OCP 052 课堂笔记

第一部分：Oracle 体系架构

第一章：实例和数据库
第二章：实例管理及数据库的启动/关闭
第三章：控制文件
第四章：redo 日志
第五章：归档日志 archivelog
第六章：日志挖掘 logminer
第七章：管理 undo
第八章：检查点 checkpoint
第九章：实例恢复机制

第二部分：Oracle 存储架构

第十章：　数据字典和动态视图
第十一章：Oracle 的存储架构
第十二章：Oracle 中表的几种类型
第十三章：数据库审计 audit
第十四章：数据装载 SqlLoader
第十五章：Oracle 网络

第三部分：管理 Oracle 数据库

第十六章：Oracle ASM 管理
第十七章：逻辑备份与恢复
第十八章：物化视图

第一章：实例与数据库

1、Oracle 网络架构及应用环境

看 ppt,Oracle 结构的基本单元、术语

2、Oracle 体系结构

1）oracle server ：database + instance

2）database：data file 、control file 、redolog file

3）instance: an instance access a database

4）oracle memory: sga + pga

5）instance ： sga + backgroud process

6)sga 组成:sga 在一个 instance 只有一个 sga,sga 为所有 session 共享,随着 instance 启动而分配，instance down ， sga 被释放。

3、SGA

3.1 SGA 的 6 个基本组件:

1） shared pool（PPT-II-331）

共享池是对 SQL、PL/SQL 程序进行语法分析、编译、执行的内存区域。
共享池由库缓存（library cache）,和数据字典缓存（data dictionary cache）以及结果缓存（result cache）等组成。
共享池的大小直接影响数据库的性能。

关于 shared pool 中的几个概念

library cache： sql 和 plsql 的解析场所，存放着所有编译过的 sql 语句代码，以备所有用户共享。

data dictionary cache： 存放重要的数据字典信息，以备数据库使用

server result cache: 存放服务器端的 SQL 结果集及 PL/SQL 函数返回值

User Global Area (UGA) 与共享服务器模式有关

2） database buffer cache

用于存储从磁盘数据文件中读入的数据，为所有用户共享。
服务器进程（server process)将读入的数据保存在数据缓冲区中，当后续的请求需要这些数据时可以在内存中找到，则不需要再从磁盘读取。
数据缓冲区中被修改的数据块（脏块）由后台进程 DBWR 将其写入磁盘。
数据缓冲区的大小对数据库的读取速度有直接的影响。

要弄明白 Database Buffer Cache 中的几个 cache 概念：

Buffer pool=(default pool)+(nodefault pool)

其中：
default pool（db_cache_size） //是标准块存放的内存空间大小，SGA 自动管理时此参数不用设置。使用 LRU 算法清理空间

nodefault pool:

db_nk_cache_size //指定非标准块大小内存空间，比如 2k、4k、16k、32k。
db_keep_cache_size //keep 存放经常访问的小表或索引等。
db_recycle_cache_size //与 keep 相反，存放偶尔做全表扫描的大表的数据。

```
SQL> alter table scott.emp1 storage(buffer_pool keep);

SQL> select segment_name,buffer_pool from dba_segments where segment_name='EMP1';

SEGMENT_NAME
BUFFER_
--------------------------------------------------------------------------------
- -------
EMP1
KEEP
```

2.2）default pool 对应的参数是 db_cache_size 与标准块 default block 是配套的，如果 default block 是 8k, db_cache_size 这个参数将代替 db_8k_cache_size。

如果要建立非标准块的表空间，先前要设定 db buffer 中的与之对应的 db_nk_cache_size 参数。

09:50:46 SQL> alter system set db_16k_cache_size=8m; //改参数，先把 db buffer 里的 16k cache 建上。

09:50:49 SQL> create tablespace tbs_16k datafile '/u01/oradata/timran11g/tbs16k01.dbf' size 10m blocksize 16k;

09:51:29 SQL> select TABLESPACE_NAME,block_size from dba_tablespaces;

TABLESPACE_NAME	BLOCK_SIZE
SYSTEM	8192
UNDOTBS1	8192
SYSAUX	8192
TEMP	8192
USERS	8192
EXAMPLE	8192
TBS_16K	16384

2.3）查看 buffer cache 命中率：

```
18:28:20                SQL>select        (1-(sum(decode(name,        'physical
reads',value,0))/(sum(decode(name, 'db block gets',value,0))+
sum(decode (name,'consistent gets',value,0))))) * 100 "Hit Ratio" from v$sysstat;

 Hit Ratio
----------
97.6811923
```

3）redo log buffer

日志条目（redo entries ）记录了数据库的所有修改信息(包括 DML 和 DDL）， 为的是数据库恢复，日志条目首先产生于日志缓冲区。 日志缓冲区较小，它是以字节为单位的，它极其重要。

```
18:29:04 SQL>  show parameter log_buffer

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_buffer                           integer     7057408
```

*考点:日志缓冲区的大小启动后就是固定不变的，如要调整只能通过修改参数文件后重新启动生效。不能动态修改！不能由 SGA 自动管理！

如果想让它是一个最小值，这样可以做：

```
18:30:24 SQL> alter system set log_buffer =1 scope=spfile;   //修改动态参数文件，
下次启动有效。
18:31:20 SQL> startup force
18:31:35 SQL> show parameter log_buffer;

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
log_buffer                           integer     2927616            //这就是最小
值了
```

4）large pool（可选）

为了进行大的后台进程操作而分配的内存空间，与 shared pool 管理不同，主要用于共享服务器的 session memory，RMAN 备份恢复以及并行查询等。

5）java pool(可选)

为了 java 虚拟机及应用而分配的内存空间，包含所有 session 指定的 JAVA 代码和数据。

6) stream pool（可选）

为了 stream process 而分配的内存空间。stream 技术是为了在不同数据库之间共享数据，因此，它只对使用了 stream 数据库特性的系统是重要的。

3.2 sga 的 granules(颗粒)：组成 oracle 内存的最小单位

SGA_MAX_SIZE	Granule Size
<=1 GB	4 MB
1GB -- 8GB	16 MB
8GB --16GB	32 MB
16GB--32GB	64 MB
32GB--64GB	128 MB
64GB--128GB	256 MB
>128GB	512 MB

20:12:30 SQL> select name ,bytes/1024/1024 "Size(M)" from v$sgainfo; //在 oracle 里查看 SGA 分配情况

NAME	Size(M)
Fixed SGA Size	1.2401123
Redo Buffers	1.84765625
Buffer Cache Size	56
Shared Pool Size	152
Large Pool Size	4
Java Pool Size	12
Streams Pool Size	4
Shared IO Pool Size	0
Granule Size	4
Maximum SGA Size	403.089844
Startup overhead in Shared Pool	40
Free SGA Memory Available	172

4、Oracle 的进程:

三种 process： 1）user process、 2）server process 、3）background process

user process：属于客户端的 process，一般分为三种形式，1）sql*plus, 2)应用程序，3）web 方式（OEM）

客户端请求，sqlplus 是客户端命令。

*考点：由 user process 造成的会话终止，系统将自动回滚这个会话上的处于活动状态的事务。

如果 windows 作为客户端：可以通过查看任务管理器看到 sqlplus 用户进程:

C:\Documents and Settings\timran>sqlplus sys/system@timran11g as sysdba

linux 作为客户端时可以使用 ps 看到 sqlplus 关键字:

```
[oracle@timran ~]$ ps -ef |grep sqlplus
oracle    2353  2325  0 17:02 pts/0    00:00:00 rlwrap sqlplus / as sysdba
oracle    2354  2353  0 17:03 pts/1    00:00:00 sqlplus   as sysdba
oracle    2603  2445  0 17:25 pts/2    00:00:00 grep sqlplus
```

server process：这是服务器端的进程，user process 不能直接访问 Oracle，必须通过相应的 server process 访问实例，进而访问数据库。

```
[oracle@timran ~]$ ps -ef |grep LOCAL
oracle       2399    2354    1  17:03  ?              00:00:04  oracletimran11g
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
oracle    2503     1  0 17:05 ?        00:00:00 oracletimran11g (LOCAL=NO)
oracle    2512  2445  0 17:07 pts/2    00:00:00 grep LOCAL
[oracle@timran ~]$
```

//注意：在 linux 下看到的 server process, (LOCAL=YES)是本地连接，(LOCAL=NO)是远程连接。

可以在 oracle 查看 V$process 视图，它包括了当前所有的 oracle 进程，即后台进程和服务器进程。

```
SQL> select pid,program,background from v$process;
```

background 字段为 1 是 background process，其余都是 server process

六个基本的后台进程（background process）

smon：系统监控进程

在实例崩溃之后,Oracle 会自动恢复实例。另一个作用是释放不再使用的临时段。

pmon：进程监控

1、当 user process 失败时，清理出现故障的进程。 释放所有当前挂起的锁定。释放服务器端使用的资源

2、监控空闲会话是否到达阀值

3、动态注册监听

dbwr：数据写入进程

1、将修改后的缓冲区（脏 buffer)数据写入数据文件中。写脏块。

2、释放 data buffer 空间。

注意：以下几种情况发生时 dbwr 都会写

1）ckpt 发生，2）脏块太多时，3）db_buffer 自由空间不够时，4）3 秒，5）表空间 read only/offline/backup 等

考点：
1)服务器进程对数据文件执行读操作，而 DBWR 负责对数据文件执行写操作。
2)commit 时 dbwn 有何举动？答案是：它什么也不做！

lgwr：写日志条目，从 redo log buffer 到 redo logfile （必须在 dbwr 写脏块之前写入日志）

负责将日志缓冲区中的日志条目写入日志文件。 有多个日志文件，该进程以循环的方式将数据写入文件。

注意：以下 5 个状况发生时， lgwr 都会写

1）commit， 2)三分之一满，3）先于 dbwr 写而写（先记后写，即 dbwn 正好要执行写入前），4）3 秒（由先记后写引发）

ckpt：生成检查点， 通知或督促 dbwr 写脏块

完全检查点：保证数据一致性。增量检查点：不断更新控制文件中的检查点位置，当发生实例崩溃时，可以尽量缩短实例恢复的时间。

arcn：归档模式下，发生日志切换时，把当前日志组中的内容写入归档日志，作为备份历史日志。

考点：lgwr 负责对联机日志文件写操作，arcn 负责读取联机日志文件。其他进程与日志文件不接触。

11g 里又强调了其他几个后台进程，它们都和数据库性能有关，有关内容将在 053 课程介绍：

MMON：Oracle 自我监视和自我调整的支持进程(与 AWR 有关)

MMNL：MMON 的辅助进程(与 ASH 有关)

MMAN：内存自动管理，10g 时推出，11g 得到加强，在 11g 里这个进程负责 Oracle 内存结构（SGA+PGA）的自动调整。

CJQN: 与 job 队列有关

5、PGA

属于 oracle 内存结构，存放用户游标、变量、控制、数据排序、存放 hash 值。与 SGA 不同，PGA 是独立的，非共享。

6、用户与 Oracle 服务器的连接方式

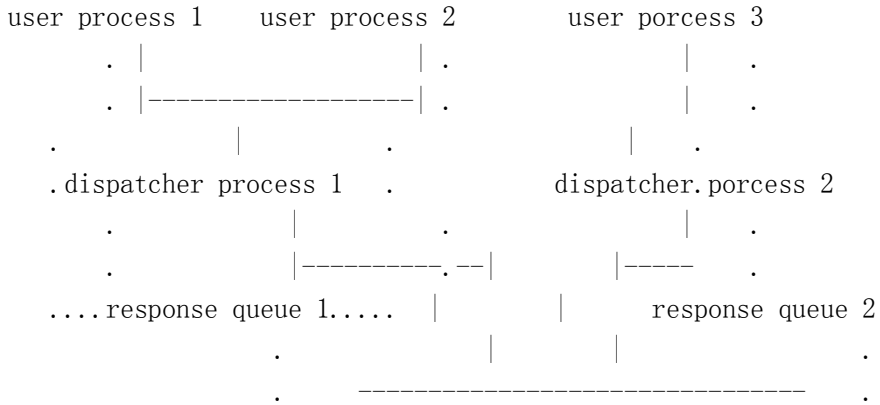
6.1 专用连接模式(dedicated server process)：(PPT-I-218)
对于客户端的每个 user process，服务器端都会出现一个 server process，会话与专用服务器之间存在一对一的映射。

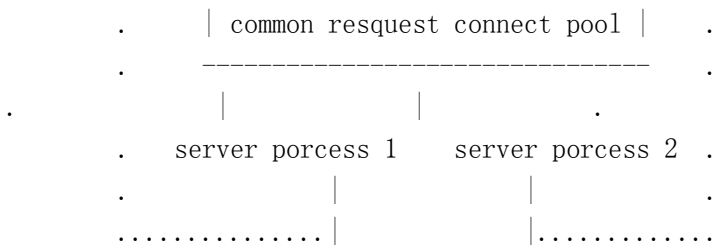
对专用连接来说，用户在客户端启动了一个应用程序，例如 sql*plus，就是在客户端启动一个用户进程；与 oracle 服务器端连接成功后，会在服务器端生成一个服务器进程，该服务器进程作为用户进程的代理进程，代替客户端执行各种命令并把结果返回给客户端。用户进程一旦中止，与之对应的服务器进程立刻中止。

专用连接的 PGA 的管理方式是私有的。Oracle 缺省采用专用连接模式。

6.2 共享连接模式(shared server process)：(PPT-I-219-222)

多个 user process 共享一个 server process。它通过调度进程（dispatcher)与共享服务器连接,共享服务器实际上就是一种连接池机制（connectionpooling),连接池可以重用已有的超时连接，服务于其它活动会话。但容易产生锁等待。此种连接方式现在已经很少见了，但是在 OCP11g 考试中有几个考点与其有关:(具体配置方法见第十五章 Oracle 网络)。





上图实线表示发送信息，虚线表示返回信息。

*考点：
1）所有调度进程(dispatcher)共享一个公共的请求队列(resquest queue)，但是每个调度进程都有与自己响应的队列(response queue)。
2）在共享服务器中会话是在 SGA 中的（UGA）存储信息，而不像专用连接那样在 PGA 中存储信息，这时的 PGA 的存储结构为堆栈空间。

6.3 驻留连接池模式（database resident connection pooling，简称 DRCP）：
适用于必须维持数据库的永久连接。结合了专用服务器模式和共享服务器模式的特点，它提供了服务器连接池，但是放入连接池的是专用服务器。它使用连接代理（而不是专用服务器）连接客户机到数据库，优点是可以用很少的内存处理大量并发连接（11g 新特性，特别适用于 Apache 的 PHP 应用环境）。

第二章：实例管理及数据库的启动/关闭

2.1 实例和参数文件

1、instance 功能:用于管理和访问 database。instance 在启动阶段读取初始化参数文件（init parameter files)。

2、init parameter files ：管理实例相关启动参数 。位置：$ORACLE_HOME/dbs

3、pfile ：(parameter file）静态参数文件。

1）文本文件，必须通过编辑器修改参数。 2）修改参数下次重启实例才生效。3）pfile 参数文件可以不在 database server 上。

命名方式：init+SID.ora

4、spfile ：(system parameter file) 动态参数文件。

1）二进制文件，不可以通过编辑器修改。2)Linux 下 strings 可以查看 。3）必须在 database server 段的指定路径下。

命名方式: spfile+SID.ora

静态参数和动态参数

在 spfile 读到内存后，有一部分参数是可以直接在内存中修改，并对当前 instance 立即生效，这样的参数叫动态参数。除了动态参数都是静态参数。静态参数修改 spfile 文件。动态参数在 instance 关闭后失效，而静态参数是下次 instance 启动后才生效。

修改 spfile 文件的方法：

alter system set 参数=值 [scope=memory|spfile|both]

alter system reset 参数 [scope=memory|spfile|both] SID='*'　　//恢复缺省值。

第一种 scope=memory 参数修改立刻生效，但不修改 spfile 文件。
第二种 scope=spfile 修改了 spfile 文件，重启后生效。
第三种 scope=both 前两种都要满足。

如果不写 scope，即缺省，那就是第三种。

*考点：如果不修改 spfile，将无法更改静态参数。

通过查看 v$parameter ，可以确定哪些参数可以在 memory 修改，制定 scope。

```
10:38:35 SQL> desc v$parameter;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NUM                                                NUMBER
 NAME                                               VARCHAR2(80)
 TYPE                                               NUMBER
 VALUE                                              VARCHAR2(512)
 DISPLAY_VALUE                                      VARCHAR2(512)
 ISDEFAULT                                          VARCHAR2(9)
 ISSES_MODIFIABLE                                   VARCHAR2(5)
 ISSYS_MODIFIABLE                                   VARCHAR2(9)
 ISINSTANCE_MODIFIABLE                              VARCHAR2(5)
 ISMODIFIED                                         VARCHAR2(10)
 ISADJUSTED                                         VARCHAR2(5)
 ISDEPRECATED                                       VARCHAR2(5)
 DESCRIPTION                                        VARCHAR2(255)
 UPDATE_COMMENT                                     VARCHAR2(255)
 HASH                                               NUMBER
```

其中：

ISSYS_MODIFIABLE 参数：对应 alter system 命令，即系统级修改

10:38:35 SQL> select distinct issys_modifiable from v$parameter;

```
ISSYS_MODIFIABLE
-----------------
IMMEDIATE           //对应 scope=memory
FALSE               //只能 scope=spfile,即修改 spfile 文件，下次启动才生效。
DEFERRED            //其他 session 有效
```

ISSES_MODIFIABLE 参数：对应 alter session 命令，即 session 级修改

10:38:35 SQL> select distinct isses_modifiable from v$parameter;

```
ISSES_MODIFIABLE
-----------------
TRUE                //表示可以修改
FALSE               //表示不能修改
```

10:38:35 SQL> select ISSES_MODIFIABLE,ISSYS_MODIFIABLE from v$parameter where name='sql_trace';

```
ISSES ISSYS_MOD
----- ---------
TRUE  IMMEDIATE
```

这个结果表示 sql_trace 参数在 session 级别可以改，在 system 级也可以 both 修改（动态参数）。

5、startup 时读取参数文件，找到$ORACLE_HOME/dvs 目录，顺序是优先 spfile 启动，没有 spfile 才从 pfile 启动。

pfile 和 spfile 可以相互生成：

```
SQL>create pfile from spfile
SQL>create spfile from pfile（使用 spfile 启动后不能在线生成 spfile，ORA-32002: 无法创建已由实例使用的 SPFILE）
```

*考点：

1)如果使用 pfile 启动,设置 scope=spfile 将失败!但可以设置 scope=memory,要修改 pfile 文件，使用文本编辑工具。

可以通过当前内存参数生成 pfile 和 spfile（11g 新特性）：

```
SQL>create pfile from memory;
SQL>create spfile from memory;
```

有了 spfile，pfile 一般留做备用，特殊情况也可以使用 pfile 启动,命令如下：

```
10:38:35 SQL> startup pfile=$ORACLE_HOME/dbs/inittimran.ora
```

怎样知道实例是 spfile 启动还是 pfile 启动的?

```
10:38:35 SQL> show parameter spfile
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
spfile                               string      /u01/app/oracle/product/10.2.0
                                                 /db_1/dbs/spfileprod.ora
```

//如果 value 有值，说明数据库启动时读的是 spfile

另一个办法是看 v$spparameter(spfile 参数视图)中的参数 memory_target 的 isspecified 字段值，如果是 TRUE 说明是 spfile 启动的（考点）

```
10:42:35 SQL> select name,value,isspecified from v$spparameter where name like
'memory_target';

NAME                                                                 VALUE
ISSPECIFIED
-------------------------------------------------------------------------------
---------------------------------------------
memory_target                                                        423624704
TRUE
```

OEM 对初始参数有较好的可视化界面，可以看看，

2.2 数据库启动与关闭：

2.2.1 启动分为三个阶段

1）nomount 阶段：读取 init parameter

```
10:38:35 SQL> select status from v$instance;     (这条命令很实用，是看当前数据库
启动的状态，有三个 stated,mounted,open.)
STATUS
```

```
------------
STARTED
```

2）mount 阶段： 读取控制文件

```
20:32:53 SQL> select status from v$instance;
STATUS
------------
MOUNTED
```

3）open 阶段： 1、检查所有的 datafile、redo log、 group 、password file。
2、检查数据库的一致性（controlfile、datafile、redo file 的检查点是否一致）

```
10:38:35 SQL> select file#,checkpoint_change# from v$datafile;      //从控制文件读出
     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1             570836
         2             570836
         3             570836
         4             570836
         5             570836
         6             570836

6 rows selected.

10:38:35 SQL> select file#,checkpoint_change# from v$datafile_header;      //从datafile header 读出

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1             570836
         2             570836
         3             570836
         4             570836
         5             570836
         6             570836

6 rows selected.
```

数据库 open 之前要检查 controlfile 所记录 SCN 和 datafile header 记录的 SCN 是否一致；一致就正常打开库，不一致需要做 media recover

```
10:38:35 SQL> select status from v$instance;
```

```
STATUS
------------
OPEN
```

2.2.2 启动数据库时的一些特殊选项

```
SQL> alter database open read only;
SQL> startup force
SQL> startup upgrade    (只有 sysdba 能连接)
SQL> startup restrict    (有 restrict session 权限才可登录，sys 不受限制)
SQL> alter system enable restricted session;  (open 后再限制)
```

2.2.3 实例关闭：

shutdown normal： 拒绝新的连接，等待当前事务结束，等待当前会话结束，生成检查点

shutdown transaction ：拒绝新的连接，等待当前事务结束，生成检查点

shutdown immediate： 拒绝新的连接，未提交的事务回滚，生成检查点

shutdown abort（startup force）：事务不回滚，不生成检查点，下次启动需要做 instance recovery

*考点：shutdown abort 不会损坏 database。

2.3 自动诊断信息库 ADR（Automatic Diagnostic Repository）11g 新特性

存储在操作系统下的一个目录（树）结构，包括：预警日志文件，跟踪文件，健康检查，DUMP 转储文件等

11g 用 DIAGNOSTIC_DEST 一个参数代替了许多老版本的参数，如 BACKGROUND_DUMP_DEST，CORE_DUMP_DEST，USER_DUMP_DEST 等。

```
SQL> show parameter diag

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
diagnostic_dest                      string      /u01
```

考点：这是 ADR 的基目录，如果你设置了 ORACLE_BASE 环境变量，则 diagnostic_dest 默认值被设置为同样的目录，否则，oracle 给你设置的目录是$ORALE_HOME/log

```
10:38:35 SQL> show parameter dump       //这是 Oracle11g 的。
SQL> show parameter dump
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
background_core_dump                 string      partial
background_dump_dest                 string      /u01/diag/rdbms/timran11g/timr
                                                 an11g/trace
core_dump_dest                       string      /u01/diag/rdbms/timran11g/timr
                                                 an11g/cdump
max_dump_file_size                   string      unlimited
shadow_core_dump                     string      partial
user_dump_dest                       string      /u01/diag/rdbms/timran11g/timr
                                                 an11g/trace
```

在 oracle 11g 中，故障诊断及跟踪的文件路径改变较大，告警文件分别以两种文件格式存在，xml 的文件格式和普通文本格式。这两份文件的位置分别是 V$DIAG_INFO 中的 Diag Alert 和 Diag Trace 对应的目录。

如果熟悉 9i 的话，你会发现 11g 将 bdump 和 udump 合并到一个目录 /u01/diag/rdbms/timran11g/timran11g/trace 下了。

其命名方法依然是

Background Trace Files（针对 bg process）:SID_processname_PID.trc 如 : timran11g_m001_5616.trc
User Trace Files(针对 server process) :SID_ora_PID.trc 如 : timran11g_ora_10744.trc

另外增加.trm（trace map)文件，记录 trc 文件的结构信息。

```
SQL> select * from v$diag_info;

   INST_ID NAME                                     VALUE
---------- ---------------------------------------- ------------------------
----------------------------------------
         1 Diag Enabled                             TRUE
         1 ADR Base                                 /u01
         1                                  ADR                                 Home
/u01/diag/rdbms/timran11g/timran11g
         1                                 Diag                                 Trace
    /u01/diag/rdbms/timran11g/timran11g/trace
         1                                 Diag                                 Alert
/u01/diag/rdbms/timran11g/timran11g/alert
         1                                Diag                                Incident
/u01/diag/rdbms/timran11g/timran11g/incident
```

```
         1                                Diag                                Cdump
/u01/diag/rdbms/timran11g/timran11g/cdump
         1 Health Monitor
/u01/diag/rdbms/timran11g/timran11g/hm
         1 Default Trace File
         1 Active Problem Count                                               0
         1 Active Incident Count                                              0
```

其中 Diag Trace 对应的目录里为文本格式的告警日志和跟踪文件，并沿用在 10g 中命名方法

告警日志：alter_SID.log 包含通知性的消息，如数据库启动或关闭，以及有关表空间创建和删除的信息,也包括一些内部错误信息等。
alter_SID.log 不断增长，定期清理是必要的

```
$cat dev/null > alert_timran11g.log      //将文件清空
```
或
```
$rm alter_timran11g.log              //删掉也没有关系，下次启动会自动创建（考点）
```

检查告警日志和跟踪文件的有关错误信息的记录，如 lwgr 不能写日志组时，会创建一个跟踪文件，并将一条信息放入告警日志。

```
[oracle@timran                  trace]$                  tail                  -f
/u01/diag/rdbms/timran11g/timran11g/trace/alert_timran11g.log
space available in the underlying filesystem or ASM diskgroup.
Tue Sep 04 09:12:19 2012
Completed: ALTER DATABASE OPEN
Tue Sep 04 09:16:41 2012
Starting background process CJQ0
Tue Sep 04 09:16:41 2012
CJQ0 started with pid=29, OS id=2483
Tue Sep 04 10:19:11 2012
drop tablespace tb1
Completed: drop tablespace tb1
```

ADR 的概念在 053 试题中较多，因为它涉及了 11g 在数据库自动管理方面的一些重要知识，如度量，阀值，预警系统，健康监测等等，我们在 053 课程里会继续介绍。

2.4 口令文件

oracle 登录认证方式有多种

2.4.1 sys 的两种常用的登录认证方式：OS 认证和口令文件认证。

1）OS 认证：本地认证方式。Oracle 不验证用户密码，前提：用户必须属于 DBA 组，且使用本地登录。

如：sqlplua / as sysdba

2）口令文件认证：是一种网络远程认证方式，只有 sysdba 权限的用户可以使用口令文件，必须输入密码和网络连接符。

如：sqlplus sys/oracle@timran11g as sysdba

2.4.2 普通用户登录

1）普通用户是指没有 sysdba 权限的用户，比如 system 、scott,或者是 tim 什么的，登录都必须输入口令，不管是本地还是远程，它们的口令密码不是以文件形式存放的，而是由 oracle 保管在其内部的数据字典里。

2）通过设置这个参数为 TURE，可以让口令是大小写敏感的（11g 新特性）

```
SQL> show parameter case

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
sec_case_sensitive_logon             boolean     TRUE
```

本节讨论的口令文件是 sysdba 用户的远程登录认证密码文件，主要用于 sys 用户远程登录的认证。

位置：$ORACLE_HOME/dbs/orapwSID， 所谓口令文件，指的就是 sys 的口令文件，可以通过 remote_login_passwordfile 参数控制是否生效

参数 remote_login_passwordfile 的三种模式：

1）none 拒绝 sys 用户从远程连接
2）exclusive sys 用户可以从远程连接
3）share 多个库可以共享口令文件

```
[oracle@timran ~]$ cd /u01/oracle/dbs
[oracle@timran dbs]$ ll
总计 52
-rw-rw---- 1 oracle oinstall  1544 08-17 07:19 hc_timran11g.dat
-rw-r--r-- 1 oracle oinstall 12920 2001-05-03 initdw.ora
```

```
-rw-r--r-- 1 oracle oinstall  8385 1998-09-11 init.ora
-rw-r--r-- 1 oracle oinstall  1024 08-17 13:23 inittimran11g.ora
-rw-r----- 1 oracle oinstall    24 08-17 07:21 lkTIMRAN11
-rw-r----- 1 oracle oinstall    24 08-17 10:36 lkTIMRAN11G
-rw-r----- 1 oracle oinstall  1536 08-31 10:47 orapwtimran11g
-rw-r----- 1 oracle oinstall  3584 09-04 17:49 spfiletimran11g.ora
```

这里是放参数文件和（sys)口令文件的地方,orapwtimran11g 就是我的 sys 口令文件

使用 orapwd 命令创建新的 sys 口令文件：

你可以先删掉它，再创建它，在 linux 下做：

```
[oracle@timran dbs]$ rm orapwtimran11g       //把 sys 口令文件删了

[oracle@timran dbs]$orapwd file=orapwtimran11g password=oracle entries=5 force=y
//重新建立口令文件
```

注意：file=orapw+sid 的写法

entries 的含义是表示口令文件中可包含的 SYSDBA/SYSOPER 权限登录的最大用户数。

2.5 添加 scott 案例

有时候，scott 用户被误删掉了，不要紧，可以通过执行下列脚本重新建立。

```
10:38:35 SQL> @$ORACLE_HOME/rdbms/admin/utlsampl.sql
```

大家可以试一下：先删除 scott 用户，再执行一下脚本即可。（不是所有案例都有恢复脚本，HR 就没有）

第三章：控制文件

3.1 控制文件的功能和特点：

1）定义数据库当前物理状态
2）维护数据的一致性
3）是一个二进制文件
4）在 mount 阶段被读取
5）记录 RMAN 备份的元数据

查看 database 控制文件位置：

```
19:02:27 SQL> show parameter control_file
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
control_file_record_keep_time        integer     7
control_files                                    string
/u01/oradata/timran11g/control01.
                                                 ctl,
/u01/oradata/timran11g/contr
                                                 ol02.ctl,
/u01/oradata/timran11g/
                                                 control03.ctl
19:02:42 SQL> select name from v$controlfile;

NAME
--------------------------------------------------------------------------------
--------------------
/u01/oradata/timran11g/control01.ctl
/u01/oradata/timran11g/control02.ctl
/u01/oradata/timran11g/control03.ctl
```

3.2 控制文件多元化

1）配置多个 control_files，控制文件最好有三个，是相互镜像的(shutdown 下 cp 命令复制即可)，然后修改 spfile 中的 control_files 参数，Oracle 建议分配在不同的物理磁盘上。

*考点：最多可以有 8 个 control files 多路复用。

```
19:10:25 SQL> alter system set
control_files='/u01/oradata/timran11g/control01.ctl','/u01/disk1/control02.ctl',
'/u01/disk1/control03.ctl' scope=spfile;

System altered.
```

2）三个 control 文件要一致（同一版本，scn 相同），本来就是镜像关系

```
-rw-r----- 1 oracle oinstall    7356416 07-16 20:00 control01.ctl
-rw-r----- 1 oracle oinstall    7356416 07-16 20:01 control02.ctl
-rw-r----- 1 oracle oinstall    7356416 07-16 20:01 control03.ctl
```

3）控制文件查看：v$controlfile 、show parameter controlfile、 v$parameter

3.3 控制文件的重建与备份

1）trace 文件：可以在 mount 或 open 模式下生成一个脚本，用于控制文件重建。

19:59:24 SQL> alter database backup controlfile to trace; //生成的 trace 文件在 udump 的最新 trc 文件里

或者 SQL>alter database backup controlfile to trace as '/u01/oradata/timran11g/con.trace'; //存到自己起的文件名里。

2）binary 文件：控制文件的备份
20:00:20 SQL> alter database backup controlfile to '/u01/oradata/timran11g/con.bak';

3.4 控制文件手工恢复

控制文件一但损坏，系统将崩溃或死机。

1）单个文件损坏了：数据库关闭，使用操作系统命令复制其副本到指定的位置。

2）所有的控制文件丢失，（正常关闭，shutdown immediate 后删除控制文件）

假设控制文件的备份也没有，利用前面做的 trace 文件重新生成控制文件

在 nomount 状态下执行 trace 脚本

```
15:37:16 SQL> startup force nomount
ORACLE instance started.

SQL>
CREATE CONTROLFILE REUSE DATABASE "TIMRAN11" NORESETLOGS  ARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 3
    MAXDATAFILES 100
    MAXINSTANCES 8
    MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/oradata/timran11g/redo01.log'  SIZE 50M,
  GROUP 2 '/u01/oradata/timran11g/redo02.log'  SIZE 50M,
  GROUP 3 '/u01/oradata/timran11g/redo03.log'  SIZE 50M
-- STANDBY LOGFILE
DATAFILE
  '/u01/oradata/timran11g/system01.dbf',
  '/u01/oradata/timran11g/sysaux01.dbf',
```

```
  '/u01/oradata/timran11g/user01.dbf',
  '/u01/oradata/timran11g/example01.dbf',
  '/u01/oradata/timran11g/test01.dbf',
  '/u01/oradata/timran11g/undotbs01.dbf'
CHARACTER SET ZHS16GBK
;
```

可以看到执行后三个控制文件又重新建立了。

说明：这个重建控制文件的过程主要有两大部分内容：

第一部分是脚本中的可见信息：1）定义了几个参数的最大值，2）在线日志的物理信息，3）数据文件的物理信息，4）使用的字符集。
第二部分是隐含的不可见信息，比如 SCN 信息，重建最关键的是将当前所有数据文件头部的最新 SCN 信息复制到了控制文件中。以便接下来打开数据库。

```
SQL> select file#,checkpoint_change# from v$datafile;        //从控制文件读出

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            5629150
         2            5629150
         3            5629150
         4            5629150
         5            5629150
         6            5629150


SQL> select file#,checkpoint_change# from v$datafile_header;     //从数据文件读出

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            5629150
         2            5629150
         3            5629150
         4            5629150
         5            5629150
         6            5629150


15:39:49 SQL> alter database open;
```

2）所有的控制文件丢失，利用控制文件备份（手动备份或 RMAN 备份都可以）恢复控制文件。（有点复杂，留在 053 备份恢复中再介绍吧）

第四章： redo 日志

4.1 redo （重做） log 的功能：数据 recovery
4.2 redo log 特征：
1）记录数据库的变化（DML、DDL）
2）用于数据块的 recover
3）以组的方式管理 redo file ，最少两组 redo ，循环使用
4）和数据文件存放到不同的磁盘上，需读写速度快的磁盘(比如采用 RAID10)

日志切换：
1）归档模式：将历史日志连续的进行保存。
2）非归档： 历史日志被覆盖
3）并产生 checkpoint，通知 redo log 所对应的 dirty block 从 data buffer 写入到 datafile，并且更新控制文件

4.3 redo 日志组
1）最少两组（考点），最好每组有两个成员（考点），并存放到不同的磁盘上，大小形同，互相镜像
2）日志在组写满时发生切换，或手工切换： alter system switch logfile ；
3）在归档模式，日志进行归档，并把相关的信息写入 controlfile

4.4 如何添加日志组

```
15:49:43 SQL> select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- --------------
         1             1           143     52428800             1  YES  INACTIVE
2144594 17-7 月 -12
         2             1           144     52428800             1  NO   CURRENT
2145200 17-7 月 -12
         3             1           142     52428800             1  YES  INACTIVE
2113981 17-7 月 -12

15:50:31 SQL> col member for a50;
15:50:47 SQL> select group#, member from v$logfile;

    GROUP# MEMBER
---------- --------------------------------------------------
         1 /u01/oradata/timran11g/redo01.log
```

```
         3 /u01/oradata/timran11g/redo03.log
         2 /u01/oradata/timran11g/redo02.log
```

增加一个组 group4,

```
15:53:53 SQL> alter database add logfile '/u01/oradata/timran11g/redo04.log' size
50m;

15:53:56 SQL> select group#, member from v$logfile order by group#;

    GROUP# MEMBER
---------- --------------------------------------------------
         1 /u01/oradata/timran11g/redo01.log
         2 /u01/oradata/timran11g/redo02.log
         3 /u01/oradata/timran11g/redo03.log
         4 /u01/oradata/timran11g/redo04.log


15:55:27 SQL> select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- --------------
         1              1           143     52428800              1  YES  INACTIVE
2144594 17-7 月  -12
         2              1           144     52428800              1  NO   CURRENT
2145200 17-7 月  -12
         3              1           142     52428800              1  YES  INACTIVE
2113981 17-7 月  -12
         4              1             0     52428800              1  YES  UNUSED
0
```

4.5 如何添加日志组的成员

加 member 为每个组（一共是 4 个组）
先建好目录，准备放在/u01/disk2/timran/下

```
[oracle@timran timran]$ mkdir -p /u01/disk2/timran
[oracle@timran timran]$

16:00:39 SQL> alter database add logfile member
'/u01/disk2/timran/redo01b.log' to group 1,
'/u01/disk2/timran/redo02b.log' to group 2,
```

'/u01/disk2/timran/redo03b.log' to group 3,
'/u01/disk2/timran/redo04b.log' to group 4;

SQL> select group#,member,status from v$logfile;

```
    GROUP#                                                                  MEMBER
STATUS
---------- ------------------------------------------------------------------------
---------- -------
         3 /u01/oradata/timran11g/redo03.log
         2 /u01/oradata/timran11g/redo02.log
         1 /u01/oradata/timran11g/redo01.log
         4 /u01/oradata/timran11g/redo04.log
         1                                          /u01/disk2/timran/redo01b.log
INVALID
         2                                          /u01/disk2/timran/redo02b.log
INVALID
         3                                          /u01/disk2/timran/redo03b.log
INVALID
         4                                          /u01/disk2/timran/redo04b.log
INVALID
```

16:01:54 SQL> select * from v$log; //看到 MEMBERS 列已经是 2 了

```
    GROUP#       THREAD#    SEQUENCE#           BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- --------------
         1             1           143     52428800             2  YES  INACTIVE
2144594 17-7 月 -12
         2             1           144     52428800             2  NO   CURRENT
2145200 17-7 月 -12
         3             1           142     52428800             2  YES  INACTIVE
2113981 17-7 月 -12
         4             1             0     52428800             2  YES  UNUSED
0
```

16:03:06 SQL> alter system switch logfile; //多做几次切换，消除 invalid //同步组里的 member，这步很重要。

4.6 如何查看日志信息

16:03:13 SQL> select * from v$log;

```
    GROUP#       THREAD#    SEQUENCE#          BYTES       MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- --------------
         1             1           143     52428800             2  YES  INACTIVE
2144594 17-7 月 -12
         2              1            144     52428800              2  YES  ACTIVE
2145200 17-7 月 -12
         3             1           142     52428800             2  YES  INACTIVE
2113981 17-7 月 -12
         4             1           145     52428800              2  NO   CURRENT
2146613 17-7 月 -12
```

说明一下 v$log 这个重要的视图

status 有四种状态：

unused:　　新添加的日志组，还没有使用
inactive:　日志组对应的脏块已经从 data buffer 写入到 data file ，可以覆盖
active:　　日志组对应的脏块还没有从 data buffer 写入到 data file，不能被覆盖
current:　 当前日志组，日志组对应的脏块还没有从 data buffer 写入到 data file，不能被覆盖

thread:　　线程（通过后台进程 lgwr 启动），在单实例的环境下，thread# 永远是 1
sequence:　日志序列号。在日志切换时会递增。
FIRST_CHANGE#:　在当前日志中记录的首个数据块的 scn。（当事务完成的时候会在数据块上写入一个 scn，代表数据块的变化）。

4.7、日志恢复(PPT-II-146-148)

例 1 inactive 日志组丢失

SQL> select * from v$log;

```
    GROUP#       THREAD#    SEQUENCE#          BYTES       MEMBERS ARCHIVED  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- -------- ----------------
------------- -----------
         1             1            59     52428800             2 NO          CURRENT
```

```
7108854 2013-3-25 1
         2            1          56     52428800          2 YES       INACTIVE
7087087 2013-3-25 1
         3            1          58     52428800          2 YES       INACTIVE
7108852 2013-3-25 1
         4            1          57     52428800          2 YES       INACTIVE
7108300 2013-3-25 1
```

注意：本例日志组 4 是 INACTIVE 状态的。

```
[oracle@timran timran]$ rm /u01/oradata/timran11g/redo04.log
[oracle@timran timran]$ rm /u01/disk2/timran/redo04b.log

SQL> alter database clear logfile group 4;       //注意:这一步使刚才在 os 里删掉的
那两个 group4 的文件又建立上了。
```

例 2 active 日志组丢失

注：本例日志组 3 状态是 ACTIVE 状态的。

```
[oracle@timran timran]$ rm /u01/oradata/timran11g/redo03.log
[oracle@timran timran]$ rm /u01/disk2/timran/redo03b.log

SQL> alter database clear logfile group 3;
alter database clear logfile group 3
*
第 1 行出现错误:
ORA-01624: 日志 3 是紧急恢复实例 timran11g (线程 1) 所必需的
ORA-00312: 联机日志 3 线程 1: '/u01/oradata/timran11g/redo03.log'
ORA-00312: 联机日志 3 线程 1: '/u01/disk2/timran/redo03b.log'

SQL> alter system checkpoint;

SQL> alter database clear logfile group 3;
```

例 3 current 日志组丢失。

注：本例日志组 1 状态是 ACTIVE 状态的。

```
[oracle@timran timran]$ rm /u01/oradata/timran11g/redo01.log
[oracle@timran timran]$ rm /u01/disk2/timran/redo01b.log
[oracle@timran timran]$
```

01:10:11 SQL> alter system switch logfile; 切换几此，触动它一下。

告警日志会记录有关信息

暂时好像没有什么问题发生，继续切换，当 current 又转会到 group1 时，死机！

当前日志损坏的问题比较复杂，可以分以下几种情况讨论（PPT-II-147）

1）如果数据库没有崩溃

第一步，可以做一个完全检查点，将 db buffer 中的所有 dirty buffer 全部刷新到磁盘上。

SQL> alter system checkpoint;

第二步，尝试数据库在打开状态下进行不做归档的强制清除。

SQL> alter database clear unarchived logfile group n;

数据库此时为打开状态，这步若能成功，一定要做一个新的数据库全备（考点）。因为当前日志没有归档，归档日志 sequence 已无法保持连续性。

2）如果数据库已经崩溃，准备做传统的基于日志的不完全恢复或闪回数据库。

SQL> recover database until cancel;
SQL> alter database open resetlogs;

3）如果严重到以上方法都不能 resetlogs 打开数据库，可以试试下面的最后一招:

修改 pfile 文件，第一行添加_allow_resetlogs_corruption=TRUE
该参数的含义是：允许在破坏一致性的情况下强制重置日志，打开数据库。_allow_resetlogs_corruption 将使用所有数据文件最旧的 SCN 打开数据库，所以通常来讲需要保证 SYSTEM 表空间拥有最旧的 SCN。在强制打开数据库之后，可能因为各种原因会有 ora-600 错误。

[oracle@work dbs]$ vi inittest11g.ora

_allow_resetlogs_corruption=TRUE
*.audit_file_dest='/u01/admin/timran11g/adump'
*.audit_trail='db'
*.compatible='11.1.0.0.0'
...

//再以 pfile 启动 instance 到 mount 状态，然后 alter database open resetlogs

//这是在不一致状态下强行打开了数据库，建议做一个逻辑全备。

4.8 使日志恢复到原来的配置

尝试使用 OEM 方式恢复原状。删除增加的 member 和 group4，注意当前日志组要切换后才能删除，最后，验证无误后删掉/u01/disk2 目录。

第五章： 归档日志 archivelog

5.1 归档和非归档的区别

1）归档会在日志切换时，备份历史日志，用于 OLTP，可以进行冷备份和热备份，可以实现数据库完全恢复、不完全恢复(基于时间点）

2）归档会启用 arcn 的后台进程、占用磁盘空间

3）非归档用于 OLAP/DSS，只能冷备份，只能恢复到最后一次备份状态

5.2 查看归档模式：

```
02:34:50 SQL>archive log list;
Database log mode              Archive Mode
Automatic archival             Enabled
Archive destination            /u01/disk1/timran/
Oldest online log sequence     1
Next log sequence to archive   2
Current log sequence           2
```

5.3 设置归档模式

```
02:35:50 SQL> shutdown immediate    //这里 shutdown 一定要 immediate 方式
Database closed.
Database dismounted.
ORACLE instance shut down.

02:36:40 SQL> startup mount     //到 mount 下设置

02:37:50 SQL>alter database noarchivelog;

Database altered.

02:37:55 SQL> archive log list;
Database log mode              No Archive Mode
Automatic archival             Disabled
Archive destination            /u01/disk1/timran/
Oldest online log sequence     1
Current log sequence           2
```

02:38:15 SQL> alter database archivelog;　　//换回来，不要 No Archive Mode

02:38:30 SQL> alter database open;

5.4、归档日志的路径及命名方法

指定归档日志存放处及命名，

如果 log_archive_dest_n 为空，归档日志文件目录为 log_archive_dest，如果 log_archive_dest 值也是空，则默认的是 db_recover_file_dest 参数指定的位置。缺省安装后 db_recover_file_des 指向了你的 flash_recover_area

注意：另外指定并不意味着你要取消闪回恢复区的参数，因为这个闪回恢复区不仅归档日志，还有 RMAN 的备份及闪回日志等等。

02:39:20 SQL> show parameter archive

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
-------
archive_lag_target                   integer     0
log_archive_config                   string
log_archive_dest                     string
log_archive_dest_1                      string       location=/u01/disk1/timran/
mandatory
log_archive_dest_10                  string
log_archive_dest_2                   string
log_archive_dest_3
...
log_archive_duplex_dest              string
log_archive_format                   string      arch_%t_%r_%s.log
...
```

SQL> show parameter db_recovery

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_recovery_file_dest                string      /u01/flash_recovery_area
db_recovery_file_dest_size           big integer 2G
SQL>
```

首先来看这两个参数：

log_archive_dest_n
log_archive_format

log_archive_dest_n （n:1-10） 表示可以有 10 个目标路径存放归档日志（镜像关系），即可以多路复用 10 个归档日志的备份。如上显示我只使用了 log_archive_dest_1，也就是说只有一套归档日志，没有做镜像，若是生产系统最好能再加一个指定。

SQL> alter system set log_archive_dest_1='location=/u01/disk1/timran';

//把历史日志归档到本机目录下 （location 代表本机）

SQL> alter system set log_archive_dest_2='service=standby';

//远程备份，把历史日志备份到服务名为 test 的另外的数据库上。(service 代表远程),配置双机时有用。

SQL> alter system set log_archive_dest_1='location=/u01/disk1/timran mandatory';
//mandatory 强制归档：只有在归档成功之后，重做日志才能被覆盖，在设置时至少应该有一个本地(location)
强制(mandatory)归档目录(考点)
//默认 optional, 即使归档没有成功也可以覆盖联机日志文件。

log_archive_format 是定义命名格式的,我使用了下面三个内置符号（模板），其含义是：

%t ，thread# ， 日志线程号
%s ，sequence ，日志序列号
%r ，resetlog ，代表数据库的周期

SQL> alter system set log_archive_format ='arch_%t_%r_%s.log' scope=spfile;

再来看看这两个参数：
log_archive_dest
log_archive_duplex_dest
如果都定义了可以完成两路复用（镜像）但只能指定本机 location，无法指定远程。

一旦使用 log_archive_dest_n，log_archive_dest 参数就失效了，log_archive_dest 与 log_archive_dest_n 互斥。

5.5 在 liunx 下查看归档进程

```
[oracle@timran timran]$ ps -ef |grep ora_arc
oracle    1215  2435  0 13:26 pts/2    00:00:00 grep ora_arc
oracle   31796     1  0 13:00 ?        00:00:00 ora_arc0_timran11g
```

oracle 31798 1 0 13:00 ? 00:00:00 ora_arc1_timran11g

ARCn 就是归档进程，这里启动了两个 arc0 和 arc1，最多可达 30 个，由 log_archive_max_processes 参数指定。

5.6 日志归档：

1）自动归档，日志切换时

2）手工

3）在归档时，会把归档信息写入到控制文件

02:44:00 SQL> alter system switch logfile; //手工归档方法一。

02:46:30 SQL> alter system archive log current; //手工归档方法二,此方式仅限于 Archive mode。

查看已经归档日志：

02:48:08 SQL> select name from v$archived_log;

NAME

/u01/disk1/timran/arch_1_782662700_141.log
/u01/disk1/timran/arch_1_782662700_142.log
/u01/disk1/timran/arch_1_782662700_143.log
/u01/disk1/timran/arch_1_782662700_144.log
/u01/disk1/timran/arch_1_782662700_145.log
/u01/disk1/timran/arch_1_788918717_1.log
/u01/disk1/timran/arch_1_788918717_2.log
/u01/disk1/timran/arch_1_788918717_3.log
05:47:10 SQL>

第六章 日志挖掘 log miner

6.1 log miner 的作用：

数据库恢复中有时会需要对 Redo log 进行分析， 要会使用 log miner，以便确定要恢复的时间点或 SCN

6.2 有两种日志挖掘方法 针对 DML 和 DDL，整理如下 ：

6.2.1 对语句 DML 进行日志挖掘：

1）添加 database 补充日志

SQL>ALTER DATABASE ADD SUPPLEMENTAL LOG DATA

//注意：通过 PL/SQL 包的 DML 的日志挖掘，这步要先执行，在此之后的 DML 操作才能从日志里挖到。在 OEM 中也是要求先做这一步，不同的是之前的 DML 操作是可以挖到的。

2）确定要分析的日志范围，添加日志，分析
SQL>execute dbms_logmnr.add_logfile(logfilename=>'日志',options=>dbms_logmnr.new); //第一个要加载的日志文件

SQL>execute dbms_logmnr.add_logfile(logfilename=>'补充日志',options=>dbms_logmnr.addfile); //可以反复添加补充多个日志文件

3）执行 logmnr 分析
SQL>execute
dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);

4）查询分析结果，可以设置时间格式，也可以在显示方式里再确定格式.

SQL>select username,scn,timestamp,sql_redo from v$logmnr_contents where seg_name='表名';

5）关闭日志分析
SQL>execute dbms_logmnr.end_logmnr;

例
sys:
11:33:20 SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;

scott:
09:44:08 SQL> create table a (id int);
09:44:20 SQL> insert into a values(1);
09:44:29 SQL> update a set id=5;
09:44:45 SQL> commit;
09:44:47 SQL> delete a;
09:44:51 SQL> commit;
Commit complete.

sys:

11:32:12 SQL> select * from v$log;

GROUP# THREAD# SEQUENCE# BYTES MEMBERS ARCHIVED STATUS FIRST_CHANGE# FIRST_TIME

---------- ---------- ---------- ---------- ---------- -------- ----------------
------------ -----------
         1   1   26   52428800   2 NO   CURRENT
2257870 2012-7-23 9
         2   1   25   52428800   2 YES   INACTIVE
2257866 2012-7-23 9
         3   1   23   52428800   2 YES   INACTIVE
2257862 2012-7-23 9
         4   1   24   52428800   2 YES   INACTIVE
2257864 2012-7-23 9

//上面 a 表的 DML 操作都写进了 current 组里，记住 sequence#是 26 号。然后手工切换当前日志进 archive 里。

11:32:18 SQL> alter system switch logfile;

11:33:00 SQL> /

11:33:02 SQL> /

11:33:02 SQL> /

11:33:09 SQL> select name from v$archived_log;

/u01/disk1/timran/arch_1_789252862_21.log
/u01/disk1/timran/arch_1_789252862_22.log
/u01/disk1/timran/arch_1_789252862_23.log
/u01/disk1/timran/arch_1_789252862_24.log
/u01/disk1/timran/arch_1_789252862_25.log
/u01/disk1/timran/arch_1_789252862_26.log
/u01/disk1/timran/arch_1_789252862_27.log
/u01/disk1/timran/arch_1_789252862_28.log
/u01/disk1/timran/arch_1_789252862_29.log

62 rows selected

//切换后，sequence#26 的日志的应该对应的是 /u01/disk1/timran/arch_1_789252862_26.log

11:33:48 SQL> execute dbms_logmnr.add_logfile(logfilename=>'/u01/disk1/timran/arch_1_789252862_26.log',options=>dbms_logmnr.new);

11:34:13 SQL> execute

```
dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);

11:34:49      SQL>select      scn,to_char(timestamp,'yyyy-mm-dd      hh24:mi:ss')
timestamp,sql_redo from v$logmnr_contents where seg_name='A';

       SCN TIMESTAMP           SQL_REDO
---------- ------------------- ------------------------------------------------
--------------------------------
   2258232 2012-07-23 09:43:16 drop table a purge;
   2258334 2012-07-23 09:44:20 create table a (id int);
   2258341 2012-07-23 09:44:28 insert into "SCOTT"."A"("ID") values ('1');
   2258349 2012-07-23 09:44:46 update "SCOTT"."A" set "ID" = '5' where "ID" = '1'
and ROWID = 'AAANBAAAEAAAAGEA
   2258353 2012-07-23 09:44:52 delete from "SCOTT"."A" where "ID" = '5' and ROWID
= 'AAANBAAAEAAAAGEAAA';
```

6.2.2 对 DDL 的操作的 log miner：

1）如果是第一次做，先要建好 logmnr 目录，

设置 logmnr 参数，存放数据字典文件 dict.ora

```
$ mkdir /home/oracle/logmnr

SQL> alter system set utl_file_dir='/home/oracle/logmnr' scope=spfile;
```

2）建立数据字典文件 dict.ora

```
SQL>                                                                    execute
dbms_logmnr_d.build('dict.ora','/home/oracle/logmnr',dbms_logmnr_d.store_in_flat
_file);
```

3）添加日志分析

```
SQL>    execute    dbms_logmnr.add_logfile(logfilename=>'  日  志  文  件
',options=>dbms_logmnr.new);
SQL>    execute    dbms_logmnr.add_logfile(logfilename=>'  追  加  日  志
',options=>dbms_logmnr.addfile);
```

4）执行分析

```
SQL>                                                                    execute
dbms_logmnr.start_logmnr(dictfilename=>'/home/oracle/logmnr/dict.ora',options=>d
bms_logmnr.ddl_dict_tracking);
```

5）查看分析结果

```
SQL> select username,scn,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss'),sql_redo from
```

v$logmnr_contents WHERE USERNAME ='SCOTT' and lower(sql_redo) like '%table%';

6）关闭日志分析
SQL> execute dbms_logmnr.end_logmnr;

例子将在 053 的不完全恢复一节中演示。

在 oracle11g 的 OEM 里已经加入了有关 log miner 的功能，可以根据时间段由 oracle 自动选择需要的日志，比较方便，但只有提交的事务的信息，看来是从 transaction 的角度设计的。

OEM-->Availability-->Manage-->View and Manage Transctions

第七章： 管理 undo(PPT-I-299-309)

7.1 undo 的作用

使用 undo tablespace 存放从 datafiles 读出的数据块的前镜像，提供以下四种情况所需要的信息

1）回滚事务：rollback
2）读一致性：正在做 DML 操作的数据块，事务结束前，其他用户读 undo 里面的数据前镜像
3）实例的恢复：instance recover(undo -------->rollback)
4）闪回技术 ：flashback query、flashback table 等

7.2 undo 的管理模式：

1）manaual 手工：roll segment（淘汰）
2）auto 自动：undo tablespace （init parameter ：undo_management = auto)

7.3 undo 表空间管理

1）可以建立多个 undo 表空间，但一个时刻只有一个处于 active
2）处于 active 状态的 undo tablespace 不能 offline 和 drop

01:08:31 SQL> select tablespace_name,status,contents from dba_tablespaces;

TABLESPACE_NAME	STATUS	CONTENTS
SYSTEM	ONLINE	PERMANENT
UNDOTBS1	ONLINE	UNDO
SYSAUX	ONLINE	PERMANENT

```
TEMP                           ONLINE    TEMPORARY
USERS                          ONLINE    PERMANENT
EXAMPLE                        ONLINE    PERMANENT
TEST                           ONLINE    PERMANENT

09:47:08 SQL> create undo tablespace undotbs2 datafile
'/u01/oradata/timran11g/undotbs02.dbf' size 100m autoextend on;
09:47:55 SQL> select tablespace_name,status,contents from dba_tablespaces;

SQL> select tablespace_name,status ,contents from dba_tablespaces;

TABLESPACE_NAME                STATUS    CONTENTS
------------------------------ --------- ---------
SYSTEM                         ONLINE    PERMANENT
SYSAUX                         ONLINE    PERMANENT
UNDOTBS1                       ONLINE    UNDO
TEMP                           ONLINE    TEMPORARY
USERS                          ONLINE    PERMANENT
UNDOTBS2                       ONLINE    UNDO
EXAMPLE                        ONLINE    PERMANENT
TEST                           ONLINE    PERMANENT
```

7.4 查看当前正在使用的 undo tablespace

```
09:48:00 SQL> show parameter undo
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_tablespace                      string      UNDOTBS1

00:20:50
SQL> select * from v$rollname;

       USN NAME
---------- ------------------------------
         0 SYSTEM
         1 _SYSSMU1_1363316212$
         2 _SYSSMU2_1363316212$
         3 _SYSSMU3_1363316212$
         4 _SYSSMU4_1363316212$
         5 _SYSSMU5_1363316212$
         6 _SYSSMU6_1363316212$
         7 _SYSSMU7_1363316212$
```

```
         8 _SYSSMU8_1363316212$
         9 _SYSSMU9_1363316212$
        10 _SYSSMU10_1363316212$
```

7.5 切换 undo

```
09:50:10 SQL> alter system set undo_tablespace=undotbs2;     //memory 动态修改
09:50:28 SQL> show parameter undo

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_tablespace                      string      UNDOTBS2

SQL> select * from v$rollname;

       USN NAME
---------- ------------------------------
         0 SYSTEM
        11 _SYSSMU11_1357956213$
        12 _SYSSMU12_1357956213$
        13 _SYSSMU13_1357956213$
        14 _SYSSMU14_1357956213$
        15 _SYSSMU15_1357956213$
        16 _SYSSMU16_1357956213$
        17 _SYSSMU17_1357956213$
        18 _SYSSMU18_1357956213$
        19 _SYSSMU19_1357956213$
        20 _SYSSMU20_1357956213$
```

7.6 删除 undo tablespace

```
SQL> drop tablespace undotbs1 including contents and datafiles;
SQL> select * from v$tablespace;

       TS# NAME                           INC BIG FLA ENC
---------- ------------------------------ --- --- --- ---
         0 SYSTEM                         YES NO  YES
         1 SYSAUX                         YES NO  YES
         4 USERS                          YES NO  YES
         6 EXAMPLE                        YES NO  YES
         8 TEST                           YES NO  YES
         3 TEMP                           NO  NO  YES
```

5 UNDOTBS2 YES NO YES

7.7 undo 数据的 4 种状态

1）active： 表示 transaction 还没有 commit，不可覆盖，
2)unexpired:已经 commit,但是还在 undo_retention 内,不可以覆盖(非强制),加 GUARANTEE 属性后强制 undo_retention 内不覆盖。
3）expired： 已经 commit，且时间超过了 undo_retention，随时可以覆盖。
4）free： 分配了但未使用过。

undo retention 参数和 undo autoextend on 特性

undo retention 参数规定了 unexpired commit 数据的保留期，它是保证一致性读，和大多数闪回技术成功的关键，
将 undo 表空间设为 autoextend on， 这是 DBCA 创建数据库时的缺省设置， 这一个特性将在 undo 空间不足时优先扩展新的空间,其次才是覆盖 unexpired commit。

考点：如果要减少 ORA_01555 错误（snapshot too old),考虑延长 undo retention 或使能 undo autoextend on。

7.8 关于 undo_retention 参数 //这个参数只对已 commit 的 undo 状态有效

01:10:46 SQL> select tablespace_name,status,contents,retention from dba_tablespaces;

```
TABLESPACE_NAME                STATUS    CONTENTS  RETENTION
------------------------------ --------- --------- -----------
SYSTEM                         ONLINE    PERMANENT NOT APPLY
SYSAUX                         ONLINE    PERMANENT NOT APPLY
TEMP                           ONLINE    TEMPORARY NOT APPLY
USERS                          ONLINE    PERMANENT NOT APPLY
EXAMPLE                        ONLINE    PERMANENT NOT APPLY
TEST                           ONLINE    PERMANENT NOT APPLY
UNDOTBS2                       ONLINE    UNDO      NOGUARANTEE
```

guarantee 属性随 undo 表空间建立， 可以修改

09:52:22 SQL> alter tablespace undotbs2 retention guarantee; // 保 证 在 retention 期间不允许被覆盖

01:11:16 SQL> select tablespace_name,status ,contents,retention from dba_tablespaces;

```
TABLESPACE_NAME                STATUS    CONTENTS  RETENTION
```

```
------------------------------ --------- --------- -----------
SYSTEM                         ONLINE    PERMANENT NOT APPLY
SYSAUX                         ONLINE    PERMANENT NOT APPLY
TEMP                           ONLINE    TEMPORARY NOT APPLY
USERS                          ONLINE    PERMANENT NOT APPLY
EXAMPLE                        ONLINE    PERMANENT NOT APPLY
TEST                           ONLINE    PERMANENT NOT APPLY
UNDOTBS2                       ONLINE    UNDO      GUARANTEE
```

缺省配置下 undo retention 是 noguarantee，

```
SQL> alter tablespace undotbs2 retention noguarantee;
```

7.9 undo 信息的查询

1） v$session 查看用户建立的 session
2） v$transaction 当前的事务
3） v$rollname undo 段的名称
4） v$rollstat undo 段的状态
5） dba_rollback_segs 数据字典里记录的 undo 段状态

一般来说，一个 session 只能对应一个事务，建立了 session 未必有事务，只有事务处于活动状态时，v$transaction 才能看到这个事务。换句话说，如果看到了事务(在 v$transaction 里)，那一定有个 session 和它对应，将两个视图里连在一起，信息看得更为清楚。

cmd 下 scott 登录：

```
10:03:28 SQL> select username,sid,serial# from v$session where username is not
null;    //看看有几个 session 连上来了。
USERNAME                         SID    SERIAL#
------------------------------ ---------- ----------
SCOTT                             131         18
SYS                               170          5
```

```
cmd 下 update emp1 set sal=1000 where empno=7788;       // 产生一个活动事务
```

```
SQL>  select  a.sid,a.serial#,a.username,b.xidusn,xidslot,b.ubablk,b.status  from
v$session a,v$transaction b where a.saddr=b.ses_addr;
```

```
10:08:29 SQL> col name for a10
     SID      SERIAL#  USERNAME                    XIDUSN     XIDSLOT      UBABLK
STATUS
---------- ---------- -------------------- ---------- ---------- ---------- ----
```

```
------------

       131          18 SCOTT                          18          5          305
ACTIVE
```

//两表联查，sid 和 serial#在 v$session 里，在 v$transaction 里 XIDUSN 是 undo segment 的 id ， XIDSLOT 是事务槽的 id,UBABLK 是 undo 块号

对照上面看，下面语句显示出_SYSSMU18 是一个活动段， 与 XIDUSN=8 吻合， 说明这个段被读进 buffer 了.

```
SQL> select a.usn,b.name,a.xacts from v$rollstat a, v$rollname b where a.usn=b.usn;

       USN NAME                 XACTS
---------- ------------------------   --------
         0 SYSTEM                    0
        11 _SYSSMU11_1357956213$           0
        12 _SYSSMU12_1357956213$                0
        13 _SYSSMU13_1357956213$                0
        14 _SYSSMU14_1357956213$                0
        15 _SYSSMU15_1357956213$                0
        16 _SYSSMU16_1357956213$                0
        17 _SYSSMU17_1357956213$                0
        18 _SYSSMU18_1357956213$                1
        19 _SYSSMU19_1357956213$                0
        20 _SYSSMU20_1357956213$                0
```

7.10 system 表空间的 undo

默认 system 表空间会有一个单独的 undo segment(usn 为 0)。其他表空间的事务不能使用它。缺省下 undo tablespace 会被分配 10 个 undo segment，理论上一个段可以有多个事务，但 Oracle 的策略并不情愿如此，实验表明，对应并发的事务，通常每个事务会分配一个 UNDO 段，也就是说如果这 10 个段不够，Oracle 会自动再增加段。

当数据库启动时，如果碰到 undo 数据文件损坏或不同步，可以将其 offline，即只有一个系统表空间的 undo 也能进入系统，使数据字典得以维护，但不能对其他表空间做 DML 操作。

7.11 测试：模拟数据库 open 下的 undo 损坏和修复。

如果数据库打开的情况下，当前 undo 坏掉的话，比如现在 UNDOTBS2 出现了介质损坏，那么数据库就不能继续 DML 操作了。如果这时 UNDOTBS2 表空间上还有 active 状态的事务（未提

交),Oracle 会将其下的所有段都标志为 NEEDS RECOVERY,这时有备份可以恢复这个 UNDOTBS2,但如果没有备份，你需要用新建的 UNDO 替代损坏的 UNDO，那么损坏 UNDO 上的未提交事务也将不得不丢弃。但 Oracle 没有提供自动解决 NEEDS RECOVERY 的机制，如果你想删除这个损坏的 UNDO 表空间，必须另做处理（需要系统择时重启），我们模拟一下这种情况：

```
[oracle@timran timran]$ mv undotbs02.dbf undotbs02.bak

00:29:28 SQL>alter system checkpoint;
```

cmd 端再做 update 语句时会报错

```
SQL> update emp1 set sal=1000 where empno=7902;
ORA-01116: 打开数据库文件 3 时出错
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/undotbs02.dbf'
ORA-27041: 无法打开文件

00:29:28 SQL> startup force mount

SQL> select file#,checkpoint_change# from v$datafile;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            6708724
         2            6708724
         3            6708724
         4            6708724
         5            6708724
         6            6708724

SQL> select file#,checkpoint_change# from v$datafile_header;

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            6708724
         2            6708724
         3                  0
         4            6708724
         5            6708724
         6            6708724

SQL> alter database datafile 3 offline;
SQL> alter database open;
SQL> select * from v$rollname;
```

```
       USN NAME
---------- --------------------------------
         0 SYSTEM
```

SQL> select segment_name,status from dba_rollback_segs; //这个静态视图可以列出所有（online 或 offline）UNDO 段信息

```
SEGMENT_NAME                   STATUS
------------------------------ ----------------
SYSTEM                         ONLINE
_SYSSMU20_1357956213$          NEEDS RECOVERY
_SYSSMU19_1357956213$          NEEDS RECOVERY
_SYSSMU18_1357956213$          NEEDS RECOVERY
_SYSSMU17_1357956213$          NEEDS RECOVERY
_SYSSMU16_1357956213$          NEEDS RECOVERY
_SYSSMU15_1357956213$          NEEDS RECOVERY
_SYSSMU14_1357956213$          NEEDS RECOVERY
_SYSSMU13_1357956213$          NEEDS RECOVERY
_SYSSMU12_1357956213$          NEEDS RECOVERY
_SYSSMU11_1357956213$          NEEDS RECOVERY
```

SQL>create undo tablespace undotbs1 datafile '/u01/oradata/timran11g/undotbs01.dbf' size 100m autoextend on;

SQL>select * from v$tablespace;

```
       TS# NAME                           INC BIG FLA ENC
---------- ------------------------------ --- --- --- ---
         0 SYSTEM                         YES NO  YES
         1 SYSAUX                         YES NO  YES
         2 UNDOTBS1                       YES NO  YES
         4 USERS                          YES NO  YES
         3 TEMP                           NO  NO  YES
         6 EXAMPLE                        YES NO  YES
         8 TEST                           YES NO  YES
         5 UNDOTBS2                       YES NO  YES
```

SQL> alter system set undo_tablespace=UNDOTBS1;

注意：此时原有活动事务的信息（未提交)可能仍然保存在 UNDOTBS2, 已经没有恢复的必要了。

```
SQL> drop tablespace undotbs2 including contents and datafiles;
drop tablespace undotbs2 including contents and datafiles
*
第 1 行出现错误:
ORA-01548: 已找到活动回退段 'SYSSMU11_1357956213$', 终止删除表空间
```

//这个 UNDOTBS2 删不掉是因为 Oracle 把其下的所有段都标志成 NEEDS RECOVERY 了，再查看一下相关的数据字典：

```
SQL> select segment_name,status from dba_rollback_segs;
```

解决有两个办法：1）使用备份恢复，2）使用 oracle 提供的隐含参数 _CORRUPTED_ROLLBACK_SEGMENTS
我们假设没有备份，就使用第 2 种方法

```
SQL>create pfile from spfile;            //建立静态参数文件
SQL>shutdown abort

#vi /u01/oracle/dbs/inittimran11g.ora        //在静态参数文件里第一行插入以下内容
_CORRUPTED_ROLLBACK_SEGMENTS=(_SYSSMU11_1357956213$,_SYSSMU12_1357956213$,_SYSSMU13_1357956213$,_SYSSMU14_1357956213$,_SYSSMU15_1357956213$,_SYSSMU16_1357956213$,_SYSSMU17_1357956213$,_SYSSMU18_1357956213$,_SYSSMU19_1357956213$,_SYSSMU20_1357956213$)
```

然后存盘,再使静态参数文件启动数据库

```
SQL> startup pfile='/u01/oracle/dbs/inittimran11g.ora'

SQL> drop rollback segment "_SYSSMU11_1357956213$";

回退段已删除。
......
SQL> drop rollback segment "_SYSSMU20_1357956213$";

SQL> select segment_name,status from dba_rollback_segs;

SEGMENT_NAME                   STATUS
------------------------------ ----------------
SYSTEM                         ONLINE
_SYSSMU10_1384520126$          ONLINE
_SYSSMU9_1384520126$           ONLINE
_SYSSMU8_1384520126$           ONLINE
_SYSSMU7_1384520126$           ONLINE
_SYSSMU6_1384520126$           ONLINE
```

```
_SYSSMU5_1384520126$            ONLINE
_SYSSMU4_1384520126$            ONLINE
_SYSSMU3_1384520126$            ONLINE
_SYSSMU2_1384520126$            ONLINE
_SYSSMU1_1384520125$            ONLINE

已选择 11 行。

SQL> drop tablespace undotbs2 including contents and datafiles;

表空间已删除。
```

如果还删除不了，则尝试修改字典

select * from v$tablespace; 记下要删除的 undo 对应的 ts#号，比如是=2，则执行下行语句：
update seg$ set type# = 3 where ts#=2;

然后再删除该表空间即可。

第八章 检查点（checkpoint）

8.1 什么是 checkpoint

checkpoint 是数据库的一个内部事件，检查点激活时会触发数据库写进程(DBWR)，将数据缓冲区里的脏数据块写到数据文件中。

8.2 checkpoint 主要 2 个作用：

1）保证数据库的一致性，这是指将脏数据写出到硬盘，保证内存和硬盘上的数据是一样的；

2）缩短实例恢复的时间，实例恢复要把实例异常关闭前没有写到硬盘的脏数据通过日志进行恢复。如果脏块过多，实例恢复的时间也会很长，检查点的发生可以减少脏块的数量，从而提高实例恢复的时间。

8.3 checkpoint 分类

完全检查点 full checkpoint
增量检查点 incremental checkpoint
局部检查点 partial checkpoint

8.3.1 完全检查点工作方式：记下当前的 scn, 将此 scn 之前所有的脏块一次性写完，再将该 scn 号同步更新控制文件和数据文件头。

可以引起完全检查点的两个动作

a)正常关闭数据库:shutdown immediate
b)手动检查点切换:alter system checkpoint;
c)日志切换：alter system switch logfile;
d)数据库热备模式：alter database begin backup;

验证以上概念可以做一下 alter system checkpoint，然后观察 v$datafile 和 v$datafile_header 中 scn 被更新。
另外，我们可以研究一下日志切换:alter system switch logfile;

如果 FAST_START_MTTR_TARGET<>0,v$log 视图中的 active 状态几分钟后会变成 inactive 状态，然后更新了控制文件和日志文件头部的 SCN。

8.3.2 增量检查点概念及相关参数：

8.3.2.1 概念：

1）被修改过的块，在 oracle 中都被统称为脏块.脏块按照首次变脏的时间顺序被一个双向链表指针串联起来,这称做检查点队列。

2)当增量检查点发生时,DBWR 就会被触发,沿着检查点队列的顺序将部分脏块刷新到磁盘上，每次刷新截止的那个块的位置就叫检查点位置，检查点位置之前的块,都是已经刷新到磁盘上的块。而检查点位置对应的日志地址（RBA）又总是被记录在控制文件中。如果发生系统崩溃，这个最后的检查点位置就是实例恢复的起点。

3）增量检查点使检查点位置前移。进而缩短实例恢复需要的日志起点和终点之间的距离，触发增量检查点越频繁，实例恢复的时间越少，但数据库性能受到频繁 IO 影响会降低。

4）增量检查点不会同步更新数据文件头和控制文件的 SCN。

8.3.2.2 与增量检查点有关的几个知识点

1）FAST_START_MTTR_TARGET 参数：

这个参数是考点。它给出了你希望多长时间恢复实例。

此参数单位为秒，缺省值 0，范围 0-3600 秒，根据这个参数，Oracle 计算出在内存中累积的 dirty buffer 所需的日志恢复时间,如果到达该参数指定的时间,则增量检查点进程被触发。该参数如果为 0，ORACLE 则会根据 DBWN 进程自身需要尽量减少写入量，这样虽然实现了性能最大化，但实例恢复时间可能会比较长。

早期还有几个有关增量检查点的参数，

log_checkpoint_interval

规定了 redo 日志积累多少 block 后激活增量检查点，对用户来讲要给出这个参数不太方便，所谓 block 指的是 os block，而不是 oracle block。

log_checkpoint_timeout 给一个触发增量检查点的间隔，单位是秒。

如果设置了 FAST_START_MTTR_TARGET 参数，就不要用早期的一些参数，容易引起冲突。

*考点：

如果将 fast_start_mttr_target 设置为非 0，1）将启用检查点自动调整机制。2）log_checkpoint_interval 参数将被覆盖掉。

2）90% OF SMALLEST REDO LOG(Oarcle 内部机制)，将当前日志文件末尾还剩 10%的位置设为检查点位置。

3）每 3s 查看 checkpoint 队列脏块的写出进度，注意，3s 不触发检查点，它只是记录当时的检查点位置，并将相关信息写入到 controlfile。

8.3.3 实例恢复有关的顾问叫做 MTTR Advisory 需要设置两个参数

1）STATISTICS_LEVEL -->置为 typical（缺省）或者 all
2）FAST_START_MTTR_TARGET -->置为非零值

8.3.4 设置合理的 MTTR 参数

查看视图 v$instance_recovery

```
SQL>select
recovery_estimated_ios,actual_redo_blks,target_redo_blks,target_mttr,estimated_m
ttr from v$instance_recovery;

RECOVERY_ESTIMATED_IOS      ACTUAL_REDO_BLKS      TARGET_REDO_BLKS      TARGET_MTTR
ESTIMATED_MTTR
----------------------- ---------------- ---------------- ------------ -----------
---
                    72                   333                  3700                   33
12
```

//系统计算出来的 target_mttr 为 33 秒,当前估算的恢复时间是 12 秒。

8.5 局部检查点的触发条件：

对于某些操作，局部检查点是必须的，并会自动执行。
比如：表空间 offline,数据文件 offline,删除 extent,表 truncate,begin backup(将表空间置于备份模式）等。Oracle 会根据需要自动执行。

第九章 实例恢复机制

有了 redo,undo 和 ckpt 概念后，有助于更好的理解实例恢复(PPT-I：404-408）

实例崩溃时，内存中的 db buffer 和磁盘上的 datafile 不一致。

实例恢复要解决的两个问题是：

重新构成崩溃时内存中未被保存到磁盘的已 commit 事务。
回滚已被写至数据文件的 uncommit 事务。

由于引入了增量检查点，当实例恢复时，oracle 首先从控制文件里找到最后一次检查点位置，这个位置就是实例恢复时运用日志的起点。

然后是 smon 监控下的一系列动作：

1） roll forward ：利用 redo，将检查点位置之后的变更，包括 commit 和 uncommit 的都前滚出来了，然后统统写到磁盘(datafile)里。
2） open： 用户可以连接进来，访问数据库。
3） roll back ： 通过 undo，把写入磁盘里（datafile)的 uncommit 的数据回滚掉。

第二部分 Oracle 的存储架构

第十章 数据字典表和动态性能视图

Oracle 提供了大量的内部表，它们记录了数据库对象的更改和修正。可以将这些内部表划分为两种主要类型：静态的数据字典表和动态的性能表。这些内部表是由 oracle 维护的，可以说它们是只读表。用户包括 sys 都不能修改，只能查看。

10.1 数据字典

10.1.1 数据字典的功能（Data dictionary)

1) central of database
2) read_only table and views
3) owner ： sys
4) oracle server 维护，ddl 操作会更新
5) 通过 select 访问

6）记录数据库的物理、逻辑结构（表空间）和模式信息及对象信息（用户、约束、权限、审计等）

7）存放在 system tablespace

通过 dict 记录了所有数据字典表的名称

```
SQL> select * from dict where table_name='DBA_OBJECTS';

TABLE_NAME                     COMMENTS
------------------------------------------   ------------------------------
------------------------------
DBA_OBJECTS                    All objects in the database

SQL> select count(*) from dict;

  COUNT(*)
----------
      2323
```

10.1.2 数据字典的组成：

字典基表：由 create database 时，执行/u01/oracle/rdbms/admin/sql.bsd 脚本。
静态视图：执行/u01/oracle/rdbms/admin/catalog.sql

10.1.3 数据字典：静态（static）视图

static ：在数据库 open 状态下访问，可以通过静态视图了解 database 的架构(记录 database 的架构，object 的数据定义和存储等信息）

dba_ ：存储所有用户对象的信息（默认只能有 sys/system 用户访问）
all_ ： 存储当前用户能够访问的对象（包括用户所拥有的对象和别的用户授权访问的对象）的信息。
user_：存储当前用户所拥有的对象的相关信息。

考点:实例参数 O7_DICTIONARY_ACCESSIBILITY 在缺省时为 FALSE,这将限制具有 select any table 权限的非 DBA 用户访问 sys 用户下的对象，否则这些非 DBA 用户也能查看 dba_。

10.2 动态性能表（V$）

是维护当前实例信息的，由于不断的更新，所以也叫动态视图。其底层是一组虚拟的动态表称为 X$表，oracle 不允许直接访问 X$表，而是在这些表上创建视图，然后再创建这些视图的同义词。

基表（x$)----------视图（v_$)----------同义词 v$-------User access

可以通过 v$fixed_table 视图 查到所有的动态视图的名称；用于调优和数据库监控。从 Oracle8 开始， GV$视图开始被引入，其含义为 Global V$,GV$的产生是为了满足 OPS 环境的需要，除了一些特例以外，每个 V$视图都有一个 GV$视图存在。

```
SQL> select count(*) from v$fixed_table;

  COUNT(*)
----------
      1741
```

*考点：动态性能视图填充了来自实例和控制文件的信息，前缀为 DBA_、ALL_、USER_的视图则填充了来自数据字典的信息，此差异决定了可以 在不同启动阶段查询那些视图。

第十一章： Oracle 的存储架构

DATABASE-->TABLESPACES-->SEGMENTS-->EXENTS-->BLOCKS (DBA-I-PPT36-)

11.1 TABLESPACE(表空间）分类

PERMANENT 永久表空间
UNDO 撤销表空间
TEMPORARY 临时表空间

11.1.1 表空间的管理方式：

重点是段的管理方式和区的管理方式是在建立表空间时确定的。

段管理方式有 AUTO 和 MANUAL 两种，区管理方式有本地管理和字典管理（已淘汰）两种。

```
03:32:36 SQL> select tablespace_name,contents ,extent_management,segment_space_management from dba_tablespaces;

TABLESPACE_NAME                CONTENTS  EXTENT_MAN SEGMEN
------------------------------ --------- ---------- ------
SYSTEM                         PERMANENT DICTIONARY MANUAL
SYSAUX                         PERMANENT LOCAL      AUTO
TEMP                           TEMPORARY LOCAL      MANUAL
USERS                          PERMANENT LOCAL      AUTO
EXAMPLE                        PERMANENT LOCAL      AUTO
```

```
UNDO_TBS01                     UNDO      LOCAL      MANUAL
TMP01                          TEMPORARY LOCAL      MANUAL
TBS_16K                        PERMANENT LOCAL      AUTO
BIG_TBS                        PERMANENT LOCAL      AUTO
TEST                           PERMANENT DICTIONARY MANUAL
```

注意两点：
1）如果 system 表空间是数据字典管理，其他表空间可以是数据字典管理或 local 管理（默认）
2)字典管理可以转换成本地管理,但是对于系统表空间,要求执行一些附加步骤,比较麻烦。

```
execute dbms_space_admin.tablespace_migragte_to_local('tablespacename');
```

11.1.2 表和表空间的关系

建一个使用缺省值的表空间

```
SQL> create tablespace a datafile '/u01/oradata/timran11g/a01.dbf' size 10m;
```

利用 oracle 提供的 dbms_metadata.get_ddl 包看看缺省值都给的是什么？

```
SQL> set serverout on;
SQL>
declare
aa  varchar2(2000);
begin
select dbms_metadata.get_ddl('TABLESPACE','B') into aa FROM dual;
dbms_output.put_line(aa);
end;
/
```

结果：

```
  CREATE TABLESPACE "A" DATAFILE
  '/u01/oradata/timran11g/a01.dbf' SIZE 10485760
  LOGGING ONLINE PERMANENT BLOCKSIZE
8192
  EXTENT MANAGEMENT LOCAL AUTOALLOCATE SEGMENT SPACE MANAGEMENT AUTO
PL/SQL 过程已成功完成。
```

关注最后一行，两个重要信息是:(1)区本地管理且自动分配空间，(2)段自动管理。

```
SQL>
create tablespace b datafile '/u01/oradata/timran11g/b01.dbf' size 10m
extent management local uniform size 128k
```

segment space management manual

同上，调 dbms_metadata.get_ddl 包看 oracle 对该语句的 ddl 操作是：

```
  CREATE TABLESPACE "B" DATAFILE
  '/u01/oradata/timran11g/a01.dbf' SIZE 10485760
  LOGGING ONLINE PERMANENT BLOCKSIZE
8192
  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 131072 SEGMENT SPACE MANAGEMENT MANUAL
```

最后一行信息是：区本地管理且统一分配 128K， 段手动管理。如果在建表时使用缺省说明，则该表将服从其表空间的这些定义，

11.1.3 删除表空间

数据库 OPEN 下不能删除的表空间是
1）system 2）active undo tablespace 3） default temporary tablespace 4）default tablespace

数据库 OPEN 下不能 offine 的表空间是
1）system 2）active undo tablespace 3） default temporary tablespace

查看表空间空闲大小

```
09:47:04 SQL> select TABLESPACE_NAME,sum(bytes)/1024/1024 from dba_free_space group by tablespace_name;

TABLESPACE_NAME                SUM(BYTES)/1024/1024
------------------------------ --------------------
UNDOTBS1                                    98.4375
SYSAUX                                       14.625
USERS                                       48.1875
SYSTEM                                        1.875
EXAMPLE                                       31.25
```

11.1.4 大文件（bigfile）表空间（默认 small file）

1）small file，在一个表空间可以建立多个数据文件
2）bigfile ：在一个表空间只能建立一个数据文件 （8k 的 block ，datafile maxsize 可以 32T），可以简化对数据文件管理

```
09:54:49 SQL> create bigfile tablespace big_tbs datafile
```

’/u01/oradata/timran11g/bigtbs01.dbf’ size 100m;

试图在该表空间下增加一个数据文件会报错

```
09:55:01 SQL> alter tablespace big_tbs add datafile
’/u01/oradata/timran11g/bigtbs02.dbf’ size 100m;
alter tablespace big_tbs add datafile ’/u01/oradata/timran11g/bigtbs02.dbf’ size
100m
*
ERROR at line 1:
ORA-32771: cannot add file to bigfile tablespace
```

查看大文件表空间：

```
09:55:46 SQL> select name,bigfile from v$tablespace;

NAME                           BIG
------------------------------ ---
SYSTEM                         NO
UNDOTBS1                       NO
SYSAUX                         NO
USERS                          NO
TEMP                           NO
EXAMPLE                        NO
TBS_16K                        NO
BIG_TBS                        YES
```

11.2 SEGMENT(段）

11.2.1 SEGMENT（段）的特点：

1）表空间在逻辑上可以对应多个段，物理上可以对应多个数据文件，一个段比较大时可以跨多个数据文件。
2）创建一个表，ORACLE 为表创建一个（或多个）段，在一个段中保存该表的所有表数据(表数据不能跨段)。
3）段中至少有一个初始区。当这个段数据增加使区（extent）不够时，将为这个段分配新的区。

段管理有两种方式：

1）自动管理方式(ASSM(Auto Segment Space Management)) --采用位图管理段的存储空间

简单说就是每个段的段头都有一组位图（5 个位图），位图描述每个块的满度，根据满度的不同将每个块登记到相应的位图上，位图自动跟踪每个块的使用空间，这 5 个位图的满度

按如下定义：满度 100%，75%、50%、25%和 0%，比如块大小为 8k，你要插入一行是 3k 的表行，那么 oracle 就给你在满度 50%的位图上找个登记的块。

ASSM 的前提是 EXTENT MANAGEMENT LOCAL，在 ORACLE9I 以后，缺省状态为自动管理方式，ASSM 废弃 pctused 属性。

2）手工管理方式(MSSM(Manual Segment Space Management)) --采用 FREELIST(空闲列表）管理段的存储空间

这是传统的方法，现在仍然在使用，未被淘汰，保留 pctfree 和 pctused 属性，这些概念后面介绍 block 时再讨论。

11.2.2 表和段（segment)的关系

一般来讲 一个单纯的表就分配一个段，但往往表没那么单纯，比如表上经常会有主键约束，那么就会有索引，索引有索引段，还有分区表，每个分区会有独立的段，再有就是 oracle 的大对象, 如果你的表里引用 blob,clob,那么这个表就又被分出多个段来。

```
SQL> conn / as sysdba
SQL> create user tim identified by tim;
SQL> grant connect,resource to tim;

SQL> conn tim/tim
SQL> select * from user_segments;

未选定行

SQL> create table t1 (id int);

SQL> select segment_name from user_segments;

SEGMENT_NAME
--------------------------------------------------------------------------------
-
T1

SQL> create table t2 (id int constraint pk_t2 primary key, b blob, c clob);

SQL> select segment_name from user_segments;

SEGMENT_NAME
SEGMENT_TYPE
--------------------------------------------------------------------------------
- ------------------
```

PK_T2
INDEX
SYS_IL0000071160C00003$$
 LOBINDEX
SYS_LOB0000071160C00003$$
LOBSEGMENT
SYS_IL0000071160C00002$$
LOBINDEX
SYS_LOB0000071160C00002$$
 LOBSEGMENT
T2
TABLE
T1
TABLE

注：Oracle11gR2 又增加了一个新的初始化参数 DEFERRED_SEGMENT_CREATION(仅适用未分区的 heap table)，此参数设为 TRUE 后，create table 后并不马上分配 segment，当第一个 insert 语句后才开始分配 segment。这对于应用程序的部署可能有些好处。(PPT-II-476-478)

也可以使局部设置改变这一功能(覆盖 DEFERRED_SEGMENT_CREATION)，在 create table 语句时加上 SEGMENT CREATION 子句指定。如：

```
create table scott.t1(id int,name char(10) SEGMENT CREATION IMMEDIATE TABLESPACE TB1
或
create table scott.t1(id int,name char(10) SEGMENT CREATION DEFERRED;        //缺省在 11gR2
```

11.3 EXTENT（区）

11.3.1 EXTENT（区）的特点：

区是 ORACLE 进行存储空间分配的最小单位。是由一系列逻辑上连续的 Oracle 数据块组成的逻辑存储结构。段中第一个区叫初始区，随后分配的区叫后续区。当段中所有的空间使用完后，ORACLE 自动为该段分配新的区。

11.3.2 区的管理方式：

1)字典管理:在数据字典中管理表空间的区空间分配。Oracle 8i 以前只有通过 uet$和 fet$的字典管理。
缺点：某些在字典管理方式下的存储分配有时会产生递归操作，并且容易产生碎片，从而影响了系统的性能，现在已经淘汰了。

2）本地管理：在每个数据文件中使用位图管理空间的分配。表空间中所有区（extent)的分配信息都保存在该表空间对应的数据文件的头部。
优点：速度快，存储空间的分配和回收只是简单地改变数据文件中的位图，而不像字典管理方式还需要修改数据库。无碎片，更易于 DBA 维护。

11.3.3 表和区（extent）的关系：

当建立表的时候建立段，然后自动分配相应的 extent（1 个或者多个），亦可以手工提前分配 extent（用于需大量插入数据的表）

11.3.4 实验：查看段的初始区分配情况

```
sys:
SQL> create tablespace test datafile '/u01/oradata/timran11g/test01.dbf' size 10m;
SQL> create table scott.t1 tablespace test as select * from scott.dept;
SQL> col segment_name for a20;
SQL> select segment_name,file_id,extent_id,bytes from dba_extents where segment_name='T1';

SEGMENT_NAME            FILE_ID  EXTENT_ID      BYTES
-------------------- ---------- ---------- ----------
T1                             6          0      65536
```

可以看到段 T1 的初始区 ID 为 0，大小为 65536 bytes;

向表段中自插表数据，看 Oracle 为该段分配更多的区

```
SQL> insert into scott.t1 select * from scott.t1;
已创建 2048 行。

SQL> select segment_name,file_id,extent_id,bytes from dba_extents where segment_name='T1';

SEGMENT_NAME            FILE_ID  EXTENT_ID      BYTES
-------------------- ---------- ---------- ----------
T1                             6          0      65536
T1                             6          1      65536
T1                             6          2      65536
```

此时看到随着数据的插入，T1 段动态扩展为三个区;

```
SQL> delete scott.t1;
```

已删除 4096 行。

SQL> select segment_name,file_id,extent_id,bytes from dba_extents where segment_name='T1';

此时表段的数据已经删除，但所有 extent 依然健在，无法回收 T1 段的所有区,

可以要求一个预分配所需要的空间（但要注意，所要的空间 一定是在表空间可达到的 size 范围内）：

alter table scott.t1 allocate extent (datafile '/u01/oradata/timran11g/test01.dbf' size 5m);

SQL> select segment_name,extent_id,file_id,bytes from dba_extents where segment_name='T1';

SEGMENT_NAME	EXTENT_ID	FILE_ID	BYTES
T1	0	6	65536
T1	1	6	65536
T1	2	6	65536
T1	3	6	1048576
T1	4	6	1048576
T1	5	6	1048576
T1	6	6	1048576
T1	7	6	1048576

回收 free extent, 使用 deallocate, 注意：只能收回从未使用的 extent。

SQL> alter table scott.t1 deallocate unused;

表已更改。

SQL> select segment_name,extent_id,file_id,bytes from dba_extents where segment_name='T1';

SEGMENT_NAME	EXTENT_ID	FILE_ID	BYTES
T1	0	6	65536
T1	1	6	65536
T1	2	6	65536

顺便提一句：如何查看一个表所对应的数据文件及表空间呢？

抓住上面 dba_extents 中的 file_id 字段（user_extents 里没有这个字段）,然后；

SQL> select file_id,file_name,tablespace_name from dba_data_files;

11.4 BLOCK(数据块)

11.4.1 BLOCK(数据块）的特点：

BLOCK 是 Oracle 进行存储空间 IO 操作的最小单位，BLOCK 的管理方法是区的管理和段管理的具体体现：

1、自动管理方式　如创建表空间时区为本地管理方式，并且将段的存储空间方式设置为 AUTO（即 ASSM），该表空间的所有块均采用位图自动管理方式。这是系统默认的。

2、空闲列表方式(MSSM)　引入 FREELIST 概念，以及 PCTFREE 和 PCTUSED 两个参数控制可用存储区的大小，避免行迁移现象的发生。这两个参数可在创建表空间时设置，也可在建立数据库的模式对象（表，索引）中设置。模式对象中设置的优先级比表空间的要高。就是说；如表和索引中没有设置，则按表空间的设置，如表空间也没设置，则按自动管理方式管理块。

data block ：oracle 11g 标准块：8k，支持 2-32k，有 block header 、free space 、data 组成

数据块头部：

ITL：事务槽，可以有多个 ITL 以支持并发事务，每当一个事务要更新数据块里的数据时，必须先得到一个 ITL 槽，然后将当前事务 ID，事务所用的 undo 数据块地址，SCN 号，当前事务是否提交等信息写到 ITL 槽里。
initrans ：初始化事务槽的个数，表默认 1, index 默认为 2;
maxtrans:　最大的事务槽个数 （默认 255）

ROW DIR: 行目录, 指向空闲行起始和结束的偏移量。

考点：使块头增加的可能情况是，row entries 增加，增加更多的 ITL 空间。

空闲列表方式的数据块的管理：

freelist：空闲列表中登记了可以插入数据的可用块，位置在段头，插入表行数据时首先查找该列表。

pctfree：用来为一个块保留的空间百分比，以防止在今后的更新操作中增加一列或多列值的长度。达到该值，从 freelist 清除该块信息。

pctused：一个块的使用水位的百分比，这个水位将使该块返回到可用列表中去等待更多的

插入操作。达到该值，该块信息加入 freelist
。这个参数在 ASSM 下不使用。

行链接：指一行存储在多个块中的情况，这是因为该行的长度超过了一个块的可用空间大小，即行链接是跨越多块的行。

行迁移：指一个数据行由于 update 语句导致当前块被重新定位到另一个块（那里有充足的空间）中，但在原始块中保留一个指针的情形(PPT -II-470)。原始块中的指针是必需的，因为索引的 ROWID 项仍然指向原始位置。行迁移是 update 语句当 pctfree 空间不足时引起的，它 与 insert 和 delete 语句无关（考点）。

如何能够知道发生了行链接或行迁移？

查看 dba_tables 的 AVG_ROW_LEN 列和 CHAIN_CNT 列，当 CHAIN_CNT 有值时，看 AVG_ROW_LEN，它表示行的平均长度（byte），如果 AVG_ROW_LEN<块大小，那么行是迁移行，如果>块大小，那么是链接行。

```
SQL> create table t1 (c1 varchar2(20));

SQL>
begin
for i in 1..1000 loop
insert into t1 values(null);
end loop;
end;
/
```

分析 t1 表确定无行迁移

```
SQL> analyze table t1 compute statistics;
SQL> select pct_free,pct_used,avg_row_len,chain_cnt from user_tables where table_name='T1';

  PCT_FREE   PCT_USED AVG_ROW_LEN  CHAIN_CNT
---------- ---------- ----------- ----------
        10                     3          0
```

填充这些空列，再分析 t1,有了行迁移

```
SQL> update t1 set c1='timran is my name';
SQL> analyze table t1 compute statistics;
SQL> select pct_free,pct_used,avg_row_len,chain_cnt from user_tables where table_name='T1';
```

PCT_FREE	PCT_USED	AVG_ROW_LEN	CHAIN_CNT
10		26	865

move 表，再分析 t1,行迁移消失。 思考：段重组对于行链接有效吗？

```
SQL> alter table t1 move;
SQL> analyze table t1 compute statistics;
SQL> select pct_free,pct_used,avg_row_len,chain_cnt from user_tables where table_name='T1';
```

PCT_FREE	PCT_USED	AVG_ROW_LEN	CHAIN_CNT
10		21	0

考点：对于大部分目的而言，应该用 DBMS_STATS 包中的过程分析表，但要查看行链接或行迁移信息，只能通过 ANALYZE 命令检测。

11.4.2 表和数据块（block）的关系

1）什么是高水位线？

高水位线（high-water mark，HWM）

在数据库中，如果把表想象成从左到右依次排开的一系列块，高水位线就是曾经包含了数据的最右边的块。原则上 HWM 只会增大，即使将表中的数据全部删除，HWM 也不会降低。

HWM 不是好事，使用全表扫描时通常要读出 HWM 以下的所有数据块（尽管该表中可能仅有少量数据），这将占用大量 IO 资源。

2）两个解决办法可降低 HWM：

2.1)移动表,move 方法，将表从一个表空间移动到另一个表空间(也可以在本表空间内 move)，可以清除表里的碎片。

语法：alter table t1 move [tablespace users];

优点：可以清除数据块中的碎片,降低高水位线。
缺点：move 需要额外（一倍）的空间。
move 过程中会锁表，其他用户不能在该表上做 DML 或 DDL 操作。
move 之后，相关索引都不可用了，表上的索引需要重建（考点）。

2.2）收缩表，shrink 也叫段重组，表收缩的底层实现的是通过匹配的 INSERT 和 DELETE 操

作。

它分两个不同的阶段：压缩阶段和 DDL 命令阶段。(PPT-II-491)

语法：alter table t2 shrink space [cascade][compact];

两个前提：1）表所在的表空间段管理是 ASSM 方式,因为位图方法才记录有关块实际的满度信息(考点)。2）表上启用了 row movement。

你发出 alter table t2 shrink space compact; 那么只完成了第一阶段。这是压缩阶段。在业务高峰时可以先完成第一阶段

高峰过后，再次 alter table t2 shrink space; 因压缩阶段工作大部分已完成，将很快进入第二阶段，DML 操作会有短暂的锁等待发生。

测试：

```
create tablespace timran datafile '/u01/oradata/timran11g/timran01.dbf' size 100m;
create table scott.t2 tablespace timran as select * from dba_objects;

scott:
select max(rownum) from t2;
select table_name, blocks, empty_blocks, num_rows from user_tables where table_name='T2';
analyze table t2 compute statistics;
delete t2 where rownum<=40000;
commit;

analyze table t2 compute statistics for table;
select table_name, blocks, num_rows from user_tables where table_name='T2';
```

这时候，num_rows 已经减掉了 40000 条， 但 blocks 并没有减少， 说明 HWM 没有下降。

做 shrink

```
alter table t2 enable row movement;     //使能行移动

进行第一步----压缩阶段
alter table t2 shrink space compact;
analyze table t2 compute statistics for table;
select table_name, blocks, num_rows from user_tables where table_name='T2';
    //HWM 不会降低。
```

进行第二步----DDL 命令阶段

alter table t2 shrink space;
analyze table t2 compute statistics for table;
select table_name, blocks, num_rows from user_tables where table_name='T2';
//HWM 已经降低。

考点：
1，表收缩操作生成 undo 和 redo 数据，索引可以得到维护。
2，收缩分为两个阶段 第一阶段是压缩阶段，第二阶段是降低 HWM 阶段。SHRINK 不占用额外的空间。
3，可以单独完成第一阶段，即 SHRINK SPACE COMPACT 此阶段不降低 HWM，DML 操作几乎不受影响。
4，可以级联相关的段一起收缩，即 SHRINK SPACE CASCADE。
5，段必须 ASSM 管理方式，且使能行移动，否则不能收缩，如果不满足这两个前提，MOVE 就是重组表的唯一方式。
6，使用位图管理块，不能收缩 MSSM 管理，或有 LONG 列表或是有 refresh_on_commit 物化视图的表。

11.5 临时表空间

11.5.1 temporary tablespace 用途:

用于排序，可以建立多个临时表空间，但默认的临时表空间只能有一个，default temporary tablespace 不能 offline 和 drop。如果未指定默认的临时表空间,oracle 将会使用 system 作为临时表空间(非本地管理)，只有 temp 表空间是 nologing。

09:00:53 SQL> alter tablespace temp add tempfile '/u01/oradata/timran11g/temp01.dbf' size 100m reuse;

这等于在原地重置了临时表空间，可以在 dba_temp_files 里没有任何记录时使用这条命令;

09:01:14 SQL> select file_id,file_name,tablespace_name from dba_temp_files;

```
   FILE_ID FILE_NAME                                          TABLESPACE_NAME
---------- -------------------------------------------------- ------------------
------------
         1 /u01/oradata/timran11g/temp01.dbf                  TEMP
```

09:01:17 SQL> col name for a60;
09:01:19 SQL> select file#,name ,bytes/1024/1024 from v$tempfile;

```
     FILE# NAME                                               BYTES/1024/1024
---------- -------------------------------------------------- ---------------
         1 /u01/oradata/timran11g/temp01.dbf                              100
```

11.5.2 建立临时表空间 temp2，增加或删除 tempfile。

```
09:04:18 SQL> create temporary tablespace temp2 tempfile
'/u01/oradata/timran11g/temp02.dbf' size 10m;
09:05:00 SQL> alter tablespace temp2 add tempfile
'/u01/oradata/timran11g/temp03.dbf' size 5m;

SQL> select file_id,file_name,tablespace_name from dba_temp_files;

   FILE_ID                                                    FILE_NAME
TABLESPACE_NAME
---------- ------------------------------------------------------------
---------- ------------------------------
         1                                  /u01/oradata/timran11g/temp01.dbf
TEMP
         2                                  /u01/oradata/timran11g/temp02.dbf
TEMP2
         3                                  /u01/oradata/timran11g/temp03.dbf
TEMP2
```

将 temp2 里删掉一个 tempfile。

```
SQL> alter tablespace temp2 drop tempfile '/u01/oradata/timran11g/temp03.dbf';

SQL> select file_id,file_name,tablespace_name from dba_temp_files;

   FILE_ID                                                    FILE_NAME
TABLESPACE_NAME
---------- ------------------------------------------------------------
---------- ------------------------------
         1                                  /u01/oradata/timran11g/temp01.dbf
TEMP
         2                                  /u01/oradata/timran11g/temp02.dbf
TEMP2
```

11.5.3 查看默认的临时表空间

```
09:06:52 SQL> col PROPERTY_VALUE for a30
09:06:59 SQL> col description for a40
09:07:04 SQL> select * from database_properties;

PROPERTY_NAME                  PROPERTY_VALUE                 DESCRIPTION
------------------------------ ------------------------------ ------------------
----------------------
```

```
DICT.BASE                      2                              dictionary base
tables version #
DEFAULT_TEMP_TABLESPACE        TEMP                           Name of default
temporary tablespace
DEFAULT_PERMANENT_TABLESPACE   USERS                          Name of default
permanent tablespace
DEFAULT_TBS_TYPE               SMALLFILE                      Default tablespace
type
NLS_LANGUAGE                   AMERICAN                       Language
NLS_TERRITORY                  AMERICA                        Territory
.....

27 rows selected.
```

11.5.4 指定用户使用临时表空间

```
20:55:00 SQL> alter user scott temporary tablespace temp2;
```

//注意，与 default profile 不同，删除了 temp2，scott 的 temporary 不会转回到 temp。

11.5.5 切换默认的临时表空间

```
09:07:05 SQL> alter database default temporary tablespace temp2;
09:07:34 SQL> select * from database_properties;

PROPERTY_NAME                  PROPERTY_VALUE                 DESCRIPTION
------------------------------ ------------------------------ ------------------
----------------------
DICT.BASE                      2                              dictionary base
tables version #
DEFAULT_TEMP_TABLESPACE        TEMP2                          Name of default
temporary tablespace
DEFAULT_PERMANENT_TABLESPACE   USERS                          Name of default
permanent tablespace
DEFAULT_TBS_TYPE               SMALLFILE                      Default tablespace
type
NLS_LANGUAGE                   AMERICAN                       Language
...

09:07:41 SQL>
```

11.5.6 建立临时表空间组（10g 新特性）

在很多情况下，会有多个 session 使用同一个用户名去访问 Oracle，而临时表空间又是基于用户的，那么可以建立一个临时表空间组，组中由若干临时表空间构成，从而可以提高单个

用户多个会话使用表空间的效率。

1)临时表空间组无法显式创建，组是通过第一个临时表空间分配时自动创建。

```
09:07:41 SQL> alter tablespace temp tablespace group tmpgrp;

09:09:33 SQL> alter tablespace temp2 tablespace group tmpgrp;

09:09:38 SQL> select * from dba_tablespace_groups;

GROUP_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
TMPGRP                         TEMP
TMPGRP                         TEMP2
```

2)将临时表空间组设成默认临时表空间，实现负载均衡。

```
09:09:52 SQL> alter database default temporary tablespace tmpgrp;

Database altered.

09:10:10 SQL> select * from database_properties;

PROPERTY_NAME                  PROPERTY_VALUE                 DESCRIPTION
------------------------------ ------------------------------ ------------------
----------------------
DICT.BASE                       2                              dictionary base
tables version #
DEFAULT_TEMP_TABLESPACE         TMPGRP                         Name of default
temporary tablespace
DEFAULT_PERMANENT_TABLESPACE    USERS                          Name of default
permanent tablespace
DEFAULT_TBS_TYPE                SMALLFILE                      Default tablespace
type
NLS_LANGUAGE                    AMERICAN                       Language
NLS_TERRITORY                   AMERICA                        Territory
```

3)要移除表空间组时，该组不能是缺省的临时表空间。

```
SQL>alter database default temporary tablespace temp;


05:38:11 SQL> alter tablespace temp tablespace group '';
05:38:16 SQL> alter tablespace temp2 tablespace group '';
```

4)当组内所有临时表空间被移除时，组也被自动删除。

```
05:38:23 SQL> select * from dba_tablespace_groups;
no rows selected

SQL> drop tablespace temp2 including contents and datafiles;
```

考点：某个 tempfile 坏掉使得 default temporary tablespace 不能正常工作，数据库不会 crash, 解决的办法是 add 一个新的 tempfile，然后再 drop 掉坏的 tempfile.（default temporary tablespace 不能 offline，但 temporary file 可以 offline）

11.6 如何调整表空间的尺寸（表空间的大小等同它下的数据文件大小之和）

当发生表空间不足的问题时常用的 3 个解决办法：

1）增加数据文件大小（resize）

```
alter database datafile '/u01/oradata/timran11g/timran01.dbf' resize 10m;
```

2）增加数据文件（add datafile）

```
alter tablespace timran add datafile '/u01/oradata/timran11g/timran02.dbf' size 20m;
```

3）设置表空间自动增长（autoextend)

```
alter database datafile '/u01/oradata/timran11g/timran01.dbf' autoextend on next 10m maxsize 500m;
```

例:

```
SQL> create tablespace timran datafile '/u01/oradata/timran11g/timran01.dbf' size 5m;
05:46:08 SQL> create table scott.test1 (id int) tablespace timran;
05:47:12 SQL> insert into scott.test1 values(1);
05:47:15 SQL> insert into scott.test1 select * from scott.test1;
05:47:23 SQL> /
05:47:23 SQL> /
32768 rows created.
05:47:23 SQL> /
insert into scott.test1 select * from scott.test1
*
ERROR at line 1:
```

```
ORA-01653: unable to extend table SCOTT.TEST1 by 8 in tablespace TIMRAN

//用第一种方法扩充表空间
05:47:23 SQL> alter database datafile '/u01/oradata/timran11g/timran01.dbf'
resize 10m;

05:48:18 SQL> insert into scott.test1 select * from scott.test1;

05:48:25 SQL> /

131072 rows created.

05:48:26 SQL> /
insert into scott.test1 select * from scott.test1
*
ERROR at line 1:
ORA-01653: unable to extend table SCOTT.TEST1 by 128 in tablespace TIMRAN

//用第二种方法扩充表空间：

05:48:57 SQL> alter tablespace timran add datafile
'/u01/oradata/timran11g/timran02.dbf' size 20m;

05:49:04 SQL> insert into scott.test1 select * from scott.test1;

05:49:13 SQL> /

524288 rows created.

05:49:14 SQL> /
insert into scott.test1 select * from scott.test1
*
ERROR at line 1:
ORA-01653: unable to extend table SCOTT.TEST1 by 128 in tablespace TIMRAN

//用第三种方法扩充表空间：

05:49:15 SQL> alter database datafile '/u01/oradata/timran11g/timran01.dbf'
autoextend on next 10m maxsize 500m;

05:49:33 SQL> insert into scott.test1 select * from scott.test1;

05:49:37 SQL> drop tablespace timran including contents and datafiles;
```

11.7 可恢复空间分配 Oracle 的 Resumable（可恢复）功能 (PPT-II-502)

当我们往一个表里面插入大量数据时，如果某条 insert 语句因表空间的空间不足(没有开启自动扩展)，会报 ORA-01653:无法扩展空间的错误，该条 SQL 语句会中断，白白浪费了时间及数据库资源。为防范这个问题，Oracle 设计了一个功能：resumable。在 resumable 开启的情况下，如果 Oracle 执行某条 SQL 申请不到空间了，比如数据表空间，undob 表空间，temporary 空间等，则会将该事务的语句挂起（suspended)，等你把空间扩展后，Oracle 又会使该 insert 语句继续进行。

可以通过两个级别设置 resumable

system 级别：初始化参数 RESUMABLE_TIMEOUT 非 0，这将使数据库中所有 session 使用可恢复的空间分配
session 级别：alter session enable|disable resumable [TIMEOUT]; 这将为当前 session 设置可恢复的空间分配

因为 resumable 是有资源消耗代价的, 所以 session 级的 resumable 是比较实际的：

注意 TIMEOUT 的用法，单位为秒, 进一步要理解初始化参数 RESUMABLE_TIMEOUT 的含义

RESUMABLE_TIMEOUT=0, enable session 时应该指定 TIMEOUT。否则使用缺省值 7200 秒。
RESUMABLE_TIMEOUT<>0，enable session 时可以省略 TIMEOUT，此时指定 TIMEOUT 会覆盖掉参数 RESUMABLE_TIMEOUT 值。

举例：

session 1:

1）建个小表空间，固定 2m 大小，然后建个表属于这个表空间

```
SQL> create tablespace small datafile '/u01/oradata/timran11g/small01.dbf' size 2m;
SQL> create table scott.test(n1 char(1000)) tablespace small;
```

2）向这个表插入数据，表空间满了，使 for 语句没有完成循环，2000 条语句整体失败。

```
SQL>
begin
for i in 1..2000 loop
insert into scott.test values('this is test');
end loop;
commit;
end;
/
```

```
begin
*
第 1 行出现错误:
ORA-01653: 表 SCOTT.TEST 无法通过 128 (在表空间 SMALL 中) 扩展
ORA-06512: 在 line 3

SQL> select count(*) from scott.test;

  COUNT(*)
----------
         0
```

3）使能 resumable 功能

```
SQL> alter session enable resumable;
```

4）再重复第 2）步，会话被挂起；

session 2:

5）查看视图的有关信息

```
SQL> select session_id,sql_text,error_number from dba_resumable;

SESSION_ID SQL_TEXT                                           ERROR_NUMBER
---------- -------------------------------------------------- ------------
       136 INSERT INTO SCOTT.TEST VALUES('this is test')              1653

SQL> select sid,event,seconds_in_wait from v$session_wait where sid=136;

       SID                                                              EVENT
SECONDS_IN_WAIT
---------- ---------------------------------------------------------------- ----
-----------
       136    statement    suspended,    wait    error    to    be    cleared
1
```

6）加扩表空间，看到 session1 里挂起的会话成功完成了，不需要干预

```
SQL> alter tablespace small add datafile '/u01/oradata/timran11g/small02.dbf'
size 4m;

SQL> select count(*) from scott.test;
```

```
  COUNT(*)
----------
      2000
```

7）查看 EM 告警日志报告了以上信息。验证结束后可以 disable resumable, 并删除 small 表空间及数据文件。

session 1:

```
SQL> alter session disable resumable;
SQL> drop tablespace small including contents and datafiles;
```

考点：
1.下列三种情况可引起 resumable a)表空间上限超出，b)extents 到达最大值，c)quota 超出。
2.enable resumable 可以在一个 session 中多次挂起执行的语句,直到 disable resumable。
3.DBMS_RESUMABLE.SET_SESSION_TIMEOUT 可以延长当前 session 的 TIMEOUT，并立即有效。

第十二章： Oracle 中表的几种类型

1、表的功能：存储、管理数据的基本单元（二维表：有行和列组成）
2、表的类型：
   1）堆表：heap table ：数据存储时，行是无序的，对它的访问采用全表扫描。
   2）分区表 表>2G
   3）索引组织表（IOT）
   4）簇表
   5）临时表
   6）压缩表
   7）嵌套表

3、如何将普通表转换为分区表；
   11g 以前，1)create 分区表， 2)insert into 分区表 select * from 普通表; 3)rename 分区表名; 4)重建约束、索引、触发器。
   11g 以后，在线重定义分区表

12.1 分区表及其种类（10g）

1）Range Partitioning（范围分区）

scott:

```
SQL>create table sale(
product_id varchar2(5), sales_count number(10,2)
)
partition by range(sales_count)
(
  partition p1 values less than(1000),
  partition p2 values less than(2000),
  partition p3 values less than(3000)
);
```

查看信息：

```
select * from user_tab_partitions where table_name='SALE';

insert into sale values('1',600);
insert into sale values('2',1000);
insert into sale values('3',2300);
insert into sale values('4',6000);
commit;

select * from sale partition(p1);
select * from sale partition(p2);
```

增加一个分区

```
alter table sale add partition p4 values less than(maxvalue);
```

再看一下， 可以插入 6000 值了

```
select * from user_tab_partitions where table_name='SALE';
insert into sale values('4',6000);
```

看一下段的分配

```
SQL> select segment_name,segment_type,partition_name from user_segments;
```

12.1.1 默认情况下，如果对分区表的分区字段做超范围（跨段）update 操作，会报错——ORA-14402: 。如果一定要改，可以通过打开表的 row movement 属性来完成。

```
SQL> select rowid,t1.* from sale partition(p1) t1;

ROWID              PRODU SALES_COUNT
------------------ ----- -----------
AAASvUAAEAAAAGVAAA 1             600

SQL> update sale set sales_count=1200 where sales_count=600;
update sale set sales_count=1200 where sales_count=600
       *
```

第 1 行出现错误:
ORA-14402: 更新分区关键字列将导致分区的更改

```
SQL> alter table sale enable row movement;
SQL> update sale set sales_count=1200 where sales_count=600;
```

已更新 1 行。

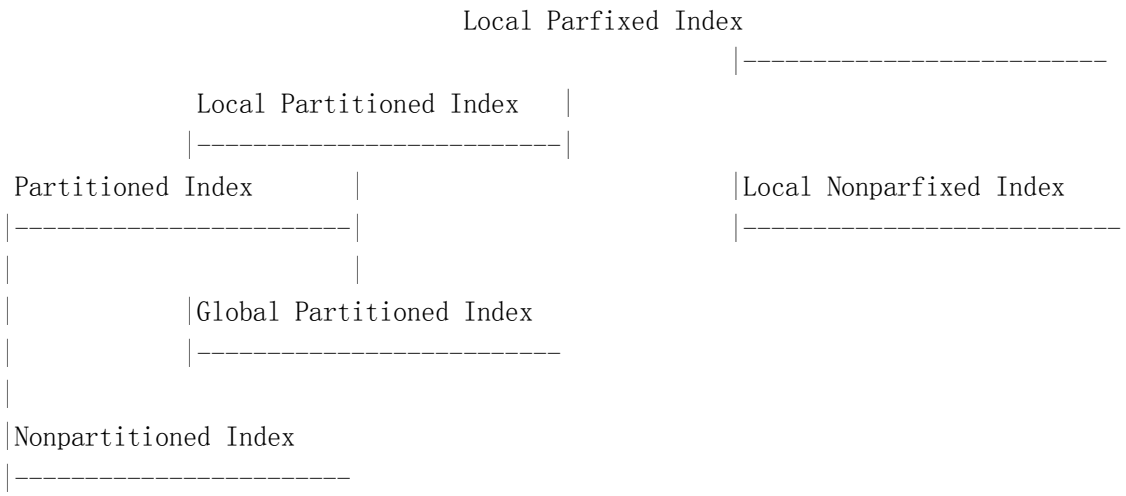
```
SQL> select rowid,t1.* from sale partition(p2) t1;

ROWID              PRODU SALES_COUNT
------------------ ----- -----------
AAASvVAAEAAAAGdAAA 2             1000
AAASvVAAEAAAAGdAAB 1             1200
```

一般来说范围分区的分区字段使用数字类型或日期类型，使用字符类型的语法是可以的，实际工作中使用较少。这或许跟 values less than 子句有关。

12.1.2 关于建立分区索引

一般使用分区都会建立索引，分区索引有 local 与 global 之分。



1) local：一个索引分区对应一个表分区，分区 key 就是索引 key，分区边界就是索引边界。更新一个表分区时仅仅影响该分区的索引。

```
SQL>create index sale_idx on sale(sales_count) local;
SQL>select * from user_ind_partitions;
```

Local Parfixed Index,所谓前缀索引,是指组合索引中的 first column 使用的是分区 key。

global：全局索引：

2）分区全局索引：索引分区不与表分区对应，分区 key 是索引 key。另外一定要将 maxvalue 关键字做上限。适用于 OLTP。

```
create index sale_global_idx on sale(sales_count) global
partition by range (sales_count)
(
partition p1 values less than(1500),
partition p2 values less than(maxvalue)
);

SQL>select * from user_indexes;
```

12.1.2 Hash Partitioning (散列分区,也叫 hash 分区)

实现均匀的负载值分配，增加 HASH 分区可以重新分布数据。

```
create table my_emp(
  empno number, ename varchar2(10)
)
partition by hash(empno)
(
  partition p1, partition p2
);

select * from user_tab_partitions where table_name='MY_EMP';
```

插入几个值，看是否均匀插入。

```
insert into my_emp values(1,'A');
insert into my_emp values(2,'B');
insert into my_emp values(3,'C');

select * from my_emp partition(P1);
select * from my_emp partition(P2);
```

12.1.3 列表分区(list)： 将不相关的数据组织在一起

```
create table personcity(
  id number, name varchar2(10), city varchar2(10)
)
partition by list(city)
(
  partition east values('tianjin','dalian'),
  partition west values('xian'),
```

```
  partition south values ('shanghai'),
  partition north values ('herbin'),
  partition other values (default)
);

insert into personcity values(1,'sohu','tianjin');
insert into personcity values(2,'sina','herbin');
insert into personcity values(3,'yahoo','dalian');
insert into personcity values(4,'360','zhengzhou');
insert into personcity values(5,'baidu','xian');
```

看结果

```
select * from personcity partition(east);
```

12.1.4 Composite Partitioning（复合分区）

把范围分区和散列分区相结合或者 范围分区和列表分区相结合。

```
create table student(
   sno number, sname varchar2(10)
)
partition by range(sno)
subpartition by hash(sname)
subpartitions 4
(
  partition p1 values less than(1000),
  partition p2 values less than(2000),
  partition p3 values less than(maxvalue)
);
```

有三个 range 分区，对每个分区会有 4 个 hash 分区，共有 12 个分区。

```
SQL> select * from user_tab_partitions where table_name='STUDENT';

SQL> select * from user_tab_subpartitions where table_name='STUDENT';
```

用 OEM 查看，看 scott 的 student table 子分区里的名字是由 oracle 取名。

12.2 Oracle11g 新增分区

Partition（分区），一直是 Oracle 数据库引以为荣的一项技术，正是分区的存在让 Oracle 高效的处理海量数据成为可能。在 Oracle11g 在 10g 的分区技术基础上又有了新的发展，使分区技术在易用性和可扩展性上再次得到了增强。

12.2.1 Interval Partitioning （间隔分区）

实际上是由 range 分区引申而来，最终实现了 range 分区的自动化。

```
scott:
SQL>
create table interval_sales (s_id int,d_1 date)
partition by range(d_1)
interval (numtoyminterval(1,'MONTH'))
(
   partition p1 values less than ( to_date('2010-02-01','yyyy-mm-dd') )
);

SQL> insert into interval_sales values(1, to_date('2010-01-21','yyyy-mm-dd') );
SQL> insert into interval_sales values(1, to_date('2010-02-01','yyyy-mm-dd') );
    --越过 p1 分区上线，将自动建立一个分区
SQL> select partition_name from user_tab_partitions;

PARTITION_NAME
------------------------------
P1
SYS_P61
```

注意：interval (numtoyminterval(1,'MONTH'))的意思就是每个月有一个分区，每当输入了新的月份的数据，这个分区就会自动建立，而不同年的相同月份是两个分区。

12.2.2 System Partitioning (系统分区)

这是一个人性化的分区类型，System Partitioning，在这个新的类型中，不需要指定任何分区键，数据会进入哪个分区完全由应用程序决定，即在 Insert 语句中决定记录行插入到哪个分区。

先建立三个表空间 tbs1,tbs2,tbs3， 然后建立三个分区的 system 分区表，分布在三个表空间上。

```
create table test (c1 int,c2 int)
partition by system
(
  partition p1 tablespace tbs1,
  partition p2 tablespace tbs2,
  partition p3 tablespace tbs3
);
```

现在由 SQL 语句来指定插入哪个分区：

```
SQL> INSERT INTO test PARTITION (p1) VALUES (1,3);
SQL> INSERT INTO test PARTITION (p3) VALUES (4,5);

SQL> select * from test;

        C1         C2
---------- ----------
         1          3
         4          5
```

注意：如果要删除以上表空间，必须先删除其上的分区表，否则会报错 ORA-14404: 分区表包含不同表空间中的分区。

12.2.3 Reference Partitioning (引用分区)

当两个表是主外键约束关联时，我们可以利用父子关系对这两个表进行分区。只要对父表做形式上的分区，然后子表就可以继承父表的分区键。

如果没有 11g 的引用分区，你想在两个表上都建立对应的分区，那么需要使两表分别有相同名称的键值列。引用分区的好处是避免了在子表上也建立父表同样的一个分区键列，父表上的任何分区维护操作都将自动的级联到子表上。

例：

```
SQL>
CREATE TABLE purchase_orders
  (po_id NUMBER(4),
   po_date TIMESTAMP,
   supplier_id NUMBER(6),
   po_total NUMBER(8,2),
   CONSTRAINT order_pk PRIMARY KEY(po_id))
PARTITION BY RANGE(po_date)
  (PARTITION Q1 VALUES LESS THAN (TO_DATE('2007-04-01','yyyy-mm-dd')),
   PARTITION Q2 VALUES LESS THAN (TO_DATE('2007-06-01','yyyy-mm-dd')),
   PARTITION Q3 VALUES LESS THAN (TO_DATE('2007-10-01','yyyy-mm-dd')),
   PARTITION Q4 VALUES LESS THAN (TO_DATE('2008-01-01','yyyy-mm-dd')));

//父表做了一个 Range 分区（可对引用分区使用除间隔分区外的所有分区策略）

SQL>
CREATE TABLE purchase_order_items
  (po_id NUMBER(4) NOT NULL,
   product_id NUMBER(6) NOT NULL,
   unit_price NUMBER(8,2),
```

```
quantity NUMBER(8),
CONSTRAINT po_items_fk FOREIGN KEY (po_id) REFERENCES purchase_orders(po_id))
PARTITION BY REFERENCE(po_items_fk);
```

//主表使用 po_date 键值列做范围分区，子表中没有 po_date 列，也想做相应的分区，那么使用引用分区吧。
//子表最后一句 PARTITION BY REFERENCE()子句给出了引用分区约束名。
//子表的 po_id 列必须是 NOT NULL。这与通常的外键可以是 NULL 是有区别的。

```
SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE from user_tab_partitions;

TABLE_NAME                     PARTITION_NAME                 HIGH_VALUE
------------------------------ ------------------------------ ------------------
------------------------------------------------------------
PURCHASE_ORDERS                 Q1                              TIMESTAMP' 2007-
04-01 00:00:00'
PURCHASE_ORDERS                 Q2                              TIMESTAMP' 2007-
06-01 00:00:00'
PURCHASE_ORDERS                 Q3                              TIMESTAMP' 2007-
10-01 00:00:00'
PURCHASE_ORDERS                 Q4                              TIMESTAMP' 2008-
01-01 00:00:00'
PURCHASE_ORDER_ITEMS           Q1
PURCHASE_ORDER_ITEMS           Q2
PURCHASE_ORDER_ITEMS           Q3
PURCHASE_ORDER_ITEMS           Q4

8 rows selected
```

//子表 purchase_order_items 也自动产生了四个分区，Q1,Q2,Q3,Q4.高值为空，意味者此处边界由父表派生。

```
SQL>    select    TABLE_NAME,PARTITIONING_TYPE,REF_PTN_CONSTRAINT_NAME    from
user_part_tables;

TABLE_NAME                     PARTITIONING_TYPE REF_PTN_CONSTRAINT_NAME
------------------------------ ----------------- ------------------------------
PURCHASE_ORDERS                RANGE
PURCHASE_ORDER_ITEMS           REFERENCE         PO_ITEMS_FK
```

// PO_ITEMS_FK 列是外键约束名称

12.2.4 Virtual Column-Based Partitioning（虚拟列分区）

先了解一下什么叫虚拟列。

虚拟列是 11g 的新特性：

1> 只能在堆组织表（普通表）上创建虚拟列
2> 虚拟列的值并不是真实存在的，只有用到时，才根据表达式计算出虚拟列的值，磁盘上并不存放。
3> 可在虚拟列上建立索引。
4> 如果在已经创建的表中增加虚拟列时，若没有指定虚拟列的字段类型，ORACLE 会根据 generated always as 后面的表达式计算的结果自动设置该字段的类型。
5> 虚拟列的值由 ORACLE 根据表达式自动计算得出，不可以做 UPDATE 和 INSERT 操作，可以对虚拟列做 DELETE 操作。
6> 表达式中的所有列必须在同一张表。
7> 表达式不能使用其他虚拟列。

8> 可把虚拟列当做分区关键字建立虚拟列分区表，这正是我们要讲的虚拟列分区。

```
create table emp1
  (empno number(4) primary key,
   ename char(10) not null,
   salary number(5) not null,
   bonus number(5)  not null,
   total_sal AS (salary+bonus))
partition by range (total_sal)
  (partition p1 values less than (5000),
   partition p2 values less than (maxvalue))
   enable row movement;

insert into emp1(empno,ename,salary,bonus) values(7788,'SCOTT',3000,1000);
insert into emp1(empno,ename,salary,bonus) values(7902,'FORD',4000,1500);
insert into emp1(empno,ename,salary,bonus) values(7839,'KING',5000,3500);
commit;

SQL> select * from user_tab_partitions;
SQL> select * from user_part_key_columns;

SQL> select * from emp1 partition (p1);

     EMPNO ENAME          SALARY       BONUS  TOTAL_SAL
---------- ---------- ---------- ---------- ----------
      7788 SCOTT            3000        1000       4000

SQL> select * from emp1 partition (p2);
```

EMPNO	ENAME	SALARY	BONUS	TOTAL_SAL
7902	FORD	4000	1500	5500
7839	KING	5000	3500	8500

```
SQL> update emp1 set bonus=500 where empno=7902;
```

在建表时就使能了行移动(enable row movement),当更新分区键值时就不会报错(ORA-14402:更新分区关键字列将导致分区的更改)

12.2.5 More Composite Partitioning
在 10g 中，我们知道复合分区只支持 Range-List 和 Range-Hash，而在在 11g 中复合分区的类型大大增加，现在 Range，List，Interval 都可以作为 Top level 分区，而 Second level 则可以是 Range，List，Hash，也就是在 11g 中可以有 3*3=9 种复合分区，满足更多的业务需求。

12.3 Oracle11g 的联机功能

联机功能包括

联机数据重组：主要是移动表，添加约束等
联机数据重定义：主要指的是在线对列进行添加，删除，重命名，改变列类型，联机创建一个分区表等

例：联机创建分区表：将 emp1 表联机重定义，要求完成两个任务，使其按照 sal 分区（以 2500 为界），并去掉 comm 列。这个过程需要借用一个临时表 emp1_temp。

sys 下执行

```
create table scott.emp1 as select * from scott.emp;

alter table scott.emp1 add constraint pk_emp1 primary key(empno);
```

1) 检查原始表是否具有在线重定义资格,(要求表自包含及之前没有建立实体化视图及日志)

```
SQL>
BEGIN
  DBMS_REDEFINITION.CAN_REDEF_TABLE('scott','emp1');
END;
/
```

2) 创建一个临时表:emp1_temp, 含有 7 列（删去 comm 列），然后 range 分区，两个区以 sal=2500 为界。

```
SQL>
CREATE TABLE scott.emp1_temp
```

```
  (empno        number(4) not null,
   ename        varchar2(10),
   job          varchar2(9),
   mgr          number(4),
   hiredate     date,
   sal          number(7,2),
   deptno       number(2))
PARTITION BY RANGE(sal)
   (PARTITION sal_low VALUES LESS THAN(2500),
   PARTITION sal_high VALUES LESS THAN (maxvalue));
```

3）启动联机重定义处理过程

```
SQL>
BEGIN
  dbms_redefinition.start_redef_table('scott','emp1','emp1_temp',
   'empno empno,
   ename ename,
   job job,
   mgr mgr,
   hiredate hiredate,
   sal sal,
   deptno deptno');
END;
/
```

```
SQL> select count(*) from scott.emp1_temp;

  COUNT(*)
----------
        14

SQL> select * from scott.emp1_temp partition(sal_low);

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL
DEPTNO
---------- ---------- --------- ---------- ------------------- ---------- ------
----
      7369 SMITH      CLERK             7902 1980-12-17 00:00:00         800
20
      7499 ALLEN      SALESMAN          7698 1981-02-20 00:00:00        1600
30
```

```
      7521 WARD       SALESMAN        7698 1981-02-22 00:00:00       1250
30
      7654 MARTIN     SALESMAN        7698 1981-09-28 00:00:00       1250
30
      7782 CLARK      MANAGER         7839 1981-06-09 00:00:00       2450
10
      7844 TURNER     SALESMAN        7698 1981-09-08 00:00:00       1500
30
      7876 ADAMS      CLERK           7788 1987-05-23 00:00:00       1100
20
      7900 JAMES      CLERK           7698 1981-12-03 00:00:00        950
30
      7934 MILLER     CLERK           7782 1982-01-23 00:00:00       1300
10
```

已选择 9 行。

```
SQL> select * from scott.emp1_temp partition(sal_high);

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL
DEPTNO
---------- ---------- --------- ---------- ------------------- ---------- ------
----
      7566 JONES      MANAGER         7839 1981-04-02 00:00:00       2975
20
      7698 BLAKE      MANAGER         7839 1981-05-01 00:00:00       2850
30
      7788 SCOTT      ANALYST         7566 1987-04-19 00:00:00       3000
20
      7839 KING       PRESIDENT            1981-11-17 00:00:00       5000
10
      7902 FORD       ANALYST         7566 1981-12-03 00:00:00       3000
20
```

已选择 5 行。

这个时候 emp1_temp 的主键，索引，触发器，授权等还没有从原始表继承过来，

```
SQL> select constraint_name,constraint_type,table_name from user_constraints
where table_name like 'EMP1%';

CONSTRAINT_NAME                CONSTRAINT_TYPE TABLE_NAME
------------------------------ --------------- ------------------------------
PK_EMP1                        P               EMP1
```

```
SYS_C009652                    C               EMP1_TEMP
```

4) 复制依赖对象

```
SQL>
DECLARE
  num_errors PLS_INTEGER;
BEGIN
  DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('scott','emp1','emp1_temp',
  DBMS_REDEFINITION.CONS_ORIG_PARAMS,TRUE,TRUE,TRUE,TRUE,num_errors);
END;
/

SQL> select constraint_name,constraint_type,table_name from user_constraints
where table_name like 'EMP1%';

CONSTRAINT_NAME                CONSTRAINT_TYPE TABLE_NAME
------------------------------ --------------- ------------------------------
PK_EMP1                        P               EMP1
SYS_C009652                    C               EMP1_TEMP
TMP$$_PK_EMP10                 P               EMP1_TEMP
```

这时候原始表 emp1 还没有分区，

```
SQL> select table_name,partition_name,high_value from user_tab_partitions;

TABLE_NAME                     PARTITION_NAME                 HIGH_VALUE
------------------------------ ------------------------------ ------------------
----------------------------------------

EMP1_TEMP                      SAL_HIGH                       MAXVALUE
EMP1_TEMP                      SAL_LOW                        2500
```

5) 完成重定义过程。

```
SQL> EXECUTE dbms_redefinition.finish_redef_table('scott','emp1','emp1_temp');

SQL> select table_name,partition_name,high_value from user_tab_partitions;

TABLE_NAME                     PARTITION_NAME                 HIGH_VALUE
------------------------------ ------------------------------ ------------------
----------------------------------------

EMP1                           SAL_HIGH                       MAXVALUE
EMP1                           SAL_LOW                        2500
```

最后这一步发生了什么事情：

1.Oracle 读取原始表的实体化日志，向临时表添加内容,即原始表同步临时表。
2.原始表 emp1 被重新定义，并具有临时表 emp1_temp 的全部属性：分区、索引、约束和授权以及触发器
3.实体化视图和日志将被删除。

12.4 索引组织表(IOT 表：如果表经常以主键查询，可以考虑建立索引组织表，加快表的访问速度

heap table 数据的存放是随机的，获取表中的数据时没有明确的先后之分，在进行全表扫描的时候，并不是先插入的数据就先获取。而 IOT 表是一个完全 B_tree 索引结构的表，表结构按照索引（主键）有序组织，因此数据存放在插入以前就已经确定了其位置，也就是说，数据在那个物理位置与插入的先后顺序无关。

由于 IOT 表是把普通表和索引合二而一了,这样在进行查询的时候就可以少访问很多 blocks，但是插入和删除的时，速度比普通的表要慢一些。

IOT 表的叶子节点存储了所有表列,因为已按主键排序,所以叶子节点上不需要再存储 rowid。

表较大时，设置溢出段将主键和其他字段数据分开来存储以提高效率。

溢出段是个可选项,如果选择了溢出段,Oracle 将为一个 IOT 表分配两个段，一个是索引段，另一个是溢出段。

溢出段有两个子句 pctthreshold 和 including

说明：
pctthreshold 给出行大小和块大小的百分比，当行数据在叶子节点占用大小超出这个阀值时，就以这个阀值将索引 entry 一分为二，包含主键的一部分列值保留在索引段，而其他列值放入溢出段，即 overflow 到指定的存储空间去。

including 后面指定一个或多个列名（不含主键列），将这些列都放入索引段。即让主键和一些常用的列在索引段，其余的列在溢出段。

例：

```
create table iot_timran(id int, name char(50), sal int,
constraint pk_timran primary key (id))
organization index pctthreshold 30 overflow tablespace users;
```

使用 select * from user_indexes 查看是否单独有索引。

```
SQL> select index_name,index_type,table_name from user_indexes;

INDEX_NAME                     INDEX_TYPE                  TABLE_NAME
------------------------------ --------------------------- ----------------------
---------
PK_TIMRAN                      IOT - TOP                   IOT_TIMRAN
PK_EMP                         NORMAL                      EMP
PK_DEPT                        NORMAL                      DEPT
```

通过 user_segments 视图查看产生了两个段。

```
SQL> select segment_name,segment_type,partition_name from user_segments;
```

12.5 簇表(cluster table)：

两个相互关联的表的数据，物理上同时组织到一个簇块中，当以后进行关联读取时，只要扫描一个数据块就可以了，可以提高了 IO 效率。

建立簇表的三个步骤：

1）建立簇段 cluster segment
2）基于簇，创建两个相关表，这两个表不建立单独的段，每个表都关联到 cluster segment 上。
3）为簇创建索引，生成索引段。

```
create cluster cluster1(code_key number);
create table student(sno1 number, sname varchar2(10)) cluster cluster1(sno1);
create table address(sno2 number,zz varchar2(10)) cluster cluster1(sno2);
create index index1 on cluster cluster1;
```

生成了 cluster1 段和 index1 段。

查看簇的信息：

```
select * from user_clusters;
select * from user_clu_columns;
```

删除簇：

```
drop table student;
drop table address;
drop cluster cluster1;
```

12.6 临时表：多用于电子商务的网上购物

临时表存放在当前登录的临时表空间下，它被每个 session 单独使用，即：不同 session 看

到的临时表中的数据不一样。

每个 session 独立支持 rollback，基于事务的临时段在事务结束后收回临时段，基于会话的临时段在会话结束后收回临时段，总之没有 DML 锁，没有约束，可以建索引，视图和触发器，由于会产生少量 UNDO 信息所以会产生少量 redo,节省资源，访问数据快。

两种模式：
1）基于事务的临时段：在事务提交时，就会自动删除记录，on commit delete rows。
2）基于会话的临时段：当用户退出 session 时，才会自动删除记录, on commit preserve rows。

例：scott：
```
create global temporary table tmp_student(sno int,sname varchar2(10), sage int)
on commit preserve rows;
```

再用 Scott 开一个 session
两边插入记录看看， 你可以在两个 session 里插入同样的记录，井水不犯河水！

要删除临时表，要所有 session 断开连接，再做删除。

```
drop table tmp_table
```

12.7 只读表 (11g 新特性)

在以前版本中，有只读表空间但没有只读表。11g 中增加了新特性----只读表。

```
SQL> alter table t read only;
SQL> update t set id=2;
update t set id=2
       *
第 1 行出现错误:
ORA-12081: 不允许对表 "SCOTT"."T" 进行更新操作

SQL> alter table t read write;
```

考点：只读表可以 drop，因为只需要在数据字典做标记，但是不能做 DML，另外，truncate 也不行，因为它们都在对只读表做写操作。

12.8 压缩表 （11g 新特性）

目的：去掉表列中数据存储的重复值，提高空间利用率。对数据仓库类的 OLAP 有意义

可以压缩：堆表（若指定表空间则压缩该表空间下所有表），索引表，分区表，物化视图。

主要压缩形式有两种：

1)Basic table compression 使用 direct path loads(缺省)，典型的是建立大批量的数据，如：create table as select...结构

Basic 对应的语法是：CREATE TABLE ... COMPRESS BASIC;

2)Advanced row compression 针对 OLTP 的任何 SQL 操作。

Advanced 11gR2 较之前版本在语法上有了变化

CREATE TABLE ... COMPRESS FOR OLTP...
代替
CREATE TABLE ... COMPRESS FOR ALL OPERATIONS

两种压缩的原理类似(PPT-II-481-482)：当 insert 达到 pctfree=阀值（basic 对应的 pctfree=0, Advanced 对应的是 pctfree=10)，触发 compress，之后可以继续 insert,再达到 pctfree，再触发 compress....直至 compress 数据填满 block 的 pctfree 以下部分。

压缩的是 block 中的冗余数据，这对节省 db buffer 有益。例如一个表有 7 个 columns，5 rows，其中的一些 column 有重复的行值

2190,13770,25-NOV-00,S,9999,23,161
2225,15720,28-NOV-00,S,9999,25,1450
34005,120760,29-NOV-00,P,9999,44,2376
9425,4750,29-NOV-00,I,9999,11,979
1675,46750,29-NOV-00,S,9999,19,1121

压缩这个表后，存储形式成为如下，重复值用符号替代。

2190,13770,25-NOV-00,S,%,23,161
2225,15720,28-NOV-00,S,%,25,1450
34005,120760,*,P,%,44,2376
9425,4750,*,I,%,11,979
1675,46750,*,S,%,19,1121

那么自然要对这些符号做些说明，相当于有个符号表

Symbol	Value	Column	Rows
*	29-NOV-00	3	958-960
%	9999	5	956-960

第十三章： 数据库审计 audit(PPT-I-320-334)

13.1 审计的功能：监控特定用户在 database 的 action（操作）

13.2 审计种类：

1）标准数据库审计（语句审计、权限审计、对象审计）
2）基于值的审计（Value-Based, 触发器审计）
3）精细审计（FGA）

13.3 启用审计（默认不启用）

09:55:23 SQL> show parameter audit

NAME	TYPE	VALUE
audit_file_dest	string	/u01/admin/timran11g/adump
audit_sys_operations	boolean	FALSE
audit_syslog_level	string	
audit_trail	string	DB

SQL>

audit_trail 参数三个选项

1）none 不启用 audit
2）db 将审计结果放在数据字典基表 sys.aud$中，(一般用于审计非 sys 用户，也可以移出 system 表空间，在 DB 中的好处是方便检索）
3）os 将审计结果存放到操作系统的文件里（由 audit_file_dest 指定的位置,一般用于审计 sys）

审计 sys
1)Oracle 强制审计 sys 用户的特权操作，如启动和关闭数据库，结果记录在参数 audit_file_dest 指向的.aud 文件中。
2)指定参数 audit_sys_operations = true 和 audit_trail = os

13.4 标准数据库审计的三个级别

查看标准审计结果可以通过视图 dba_audit_taile, 该视图读取 aud$内容。

13.4.1 语句审计

按语句来审计，比如 audit table 会审计数据库中所有的 create table, drop table, truncate table 语句，执行成功或不成功都可审计。

SQL> audit table;

13.4.2 权限审计

按权限来审计，当用户使用了该权限则被审计，如执行 grant select any table to a; 当用户 a 访问了用户 b 的表时（如 select * from b.t;）会用到 select any table 权限，故会被审计。用户访问自己的表不会被审计。

```
SQL> audit select any table;
```

13.4.3 对象审计

按对象审计，只审计 on 关键字指定对象的相关操作，如：aduit alter,delete,drop,insert on cmy.t by scott; 这里会对 cmy 用户的 t 表进行审计，但同时使用了 by 子句，所以只会对 scott 用户发起的操作进行审计。

```
SQL> audit update on scott.emp;
```

13.5 基于值的审计。

它拓展了标准数据库审计，不仅捕捉审计事件，还捕捉那些被 insert,update 和 delete 的值。由于基于值的审计是通过触发器来实现。所以你可以选择哪些信息进入审计记录，比如，只记录提交的信息，或不记录已改变的数据等。（考点）

例，记录下 scott 的登录轨迹。

```
linux 下 sys:
SQL> truncate table aud$;
SQL> audit session by scott;
SQL> select count(*) from aud$;

  COUNT(*)
----------
         0
cmd 下 scott：

C:\Documents and Settings\timran>sqlplus scott/scott@timran11g

linux 下 sys:

SQL> select count(*) from aud$;

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

```
col username for a10;
col userhost for a30;
SQL> select username,userhost,timestamp,action_name from dba_audit_trail;

USERNAME   USERHOST                       TIMESTAMP           ACTION_NAME
---------- ------------------------------ ------------------- ----------------
----------
SCOTT      WORKGROUP\TIMRAN-222C75E5      2014-01-09 13:35:53 LOGON

SQL> noaudit session by scott;
```

13.6 精细审计 Fine Grained Auditing (FGA)。它拓展了标准数据库审计，捕捉准确的 SQL 语句。

OEM 中只有标准数据库审计，目前还没有包括基于值的审计和精细审计。

13.6.1 举例：精细审计 Fine Grained Auditing (FGA)，审计访问特定行或特定行的特定列。操作可以使用 dbms_fga 包。

```
sys:
SQL> create table scott.emp1 as select * from scott.emp;
SQL> grant all on scott.emp1 to tim;
```

1）添加一个精细度审计策略

```
begin
dbms_fga.add_policy(
object_schema=>'scott',
object_name=>'emp1',
policy_name=>'chk_emp1',
audit_condition =>'deptno=20',
audit_column =>'sal',
statement_types =>'update,select');
end;
/
```

2）测试一下步骤，再查看审计结果

```
scott:
SQL> select * from emp1 where deptno=20;

tim:
SQL>update scott.emp1 set sal=8000 where empno=7902;
SQL>select empno,ename from scott.emp1 where deptno=20;     //缺少 sal 列，不审计
```

sys:
SQL> select empno,ename,sal from scott.emp1 where deptno=20; //虽然符合条件，但 sys 默认不审计

3）验证审计结果

11:32:24 SQL> conn /as sysdba
SQL> select db_user,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss') "time" ,sql_text from dba_fga_audit_trail;

DB_USER time SQL_TEXT
------------------------------ ------------------- ------------------------------
----
SCOTT 2013-08-17 16:57:36 select * from emp1 where deptno=20
TIM 2013-08-17 16:57:52 update scott.emp1 set sal=8000 where empno=7902

可以看出，必须同时满足了所有审计条件（前面定义的）才能入选。另外没有审计 SYS.

SQL> delete from fga_log$; //清除审计记录
SQL> commit;
SQL> select db_user,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss') "time" ,sql_text from dba_fga_audit_trail;

no rows selected

4）删除 FGA 的 PL/SQL API 命令

exec
dbms_fga.drop_policy(object_schema=>'scott',object_name=>'emp1',policy_name=>'chk_emp1');

考点:视图 DBA_AUDIT_TRIAL 显示标准数据库审计，DBA_FGA_AUDIT_TRIAL 显示 FGA，DBA_COMMON_AUDIT_TRIAL 则两者都显示.要查看触发器审核结果，必须创建处理自己定义的表的自定义视图。

第十四章： 数据装载 sql loader(PPT-I-490-498)

14.1 sql*loader：将外部数据（比如文本型）数据导入 oracle database。(用于数据导入、不同类型数据库数据迁移)

14.2 sql*loader 导入数据原理：在段（segment 表）insert 记录

1）conventional：将记录插入到 segment 的 HWM(高水位线）以下的块，要首先访问 bitmap ，来确定那些 block 有 free space

2）direct path：将记录插入到 segment 的 HWM(高水位线）以上的从未使用过的块，绕过 db_buffer,不检查约束。还可以关闭 redo, 也支持并行操作，加快插入速度。

例：

```
SQL> create table emp1 as select * from emp where 1=2;

SQL> insert into emp1 select * from emp;            //conventional 方式插入数据

SQL> insert /*+ APPEND */ into emp1 select * from emp;          //direct path方式插入数据, 必须 commit 后才能查看数据
```

14.3 sql*loader 用法

```
SQLLDR keyword=value [,keyword=value,...]
```

看帮助信息

$/u01/oracle/bin/sqlldr(回车)。如果要使用 direct path 方式，在命令行中使用关键字 direct=TRUE

*考点：sql*loader 与 data dump 的一个区别：data dump 只能读取由它导出的文件，而 sql*loader 可以读取任何它能解析的第三方文件格式

14.4 例子：

1）模拟生成数据源

```
11:02:13                                SQL>                                select empno||','||ename||','||job||','||mgr||','||hiredate||','||sal||','||comm||','||deptno from scott.emp;


EMPNO||','||ENAME||','||JOB||','||MGR||','||HIREDATE||','||SAL||','||COMM||','||DEPTNO
--------------------------------------------------------------------------------
----------------------------------------
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
```

```
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10

14 rows selected.
```

2)建个目录

```
[oracle@timran]$mkdir -p /home/oracle/sqlload
[oracle@timran]$cd /home/oracle/sqlload
[oracle@timran sqlload]$vi emp.dat      --生成平面表
```

--------查看数据源

```
[oracle@timran sqlload]$ more emp.dat
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
```

3）conventional 方式导入

建立控制文件

```
[oracle@work sqlldr]$ vi emp.ctl

load data
```

```
infile '/home/oracle/sqlload/emp.dat'
insert              --insert 插入表必须是空表，非空表用 append
into table emp1
fields terminated by ','
optionally enclosed by '"'
(
empno,
ename,
job,
mgr,
hiredate,
comm,
sal,
deptno)
```

4)在 scott 下建立 emp1 表（内部表），只要结构不要数据

```
11:10:13 SQL> create table emp1 as select * from emp where 1=2;
```

5)执行导入（normal）

```
[oracle@timran timran]$ sqlldr scott/scott control=emp.ctl log=emp.log

SQL*Loader: Release 10.2.0.1.0 - Production on Thu Aug 11 12:18:36 2011
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
Commit point reached - logical record count 14
```

5)验证：

```
11:07:12 SQL>
11:07:12 SQL> select * from emp1;
```

上例的另一种形式是将数据源和控制文件合并在.ctl 里描述

```
[oracle@work sqlldr]$ vi emp.ctl
load data
infile *
append
into table emp1
fields terminated  by   ','
optionally enclosed by '"'
(
empno,
ename,
job,
```

```
mgr,
hiredate,
comm,
sal,
deptno)
begindata
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10

[oracle@timran sqlload]$ sqlldr scott/scott control=emp.ctl log=emp.log

Commit point reached - logical record count 15
[oracle@timran sqlload]$
[oracle@timran sqlload]$
[oracle@timran sqlload]$
[oracle@timran sqlload]$ ll
总计 12
-rw-r--r-- 1 oracle oinstall    1 07-17 11:09 emp.bad
-rw-r--r-- 1 oracle oinstall  782 07-17 11:09 emp.ctl
-rw-r--r-- 1 oracle oinstall 2055 07-17 11:09 emp.log
[oracle@timran sqlload]$ more emp.bad

11:09:34 SQL>SQL> select count(*) from emp1;

  COUNT(*)
----------
        28
```

第十五章： Oracle 网络

15.1 Oracle Net 基本要素：

15.1.1 服务器端的 listener (监听器)
1)listener:在 oracle server 上启动,负责接收 user process 并派生 server process ,与 user process 建立 session
2) 建立 listener : 通过 netca 或 netmgr
3) listener 的注册: oracle server 信息,有动态注册和静态注册两种
4) listener 的启动/关闭/查看 lsnrctl start|stop|status
5)$ORACLE_HOME/network/admin/listener.ora 配置程序,动态注册可以不使用这个配置程序。

15.2 客户端链接:

1) $ORACLE_HOME/network/admin/tnsnames.ora 配置文件。
如:

```
myoracle =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.88)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = timran11g)
    )
  )
```

2) 注意几个地方
2.1 HOST=目标机的 IP,如果是 windows 平台,习惯使用主机名。PORT=1521 这是动态注册的默认端口号。
2.2 SERVICE_NAME 是目标机的 db_name.db_domain 单实例下缺省就是 instance 名。
2.3 myoracle 是连接符,代表的是等号后面的网络连接描述,代表了对三个要素(ip 地址,端口号, server_name)的网络描述。为便于记忆,它也常常和 SERVICE_NAME 同名。但含义不同。

```
C:\Documents and Settings\timran>sqlplus sys/oracle@myoracle as sysdba
```

试试没有配置 tnsnames.ora 下的轻松连接模式

```
C:\Documents and Settings\timran>sqlplus scott/scott@192.168.0.88:1521/timran11g
```

15.3 lisenter 注册

1) 动态监听: listener 采用的是默认端口(1521),当实例启动时,由 pmon 每分钟自动将 service name,本机的 1521 端口号注册到 listener
2) 静态监听:当 listener 一般使用的是非标准端口(如 1522),在 listener.ora 的文件里

手工注册（添加 GLOBAL_DBNAME 和 instance name）

15.3.1 监听器的动态注册和静态注册

静态注册：
1)静态注册不需要数据库打开,通过读取 listener.ora 的静态注册描述完成监听器的注册，因为不需要数据库 open,所以如果服务器端一旦启动了静态监听，便可以通过 sqlplu 以 sys 用户连接到服务器，实现远程启动/关闭数据库的任务。
2）静态注册可以使用用户指定的端口号（非 1521），相对隐蔽，安全。
3）静态注册在一些特殊场合，如使用数据库复制技术时是很有用处的。

动态注册：
1）需要数据库打开才能注册成功，所以动态注册无法使用 sysdba 身份远程启动数据库。一般都是通过远程 TELNET 先以 root 登录服务器，这时你已经在服务器本地了，再转入 oracle 以 sysdba 身份打开数据库。之后监听器才可以进行动态注册。
2）使用标准 1521 端口，自动注册。
3）可以不使用 listener.ora，因为动态注册由 PMON 后台进程自动注册信息，PMON 每 60 秒查看 listener 进程是否启动，启动了就注册相关服务器信息。

例：静态注册监听器一定要描述 listener.ora，有两种方法：

第一种 静态和动态都使用 1521 端口注册。

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.88)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )

SID_LIST_LISTENER =
    (SID_LIST =
    (SID_DESC =
        (GLOBAL_DBNAME= timran11g)
        (ORACLE_HOME = /u01/oracle)
        (SID_NAME = timran11g)
      )
)
```

启动|停止|查看---静态监听器

```
[oracle@timran admin]$ lsnrctl start|stop|status
```

第二种，静态注册监听器使用其他端口，如 1522 端口。

```
LSN2 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.88)(PORT = 1522))
    )
  )

SID_LIST_LSN2 =
    (SID_LIST =
    (SID_DESC =
        (GLOBAL_DBNAME= timran11g)
        (ORACLE_HOME = /u01/oracle)
        (SID_NAME = timran11g)
      )
)
```

启动|停止|查看---静态监听器 lsn2

```
[oracle@timran admin]$ lsnrctl start|stop|status lsn2

...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=192.168.0.66)(PORT=1522)))
Services Summary...
Service "prod" has 1 instance(s).
  Instance "prod", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
```

注意：端口是 1522，且 status UNKNOWN 这两个信息都说明监听是以静态注册的

客户端的 tnsnames.ora 中的静态注册描述部分

```
lsn2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.8.88)(PORT = 1522))
    )
    (CONNECT_DATA =
      (SERVICE_NAME =timran11g)
    )
```

)

*考点：

1）如果没有使用 RAC，那么监听器必须与实例在同一台主机上运行。在 RAC 环境下，集群中任何主机的任何监听器作用都是一样的。
2）如果监听器停止运行，那么不能启动任何新的服务器进程，但不会影响先前已经建立的会话。

15.4 配置共享连接模式：sharded server mode

服务器端的几个参数

dispatchers 调度进程数，以此参数一次性作为 dispatcher process 预先启动。一个 despatcher process 理论上可以支持 256 个 user process 请求。
max_dispatchers 此参数规定了 dispatcher process 最大数量，dispatchers<=max_dispatchers

shared_servers 共享服务器进程数，与 dispatcher process 数有关 最大不超过 shared_server_sessions
shared_server_sessions 此参数是所有 session 可使用的共享进程的最大值。

```
SQL> alter system set dispatchers='(protocol=tcp)(dispatchers=4)';
SQL> alter system set shared_servers=3;
SQL> show parameter dispatcher

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
dispatchers                          string      (protocol=tcp)(dispatchers=4)
max_dispatchers                      integer

SQL> show parameter shared_server

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
max_shared_servers                   integer
shared_server_sessions               integer
shared_servers                       integer     3
```

查看状态：

```
[oracle@timran ~]$ ps -ef |grep ora_d0
oracle    4647     1  0 12:05 ?        00:00:00 ora_d000_timran11g
oracle    4803     1  0 12:15 ?        00:00:00 ora_d001_timran11g
```

```
oracle    4807     1  0 12:15 ?        00:00:00 ora_d002_timran11g
oracle    4811     1  0 12:15 ?        00:00:00 ora_d003_timran11g
oracle    4815  3303  0 12:15 pts/2    00:00:00 grep ora_d0

[oracle@timran ~]$ ps -ef |grep ora_s0
oracle    4743     1  0 12:08 ?        00:00:00 ora_s000_timran11g
oracle    4747     1  0 12:08 ?        00:00:00 ora_s001_timran11g
oracle    4751     1  0 12:08 ?        00:00:00 ora_s002_timran11g
oracle    4827  3303  0 12:15 pts/2    00:00:00 grep ora_s0

C:\Documents and Settings\timran>sqlplus sys/system@timran11g as sysdba

SQL> select circuit,dispatcher,status from v$circuit;

CIRCUIT  DISPATCH STATUS
-------- -------- ----------------
36F80678 3925F5AC NORMAL

[root@timran ~]# netstat -anp |grep lsn
```

客户端:tnsnames.ore
1(SERVER = DEDICATED): 表示此连接使用 DEDICATED SERVER MODE 缺省的配置。
2(SERVER = SHARED): 表示此连接使用 SHARED SERVER MODE, 如果服务器端未配置 dispatchers, 此连接失败
3 不做任何说明: 如果服务器端配置了 dispatchers, SHARED SERVER MODE 优先。

```
timran11g =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.88)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = SHARED)
      (SERVICE_NAME = timran11g)
    )
  )
```

第三部分: 管理 ORACLE 数据库

第十六章：Oracle ASM 管理(PPT-II:602-636)

16.1 什么是 ASM

自动存储管理即 ASM（Automatic Storage Management)，是 Oracle 提供的一项管理磁盘的功能，它是一种以纯软件方式实现的逻辑卷管理器，功能上提供了基于 Oracle 数据文件的条

带化和可选的镜像。ASM 的知识同时涉及了数据库管理和系统管理两大领域。

16.2 系统级的磁盘管理

物理卷：指的是独立磁盘，或磁盘分区或磁盘 RAID 结构。
逻辑卷：LVM（Logical Volume Manager),以软件方式将虚拟磁盘呈现给用户，LVM 是操作系统级别的。提供了磁盘管理的灵活性，可靠性，具 有条带，镜像和快照等功能。
磁盘阵列上的RAID技术:希望通过多个物理卷增强性能和容错性,常用的有4个级别,RAID0，RAID1，RAID5，RAID0+1。

16.3 ASM 和 LVM 的比较

1）ASM 了解 Oracle 特性，ASM 可以对 ASM 数据文件制定不同的条带(两种)和镜像（三种）策略，这对提高数据库性能来说是一个优势。
2）ASM 增加/删除磁盘后会使磁盘组上的 ASM 文件重新均衡分布，这个过程是自动完成的。
3）ASM 是 Oracle 绑定的，不同的平台上可移植性好。

16.4 ASM 体系结构（以下概念全是考点）

16.4.1 ASM 实例
1）ASM 数据库需要启动 ASM 实例，它和 RDBMS 实例分工协作，ASM 实例负责管理 ASM 磁盘组和 ASM 磁盘以及定位 ASM 文件，注意仅仅是定位，ASM 文件的 IO 读写操作还是由 RDBMS 实例完成的。

2）ASM 实例合法参数很少，如 ASM_DISKTRING 和 ASM_DISKGROUP，这些参数在 RDBMS 实例中是没有的。

16.4.2 ASM 文件
ASM 可以管理的文件有数据文件，临时文件，控制文件，联机日志和归档日志文件，spfile,及 RMAN 备份集和映像副本等。ASM 不支持口令文件、跟踪文件、告警文件和 Oracle 二进制文件（考点）。

16.4.3 ASM 磁盘和磁盘组概念

1）ASM disk 可以是实际的磁盘，也可以是磁盘的某个分区，或 LVM 管理的逻辑卷，实际工作中 ASM disk 一般不使用文件系统格式化的磁盘。
2）ASM diskgroup 由 ASM disk 组成，可以包含一个或多个 ASM disk。
3）建立 ASM diskgroup 时会将所有 ASM disk 划分成单元（AU）。AU 大小缺省 1M。也可指定其属性为 2,4,8,16,32 或 64M。（考点）
4）建立 ASM file 会依据文件的动态增长分配 ASM extent，ASM extent 是(可变长）基于一个或多个 ASM AU ,一个 ASM file 会分布到一个 ASM diskgroup 的所有 ASM disk 上。
5）只能使用 RMAN 备份 ASM file，操作系统命令对 ASM file 是不可知的。
6)可以通过 ALTER DISKGROUP MOUNT 手动加载 ASM disk,也可以通过指定参数 ASM_DISKGROUP 自动加载。

7）当 ASM diskgroup 组中的 ASM disk 发生增减后，ASM diskgroup 会自动再平衡，目的是使 IO 均衡在 ASM diskgroup 的所有 ASM disk 上。

16.4.4 利用磁盘组增加性能和冗余

1）ASM striping（条带）

为了提高 IO 读写性能，ASM 需要进行条带化处理，注意 striping（条带）是基于 AU 的。

2）ASM 不能禁用 striping。有两种类型的条带化方式：粗粒度条带（coarse)和细粒度条带(fine)。粗粒度条带大小=AU 大小，一般来说相对 1M 的 AU，粗粒度条带也是 1M,而细粒度条带总是 128K, 条带策略已由 Oracle 模板缺省定义了（见下图)。若有特殊需求，你可自定义模板，然后在建立表空间时使用模板子句引用你自定义的文件模板。

ASM 文件模板的默认值

Oracle 文件类型	默认系统模板名称	条带化
控制文件	CONTROLFILE	细密
数据文件	DATAFILE	粗糙
日志文件	ONLINELOG	细密
归档日志文件	ARCHIVELOG	粗糙
临时文件	TEMPFILE	粗糙
RMAN 备份文件	BACKUPSET	粗糙
动态参数文件	PARAMETERFILE	粗糙
.....		

16.4.5 ASM 文件(file)、ASM 区（extent)、ASM 单元（unit)、ASM 镜像（mirror)、ASM 条带（stripe）之间的关系:

```
ASM file              ----->spread across all of the disks in a disk group
   /|\
ASM extent(3 种规格）  ----->mirror（3 种规格）
   /|\
ASM au （7 种规格）     ----->stripe（2 种规格）
```

16.5 ASM 的一些重要特性

16.5.1 动态可变分配 ASMextent 的大小

```
ASM file (0--19999) extents extent=AU
ASM file (20000--39999) extents extent=8AU
ASM file (40000+) extents   extent=64AU
```

16.5.2 ASM 镜像及故障组

1）ASM 镜像提供了数据的冗余支持，镜像有三种模式：

NORMAL REDUNDANCY(两路)，
HIGH REDUNDANCY(三路)，
EXTERNAL REDUNDANCY(无镜像,使用 RAID)

2)ASM 的镜像与操作系统提供的镜像有所不同。操作系统镜像是基于整个盘(分区)的镜像。而 ASM 是对 ASM 文件的 extent 做镜像，当 ASM 在某个磁盘上分配了一个主 extent 时,必定在另一个磁盘上分配一个与之对应的副 extent（镜像)。

3）对于双路磁盘控制器，为防止单路控制器同时损毁主副 extent，ASM 又引入故障组容错概念，使主副 extent 分布在不同的故障组下，即主 extent 和副 extent 分别属于不同的故障组。所以，如果不设故障组，则每个磁盘就是一个独立的故障组。

4）一个磁盘组可有两个或多个故障组（failgroup)，一个故障组由一个或多个 ASM 磁盘组成，故障组只能在两路或三路镜像模式下进行。

故障组是在标准冗余的基础上指定镜像策略，它是一种镜像功能的补充。假定有磁盘组 DG1,且创建了两个故障组 controller1,controller2,每个故障组由 2 个 ASM 磁盘组成,则对标准冗余而言,指定两个故障组互为镜像

controller1 --> asmdiskA , asmdiskB
controller2 --> asmdiskC , asmdiskD

就是说：asmdiskA 的 extent 镜像一定会建立在另一个故障组的 asmdiskC 或 asmdiskD 磁盘上，而不会在本故障组 conntroller1 内。

16.5.3 重新平衡磁盘组

ASM 重新平衡磁盘组的操作是自发的，动态的。无论是向磁盘组加盘还是减盘，只要改变了磁盘组中磁盘的状态，ASM 就会重新分布磁盘上的数据。如果是增加磁盘还会为其划分 au,进而划分条带，这样，文件会立即分布到新加的 ASM disk 上。

自动重新平衡会消耗系统资源，有条件的话应该选择系统比较空闲的时段加减磁盘。

ASM_POWER_LIMIT 这个参数值取 1-11（在 11gR2 后可以选的范围是 1-1024)，此值越高平衡时消耗服务器资源越多（考点)。

如果需要也可以使用手动方式对磁盘组进行重新平衡。可以为 power 子句分配 1-11 的值。

SQL> ALTER DISKGROUP dgroup1 REBALANCE POWER 5;

16.5.4 快速镜像再同步特性(PPT-II-635)

一般情况下，如果由于一个磁盘损坏使之不可用，系统会自动将冗余 extent 的复制到另一个磁盘上，即重新创建该盘的 extent 副本，然后删除有故障的磁盘，这样做的缺点是迁移总会消耗一定的系统资源。但是，如果仅仅是电缆或控制器接口或电源等故障（干脆说不是介质损坏）而导致 ASM 磁盘不可用，这样的问题可以快速修复，为此 Oracle 引入 ASM 快速镜像再同步特性。

这个特性的要点是：

1）在给定修复期内，仅使磁盘 offline，不会从磁盘组 drop 磁盘。（考点）。
2）不会复制整个副本，仅重写 offline 之后发生变更的数据(增量数据)。
3）将修复的磁盘 online 后，ASM 不必执行冗长的再平衡。

可以通过设置属性参数 disk_repair_time 指定修复期。
如：alter diskgroup dg1 set attribute 'disk_repair_time'=4h;

可以临时更改修复期
alter diskgroup dg1 offline disks in failuregroup controller1 drop after 2h;
这将意味着忽略了 disk_repair_time 属性，指定 2 小时为修复时间。

修复后重新进行联机即可
alter diskgroup dg1 ONLINE;

超过修复期，将转为常规方式处理，即自动 drop 掉这些磁盘，然后开始复制整个副本并做 REBALANCE。
也可以在修复期内提前使用 FORCE 选项删除这些磁盘：
alter diskgroup dg1 drop disks in failuregroup controller1 FORCE;

16.5.5 ASM 首选镜像读

正常冗余需要两个故障组，高冗余需要三个故障组，ASM 分配新的 extent 时总是定义一个主本和一个副本。ASM 总是自动的读取镜像 extent 的主本。但对于一个远程系统（集群延伸），读取它本地的副本更可以减少网络流量。于是有了首选镜像读特性。

通过设置 ASM_PREFERRED_READ_FAILURE_GROUPS 初始化参数，指定一列首选镜像读故障组的名字。
通过查询 V$ASM_DISK 视图可以确定首选故障组中有哪些磁盘：

例
ASM_PREFERRED_READ_FAILURE_GROUPS= = diskgroup_name.failure_group_name , diskgroup_name1.failure_group_name1...

SQL> alter system set

asm_prefeeed_read_failure_groups=diskgroup_name.failure_group_name,diskgroup_name1.failure_group_name1...;
SQL> select preferred_read from v$asm_disk;

16.5.6 ASM 兼容性参数(PPT-II-631)

为使用一些 ASM 的新功能，你可以对不同的磁盘组设置不同的兼容性属性，

COMPATIBLE.RDBMS	指定能 mount 这个 diskgroup 的 RDBMS 实例的最小兼容版本
COMPATIBLE.ASM	定义了这个 dsikgroup 的 ASM metadata 格式
COMPATIBLE.ADVM	指定该 diskgroup 是否可以包含 ASM volume

请记住：RDBMS 兼容性级别必须小于或等于磁盘组的 ASM 兼容性级别。
即：COMPATIBLE.RDBMS<=COMPATIBLE.ASM。

16.5.7 ASM 的同步集群服务
CSS 是为 claster 服务一组软件，即使不是 RAC 配置，配置一个单实例的 ASM 也要使用 CSS 部分功能，另外还需要了解 ASM 也有后台进程，特别是有两个进程,一个是 RBAL,一个是 ARBn

ASM 实例上的两个重要进程

RBAL:用于协调和管理磁盘组之间的动态平衡
ARBn:用于完成 AU 的移动

RDBMS 实例上两个与 ASM 有关的后台进程：

RBAL:完成 ASM 磁盘打开的任务，其中的磁盘是 ASM 磁盘组的一部分
ASMB:作为一个重要的进程将 RDBMS 实例与 ASM 实例相连起来，对于 RDBMS 实例来说，它是一个 foreground process（考题）

特别注意 RBAL 在 ASM 实例和 RDBMS 实例中各有一个，但功能不同，前者是负责协调 ASM 再平衡活动，后者负责打开和关闭 ASM 磁盘。

```
[oracle@timran timran11g]$ ps -ef |grep rbal
oracle    4790     1  0 14:10 ?        00:00:00 asm_rbal_+ASM
oracle    5021     1  0 14:13 ?        00:00:00 ora_rbal_timran11g
oracle    5133  4112  0 14:22 pts/2    00:00:00 grep rbal
```

16.5.8 关于 ASM 实例和 RDBMS 实例的启动和关闭

1）只能在 nomount 下启动 ASM 实例，它永远不能 mount 控制文件，也不能打开 datafile
2）启动：ASM 实例先启动，RDBMS 实例后启动。退出：RDBMS 实例先 shutdwon immediate，ASM 实例后退出.
3）ASM 实例做 shutdown immediate，而此时 RDBMS 还有用户连接，ASM shutdown 不能成功。

会报错!
4）如果 RDBMS shutdown abort,ASM 实例什么也不做。

16.5.9 ASMCMD 命令行工具

$asmcmd

ASMCMD>help

可以使用命令行工具更灵活的管理 ASM。关注 11g 下在 ASMCMD 中新加入的两个命令，ASM 元数据的备份与还原：即 md_backup 和 md_resotre

md_backup 进行磁盘组的备份

ASMCMD> md_backup -b /tmp/asmbkp1 -g g1
-g 选项指定要备份的单个磁盘组，-b 选项指定文件 asmbkp1 作为备份的文本文件。

ASMCMD> md_recover -b /tmp/asmbkp1 -t full -g g1

考点：md_recover 不能恢复存储在 ASM 磁盘中的实际数据，但是能恢复磁盘组，修改模板，创建目录。当 md_recover 恢复了 ASM 元数据后，才可以使用 RMAN 备份在重建的磁盘组中恢复数据。

16.6 在 Linux 上创建 ASM 实例的范例

1) 在 linux 里增加两个虚盘

ASMDISK1, /dev/sdb，4G, 分 4 个区，每个区 1000M，分别对应 sdb1,sdb2,sdb3,sdb4。
ASMDISK2, /dev/sdc, 2G 分 1 个区，对应 sdc1。

```
[root@timran dev]# fdisk -l

Disk /dev/sda: 21.4 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1        2349    18868311   83  Linux
/dev/sda2            2350        2610     2096482+  82  Linux swap / Solaris

Disk /dev/sdb: 4294 MB, 4294967296 bytes
255 heads, 63 sectors/track, 522 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

```
   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1               1         123      987966   83  Linux
/dev/sdb2             124         246      987997+  83  Linux
/dev/sdb3             247         369      987997+  83  Linux
/dev/sdb4             370         492      987997+  83  Linux

Disk /dev/sdc: 2147 MB, 2147483648 bytes
255 heads, 63 sectors/track, 261 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sdc1               1         261     2096451   83  Linux
[root@timran dev]#
```

2) Oracle ASMlib 安装

ASMlib 是 Oracle 针对 linux 操作系统的 ASM 管理套件，ASMlib 简化磁盘管理，取代原来我们在 linux 上常用 rawdevices 服务。

根据 linux 不同的内核版本，对应有不同的 ASMlib 包，下载地址：http://www.oracle.com/technetwork/server-storage/linux/asmlib/index-101839.html

比如，我的 linux 内核版本是：

```
[root@timran ~]# uname -a
Linux timran.localdomain 2.6.18-348.el5 #1 SMP Wed Nov 28 21:25:39 EST 2012 i686 athlon i386 GNU/Linux
```

根据这个 kernel 版本，我找到的三个对应的 ASMlib 包应该是:

```
[root@timran timran11g]# ll -al *.rpm
-rw-rw-r-- 1 oracle oracle 22751 07-01 13:12 oracleasm-2.6.18-348.el5-2.0.5-1.el5.i686.rpm
-rw-rw-r-- 1 oracle oracle 13929 07-01 13:12 oracleasmlib-2.0.4-1.el5.i386.rpm
-rw-rw-r-- 1 oracle oracle 85303 07-01 13:12 oracleasm-support-2.1.8-1.el5.i386.rpm

[root@timran timran11g]# rpm -qa |grep asm
[root@timran timran11g]#
```

将这三个包加载到 linux kernel 中

```
[root@timran timran11g]# rpm -ivh *.rpm
warning: oracleasm-2.6.18-348.el5-2.0.5-1.el5.i686.rpm: Header V3 DSA signature:
```

```
NOKEY, key ID 1e5e0159
Preparing...                ########################################### [100%]
   1:oracleasm-support      ########################################### [ 33%]
   2:oracleasm-2.6.18-348.el########################################### [ 67%]
   3:oracleasmlib           ########################################### [100%]
```

显示一下已经加载成功

```
[root@timran timran11g]# rpm -qa |grep asm
oracleasmlib-2.0.4-1.el5
oracleasm-support-2.1.8-1.el5
oracleasm-2.6.18-348.el5-2.0.5-1.el5
[root@timran timran11g]#
```

```
[root@timran oracle]# reboot          //重启一下，看看主控台显示 ASM 驱动能否正常加载.
```

ASMlib 包装好后会在/etc/init.d/下出现 oracleasm 这个 service，它可以支持很多 linux 上的 ASM 操作。

3）配置 ASMlib 驱动

```
[root@timran init.d]# /etc/init.d/oracleasm configure
Configuring the Oracle ASM library driver.
This will configure the on-boot properties of the Oracle ASM library
driver.  The following questions will determine whether the driver is
loaded on boot and what permissions it will have.  The current values
will be shown in brackets ('[]').  Hitting <ENTER> without typing an
answer will keep that current value.  Ctrl-C will abort.

Default user to own the driver interface []: oracle
Default group to own the driver interface []: dba
Start Oracle ASM library driver on boot (y/n) [n]: y
Scan for Oracle ASM disks on boot (y/n) [y]:
Writing Oracle ASM library driver configuration: done
Initializing the Oracle ASMLib driver: [  OK  ]
Scanning the system for Oracle ASMLib disks: [  OK  ]
[root@timran ~]#
```

4）创建 ASM 磁盘

```
[root@timran init.d]# /etc/init.d/oracleasm createdisk VOL1 /dev/sdb1
Marking disk "VOL1" as an ASM disk: [  OK  ]
```

```
[root@timran init.d]# /etc/init.d/oracleasm createdisk VOL2 /dev/sdb2
Marking disk "VOL2" as an ASM disk: [  OK  ]
[root@timran init.d]# /etc/init.d/oracleasm createdisk VOL3 /dev/sdb3
Marking disk "VOL3" as an ASM disk: [  OK  ]
[root@timran init.d]# /etc/init.d/oracleasm createdisk VOL4 /dev/sdb4
Marking disk "VOL4" as an ASM disk: [  OK  ]
[root@timran init.d]# /etc/init.d/oracleasm createdisk VOL5 /dev/sdc1
Marking disk "VOL5" as an ASM disk: [  OK  ]
[root@timran init.d]#

[root@timran init.d]# ls /dev/oracleasm/disks
VOL1  VOL2  VOL3  VOL4  VOL5            //可以看到生成了 5 个 ASM 磁盘。

#/etc/init.d/oracleasm querydisk  -d  VOL1      //试一下，查询 ASM 磁盘对应的物理磁盘号
Disk "VOL1" is a valid ASM disk on device /dev/sdb1[8,17]
```

5）创建 ASM 实例

5.1）启用 css 服务(Cluster Synchronization Services)，

单实例的 ASM 也要用到 RAC 套件中的一些驱动，它们负责多实例间的通信。这里用于同步 ASM 实例与 RDBMS 实例

使用 root 帐户进行配置,配置程序位于$ORACLE_HOME/bin

```
[root@timran ~]# /u01/oracle/bin/localconfig add

Successfully accumulated necessary OCR keys.
Creating OCR keys for user 'root', privgrp 'root'..
Operation successful.
Configuration for local CSS has been initialized

Cleaning up Network socket directories
Setting up Network socket directories
Adding to inittab
Startup will be queued to init within 30 seconds.
Checking the status of new Oracle init process...
Expecting the CRS daemons to be up within 600 seconds.
Cluster Synchronization Services is active on these nodes.
        timran
Cluster Synchronization Services is active on all the nodes.
Oracle CSS service is installed and running under init(1M)
[root@timran ~]#
```

```
[root@timran ~]# ps -ef |grep css
oracle    3316     1  1 12:01 ?        00:00:00 /u01/oracle/bin/ocssd.bin
root      3594  2537  0 12:01 pts/0    00:00:00 grep css
```

5.2）创建 ASM 参数文件(使用 VI 或 VIM)

```
[root@timran dbs]# vi /u01/oracle/dbs/init+ASM.ora
#*.asm_diskgroups='DG1'
*.asm_diskstring=''
*.diagnostic_dest='/u01'
*.instance_type='ASM'
*.instance_name='+ASM'
*.large_pool_size=12M
*.remote_login_passwordfile='SHARED'
*.asm_power_limit=1
```

说明一下：

asm_diskgroups='DG1'　这行先注释掉，等到建完磁盘组后再去掉#号。

asm_diskstring=''　这里有个技巧，设成空表示可以搜索任意的 ASM 磁盘。

instance_type='ASM'　这个参数对应 ASM 实例是强制性的，其他都不是。

存盘后改一下属主：

```
[root@timran dbs]# chown oracle:oinstall /u01/oracle/dbs/init+ASM.ora
```

5.3) 创建 ASM 的口令文件

```
[oracle@timran ~]$orapwd file=$ORACLE_HOME/dbs/orapw+ASM password=system
entries=5
```

5.3) 启动 ASM 实例

```
[oracle@timran ~]$export ORACLE_SID=+ASM
[oracle@timran ~]$sqlplus / as sysdba
Connected to an idle instance.

SQL> startup
ASM instance started

Total System Global Area  284565504 bytes
Fixed Size                  1299428 bytes
Variable Size             258100252 bytes
ASM Cache                  25165824 bytes
ORA-15110: no diskgroups mounted           //因为 ASM 磁盘组还没有建立
```

实例启动了，可以在 linux 下看一下+ASM 进程

```
[root@timran dbs]# ps -ef |grep +ASM
oracle    3993     1  0 14:31 ?        00:00:00 asm_pmon_+ASM
oracle    3995     1  0 14:31 ?        00:00:00 asm_vktm_+ASM
oracle    3999     1  0 14:31 ?        00:00:00 asm_diag_+ASM
oracle    4001     1  0 14:31 ?        00:00:00 asm_psp0_+ASM
oracle    4005     1  0 14:31 ?        00:00:00 asm_dia0_+ASM
oracle    4007     1  0 14:31 ?        00:00:00 asm_mman_+ASM
oracle    4009     1  0 14:31 ?        00:00:00 asm_dbw0_+ASM
oracle    4011     1  0 14:31 ?        00:00:00 asm_lgwr_+ASM
oracle    4013     1  0 14:31 ?        00:00:00 asm_ckpt_+ASM
oracle    4015     1  0 14:31 ?        00:00:00 asm_smon_+ASM
oracle    4017     1  0 14:31 ?        00:00:00 asm_rbal_+ASM
oracle    4019     1  0 14:31 ?        00:00:00 asm_gmon_+ASM
oracle        4021     3988     0   14:31   ?                 00:00:00   oracle+ASM
(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))
root      4032  3235  0 14:37 pts/2    00:00:00 grep +ASM
```

6）创建 ASM 磁盘组

建立磁盘组有三个办法，一个是使用 DBCA 工具，另一个是使用 sqlplus 命令，还有一种是使用 EM。

使用 sqlplus 命令：

注意：创建磁盘组要在 ASM 实例下去做，不可以在 RDBMS 实例下完成

创建磁盘组前要改一下这个参数

```
SQL> alter system set asm_diskstring='/dev/oracleasm/disks/VOL*';

SQL>
create diskgroup DG1 normal redundancy
failgroup                              controller1                              disk
'/dev/oracleasm/disks/VOL1','/dev/oracleasm/disks/VOL2'
failgroup                              controller2                              disk
'/dev/oracleasm/disks/VOL3','/dev/oracleasm/disks/VOL4';

col name for a15;
col failgroup for a20;

SQL>   select   NAME,STATE,FREE_MB,REQUIRED_MIRROR_FREE_MB,USABLE_FILE_MB   from
v$asm_diskgroup;
```

```
NAME        STATE           FREE_MB REQUIRED_MIRROR_FREE_MB USABLE_FILE_MB
----------- ----------- ---------- ------------------------ --------------
DG1         MOUNTED            3750                      964           1393


SQL>    select    GROUP_NUMBER,DISK_NUMBER,NAME,FAILGROUP,STATE,TOTAL_MB    from
v$asm_disk;

GROUP_NUMBER  DISK_NUMBER  NAME                                                FAILGROUP
STATE       TOTAL_MB
------------ ----------- ------------------------------ -------------------- ---
----- ----------
            0                                                                      5
NORMAL              0
            1               3 DG1_0003                                        CONTROLLER2
NORMAL            964
            1               2 DG1_0002                                        CONTROLLER2
NORMAL            964
            1               1 DG1_0001                                        CONTROLLER1
NORMAL            964
            1               0 DG1_0000                                        CONTROLLER1
NORMAL            964
```

看一下 ASM 磁盘的情况：

```
SQL> col path for a40;
SQL> select path,os_mb from v$asm_disk order by path;

PATH                                        OS_MB
---------------------------------------- ----------
/dev/oracleasm/disks/VOL1                      964
/dev/oracleasm/disks/VOL2                      964
/dev/oracleasm/disks/VOL3                      964
/dev/oracleasm/disks/VOL4                      964
/dev/oracleasm/disks/VOL5                     2047
```

重新启动前将参数文件的第一行#去掉

```
[root@timran dbs]#vi /u01/oracle/dbs/init+ASM.ora
*.asm_diskgroups='DG1'
```

到此，创建 ASM 的基本内容已经完成了。重启系统(reboot)，检查一下看看磁盘组是否能够

自动 mounted 了。

测试 ASM 是否可以工作

启动了+ASM 实例正常后，再启动 RDBMS 实例，然后建立一个+ASM 上的表空间 testasm 做测试。

```
SQL> exit
[oracle@timran ~]$ export ORACLE_SID=timran11g
[oracle@timran ~]$ sqlplus / as sysdba
Connected to an idle instance.

SQL> startup

SQL> create tablespace testasm datafile '+DG1' size 200m;

Tablespace created.
```

//这时，你可以使用 ps -ef 看看，ASM 实例和 RDBMS 实例中都有了 RABL 后台进程。

```
SQL> select * from v$tablespace;

       TS# NAME                           INC BIG FLA ENC
---------- ------------------------------ --- --- --- ---
         0 SYSTEM                         YES NO  YES
         1 SYSAUX                         YES NO  YES
         2 UNDOTBS1                       YES NO  YES
         4 USERS                          YES NO  YES
         3 TEMP                           NO  NO  YES
         6 EXAMPLE                        YES NO  YES
         7 TESTASM                        YES NO  YES

SQL>select file#,name from v$datafile;

     FILE# NAME
---------- --------------------------------------------------
         1 /u01/oradata/timran11g/system01.dbf
         2 /u01/oradata/timran11g/sysaux01.dbf
         3 /u01/oradata/timran11g/undotbs01.dbf
         4 /u01/oradata/timran11g/users01.dbf
         5 /u01/oradata/timran11g/example01.dbf
         6 +DG1/timran11g/datafile/testasm.256.803235039

create table scott.test(id int) tablespace testasm;
```

```
insert into scott.test values(1);
commit;

select * from scott.test;

        ID
----------
         1
```

OK! 我们进入 OEM，看看是否也可以看到 ASM 磁盘组了，当 ASM 实例和 RDBMS 实例都启动之后，在 OEM 里应该出现管理 ASM 磁盘组的栏目，

OEM-->Server-->Storage-->Disk Group

第一次加载 ASM 有可能看不到 Disk Group 项，可以尝试查询 testasm 表空间-->datafile-->DG1 使其出现。如果还不能出现，说明 EM 的资料库里缺少我们加载的 ASM 信息。重新配置一下 EM，这需要花点时间了。

```
[oracle@timran ~]$ emca -config dbcontrol db
```

过程较长，简单说一下：

a）不需要 shutdown 数据库，也不需要关闭 OEM，配置过程会自动关闭启动 OEM

b）数据库 SID：timran11g
监听端口号：1521
有关 ASM 信息使用缺省值即可
所有管理用户口令：system

c）继续，等待最终结果：“已成功完成 Enterprise Manager 的配置” ok

e) 重新登录 EM，按前面所述再试，使之出现 Disk Group 项

```
[oracle@timran ~]$
```

8）动态增加/删除 ASM 磁盘，观察动态再平衡,你将看到动态再平衡的特点：ASM 文件以 failgroup 为“池”均衡分布在所有 ASM 磁盘上。

8-1）将 VOL5 加入 dg1 组并指定一个故障组

```
SQL> alter diskgroup dg1 ADD FAILGROUP CONTROLLER1 DISK 'ORCL:VOL5';

SQL>    select    GROUP_NUMBER,DISK_NUMBER,NAME,FAILGROUP,STATE,TOTAL_MB    from
```

v$asm_disk;

GROUP_NUMBER	DISK_NUMBER	NAME	FAILGROUP	STATE	TOTAL_MB
1	0	DG1_0000	CONTROLLER1	NORMAL	964
1	1	DG1_0001	CONTROLLER1	NORMAL	964
1	2	DG1_0002	CONTROLLER2	NORMAL	964
1	3	DG1_0003	CONTROLLER2	NORMAL	964
1	4	VOL5	CONTROLLER1	NORMAL	2047

8-2）增扩表空间

```
SQL> alter tablespace testasm add datafile '+DG1' size 5m;
```

注意：

1）动态再平衡的范围是以该 ASM 磁盘所在的故障组中进行的，其他故障组不会做 IO 平衡。
2）如果由于 ASM 文件的增长使空间不够时，可以扩容表空间，而这会扩展所有 ASM 磁盘的使用空间。

9）彻底删除 ASMlib

9.1) 删除表空间

```
SQL> drop tablespace testasm including contents and datafiles;
```

9.2)删除 ASM 组

```
[root@timran ~]# su - oracle
[oracle@timran ~]$ export ORACLE_SID=+ASM
[oracle@timran ~]$ sqlplus / as sysdb

SQL> drop diskgroup dg1 including contents;       //删除 ASM 组要在实例启动时做。

SQL> drop diskgroup dg1 force including contents;  //如果删除失败，再试试这句。
```

9.3）删除 ASM 磁盘

```
SQL>shutdown abort       //关闭 ASM 实例
#/etc/init.d/oracleasm listdisks
VOL1
VOL2
VOL3
VOL4
```

VOL5

#/etc/init.d/oracleasm deletedisk vol1
#/etc/init.d/oracleasm deletedisk vol2
#/etc/init.d/oracleasm deletedisk vol3
#/etc/init.d/oracleasm deletedisk vol4

#/etc/init.d/oracleasm deletedisk vol5

9.4）删除 CSS 服务

#/u01/oracle/bin/localconfig delete　　//删除 css 信息

9.5）删除 ASM 参数文件和口令文件

#rm /u01/oracle/dbs/*ASM*

9.6) 删除 ASMlib 的 RPM 包

[root@timran init.d]# rpm -qa |grep asm　//查询 asm 有关的 rpm 包，
oracleasm-2.6.18-53.el5-2.0.4-1.el5
oracleasm-support-2.1.7-1.el5
oracleasmlib-2.0.4-1.el5

[root@timran init.d]# rpm -e oracleasmlib-2.0.4-1.el5　　//删除这三个 rpm 包有顺序
[root@timran init.d]# rpm -e oracleasm-2.6.18-53.el5-2.0.4-1.el5
[root@timran init.d]# rpm -e oracleasm-support-2.1.8-1.el5
[root@timran init.d]#

[root@timran init.d]# ll /etc/init.d/oracleasm　　//确信 oracleasm 文件删掉了

9.7）物理删除 Vbox 上的 ASM 虚盘
vbox->存储->controller stat->vdi　　//vobx 里删除 ASMDISK1，ASMDISK2。

重启 linux，看到 sdb，sbc 都没有了，再启动 RDBMS，可以正常打开访问。

[root@timran dev]# ll /dev/sd*
brw-r----- 1 root disk 8, 0 Dec 28 17:35 sda
brw-r----- 1 root disk 8, 1 Dec 28 17:35 sda1
brw-r----- 1 root disk 8, 2 Dec 28 17:35 sda2
[root@timran dev]#

9.8 本机安装测试完成后碰到的一个问题：当启动 ASM 实例后再启动 RDBMS 实例时报错：
ORA-00845: MEMORY_TARGET not supported on this system

这个问题是由于设置 SGA 的大小超过了操作系统/dev/shm 的大小：

查看并更改/etc/fstab 文件后，问题解决。

```
[root@timran ~]# df -h /dev/shm
Filesystem            Size  Used Avail Use% Mounted on
tmpfs                 506M  158M  348M  32% /dev/shm

[root@timran ~]#
[root@timran ~]# vi /etc/fstab

LABEL=/                 /                       ext3    defaults        1 1
tmpfs                   /dev/shm        tmpfs   defaults,size=800m      0 0
devpts                  /dev/pts                devpts  gid=5,mode=620  0 0
sysfs                   /sys                    sysfs   defaults        0 0
proc                    /proc                   proc    defaults        0 0
LABEL=SWAP-sda2         swap                    swap    defaults        0 0
```

第十七章：逻辑备份与恢复

17.1 传统的导入导出 exp/imp：

传统的导出导入程序指的是 exp/imp,用于实施数据库的逻辑备份和恢复。

导出程序 exp 将数据库中的对象定义和数据备份到一个操作系统二进制文件中。
导入程序 imp 读取二进制导出文件并将对象和数据载入数据库中

传统的导出导入是基于客户端的。在$ORACLE_HOME/bin 下

导出和导入实用程序的特点有：

1)可以按时间保存表结构和数据
2)允许导出指定的表，并重新导入到新的数据库中
3)可以把数据库迁移到另外一台异构服务器上
4)在两个不同版本的 Oracle 数据库之间传输数据
5)在联机状态下进行备份和恢复
6)可以重新组织表的存储结构，减少链接及磁盘碎片

使用以下三种方法调用导出和导入实用程序：

1，交互提示符：以交互的方式提示用户逐个输入参数的值。
2，命令行参数：在命令行指定执行程序的参数和参数值。
3，参数文件：允许用户将运行参数和参数值存储在参数文件中，以便重复使用参数

导出和导入数据库对象的四种模式是：

1，数据库模式：导出和导入整个数据库中的所有对象
2，表空间模式：导出和导入一个或多个指定的表空间中的所有对象
3，用户模式：导出和导入一个用户模式中的所有对象
4，表模式：导出和导入一个或多个指定的表或表分区

17.1.1 导入导出表

1）scott 导入导出自己的表，一般是从服务器导出到客户端(在 cmd 下操作)

```
create table emp1 as select * from emp;
create table dept1 as select * from dept;
```

```
C:\Documents and Settings\timran>exp scott/scott@timran11g file=d:empdept1.dmp
tables=(emp1,dept1)
```

再导入 server 里

```
SQL> drop table emp1 purge;
SQL> drop table dept1 purge;
```

```
C:\Documents and Settings\timran>imp scott/scott@timran11g file=d:empdept1.dmp
```

sys 导出 scott 表，

SYS 用户可以 exp/imp 其他用户的 object,是因为 SYS 含有 EXP_FULL_DATABASE 和 IMP_FULL_DATABASE 角色。

```
C:\Documents and Settings\timran>exp 'sys/system@timran11g as sysdba'
file=d:sysscott.dmp tables=(scott.emp1,scott.dept1)
```

scott 导入（报错）

```
C:\Documents and Settings\timran>imp scott/scott@timran11g file=d:sysscott.dmp
```

连接到: Oracle Database 10g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP and Data Mining options

经由常规路径由 EXPORT:V10.02.01 创建的导出文件

IMP-00013: 只有 DBA 才能导入由其他 DBA 导出的文件
IMP-00000: 未成功终止导入

C:\>imp 'sys/system@timran11g as sysdba' file=d:sysscott.dmp fromuser=scott

17.1.2 导入导出用户

当前用户 scott 导出自己的所有对象， 注意仅仅导出的是 schema 的 object，也就是说这个导出不包括数据字典中的信息，比如用户账户，及原有的一些系统权限等等。

C:\Documents and Settings\timran>exp scott/scott@timran11g file=d:scott.dmp owner=scott

SQL> drop user scott cascade;
SQL> grant connect,resource to scott identified by scott;

C:\Documents and Settings\timran>imp scott/scott@timran11g file=d:scott.dmp

//如果用 sys 来完成也可以使用如下命令：

C:\Documents and Settings\timran>imp 'sys/system@timran11g as sysdba' file=d:scott.dmp fromuser=scott touser=scott

//sys 用户也可以将导出的 scott 的内容导入给其他用户

C:\Documents and Settings\timran>imp 'sys/system@timran11g as sysdba' file=d:scott.dmp fromuser=scott touser=tim

17.1.3 导入导出表空间

Oracle10g 后，引入了导入导出可传输表空间技术，使表空间的迁移更加快速高效

模拟场景：xp/orcl 到 linux/timran11g(中文字符集) 可传输表空间的导入导出：

1)在 xp/orcl 上建立表空间

sys：
create tablespace tb1 datafile 'd:/mytb1.dbf' size 5m;

scott:
create table 测试 (姓名 char(10),年龄 int) tablespace tb1;
insert into 测试 values('张三',20);
insert into 测试 values('王小二',18);
commit;

2)导出 tb1 表空间,先设为只读;
sys:
alter tablespace tb1 read only;

xp:cmd 下
exp '/ as sysdba' tablespaces=tb1 transport_tablespace=y file=d:\exp_tb1.dmp

3)以 xmanager 把 exp_tb1.dmp 和 MYTB1.DBF 都传输到 linux/timran 里
目录如下：/u01/oradata/timran11g

4)在 linux 的$下执行导入

[oracle@timran ~]$ imp userid=\'/ as sysdba\' tablespaces=tb1 file=/u01/oradata/timran11g/exp_tb1.dmp transport_tablespace=y datafiles=/u01/oradata/timran11g/MYTB1.DBF

5)进入 linux/timran 下验证
sys:
select tablespace_name,status from dba_tablespaces;

6)重设回读写方式
alter tablespace tb1 read write;

7）验证
scott:
select * from tab;
select * from 测试;
select table_name,tablespace_name from user_tables;

说明：

可传输表空间需要满足几个前提条件：

1）原库和目标库字符集要一致，若不一致可以通过转换环境变量 NLS_LANG 解决。

2）字符序有大端（big endian)和小端（little endian)之分， 可以通过
SQL> select * from v$transportable_platform; 查看，如果不一致可以使用 rman 转换

3）compatible 10.0.0.或更高

4）迁移的表空间要自包含 （self contained)，

什么叫自包含：当前表空间中的对象不依赖该表空间之外的对象。

例如：我有 TEST 表空间,里面有个表叫 T1，如果在 T1 上建个索引叫 T1_idx,而这个索引建在 USERS 表空间上，由于 T1_idx 索引是依赖 T1 表的，

那么，TEST 表空间是自包含的，可以迁移，但会甩掉 T1_idx 索引，USERS 表空间不是自包含的，不符合迁移条件，会报错。

检查表空间是否自包含可以使用程序包
SQL> execute dbms_tts.transport_set_check('表空间名');
然后查看看结果
SQL> select * from transport_set_violations;

如上面的例子
SQL> execute dbms_tts.transport_set_check('USERS');
SQL> select * from TRANSPORT_SET_VIOLATIONS;

VIOLATIONS

ORA-39907: 索引 SCOTT.ID_IDX (在表空间 USERS 中) 指向表 SCOTT.T1 (在表空间 TEST 中)。

4）将表空间设置成只读

17.1.4 导出整个数据库的对象

C:\Documents and Settings\timran>exp 'sys/system@timran11g as sysdba' file=d:full.dmp full=y

17.2 数据泵(PPT-I-470-472)

17.2.1 数据泵优点：

1）改进性能（较传统的 exp/imp 速度提高 1-2 个数量级）
2）重启作业能力
3）并行执行能力
4）关联运行作业能力
5）估算空间需求能力
6）操作网络方式

17.2.2 数据泵组成部分：

数据泵核心部分程序包：DBMS_DATAPUMP
提供元数据的程序包：DBMS_MATADATA
命令行客户机（实用程序）：EXPDP,IMPDP

17.2.3 数据泵文件：

转储文件：此文件包含对象数据
日志文件：记录操作信息和结果
SQL 文件： 将导入作业中的 DDL 语句写入 SQLFILE 指定的参数文件中

17.2.4 数据泵的目录及文件位置

以 sys 或 system 用户完成数据泵的导入导出时，可以使用缺省的目录 DATA_PUMP_DIR

```
SQL> select * from dba_directories;
```

如果设置了环境变量 ORACLE_BASE,则缺省目录位置是：
$ORACLE_BASE/admin/database_name/dpdump
否则是：
$ORACLE_HOME/admin/database_name/dpdump

17.3 数据泵的两种操作方式

17.3.1 使用 expdp 和 impdp 客户端实用程序

1)server 端先建好一个存放 MT 表的物理目录

```
[oracle@timran ~]$mkdir -p /u01/oradata/timran11g/dir1
```

2)server 端用 SYS 建立目录对象:

```
SQL> create directory MY_DIR as '/u01/oradata/timran11g/dir1';
```

3)为 scott 授予目录权限

```
SQL> grant read,write on directory MY_DIR to scott;
```

4)导出 scott 的 emp dept 表, 导出过程中在 server 端有 MT 表出现 SYS_EXPORT_TABLE_01，导出完成后 MT 表自动消失

```
C:\Documents and Settings\timran>expdp scott/scott@timran11g directory=MY_DIR dumpfile=expdp_scott1.dmp tables=(emp,dept)
```

看看目录下的导出的文件

```
[oracle@timran dir1]$ ll /
```

```
总计 132
-rw-r----- 1 oracle oinstall 126976 07-30 09:51 expdp_scott1.dmp
-rw-r--r-- 1 oracle oinstall   1387 07-30 09:51 export.log
```

5)导入 expdp_scott1.dmp 实验 先删掉原来的 emp,dept 两个表

```
SQL> conn scott/scott
SQL> drop table emp purge;
SQL> drop table dept purge;

C:\Documents and Settings\timran>impdp scott/scott@timran11g directory=MY_DIR
dumpfile=expdp_scott1.dmp
```

6)导出 scott 的 emp detp 的数据，但不导出结构

```
expdp scott/scott@timran11g directory=MY_DIR dumpfile=expdp_scott1.dmp
tables=(emp,dept) content=data_only reuse_dumpfiles=y
impdp scott/scott@timran11g directory=MY_DIR dumpfile=expdp_scott1.dmp
```

7)导出 scott 用户

```
expdp scott/scott@timran11g directory=MY_DIR dumpfile=expdp1.dmp schemas=scott
//注意与 exp 的区别，schemas 代替了 owner 的写法
impdp scott/scott@timran11g directory=MY_DIR dumpfile=expdp1.dmp
```

17.3.2 使用 OEM 操作数据泵

OEM 数据泵操作是以 job 形式设计的。其中：

```
Export to Export Files      定义数据泵导出作业
Import from Export Files    定义数据泵导入作业
Import from Database        定义数据泵网络模式导入
Monitor Exoirt and Import Jobs  连接到正在运行的作业，可以观察暂停重启它们
```

例：使用 OEM 导出/导入 scott 的 emp1 表,
expdp 导出部分步骤如下：

1）使用 system 登录 OEM-->Data Movement-->Move Row Data-->Export to Export Files 链接

2）选 Table 按钮，在下方输入主机的操作系统用户名和口令（我这里两项是都是 oracle)，然后 Continue

3）在 Export:Table 窗口中，单击 Add,并 schema 下输入 scott,table 下输入 emp1，然后 go

按钮查找

4) 在 Export:Options 窗口中，下方 Optional File 选择目录 Directory Object 下拉菜单选择 DATA_PUMP_DIR

5）在 Export:File 窗户中，选择继续选择 DATA_PUMP_DIR 目录，DMP 文件名默认。

6）在 Export:Schedule 窗口中，给出 job 名（否则 oracle 会命名），并单击 Next 使之立即运行此 job

7）在 Review 窗口中，单击 Submit job 按钮

8) 作业完成后查看 /u01/admin/timran11g/dpdump 下的文件,至此数据泵导出结束

```
[oracle@timran dpdump]$ ll
-rw-rw---- 1 oracle oracle 94208 06-26 13:33 EXPDAT01.DMP
-rw-rw-r-- 1 oracle oracle   772 06-26 13:33 EXPDAT.LOG
```

SQL> drop table emp1 purge; 删除 scott 下的 emp1 文件，准备再导入

impdp 导入部分步骤如下：

9）使用 system 登录 OEM-->Data Movement-->Move Row Data-->Import From Export Files 链接

10）在 Import:Files 窗口中，选择目录（DATA_PUMP_DIR),选择 Table 对应选项，这里还是 emp1(当然你可以还原为另一个名字）

11) 显示 Import Read Succeeded 说明 impdp 已经找到了要导入的.DMP 文件

12）在 Import:Tables 窗口中，单击 Add，搜索并选中 scott.emp1，单击 Continue 和 Next 按钮

13）然后完成从 Import:Re-Mapping-->Schedule-->Review 全过程，每一步骤有 Next 或 Submit 按钮。

至此，已完成数据泵导入作业，查看 scott 下是否有了 emp1 表。

第十八章 物化视图

18.1、物化视图作用

1) 物化视图起源于数据仓库，早期的考虑是用于预先计算并保存表连接或聚集等耗时较多

的操作的结果，这样，在执行查询时，就可以避免在基表上进行这些耗时的操作，从而快速的得到结果。

2) 物化视图和表一样可以直接进行查询。物化视图还用于复制、移动计算等方面。

18.2 物化视图创建时的权限

如果创建基于主键的物化视图,则必须具有访问主表、访问主表的日志、create MATERIALIZED VIEW 这三个权限。
如果创建基于 rowid 的物化视图，则必须具有访问主表、create MATERIALIZED VIEW 这两个权限。

18.3 创建物化视图语法

```
create materialized view [view_name]
refresh [fast|complete|force]
[
on [commit|demand] |
start with (start_time) next (next_time)
]
as
{创建物化视图用的查询语句}
```

18.4 物化视图创建时的选项

1) 查询重写（Query Rewrite）：查询重写是指当对物化视图的基表进行查询时，Oracle 会自动判断能否通过查询物化视图来得到结果。默认为 DISABLE QUERY REWRITE。

2) 物化视图日志：在基于主键的物化视图上，需要进行快速刷新时，需要建立物化视图日志。

3) 刷新（Refresh）：指当基表发生了 DML 操作后，物化视图何时采用哪种方式和基表进行同步。刷新的模式有两种：ON DEMAND 和 ON COMMIT。ON DEMAND 指物化视图在用户需要的时候进行刷新。刷新的方法有四种：FAST、COMPLETE、FORCE 和 NEVER。FAST 刷新采用增量刷新，只刷新自上次刷新以后进行的修改。COMPLETE 刷新对整个物化视图进行完全的刷新。如果选择 FORCE 方式，则 Oracle 在刷新时会去判断是否可以则采用 FAST 方式，否则采用 COMPLETE 的方式。NEVER 指物化视图不进行任何刷新。默认值是 FORCE ON DEMAND。

18.5 示例

测试环境：生产库（linux)+备份库（xp)

1)建立 link 连接, database link 是远程连接的基础，也是 oracle 分布式数据库技术的组成部分。

备用库：

C:\Documents and Settings\timran>sqlplus sys/system@orcl as sysdba

sys:

SQL> create public database link my_link connect to scott identified by scott using 'timran11g';

//如果以前建立过，提示重名的话可以使用下面语句删掉 SQL> drop public database link my_link;

若想查看所有的数据库链接，进入系统管理员 SQL>操作符下，运行命令：

SQL>select owner,object_name from dba_objects where object_type='DATABASE LINK';

OWNER	OBJECT_NAME
PUBLIC	MY_LINK.REGRESS.RDBMS.DEV.US.ORACLE.COM

2）在备份库上测试与生产库（启动监听）的访问连接是否成功。

scott:
SQL> select * from tab; //看备份库（自己）

TNAME	TABTYPE	CLUSTERID
BONUS	TABLE	
DEPT	TABLE	
EMP	TABLE	
SALGRADE	TABLE	
测试	TABLE	

SQL> select * from tab@my_link; //看生产库（别人）

TNAME	TABTYPE	CLUSTERID
BONUS	TABLE	
DEPT	TABLE	
EMP	TABLE	
SALGRADE	TABLE	

范例 1 基于主键的物化视图

生产库：(远端 linux)

scott:
SQL>create table test(id int primary key,name char(10));
SQL>create materialized view log on test;

SQL> select * from tab;

TNAME	TABTYPE	CLUSTERID
BONUS	TABLE	
DEPT	TABLE	
EMP	TABLE	
MLOG$_TEST	TABLE	
RUPD$_TEST	TABLE	
SALGRADE	TABLE	
TEST	TABLE	

备份数据库（本地 xp)
sys:
SQL>grant create materialized view to scott;

scott:
SQL>create materialized view test_view refresh
fast
start with sysdate
next sysdate+1/2880
with primary key
as select * from scott.test@my_link;

//1440 分钟是 24 小时，1/1440 是 1 分钟，1/2880 是 30 秒。

SQL> select * from tab;

TNAME	TABTYPE	CLUSTERID
BONUS	TABLE	
DEPT	TABLE	
EMP	TABLE	
SALGRADE	TABLE	
TEST_VIEW	TABLE	
测试	TABLE	

生产库 scott:
SQL>insert into test values(1,'sina');
SQL>commit;

备份库 scott:
SQL> select * from test_view;

未选定行

---30 秒以后...
SQL> select * from test_view;

```
        ID NAME
---------- ----------
         1 sina
```

生产库 scott:
SQL> insert into test values(2,'sohu');
SQL> commit;

备份库 scott:

SQL> select * from test_view;

```
        ID NAME
---------- ----------
         1 sina
         2 sohu
```

范例 2 基于 rowid 的物化视图

生产库 scott:
SQL>drop table test purge;

SQL>create table emp1 as select * from emp;

备份库 scott:
SQL>drop materialized view test_view;

```
SQL>create materialized view emp1_view refresh with rowid
start with sysdate
next sysdate+1/2880
as select empno,ename,sal,deptno from scott.emp1@my_link where deptno=10;
```

备份库 scott:

```
SQL> select * from emp1_view;

     EMPNO ENAME              SAL     DEPTNO
---------- ---------- ---------- ----------
      7782 CLARK            2450         10
      7839 KING             5000         10
      7934 MILLER           1300         10
```

生产库 scott:

```
SQL> update emp1 set sal=1000 where empno=7839;
SQL> commit;
```

备份库 scott:

```
SQL> select * from emp1_view;

     EMPNO ENAME              SAL     DEPTNO
---------- ---------- ---------- ----------
      7782 CLARK            2450         10
      7839 KING             1000         10
      7934 MILLER           1300         10
```

--完--