAZ83N)[@OYXIJ9AEB}`M[Y.png

|  |
| --- |
| STARTUP NOMOUNT; |

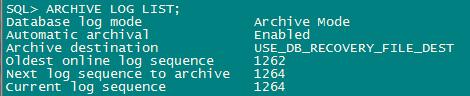


|  |
| --- |
| ALTER DATABASE MOUNT; |

|  |
| --- |
| ALTER DATABASE ARCHIVELOG; |

|  |
| --- |
| ALTER DATABASE OPEN; |

|  |
| --- |
| ARCHIVE LOG LIST; |



# 数据库SGA

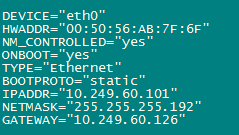
## SGA

## SGA使用统计

# 数据库网络相关

## 设置网络IP

|  |
| --- |
| $> vi /etc/sysconfig/network-scripts/ifcfg-eth0 |

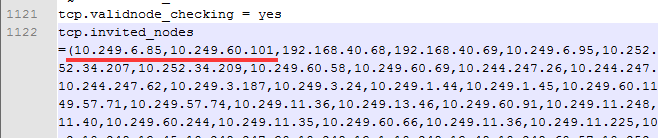


## 关闭主机的防火墙

|  |
| --- |
| $> service iptables stop |

## sqlnet.ora的配置

### 增加或者减少可访问数据库的IP

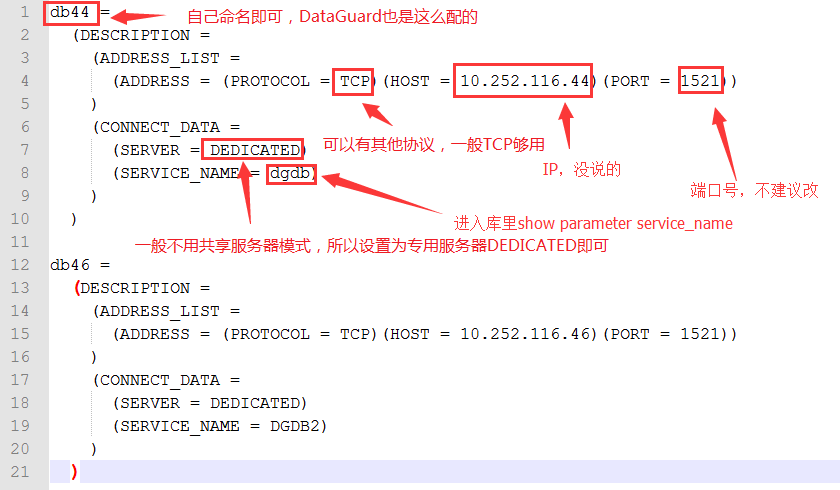
、

然后要重启监听：lsnrctl reload。

## listener.ora的配置

## tnsname.ora的配置

|  |
| --- |
| <alias>= [ (DESCRIPTION\_LIST = # Optional depending on whether u have  # one or more descriptions  # If there is just one description, unnecessary ]  (DESCRIPTION=  [ (SDU=2048) ] # Optional, defaults to 2048  # Can take values between 512 and 32K  [ (ADDRESS\_LIST= # Optional depending on whether u have  # one or more addresses  # If there is just one address, unnecessary ]  (ADDRESS=  [ (COMMUNITY=<community\_name>) ]  (PROTOCOL=tcp)  (HOST=<hostname>)  (PORT=<portnumber (1521 is a standard port used)>)  )  [ (ADDRESS=  (PROTOCOL=ipc)  (KEY=<ipckey (PNPKEY is a standard key used)>)  )  ]  [ (ADDRESS=  [ (COMMUNITY=<community\_name>) ]  (PROTOCOL=decnet)  (NODE=<nodename>)  (OBJECT=<objectname>)  )  ]  ... # More addresses  [ ) ] # Optional depending on whether ADDRESS\_LIST is used or not  [ (CONNECT\_DATA=  (SID=<oracle\_sid>)  [ (GLOBAL\_NAME=<global\_database\_name>) ]  )  ]  [ (SOURCE\_ROUTE=yes) ]  )  (DESCRIPTION=  [ (SDU=2048) ] # Optional, defaults to 2048  # Can take values between 512 and 32K  [ (ADDRESS\_LIST= ] # Optional depending on whether u have more  # than one address or not  # If there is just one address, unnecessary  (ADDRESS  [ (COMMUNITY=<community\_name>) ]  (PROTOCOL=tcp)  (HOST=<hostname>)  (PORT=<portnumber (1521 is a standard port used)>)  )  [ (ADDRESS=  (PROTOCOL=ipc)  (KEY=<ipckey (PNPKEY is a standard key used)>)  )  ]  ... # More addresses  [ ) ] # Optional depending on whether ADDRESS\_LIST  # is being used  [ (CONNECT\_DATA=  (SID=<oracle\_sid>)  [ (GLOBAL\_NAME=<global\_database\_name>) ]  )  ]  [ (SOURCE\_ROUTE=yes) ]  )  [ (CONNECT\_DATA=  (SID=<oracle\_sid>)  [ (GLOBAL\_NAME=<global\_database\_name>) ]  )  ]  ... # More descriptions  [ ) ] # Optional depending on whether DESCRIPTION\_LIST is used or not |



## 创建dblink

|  |
| --- |
| CREATE PUBLIC DATABASE LINK TO\_DB134 CONNECT TO zabbix IDENTIFIED BY zabbix  USING '  (  DESCRIPTION =  (  ADDRESS\_LIST =  (  ADDRESS = (PROTOCOL = TCP)(HOST = 10.252.116.134)(PORT = 1521)  )  )  (  CONNECT\_DATA =  (  SERVICE\_NAME = dgdbstandby  )  )  )'; |

## 查看数据库的所有DBLINK

|  |
| --- |
| SELECT \* FROM DBA\_DB\_LINKS; |

# 应用场景

## 有哪些应用系统使用，重要性评估

## 开发和运维人员的水平

# 数据库文件

## 控制文件路径

|  |
| --- |
| SELECT \* FROM V$CONTROLFILE; |

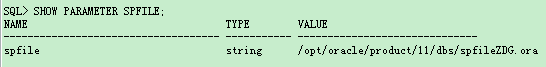
C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\QK0{$W`]QN9[0B07TD9}LIG.png

## 查看控制文件的内容

|  |
| --- |
| SQL>ALTER DATABASE BACKUP CONTROLFILE TO TRACE ;  SQL>SHOW PARAMETER USER\_DUMP\_DEST |

## 参数文件路径

|  |
| --- |
| SHOW PARAMETER SPFILE; |



|  |
| --- |
| SHOW PARAMETER PFILE; |

C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\W~C$SH)Z9[CT){LJ2@3VAV9.png

## 密码文件路径

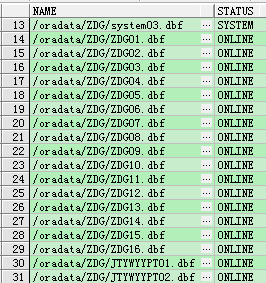
## 告警文件路径

|  |
| --- |
| SHOW PARAMETER BACKGROUND\_DUMP\_DEST; |

C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\F{8`4Y_GY2R@)LQSTDYM0OR.png

## 数据文件路径

|  |
| --- |
| SELECT NAME,STATUS FROM V$DATAFILE; |



## 监听日志文件路径

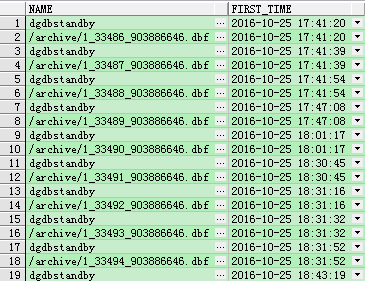
|  |
| --- |
| $> adrci  adrci < SHOW TRACEFILE; |

显示出：diag/tnslsnr/DG\_BAP\_QDSSK/listener/trace/listener.log

最终的路径为$ORACLE\_BASE/diag/tnslsnr/DG\_BAP\_QDSSK/listener/trace/listener.log

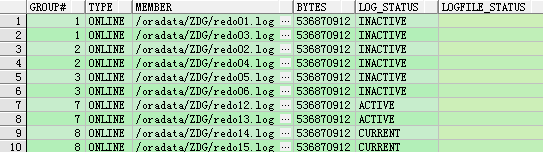
## 归档日志文件路径

|  |
| --- |
| SELECT NAME,FIRST\_TIME FROM V$ARCHIVED\_LOG; |



## 在线重做日志路径

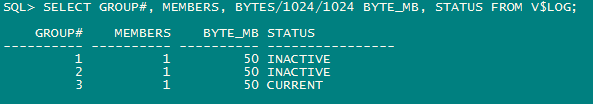
|  |
| --- |
| SELECT  L.GROUP#,  LF.TYPE,  LF.MEMBER,  L.BYTES,  L.STATUS LOG\_STATUS,  LF.STATUS LOGFILE\_STATUS  FROM  V$LOG L,  V$LOGFILE LF  WHERE  L.GROUP# = LF.GROUP#  ORDER BY  1,3; |



## 调整redo文件的大小（非DataGuard）

查看redo文件的组数

|  |
| --- |
| SELECT GROUP#, MEMBERS, BYTES/1024/1024 BYTE\_MB, STATUS FROM V$LOG; |



增加新的redo文件组及文件

|  |
| --- |
| ALTER DATABASE ADD LOGFILE GROUP 4 ('/oracle/app/oradata/baptestdb/redo0401.log','/oracle/app/oradata/baptestdb/redo0402.log') SIZE 1024M;  ALTER DATABASE ADD LOGFILE GROUP 5 ('/oracle/app/oradata/baptestdb/redo0501.log','/oracle/app/oradata/baptestdb/redo0502.log') SIZE 1024M;  ALTER DATABASE ADD LOGFILE GROUP 6 ('/oracle/app/oradata/baptestdb/redo0601.log','/oracle/app/oradata/baptestdb/redo0602.log') SIZE 1024M; |

查看增加是否成功

|  |
| --- |
| SELECT GROUP#, MEMBERS, BYTES/1024/1024 BYTE\_MB, STATUS FROM V$LOG; |

切换几次日志并查看日志是否切换到新增的几组日志上，当某组为CURRENT时，进行checkpoint，其他组将变为INACTIVE状态

|  |
| --- |
| ALTER SYSTEM SWITCH LOGFILE;  ALTER SYSTEM CHECKPOINT; |

当某个GROUP变为INACTIVE状态时，就可以DROP LOGFILE

|  |
| --- |
| ALTER DATABASE DROP LOGFILE GROUP 1;  ALTER DATABASE DROP LOGFILE GROUP 2;  ALTER DATABASE DROP LOGFILE GROUP 3; |

## 改变数据文件的路径

|  |
| --- |
| ALTER DATABASE OPEN;  ALTER TABLESPACE USERS OFFLINE;  cp /oracle/oracle/app/oradata/dgjszxdb/users01.dbf /oradata/dgjszx\_data1/users01.dbf  ALTER TABLESPACE USERS RENAME DATAFILE '/oracle/oracle/app/oradata/dgjszxdb/users01.dbf' TO '/oradata/dgjszx\_data1/users01.dbf';  ALTER TABLESPACE USERS ONLINE;  rm /oracle/oracle/app/oradata/dgjszxdb/users01.dbf |

## 归档日志的生成量

|  |
| --- |
| -- 每日的生成量  SELECT  TRUNC(COMPLETION\_TIME) TIME,  SUM(BLOCKS \* BLOCK\_SIZE)/1024/1024/1024 SIZE\_GB  FROM  V$ARCHIVED\_LOG  GROUP BY  TRUNC (COMPLETION\_TIME)  ORDER BY  1 DESC;  -- 每个小时的生成量  ALTER SESSION SET NLS\_DATE\_FORMAT = 'YYYY-MM-DD HH24';  SELECT  TRUNC(COMPLETION\_TIME,'HH24') TIME,  SUM(BLOCKS \* BLOCK\_SIZE)/1024/1024/1024 SIZE\_GB  FROM  V$ARCHIVED\_LOG  GROUP BY  TRUNC (COMPLETION\_TIME,'HH24')  ORDER BY  1 DESC; |

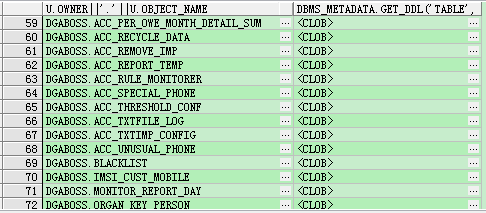
## 归档日志的切换频率

|  |
| --- |
| SELECT  TRUNC(FIRST\_TIME) AS "DATE",  TO\_CHAR(FIRST\_TIME, 'DY') AS "DAY",  COUNT(1) AS "COUNT",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '00', 1, 0)) AS H0,  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '01', 1, 0)) AS "H1",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '02', 1, 0)) AS "H2",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '03', 1, 0)) AS "H3",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '04', 1, 0)) AS "H4",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '05', 1, 0)) AS "H5",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '06', 1, 0)) AS "H6",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '07', 1, 0)) AS "H7",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '08', 1, 0)) AS "H8",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '09', 1, 0)) AS "H9",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '10', 1, 0)) AS "H10",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '11', 1, 0)) AS "H11",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '12', 1, 0)) AS "H12",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '13', 1, 0)) AS "H13",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '14', 1, 0)) AS "H14",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '15', 1, 0)) AS "H15",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '16', 1, 0)) AS "H16",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '17', 1, 0)) AS "H17",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '18', 1, 0)) AS "H18",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '19', 1, 0)) AS "H19",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '20', 1, 0)) AS "H20",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '21', 1, 0)) AS "H21",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '22', 1, 0)) AS "H22",  SUM(DECODE(TO\_CHAR(FIRST\_TIME, 'HH24'), '23', 1, 0)) AS "H23",  TO\_CHAR(ROUND(COUNT(1) / 24, 2), 'FM99999999990.00') AS "AVG"  FROM  GV$LOG\_HISTORY  WHERE  FIRST\_TIME >= TRUNC(SYSDATE) - 30  AND THREAD# = INST\_ID  GROUP BY  TRUNC(FIRST\_TIME), TO\_CHAR(FIRST\_TIME, 'DY')  ORDER BY  1 DESC; |

# 数据库对象

## 数据库对象定义

|  |
| --- |
| SELECT  U.OWNER ||'.'|| U.OBJECT\_NAME,  DBMS\_METADATA.GET\_DDL('TABLE', U.OBJECT\_NAME, U.OWNER)  FROM  DBA\_OBJECTS U  WHERE  U.OBJECT\_TYPE = 'TABLE'  AND U.OWNER = 'DGABOSS'; |



## 关键存储过程和定时任务

# 表空间

## 创建表空间

|  |
| --- |
| CREATE SMALLFILE TABLESPACE "DB133\_ZDG"  DATAFILE '/data/hisdata/db133\_zdg01.dbf' SIZE 1024M AUTOEXTEND ON NEXT 100M MAXSIZE 30G NOLOGGING  EXTENT MANAGEMENT LOCAL  SEGMENT SPACE MANAGEMENT AUTO; |

## 创建大表空间

一般小表空间的话，数据文件的大小是有限制的。譬如块大小是8K的话，建立的小空间的单个数据文件的大小为32G。当数据表的体量比较大，那么需要扩成多个数据文件就比较麻烦。

|  |
| --- |
| CREATE BIGFILE TABLESPACE NETWORK\_DATA  DATAFILE '/u01/app/oracle/oradata/orclai02/network.dbf '  SIZE 10G AUTOEXTEND ON NEXT 1000M MAXSIZE 800G; |

## 表空间连同表空间的数据一起删除

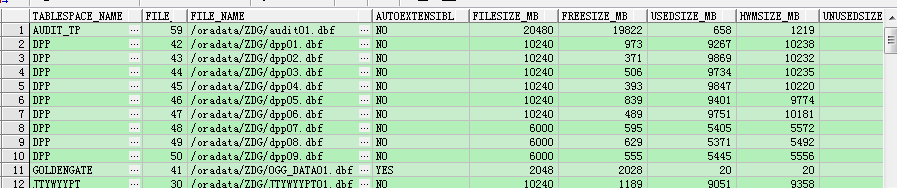
|  |
| --- |
| DROP TABLESPACE DB133\_ZDG INCLUDING CONTENTS AND DATAFILES; |

## 给表空间添加新的数据文件

|  |
| --- |
| ALTER TABLESPACE DB133\_ZDG ADD DATAFILE '/data/hisdata/db133\_zdg02.dbf' SIZE 1024M AUTOEXTEND ON NEXT 100M MAXSIZE 30G; |

## 查看数据文件的使用情况

|  |
| --- |
| SELECT */\*+ ORDERED USE\_HASH(A,B,C) \*/*  A.TABLESPACE\_NAME,  A.FILE\_ID,  A.FILE\_NAME,  A.AUTOEXTENSIBLE,  A.FILESIZE\_GB,  B.FREESIZE\_GB,  (A.FILESIZE\_GB-B.FREESIZE\_GB) USEDSIZE\_GB,  C.HWMSIZE\_GB,  C.HWMSIZE\_GB - (A.FILESIZE\_GB-B.FREESIZE\_GB) UNUSEDSIZE\_BELOWHWM\_GB,  A.FILESIZE\_GB - C.HWMSIZE\_GB CANSHRINKSIZE\_GB  FROM  (  SELECT  TABLESPACE\_NAME,  FILE\_ID,  FILE\_NAME,  ROUND(BYTES/1024/1024/1024, 2) FILESIZE\_GB,  AUTOEXTENSIBLE  FROM  DBA\_DATA\_FILES  ) A,  (  SELECT  FILE\_ID,  ROUND(SUM(DFS.BYTES)/1024/1024/1024, 2) FREESIZE\_GB  FROM  DBA\_FREE\_SPACE DFS  GROUP BY  FILE\_ID  ) B,  (  SELECT  FILE\_ID,  ROUND(MAX(BLOCK\_ID)\*8/1024/1024, 2) HWMSIZE\_GB  FROM  DBA\_EXTENTS  GROUP BY  FILE\_ID  ) C  WHERE  A.FILE\_ID = B.FILE\_ID  AND A.FILE\_ID = C.FILE\_ID  ORDER BY  A.TABLESPACE\_NAME,  A.FILE\_ID; |



## 收缩数据文件

收缩数据文件或者表空间，是在段空间收缩的基础上做的。

当做完统计信息收集，对段进行收缩和水位线的重置，就可以查看数据文件的使用情况，得到数据文件的一个水位线之后，将数据文件的大小resize到水位线上一点点即可。

**一定要注意，在收缩前，一定要看一下，数据文件以及对应的表空间，是否都是非自动扩展的。如果是非自动扩展，做收缩留的空间又不多，那数据库很容易因为无法写入而hang住！**

|  |
| --- |
| ALTER DATABASE DATAFILE '/oradata/dgjszx\_data1/users01.dbf' RESIZE 1000M; |

注意，在物理DG的配置下，主库做数据文件的收缩，会将收缩的操作结果自动同步到备库，使得备库的相应的数据文件也进行相应的收缩。在物理DG下，两个文件的数据块是一样的。注意，是操作的结果同步到备库，而不是将操作同步到备库。

## 收缩临时表空间

|  |
| --- |
| ALTER TABLESPACE TEMP SHRINK SPACE; |

## 收缩UNDO表空间

统计undo表空间里各种段（ACTIVE、EXPIRED、EXPIRED）的占比（Undo整体的使用状况）

|  |
| --- |
| SELECT  SEG.TABLESPACE\_NAME "TABLESPACE NAME",  TS.BYTES/1024/1024 "TABLESPACE SIZE(MB)",  UE.STATUS "UNDO STATUS",  COUNT(\*) "USED EXTENTS",  ROUND(SUM(UE.BYTES)/1024/1024, 2) "USED SIZE(MB)",  ROUND(SUM(UE.BYTES)/TS.BYTES\*100, 2) "USED RATE(%)"  FROM  DBA\_SEGMENTS SEG,  DBA\_UNDO\_EXTENTS UE,  (  SELECT  TABLESPACE\_NAME,  SUM(BYTES) BYTES  FROM  DBA\_DATA\_FILES  GROUP BY  TABLESPACE\_NAME  ) TS  WHERE  UE.SEGMENT\_NAME = SEG.SEGMENT\_NAME  AND SEG.TABLESPACE\_NAME = TS.TABLESPACE\_NAME  GROUP BY  SEG.TABLESPACE\_NAME,  TS.BYTES,  UE.STATUS  ORDER BY  SEG.TABLESPACE\_NAME; |

统计每个会话的UNDO使用情况

|  |
| --- |
| SELECT  S.USERNAME,  S.SID,  S.SERIAL#,  T.UBAFIL "UBA FILENUM",  T.UBABLK "UBA BLOCK NUMBER",  T.USED\_UBLK "NUMBER OS UNDO BLOCKS USED",  T.START\_TIME,  T.STATUS,  T.START\_SCNB,  T.XIDUSN ROLLID,  R.NAME ROLLNAME  FROM  V$SESSION S,  V$TRANSACTION T,  V$ROLLNAME R  WHERE  S.SADDR = T.SES\_ADDR  AND T.XIDUSN = R.USN; |

查看本库的UNDO表空间和对应的数据文件

|  |
| --- |
| SELECT  A.NAME,  B.NAME  FROM  V$TABLESPACE A,  V$DATAFILE B  WHERE  A.NAME = (SELECT VALUE FROM V$PARAMETER WHERE NAME = 'undo\_tablespace')  AND A.TS# = B.TS#; |

创建新的UNDO表空间时，貌似所需要等到老的UNDO表空间的事务结束

|  |
| --- |
| CREATE UNDO TABLESPACE UNDOTBS2 DATAFILE '/home/oracle/app/oracle/oradata/baptestdb/undotbs02.dbf' SIZE 100M REUSE AUTOEXTEND ON; |

切换前看看有没有事务，事务就有可能在使用UNDO表空间

|  |
| --- |
| SELECT COUNT(\*) FROM V$TRANSATION; |

要等到所有的回滚段都offline了，就可以切换了，这样更保险一些。当然直接切换，如果后续不用到UNDO，其实也没关系

|  |
| --- |
| SELECT USN,  XACTS,  STATUS,  RSSIZE / 1024 / 1024 / 1024 RSSIZE,  HWMSIZE / 1024 / 1024 / 1024 HWMSIZE,  SHRINKS  FROM  V$ROLLSTAT  ORDER BY  RSSIZE; |

切换UNDO表空间

|  |
| --- |
| ALTER SYSTEM SET UNDO\_TABLESPACE=UNDOTBS2 SCOPE = BOTH;  DROP TABLESPACE UNDOTBS1 INCLUDING CONTENTS AND DATAFILES;  SELECT \* FROM DBA\_TABLESPACES; |

## 数据文件的可扩展性查看

## 给某些用户制定访问配额

# 表

## 大表(不包括LOB对象)

|  |
| --- |
| SELECT  TABLESPACE\_NAME,  OWNER,  SEGMENT\_NAME,  SEGMENT\_TYPE,  BYTES/1024/1024 AS MBs  FROM  DBA\_SEGMENTS  WHERE  TABLESPACE\_NAME NOT IN ('SYSTEM', 'SYSAUX', 'UNDOTBS1')  AND SEGMENT\_TYPE = 'TABLE'  ORDER BY  MBs DESC,  TABLESPACE\_NAME,  OWNER; |

## 某个表的大小(包含LOB对象)

|  |
| --- |
| SELECT  NVL(SUM(BYTES),0)/1024/1024/1024  FROM  DBA\_SEGMENTS  WHERE  SEGMENT\_NAME IN  (  SELECT  SEGMENT\_NAME  FROM  DBA\_LOBS  WHERE  OWNER = 'DGABOSS'  AND TABLE\_NAME= 'TL\_NG3\_LOG'  )  OR  (  SEGMENT\_NAME = 'TL\_NG3\_LOG'  AND OWNER = 'DGABOSS'  ); |

## 快速克隆一个表

|  |
| --- |
| DECLARE  CURSOR cur IS  SELECT \* FROM aa;*--定义游标（aa换成你自己表名）*  TYPE rec IS TABLE OF aa%ROWTYPE;*--定义类型（这里的aa也要替换成你自己表名）*  recs rec;  BEGIN  OPEN cur; *--打开游标*  WHILE (TRUE) LOOP *--循环条件*  FETCH cur BULK COLLECT *--取游标里的值*  INTO recs LIMIT 10000; *--提交条件，每10000条提交*  FORALL i IN 1 .. recs.COUNT *--这个是个计数器，用来确保提交的条数，也是循环*  INSERT INTO bb VALUES recs (i);*--插入目标表（bb替换成你目标表名）*  COMMIT; *--提交*  EXIT WHEN cur%NOTFOUND;*--游标结束条件*  END LOOP; *--停止循环*  CLOSE cur;*--关闭游标*  END; |

## 创建散列分区表

|  |
| --- |
| CREATE TABLE DB133\_HIS.DGABOSS\_LX\_MF\_DETAIL  (  BATCH\_YEAR VARCHAR2(8),  BATCH\_NO VARCHAR2(32),  IMP\_INFO VARCHAR2(2000),  FILE\_NAME VARCHAR2(100)  )  PARTITION BY LIST(BATCH\_YEAR)  (  PARTITION P2014 VALUES ('2014') TABLESPACE DB133\_ZDG,  PARTITION P2015 VALUES ('2015') TABLESPACE DB133\_ZDG,  PARTITION P2016 VALUES ('2016') TABLESPACE DB133\_ZDG  ); |

## 创建范围分区表

|  |
| --- |
| CREATE TABLE DB133\_HIS.DGABOSS\_TL\_NG3\_LOG  (  REQ\_TIME DATE,  ID NUMBER(19),  SERVNUMBER VARCHAR2(12),  PROCESS\_CODE VARCHAR2(32),  RESP\_TIME VARCHAR2(32),  REQ\_SEQ VARCHAR2(20),  ROUTE\_TYPE VARCHAR2(2)  )  PARTITION BY RANGE(REQ\_TIME)  (  PARTITION P201608 VALUES LESS THAN (TO\_DATE('2016-09-01', 'YYYY-MM-DD')) TABLESPACE DB133\_ZDG,  PARTITION P201609 VALUES LESS THAN (TO\_DATE('2016-10-01', 'YYYY-MM-DD')) TABLESPACE DB133\_ZDG  ) |

## 查询某个分区

|  |
| --- |
| SELECT TABLE\_NAME,PARTITION\_NAME FROM DBA\_TAB\_PARTITIONS WHERE TABLE\_NAME = 'DGABOSS\_LX\_MF\_DETAIL';  SELECT \* FROM DB133\_HIS.DGABOSS\_LX\_MF\_DETAIL PARTITION (P2016); |

## 查询数据库某个用户的所有表的结构

|  |
| --- |
| SELECT  A.TABLE\_NAME AS "表名",  A.COLUMN\_NAME AS "字段名",  DECODE(A.CHAR\_LENGTH,0,DECODE(A.DATA\_SCALE,NULL,A.DATA\_TYPE,A.DATA\_TYPE||'('||A.DATA\_PRECISION||','||A.DATA\_SCALE||')'),A.DATA\_TYPE||'('||A.CHAR\_LENGTH||')') AS "字段类型",  A.DATA\_DEFAULT AS "默认值",  A.NULLABLE AS "能否为空",  B.COMMENTS AS "备注"  FROM  SYS.DBA\_TAB\_COLUMNS A,  SYS.DBA\_COL\_COMMENTS B  WHERE  A.OWNER = B.OWNER  AND A.TABLE\_NAME = B.TABLE\_NAME  AND A.COLUMN\_NAME = B.COLUMN\_NAME  AND A.OWNER = 'CONGXING'  AND A.TABLE\_NAME NOT LIKE 'BIN%'  ORDER BY  A.TABLE\_NAME |

# 约束

## 查看表上的约束及其定义

|  |
| --- |
| SELECT  CU.\*,  AU.CONSTRAINT\_TYPE  FROM  DBA\_CONS\_COLUMNS CU,  DBA\_CONSTRAINTS AU  WHERE  CU.CONSTRAINT\_NAME = AU.CONSTRAINT\_NAME  AND AU.TABLE\_NAME = 'QD\_GIS\_LOGINHIS'; |

|  |
| --- |
| SELECT DBMS\_METADATA.GET\_DDL('CONSTRAINT','SYS\_C0032550','HGQD') FROM DUAL; |

# 锁

## 查看锁

|  |
| --- |
| SELECT DISTINCT  S.SID,  S.BLOCKING\_SESSION,  S.EVENT,  S.USERNAME,  S.MACHINE,  DECODE(L.TYPE,  'MR', 'MEDIA RECOVERY',  'RT', 'REDO THREAD',  'UN', 'USER NAME',  'TX', 'TRANSACTION',  'TM', 'DML',  'UL', 'PL/SQL USER LOCK',  'DX', 'DISTRIBUTED XACTION',  'CF', 'CONTROL FILE',  'IS', 'INSTANCE STATE',  'FS', 'FILE SET',  'IR', 'INSTANCE RECOVERY',  'ST', 'DISK SPACE TRANSACTION',  'TS', 'TEMP SEGMENT',  'IV', 'LIBRARY CACHE INVALIDATION',  'LS', 'LOG START OR SWITCH',  'RW', 'ROW WAIT',  'SQ', 'SEQUENCE NUMBER',  'TE', 'EXTEND TABLE',  'TT', 'TEMP TABLE', L.TYPE) TYPE,  DECODE(L.LMODE,  0, 'NONE',  1, 'NULL',  2, 'ROW-S (SS)',  3, 'ROW-X (SX)',  4, 'SHARE',  5, 'S/ROW-X (SSX)',  6, 'EXCLUSIVE', L.LMODE) LMODE,  DECODE(L.REQUEST,  0, 'NONE',  1, 'NULL',  2, 'ROW-S (SS)',  3, 'ROW-X (SX)',  4, 'SHARE',  5, 'S/ROW-X (SSX)',  6, 'EXCLUSIVE', L.REQUEST) REQUEST,  DECODE(L.BLOCK,  0, 'NOT BLOCKING',  1, 'BLOCKING',  2, 'GLOBAL', L.BLOCK) BLOCK,  AO.OWNER,  AO.OBJECT\_NAME,  W.SECONDS\_IN\_WAIT,  A.SQL\_ID  FROM  V$LOCKED\_OBJECT LO,  ALL\_OBJECTS AO,  V$LOCK L,  V$SESSION S,  V$SQLAREA A,  V$SESSION\_WAIT W  WHERE  LO.OBJECT\_ID = AO.OBJECT\_ID  AND L.SID = LO.SESSION\_ID  AND S.SID = L.SID  AND A.ADDRESS = S.SQL\_ADDRESS(+)  AND A.HASH\_VALUE = S.SQL\_HASH\_VALUE(+)  AND S.SID = W.SID  ORDER BY  S.USERNAME |

## 查看锁等待

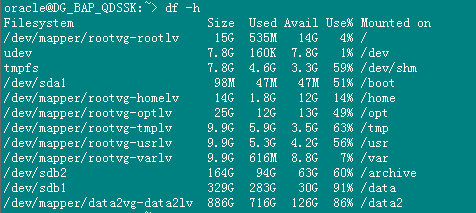
|  |
| --- |
| SELECT DISTINCT  B.OWNER,  B.OBJECT\_NAME,  A.LOCKED\_MODE,  A.SESSION\_ID,  C.SERIAL#,  C.LOGON\_TIME,  C.STATUS,  C.BLOCKING\_SESSION,  C.BLOCKING\_SESSION\_STATUS,  C.SADDR,  C.PADDR,  C.USERNAME,  C.MACHINE,  C.PROGRAM,  C.OSUSER  FROM  V$LOCKED\_OBJECT A,  DBA\_OBJECTS B,  V$SESSION C  WHERE  A.OBJECT\_ID = B.OBJECT\_ID  AND A.SESSION\_ID = C.SID; |

## 杀死对应有锁的session

|  |
| --- |
| ALTER SYSTEM KILL SESSION 'SID,SERIAL#'; |

# 容量

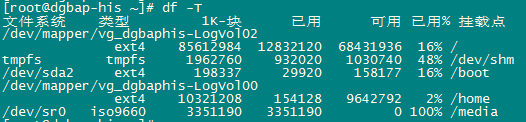
## 文件系统的大小



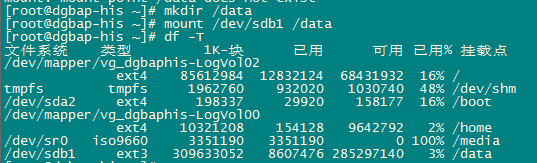
## Linux操作系统存储划分

### 挂载文件系统和卸载文件系统

用df命令列出文件系统，发现/dev/sdb1并不在列表里，这意味着这个分区没有挂载



通过mount命令，可以将分区临时挂载到某个挂载点

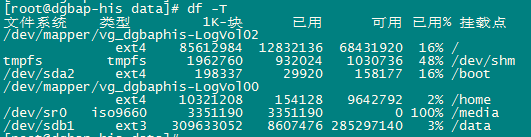


要将挂载点固定下来，就要编辑/etc/fstab



### 查看文件系统类型

|  |
| --- |
| $> df -T |



### 对某个分区创建文件系统

|  |
| --- |
| $> 某个分区命令 /dev/sdb1 |

其中，可选的分区命令按照目标文件系统的格式可以为：

mkfifo

mkfs

mkfs.cramfs

mkfs.ext2

mkfs.ext3

mkfs.ext4

mkfs.ext4dev

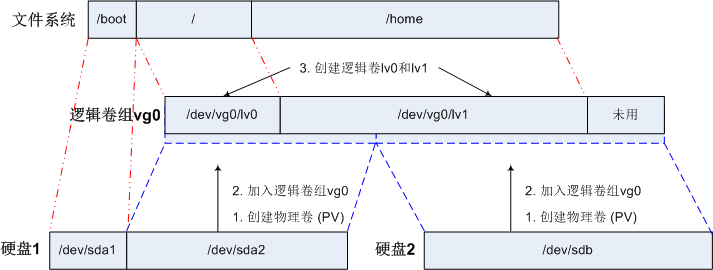
mkfs.gfs2

mkfs.msdos

mkfs.vfat

mkfs.xfs

### 查看整个主机的存储结构

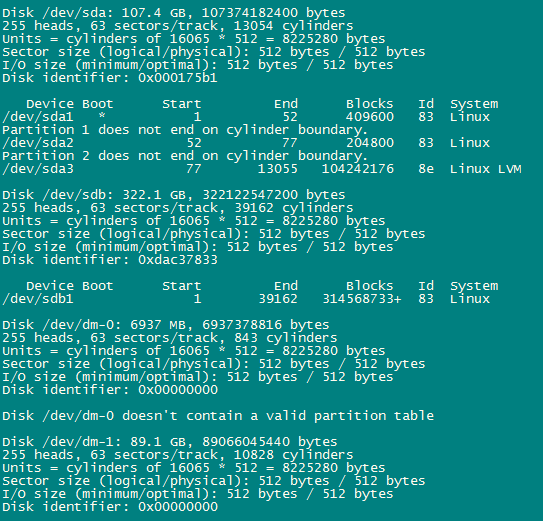


一个主机可能有好几个硬盘，一块硬盘可以划分物理分区，然后在物理分区上直接做文件系统。也可以将整块硬盘，或者硬盘的某些分区通过LVM管理，在LVM的逻辑分区上做文件系统。

整块硬盘或者硬盘的一些分区，将会打散成一个个PE(物理分区，physical extent)，那么这块硬盘就成为PV(物理卷，physical volume)。一个或者几个PV，可以加到一个VG(逻辑卷组，volume group)里。VG就相当于逻辑上的一个硬盘。在VG上会再划分LV(逻辑分区，logical volume)，最后在LV上创建文件系统。

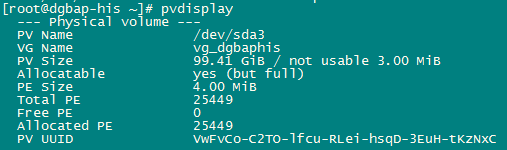
无论是物理分区的方式还是LVM的方式，最终存储都要划分成文件系统，才能被操作系统使用。

**查看磁盘：fdisk -l**



从上边可以看到，本机有两个磁盘，一个是/dev/sda，一个是/dev/sdb。/dev/sda分成了三个分区：/dev/sda1，/dev/sda2，/dev/sda3 ,其中，/dev/sda1和/dev/sda2的System类型是Linux，表示直接在上边创建文件系统。而/dev/sda3的System类型是LVM，表示/dev/sda3是加入了LVM管理，由操作系统在LVM的基础上再创建文件系统。其中的区别是，直接在分区上建立的文件系统，以后无法扩展空间，而在 LVM上建立的文件系统，以后加上新的分区，是可以将存储加到文件系统来。

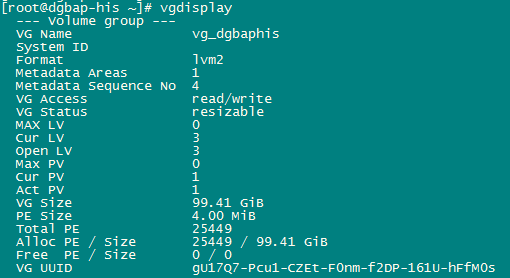
**查看物理卷：pvdisplay**



**创建物理卷：pvcreate**

**删除物理卷：pvremove**

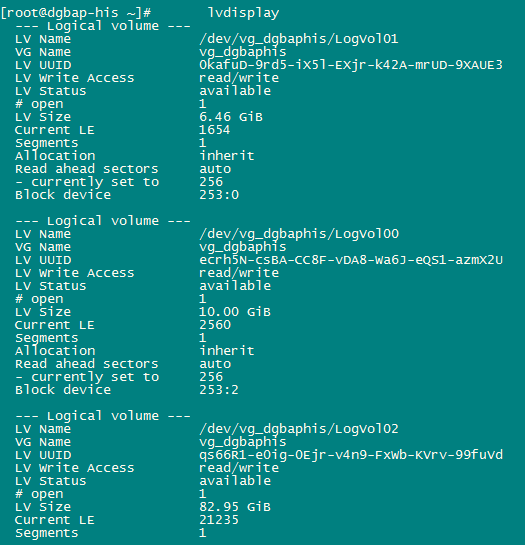
**查看逻辑卷：vgdisplay**



**创建逻辑卷：vgcreate**

**删除逻辑卷：vgremove**

**查看逻辑分区：lvdisplay**



**创建逻辑分区：lvcreate**

**删除逻辑分区：lvremove**

## 数据量

|  |
| --- |
| SELECT SUM(BYTES/1024/1024/1024) FROM V$DATAFILE; |

## 每个数据文件的大小

|  |
| --- |
| SELECT NAME, STATUS, BYTES/1024/1024/1024 FROM V$DATAFILE ORDER BY NAME; |

## 表空间维度数据增长量统计

创建统计记录表

|  |
| --- |
| CREATE TABLE DGLJW\_TABLESPACE\_STAT  (  SAMPLE\_TIME DATE,  TABLESPACE\_NAME VARCHAR2(32),  SUM\_SPACE\_M FLOAT,  SUM\_BLOCKS NUMBER,  USED\_SPACE\_M FLOAT,  USED\_RATE FLOAT,  FREE\_SPACE\_M FLOAT  ); |

创建统计的存储过程

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DGLJW\_TABLESPACE\_STAT\_PROC AS  BEGIN  INSERT INTO DGLJW\_TABLESPACE\_STAT  SELECT  SYSDATE,  T.\*  FROM  (  (SELECT D.TABLESPACE\_NAME,  SPACE "SUM\_SPACE(M)",  BLOCKS SUM\_BLOCKS,  SPACE - NVL(FREE\_SPACE, 0) "USED\_SPACE(M)",  ROUND((1 - NVL(FREE\_SPACE, 0) / SPACE) \* 100, 2) "USED\_RATE(%)",  FREE\_SPACE "FREE\_SPACE(M)"  FROM (SELECT TABLESPACE\_NAME,  ROUND(SUM(BYTES) / (1024 \* 1024), 2) SPACE,  SUM(BLOCKS) BLOCKS  FROM DBA\_DATA\_FILES  GROUP BY TABLESPACE\_NAME) D,  (SELECT TABLESPACE\_NAME,  ROUND(SUM(BYTES) / (1024 \* 1024), 2) FREE\_SPACE  FROM DBA\_FREE\_SPACE  GROUP BY TABLESPACE\_NAME) F  WHERE D.TABLESPACE\_NAME = F.TABLESPACE\_NAME(+)  UNION ALL  SELECT D.TABLESPACE\_NAME,  SPACE "SUM\_SPACE(M)",  BLOCKS SUM\_BLOCKS,  USED\_SPACE "USED\_SPACE(M)",  ROUND(NVL(USED\_SPACE, 0) / SPACE \* 100, 2) "USED\_RATE(%)",  SPACE - USED\_SPACE "FREE\_SPACE(M)"  FROM (SELECT TABLESPACE\_NAME,  ROUND(SUM(BYTES) / (1024 \* 1024), 2) SPACE,  SUM(BLOCKS) BLOCKS  FROM DBA\_TEMP\_FILES  GROUP BY TABLESPACE\_NAME) D,  (SELECT TABLESPACE,  ROUND(SUM(BLOCKS \* 8192) / (1024 \* 1024), 2) USED\_SPACE  FROM V$SORT\_USAGE  GROUP BY TABLESPACE) F  WHERE D.TABLESPACE\_NAME = F.TABLESPACE(+))  ) T;  END;  / |

创建JOB并执行

|  |
| --- |
| VARIABLE JOB1 NUMBER;  BEGIN  DBMS\_JOB.SUBMIT(:JOB1,'DGLJW\_TABLESPACE\_STAT\_PROC;',SYSDATE,'SYSDATE+1'); *-- 每天执行一次*  END;  /  BEGIN  DBMS\_JOB.RUN(:JOB1);  END;  / |

## 段维度数据增长量统计

创建统计记录表

|  |
| --- |
| CREATE TABLE DGLJW\_LARGE\_SEGMENT\_STAT  (  SAMPLE\_TIME DATE,  OWNER VARCHAR2(32),  SEGMENT\_NAME VARCHAR2(32),  PARTITION\_NAME VARCHAR2(32),  SEGMENT\_TYPE VARCHAR2(32),  TABLESPACE\_NAME VARCHAR2(32),  BYTES\_M FLOAT,  BLOCKS NUMBER,  EXTENTS NUMBER  ); |

创建统计的存储过程

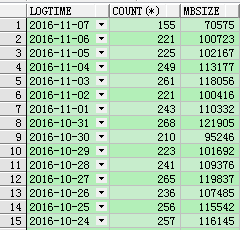
|  |
| --- |
| CREATE OR REPLACE PROCEDURE DGLJW\_LARGE\_SEGMENT\_STAT\_PROC AS  BEGIN  INSERT INTO DGLJW\_LARGE\_SEGMENT\_STAT  SELECT  SYSDATE,  T.\*  FROM  (  SELECT  OWNER,  SEGMENT\_NAME,  PARTITION\_NAME,  SEGMENT\_TYPE,  TABLESPACE\_NAME,  BYTES/1024/1024,  BLOCKS,  EXTENTS  FROM  DBA\_SEGMENTS  WHERE  BYTES/1024/1024 > 500  ) T;  END;  / |

创建JOB并执行

|  |
| --- |
| VARIABLE JOB1 NUMBER;  BEGIN  DBMS\_JOB.SUBMIT(:JOB1,'DGLJW\_LARGE\_SEGMENT\_STAT\_PROC ;',SYSDATE,'SYSDATE+1'); *-- 每天执行一次*  END;  /  BEGIN  DBMS\_JOB.RUN(:JOB1);  END;  / |

## 每日归档量统计

|  |
| --- |
| SELECT  LOGTIME,  COUNT(\*),  ROUND(SUM(BLOCKS \* BLOCK\_SIZE) / 1024 / 1024 / 1024) GBSIZE  FROM  (  SELECT  TRUNC(FIRST\_TIME, 'DD') AS LOGTIME,  A.BLOCKS,  A.BLOCK\_SIZE  FROM  V$ARCHIVED\_LOG A  WHERE  A.DEST\_ID = 1  )  GROUP BY  LOGTIME  ORDER BY  LOGTIME DESC; |



## 数据库用户数

## 数据库表数目

## 连接数统计

|  |
| --- |
| SELECT  A.SNAP\_ID,  B.SAMPLE\_HOUR,  A.TOTAL\_PER\_HOUR,  A.AVG\_PER\_MINUTE  FROM  (  SELECT  SNAP\_ID,  COUNT(\*) AS TOTAL\_PER\_HOUR,  ROUND(COUNT(\*)/60, 2) AS AVG\_PER\_MINUTE  FROM  DBA\_HIST\_ACTIVE\_SESS\_HISTORY  GROUP BY  SNAP\_ID  ) A  LEFT JOIN  (  SELECT  SNAP\_ID,  TO\_CHAR(MIN(SAMPLE\_TIME), 'YYYY-MM-DD HH24') AS SAMPLE\_HOUR  FROM  DBA\_HIST\_ACTIVE\_SESS\_HISTORY  GROUP BY  SNAP\_ID  ) B  ON A.SNAP\_ID = B.SNAP\_ID  ORDER BY  A.SNAP\_ID |

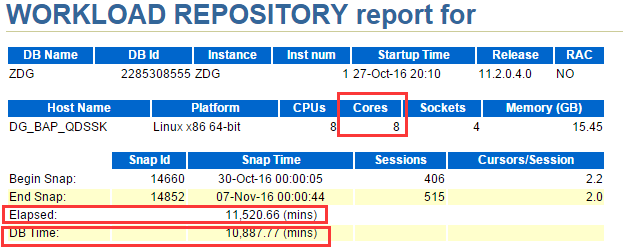
# 性能瓶颈

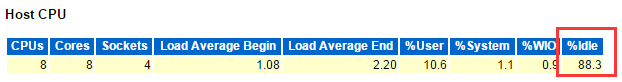
## 总体负载

从AWR报告中，DB Time =所有前台session花费在database调用上的总和时间，包括CPU时间、IO Time、和其他一系列非空闲等待时间，和cpu on queue time。

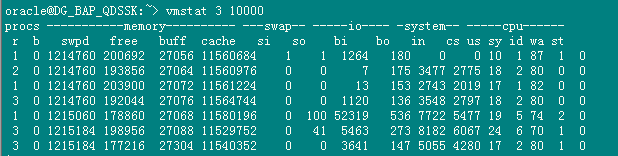
数据库启动以来到现在的时间为11,520.66分钟，CPU有8核，所以总的时间为11,520.66×8=92,165.28分钟，其中，用于前台session调用的时间为10,887.77分钟。

实际上，前台session占用CPU的时间为10,887.77÷92,165.28=0.1181331，实际上示例中的数据库是一点都不忙的。





## CPU使用率



## 内存？

## 网络IO?

## 磁盘IO?

## TopSQL?

## 压测？

## 业务高峰

# 数据迁移之数据泵

## expdp导出某个用户的所有表结构

|  |
| --- |
| CREATE OR REPLACE DIRECTORY export\_dir AS '/archive/backup';  SELECT \* FROM DBA\_DIRECTORIES;  expdp userid=\' sys/I\_Love\_Gmcc\_130 as sysdba\' directory=export\_dir  schemas=hgqd dumpfile=exp\_hgqd.dmp logfile=exp\_hgqd.log  CONTENT=METADATA\_ONLY parallel=16 |

## expdp评估导出的文件大小

|  |
| --- |
| expdp \' sys/password as sysdba\' TABLES=DGABOSS.TL\_NG3\_LOG estimate\_only=y estimate=statistics |

## expdp导出文件

|  |
| --- |
| CREATE DIRECTORY EXPDPDIR AS '/oradata'  SELECT \* FROM DBA\_DIRECTORIES WHERE DIRECTORY\_NAME = 'EXPDPDIR' ;  GRANT READ,WRITE ON DIRECTORY EXPDPDIR TO PUBLIC; |

|  |
| --- |
| expdp \' sys/dgitc123 as sysdba\' TABLES=DGABOSS\_TL\_NG3\_LOG DUMPFILE=DGABOSS\_TL\_NG3\_LOG.dmp DIRECTORY=EXPDPDIR PARALLEL=4 LOGFILE=DGABOSS.TL\_NG3\_LOG.log |

## SCP传输

|  |
| --- |
| scp urlLocalFile remoteUser@remoteIP:urlRemoteFile |

## impdp导入文件

|  |
| --- |
| impdp \' sys/Dgbap\_2345 as sysdba \' DIRECTORY=IMPDPDIR DUMPFILE=DGABOSS\_LX\_MF\_DETAIL.dmp REMAP\_SCHEMA=DGABOSS:DB133\_HIS REMAP\_TABLESPACE=ZDG:DB133\_ZDG LOGFILE=DGABOSS\_LX\_MF\_DETAIL.log TABLE\_EXISTS\_ACTION=REPLACE EXCLUDE=GRANT |

# Data Guard

## DataGuard与GoldenGate对比

|  |  |  |
| --- | --- | --- |
| **对比维度** | **Oracle DataGuard** | **Oracle GoldenGate** |
| 原理 | 复制归档日志或在线日志 | 抽取在线日志中的数据变化，转换为GGS自定义的数据格式存放在本地队列或远端队列中 |
| 稳定性 | 作为灾备的稳定性极高 | 稳定性不如DataGuard |
| 维护 | 维护简单，极少出现问题 | 命令行方式，维护较复杂 |
| 对象支持 | 完全支持 | 部分对象需手工创建于维护 |
| 备份端可用性 | 备份端处于恢复或只读状态，在只读状态下不能同时进行恢复。 | 两端数据库是活动的，备份端可以提供实时的数据查询及报表业务等，从而提高系统整体的业务处理能力，充分利用备份端的计算能力，提升系统整体业务处理性能。可以实现两端数据的同时写入 |
| 接管时间 | 数据库工作在mount状态下，接管业务时，数据库要open | 可实现立即接管 |
| 复制方式 | 通过恢复机制实现的，无法实现同步复制 | GoldenGate可以提供秒一级的大量数据实时捕捉和投递，异步复制方式，无法实现同步复制 |
| 资源占用 | 复制通过数据库的LGWR进程或ARCN进程完成，占用数据库少量资源 | 业务高峰时在数据抽取转换时消耗系统资源较多，低峰时占用较小 |
| 异构数据库支持 | 单一数据库解决方案，仅运行在Oracle数据库上，源端和目标端操作系统必须相同，版本号可以不同 | 可以在不同类型和版本的数据库之间进行数据复制。如ORACLE，DB2，SYBASE，SQL SERVER，INFORMIX、Teradata等。 适用于不同操作系统如windows、linux、unix、aix等 |
| 带宽占用 | 使用Oracle Net传输日志，可通过高级压缩选项进行压缩，压缩比在2-3倍 | 利用TCP/IP传输数据变化，集成数据压缩，提供理论可达到9：1压缩比的数据压缩特性 |
| 拓扑结构 | 可以实现一对多模式 | 可以实现一对一、一对多、多对一、双向复制等多种拓扑结构 |

## DataGuard启停

Dataguard关闭（先关主库再关备库）

Dataguard开启（先开备库再开主库）

## SCN与系统时间戳的对应关系

每套库的SCN消耗的速度是不一样的。因此这个转换也是针对特定的库的，同一个SCN号，在不同的库里转换出的时间戳是不一样的。

|  |
| --- |
| SELECT SCN\_TO\_TIMESTAMP(49733520950) SCN FROM DUAL;  SELECT TIMESTAMP\_TO\_SCN(TO\_TIMESTAMP('2014-11-10 18:19:20.123456789','YYYY-MM-DD HH24:MI:SS.FF')) SCN FROM DUAL; |

## 查看DataGuard同步情况

主备库都查一下

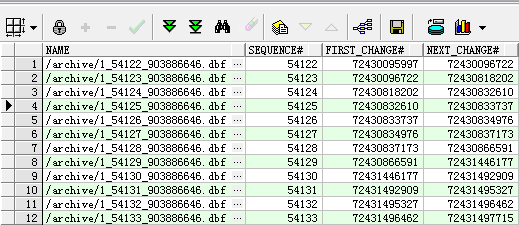
|  |
| --- |
| SELECT  A.SEQUENCE#,  A.NAME,  A.STANDBY\_DEST,  A.ARCHIVED,  A.APPLIED,  A.STATUS,  A.DELETED,  A.COMPLETION\_TIME,  BLOCKS\*BLOCK\_SIZE/1024/1024 AS MBs  FROM  V$ARCHIVED\_LOG A  ORDER BY  A.STANDBY\_DEST,  A.SEQUENCE#; |

|  |
| --- |
| SELECT  A.FILE#,  A.NAME,  TO\_CHAR(A.CHECKPOINT\_CHANGE#) AS SCN,  TO\_CHAR(A.CHECKPOINT\_TIME,'YYYYMMDD HH24:MI:SS')  FROM  V$DATAFILE A ; |

其中，主、备库V$ARCHIVED\_LOG的序号对得上，只能证明主库的归档日志有同步到备库，不能证明数据是同步的。每个归档日志都有一段SCN范围。对比主、备库数据文件的SCN才知道是否真正做到数据同步。

## 每个归档日志对应的SCN范围

|  |
| --- |
| SELECT A.NAME, A.SEQUENCE#, A.FIRST\_CHANGE#, A.NEXT\_CHANGE# FROM V$ARCHIVED\_LOG A WHERE A.SEQUENCE# > '54121' |



## 对备库启用read only方式

|  |
| --- |
| SQL> alter database open;  alter database open  \*  **ERROR** at line 1:  ORA-10456: cannot open standby database; media recovery session may be in progress  SQL> alter database recover managed standby database cancel; Database altered.  SQL> alter database open;  Database altered.  SQL> alter database recover managed standby database using current logfile disconnect ;  Database altered. |

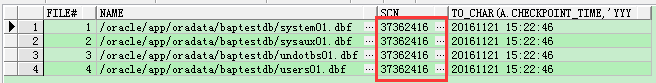
## 对不同步的情况进行处理(日志缺失比较多)

(1) 查主备库的数据文件差多少个SCN

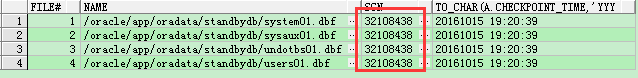
|  |
| --- |
| SELECT  A.FILE#,  A.NAME,  TO\_CHAR(A.CHECKPOINT\_CHANGE#) AS SCN,  TO\_CHAR(A.CHECKPOINT\_TIME,'YYYYMMDD HH24:MI:SS')  FROM  V$DATAFILE A ; |

其中，

主库：



备库：



差5253978个SCN号。

(2) 查看自失去同步的SCN起，主库有无添加新的数据文件。如果主库有添加新的数据文件，需要手工在备库也进行添加。

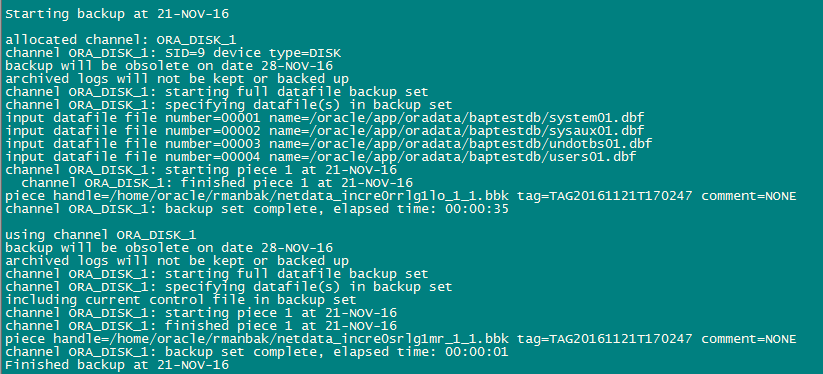
|  |
| --- |
| SELECT FILE#,NAME FROM V$DATAFILE WHERE CREATION\_CHANGE#> =32108438; |

(3) 停止备库的日志应用

|  |
| --- |
| ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL; |

(4) RMAN将主库上差异的部分备份出来

|  |
| --- |
| $> mkdir /home/oracle/rmanbak  $> rman target / nocatalog;  # 生成增量  RMAN> backup device type disk incremental from scn 73559400965 database format '/home/oracle/rmanbak/netdata\_incre%U.bbk';  # 生成控制文件  RMAN> ALTER DATABASE CREATE standby controlfile AS '/home/oracle/rmanbak/standby.ctl'; |



(5) 将控制文件备出来

|  |
| --- |
| SQL> ALTER DATABASE CREATE standby controlfile AS '/home/oracle/rmanbak/standby.ctl'; |

(6) 将主库从RMAN导出的增量数据同步到备库

备库用Oracle用户建立相应目录

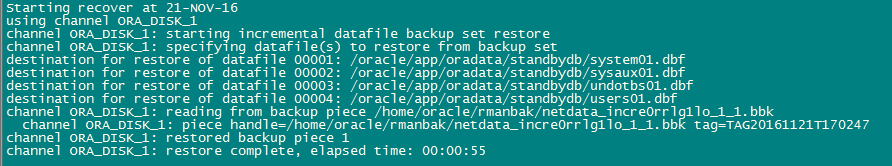
|  |
| --- |
| #> mkdir /home/oracle/rmanbak |

主库将数据传到备库

|  |
| --- |
| #> scp \* 10.249.61.23:/home/oracle/rmanbak |

(7) 在备库应用增量数据

|  |
| --- |
| RMAN> CATALOG START WITH '/home/oracle/rmanbak';  RMAN> RECOVER DATABASE NOREDO; |

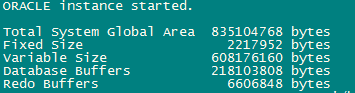


(8) 在备库恢复控制文件

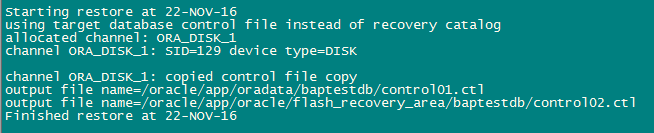
|  |
| --- |
| SQL> shutdown abort; |

C:\Users\Administrator\AppData\Roaming\Tencent\Users\1605396217\QQ\WinTemp\RichOle\4_K2URQ}5EQ3K`1SX%LR27Q.png

|  |
| --- |
| SQL> startup nomount; |



|  |
| --- |
| $> rman target/  RMAN> RESTORE STANDBY CONTROLFILE FROM '/home/oracle/rmanbak/standby.ctl'; |



(9) 启动数据库

|  |
| --- |
| SQL> ALTER DATABASE MOUNT; |

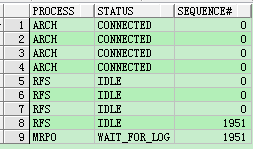
|  |
| --- |
| SQL> ALTER DATABASE OPEN; |

(10) 恢复日志应用

|  |
| --- |
| SQL> alter database recover managed standby database using current logfile disconnect ; |

(11) 查看备库的DataGuard相关进程

|  |
| --- |
| SQL> SELECT PROCESS,STATUS,SEQUENCE# FROM V$MANAGED\_STANDBY; |



ARCH 进程就是负责在重做日志文件切换后将已经写满的重做日志文件复制到归档日志文件中，以防止循环写入重做日志文件时将其覆盖。

RFS日志接收进程；

MRP0是管理恢复进程；

也就是说，ARCH进行redo log的归档，然后RFS就接收这个归档的日志，MRP0就进行这个归档日志的恢复，三者缺一不可。

|  |
| --- |
| 二、缺失较少日志，可找回日志处理：   --指定路径恢复数据文件： run { set newname for datafile 25 to '/oracle/oradata/dgsms1/TBS\_HGQD25.dbf'; restore datafile 25; recover DATABASE noredo; } |

## DataGuard下sys用户的密码的修改

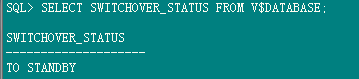
Oracle Dataguard环境的日志传输安全机制依靠Oracle生产库的密码文件，即sys用户的密码。在主库修改的sys密码不能自动同步到备库。因此，在主库不能随随便便修改sys用户的密码，而是要遵循一定的步骤。

|  |
| --- |
| 第1步：主库执行，做日志同步，SQL> alter system archive log current ;  第2步：备库执行，停库，SQL> shutdown immediate;  第3步：主库执行，备份主库的sys密码文件，  #> cp $ORACLE\_HOME/dbs/orapworcl1 $ORACLE\_HOME/dbs/orapworcl1.bak  第3步：主库执行，修改主库sys密码，SQL > alter user sys identified by newpasswd;  第4步：主库执行，将主库密码文件同步到备库，要检查主库和备库的密码文件是不是同一个。  #> cp $ORACLE\_HOME/dbs/orapworcl1 $ORACLE\_HOME/dbs/orapworcl1.new  #> scp orapworcl1.new [oracle@192.168.1.6:/oracle/product/10.2.0/dbs](mailto:oracle@192.168.1.6:/oracle/product/10.2.0/dbs)  第5步：备库操作，备份备库的旧的sys密码文件  #> cp $ORACLE\_HOME/dbs/orapworcl1 $ORACLE\_HOME/dbs/orapworcl1.bak  第6步：备库操作，应用从主库同步过来的sys密码文件  # mv orapworcl1.new orapworcl1  第7步：备库操作：启动到nomount状态，SQL> startup nomount  第8步：备库操作：已备库角色启动，SQL> alter database mount standby database;  第9步：备库操作：切换到read only，SQL> alter database open read only;  第10步：备库操作：开启dg，SQL> alter database recover managed standby database using current logfile disconnect from session; |

## Switchover方式切换DataGuard主备库

(1) 查看主库的状态

|  |
| --- |
| SQL> SELECT SWITCHOVER\_STATUS FROM V$DATABASE; |



## 估算SCN差异的大小

select sum(blocks) from v$backup\_datafile;  可以通过这个表中备份信息估算   ;

# Golden Gate：Oracle -> Oracle

# Golden Gate：MySQL -> Oracle

# Golden Gate：Oracle -> Hadoop

# 备份与恢复

## 热备还是冷备？

## 每周备份几次？

## 全备还是增备？

## 备份在磁盘还是磁阵还是磁带？

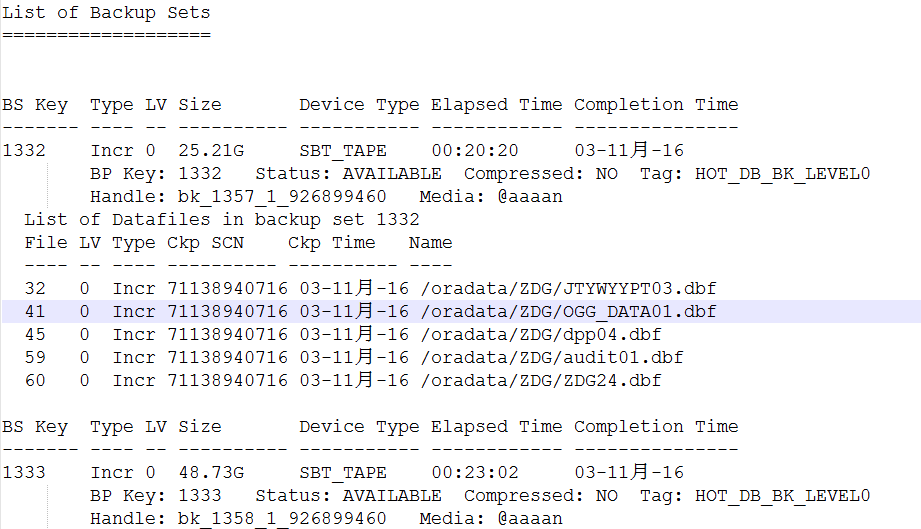
## 数据的保留时间是多久？

# RMAN用于备份

## RMAN备份结果的路径

## 查看可用的RMAN全库备份

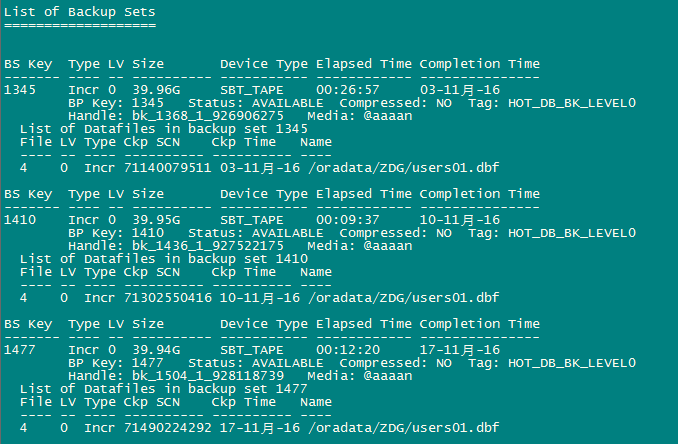
|  |
| --- |
| RMAN> LIST BACKUP OF DATABASE; |



可以看出，这里一个Backup Set只包含几个数据文件。但把明显是属于一个全备的Backup Set的size加起来，刚好就是全库的大小。

## 查看可用的RMAN某个数据文件的备份

|  |
| --- |
| RMAN> LIST BACKUP OF DATAFILE '/oradata/ZDG/users01.dbf'; |



## 不压缩的RMAN备份数据库

|  |
| --- |
| RMAN> BACKUP DATABASE; |

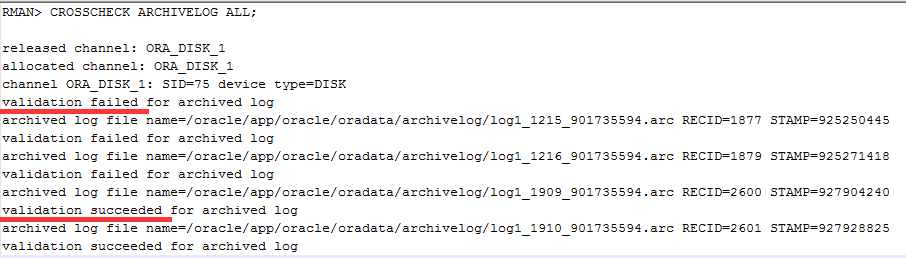


## 压缩的RMAN备份数据库

|  |
| --- |
| RMAN> BACKUP AS COMPRESSED BACKUPSET DATABASE; |

## 检查失效、缺失的归档日志

|  |
| --- |
| RMAN> CROSSCHECK ARCHIVELOG ALL; |



## 基于时间的备份保留策略

|  |
| --- |
| RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF n DAYS; |

n=大于0的正整数

执行该命令后，RMAN将始终保留那些将数据库恢复到n天前的状态时需要用到的备份，比如，恢复时间段被设置为7天，那么各个数据文件的备份必须满足如下条件：

SYSDATE-(SELECT CHECKPOINT\_TIME FROM V$DATAFILE)>=7

任何不满足上述条件的备份都将被RMAN废弃并可通过DELETE OBSOLETE命令删除。

通过REPORT OBSOLETE;可以看到过期的备份。

## RMAN参数的配置

(1) CONFIGURE RETENTION POLICY配置备份保留策略

基于时间：

CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF n DAYS;

基于冗余数量：

CONFIGURE RETENTION POLICY TO REDUNDANCY n;

也可以取消备份保留策略：

CONFIGURE RETENTION POLICY TO NONE;

(2) CONFIGURE BACKUP OPTIMIZATION配置备份优化

备份优化，包括OFF和ON两个状态

打开备份优化：

CONFIGURE BACKUP OPTIMIZATION ON;

关闭备份优化：

CONFIGURE BACKUP OPTIMIZATION OFF;

(3) CONFIGURE DEFAULT DEVICE TYPE 配置IO设备类型

RMAN支持的IO设备类型有两种：磁盘(DISK)和磁带(SBT)，默认情况下为磁盘。

使用磁盘设备：

CONFIGURE DEFAULT DEVICE TYPE TO DISK;

使用磁带设置：

CONFIGURE DEFAULT DEVICE TYPE TO SBT;

在这里需要注意的一点是：如果IO设备发生变化，相关配置项也需要修改。例如：

RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 2;

(4) CONFIGURE CONTROLFILE AUTOBACKUP配置控制文件自动备份

是否自动备份，包含两个状态：OFF和ON

打开自动备份

CONFIGURE CONTROLFILE AUTOBACKUP ON

禁止自动备份

CONFIGURE CONTROLFILE AUTOBACKUP OFF

同时可以通过如下配置指定备份的控制格式，路径。例如：

CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/oracle/app/oracle/oradata/%F';

在备份期间，将产生一个控制文件的快照，用于控制文件的读一致性，这个快照可以通过如下配置：

CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/oracle/app/oracle/oradata/SNCFJSSWEB.ORA';

(5) CONFIGURE DEVICE TYPE 设置并行备份

RMAN支持并行备份与恢复，也可以在配置中指定默认的并行程度。例如：

CONFIGURE DEVICE TYPE DISK PARALLELISM 2;

指定在以后的备份与恢复中，将采用并行度为2，同时开启2个通道进行备份与恢复，当然也可以在run中指定通道来决定备份与恢复的并行程度。

并行的数目决定了开启通道的个数。如果指定了通道配置，将采用指定的通道，如果没有指定通道，将采用默认通道配置。有点儿绕是吧，我来给你白话一把。

默认情况下，自动分配通道的并行度为1，如果你通过设置PARALLELISM设置了并行通道为2，那么在run块中，如果你没有单独通过ALLOCATE CHANNEL命令指定通道，它会默认使用2条并行通道，如果你在run命令块中指定了数个ALLOCATE CHANNEL，那么rman在执行备份命令时会以你设置的channel为准，而不管configure中配置了多少个并行通道。需要注意的一点是，在backup命令中有一个FILESPERSET参数，该参数是指rman建立的每个备份集中所能包含的备份片段(即磁盘文件)的最大数，该参数默认值为64，如果在执行backup命令时没有指定该参数值，那么rman会仅使用第一个通道来执行备份，其它通道将处于空闲状态。关于通道数与FILESPERSET值之间也有一个大小关系，逻辑稍显复杂这些就不多废话了，总之一条，filesperset值不要小于你设定的通道数。

(6) CONFIGURE DATAFILE BACKUP COPIES 设置备份文件冗余度

(7) CONFIGURE MAXSETSIZE 配置备份集的最大尺寸

该配置限制通道上备份集的最大尺寸。单位支持bytes,K,M,G。默认值是unlimited。

## 建立增量备份

建立增量备份也是相当简单，实质就是一个参数INCREMENTAL LEVEL=n，在执行BACKUP命令时加上即可，例如，建立一个增量级别0的全库备份：

|  |
| --- |
| RMAN> BACKUP INCREMENTAL LEVEL=0 DATABASE; |

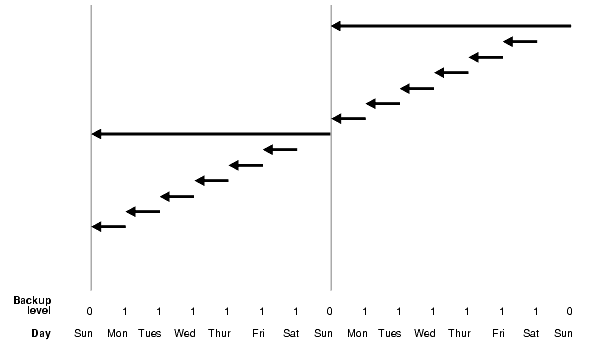
再例如，建立一个增量级别1的users01.dbf数据文件备份：

|  |
| --- |
| RMAN> BACKUP INCREMENTAL LEVEL=1 TABLESPACE SYSTEM DATAFILE ' /oracle/app/oracle/oradata/USERS01.DBF'; |

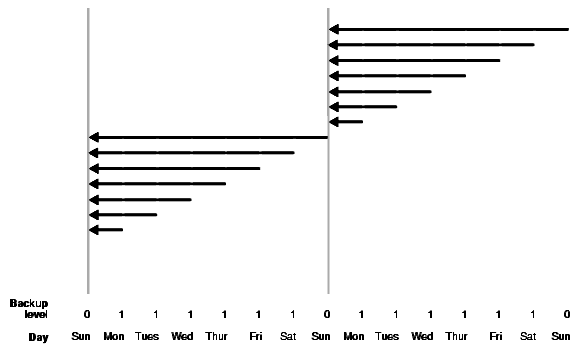
增量备份时分等级的，级别从0开始，一级一级递增，不过实际上用的最多的也就是0级和1级了，0级增量备份是后面级别的增量备份的 基础，0级备份实际上就是一个完全备份，与普通的完全备份唯一的不同点是0级备份可以作为其他级别增量备份的基础，而普通的完全备份是不能的。

从级别1开始，Oracle的增量备份分为差异增量备份和积累增量备份两种，其中差异增量备份备份的是自上一次同级别的差异备份或者是上一次更高级 别的备份完成之后的数据库发生改变的数据块；而积累增量备份则是备份的自上一次上一级增量备份完成以来数据库发生改变的数据块。

以下是差异增量的示意图：



以下是累积增量的示意图



RMAN默认创建的增量备份是Differential方式，如果要建立Cumulative方式的增量备份，在执行BACKUP命令时显式指定即可，例如：

|  |
| --- |
| RMAN> BACKUP INCREMENTAL LEVEL=2 CUMULATIVE DATABASE; |

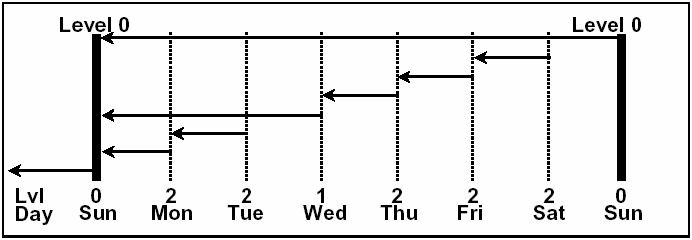
## Full、0级增量、1级增量、2级增量

在differential（差异）模式下（缺省）

0=全备

1=上次全备或LV1至今的增量（无论是否有，有N次 LV2的备份，一律从上次lv1或0至今的变化） 0/1 ~ NOW

2=上次任意级别备份至今的增量（任意一次备份至今） 0/1/2 ~ NOW

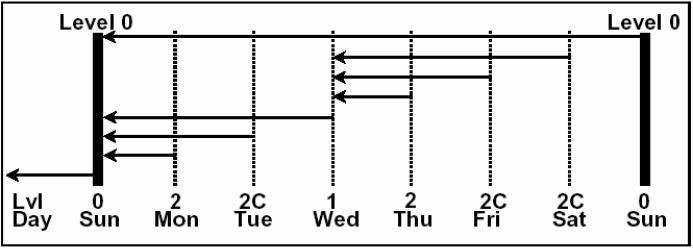


在cumulative（累积的）模式下，

0=全备

1=上次全备至今的增量（无论是否有，有N次LV1的备份，一律从全备至今的变化） 0 ~ NOW

2=上次0或1级别备份至今的增量（2次LV2间的累积，也汇总备份至今）0/1 ~ NOW （differential下的LV1）



## 建立RMAN定时备份策略

(1) 开启Block Change Tracking选项

|  |
| --- |
| ALTER DATABASE ENABLE BLOCK CHANGE TRACKING USING FILE '/data/hisdata/block\_change\_track.log'; |

(2) 设计RMAN备份策略

|  |
| --- |
| RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;  RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;  RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/data/backup/Data\_%T\_%U';  RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/data/backup/Control\_T%\_%F';  RMAN> CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/data/backup/snapcf\_%T\_%U.f';  每周一上午9点做一次全备，每周二~周六上午9点做一次差异增量备份。 |

(3) 每周0级备份的RMAN脚本

|  |
| --- |
|  |

(4) 每周1级增备的RMAN脚本

|  |
| --- |
|  |

1. 每周2级增备的RMAN脚本

## 查看备份执行的结果

|  |
| --- |
| SELECT  S.STATUS AS BACKUP\_STATUS,  B.INPUT\_TYPE AS BACKUP\_TYPE,  TO\_CHAR(B.START\_TIME,  'YYYY-MM-DD HH24:MI:SS') AS TOTAL\_BEGIN\_BACKUP\_TIME, TO\_CHAR(B.END\_TIME, 'YYYY-MM-DD HH24:MI:SS') AS TOTAL\_END\_BACKUP\_TIME, TRUNC(B.ELAPSED\_SECONDS/60,0) AS BACKUP\_USE\_TYPE, B.INPUT\_BYTES\_PER\_SEC\_DISPLAY AS IN\_SEC\_PER\_S, B.OUTPUT\_BYTES\_PER\_SEC\_DISPLAY AS OUT\_PER\_S, TRUNC((S.END\_TIME-S.START\_TIME)\*24\*60,0) SINGLE\_FILE\_USE\_TIME, TO\_CHAR(S.START\_TIME, 'YYYY-MM-DD HH24:MI:SS') AS BEGIN\_BACKUP\_TIME, TO\_CHAR(S.END\_TIME, 'YYYY-MM-DD HH24:MI:SS') AS END\_BACKUP\_TIME, S.OPERATION AS OPERATION, TRUNC(S.INPUT\_BYTES/1024/1024,2) AS INPUT\_MB, TRUNC(S.OUTPUT\_BYTES/1024/1024,2) AS OUTPUT\_MB, S.OBJECT\_TYPE AS OBJECT\_TYPE, S.MBYTES\_PROCESSED AS PERCENT, S.OUTPUT\_DEVICE\_TYPE AS DEVICE\_TYPE  FROM  V$RMAN\_STATUS S,  V$RMAN\_BACKUP\_JOB\_DETAILS B  WHERE  TO\_CHAR(S.START\_TIME, 'YYYY-MM-DD HH24:MI:SS') < TO\_CHAR(SYSDATE,'YYYY-MM-DD HH24:MI:SS')  AND TO\_CHAR(S.END\_TIME, 'YYYY-MM-DD HH24:MI:SS') > TO\_CHAR(SYSDATE-7,'YYYY-MM-DD HH24:MI:SS')  AND S.COMMAND\_ID=B.COMMAND\_ID  ORDER BY  S.START\_TIME DESC |

## RMAN执行全备的脚本

|  |
| --- |
| #!/bin/bash  # incremental level 0 backup script  source ~/.bash\_profile  $ORACLE\_HOME/bin/rman target / log=/data/ora\_backup/rmanscript/fullbackup\_orcl1\_ `date +%y%m%d%H`.log <<EOF  CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;  run  {  crosscheck backup;  delete noprompt obsolete;  delete noprompt expired backup;  backup as compressed backupset incremental level 0 format '/data/ora\_backup/bak/ fullbak/fullbackup\_%T\_%d\_%s\_%p.bus' database;  sql 'alter system archive log current';  backup format='/data/ora\_backup/bak/fullbak/fullbackup\_%T\_%d\_%s\_%p.ctl' current controlfile;  }  EOF |

## RMAN执行增备的脚本

|  |
| --- |
| #!/bin/bash  # incremental level 1 backup script  source ~/.bash\_profile  current\_day=`date +%y%m%d`  incr\_day=incr$current\_day  mkdir /data/ora\_backup/bak/incr1/$incr\_day  $ORACLE\_HOME/bin/rman target / log=/data/ora\_backup/bak/incr1/$incr\_day/incr1\_orcl1\_`date +%y%m%d%H`.log <<EOF  CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;  run  {  crosscheck backup;  delete noprompt obsolete;  delete noprompt expired backup;  backup as compressed backupset incremental level 1 format '/data/ora\_backup/bak/incr1/$incr\_day/incr1\_%T\_%d\_%s\_%p.bus' database;  sql 'alter system archive log current';  backup format='/data/ora\_backup/bak/incr1/$incr\_day/incr1\_%T\_%d\_%s\_%p.ctl' current controlfile;  }  EOF |

# RMAN用于恢复

# 权限控制

## 回收DBA权限

|  |
| --- |
| REVOKE DBA,EXP\_FULL\_DATABASE, IMP\_FULL\_DATABASE FROM DB133\_HIS; |

# 关键配置

## Enterprise Manager

# Oracle卸载

## 卸载Oracle

(1) 进入数据库并把数据库停掉（shutdown immediate）

(2) 停止Listener进程（lsnrctl stop）

(3) 进入$ORACLE\_HOME/deinstall/下，运行deinstall脚本删除oracle

(4) 删除/etc下的oraInst.loc、oratab和/opt下的ORCLfmap(rm -rf 文件名)

(5) 删除/usr/local/bin里面的所有文件（rm -rf \*）

(6) 删除/tmp下的所有用户是oracle的文件（不要删错）

(7) 删除oracle的安装目录/data/database（每个人的安装目录放置的位置可能不一样，要自己找）

(8) 把oracle的用户级dba、oinstall用户组删除（userdel oracle；groupdel dba；groupdel oinstall）

(9) 用户删除之后还可以登录一下看看是否已经删除（su - oracle）

# 数据库对象

## 失效对象确认及确认

存储过程往往作为业务逻辑的一部分，如果存储过程失效，将会影响正常业务。另外，表、视图的失效，也会影响数据的存储和检索。需要定期分析数据库中的失效对象，如果有必要，对存储过程、视图等进行重新的编译。

|  |
| --- |
| SELECT  CREATED,  OWNER,  OBJECT\_NAME,  OBJECT\_TYPE,  STATUS,  TIMESTAMP  FROM  DBA\_OBJECTS  WHERE  STATUS = 'INVALID'  ORDER BY  OWNER,  OBJECT\_TYPE; |

通过重新编译，可以

附：

查找Procedure、Package、Function的定义内容

SELECT

TEXT

FROM

DBA\_SOURCE

WHERE

OWNER = 'CONGXING'

AND NAME = 'F\_CREATE\_WEEK'

ORDER BY

LINE;

对失效对象进行重新编译的逻辑

DECLARE

V1 VARCHAR2(500);

BEGIN

FOR K IN (SELECT CREATED,OWNER,OBJECT\_NAME,OBJECT\_TYPE,STATUS,TIMESTAMP FROM DBA\_OBJECTS WHERE STATUS = 'INVALID')

LOOP BEGIN

V1:='ALTER '|| K.OBJECT\_TYPE ||' '|| K.OWNER ||'.'|| K.OBJECT\_NAME||' COMPILE';

***DBMS\_OUTPUT.PUT\_LINE***('Execute ' || V1);

EXECUTE IMMEDIATE V1;

***DBMS\_OUTPUT.PUT\_LINE***(V1 || ' FINISHED');

EXCEPTION

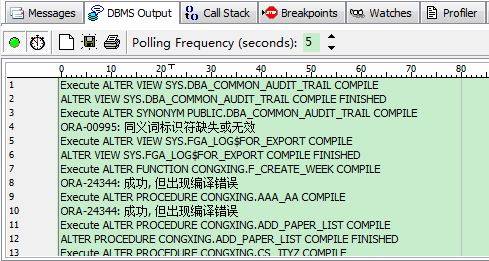
WHEN OTHERS THEN

***DBMS\_OUTPUT.PUT\_LINE***(SQLERRM);

END;

END LOOP;

END;



**成效：**

**133库：修复invalid对象213个。**

**130库：修复invalid对象17个。**

**44库：修复invalid对象3个。**

## 表空间

### 默认的永久表空间和临时表空间

|  |
| --- |
| SELECT  \*  FROM  DATABASE\_PROPERTIES  WHERE  PROPERTY\_NAME IN ('DEFAULT\_TEMP\_TABLESPACE','DEFAULT\_PERMANENT\_TABLESPACE'); |

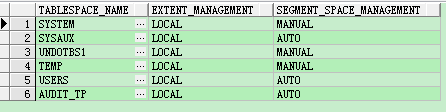
(YJU0O[Q(N8RMANSKF]2M20

如果没有设置默认的永久表空间和默认的临时表空间，那么用户的永久数据和临时数据将会默认创建在SYSTEM表空间，这个是会影响性能的。

经检查，目前130、133、44库均已设置了默认的永久表空间和默认的临时表空间。

### 表空间的空间管理方式和扩展方式

|  |
| --- |
| SELECT  TABLESPACE\_NAME,  EXTENT\_MANAGEMENT,  SEGMENT\_SPACE\_MANAGEMENT  FROM  DBA\_TABLESPACES; |



查看各个表空间，空间管理方式应该都是本地管理(LOCAL)而不是字典管理(DIRECTORY)，因为字典管理效率很差，基本是要废弃的。

除了SYSTEM、默认的UNDO表空间和默认的临时表空间，其他空间的分区扩展方式选择自动是比较好的，会节省大量维护的时间。不过，也有另外一种观点是，不要设置AUTO，而是应该设置一个比较大的值。以免把表空间撑得非常大。例如UNDO表空间和临时表空间，如果设成AUTO方式，不定期shrink的话，能把磁盘空间吃光。

经检查，目前130、133、44库设置合理。

### UNDO表空间的管理

#### UNDO表空间扩展方式

UNDO表空间的扩展方式使用MANUAL的方式比AUTO的方式可能好一点

#### UNDO表空间的大小设置

UNDO表空间的大小设置多大合适呢？业界给的估算公式如下：

UndoSpace = [UR \* (UPS \* DBS)] + (DBS \* 24)，其中：

UR：以秒为单位的UNDO\_RETENTION，通过SHOW PARAMETER UNDO\_RETENTION;查看，

以秒为单位。

UPS：每秒生成的还原数据块的数量，这个最好在业务高峰期看。

DBS：db\_block\_size，目前是8K。

|  |
| --- |
| SELECT  UR AS RETENTION\_SECONDS,  UPS AS UNDO\_BLOCKS,  DBS AS DBBLOCK\_BYTES,  ((UR \* (UPS \* DBS)) + (DBS \* 24))/1024/1024 AS "Need MB"  FROM  (SELECT value AS UR FROM v$parameter WHERE name = 'undo\_retention'),  (SELECT (SUM(undoblks)/SUM(((end\_time -begin\_time)\*86400))) AS UPS FROM v$undostat),  (SELECT value AS DBS FROM v$parameter WHERE name = 'db\_block\_size'); |

经检查，目前130、133、44库均已预留了30G的UNDO表空间，RETENTION的时间为15分钟，是远远超出需求的，RETENTION的时间还可以延长。

#### UNDO表空间参数

UNDO\_RETENTION，如果UNDO表空间是AUTO EXTENT，UNDO\_RETENTION必须要设置。

UNDO信息至少要满足UNDO\_RETENTION的需求。如果UNDO表空间是FIX，又没有设置GUARANTEE，那么UNDO\_RETENTION不一定能保证，只能说尽量保证。但是如果设置了GUARANTEE，那么UNDO\_RETENTION就能够保证。当UNDO空间不够，DML操作会被挂起。

|  |
| --- |
| SELECT  TABLESPACE\_NAME,  RETENTION  FROM  DBA\_TABLESPACES  WHERE  TABLESPACE\_NAME = 'UNDOTBS1'; |

如果查出是GUARANTEE，那么就需要修改为非GUARANTEE了：

|  |
| --- |
| ALTER TABLESPACE UNDOTBS1 RETENTION NOGUARANTEE; |

经检查，目前130、133、44库均已设置为NOGUARANTEE。

### TEMP表空间的管理

### 表空间的扩容

#### 寻找需要扩容的表空间

A: 分析各表空间的容量：

SELECT

C.TABLESPACE\_NAME TABLESPACE\_NAME,

ROUND(A.BYTES / 1048576, 2) TOTAL\_SIZE,

ROUND((A.BYTES - B.BYTES) / 1048576, 2) USED\_SIZE,

ROUND(B.BYTES / 1048576, 2) FREE\_SIZE,

ROUND(B.BYTES / A.BYTES \* 100, 2) || '%' FREE\_PCT

FROM

(

SELECT

TABLESPACE\_NAME,

SUM(A.BYTES) BYTES

FROM

SYS.DBA\_DATA\_FILES A

GROUP BY

TABLESPACE\_NAME

) A,

(

SELECT

A.TABLESPACE\_NAME,

NVL(SUM(B.BYTES), 0) BYTES

FROM

SYS.DBA\_DATA\_FILES A,

SYS.DBA\_FREE\_SPACE B

WHERE

A.TABLESPACE\_NAME = B.TABLESPACE\_NAME(+)

AND A.FILE\_ID = B.FILE\_ID(+)

GROUP BY A.TABLESPACE\_NAME

) B,

SYS.DBA\_TABLESPACES C

WHERE

A.TABLESPACE\_NAME = B.TABLESPACE\_NAME(+)

AND A.TABLESPACE\_NAME = C.TABLESPACE\_NAME

ORDER BY ROUND(B.BYTES / 1048576, 2);

B: 分析各表空间是否自动扩展

SELECT

D.FILE\_NAME,

D.TABLESPACE\_NAME,

(D.MAXBYTES-D.BYTES)/1048576||'M'

FREE\_SIZE,

D.STATUS,

D.AUTOEXTENSIBLE

FROM

DBA\_DATA\_FILES D

WHERE

D.TABLESPACE\_NAME IN

(

SELECT TABLESPACE\_NAME FROM SYS.DBA\_TABLESPACES

);

C: 对剩余容量较少，且表空间并非自动扩展的表空间，需要执行扩容操作。

#### 表空间扩容时间预测

#### 执行表空间扩容

A：添加固定大小的数据文件：

alter tablesapce tablespace\_name add datafile '/xxx/xxx.dbf' size xxxxM;

例如：

alter TABLESPACE TBS\_HGQD add datafile '/backup/oradata/TBS\_HGQD10.dbf' size 10240M

B:添加自动扩展的数据文件：

alter TABLESPACE tablespace\_name add datafile '/xxx/xxx.dbf' size xxxM AUTOEXTEND ON NEXT xxxM MAXSIZE xxxG;

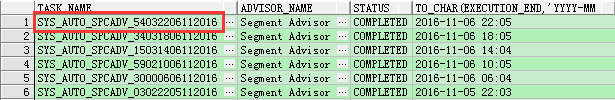
例如：

alter TABLESPACE TBS\_HGQD add datafile '/backup/oradata/TBS\_HGQD10.dbf' size 10240M AUTOEXTEND ON NEXT 100M MAXSIZE 20G;

## Segment Advisor报告查看

### 查看系统在维护窗口已执行的Segment分析

|  |
| --- |
| SELECT  TASK\_NAME,  ADVISOR\_NAME,  STATUS,  TO\_CHAR(EXECUTION\_END, 'YYYY-MM-DD HH24:MI')  FROM  DBA\_ADVISOR\_EXECUTIONS  WHERE  ADVISOR\_NAME = 'Segment Advisor'  ORDER BY  EXECUTION\_END DESC; |

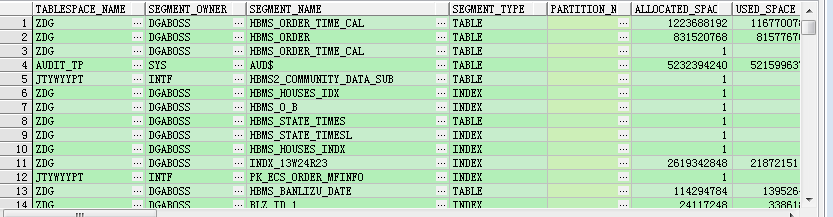


### 手工执行Segment Advisor

|  |
| --- |
| DECLARE  MY\_TASK\_ID NUMBER;  OBJ\_ID NUMBER;  MY\_TASK\_NAME VARCHAR2(100);  MY\_TASK\_DESC VARCHAR2(500);  BEGIN  *--运行任务名，可以任意指定，不过建议为有意义的名称*  MY\_TASK\_NAME :='advisor\_test tab Advice';  *--运行任务描述，可以任意指定，不过建议为有意义的描述*  MY\_TASK\_DESC :='Manual Segment Advisor Run';    *-----step 1*  */\* 创建一个段顾问任务 \*/*  DBMS\_ADVISOR.CREATE\_TASK(  *--运行段顾问任务这个参数必须指定为Segment Advisor*  ADVISOR\_NAME => 'Segment Advisor',  TASK\_ID => MY\_TASK\_ID,  TASK\_NAME => MY\_TASK\_NAME,  TASK\_DESC =>MY\_TASK\_DESC);    *-----STEP 2*  */\* 为这个任务分配一个对象 \*/*  DBMS\_ADVISOR.CREATE\_OBJECT(  TASK\_NAME=>MY\_TASK\_NAME,  *--指定对象级别，如果为表对象则为'TABLE'，如果为表空间级别则为'TABLESPACE'*  OBJECT\_TYPE=>'TABLE',  *---如果在表对象级别运行，这个属性为用户名，表空间级别这个属性为表空间名字*  ATTR1=>'DBMON',  *---如果在表对象级别运行，这个属性为表名，表空间级别这个属性为NULL*  ATTR2 => 'ADVISOR\_TEST',  ATTR3 => NULL,  ATTR4=>NULL,  ATTR5=>NULL,  OBJECT\_ID=>OBJ\_ID);    *-----step 3*  */\* 设置任务参数 \*/*  DBMS\_ADVISOR.SET\_TASK\_PARAMETER(  TASK\_NAME => MY\_TASK\_NAME,  */\* 设置段顾问运行参数"ecommend\_all"的值,为TRUE则为所有类型的对象*  *的生成建议，为FALSE则仅生成与空间相关的建议 \*/*  */\* 另一个滚问运行参数"time\_limit",制定顾问运行的时间限制，默认值*  *为无限制 \*/*  PARAMETER=>'recommend\_all', *---*  value=>'TRUE');    *-----step 4*  */\* 执行这个任务 \*/*  DBMS\_ADVISOR.EXECUTE\_TASK(MY\_TASK\_NAME);  END;  / |

### 查看Segment Advisor结果

|  |
| --- |
| SELECT  A.\*,  B.EXECUTION\_START,  B.EXECUTION\_END  FROM  (  SELECT  \*  FROM  TABLE(DBMS\_SPACE.ASA\_RECOMMENDATIONS(ALL\_RUNS=>'TRUE',SHOW\_MANUAL=>'TRUE',SHOW\_FINDINGS=>'FALSE'))  ) A LEFT JOIN DBA\_ADVISOR\_EXECUTIONS B  ON A.TASK\_ID = B.TASK\_ID  ORDER BY  B.EXECUTION\_END DESC; |



### ALTER TABLE MOVE VS ALTER TABLE SHRINK SPACE

|  |  |  |
| --- | --- | --- |
|  | ALTER TABLE MOVE | ALTER TABLE SHRINK SPACE |
| 共同点 | 1、收缩段  2、消除部分行迁移  3、消除空间碎片  4、使数据更紧密 | |
| 使用 | 1. alter table t1 enable ROW MOVEMENT;  2. shrink 操作,例如：  (1) 只整理碎片，不回收空间  alter table TABLE\_NAME shrink space compact;  (2) 整理碎片并回收空间  alter table TABLE\_NAME shrink space;  (3) 整理碎片回收空间 并连同表的级联对象一起整理(比如索引)  alter table TABLE\_NAME shrink space cascade;  (4) 针对分区表  alter table ticket modify  PARTITION P28071 shrink space cascade  3. alter table t1 disable ROW MOVEMENT; |  |
| 特点 | segment shrink执行的两个阶段： 1、数据重组(compact):  通过一系列insert、delete操作，将数据尽量排列在段的前面。在这个过程中需要在表上加RX锁，即只在需要移动的行上加锁。  由于涉及到rowid的改变，需要enable row movement.同时要disable基于rowid的trigger.这一过程对业务影响比较小。  2、HWM调整：第二阶段是调整HWM位置，释放空闲数据块。  此过程需要在表上加X锁，会造成表上的所有DML语句阻塞。在业务特别繁忙的系统上可能造成比较大的影响。  注意：shrink space语句两个阶段都执行。 shrink space compact只执行第一个阶段。  如果系统业务比较繁忙，  可以先执行shrink space compact重组数据,然后在业务不忙的时候再执行shrink space降低HWM释放空闲数据块。 | 1、将一个table从当前的tablespace上移动到另一个tablespace上：  alter table t move tablespace tablespace\_name;  2. 在原来的表空间内部移动。  alter table table\_name move ;  3、来改变table已有的block的存储参数,如：  alter table t move storage (initial 30k next 50k);  4、 另外，move操作也可以用来解决table中的行迁移的问题。 |
| 优点 | 1.可在线执行  2.可使用参数cascade，同时收缩表上的索引  3.执行后不会导致索引失效  4.不占用额外的空间 |  |
| 缺点 | Shrink 操作需满足表空间是本地管理和自动段空间管理（10g、[11g](http://www.itpub.net/thread-1318149-1-1.html)默认就是这样），  以下情况不能用shrink：  1.IOT索引组织表  2.用rowid创建的[物化视图](http://www.itpub.net/thread-1308625-1-1.html)的基表  3.带有函数索引的表  4.SECURE FILE 类型的大对象  5.压缩表 | 1、table上的index需要rebuild：  move操作后，数据的rowid发生了改变，我们知道，index是通过rowid来fetch数据行的，所以，table上的index是必须要rebuild的。  alter index index\_name rebuild online；  2、move时对table的锁定  当我们对table进行move操作时，查询v$locked\_objects视图可以发现，table上加了exclusive lock  3、关于move时空间使用的问题：  当我们使用alter table move来降低table的HWM时，有一点是需要注意的，这时，当前的tablespace中需要有1倍于table的空闲空间以供使用。 |
| 区别 | 1、move后，表在表空间中的位置肯定会变，可能前移也可能后移，一般来说如果该表前面的表空间中有足够空间容纳该表，则前移，否则后移。  2、hrink后，表在表空间中的位置肯定不变，也就是表的段头位置不会发生变化。  3、Move会移动高水位，但不会释放申请的空间，是在高水位以下(below HWM)的操作。  4、shrink space 同样会移动高水位，但也会释放申请的空间，是在高水位上下(below and above HWM)都有的操作。  5、使用move时，会改变一些记录的ROWID，所以MOVE之后索引会变为无效，需要REBUILD。 6、使用shrink space时，索引会自动维护。如果在业务繁忙时做压缩，可以先shrink space compact，来压缩数据而不移动HWM，等到不繁忙的时候再shrink space来移动HWM。  7、shrink可以单独压缩索引，alter index xxx shrink space来压缩索引。另外、压缩表时指定Shrink space cascade会同时压缩索引 | |

## 段分析

### 非ASSM管理的段分析

|  |
| --- |
| SET DEFINE OFF  *-- Call Syntax : EXEC Show\_Space('Tablename');*  *-- Requirements : SET SERVEROUTPUT ON*  CREATE OR REPLACE PROCEDURE DGLJW\_NOTASSM\_SHOW\_SPACE  (  P\_SEGNAME IN VARCHAR2,  P\_OWNER IN VARCHAR2 DEFAULT USER,  P\_TYPE IN VARCHAR2 DEFAULT 'TABLE',  P\_PARTITION IN VARCHAR2 DEFAULT NULL  )  *-- THIS PROCEDURE USES AUTHID CURRENT USER SO IT CAN QUERY DBA\_\**  *-- VIEWS USING PRIVILEGES FROM A ROLE AND SO IT CAN BE INSTALLED*  *-- ONCE PER DATABASE, INSTEAD OF ONCE PER USER THAT WANTED TO USE IT*  AUTHID CURRENT\_USER  AS  L\_FREE\_BLKS NUMBER;  L\_TOTAL\_BLOCKS NUMBER;  L\_TOTAL\_BYTES NUMBER;  L\_UNUSED\_BLOCKS NUMBER;  L\_UNUSED\_BYTES NUMBER;  L\_LASTUSEDEXTFILEID NUMBER;  L\_LASTUSEDEXTBLOCKID NUMBER;  L\_LAST\_USED\_BLOCK NUMBER;  L\_SEGMENT\_SPACE\_MGMT VARCHAR2(255);  L\_UNFORMATTED\_BLOCKS NUMBER;  L\_UNFORMATTED\_BYTES NUMBER;  L\_FS1\_BLOCKS NUMBER; L\_FS1\_BYTES NUMBER;  L\_FS2\_BLOCKS NUMBER; L\_FS2\_BYTES NUMBER;  L\_FS3\_BLOCKS NUMBER; L\_FS3\_BYTES NUMBER;  L\_FS4\_BLOCKS NUMBER; L\_FS4\_BYTES NUMBER;  L\_FULL\_BLOCKS NUMBER; L\_FULL\_BYTES NUMBER;    *-- INLINE PROCEDURE TO PRINT OUT NUMBERS NICELY FORMATTED*  *-- WITH A SIMPLE LABEL*    PROCEDURE P( P\_LABEL IN VARCHAR2, P\_NUM IN NUMBER )  IS  BEGIN  DBMS\_OUTPUT.PUT\_LINE( RPAD(P\_LABEL,40,'.') || TO\_CHAR(P\_NUM,'999,999,999,999') );  END;  BEGIN  *-- THIS QUERY IS EXECUTED DYNAMICALLY IN ORDER TO ALLOW THIS PROCEDURE*  *-- TO BE CREATED BY A USER WHO HAS ACCESS TO DBA\_SEGMENTS/TABLESPACES*  *-- VIA A ROLE AS IS CUSTOMARY.*  *-- NOTE: AT RUNTIME, THE INVOKER MUST HAVE ACCESS TO THESE TWO*  *-- VIEWS!*  *-- THIS QUERY DETERMINES IF THE OBJECT IS A ASSM OBJECT OR NOT*  BEGIN  EXECUTE IMMEDIATE  'SELECT TS.SEGMENT\_SPACE\_MANAGEMENT  FROM DBA\_SEGMENTS SEG, DBA\_TABLESPACES TS  WHERE SEG.SEGMENT\_NAME = :P\_SEGNAME  AND (:P\_PARTITION IS NULL OR  SEG.PARTITION\_NAME = :P\_PARTITION)  AND SEG.OWNER = :P\_OWNER  AND SEG.SEGMENT\_TYPE = :P\_TYPE  AND SEG.TABLESPACE\_NAME = TS.TABLESPACE\_NAME'  INTO L\_SEGMENT\_SPACE\_MGMT  USING P\_SEGNAME, P\_PARTITION, P\_PARTITION, P\_OWNER, P\_TYPE;  EXCEPTION  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE ( 'THIS MUST BE A PARTITIONED TABLE, USE P\_PARTITION => ');  RETURN;  END;    *-- IF THE OBJECT IS IN AN ASSM TABLESPACE, WE MUST USE THIS API*  *-- CALL TO GET SPACE INFORMATION, ELSE WE USE THE FREE\_BLOCKS*  *-- API FOR THE USER MANAGED SEGMENTS*  IF L\_SEGMENT\_SPACE\_MGMT = 'AUTO' THEN  DBMS\_SPACE.SPACE\_USAGE  (  P\_OWNER, P\_SEGNAME, P\_TYPE, L\_UNFORMATTED\_BLOCKS,  L\_UNFORMATTED\_BYTES, L\_FS1\_BLOCKS, L\_FS1\_BYTES,  L\_FS2\_BLOCKS, L\_FS2\_BYTES, L\_FS3\_BLOCKS, L\_FS3\_BYTES,  L\_FS4\_BLOCKS, L\_FS4\_BYTES, L\_FULL\_BLOCKS, L\_FULL\_BYTES, P\_PARTITION  );  P( 'UNFORMATTED BLOCKS ', L\_UNFORMATTED\_BLOCKS );  P( 'FS1 BLOCKS (0-25) ', L\_FS1\_BLOCKS );  P( 'FS2 BLOCKS (25-50) ', L\_FS2\_BLOCKS );  P( 'FS3 BLOCKS (50-75) ', L\_FS3\_BLOCKS );  P( 'FS4 BLOCKS (75-100)', L\_FS4\_BLOCKS );  P( 'FULL BLOCKS ', L\_FULL\_BLOCKS );  ELSE  DBMS\_SPACE.FREE\_BLOCKS  (  SEGMENT\_OWNER => P\_OWNER,  SEGMENT\_NAME => P\_SEGNAME,  SEGMENT\_TYPE => P\_TYPE,  FREELIST\_GROUP\_ID => 0,  FREE\_BLKS => L\_FREE\_BLKS);  P( 'FREE BLOCKS', L\_FREE\_BLKS  );  END IF;    *-- AND THEN THE UNUSED SPACE API CALL TO GET THE REST OF THE*  *-- INFORMATION*  DBMS\_SPACE.UNUSED\_SPACE  (  SEGMENT\_OWNER => P\_OWNER,  SEGMENT\_NAME => P\_SEGNAME,  SEGMENT\_TYPE => P\_TYPE,  PARTITION\_NAME => P\_PARTITION,  TOTAL\_BLOCKS => L\_TOTAL\_BLOCKS,  TOTAL\_BYTES => L\_TOTAL\_BYTES,  UNUSED\_BLOCKS => L\_UNUSED\_BLOCKS,  UNUSED\_BYTES => L\_UNUSED\_BYTES,  LAST\_USED\_EXTENT\_FILE\_ID => L\_LASTUSEDEXTFILEID,  LAST\_USED\_EXTENT\_BLOCK\_ID => L\_LASTUSEDEXTBLOCKID,  LAST\_USED\_BLOCK => L\_LAST\_USED\_BLOCK  );    P( 'TOTAL BLOCKS', L\_TOTAL\_BLOCKS );  P( 'TOTAL BYTES', L\_TOTAL\_BYTES );  P( 'TOTAL MBYTES', TRUNC(L\_TOTAL\_BYTES/1024/1024) );  P( 'UNUSED BLOCKS', L\_UNUSED\_BLOCKS );  P( 'UNUSED BYTES', L\_UNUSED\_BYTES );  P( 'LAST USED EXT FILEID', L\_LASTUSEDEXTFILEID );  P( 'LAST USED EXT BLOCKID', L\_LASTUSEDEXTBLOCKID );  P( 'LAST USED BLOCK', L\_LAST\_USED\_BLOCK );  END;  / |

### ASSM管理的段分析

|  |
| --- |
| CREATE OR REPLACE PROCEDURE DGLJW\_ASSM\_SHOW\_SPACE  (  P\_SEGNAME IN VARCHAR2,  P\_OWNER IN VARCHAR2 DEFAULT USER,  P\_TYPE IN VARCHAR2 DEFAULT 'TABLE'  )  AS  L\_FS1\_BYTES NUMBER;  L\_FS2\_BYTES NUMBER;  L\_FS3\_BYTES NUMBER;  L\_FS4\_BYTES NUMBER;  L\_FS1\_BLOCKS NUMBER;  L\_FS2\_BLOCKS NUMBER;  L\_FS3\_BLOCKS NUMBER;  L\_FS4\_BLOCKS NUMBER;  L\_FULL\_BYTES NUMBER;  L\_FULL\_BLOCKS NUMBER;  L\_UNFORMATTED\_BYTES NUMBER;  L\_UNFORMATTED\_BLOCKS NUMBER;    PROCEDURE P( P\_LABEL IN VARCHAR2, P\_NUM IN NUMBER )  IS  BEGIN  DBMS\_OUTPUT.PUT\_LINE( RPAD(P\_LABEL,40,'.') ||P\_NUM );  END;  BEGIN    DBMS\_SPACE.SPACE\_USAGE  (  SEGMENT\_OWNER => P\_OWNER,  SEGMENT\_NAME => P\_SEGNAME,  SEGMENT\_TYPE => P\_TYPE,  FS1\_BYTES => L\_FS1\_BYTES,  FS1\_BLOCKS => L\_FS1\_BLOCKS,  FS2\_BYTES => L\_FS2\_BYTES,  FS2\_BLOCKS => L\_FS2\_BLOCKS,  FS3\_BYTES => L\_FS3\_BYTES,  FS3\_BLOCKS => L\_FS3\_BLOCKS,  FS4\_BYTES => L\_FS4\_BYTES,  FS4\_BLOCKS => L\_FS4\_BLOCKS,  FULL\_BYTES => L\_FULL\_BYTES,  FULL\_BLOCKS => L\_FULL\_BLOCKS,  UNFORMATTED\_BLOCKS => L\_UNFORMATTED\_BLOCKS,  UNFORMATTED\_BYTES => L\_UNFORMATTED\_BYTES  );    P('FREE SPACE 0-25% BLOCKS:',L\_FS1\_BLOCKS);  P('FREE SPACE 25-50% BLOCKS:',L\_FS2\_BLOCKS);  P('FREE SPACE 50-75% BLOCKS:',L\_FS3\_BLOCKS);  P('FREE SPACE 75-100% BLOCKS:',L\_FS4\_BLOCKS);  P('FULL BLOCKS:',L\_FULL\_BLOCKS);  P('UNFORMATTED BLOCKS:',L\_UNFORMATTED\_BLOCKS);  END;  / |

## 归档日志分析

## 表

### 分区表改造

什么时候需要建立分区表，Oracle官方给的建议是：

1) 大于2G的表。

2) 包含大量历史数据的表，其中大量历史数据Read Only。

分区提供以下优点：

1) 由于将数据分散到各个分区中，减少了数据损坏的可能性；

2) 可以对单独的分区进行备份和恢复；

3) 可以将分区映射到不同的物理磁盘上，来分散IO；

4) 提高可管理性、可用性和性能。

### 识别行链接和行迁移

当一行的数据过长而不能插入一个单个数据块中时，可能发生两种事情：行链接(row chaining)或行迁移(row migration)。

行链接：

当第一次插入行时，由于行太长而不能容纳在一个数据块中时，就会发生行链接。在这种情况下，oracle会使用与该块链接的一块或多块数据块来容纳该行的数据。行连接经常在插入比较大的行时才会发生，如包含long, long row, lob等类型的数据。在这些情况下行链接是不可避免的。

行迁移

当修改不是行链接的行时，当修改后的行长度大于修改前的行长度，并且该数据块中的空闲空间已经比较小而不能完全容纳该行的数据时，就会发生行迁移。在这种情况下，Oracle会将整行的数据迁移到一个新的数据块上，而将该行原先的空间只放一个指针，指向该行的新的位置，并且该行原先空间的剩余空间不再被数据库使用，这些剩余的空间我们将其称之为空洞，这就是产生表碎片的主要原因，表碎片基本上也是不可避免的，但是我们可以将其降到一个我们可以接受的程度。**注意，即使发生了行迁移，发生了行迁移的行的rowid 还是不会变化，这也是行迁移会引起数据库I/O性能降低的原因。**其实行迁移是行链接的一种特殊形式，但是它的起因与行为跟行链接有很大不同，所以一般把它从行链接中独立出来，单独进行处理。

行链接和行迁移引起数据库性能下降的原因：

引起性能下降的原因主要是由于引起多余的I/O造成的。当通过索引访问已有行迁移现象的行时，数据库必须扫描一个以上的数据块才能检索到改行的数据。这主要有一下两种表现形式：

1)  导致row migration 或row chaining INSERT 或 UPDATE语句的性能比较差，因为它们需要执行额外的处理。

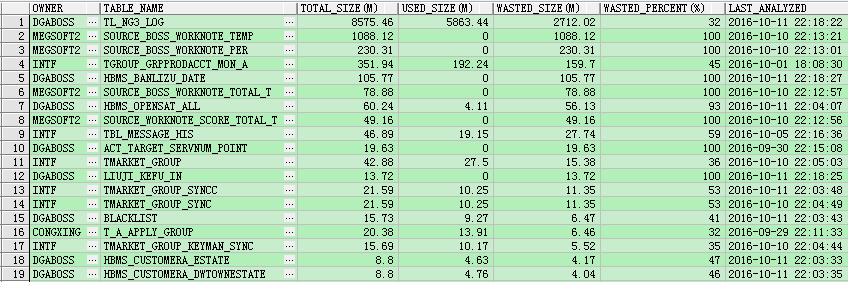
2)  利用索引查询已经链接或迁移的行的select语句性能比较差，因为它们要执行额外的I/O。

### 表的高水位回收

#### 寻找需要高水位回收的表

只有当对表进行统计信息收集之后，再进行统计，才是准确的。

|  |
| --- |
| SELECT OWNER,  TABLE\_NAME,  ROUND(BLOCKS \* 8192 / 1024 / 1024, 2) "TOTAL\_SIZE(M)",  ROUND(NUM\_ROWS \* AVG\_ROW\_LEN / 1024 / 1024, 2) "USED\_SIZE(M)",  ROUND(((BLOCKS \* 8192 / 1024 / 1024) -  (NUM\_ROWS \* AVG\_ROW\_LEN / 1024 / 1024)),2) "WASTED\_SIZE(M)",  ROUND(ROUND(((BLOCKS \* 8192 / 1024 / 1024) -  (NUM\_ROWS \* AVG\_ROW\_LEN / 1024 / 1024)),2) / ROUND(BLOCKS \* 8192 / 1024 / 1024, 2),2) \* 100 "WASTED\_PERCENT(%)",  LAST\_ANALYZED  FROM DBA\_TABLES  WHERE ROUND(BLOCKS \* 8192 / 1024 / 1024, 2) <> 0  AND ROUND(ROUND(((BLOCKS \* 8192 / 1024 / 1024) -  (NUM\_ROWS \* AVG\_ROW\_LEN / 1024 / 1024)),2) / ROUND(BLOCKS \* 8192 / 1024 / 1024, 2),2) \* 100 > 30  AND OWNER NOT LIKE '%SYS%'  AND SYSDATE-LAST\_ANALYZED<20  ORDER BY  5 DESC,  6 DESC; |



#### 执行高水位回收

有两种高水位回收的方式：一种是MOVE，一种是SHRINK。其中，MOVE需要两倍的表空间，在MOVE过程DML操作会挂起，而且索引会失效掉。而SHRINK不需要额外的空间，在SHRINK过程DML操作也不会挂起，索引也不会失效。但是SHRINK过程会产生大量的UNDO和REDO信息。

|  |
| --- |
| ALTER TABLE DGABOSS.TL\_NG3\_LOG ENABLE ROW MOVEMENT;  ALTER TABLE DGABOSS.TL\_NG3\_LOG SHRINK SPACE CASCADE; |

## 表压缩

### 压缩表

|  |
| --- |
| ALTER TABLE XXX MOVE COMPRESS FOR OLTP; |

### 取消表压缩

|  |
| --- |
| ALTER TABLE XXX MOVE NOCOMPRESS; |

### 预估表压缩的效果

|  |
| --- |
| DECLARE  BLKCNT\_COMP PLS\_INTEGER;  BLKCNT\_UNCM PLS\_INTEGER;  ROW\_COMP PLS\_INTEGER;  ROW\_UNCM PLS\_INTEGER;  COMP\_RATIO NUMBER;  COMP\_TYPE VARCHAR2(30);  OWNER VARCHAR2(30) := '&OWNER';  TABLENAME VARCHAR2(30) := '&TABLENAME' ;  BEGIN  DBMS\_COMPRESSION.GET\_COMPRESSION\_RATIO(  '&TALBESPACE',  OWNER,  TABLENAME,  NULL,  DBMS\_COMPRESSION.COMP\_FOR\_OLTP,  BLKCNT\_COMP,  BLKCNT\_UNCM,  ROW\_COMP,  ROW\_UNCM,  COMP\_RATIO,  COMP\_TYPE);  DBMS\_OUTPUT.PUT\_LINE('SAMPLING TABLE: '||OWNER||'.'||TABLENAME);  DBMS\_OUTPUT.PUT\_LINE('ESTIMATED COMPRESSION RATIO: ' ||TO\_CHAR(ROUND(COMP\_RATIO, 2)));  DBMS\_OUTPUT.PUT\_LINE('COMPRESSION TYPE: ' || COMP\_TYPE);  END;  / |

PF(E~SNF30}T@N76WDM)SIU

## 索引

### 查看索引

|  |
| --- |
| SELECT INDEX\_NAME, STATUS ,INDEX\_TYPE FROM DBA\_INDEXES WHERE TABLE\_NAME = 'TL\_NG3\_LOG' AND OWNER = 'DGABOSS';  SELECT INDEX\_NAME, COLUMN\_NAME FROM DBA\_IND\_COLUMNS WHERE TABLE\_NAME = 'TL\_NG3\_LOG' AND TABLE\_OWNER = 'DGABOSS'; |

### 索引重建

如何寻找需要重建的索引？

1. 做过高水位线回收的表索引需重建
2. 索引失效的索引：

SELECT OWNER,INDEX\_NAME,INDEX\_TYPE,TABLE\_NAME,STATUS FROM DBA\_INDEXES

WHERE STATUS = 'UNUSABLE';

说明：索引失效通常都是表做了变更，可先确认表是否被删除了，还是做了变更，若表已被删除，则删除索引，无需重建。

3) 当索引占用的空间超过了一个数据块且符合下面任何一个条件的时候：

A B－TREE的高度>3

B 数据删除率大于20％

C 索引的使用百分比小于75％

step1：分析索引

analyze index index\_name validate structure;

step2:查询出需要重建的索引：

select btree\_space,height,pct\_used,(del\_lf\_rows/decode(lf\_rows,0,1,lf\_rows))\*100 as delete\_pct from index\_stats;

4）察看 dba\_indexes 中的 blevel 。这列是说明索引从根块到叶快的级别，或是深度。如果级别大于等于4。则需要重建:

select index\_name,blevel,i.owner from dba\_indexes i where blevel>=4

重建索引：

1. 重建索引的几个方法

A:删除索引，然后再重新创建索引：

删除索引：drop index index\_name;

创建索引：create index index\_name on table\_name (col\_name);

说明：此方式耗时间，无法在24\*7环境中实现，不建议使用。

A:使用rebuild重建

alter index index\_name rebuild;

说明：会阻塞对基表的DML操作，但不会影响rebuild期间查询对原有索引的使用。

B:使用rebulid online 重建

alter index index\_name rebuild online;

说明：运行用户在索引重建期间执行DML操作。数据源是基表。

C：rebuild重建指定表空间

alter index index\_name rebuild tablespace tablespace\_name;

说明：加入表空间名，会将指定的索引移动到指定的表空间当中。

D：使用coalesce重建

alter index index\_name coalesce;

说明：coalesce 使用带有coalesce参数时重建期间不需要额外空间，它只是在重建索引时将处于同一个索引分支内的叶块拼合起来，这最大限度的减少了与查询过程中相关的潜在的加锁问题，但是，coalesce选项不能用来将一个索引转移到其他表空间。

### 索引的碎片整理

#### 寻找需要碎片整理的索引

*-- 分析大表*

DROP TABLE DGLJW\_BIGTABLE;

CREATE TABLE DGLJW\_BIGTABLE AS

(

SELECT

TABLESPACE\_NAME,

OWNER,

SEGMENT\_NAME AS TABLE\_NAME,

SEGMENT\_TYPE,

BYTES/1024/1024 AS MBs

FROM

DBA\_SEGMENTS

WHERE

TABLESPACE\_NAME NOT IN ('SYSTEM', 'SYSAUX', 'UNDOTBS1')

AND SEGMENT\_TYPE = 'TABLE'

AND BYTES/1024/1024 > 200

);

*-- 分析大表有哪些索引*

DROP TABLE DGLJW\_BIGINDEX;

CREATE TABLE DGLJW\_BIGINDEX AS

(

SELECT

　　B.OWNER,

B.TABLE\_NAME,

T.INDEX\_NAME,

T.DISTINCT\_KEYS,

T.NUM\_ROWS,

T.SAMPLE\_SIZE,

T.LAST\_ANALYZED,

T.BLEVEL,

T.LEAF\_BLOCKS

FROM

DBA\_INDEXES T,

DGLJW\_BIGTABLE B

WHERE

T.OWNER = B.OWNER

AND T.TABLE\_NAME = B.TABLE\_NAME

AND T.OWNER NOT IN ('SYSMAN','OWBSYS','OWB$CLIENT','SCHEDULER\_ADMIN','SPATIAL\_CSW\_ADMIN\_USR','SPATIAL\_WFS\_ADMIN\_USR',',SYSMAN','SCHEDULER\_ADMIN','OLAP\_DBA','OLAP\_USER','OUTLN','RECOVERY\_CATALOG\_OWNER','RESOURCE','DBA', 'OLAPSYS','AQ\_ADMINISTRATOR\_ROLE','MDSYS','JAVADEBUGPRIV','EXFSYS','CTXSYS','SYS', 'SYSTEM', 'WMSYS','IWMSYS', 'IMP\_FULL\_DATABASE', 'DATAPUMP\_IMP\_FULL\_DATABASE','DBSNMP', 'EXP\_FULL\_DATABASE')

AND T.NUM\_ROWS > 0

AND T.NUM\_ROWS IS NOT NULL

);

SELECT \* FROM DGLJW\_BIGINDEX ORDER BY NUM\_ROWS DESC;

ANALYZE INDEX OWNER.TABLE\_NAME VALIDATE STRUCTURE;

validate structure有二种模式： online, offline， 一般来讲默认的方式是offline

当以offline的模式analyze索引时，会对table加一个表级共享锁，对目前table的一些实时DMl操作会产生一定的影响。而以online模式分析时候，则不会加任何lock，但在index\_stats中是看不到任何信息的。

*-- 查看分析结果，当DELETE\_RATIO大于15%时，可考虑重建索引*

SELECT

T.NAME,

T.BLOCKS,

T.LF\_ROWS,

T.DEL\_LF\_ROWS,

T.LF\_ROWS - T.DEL\_LF\_ROWS AS LF\_ROWS\_USED,

TO\_CHAR((T.DEL\_LF\_ROWS/T.LF\_ROWS)\*100, '999.999') AS DELETE\_RATIO,

FROM

INDEX\_STATS T;

#### 通过重建索引来解决碎片问题。

# 安全性

## 访问白名单

vi sqlnet.ora

#允许访问的IP

TCP.INVITED\_NODES=(允许访问的IP)

TCP.VALIDNODE\_CHECKING=yes

编辑完之后重启Oracle的监听器：lsnrctl reload

## Oracle密码策略

可以从DBA\_PROFILES选择PROFILE=’DEFAULT’，RESOURCE\_TYPE=’PASSWORD’查得到：

|  |  |
| --- | --- |
| 密码策略标识 | 描述 |
| FAILED\_LOGIN\_ATTEMPTS | 指定锁定用户的登录失败次数 |
| PASSWORD\_LIFE\_TIME | 密码可以被使用的天数，单位是天，默认值180天 |
| PASSWORD\_REUSE\_TIME | 密码可重用的间隔时间(结合PASSWORD\_REUSE\_MAX) |
| PASSWORD\_REUSE\_MAX | 密码的最大改变次数(结合PASSWORD\_REUSE\_TIME) |
| PASSWORD\_VERIFY\_FUNCTION | 密码校验函数 |
| PASSWORD\_LOCK\_TIME | 超过错误尝试次数后，用户被锁定的天数，默认1天 |
| PASSWORD\_GRACE\_TIME | 当密码过期之后还有多少天可以使用原密码 |

可以安装Oracle的默认密码策略：



可以创建不同的PROFILE，给不同的用户指定不同的PROFILE，这样管理会更加灵活。

## Oracle有多少种权限

**群集权限**

CREATE CLUSTER 在自己的方案中创建、更改和删除群集

CREATE ANY CLUSTER 在任何方案中创建群集

ALTER ANY CLUSTER 在任何方案中更改群集

DROP ANY CLUSTER 在任何方案中删除群集

**数据库权限**

ALTER DATABASE 运行ALTER DATABASE语句，更改数据库的配置

ALTER SYSTEM 运行ALTER SYSTEM语句，更改系统的初始化参数

AUDIT SYSTEM 运行AUDIT SYSTEM和NOAUDIT SYSTEM语句，审计SQL

AUDIT ANY 运行AUDIT和NOAUDIT语句，对任何方案的对象进行审计

**索引权限**

CREATE ANY INDEX 在任何方案中创建索引注意：没有CREATE INDEX权限， CREATE TABLE 权限包含了CREATE INDEX权限

ALTER ANY INDEX 在任何方案中更改索引

DROP ANY INDEX 任何方案中删除索引

**过程权限**

CREATE PROCEDURE 在自己的方案中创建、更改或删除过程、 函数和包

CREATE ANY PROCEDURE 在任何方案中创建过程、函数和包

ALTER ANY PROCEDURE 在任何方案中更改过程、函数和包

DROP ANY PROCEDURE 在任何方案中删除过程、函数或包

EXECUTE ANY PROCEDURE 在任何方案中执行或者引用过程

**概要文件权限**

CREATE PROFILE 创建概要文件

ALTER PROFILE 更改概要文件

DROP PROFILE 删除概要文件

**角色权限**

CREATE ROLE 创建角色

ALTER ANY ROLE 更改任何角色

DROP ANY ROLE 删除任何角色

GRANT ANY ROLE 向其他角色或用户授予任何角色

注意：没有对应的REVOKE ANY ROLE权限

**回退段权限**

CREATE ROLLBACK SEGMENT 创建回退段

注意：没有对撤销段的权限

ALTER ROLLBACK SEGMENT 更改回退段

DROP ROLLBACK SEGMENT 删除回退段

**序列权限**

CREATE SEQLENCE 在自己的方案中创建、更改、删除和选择序列

CREATE ANY SEQUENCE 在任何方案中创建序列

ALTER ANY SEQUENCE 在任何方案中更改序列

DROP ANY SEQUENCE 在任何方案中删除序列

SELECT ANY SEQUENCE 在任何方案中从任何序列中进行选择

**会话权限**

CREATE SESSION 创建会话，登录进入（连接到）数据库

ALTER SESSION 运行ALTER SESSION语句，更改会话的属性

ALTER RESOURCE COST 更改概要文件中的计算资源消耗的方式

RESTRICTED SESSION 在数据库处于受限会话模式下连接到数据

**同义词权限**

CREATE SYNONYM 在自己的方案中创建、删除同义词

CREATE ANY SYNONYM 在任何方案中创建专用同义词

CREATE PUBLIC SYNONYM 创建公共同义词

DROP ANY SYNONYM 在任何方案中删除同义词

DROP PUBLIC SYNONYM 删除公共同义词

**表权限**

CREATE TABLE 在自己的方案中创建、更改和删除表

CREATE ANY TABLE 在任何方案中创建表

ALTER ANY-TABLE 在任何方案中更改表

DROP ANY TABLE 在任何方案中删除表

COMMENT ANY TABLE 在任何方案中为任何表、视图或者列添加注释

SELECT ANY TABLE 在任何方案中选择任何表中的记录

INSERT ANY TABLE 在任何方案中向任何表插入新记录

UPDATE ANY TABLE 在任何方案中更改任何表中的记录

DELETE ANY TABLE 在任何方案中删除任何表中的记录

LOCK ANY TABLE 在任何方案中锁定任何表

FLASHBACK ANY TABLE 允许使用AS OF子句对任何方案中的表、视图执行一个SQL语句的闪回查询

**表空间权限**

CREATE TABLESPACE 创建表空间

ALTER TABLESPACE 更改表空间

DROP TABLESPACE 删除表空间，包括表、索引和表空间的群集

MANAGE TABLESPACE 管理表空间，使表空间处于ONLINE（联机） 、OFFLINE（脱机）、BEGIN BACKUP（开始备份） 、END BACKUP（结束备份）状态

UNLIMITED TABLESPACE 不受配额限制地使用表空间

注意：只能将UNLIMITED TABLESPACE授予账户而不能授予角色

用户权限

CREATE USER 创建用户

ALTER USER 更改用户

BECOME USER 当执行完全装入时，成为另一个用户

DROP USER 删除用户

**视图权限**

CREATE VIEW 在自己的方案中创建、更改和删除视图

CREATE ANY VIEW 在任何方案中创建视图

DROP ANY VIEW 在任何方案中删除视图

COMMENT ANY TABLE 在任何方案中为任何表、视图或者列添加注释

FLASHBACK ANY TABLE 允许使用AS OF子句对任何方案中的表、视图执行一个SQL语句的闪回查询

**触发器权限**

CREATE TRIGGER 在自己的方案中创建、更改和删除触发器

CREATE ANY TRIGGER 在任何方案中创建触发器

ALTER ANY TRIGGER 在任何方案中更改触发器

DROP ANY TRIGGER 在任何方案中删除触发器

ADMINISTER DATABASE TRIGGER 允许创建ON DATABASE触发器。在能够创建ON DATABASE触发器之前，还必须先拥有CREATE TRIGGER或CREATE ANY TRIGGER权限

专用权限

**SYSOPER**

**（系统操作员权限）**

STARTUP

SHUTDOWN

ALTER DATABASE MOUNT/OPEN

ALTER DATABASE BACKUP CONTROLFILE

ALTER DATABASE BEGIN/END BACKUP

ALTER DATABASE ARCHIVELOG

RECOVER DATABASE

RESTRICTED SESSION

CREATE SPFILE/PFILE

**SYSDBA**

**（系统管理员权限）**

SYSOPER的所有权限，并带有WITH ADMIN

OPTION子句

CREATE DATABASE

RECOVER DATABASE UNTIL

**其他权限**

ANALYZE ANY 对任何方案中的任何表、群集或者索引执行ANALYZE语句

GRANT ANY OBJECT PRIVILEGE 授予任何方案上的任何对象上的对象权限

注意：没有对应的REVOKE ANY OBJECT PRIVILEGE

GRANT ANY PRIVILEGE 授予任何系统权限

注意：没有对应的REVOKE ANY PRIVILEGE

SELECT ANY DICTIONARY 允许从sys用户所拥有的数据字典表中进行选择

## 权限检查

### 基本的权限检查语句

*-- 现在系统定义了哪些角色*

SELECT \* FROM DBA\_ROLES;

*-- 所有的用户与角色或角色与角色之间的授予关系*

SELECT \* FROM DBA\_ROLE\_PRIVS ORDER BY GRANTEE;

*-- 授予所有用户或角色的系统权限*

SELECT \* FROM DBA\_SYS\_PRIVS ORDER BY GRANTEE;

*-- 授予所有用户或角色的对象权限*

SELECT \* FROM DBA\_TAB\_PRIVS;

### DBA权限检查

SELECT \* FROM DBA\_ROLE\_PRIVS R WHERE R.GRANTED\_ROLE='DBA';

**处理方法：如果发现有非法用户拥有了DBA角色，要及时处理。**

### 检查非法系统权限

SELECT

\*

FROM

DBA\_SYS\_PRIVS

WHERE

GRANTEE NOT IN ('SYSMAN','OWBSYS','OWB$CLIENT','SCHEDULER\_ADMIN','SPATIAL\_CSW\_ADMIN\_USR','SPATIAL\_WFS\_ADMIN\_USR',',SYSMAN','SCHEDULER\_ADMIN','OLAP\_DBA','OLAP\_USER','OUTLN','RECOVERY\_CATALOG\_OWNER','RESOURCE','DBA', 'OLAPSYS','AQ\_ADMINISTRATOR\_ROLE','MDSYS','JAVADEBUGPRIV','EXFSYS','CTXSYS','SYS', 'SYSTEM', 'WMSYS','IWMSYS', 'IMP\_FULL\_DATABASE', 'DATAPUMP\_IMP\_FULL\_DATABASE','DBSNMP', 'EXP\_FULL\_DATABASE')

AND (PRIVILEGE IN (SELECT DISTINCT PRIVILEGE FROM DBA\_SYS\_PRIVS WHERE PRIVILEGE LIKE '%ANY%')

OR PRIVILEGE IN ('CREATE JOB', 'CREATE TABLESPACE', 'CREATE PROCEDURE', 'CREATE ROLE', 'CREATE RULE', 'CREATE USER')

)

ORDER BY

GRANTEE;

**处理方法：如果发现有非法用户拥有敏感的系统权限，要及时处理。**

### 检查非法的对象权限

先定义敏感对象集合

CREATE TABLE DGLJW\_MINGAN

(

OWNER VARCHAR2(30),

TABLE\_NAME VARCHAR2(30)

);

SELECT

A.\*

FROM

DBA\_TAB\_PRIVS A,

DGLJW\_MINGAN B

WHERE

A.OWNER = B.OWNER

AND A.TABLE\_NAME = B.TABLE\_NAME;

**处理方法：如果发现有非法用户拥有对敏感表的权限，要及时处理。**

## 审计功能开启

SHOW PARAMETER AUDIT\_TRAIL;

NONE 禁用数据库审计

OS 启用数据库审计，并将数据库审计记录定向到操作系统审计记录

DB 启用数据库审计，并将数据库所有审计记录定向到数据库的SYS.AUD$表

DB,EXTENDED 启用数据库审计，并将数据库所有审计记录定向到数据库的SYS.AUD$表。另外，填充SYS.AUD$表的SQLBIND 列和SQLTEXT CLOB 列。

XML 启用数据库审计，并将所有记录写到XML格式的操作系统文件中。

XML,EXTENDED 启用数据库审计，输出审计记录的所有列，包括SQLTEXT和SQLBIND的值。

|  |
| --- |
| ALTER SYSTEM SET AUDIT\_TRAIL = DB,EXTENDED SCOPE = SPFILE; |

## 审计SYSDBA

|  |
| --- |
| ALTER SYSTEM SET AUDIT\_SYS\_OPERATIONS = TRUE SCOPE = SPFILE; |

## 当前的审计策略

|  |
| --- |
| *-- 所有默认审计项*  SELECT \* FROM ALL\_DEF\_AUDIT\_OPTS;  *--*  SELECT \* FROM DBA\_STMT\_AUDIT\_OPTS ORDER BY USER\_NAME, AUDIT\_OPTION;  SELECT \* FROM DBA\_PRIV\_AUDIT\_OPTS ORDER BY USER\_NAME, PRIVILEGE;  SELECT \* FROM DBA\_OBJ\_AUDIT\_OPTS; |

## 细粒度审计策略FGA

增加细粒度审计策略，可以做到某表某列上的很细微的条件

|  |
| --- |
| BEGIN  DBMS\_FGA.ADD\_POLICY  (  OBJECT\_SCHEMA => 'BANK',  OBJECT\_NAME => 'ACCOUNTS',  POLICY\_NAME => 'ACCOUNTS\_ACCESS',  AUDIT\_COLUMN => 'BALANCE',  AUDIT\_CONDITION => 'BALANCE >= 11000'  );  END;  / |

删除细粒度审计策略

|  |
| --- |
| BEGIN  DBMS\_FGA.DROP\_POLICY  (  OBJECT\_SCHEMA => 'BANK',  OBJECT\_NAME => 'ACCOUNTS',  POLICY\_NAME => 'ACCOUNTS\_ACCESS'  );  END;  /  这是批量删除细粒度审计  SELECT 'EXEC BEGIN DBMS\_FGA.DROP\_POLICY ( OBJECT\_SCHEMA =>' || '''' || A.object\_schema || ''''|| ', OBJECT\_NAME => ' || '''' || A.object\_name || ''''|| ', POLICY\_NAME =>' || ''''|| A.policy\_name || ''''|| '); END;' FROM DBA\_AUDIT\_POLICIES A; |

生效细粒度审计策略

|  |
| --- |
| BEGIN  DBMS\_FGA.ENABLE\_POLICY  (  OBJECT\_SCHEMA => 'BANK',  OBJECT\_NAME => 'ACCOUNTS',  POLICY\_NAME => 'ACCOUNTS\_ACCESS',  ENABLE => FALSE  );  END;  / |

查看当前配置的所有细粒度审计策略

|  |
| --- |
| SELECT \* FROM DBA\_AUDIT\_POLICIES; |

查看细粒度审计结果

|  |
| --- |
| SELECT \* FROM DBA\_FGA\_AUDIT\_TRAIL; *-- DBA\_FGA\_AUDIT\_TRAIL为基表 FGA\_LOG$的视图* |

## 对普通业务支撑帐号的敏感权限开启审计

|  |
| --- |
| SET LINESIZE 10000;  SELECT 'AUDIT ALTER DATABASE, ALTER SYSTEM, AUDIT SYSTEM, CREATE ANY PROCEDURE, ALTER ANY PROCEDURE, DROP ANY PROCEDURE, CREATE ANY TABLE, DROP ANY TABLE, UPDATE ANY TABLE, INSERT ANY TABLE, DELETE ANY TABLE, LOCK ANY TABLE, CREATE TABLESPACE, ALTER TABLESPACE, DROP TABLESPACE, MANAGE TABLESPACE, UNLIMITED TABLESPACE, CREATE ANY VIEW, DROP ANY VIEW, CREATE ANY TRIGGER, ALTER ANY TRIGGER, DROP ANY TRIGGER, GRANT ANY OBJECT PRIVILEGE, GRANT ANY PRIVILEGE BY '|| USERNAME ||' BY ACCESS;' FROM DBA\_USERS WHERE USERNAME IN ('ZABBIX'); |

创建出语句后执行就行，建议在对帐号进行操作审计的时候，不要考虑程序帐号，不然太多了。

## 对程序帐号的非增删改查表权限开启审计

|  |
| --- |
| SET LINESIZE 10000;  SELECT 'AUDIT ALTER DATABASE, ALTER SYSTEM, AUDIT SYSTEM, CREATE ANY PROCEDURE, ALTER ANY PROCEDURE, DROP ANY PROCEDURE, CREATE ANY TABLE, DROP ANY TABLE, LOCK ANY TABLE, CREATE TABLESPACE, ALTER TABLESPACE, DROP TABLESPACE, MANAGE TABLESPACE, UNLIMITED TABLESPACE, CREATE ANY VIEW, DROP ANY VIEW, CREATE ANY TRIGGER, ALTER ANY TRIGGER, DROP ANY TRIGGER, GRANT ANY OBJECT PRIVILEGE, GRANT ANY PRIVILEGE BY '|| USERNAME ||' BY ACCESS;' FROM DBA\_USERS WHERE USERNAME IN ('ZABBIX'); |

## 创建登录记录的表及触发器

|  |
| --- |
| CREATE TABLE SYS.DGLJW\_LOGON\_LOG TABLESPACE USERS AS  (  SELECT  SYSDATE LOGON\_TIME,  SYS\_CONTEXT('USERENV','SESSION\_USER') USERNAME,  SYS\_CONTEXT('USERENV','HOST') HOST,  SYS\_CONTEXT('USERENV','OS\_USER') OS\_USER,  SYS\_CONTEXT('USERENV','IP\_ADDRESS') IP\_ADDRESS,  SYS\_CONTEXT('USERENV','MODULE') MODULE  FROM  DUAL  );  GRANT SELECT ON SYS.DGLJW\_LOGON\_LOG TO DG\_LIHONGTING; |

|  |
| --- |
| CREATE OR REPLACE TRIGGER SYS.TRIGGER\_DGLJW\_LOGON\_LOG AFTER LOGON ON DATABASE  BEGIN  IF SYS\_CONTEXT('USERENV', 'MODULE') NOT IN ('JDBC Thin Client', 'plsqldev.exe') AND SYS\_CONTEXT('USERENV', 'SESSION\_USER') NOT IN ('WMSYS', 'DBSNMP', 'SYSMAN', 'MDSYS', 'OLAPSYS', 'OWBSYS', 'OWBSYS', 'OUTLN', 'ORDSYS') THEN  INSERT INTO DGLJW\_LOGON\_LOG  (  LOGON\_TIME,  USERNAME,  HOST,  OS\_USER,  IP\_ADDRESS,  MODULE  )  VALUES  (  SYSDATE,  SYS\_CONTEXT('USERENV', 'SESSION\_USER'),  SYS\_CONTEXT('USERENV', 'HOST'),  SYS\_CONTEXT('USERENV', 'OS\_USER'),  SYS\_CONTEXT('USERENV', 'IP\_ADDRESS'),  SYS\_CONTEXT('USERENV', 'MODULE')  );  END IF;  END;  / |

|  |
| --- |
| ALTER TRIGGER SYS.TRIGGER\_DGLJW\_LOGON\_LOG ENABLE;  ALTER TRIGGER SYS.TRIGGER\_DGLJW\_LOGON\_LOG DISABLE; |

## 创建触发器实现IP与MODULE级别限制登陆

|  |
| --- |
| CREATE OR REPLACE TRIGGER TRIGGER\_DGLJW\_LOGON\_IP\_CONTROL AFTER LOGON ON DATABASE  DECLARE  IP STRING(30);  USER STRING(30);  MODULE STRING(30);  BEGIN  SELECT SYS\_CONTEXT('USERENV','SESSION\_USER') INTO USER FROM DUAL;  SELECT SYS\_CONTEXT('USERENV','IP\_ADDRESS') INTO IP FROM DUAL;  SELECT UPPER(SUBSTR(SYS\_CONTEXT('USERENV','MODULE'),1,7)) INTO MODULE FROM DUAL;  IF USER = 'HTGX\_RENYINGDA' THEN  IF MODULE NOT IN ('JDBC Thin Client') THEN  RAISE\_APPLICATION\_ERROR(-20001,'USER '|| USER || ' IS NOT ALLOWED TO CONNECT FROM '|| IP);  END IF;  END IF;  END;  /  ALTER TRIGGER SYS.TRIGGER\_DGLJW\_LOGON\_IP\_CONTROL ENABLE; |

## 登录帐号审计

### 从审计表里查看留下审计记录的用户

|  |
| --- |
| SELECT  DISTINCT A.OWNER  FROM  DBA\_AUDIT\_OBJECT A |

### 查看每个用户默认的表空间是否为SYSTEM或者SYSAUX

|  |
| --- |
| SELECT USERNAME,DEFAULT\_TABLESPACE FROM DBA\_USERS; |

### 查看每个用户的有审计记录的数目

|  |
| --- |
| SELECT */\*+PARALLEL(A,4)\*/*  A.TERMINAL,  A.USERNAME,  TRUNC(A.TIMESTAMP, 'DD'),  COUNT(\*) AS OPER\_COUNT  FROM  DBA\_AUDIT\_OBJECT A  WHERE  A.USERNAME LIKE '%HT%'  GROUP BY  A.TERMINAL,  A.USERNAME,  TRUNC(A.TIMESTAMP, 'DD')  ORDER BY  A.USERNAME,  TRUNC(A.TIMESTAMP, 'DD'),  A.TERMINAL; |

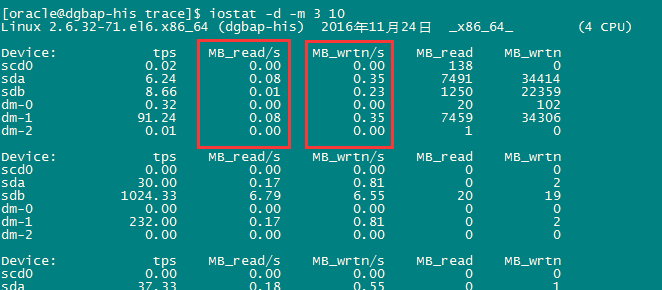


# 操作系统性能

## IO分析

|  |
| --- |
| $> iostat –d –m 3 10 |

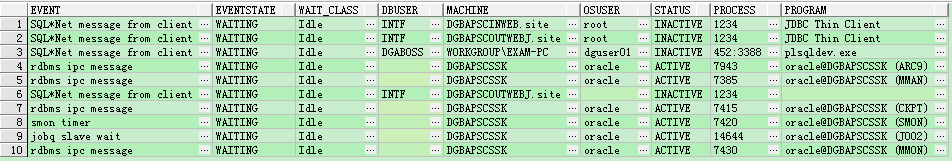
-d打印统计信息，-m表示以MB统计，3表示每3秒打印一次，10表示共打印10次。



# 数据库性能

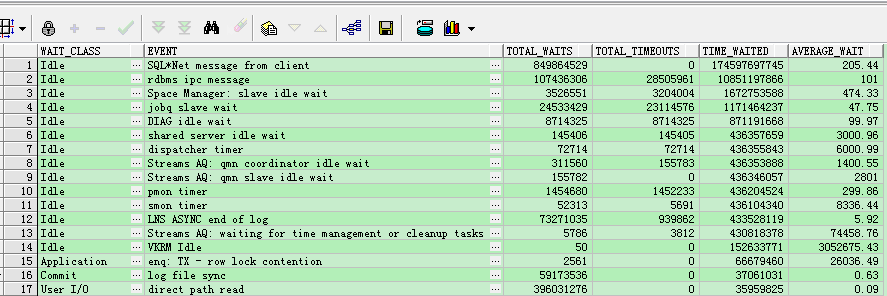
## 当前的SESSION的等待对象分析

|  |
| --- |
| SELECT DISTINCT  B.EVENT,  B.STATE AS EVENTSTATE,  B.WAIT\_CLASS,  A.USERNAME AS DBUSER,  A.MACHINE,  A.OSUSER,  A.STATUS,  A.PROCESS,  A.PROGRAM  FROM  V$SESSION\_WAIT B LEFT JOIN V$SESSION A ON B.SID = A.SID; |



## 等待事件的统计

|  |
| --- |
| SELECT  A.WAIT\_CLASS,  A.EVENT,  A.TOTAL\_WAITS,  A.TOTAL\_TIMEOUTS,  A.TIME\_WAITED,  A.AVERAGE\_WAIT  FROM  V$SYSTEM\_EVENT A  ORDER BY  A.TIME\_WAITED DESC; |



## 查看某条SQL的执行SESSION，是否有死锁

|  |
| --- |
| SELECT DISTINCT  A.USERNAME AS DBUSER,  A.MACHINE,  A.OSUSER,  A.STATUS,  A.PROCESS,  A.PROGRAM,  B.LAST\_ACTIVE\_TIME,  A.BLOCKING\_SESSION,  A.BLOCKING\_SESSION\_STATUS  FROM  V$SESSION A,  V$SQL B  WHERE  B.HASH\_VALUE = A.SQL\_HASH\_VALUE  AND B.SQL\_ID = 'fkfxyd7q4h9wf'; |

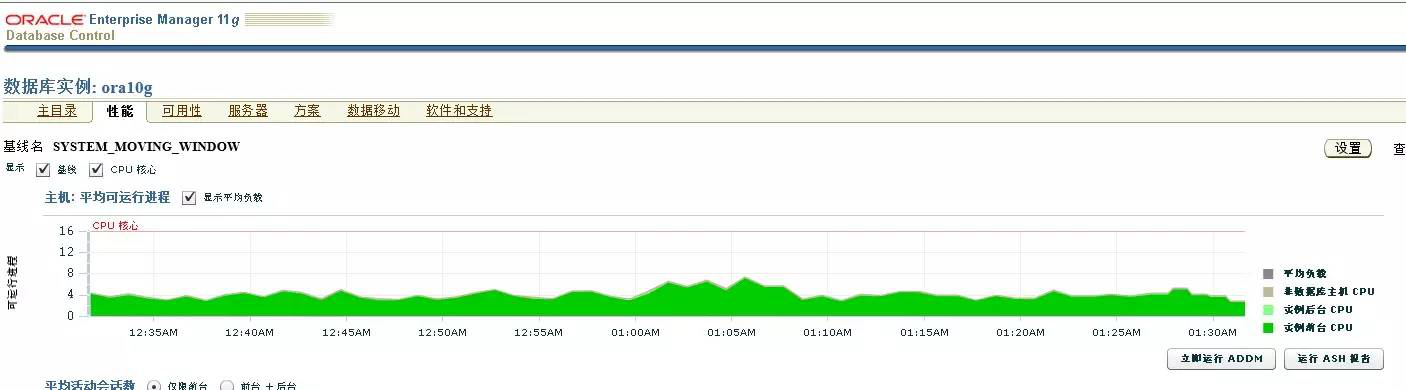
X51KJC6LV8XG_EQ%[}ZVKEL

## 命中率的统计

|  |  |  |
| --- | --- | --- |
| **命中率** | **说明** | **相关查询SQL** |
| buffer命中率 | 缓冲区命中率获得的数据与所有访问数据之间的比例,正常指标90%~100%。但在数据库繁忙运行期间也可能低于90%。 buffer命中率受[Oracle](http://www.linuxidc.com/topicnews.aspx?tid=12) sga中data block buffers参数设置影响，也可以通过调整buffer池的使用方法来提高buffer命中率。 | SELECT  ROUND((1-(PHYSICAL.VALUE-DIRECT.VALUE-LOBS.VALUE)/LOGICAL.VALUE)\*100,2) "缓冲区命中率"  FROM  V$SYSSTAT PHYSICAL,  V$SYSSTAT DIRECT,  V$SYSSTAT LOBS,  V$SYSSTAT LOGICAL  WHERE  PHYSICAL.NAME = 'physical reads'  AND DIRECT.NAME = 'physical reads direct'  AND LOBS.NAME = 'physical reads direct (lob)'  AND LOGICAL.NAME = 'session logical reads'; |
| library命中率 | 库缓冲区命中率指oracle在执行sql语句过程中，通过内存之间得到对象命名空间的比率,正常指标95%~100% --受oracle sga中的共享池参数设置影响 | select round(sum(pins-reloads)/sum(pins)\*100,2)"库缓冲区命中率" from v$librarycache; |
| latch命中率 | 内部结构维护锁命中率,正常指标99%~100%。如命中率较低，则通常是因为shared\_pool\_size参数值过大，或者没有使用绑定变量导致硬解析过多 | select round((1-sum(misses+immediate\_misses)/sum(gets+immediate\_gets))\*100,2)"latch命中率" from v$latch; |
| in-memory sort命中率 | 指内存中完成排序的数量和在硬盘上进行排序的数量比,正常指标99%~100%。受sort\_area\_size或pga\_aggregate\_target参数影响 | select round((1-disk.value/(disk.value+memory.value))\*100,2)"in-memory sort命中率" from v$sysstat disk,v$sysstat memory where        disk.NAME='sorts (disk)' and memory.NAME='sorts (memory)'; |
| buffer nowait | 指在缓冲区获取数据未等等待的比率,正常指标99%~100%。受db\_block\_buffers或db\_cache\_szie等参数影响 | select round((1-busy.value/tol.value)\*100,2)"buffer busy nowait ratio" from (select sum(count) value from v$waitstat where class in ('data block','segment header','undo header','undo block'))busy, (select value from v$sysstat where name='session logical reads')tol; |
| redo nowait | 指在重做缓冲区中获取数据未等待的比率,正常指标99%~100%。受log\_buffer\_size等参数影响，比较低 --原因可能是归档数度太慢，联机日志文件太小或联机日志文件存放在比较慢的存储设备上 | select round((1-waits.value/redos.value)\*100,2)"redo nowait" from v$sysstat waits,v$sysstat redos where  waits.NAME='redo log space requests'  and redos.NAME='redo entries'; |
| execute to parse | 指sql语句执行和分析的比率,正常指标接近100%。受share\_pool\_size等参数影响 | select round((1-hard.value/total.value)\*100,2)"execute to parse" from v$sysstat hard,v$sysstat total where hard.NAME='parse count (hard)' and total.NAME='parse count (total)'; |
| parse cpu to parse elapsed | 指数据库用于分析cpu的时间和分析完成的比率,,值比较低说明分析过程中cpu在等待其它资源 | select round((1-cpu.value/total.value)\*100,2)"parse cpu to parse elapsed" from v$sysstat cpu,v$sysstat total where cpu.NAME='parse time cpu' and total.NAME='parse time elapsed'; |
| non-parse cpu | 指查询实际运行时间占sql语句运行加解析时间之和的比例。如果该值太低则表示解析sql语句消耗的时间过长。 | select round((1-parse.value/total.value)\*100,2)"non-parse cpu" from v$sysstat parse,v$sysstat total where parse.NAME='parse time cpu' and total.NAME='CPU used by this session'; |
|  | 回滚段竞争情况,回滚段竞争情况比率越小越好，该值受回滚大小影响 | select name,waits,gets,round(waits/gets\*100,2)"回滚段竞争情况"  from v$rollstat a,v$rollname b where a.USN=b.usn; |
|  | 表空间的I/O比例,比例越小越好该值受db\_block\_size参数值的影响，并且和数据文件在磁盘的分布有关 | select df.tablespace\_name,sum(f.PHYRDS),sum(f.PHYBLKRD),sum(f.PHYRDS),sum(f.PHYBLKWRT) from v$filestat f,dba\_data\_files df where f.FILE#=df.file\_id  group by df.tablespace\_name order by df.tablespace\_name; |
|  | 数据文件的I/O比例,比例越小越好该值受db\_block\_size参数值的影响，并且和数据文件在磁盘的分布有关 | select df.name,sum(f.PHYRDS),sum(f.PHYBLKRD),sum(f.PHYRDS),sum(f.PHYBLKWRT) from v$filestat f,v$datafile df where f.FILE#=df.file#  group by df.name order by df.name; |
|  | 重做日志缓冲区命中率,越大越好正常指标应大于90%。该值log\_buffer\_size参数值的影响。 | select name,gets,misses,immediate\_gets,immediate\_misses, 100-round(decode(gets,0,0,misses/gets\*100),2)ratio1, 100-round(decode(immediate\_gets+immediate\_misses,0,0,immediate\_gets/(immediate\_gets+immediate\_misses)\*100),2)ratio2 from v$latch where name in('redo allocation','redo copy'); |
|  | 碎片程度,越大越好正常指标应大于30%。该值db\_block\_size和segment\_size参数值的影响 | select tablespace\_name,sqrt(max(blocks)/sum(blocks))\*(100/sqrt(count(blocks)))fsfl  from dba\_free\_space group by tablespace\_name order by tablespace\_name; |

## SQL Monitor

### 如何在EM中使用SQL Monitor





### 被SQL Monitor捕获的条件

当SQL满足以下条件之一就会被sql monitor捕获到，监控数据被记录在v$sql\_monitor视图中。

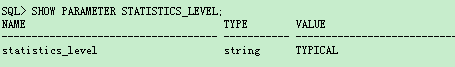
当SQL并行执行时，会立即被实时监控到。

当SQL单进程运行时，如果消耗超过5秒的CPU或I/O时间，它也会被监控到。

使用/\*+ monitor \*/提示的sql语句。

### 在sqlplus中查看SQL Monitor捕捉到的真正的执行计划

|  |
| --- |
| SHOW PARAMETER STATISTICS\_LEVEL; |



|  |
| --- |
| SELECT  STATISTICS\_NAME,  SESSION\_STATUS,  SYSTEM\_STATUS,  ACTIVATION\_LEVEL,  SESSION\_SETTABLE  FROM  V$STATISTICS\_LEVEL  WHERE  STATISTICS\_NAME = 'SQL Monitoring'; |

H5Q{D4NR9F16V])GEUTQ7)P

|  |
| --- |
| SHOW PARAMETER CONTROL\_MANAGE; |

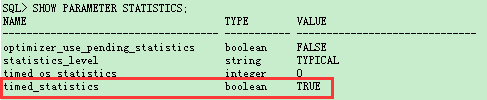
1J7}SY6~VC6L4I0_NR}CA$O

|  |
| --- |
| SET TRIMSPOOL ON;  SET ARRAYSIZE 512;  SET TRIM ON;  SET PAGESIZE 0;  SET LINESIZE 1000;  SET LONG 1000000;  SET LONGCHUNKSIZE 1000000;  SPOOL sqlmon.html  SELECT */\*+ NOPARALLEL \*/* DBMS\_SQLTUNE.REPORT\_SQL\_MONITOR (SQL\_ID=>'INPUT YOUR SQL ID', REPORT\_LEVEL=>'ALL', TYPE=>'ACTIVE') FROM DUAL;  SPOOL OFF  cat sqlmon.html |



## 追踪一条SQL语句

### 开启timed\_statistics参数为TRUE



### 开启SQL\_TRACE

|  |
| --- |
| ALTER SESSION SET SQL\_TRACE = TRUE; |

### 为本次Trace增加特殊的标识

|  |
| --- |
| ALTER SESSION SET TRACEFILE\_IDENTIFIER = 'dgljw'; |

### 查看Trace文件的路径

|  |
| --- |
| SELECT NAME ,VALUE FROM V$DIAG\_INFO WHERE NAME = 'Default Trace File'; |

FJXT@Z0RD~H$68LP2LBOHPO

### 为特定的SQL开启追踪

|  |
| --- |
| ALTER SESSION SET EVENTS 'sql\_trace [sql:fkfxyd7q4h9wf] level 12'; |

### 为特定的SQL关闭追踪

|  |
| --- |
| ALTER SESSION SET EVENTS 'sql\_trace [sql:fkfxyd7q4h9wf] off'; |

### 关闭SQL\_TRACE

|  |
| --- |
| ALTER SESSION SET SQL\_TRACE = FALSE; |

## TopSQL分析

### 查找执行次数最高的SQL

|  |
| --- |
| SELECT \* FROM (SELECT SQL\_ID,SQL\_TEXT,PARSING\_SCHEMA\_NAME AS USERNAME,EXECUTIONS FROM V$SQLAREA WHERE PARSING\_SCHEMA\_NAME NOT IN ('SYS', 'SYSMAN') ORDER BY EXECUTIONS DESC) WHERE ROWNUM < 60 |

### 寻找高消耗SQL

1) 切换到oracle用户，进入sqlplus，使用awrrpt.sql选择要查看的时间区间并生成AWR报告。

2) 在AWR报告中，分别查看最消耗CPU、读次数最多、执行次数最多等的SQL，并得到相应的SQLID。

如133库的：

353rmxakq589n select \* from CHECK\_MOBILENO where rownum < 2 and DEAL\_STS2 is null and oper\_time > to\_date('2015-04-09 00:00:00', 'yyyy-mm-dd hh24:mi:ss')

(3) 查找语句中的表名所属的数据库用户：

SELECT OWNER,TABLE\_NAME FROM DBA\_TABLES WHERE TABLE\_NAME = ‘’;

### SQL Turning Advisor分析TopSQL

#### 1) 生成SQL Turning分析任务

|  |
| --- |
| DECLARE  my\_task\_name VARCHAR2(30);  my\_sqltext CLOB;  BEGIN  my\_sqltext := 'select \* from DGABOSS.CHECK\_MOBILENO where rownum < 2 and DEAL\_STS2 is null and oper\_time > to\_date(''2015-04-09 00:00:00'', ''yyyy-mm-dd hh24:mi:ss'')';    my\_task\_name := DBMS\_SQLTUNE.CREATE\_TUNING\_TASK(  sql\_text => my\_sqltext,  user\_name => 'DGABOSS',  scope => 'COMPREHENSIVE',  time\_limit => 60,  task\_name => 'test\_sql\_tuning\_task1',  description => 'Task to tune a query');  DBMS\_SQLTUNE.EXECUTE\_TUNING\_TASK(task\_name => 'test\_sql\_tuning\_task1');  END;  / |

#### 2) 查看SQL Turning分析任务的状态

|  |
| --- |
| SELECT TASK\_NAME,ADVISOR\_NAME,STATUS FROM USER\_ADVISOR\_TASKS WHERE ADVISOR\_NAME = 'SQL Tuning Advisor'; |

#### 3) 查看SQL Turning分析建议

|  |
| --- |
| set long 999999  set LONGCHUNKSIZE 999999  set serveroutput on size 999999  set linesize 200  select dbms\_sqltune.report\_tuning\_task('test\_sql\_tuning\_task1') from dual; |

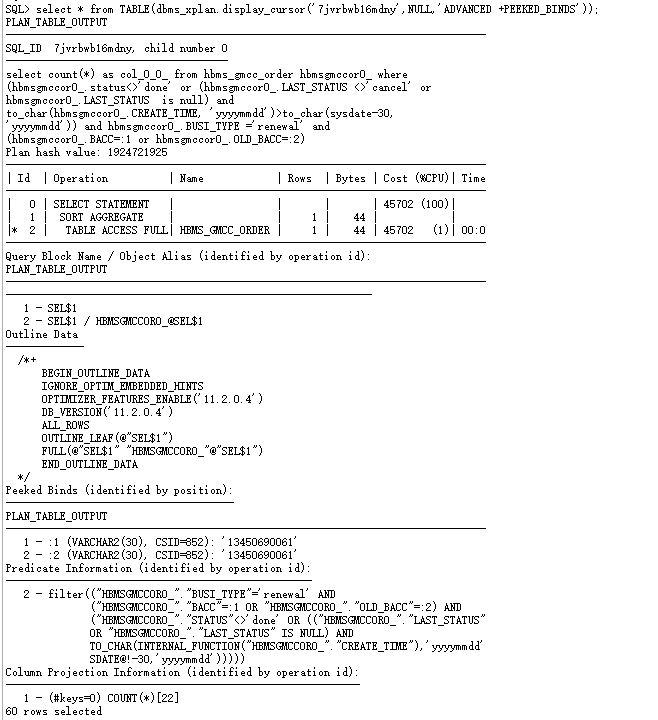
### SQL Access Advisor

### TopSQL执行计划查看

|  |  |
| --- | --- |
| Explain Plan For SQL | 不需要执行SQL语句，生成的计划未必是真实执行的计划，必须要有plan\_table |
| SQLPlus Autotrace | 除了set autotrace traceonly explain外均实际执行SQL，但仍未必是真实的执行计划，必须要有plan\_table |
| SQL TRACE | 需要启用10046或者SQL\_TRACE，一般用tkprof看得更清楚，当然10046里本身也有执行计划信息 |
| V$SQL和V$SQL\_PLAN | 可以查询得到多个子游标的计划信息，但看起来比较费劲 |
| EM | 但不是每个环境都能有EM |
| PL/SQL Devloper F5 | 未必真实 |

一般通用的方式

|  |
| --- |
| alter session set STATISTICS\_LEVEL = ALL;  select \* from TABLE(dbms\_xplan.display\_cursor('7jvrbwb16mdny',NULL,'ADVANCED +PEEKED\_BINDS')); |



### TopSQL分析示例

#### 1) 示例1

|  |  |
| --- | --- |
| **目标** | **select \* from CHECK\_MOBILENO where rownum < 2 and DEAL\_STS2 is null and oper\_time > to\_date('2015-04-09 00:00:00', 'yyyy-mm-dd hh24:mi:ss');** |
| 一天中的执行次数 | 333,919 |
| 每次的时间 | 0.16s |
| 查找对应的数据库用户 | SELECT OWNER,TABLE\_NAME FROM DBA\_TABLES WHERE TABLE\_NAME = 'CHECK\_MOBILENO';  OWNER TABLE\_NAME  ------------------------------ ------------------------------  DGABOSS CHECK\_MOBILENO |
| 收集统计信息 | EXEC DBMS\_STATS.GATHER\_TABLE\_STATS(OWNNAME=>'DGABOSS',TABNAME=>'CHECK\_MOBILENO', ESTIMATE\_PERCENT=>DBMS\_STATS.AUTO\_SAMPLE\_SIZE,METHOD\_OPT=>'for all columns size repeat',CASCADE=>TRUE,FORCE=>TRUE,DEGREE=>2); |
| 生成执行计划 | explain plan for select \* from DGABOSS.CHECK\_MOBILENO where rownum < 2 and DEAL\_STS2 is null and oper\_time > to\_date('2015-04-09 00:00:00', 'yyyy-mm-dd hh24:mi:ss'); |
| 查看执行计划 | PLAN\_TABLE\_OUTPUT  --------------------------------------------------------------------------------  Plan hash value: 39159674  --------------------------------------------------------------------------------  -----  | Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time  |  --------------------------------------------------------------------------------  -----  PLAN\_TABLE\_OUTPUT  --------------------------------------------------------------------------------  | 0 | SELECT STATEMENT | | 1 | 198 | 2 (0)| 00:00  :01 |  |\* 1 | COUNT STOPKEY | | | | |  |  |\* 2 | TABLE ACCESS FULL| CHECK\_MOBILENO | 1 | 198 | 2 (0)| 00:00  :01 |  --------------------------------------------------------------------------------  -----  PLAN\_TABLE\_OUTPUT  --------------------------------------------------------------------------------  Predicate Information (identified by operation id):  ---------------------------------------------------  1 - filter(ROWNUM<2)  2 - filter("DEAL\_STS2" IS NULL AND "OPER\_TIME">TO\_DATE(' 2015-04-09  00:00:00', 'syyyy-mm-dd hh24:mi:ss'))  16 rows selected. |
| 查看索引 | 从执行计划来看，访问走的是全表扫描，没有走索引。  SELECT \* from DBA\_INDEXES WHERE OWNER = 'DGABOSS' AND TABLE\_NAME = 'CHECK\_MOBILENO';  SELECT INDEX\_NAME, COLUMN\_NAME FROM DBA\_IND\_COLUMNS WHERE TABLE\_OWNER = 'DGABOSS' AND TABLE\_NAME = 'CHECK\_MOBILENO' ORDER BY INDEX\_NAME, COLUMN\_NAME ;  发现**DEAL\_STS2列建立了索引，但是oper\_time列没有建立索引。** |
| 使用SQL Turning Advisor分析 | 提示：There are no recommendations to improve the statement。但是为什么不走索引呢？ |
| 访问分析 | SELECT A.NUM\_ROWS FROM DBA\_TABLES A WHERE A.OWNER = 'DGABOSS' AND A.TABLE\_NAME = 'CHECK\_MOBILENO';  表共有534373行。  SELECT COUNT(\*) FROM dgaboss.CHECK\_MOBILENO where DEAL\_STS2 is null;  查询有318943行。  SELECT COUNT(\*) FROM dgaboss.CHECK\_MOBILENO where oper\_time > to\_date('2015-04-09 00:00:00', 'yyyy-mm-dd hh24:mi:ss');  查询有215430行。 |
| **结论** | 由于条件查询的行数都要接近总行数的一半，因此走索引还没有走全表扫描来得快。 |

## 表的统计信息收集

### 查找大表

|  |
| --- |
| SELECT  TABLESPACE\_NAME,  OWNER,  SEGMENT\_NAME,  SEGMENT\_TYPE,  BYTES/1024/1024 AS MBs  FROM  DBA\_SEGMENTS  WHERE  TABLESPACE\_NAME NOT IN ('SYSTEM', 'SYSAUX', 'UNDOTBS1')  AND SEGMENT\_TYPE = 'TABLE'  ORDER BY  MBs DESC,  TABLESPACE\_NAME,  OWNER; |

### 根据表名手工执行统计信息收集

|  |
| --- |
| EXEC ***DBMS\_STATS.GATHER\_TABLE\_STATS***(OWNNAME=>'DGABOSS',TABNAME=>'LX\_MF\_DETAIL',ESTIMATE\_PERCENT=>***DBMS\_STATS.AUTO\_SAMPLE\_SIZE***,METHOD\_OPT=>'for all columns size repeat',CASCADE=>TRUE,FORCE=>TRUE,DEGREE=>2); |

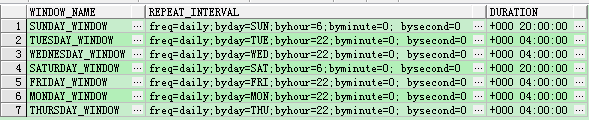
### 查看自动统计信息收集功能是否已经打开

|  |
| --- |
| SELECT CLIENT\_NAME, STATUS FROM DBA\_AUTOTASK\_CLIENT WHERE CLIENT\_NAME = 'auto optimizer stats collection'; |

6J827LG5D]E]F}(0Z49M`FQ

### 查看自动统计信息收集的维护窗口

|  |
| --- |
| SELECT DISTINCT  T1.WINDOW\_NAME,  T1.REPEAT\_INTERVAL,  T1.DURATION  FROM  DBA\_SCHEDULER\_WINDOWS T1,  DBA\_SCHEDULER\_WINGROUP\_MEMBERS T2  WHERE  T1.WINDOW\_NAME = T2.WINDOW\_NAME  AND T2.WINDOW\_GROUP\_NAME IN ('MAINTENANCE\_WINDOW\_GROUP','BSLN\_MAINTAIN\_STATS\_SCHED'); |



## 自动维护任务配置

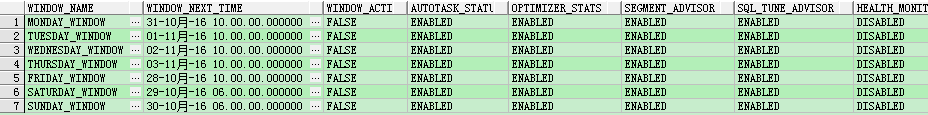
### 查看自动维护任务的开关

|  |
| --- |
| SELECT \* FROM DBA\_AUTOTASK\_CLIENT; |

{_$L4B~L03U56)0WE[14K`A

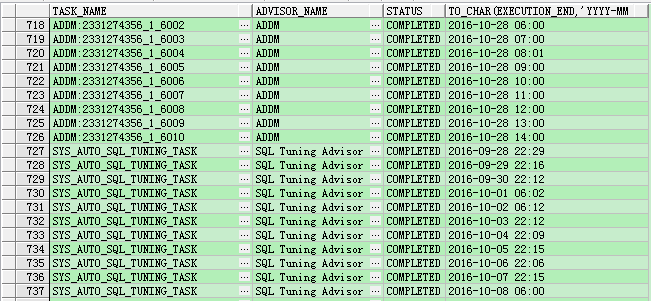
### 查看自动维护任务的窗口

|  |
| --- |
| SELECT \* FROM DBA\_AUTOTASK\_WINDOW\_CLIENTS; |



### 查看自动维护任务执行结果

|  |
| --- |
| SELECT  TASK\_NAME,  ADVISOR\_NAME,  STATUS,  TO\_CHAR(EXECUTION\_END, 'YYYY-MM-DD HH24:MI')  FROM  DBA\_ADVISOR\_EXECUTIONS  ORDER BY  ADVISOR\_NAME,  EXECUTION\_END; |

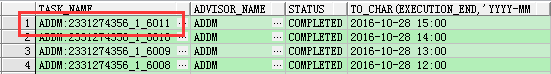


## ADDM报告查看

### 开启ADDM功能

### 查看系统在维护窗口已执行的ADDM分析

|  |
| --- |
| SELECT  TASK\_NAME,  ADVISOR\_NAME,  STATUS,  TO\_CHAR(EXECUTION\_END, 'YYYY-MM-DD HH24:MI')  FROM  DBA\_ADVISOR\_EXECUTIONS  WHERE  ADVISOR\_NAME = 'ADDM'  ORDER BY  EXECUTION\_END DESC; |

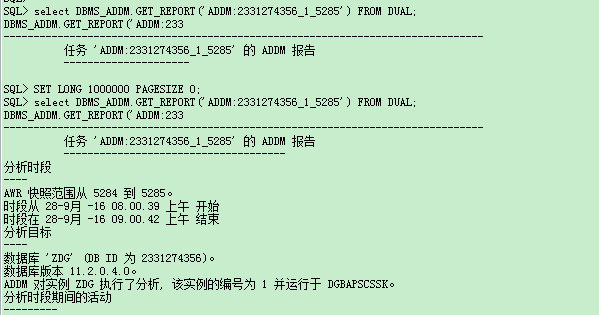


### 根据AWR快照手工生成ADDM报告

|  |
| --- |
| -- 查看AWR快照的分布  SELECT \* FROM DBA\_HIST\_SNAPSHOT ORDER BY BEGIN\_INTERVAL\_TIME;  VAR TASK\_NAME VARCHAR2(30);  EXEC DBMS\_ADDM.ANALYZE\_DB(:TASK\_NAME, 5973, 5997);  PRINT: TASK\_NAME; |

### 根据ADDM Task的名称查看ADDM报告

|  |
| --- |
| SQL > SET LONG 1000000 PAGESIZE 0;  SQL > SELECT DBMS\_ADDM.GET\_REPORT('ADDM:2331274356\_1\_6011') FROM DUAL; |



## AWR设置、报告生成和基线配置

### AWR报告生成

|  |
| --- |
| > sqlplus / as sysdba;  SQL> @$ORACLE\_HOME/rdbms/admin/awrrpt.sql |

### 查询和设置AWR报告的保留时间

|  |
| --- |
| SELECT DBMS\_STATS.GET\_STATS\_HISTORY\_RETENTION FROM DUAL; |

|  |
| --- |
| EXEC DBMS\_STATS.ALTER\_STATS\_HISTORY\_RETENTION(7); |

### 查询AWR现在有哪些快照

|  |
| --- |
| SELECT \* FROM DBA\_HIST\_SNAPSHOT ORDER BY SNAP\_ID; |

### 删除多余的AWR快照

|  |
| --- |
| BEGIN  DBMS\_WORKLOAD\_REPOSITORY.DROP\_SNAPSHOT\_RANGE  (  LOW\_SNAP\_ID => 8857,  HIGH\_SNAP\_ID => 9392,  DBID => 2331274356  );  END;  / |

### AWR设置

参数查看

|  |
| --- |
| SELECT \* FROM DBA\_HIST\_WR\_CONTROL; |

%W[_ALZ087FPLSYPQUUF8CR

设置快照收集的频率，以分钟为单位。可设置的值最小为10分钟，最大为1年。如果设置该参数值为0，就表示禁用AWR特性。

|  |
| --- |
| exec dbms\_workload\_repository.modify\_snapshot\_settings(INTERVAL =>30); |

设置快照保存的时间，单位是分钟。可设置的值最小为1天，最大为100年。设置该参数值为0的话，就表示永久保留收集的快照信息。每次保留3天。

|  |
| --- |
| exec dbms\_workload\_repository.modify\_snapshot\_settings(retention => 30 \* 24 \* 60); |

指定收集的比较占用资源的SQL数量，可设置的值最小为30，最大不超过100000000。

|  |
| --- |
| exec dbms\_workload\_repository.modify\_snapshot\_settings(TOPNSQL=>100); |

### AWR基线数据生成

|  |
| --- |
| BEGIN  DBMS\_WORKLOAD\_REPOSITORY.CREATE\_BASELINE(  START\_SNAP\_ID => 14660,  END\_SNAP\_ID => 14852,  BASELINE\_NAME => 'NORMAL\_DAYS');  END;  / |

## 关于表的读写性能和块分布的问题

以下是看一个表都有哪些区，每个区都有哪些块，哪些快分布在哪个数据文件中。

SELECT

DBA\_EXTENTS.TABLESPACE\_NAME,

DBA\_EXTENTS.EXTENT\_ID,

DBA\_EXTENTS.FILE\_ID,

DBA\_DATA\_FILES.FILE\_NAME,

DBA\_EXTENTS.BLOCK\_ID,

DBA\_EXTENTS.BLOCKS

FROM

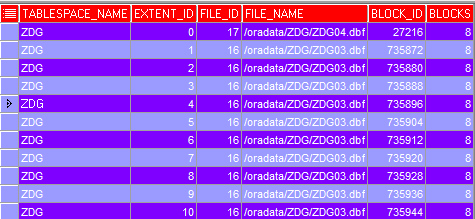
DBA\_EXTENTS LEFT JOIN DBA\_DATA\_FILES ON DBA\_EXTENTS.FILE\_ID = DBA\_DATA\_FILES.FILE\_ID

WHERE

DBA\_EXTENTS.OWNER = 'DGABOSS'

AND DBA\_EXTENTS.SEGMENT\_TYPE = 'TABLE'

AND DBA\_EXTENTS.SEGMENT\_NAME = 'HBMS\_GMCC\_ORDER';



统计一个表的数据块在每一个数据文件中分布了多少个：

SELECT

DBA\_DATA\_FILES.FILE\_NAME,

SUM(DBA\_EXTENTS.BLOCKS) AS BLOCKCOUNT

FROM

DBA\_EXTENTS LEFT JOIN DBA\_DATA\_FILES ON DBA\_EXTENTS.FILE\_ID = DBA\_DATA\_FILES.FILE\_ID

WHERE

DBA\_EXTENTS.OWNER = 'DGABOSS'

AND DBA\_EXTENTS.SEGMENT\_TYPE = 'TABLE'

AND DBA\_EXTENTS.SEGMENT\_NAME = 'HBMS\_GMCC\_ORDER'

GROUP BY

DBA\_DATA\_FILES.FILE\_NAME

ORDER BY

SUM(DBA\_EXTENTS.BLOCKS);



首先，对于直接使用物理存储，单个表来说，无论是索引扫描和全表扫描，都是按块读写的，块与块之间是连续的，是有利于提升读写性能的(磁头寻址)。

如果是多个表且有多个磁盘，那么将同一个表的不同数据块或者不同表的不同数据块相对均匀地分布到不同的磁盘上，并行读写多个磁盘，是有利于提升读写性能的。

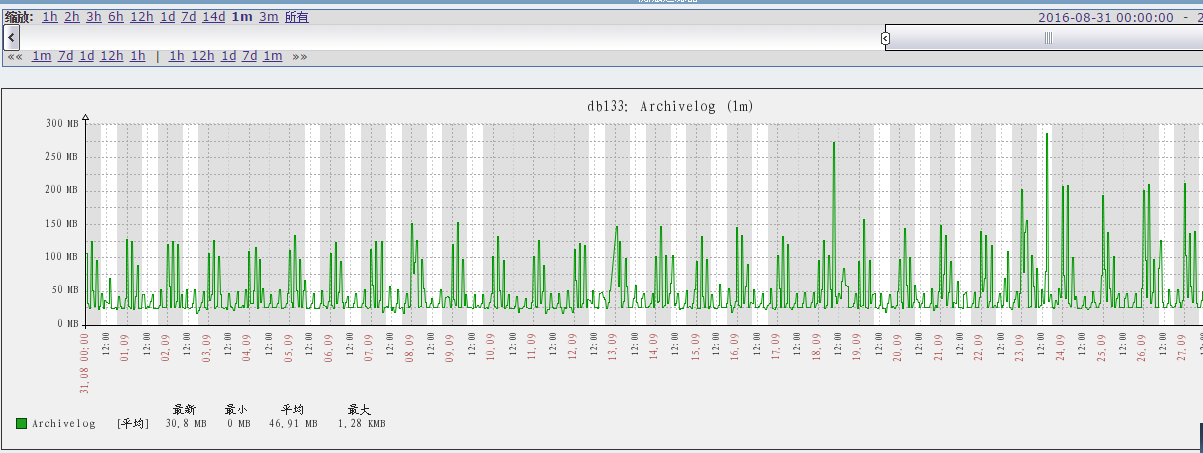
## 如何判断一个表是查询型的表还是修改型的表

# 数据库参数

## redo的数目、大小的设置

### 从归档日志量看业务高峰的分布

以下是zabbix监控到的Oracle的归档日志量。

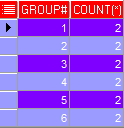


如果希望业务高峰期不会发生检查点的等待事件，防止出现DML等待，那么在线重做日志的大小最好能够支撑业务高峰期的修改量。

### 查看在线重做日志目前的配置

|  |
| --- |
| SELECT  A.GROUP#,  A.MEMBERS,  B.MEMBER,  A.BYTES/1024/1024 AS MB,  A.ARCHIVED,  A.STATUS  FROM  V$LOG A,  V$LOGFILE B  WHERE  A.GROUP# = B.GROUP# |

SELECT GROUP#,COUNT(\*) FROM V$LOGFILE GROUP BY GROUP# ORDER BY GROUP#;



### 查看在线重做日志切换的记录

这里有一个问题，在线重做日志什么时候切换，实际上，是其中一组写满了就会切换。这个要跟LGWR写在线日志的条件区分开来。

SELECT

SEQUENCE#,

FIRST\_TIME,

NEXTTIME,

ROUND(((FIRST\_TIME-NEXTTIME)\*24)\*60,2) DIFF

FROM

(

SELECT

SEQUENCE#,

FIRST\_TIME,

LAG(FIRST\_TIME) OVER(ORDER BY SEQUENCE#) NEXTTIME

FROM

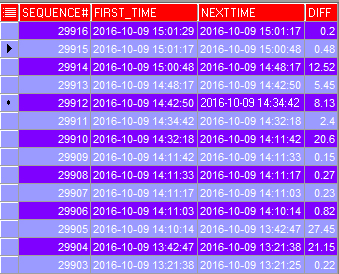
V$LOG\_HISTORY

WHERE

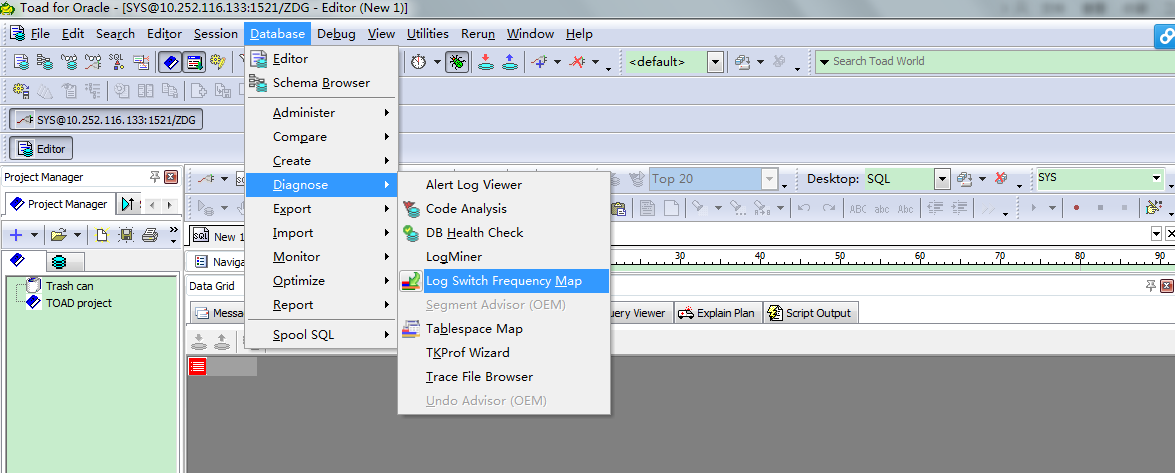
THREAD#=1)

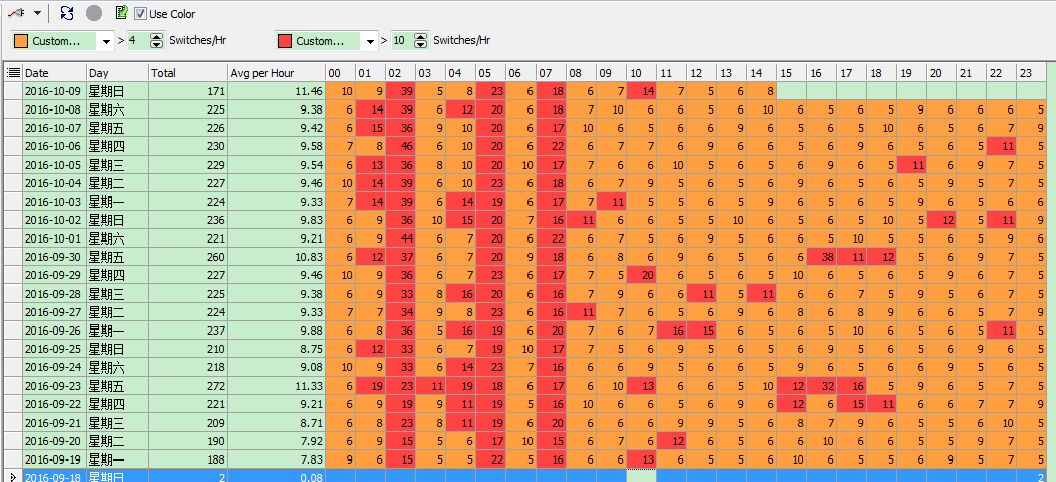
ORDER BY

SEQUENCE# DESC;



### 利用Toad查看在线重做日志的切换频率





按照Oracle的官方建议：15~20分钟切换一次是比较好的，太多和太少都不好。另外，在Alert log上可以看到频繁的online log切换，如果同时连带出现“check point not complete”或者“could not allocate log sequence”提示，说明日志切换过于频繁。因为日志文件相对于业务量过小，而几组日志是循环写的。之前的日志都还没有来得及写归档，就又要求切换了，DML操作不得不等待，造成性能的瓶颈。这时，可以通过增大在线重做日志文件的大小，或者增加在线重做日志的组数，来解决。

### 变更redo log的配置(包括组数，每组文件路数，文件大小)

当redo log文件很小，并且组数也不多，数据库的修改量一大，就很容易导致以下错误：

Thread 1 cannot allocate new log, sequence 36685。 Checkpoint not complete。

但直接增加redo log文件的大小所不行的，而所通过折衷的方式进行。

查看redo log的组数

|  |
| --- |
| SELECT GROUP#, MEMBERS, BYTES/1024/1024 BYTE\_MB, STATUS FROM V$LOG; |

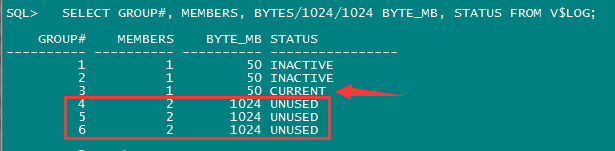


增加新的日志组及组内文件。注意，redo log的大小是有要求的，不能太小，也不能太大。

|  |
| --- |
| ALTER DATABASE ADD LOGFILE GROUP 4 ('/oracle/oradata/orcl11g/redo0401.log','/oracle/oradata/orcl11g/redo0402.log') SIZE 1024M;  ALTER DATABASE ADD LOGFILE GROUP 5 ('/oracle/oradata/orcl11g/redo0501.log','/oracle/oradata/orcl11g/redo0502.log') SIZE 1024M;  ALTER DATABASE ADD LOGFILE GROUP 6 ('/oracle/oradata/orcl11g/redo0601.log','/oracle/oradata/orcl11g/redo0602.log') SIZE 1024M; |

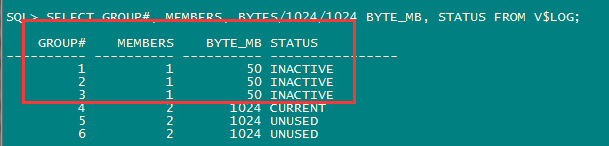
查看添加是否成功以及每一组的状态

|  |
| --- |
| SELECT GROUP#, MEMBERS, BYTES/1024/1024 BYTE\_MB, STATUS FROM V$LOG; |



切换日志以及做checkpoint

|  |
| --- |
| ALTER SYSTEM SWITCH LOGFILE;  ALTER SYSTEM CHECKPOINT; |



删掉原来的日志组

|  |
| --- |
| ALTER DATABASE DROP LOGFILE GROUP 1;  ALTER DATABASE DROP LOGFILE GROUP 2;  ALTER DATABASE DROP LOGFILE GROUP 3; |

看切换是否正常了



# 数据迁移

目前的迁移策略是怎样的，补充具体做迁移的脚本？

目前为止只做过一次数据迁移是开发提供迁移表，然后将历史数据迁往历史库

迁移133库的大表：

MEGSOFT2用户的-[+

source\_boss\_worknote\_total

source\_boss\_worknote

source\_boss\_worknote –此表较小而且有做分区，直接使用pl/sql developer导出txt文件，再连接历史库，导入txt文件

source\_boss\_worknote\_total --此表较大可先根据需要过期处理的数据，创建临时表，再使用expdp导出，然后用impdp导入

create table tmp\_worknote\_total as select \* from source\_boss\_worknote\_total where worknotetime < '2015-01-01' ;

expdp导出：

expdp MEGSOFT2/dgitc#12 directory=dirdp1 dumpfile= tmp\_worknote\_total logfile=tmp\_worknote\_total.log tables=MEGSOFT2.tmp\_worknote\_total EXCLUDE=INDEX,constraint VERSION=10.2.0.5.0

说明：由于导出库133库是11G，导入库13.2是10G的故需加上导出参数VERSION

impdp导入：

impdp MEGSOFT2/dgitc#12 dumpfile=tmp\_worknote\_total directory=dirdp1 logfile=tmp\_worknote\_total.log

导入后将数据插入到表：source\_boss\_worknote\_total

Insert into source\_boss\_worknote\_total select \* from tmp\_worknote\_total;

Commit;

# 清除垃圾数据

## 监听日志文件

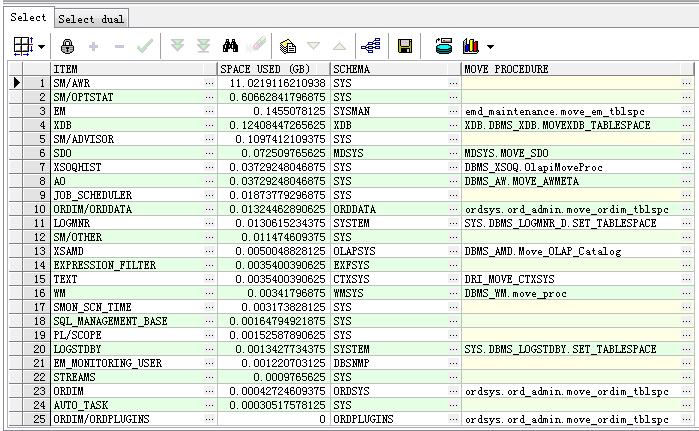
## 告警日志文件

## 各种dump\_dest

## SYSAUX垃圾

### 查看SYSAUX各组件对空间的使用率

|  |
| --- |
| SELECT  OCCUPANT\_NAME "ITEM",  SPACE\_USAGE\_KBYTES / 1048576 "SPACE USED (GB)",  SCHEMA\_NAME "SCHEMA",  MOVE\_PROCEDURE "MOVE PROCEDURE"  FROM  V$SYSAUX\_OCCUPANTS  ORDER BY  2 DESC; |



### 看哪个模块空间使用比较多，就进行哪个模块的处理

AWR报告占用空间多，那就清理AWR快照的数据，并且设置采集策略和保留策略。