053：数据库高级管理：

第一部分：数据库备份与恢复

第一章： 备份恢复概述
第二章： 手工备份
第三章： 手工完全恢复
第四章： 手工不完全恢复(略)
第五章： 闪回 flashback
第六章： RMAN 概述
第七章： RMAN 备份
第八章： RMAN 完全恢复
第九章： RMAN 不完全恢复
第十章： 目录库和辅助库
第十一章：RMAN 维护

第二部分：优化管理 Oracle 数据库

第十二章：Oracle 资源管理
第十三章：Oracle 任务调度
第十四章：AWR 存储库及顾问框架
第十五章：预警及诊断系统
第十六章：Oracle 性能调优

附录： Oracle 一周备份计划范例（基于 Linux 的 RMAN CATALOG 备份策略（选））

第一章：备份恢复概述

1.1 备份的意义:

1）保护数据，避免因为各种故障而丢失数据
2）MTBF:平均故障间隔时间
3）MTTR:平均恢复时间

1.2 数据库故障的类型：

1）user process failure： pmon 自动处理
2）instance failure: smon 自动处理
3）user errors ： 需要 dba 通过备份恢复解决
4）media failure： 必须通过备份和日志恢复

1.3 制定你的备份和恢复的计划

1）根据生产环境的恢复周期，制定详细的备份计划，然后严格执行
2）对备份，要在一定的时间内利用测试环境，进行故障恢复的练习

1.4 备份恢复分类

1.4.1 逻辑备份与恢复-- 面向 object

1）传统的导入导出：exp/imp：
2）数据泵导入导出：expdp/impdp

//逻辑备份就是热备数据库对象某一时刻状态，不能运用在 media failure 上，逻辑备份的恢复就是还原备份，没有 recover 的概念。

1.4.2 物理备份与恢复-- 面向 media failure

1）手工备份与恢复， 也叫用户管理的备份与恢复，通过 OS 的命令，完成备份与还原，然后再运用日志进行恢复。
2）自动备份与恢复， 利用 oracle 的备份恢复工具 rman （或其他备份恢复软件），还原与恢复过程自动完成，可以备份恢复 ASM FILE。
//物理备份从方式上可以有一致性备份（冷备）和非一致性备份（热备）
//完整的备份策略应该以物理备份为主，逻辑备份为辅（用于备份一些重要的表）

1.4.3 闪回技术-- 面向人为的逻辑错误

一种利用 undo 数据或闪回日志的快速恢复技术。可以针对不同层面问题进行逻辑恢复，11g 支持七种 flashback 方式。

1.5 完全恢复与不完全恢复

media failure 后，需要运用日志进行 recover。

1）完全恢复：利用完整备份或部分备份，可以将 datafile 恢复到 failure 前得最后一次 commit，不会出现数据丢失。

2）不完全恢复：需要运用完整备份和日志将 database 恢复到过去的某个时间点（或 SCN），有数据丢失。

1.6 归档与非归档

归档模式：redo log 写入 archive log

非归档模式：没有 archive log，redo log 循环覆盖

	手工冷备	手工热备	RMAN 冷备	RMAN 热备	完整还原	完全恢复	不完全恢复
归档模式:	yes	yes	yes	yes	yes	yes	yes
非归档模式:	yes	no	yes	no	yes	no	no

*考点：
1）非归档模式运行时必须备份那些文件：控制文件和整个数据文件集，备份前必须干净的关闭数据库（冷备）
2）当处于非归档模式下时，在丢失数据文件后唯一的选择是执行完整的数据库还原，而不能进行恢复。
3）非归档模式下,RMAN 只能做冷备。

第二章：手工备份与恢复

2.1 手工备份和恢复的命令

1）备份和还原都使用 OS 命令，如 linux 中的 cp
2）恢复用 sqlplus 命令:recover

2.2 备份前要对数据库进行检查：有关的视图：v$datafile\v$datafile_header\v$controlfile\v$logfile\dba_tablespaces\dba_data_files

1）检查需要备份的数据文件

```
SQL> select name from v$datafile;
SQL> select file_id,file_name,tablespace_name from dba_data_files;
```

2）检查要备份的控制文件

```
SQL> select name from v$controlfile;
```

3）在线 redo 日志不需要做备份

2.3 手工非一致性备份（热备份）的执行方式及热备份的监控（v$backup）

注意：对只读的表空间不能做热备份，临时表空间不需要备份，特别强调：NOARCHIVE 模式下不支持手工热备（考点）。

1）在备份前要进入 backup mode(backup 模式)，
即：执行 begin backup （在数据文件上生成检查点，写入 scn ，将来恢复的时候以此

scn 为起点）

```
SQL> alter database begin backup;       //对整个库做热备份
SQL> alter database end backup;

SQL> alter tablespace users begin backup;   //对表空间做备份
SQL> alter tablespace users end backup;
```

2）备份期间利用 v$backup 监控

例；

```
SQL> alter tablespace test begin backup;

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            2414314
         2            2414314
         3            2414314
         4            2414314
         5            2414314
         6            2430480          //在备份期间 ， scn 被冻结，不发生变化。
         7            2414314

SQL> select * from v$backup;

     FILE# STATUS               CHANGE# TIME
---------- ------------------ ---------- -------------------
         1 NOT ACTIVE                  0
         2 NOT ACTIVE                  0
         3 NOT ACTIVE                  0
         4 NOT ACTIVE                  0
         5 NOT ACTIVE                  0
         6 ACTIVE                2430480 2012-07-30 11:07:19
         7 NOT ACTIVE                  0
```

STATUS 是 ACTIVE，表示可以备份相应的数据文件。并且对于其中的数据块 DBWN 仍然可以刷新。

```
$cp test01.dbf test01.bak
```

备份完毕，执行 end backup

```
SQL> alter tablespace test end backup;

SQL> select * from v$backup;

     FILE# STATUS                CHANGE# TIME
---------- ------------------ ---------- -------------------
         1 NOT ACTIVE                  0
         2 NOT ACTIVE                  0
         3 NOT ACTIVE                  0
         4 NOT ACTIVE                  0
         5 NOT ACTIVE                  0
         6 NOT ACTIVE            2430480 2012-07-30 11:07:19
         7 NOT ACTIVE                  0
```

如果在 end backup 之前发生数据库 abort,那么可以在下次启动到 mount 时 end backup，从而完成实例恢复。

关于 split block

我们知道一个 Oracle block 一般包含多个 OS block,，当手工热备时，OS 的 cp 单位不是 Oracle block 而是 OS block,而 Oracle 的 DBWR 又可能不时的从内存中刷新 Oracle block(脏块)到磁盘上，如此，OS 级的拷贝便可能造成：一个 Oracle Block 是由不同的版本组成，比如未被 DBWR 刷新 Header block 加上另一部分被刷新的 foot block，这样 cp 出来的 Oracle blcok 就是 split block。

数据库的一致性是不允许 oracle block 是 split 的，Oracle 采取的办法是：在 backup mode 后，如果发现首次 DBWR 要写脏块，则将该块被刷新之前的镜像数据记录到 redo buffer，这样，虽然 cp 后的文件里仍然含有 split block,而当需要恢复时，日志会前滚该块的前镜像，以保证所有被恢复的 oracle block 是一个完整的版本。

这就是我们常常发现在热备时日志文件会急剧增大的原因。

*考点：手工热备不能备份临时表空间，甚至不能将它们置于备份模式。

2.4 dbv 检查数据文件是否有坏块

在手工备份前，应该检查 datafile 是否有坏块，备份完后对备份也做检查

```
[oracle@timran admin]$ dbv       //相当于 help 信息

DBVERIFY: Release 11.1.0.6.0 - Production on Mon Jul 30 11:11:07 2012

Copyright (c) 1982, 2007, Oracle.  All rights reserved.
```

```
Keyword     Description                    (Default)
----------------------------------------------------
FILE        File to Verify                 (NONE)
START       Start Block                    (First Block of File)
END         End Block                      (Last Block of File)
BLOCKSIZE   Logical Block Size             (8192)
LOGFILE     Output Log                     (NONE)
FEEDBACK    Display Progress               (0)
PARFILE     Parameter File                 (NONE)
USERID      Username/Password              (NONE)
SEGMENT_ID  Segment ID (tsn.relfile.block) (NONE)
HIGH_SCN    Highest Block SCN To Verify    (NONE)
            (scn_wrap.scn_base OR scn)
[oracle@timran admin]$
```

对某个 datafile 做坏块检查

```
[oracle@timran admin]$ dbv file=/u01/oradata/timran11g/users01.dbf feedback=50

DBVERIFY - 开始验证: FILE = /u01/oradata/timran11g/users01.dbf
.......

DBVERIFY - 验证完成

检查的页总数: 640
处理的页总数 (数据): 107
失败的页总数 (数据): 0
处理的页总数 (索引): 36
失败的页总数 (索引): 0
处理的页总数 (其它): 478
处理的总页数 (段)  : 0
失败的总页数 (段)  : 0
空的页总数: 19
标记为损坏的总页数: 0
流入的页总数: 0
加密的总页数         : 0
最高块 SCN             : 695469 (0.695469)
```

第三章：手工完全恢复

3.1 完全恢复；通过备份、归档日志、current redo ，将 database 恢复到 failure 前的最后一次 commit 状态。

3.2 完全恢复的步骤

1）restore: OS 拷贝命令还原所有或部分 datafile
2）recover：SQL*PLUS 利用归档日志和当前的 redo 日志做恢复

3.3 手工完全恢复可以基于三个级别

recover database： 所有或大部分 datafile 丢失，一般是在 mount 状态完成
recover tablespace：非关键表空间损坏，表空间下某些数据文件不能访问，一般是在 open 下完成
recover datafile： 单一或少数数据文件损坏，可以在 mount 或 open 状态完成

什么是关键文件：如果关键文件损坏，数据库将不能维持在 open 状态，或崩溃或死机！

考点：哪些文件是关键文件：四个：1）system01.dbf, 2) undo tablespace, 3）control file 4）current log file

3.4 恢复过程可以查看的视图：

1）v$recover_file： 查看需要恢复的 datafile
2）v$recovery_log： 查看 recover 需要的 redo 日志
3）v$archvied_log： 查看已经归档的日志

3.5 非归档模式下只能做全备的完整还原，还原后仍会缺失联机日志文件（因为只备数据文件和控制文件），需要在 mount 下发出
alter database clear logfile group <组号>. （如果是在 RMAN 下还原，这一过程是自动完成的）。

3.6 归档模式下手工完全恢复的实验

前提: 1）有一套 datafile 全备, 2）使用当前控制文件, 3）自上次备份以来的归档日志和当前联机日志是完整的,

范例 1：recover database

说明：由于 media failure 所有 datafile 丢失,

```
sys:
SQL> select * from scott.test;

        ID
----------
```

1

在这个状态下先在 OS 下做一个数据文件和控制文件的冷备。

```
SQL> shutdown immediate

[oracle@timran ~] $cp /u01/oradata/timran11g/*.dbf  /u01/back1
[oracle@timran ~] $cp /u01/oradata/timran11g/*.ctl  /u01/back1

[oracle@timran ~] $startup

SQL> insert into scott.test values(2);
SQL> commit;
SQL> select * from scott.test;

        ID
----------
         2
         1

SQL> insert into scott.test values(3);
SQL> commit;
SQL> select * from scott.test;

        ID
----------
         2
         3
         1
```

1)模拟介质失败，所有数据文件丢失

```
[oracle@timran ~]$ rm /u01/oradata/timran11g/*.dbf      //数据库在打开的情况下就删掉了所有 dbf 文件
SQL> shutdown abort                  //数据库直接 abort 了
ORACLE instance shut down.
```

2)启动 database

```
SQL> startup
ORACLE instance started.

ORA-01157: cannot identify/lock data file 1 - see DBWR trace file
ORA-01110: data file 1: '/u01/oradata/timran11g/system01.dbf'
```

```
SQL>select file#,error from v$recover_file;

     FILE# ERROR
---------- -----------------------------------------------------------------
         1 FILE NOT FOUND
         2 FILE NOT FOUND
         3 FILE NOT FOUND
         4 FILE NOT FOUND
         5 FILE NOT FOUND
         6 FILE NOT FOUND
         7 FILE NOT FOUND

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            2436025
         2            2436025
         3            2436025
         4            2436025
         5            2436025
         6            2436025
         7            2436025

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            2414314
         2            2414314
         3            2414314
         4            2414314
         5            2414314
         6            2430480
         7            2414314
```

3)启动失败，控制文件和数据文件头的 scn 不一致，需要做介质恢复。
注意：
//此例是数据库 open 状态在 OS 下直接删除了所有数据文件，所以这里最好不要做手工日志切换，否则情况复杂。
//控制文件记录的 scn 应大于需恢复的数据文件头部的 scn，归档日志+当前日志都完整，这样才能满足完全恢复的条件。

首先还原所有数据文件
[oracle@timran ~]$ cp /u01/back1/*.dbf /u01/oradata/timran11g

4)恢复 database

SQL> recover database;

5)打开数据库

SQL> alter database open;

6）验证

SQL> select * from scott.test;

```
        ID
----------
         2
         3
         1
```

范例 2：recover tablespace（状态：database open)

说明：针对的是非关键表空间的损坏，基于表空间的完全恢复实际上还是对其下的 datafile 的恢复

模拟这种情形非常实用，通常是某个非关键表空间下的数据文件受损，但并没有造成 Oracle 崩溃， 我们只需针对个别有问题的 tablespace 去做单独的在线恢复操作，也就是说恢复时数据库整体是 online 的,而局部表空间是 offline 的，数据库不需要 shutdown。

恢复表空间（删除了 tablespace 下的所有的 datafile）

1)了解一下当前状态，有个 test 表空间，

SQL> select file_id,file_name,tablespace_name from dba_data_files;

```
   FILE_ID FILE_NAME                                          TABLESPACE_NAME
---------- -------------------------------------------------- ------------------
------------
         4 /u01/oradata/timran11g/users01.dbf                 USERS
         3 /u01/oradata/timran11g/sysaux01.dbf                SYSAUX
         2 /u01/oradata/timran11g/undotbs01.dbf               UNDOTBS1
         1 /u01/oradata/timran11g/system01.dbf                SYSTEM
         5 /u01/oradata/timran11g/example01.dbf               EXAMPLE
```

```
         7 /u01/oradata/timran11g/abcd01.dbf                    ABCD
         6 /u01/oradata/timran11g/test01.dbf                    TEST

SQL> conn scott/scott
Connected.
SQL> create table t1 (name char(10)) tablespace test;
SQL> insert into t1 values('a');
SQL> commit;
SQL> select * from t1;

NAME
----------------------------------------------------
a
```

2）模拟表空间损坏，数据库 open 下，直接删除表空间下的数据文件

```
[oracle@timran ~]$ rm /u01/oradata/timran11g/test01.dbf
[oracle@timran ~]$
```

3）查证该表空间上的表不可访问了

```
SQL> alter system flush buffer_cache;   //清除 data buffer

SQL> conn / as sysdba            //换个 session 登陆，访问 t1 表，因内存里已清除了
buffer 块，只好去做物理读，所以报错！
Connected.

SQL> select * from scott.t1;
select * from scott.t1
*
ERROR at line 1:
ORA-01116: error in opening database file 6
ORA-01110: data file 6: '/u01/oradata/timran11g/test01.dbf'
ORA-27041: unable to open file
Linux Error: 2: No such file or directory
Additional information: 3
```

4）看看 scn 的情况

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            3550907
         2            3550907
```

```
         3            3550907
         4            3550907
         5            3550907
         6            3550339

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            3550907
         2            3550907
         3            3550907
         4            3550907
         5            3550907
         6                  0
```

5）表空间 offline

```
SQL> alter tablespace test offline immediate;       //immediate 使表空间能立即脱机，不等 Oracle 对任何数据文件做检查
```

6）数据库 open 下，使用备份还原这个表空间下的所有数据文件

```
[oracle@timran ~]$ cp /u01/back1/test01.dbf /u01/oradata/timran11g
[oracle@timran ~]$
```

7）恢复 tablespace

```
SQL> recover tablespace test;
完成介质恢复。
```

8）使表空间 online

```
SQL> alter tablespace test online;          //注意：此时数据库状态一直是 open 的。
```

9) 验证

```
SQL> select * from scott.t1;

NAME
--------------------------------------------------
a
```

范例 3：（recover datafile，database mount 或 open 状态）

恢复 datafile，同范例 2 不同的是模拟 UNDO 文件损坏：因 UNDO 数据文件也是关键文件，所以只能在 mount 状态下恢复。

1） 模拟环境：

```
SQL> insert into scott.t1 values('b');              //插入一行记录是为了使 t1 和备份有区别
SQL> commit;
SQL> select * from scott.t1;

NAME
--------------------------------------------------
a
b

SQL> delete scott.t1;              //注意：删掉了 t1 并没有提交，老值在 UNDO 里。
```

2）在 open 状态下删除 datafile

```
[oracle@timran ~]$ rm /u01/oradata/timran11g/undotbs01.dbf
[oracle@timran ~]$
```

3）关闭数据库

```
SQL> shtudown abort          //abort 埋下伏笔，等到完全恢复时会做 UNDO 回滚。
```

4) 启动数据库 mount

```
SQL> startup mount
...
数据库装载完毕。
ORA-01157: 无法标识/锁定数据文件 3 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/undotbs01.dbf'
```

5）还原并恢复 UNDO 数据文件

```
[oracle@timran timran11g]$ cp /u01/back1/undotbs01.dbf ./

SQL> recover datafile 3;
完成介质恢复。
```

6）打开数据库（会完成 UNDO 表空间数据的回滚）

```
SQL> alter database open;
```

数据库已更改

7) 验证
SQL> select * from scott.t1;

NAME
----
a
b

3.7 手工完全恢复特点小结：

3.7.1 recover database （全部或大部分数据文件损坏，mount 下进行）

OS: 使用 cp 还原受损的 dbf（不一定是全部，v$recover_file 记录的都需要还原）
SQLPLUS：
1)recover database;
2)alter database open;

3.7.2 recover tablespace （针对表空间的非关键数据文件损坏，一般是 open 下进行）

OS:使用 cp 还原该表空间 XXX 下的所有数据文件
SQLPLUS：
1)alter tablespace XXX offline;
2)recover tablespace XXX;
3)alter tablespace XXX online;

3.6.3 recover datafile （单个或几个数据文件损坏，关键文件在 mount 下进行，非关键文件在 open 下进行）

OS:cp 还原相关的关键数据文件(mount)
SQLPLUS：
1)recover datafile 6,8;
2)alter database open;

OS:cp 还原相关的非关键数据文件(open)
SQLPLUS：
1)alter database datafile 6,8 offline;
2)recover datafile 6,8;
3)alter database datafile 6,8 online;

第四章：手工不完全恢复

4.1 不完全恢复的特点：

1）让整个 database 回到过去某个时间点，不能避免数据丢失。

2）想跳过坏日志而继续恢复所有其他工作是不可能的，前滚没有这个功能(考点)。

3）必须以 sysdba 身份连接进行不完全恢复，普通用户或 sysoper 都不行（考点）。

4）语句只有 recover database until 这种形式，表示整个数据库回到某个时间点或 SCN，而 until 是指恢复在时间点前停止（考点）。

4.2 不完全恢复（Incomplete recover） 适用环境：

1）在过去的某个时间点重要的数据被破坏。

2）在做完全恢复时，丢失了归档日志或当前 online redo log（考点）

3）当误删除了表空间时（有控制文件备份）

4）丢失了所有的控制文件，使用备份的控制文件恢复时 （条件满足时可以完全恢复）

4.3 不完全恢复的基本类型：

1）基于时间点 （until time)： 使整个数据库恢复到过去的一个时间点前

2）基于 scn （until change)： 使整个数据库恢复到过去的某个 SCN 前

2）基于 cancel (until cancel)： 使整个数据库恢复到归档日志或当前日志的断点前

3）基于误删除表空间（使用备份的 controlfile)： 使整个数据库恢复到误删除表空间前

4.4 传统的不完全恢复的操作步骤：（效率低）

1）先通过 logmnr 找到误操作的时间点

2）对现在的 database 做新全备（所有的数据文件和当前控制文件）

3）还原该时间点前所有的 datafile

4）以当前控制文件进入 mount 状态，对 database 做 recover，恢复到误操作时间点

5）将恢复出来的 table 做逻辑备份（exp）

6）再将新全备还原

7）将导出的表导入 database（imp）

4.5、logminer 工具的使用

对 redo log 进行挖掘，找出在某个时间点所作的 DDL 或 DML 误操作（包括：时间点、scn 、sql 语句）

4.6 不完全恢复范例：

范例 1:

恢复过去某个时间点误操作的 table

4.6.1 基于时间点的不完全恢复

1）环境：scott 用户在 test 表空间下有个 t1 表

```
SQL> conn scott/scott
SQL> create table t1(id int) tablespace test;
SQL> insert into t1 values(1);
SQL> insert into t1 values(2);
SQL> insert into t1 values(3);
SQL> commit;
SQL> select * from t1;

        ID
----------
         1
         2
         3
```

2）误删除了 t1 表，并 purge 了。

```
SQL> drop table t1 purge;
SQL> select * from v$log;

    GROUP#       THREAD#    SEQUENCE#          BYTES       MEMBERS  ARCHIVED  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- -------- ----------------
------------- -----------
         1            1          131    52428800            1  YES        INACTIVE
1875893 2012-6-13 1
         2            1          132    52428800            1  YES        INACTIVE
1896385 2012-6-13 1
         3            1          133    52428800            1  NO         CURRENT
1916973 2012-7-18 1

SQL> alter system switch logfile;
SQL> /
SQL> /

SQL> select name from v$archived_log;

NAME
--------------------------------------------------------------------------------
...
```

/u01/disk1/timran/arch_1_782662700_129.log
/u01/disk1/timran/arch_1_782662700_130.log
/u01/disk1/timran/arch_1_782662700_131.log
/u01/disk1/timran/arch_1_782662700_132.log
/u01/disk1/timran/arch_1_782662700_133.log //drop table t1 purge 这个动作的日志条目记录在此归档日志里了。
/u01/disk1/timran/arch_1_782662700_134.log
/u01/disk1/timran/arch_1_782662700_135.log

116 rows selected

3）通过 logmr 找出误操作的 ddl 命令的 timestamp 或 san

SQL> show parameter utl

NAME TYPE VALUE
------------------------------------ ----------- ------------------------------
create_stored_outlines string
utl_file_dir string /home/oracle/logmnr

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

SQL> execute dbms_logmnr.add_logfile(logfilename=>'/u01/disk1/timran/arch_1_782662700_133.log',options=>dbms_logmnr.new);

SQL> execute dbms_logmnr.add_logfile(logfilename=>'/u01/disk1/timran/arch_1_782662700_134.log',options=>dbms_logmnr.addfile);

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

SQL> select username,scn,to_char(timestamp,'yyyy-mm-dd hh24:mi:ss'),sql_redo from v$logmnr_contents WHERE lower(sql_redo) like 'drop table%';

USERNAME SCN TO_CHAR(TIMESTAMP,'YYYY-MM-DDH
SQL_REDO
------------------------------ ---------- ------------------------------ -------
----

SCOTT 1917250 2012-07-18 16:44:55 drop

```
table test purge;
SCOTT                          1917267 2012-07-18 16:45:01          drop
table student purge;
SCOTT                          1918000 2012-08-01 17:28:29          drop
table t1 purge;

SQL> execute dbms_logmnr.end_logmnr;
```

4) 关闭数据库，删除所有 dbf，准备做不完全恢复

```
SQL> shutdown abort

[oracle@timran ~]$ cd /u01/oradata/timran11g
[oracle@timran ~]$ rm *.dbf
```

5）还原所有备份的数据文件

```
[oracle@timran ~]$ cp /u01/back1/*.dbf ./
```

6）根据 log miner 提供的信息，做基于时间点的不完全恢复

```
17:31:43 SQL> startup
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218968 bytes
Variable Size              75499112 bytes
Database Buffers          201326592 bytes
Redo Buffers                7168000 bytes
Database mounted.
ORA-01113: file 1 needs media recovery
ORA-01110: data file 1: '/u01/oradata/timran11g/system01.dbf'

17:33:07 SQL> recover database until time '2012-08-01 17:28:29';

ORA-00279: change 1917581 generated at 07/18/2012 16:46:34 needed for thread 1
ORA-00289: suggestion : /u01/disk1/timran/arch_1_782662700_133.log
ORA-00280: change 1917581 for thread 1 is in sequence #133

17:33:17 Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
auto

Log applied.
Media recovery complete.
```

7）resetlogs 方式打开数据库
SQL> alter database open resetlogs;

8）验证

SQL> select * from scott.t1;

```
        ID
----------
         1
         2
         3
```

9）看看在 resetlogs 后，日志 sequence 重置了。
SQL> select * from v$log;

```
    GROUP#       THREAD#    SEQUENCE#          BYTES       MEMBERS  ARCHIVED  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- -------- ----------------
------------- ------------
         1             1             0     52428800             1 YES        UNUSED
0
         2             1             0     52428800             1 YES        UNUSED
0
         3             1             1     52428800             1 NO         CURRENT
1918000 2012-8-1 17
```

4.6.2 基于 SCN 的不完全恢复（略）

在手工基于 scn 的不完全恢复的命令子句是 change 关键字，与基于时间的不完全恢复类似，其命令格式只要将 recover 命令换成下面即可：

SQL> recover database until change 1918000;

这里不多赘述了。

4.6.3 基于 cancel 的不完全恢复(略）

4.6.4 基于 backup controlfile 的恢复（略）

不完全恢复中的复杂性是恢复数据文件的时候使用备份的控制文件。

1）为什么会使用备份的控制文件? 实际工作中主要有两种情况：

第一种：当前控制文件全部损坏，而数据文件备份，控制文件备份及当前日志处于不同 SCN 版本，它们之间又增加过表空间（数据文件）。
第二种：当前控制文件没有损坏，但想要恢复被删除的表空间。

2）使用备份的控制文件恢复数据库的语法：

```
recover database until [time|change] using backup controlfile;
```

注意：[time|change]是可选的，就是说如果条件满足，仍然可以做到完全恢复。
然后会有如下选项：

```
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
```

此语法的出现是由于控制文件和当前日志中不一致，当前日志的 scn 总是最新的，而控制文件可能是老的或尚未更新的（shutdwon abort 过)。

AUTO：　　自动使用 archivelog 前滚恢复，但一般不包括前滚 current log;
filename：　输入当前文件的路径和文件名，是指 current log 的恢复
CANCEL：　　退出。

考点：
1）在控制文件丢失后进行恢复将会将会出现停机时间，因此不能联机执行控制文件的恢复。
2）使用 backup controlfile 子句的恢复数据库之后，一律要使用 alter database open resetlogs 打开数据库。

范例 1：(属于第一种情况）

环境：当前控制文件损坏，数据文件损坏，有全备但之后增加了表空间，并备份了配套的控制文件。

模式：所有数据文件备份（老）------(新建表空间 abcd)-----备份控制文件（次新）-----日志文件（新）

分析：新建表空间数据文件损坏，全备里没有该数据文件的备份及控制文件描述，当前控制文件又丢失，只能用备份的控制文件恢复。

1）环境：

```
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
         2 UNDOTBS1                     YES NO  YES
         3 TEMP                         NO  NO  YES

SQL> select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- -------------------
         1              1              7     52428800              1  NO   CURRENT
6676574 2013-01-17 13:55:19
         2              1              5     52428800              1  YES  INACTIVE
6676549 2013-01-17 13:54:14
         3              1              6     52428800              1  YES  INACTIVE
6676562 2013-01-17 13:54:48

SQL> create tablespace abcd datafile '/u01/oradata/timran11g/abcd01.dbf' size 5m;
SQL> create table scott.a1 (name char(10)) tablespace abcd;
SQL> insert into scott.a1 values('a');
SQL> commit;
SQL> select * from scott.a1;

NAME
----------
a

SQL> alter system switch logfile;
```

2）备份控制文件

```
19:17:55      SQL>      alter      database      backup      controlfile      to
'/u01/oradata/timran11g/con.bak1';
```

3) 模拟 abcd01.dbf 损坏

```
[oracle@timran ~]$rm /u01/oradata/timran11g/abcd01.dbf          //数据库 open 状
态，删除 abcd01.dbf 数据文件

SQL> alter system flush buffer_cache;                    //db buffer 清空

SQL> conn / as sysdba                              //换个 session 查看  a1 表物理读失
败
```

```
已连接。
SQL> select * from scott.a1;
select * from scott.a1
                    *
第 1 行出现错误:
ORA-00376: 此时无法读取文件 3
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/abcd01.dbf'
```

4）关闭数据库

```
SQL> shutdown abort;
```

5）恢复所有数据文件备份，准备做不完全恢复

```
[oracle@timran timran11g]$ cd /u01/oradata/timran11g
[oracle@timran timran11g]$ rm *.ctl
[oracle@timran timran11g]$ rm *.dbf
[oracle@timran timran11g]$ cp /u01/back1/*.dbf ./
[oracle@timran timran11g]$ cp con.bak1 control01.ctl
[oracle@timran timran11g]$ cp con.bak1 control02.ctl
[oracle@timran timran11g]$ cp con.bak1 control03.ctl

SQL> startup
ORACLE 例程已经启动。

Total System Global Area  422670336 bytes
Fixed Size                  1300352 bytes
Variable Size             331352192 bytes
Database Buffers           83886080 bytes
Redo Buffers                6131712 bytes
数据库装载完毕。
ORA-01589: 要打开数据库则必须使用 RESETLOGS 或 NORESETLOGS 选项

SQL> col name for a50;
SQL> select file#,checkpoint_change#,name from v$datafile;

     FILE# CHECKPOINT_CHANGE# NAME
---------- ------------------ --------------------------------------------------
         1            6676574 /u01/oradata/timran11g/system01.dbf
         2            6676574 /u01/oradata/timran11g/sysaux01.dbf
         3            6676601 /u01/oradata/timran11g/abcd01.dbf
         4            6676574 /u01/oradata/timran11g/user01.dbf
         5            6676574 /u01/oradata/timran11g/example01.dbf
         6            6676574 /u01/oradata/timran11g/test01.dbf
         7            6676574 /u01/oradata/timran11g/undotbs01.dbf
```

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            6676343
         2            6676343
         3                  0
         4            6676343
         5            6676343
         6            6676343
         7            6676343

SQL>
```

可以看出：1）file3 在控制文件里记录是 abcd01.dbf,而与之对应的数据文件 3 是不存在的，2）备份的数据备份的 scn 比控制文件 scn 的还老。

6)使用备份控制文件恢复

```
SQL> recover database using backup controlfile;

ORA-00283: 恢复会话因错误而取消
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/abcd01.dbf'
ORA-01157: 无法标识/锁定数据文件 3 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/abcd01.dbf'
```

//此错是因为老备份里没有 abcd 表空间，但只要控制文件里记录了 abcd 就好办，方法是建一个 datafile 的空文件，而其中内容可由日志文件 recover(前滚)时填补出来。

```
SQL> alter database create datafile '/u01/oradata/timran11g/abcd01.dbf';
```

---再次使用备份控制文件恢复

```
SQL> recover database using backup controlfile;
......

ORA-00308: 无法打开归档日志 '/u01/disk1/timran/arch_1_804846837_9.log'
ORA-27037: 无法获得文件状态
Linux Error: 2: No such file or directory
Additional information: 3
```

//archive 日志前滚结束了，但当前日志里还有信息需要恢复
//注意: 对于这个例子来说，一定要看清提示：如果提示的不是归档的日志（是当前日志），

则要直接要输入 filename 不能输入 auto，否则 open 时会失败。

```
SQL> recover database using backup controlfile;              //再次做恢复

指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/oradata/timran11g/redo03.log                    //把蛇头（当前日志）给它。
已应用的日志。
完成介质恢复。
```

7）resetlogs 打开数据库

```
SQL> alter database open resetlogs;
```

8）验证

```
SQL> select * from scott.a1;

NAME
----------------------------------------
a
b
```

范例 2：(属于第一种情况）(略)

环境：当前控制文件损坏，新建表空间在备份控制文件之后

模式：全备（老）-----备份控制文件（次新）-----新建表空间 timran------日志文件（新）

分析：整个恢复过程中 datafile 结构有了变化，变化发生在备份控制文件之后，新增了表空间 timran,控制文件备份里没有此表空间记录，但日志里有。

1）环境

```
SQL> drop tablespace abcd including contents and datafiles;
SQL> alter database backup controlfile to '/u01/oradata/timran11g/con.bak2';
SQL> create tablespace timran datafile '/u01/oradata/timran11g/timran01.dbf' size
5m;
SQL> create table scott.r1 (id int) tablespace timran ;
SQL> insert into scott.r1 values(1);
SQL> commit;

SQL> select * from v$tablespace;

       TS# NAME                                                   INC BIG FLA ENC
---------- ------------------------------------------------------ --- --- --- ---
```

```
         0 SYSTEM                                              YES NO  YES
         1 SYSAUX                                              YES NO  YES
        14 TIMRAN                                              YES NO  YES
         4 USERS                                               YES NO  YES
         6 EXAMPLE                                             YES NO  YES
         8 TEST                                                YES NO  YES
         3 TEMP                                                NO  NO  YES
         2 UNDOTBS1                                            YES NO  YES

SQL> select * from scott.r1;

        ID
----------
         1
SQL>  select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- --------------------
         1              1              1     52428800              1 NO   CURRENT
6677119 2013-01-17 14:08:18
         2              1              0     52428800              1 YES  UNUSED
0
         3              1              0     52428800              1 YES  UNUSED
0
```

2）模拟新建数据文件损坏

```
[oracle@timran timran11g]rm timran01.dbf

SQL>alter system flush buffer_cache;
SQL>conn / as sysdba
SQL>select * from scott.r1;
第 1 行出现错误:
ORA-01116: 打开数据库文件 3 时出错
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/timran01.dbf'
ORA-27041: 无法打开文件
Linux Error: 2: No such file or directory
Additional information: 3
```

3) 关闭数据库

```
SQL>shutdown abort
```

4）还原所有数据文件，以老控制文件替换当前控制文件

```
[oracle@timran timran11g]$ cd /u01/oradata/timran11g
[oracle@timran timran11g]$ rm *.ctl
[oracle@timran timran11g]$ rm *.dbf
[oracle@timran timran11g]$ cp /u01/back1/*.dbf ./
[oracle@timran timran11g]$ cp con.bak2 control01.ctl
[oracle@timran timran11g]$ cp con.bak2 control02.ctl
[oracle@timran timran11g]$ cp con.bak2 control03.ctl
```

5）启动数据库

```
SQL> startup
ORACLE 例程已经启动。
......
数据库装载完毕。
ORA-01589: 要打开数据库则必须使用 RESETLOGS 或 NORESETLOGS 选项

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

     FILE# CHECKPOINT_CHANGE# NAME
---------- ------------------ ----------------------------------------------------
         1            6677122 /u01/oradata/timran11g/system01.dbf
         2            6677122 /u01/oradata/timran11g/sysaux01.dbf
         4            6677122 /u01/oradata/timran11g/user01.dbf
         5            6677122 /u01/oradata/timran11g/example01.dbf
         6            6677122 /u01/oradata/timran11g/test01.dbf
         7            6677122 /u01/oradata/timran11g/undotbs01.dbf

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            6676343
         2            6676343
         4            6676343
         5            6676343
         6            6676343
         7            6676343
```

6）使用备份控制文件恢复数据库

```
SQL> recover database using backup controlfile;
```

```
ORA-00279: 更改 6676343 (在 01/16/2013 14:11:39 生成) 对于线程 1 是必需的
ORA-00289: 建议: /u01/disk1/timran/arch_1_804846837_4.log
ORA-00280: 更改 6676343 (用于线程 1) 在序列 #4 中


指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
auto
......

指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/oradata/timran11g/redo01.log
......
ORA-00283: 恢复会话因错误而取消
ORA-01244: 未命名的数据文件由介质恢复添加至控制文件
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/timran01.dbf'

ORA-01112: 未启动介质恢复

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

     FILE# CHECKPOINT_CHANGE# NAME
---------- ------------------ -----------------------------------------------
         1            6678002 /u01/oradata/timran11g/system01.dbf
         2            6678002 /u01/oradata/timran11g/sysaux01.dbf
         3            6677999 /u01/oracle/dbs/UNNAMED00003      //注意这个问题,
老控制文件不知道之后的 timran01.dbf
         4            6678002 /u01/oradata/timran11g/user01.dbf
         5            6678002 /u01/oradata/timran11g/example01.dbf
         6            6678002 /u01/oradata/timran11g/test01.dbf
         7            6678002 /u01/oradata/timran11g/undotbs01.dbf

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

     FILE# CHECKPOINT_CHANGE#
---------- ------------------
         1            6678002
         2            6678002
         3                  0
         4            6678002
         5            6678002
         6            6678002
         7            6678002
```

7）重命名数据文件

SQL> alter database create datafile '/u01/oracle/dbs/UNNAMED00003' as '/u01/oradata/timran11g/timran01.dbf';

//上面的命令一石二鸟，自动完成了两个动作 1）加了一个数据文件 timran01.dbf，2）重命名控制文件 UNNAMED00003 为 timran01.dbf

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

```
     FILE# CHECKPOINT_CHANGE# NAME
---------- ------------------ ----------------------------------------------
         1            6678002 /u01/oradata/timran11g/system01.dbf
         2            6678002 /u01/oradata/timran11g/sysaux01.dbf
         3            6677999 /u01/oradata/timran11g/timran01.dbf
         4            6678002 /u01/oradata/timran11g/user01.dbf
         5            6678002 /u01/oradata/timran11g/example01.dbf
         6            6678002 /u01/oradata/timran11g/test01.dbf
         7            6678002 /u01/oradata/timran11g/undotbs01.dbf
```

```
SQL> recover database using backup controlfile;
ORA-00279: 更改 6677999 (在 01/17/2013 14:20:50 生成) 对于线程 1 是必需的
ORA-00289: 建议: /u01/disk1/timran/arch_1_804953298_1.log
ORA-00280: 更改 6677999 (用于线程 1) 在序列 #1 中

指定日志: {<RET>=suggested | filename | AUTO | CANCEL}
/u01/oradata/timran11g/redo01.log
已应用的日志。
完成介质恢复。
```

8）resetlogs 打开数据库

SQL> alter database open resetlogs;

9）验证

SQL> select * from scott.r1;

```
        ID
----------
         1
         2
         3
```

范例 3 恢复被删除的表空间（属于第二种情况）（略）

环境:用户使用正常操作命令删除了表空间及其数据文件,但之后又希望恢复删除的表空间。全备里有这个表空间的数据文件。

模式：全备（老）------控制文件备份（次新）-----删除表空间------所需日志（新）

分析：当用户使用 drop tablesapce xxx including contents and datafiles 这条 DDL 语句后，数据库的结构发生了变更，涉及了三个地方。
a）控制文件
b）该表空间下的数据文件
c）系统表空间（数据字典）

特别提醒的是：当前的控制文件里已经没有该表空间的信息了，所以不能使用当前的控制文件做恢复。恢复这个表空间要满足三个条件：
a）要有该表空间的数据文件备份
b）使用不完全恢复（基于时间点或 scn）
c）使用备份的控制文件，而这个控制文件是删除该表空间前的控制文件，不是当前的控制文件。这个非常重要，该控制文件中的内容记录了你需要恢复数据库结构，重要的这个控制文件必须包括有你要恢复的那个表空间的信息。

1）背景：

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

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

SQL> create table scott.t1(id int) tablespace test;

SQL> insert into scott.t1 values(1);

SQL> commit;

SQL> alter system switch logfile;

SQL> /
```

```
SQL> /
```

2）记录下当前 scn

```
SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    7222848
```

3)备份控制文件

```
SQL> alter database backup controlfile to '/u01/oradata/timran11g/con.bak'
```

4）删除表空间及数据文件

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

SQL> shutdown abort
```

5）删除所有数据文件和当前控制文件，还原所有数据文件及备份的控制文件

```
[oracle@timran timran11g]$ rm *.dbf
[oracle@timran timran11g]$ rm *.ctl
[oracle@timran timran11g]$ cp /u01/back1/*.dbf ./
[oracle@timran timran11g]$ cp con.bak control01.ctl
[oracle@timran timran11g]$ cp con.bak control02.ctl
[oracle@timran timran11g]$ cp con.bak control03.ctl
```

6)启动数据库后，要做基于时间点（或 SCN）的不完全恢复

```
SQL> startup
...
数据库装载完毕。
ORA-01589: 要打开数据库则必须使用 RESETLOGS 或 NORESETLOGS 选

SQL> recover database until change 7222848 using backup controlfile;
```

7）以不完全恢复方式打开数据库

```
SQL> alter database open resetlogs;
```

8）验证

```
SQL> select * from scott.t1;

        ID
----------
         1
```

第五章 闪回 flashback

5.1 flashback 的功能：利用 flashback log 或 undo data 对 database 可以恢复到过去某个点，可以作为不完恢复的补充

5.2 flashback 分类：（DBA-II PPT：253）

1）flashback drop
2）flashback query （新添 flashback database archive)
3) flashback table
4）flashback version query
5）flashback transaction
6）flashback database

5.2.1 闪回 drop 又名闪回删除(PPT-II-299)

1）理解回收站（recyclebin)

从管理的角度为每个用户“分配”一个回收站，但这个回收站并不实际开辟空间(只是个逻辑容器)，当 drop table 时（非 purge)，原来的表所使用的段中的数据并没有真正的删除。实际上是把 table 的段名以回收站方式重命名。该段所在表空间不足需要扩展时，回收站中的信息会被自动清除（考点)。

做个例子理解一下：

```
SQL> show parameter recyclebin                  //当初始化参数 recyclebin 为 on 时，每个用户都有了自己的回收站

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
recyclebin                           string      ON


//如果参数设为 off 就取消了用户的回收站，那么当你 drop table 就相当于 purge 了。

SQL> create tablespace test datafile '/u01/oradata/timran11g/test01.dbf' size 1m;
SQL> create table scott.t1(id int) tablespace test;
SQL> select segment_name from dba_segments where tablespace_name='TEST';
    //看 test 表空间下有了一个段

SEGMENT_NAME
--------------------------------------------------------------------------------
_
```

```
T1

SQL> select sum(bytes) from dba_free_space where tablespace_name='TEST';
    //看这个段有多少空闲空间

SUM(BYTES)
----------
    917504

SQL> insert into scott.t1 values(1);
SQL> insert into scott.t1 select * from scott.t1;       //将表空间撑满
/
/
第 1 行出现错误:
ORA-01653: 表 SCOTT.T1 无法通过 string (在表空间 TEST 中) 扩展

SQL> select count(*) from scott.t1;

  COUNT(*)
----------
     65536

SQL> select sum(bytes) from dba_free_space where tablespace_name='TEST';

SUM(BYTES)
----------                              //没有空闲空间

SQL> drop table scott.t1;

SQL> select segment_name from dba_segments where tablespace_name='TEST';

SEGMENT_NAME
--------------------------------------------------------------------------------
-
BIN$4KZBTYTKocDgQAB/AQAKRA==$0

SQL> select sum(bytes) from dba_free_space where tablespace_name='TEST';

SUM(BYTES)
----------
    983040
```

请看，TEST 表空间中的空闲空间又回来了，这说明如果 test 表空间不够时，这部分空闲空间是可以被重新利用的，实际上即使你设置了表空间 autoextend 特性，Oracle 会先使用

recyclebin，若空间还不够，再考虑 autoextend.

SQL> create table scott.emp1 tablespace test as select * from scott.emp; //挤占 test 表空间

SQL> select sum(bytes) from dba_free_space where tablespace_name='TEST';

SUM(BYTES)

917504

SQL> select segment_name from dba_segments where tablespace_name='TEST'; //t1 表的数据已经被冲掉了，使用闪回删除无法找回了（考点）。

SEGMENT_NAME

-
EMP1

//t1 表的数据已经被冲掉了，使用闪回删除无法找回了（考点）。

2）关于回收站中的对象的闪回和清除

闪回和清除的顺序不同

闪回使用 LIFO （后进先出）
清除使用 FIFO （先进先出）

假设回收站里有两个 t1 表，看以下两条语句：

SQL> flashback table t1 to before drop; //闪回的是最新的那个 t1 表（考点）。
SQL> purge table t1; //清除的是最旧的那个 t1 表（考点）。

如果想避免混淆，可以直接点出回收站里的表名

SQL> flashback table " BIN$qrJLbL74ZgvgQKjA8Agb/A==$0" to befroe drop;
SQL> purge table "BIN$qrJLbL74ZgvgQKjA8Agb/A==$0";

SQL> purge recyclebin; //清空回收站
3）如何恢复同一个 schema 下准备闪回的表已有同名的对象存在，闪回 drop 需要重命名.
SQL> drop table t1;
SQL> create table t1 as select * from emp;

```
SQL> select * from tab;
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
BIN$qrJLbL76ZgvgQKjA8Agb/A==$0 TABLE
TEST                           TABLE

06:56:50 SQL> show recycle;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
---------------- ------------------------------ ------------ -------------------
T1               BIN$qrJLbL76ZgvgQKjA8Agb/A==$0 TABLE        2011-08-17:06:56:36
SQL> flashback table test to before drop;
flashback table test to before drop
*
ERROR at line 1:
ORA-38312: original name is used by an existing object

SQL> flashback table t1 to before drop rename to test_old;          //闪回 drop
语句中使用了重命名
```

4)system 表空间的对象没有回收站，所以在 sys 下缺省使用 system 表空间时，drop table 会直接删除对象（考点）

5）如果一个表上面有索引和约束，drop 后再闪回表，索引和约束还在吗？

```
SQL> select * from tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
SALGRADE                       TABLE

create table t (id int,name char(10));
alter table t add constraint pk_t primary key(id);
insert into t values (1,'sohu');
insert into t values (2,'sina');
commit;
SQL> select * from t;

        ID NAME
```

```
---------- ----------
         1 sohu
         2 sina

-----看一眼约束和索引

SQL> select * from user_indexes;
SQL> select * from user_constraints;

SQL> drop table t;

SQL> select * from tab;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
BIN$yF3hbvIcioTgQAB/AQAJlg==$0 TABLE
BONUS                          TABLE
DEPT                           TABLE
EMP                            TABLE
SALGRADE                       TABLE

-----表被 drop 到回收站，再看一眼约束和索引

SQL> select * from user_indexes;        //索引不见了
SQL> select * from user_constraints;        //约束有，但乱码(除外键约束外)

SQL> flashback table t to before drop;

闪回完成。

SQL> select * from t;

        ID NAME
---------- ----------
         1 sohu
         2 sina

SQL>
-----再看约束和索引

SQL> select * from user_indexes;        //索引回来了，有效（考点），但乱码
SQL> select * from user_constraints;        //约束也在，有效（考点），但乱码

-----分别重命名索引和约束
```

```
SQL> alter index "BIN$yF3hbvIbioTgQAB/AQAJlg==$0" rename to pk_t;

SQL> alter table t rename constraint "BIN$yF3hbvIaioTgQAB/AQAJlg==$0" to pk_t;
```

-----再看约束和索引
ok!

5.2.2 闪回查询 flashback query:（用于 DML 误操作）(PPT-II-285)

1) 要点：

利用在 undo tablespace 里已经被提交的 undo block（未被覆盖），可以通过查询的方式将表里面的记录回到过去某个时间点。
通过设置 undo_retention 参数设置前镜像的保留时间。
查询的语法：

```
select   …    as of scn | timestamp
```

2）例：

```
sys:
create table scott.student (sno int,sname char(10),sage int);
insert into scott.student values(1,'Tom',21);
insert into scott.student values(2,'Kite',22);
insert into scott.student values(3,'Bob',23);
insert into scott.student values(4,'Mike',24);
commit;
/

SQL> select * from scott.student;

       SNO SNAME            SAGE
---------- ---------- ----------
         1 Tom                21
         2 Kite               22
         3 Bob                23
         4 Mike               24

select to_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') from dual;    //取时间 1
select current_scn from v$database;  //取 scn 1

delete scott.student where sno=1;
commit;
```

```
select * from scott.student;

       SNO SNAME            SAGE
---------- ---------- ----------
         2 Kite               22
         3 Bob                23
         4 Mike               24

select to_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') from dual;    //取时间 2
select current_scn from v$database;   //取 scn 2

update scott.student set sage=50;
commit;
select * from scott.student;

       SNO SNAME            SAGE
---------- ---------- ----------
         2 Kite               50
         3 Bob                50
         4 Mike               50

select to_char(sysdate, 'yyyy-mm-dd hh24:mi:ss') from dual;    //取时间 3
select current_scn from v$database;取 scn 3

scott:
select * from student as of timestamp to_date('取时间 2','yyyy-mm-dd hh24:mi:ss');
select * from student as of scn 取 scn1;

create table student2 as select * from scott.student as of scn 取 scn1;
drop table student;
rename student2 to student;
select * from student;
```

*考点：可以查询以前某个时间点的数据库，但是永远不能对过去时间点得数据库做 DML 操作。

5.2.3 闪回表 (PPT-II-264)

1)要点：闪回表通常是把表的状态回退到以前的某个时刻或者 SCN 上。(其实向前向后都能闪)。自动恢复相关的属性，包括索引、触发器等。前提是对表启用行移动。

语法：

```
flashback table <table_name> to timestamp | scn
```

2）例：

```
delete student;
commit;
alter table student enable row movement;
flashback table student to scn XXXXX
```

考点：

1）sys 的表不能闪回。

2）必须使能行移动。

3）缺省下，闪回表的过程中有关的 trigger 都关闭。

5.2.4 11g 新特性闪回归档 (FLASHBACK ARCHIVE)(PPT-286-295)（TOTAL RECALL）

1. 概念：无限期的存储表行的前镜像，通过后台进程 FBDA，捕捉必要的数据并将其保存在归档上，然后可以使用常规闪回查询命令(as of)查询需要的数据，但闪回可以回朔到多年以前。闪回归档可以看成是闪回查询时间的延伸。

2. 有几个要点：

1) 首先要有一个（或多个）表空间存放归档表，可以加上配额，使用原有的表空间在技术上是可行的，但 Oracle 建议它们与常规的数据分开 存放更好。

2) 先要建立一个归档名，作为一个数据库的对象，可以为它指定 default 属性，含义是把所有要归档的表都建立在该缺省的归档名下。

3) 可以根据需要建立多个闪回归档，这取决于你需要多少种不同的保留时间。可以调整保留时间，一旦超过了保留期限，后台进程 FDBA 将自动删除该表记录，也可以在保留期内手动进行删除操作。

4) 关于两个权限：

```
flashback archive administer   //授予用户创建，修改或删除闪回回档
flashback archive          //授予用户对表进行归档。
```

5) 启用表的闪回归档需要使用 alter table 表 flashback archive 归档名

6) 不能对已经定义为闪回归档的表再使用 drop 或 truncate 等 DDL 命令。

7) 闪回归档 enable 也有个前提条件，undo_management 要设为 on, 否则修改表时报 ORA_55614 错误

8) 关于闪回归档的视图：

```
DBA_FLASHBACK_ARCHIVE          //描述配置的归档
DBA_FLASHBACK_ARCHIVE_TABLES       //查看归档的表
DBA_FLASHBACK_ARCHIVE_TS       //列出使用的表空间
```

3. 举例：

例 1：DBA 建一个表空间用于闪回归档，然后将闪回归档作为一个对象创建并授权。

```
sys:
create tablespace fda datafile '/u01/oradata/timran11g/fda01.dbf' size 5m;
create flashback archive fla1 tablespace fda quota 2m retention 1 year;
```

grant flashback archive on fla1 to scott;

用户将自己的表定义为闪回归档
scott:
alter table emp1 flashback archive fla1;

可以做些 DML 操作在使用闪回查询（as of)验证。

尝试做一些 DDL 操作会报错：

scott：
alter table emp1 drop column comm;
truncate table emp1;
drop table emp1;
ORA-55610: 针对历史记录跟踪表的 DDL 语句无效.

取消归档保护
sys:
SQL> select * from dba_flashback_archive　　//查看有哪些归档名
SQL> select * from dba_flashback_archive_ts;　　//查看归档使用表空间的信息
SQL> select * from dba_flashback_archive_tables;　　//查看表 emp1 和归档名的关系。
sys:
SQL> alter table scott.emp1 no flashback archive;　　//将表 scott.emp1 从闪回归档中取消，

例 2：通过一道考题我们来实验一下，什么是 default 归档：

题目：
Note the output of the following query;
SQL> SELECT flashback_archieve_name, status FROM dba_flashback_archieve;

FLASHBACK_ARCHIEVE_NAME STATUS
FLA1

You executed the following command to enable Flashback Data Archive on the EXCHANGB_PATE table:
ALTER TABLE exchange_rate FLASHBACK ARCHIEVE;
What is the outcome of this command?
A. The table uses the default Flashback Data Archive.
B. The Flashback Data Archive Is created In the SYSAUX tablespace.
C. The Flashback Data Archive is created in the same tablespace where the tables are stored.
D. The command generates an error because no flashback Data Archive name is

specified and there is no default Flashback Data Achieve.

答案：d

```
SQL> conn / as sysdba。
SQL> alter flashback archive fla1 set default;
SQL> select FLASHBACK_ARCHIVE_NAME,STATUS from dba_flashback_archive;

FLASHBACK_ARCHIVE_NAME
STATUS
--------------------------------------------------------------------------------
-------
FLA1
DEFAULT
```

SQL> alter table scott.emp1 flashback archive;　　//不用指定归档名，emp1 绑定了 FLA1（DEFAULT）。

```
SQL> select * from dba_flashback_archive_tables;
```

删除闪回归档

```
SQL> drop flashback archive fla1;
```

考点：设置闪回数据归档有两个先决条件：1）使能 automatic undo managent 2）tablespace 要 ASSM 的。

5.2.5 闪回版本查询(PPT-II-260)

1）要点：闪回查询仅仅能够得到过去某个时间点上的数据，但是无法反映出一段时间内数据表中数据变化的细节，
10g 的闪回版本查询可以对时间段内数据表的每行变化（不同版本）进行查询。

2）语法：select　…　from　…　versions　between
　其中，select 后面可以选择伪列，来获得事务的开始、结束时间、SCN 号、ID 号等。

3）举例：

```
scott:
create table t3 (id int, name char(10));
insert into t3 values(1,'tim');
insert into t3 values(2,'mike');
insert into t3 values(3,'brain');
insert into t3 values(4,'cade');
commit;
```

```
update t3 set name='nelson' where id=4;
commit;
delete t3 where id=2;
commit;
update t3 set id=id+100;
commit;
```

看看 t3 表经历的时间变化

```
SQL> select versions_startscn, versions_endscn, versions_xid,
versions_operation,id,name from scott.t3 versions between scn minvalue and
maxvalue;

SQL>select versions_xid, versions_operation,id, name from t3 versions between scn
minvalue and maxvalue;

VERSIONS_XID     VERSIONS_OPERATION                                 ID NAME
---------------- ------------------ ---------------------------------------- ----
------
03000800F3010000 U                                                   104
nelson
03000800F3010000 U                                                   103
brain
03000800F3010000 U                                                 101 tim
04000A0076010000 D                                                   2 mike
08002000F9010000 U                                                     4
nelson
                                                                     1 tim
                                                                     2 mike
                                                                     3
brain
                                                                     4 cade
```

考点：
1)闪回版本查询不能用于外部表、临时表或 V$视图。原因是这些对象都不生成撤销数据。(临时表的撤销是基于 session 的)。
2)闪回版本中的内容不包括未提交的 DML 语句。

5.2.6 闪回事务查询(PPT-II-270)

1）要点：
闪回事务查询可以提供撤销查询语句。从 flashback_transaction_query 这个视图里查询引起数据变化的事务，和撤销事务的 SQL 语句
也就是查询 operation 和 undo_sql 列。可以和闪回版本查询结合起来使用。

sys:
SQL>desc flashback_transaction_query;
SQL>select undo_sql from flashback_transaction_query where xid=hextoraw('事务号');

接上例：

SQL> select undo_sql from flashback_transaction_query where xid=hextoraw('03000800F3010000');

UNDO_SQL
----
update "SYS"."T3" set "ID" = '4' where ROWID = 'AAANByAABAAAO/yAAD';
update "SYS"."T3" set "ID" = '3' where ROWID = 'AAANByAABAAAO/yAAC';
update "SYS"."T3" set "ID" = '1' where ROWID = 'AAANByAABAAAO/yAAA';

//执行上面语句，原操作（update t3 set id=id+100;）就撤销了

SQL> update "SYS"."T3" set "ID" = '4' where ROWID = 'AAANByAABAAAO/yAAD';
SQL> update "SYS"."T3" set "ID" = '3' where ROWID = 'AAANByAABAAAO/yAAC';
SQL> update "SYS"."T3" set "ID" = '1' where ROWID = 'AAANByAABAAAO/yAAA';
SQL> commit;

闪回事务查询考点
1）Enable Supplemental Logging 并且数据库版本 10.0 compatibility
2）查询 flashback_transaction_query 视图需要 SELECT ANY TRANSACTION 权限。

5.2.7 闪回数据库

1）概念：
闪回数据库相当于不完全恢复，它通过闪回日志将数据库整体回退到某个时间点。
闪回数据库针对的是逻辑错误，如果数据库发生了物理损坏或介质丢失，闪回数据库将无能为力，闪回数据库不能完全代替传统不完全恢复（考点）
使用闪回数据库，需要开启闪回日志，闪回日志存放在闪回恢复区里。

一旦启用了闪回数据库，某些块的映像会从 db buffer 复制到 SGA 的一个新的存储区域中，即闪回缓冲区，然后再由后台进程(Recover Write RVWR)将此闪回恢复区的内容刷新到磁盘和闪回日志。这一切并没有改变 LGWR 的常规作用。与重做日志不同的是 RVWR 不是记录数据库变化的日志，而是记录完整块影像的记录。（PPT-311）

*考点：不同于重做日志，闪回日志不能被多路复用和归档。它们是自动创建和管理的。

2）闪回日志放在闪回恢复区里

闪回恢复区(flash recovery area)是一个非常重要的概念，它不仅存放闪回日志，还有许多与恢复有关的文件，比如，可以存放与 RMAN 相关的三种自动管理的文件，1.归档日志、2.控制文件自动备份 3.RMAN 备份片。当 flash recovery area 空间不够用，Oracle 还可以自动清除一些废弃（obsolete)文件。(PPT-II-49-55)

```
SQL> show parameter recovery_file

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

SQL> show parameter flash

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
db_flashback_retention_target        integer     1440
```

*考点：1.设置 db_recovery_file_dest 之前必须先设置 db_recovery_file_dest_size 。2.闪回日志存放的保留期，缺省值 1440，单位是分钟。

3）配置闪回数据库的基本步骤：

3.1) 数据库闪回要在 mount 状态下

```
SQL> STARTUP MOUNT EXCLUSIVE;
```

3.2) 配置成归档方式

```
SQL> ALTER DATABASE ARCHIVELOG;
```

闪回数据库必须配备成归档方式，是因为闪回日志里只记录了快照，这些快照可以使数据库回退到某个 SCN 点，而回退快照的 SCN 仅比你指定的 SCN 提前一点，然后会运用归档日志或当前日志前滚一小段，当到达指定的 san 时停住。然后在此 SCN 前 resetlogs 打开数据库。

3.3) 指定闪回恢复区

设置参数 db_recovery_file_dest='u01/flash_recovery_area'

3.4) 配置闪回保留时间

设置参数 db_flashback_retention_target=1440

注意单位是分钟，缺省 1440 相当于 24 小时

3.5) 使能闪回数据库

SQL> alter database flashback on; // 其结果在 /u01/flash_recovery_area/TIMRAN11G/flashback 目录下创建了一个.flb 的闪回日志文件。

```
SQL> select flashback_on from v$database;        //查看闪回数据库启用或关闭

FLASHBACK_ON
------------------
YES
```

只是检查 RVWR 进程和闪回日志是否有了，如果 alter database flashback off;闪回日志自动清除（考点）。

3.6）打开数据库

```
alter database open;
```

4) 例: 恢复被删除的用户。

首先进入 mount 下，配置相关参数，开启闪回数据库日志，然后打开数据库，接下来：

4.1)取当前 SCN

```
SQL> select current_scn from v$database;

CURRENT_SCN
-----------
    7248690
```

4.2）删除 scott 用户

```
SQL> drop user scott cascade;
```

4.3）准备到 mount 下做闪回数据库

```
SQL> shutdown immediate;
SQL> startup mount exclusive          //mount 要附加 exclusive(独占),意思是其他 sys 不能打开数据库，这是一个考点。

SQL> flashback database to scn 7248690;
```

4.4）只读方式打开，确认 scott 已经被闪回

```
SQL> alter database open read only;
SQL> select * from scott.emp;
```

4.5）确认无误后，重新以 resetlogs 方式打开数据库（属于不完全恢复）

```
startup force;
alter database open resetlogs;          //一旦 resetlogs 打开，若想再一次做闪回，只能闪回比当前更早的 scn（见 PPT-315page）。
```

提醒：要在 mount 下闪回数据库

```
flashback database to timestamp to_char('2012-03-02 19:11:11','yyyy-mm-dd hh24:mi:ss');
flashback database to scn 1264788;
```

闪回后，打开数据库，第一次最好用只读方式，看看是否恢复到你希望的那个时间点上去了，如果不是你希望的，还可以重新闪回（前闪/后闪都可以，因为只读方式 scn 是不会增长的）。

5）那些操作适合或不适合闪回数据库

适合：
drop table xxx purge
drop user xxx cascade
在某个用户操作影响到整个数据库时
错误的 truncate 表

不适合：
使用了备份的控制文件或 trace 文件
drop 表空间的操作
段重组后的表，收缩后的数据文件。

6）限制生成闪回数据量

默认情况下，如果启用了闪回数据库，那么会记录所有表空间的闪回数据，使用下面命令可以关闭个别表空间的属性，自然就不会生成该表空间的闪回数据。

```
SQL> alter tablespace <tablespace_name> flashback off;      //可以在 open 下设置
SQL> alter tablespace <tablespace_name> flashback on;       //可以在 mount 下设置
SQL> select * from v$tablesapce                  //有个字段 FLA 可显示状态
```

flashback off 的表空间在闪回数据库之前要 offline（不参与闪回），所有数据文件 scn 不一致前不能打开数据库。联机前要做部分还原和不完全恢复，但相对也能节省时间。

7）关于闪回数据库有两个视图：

v$flashback_database_log 显示所能回退到的最早时间，取决与保留的 Flashback Database

Log 的多少。
v$flashback_database_stat 以一个时间段为一行（大约 1 小时），记录单位时间内数据库的活动量。

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

OLDEST_FLASHBACK_SCN    OLDEST_FLASHBACK_TI    RETENTION_TARGET    FLASHBACK_SIZE
ESTIMATED_FLASHBACK_SIZE
---------------------- -------------------- ----------------- --------------- -------
------------------
            12233583 2014-02-08 18:41:58                  1440            8192000
0
```

考点：OLDEST_FLASHBACK_SCN 表示能够让你闪回到最早的那个 scn 点。

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

BEGIN_TIME           END_TIME             FLASHBACK_DATA     DB_DATA   REDO_DATA
ESTIMATED_FLASHBACK_SIZE
-------------------- -------------------- --------------- ----------- ----------- ---
-----------------------
2014-02-08 19:49:15 2014-02-08 20:38:32          4587520     7921664     2973696
0
2014-02-08 18:41:58 2014-02-08 19:49:15         19709952    22528000    18658304
421822464
```

相比来看，v$flashback_database_log 信息对于 flash database 更有帮助。

*考点：闪回数据库要求归档日志模式，并使用 alter database open resetlogs 来创建数据库的一个化身。

第六章：RMAN 概述

6.1 rman 的定义和功能:

1) Recovery Manager

2) 通过 oracle 提供的包，建立备份和恢复的 server process，在 oracle server 上做备份和恢复

3）rman 备份 datafile（分三个层次：database、tablespace、datafile)、controlfile、spfile、archivelog

4）在归档模式下支持一致性备份（冷备）和非一致性备份（热备）

5）非归档只支持一致性备份（冷备）

*考点：

1）非归档方式的 RMAN 只能冷备，并在 mount 下做，但手工备份在 mount 下 cp 出来的备

份对于 RMAN 是不可用的。

2）非归档方式的 RMAN 恢复只能还原最后一次备份。

6.2 rman 的优点：

1）不备份数据文件中从未使用的块（备份 segment 高水位线以下的 block），节省空间（考点）

2）备份时自动检查数据文件是否有坏块，并可以标记坏块，跳过坏块,因为 RMAN 是 ORACLE BLOCK 级备份技术

3) 可以实现增量备份

4）可以备份 ASM 文件

6.3 rman 的架构：

1）可连接三类数据库：target database(备份的目标库) ,auxiliary database(复制数据库), catalog database(目录数据库)。

2）存储设备：disk 、tape （sbt 磁带机）存放备份文件的设备

3）channel ：目标库和存储设备之间备份恢复通道（服务进程）默认最少启动一个 channel，可同时启动多个 channel 并发操作。

4）server process：用于备份和恢复的进程

5）rman 的元数据：记录备份的信息（放在目标库的 controlfile 里）

6）catalog database :集中管理、存放备份的元数据，还可以存储备份脚本

7）MML:media manage layer 介质管理层：用于管理磁带机的库文件或驱动

6.4 rman 连接目标库方法

1）本地连接

RMAN 工具和 target database 在同一台服务器

```
[oracle@timran ~]$ rman target /
```

2）远程连接

RMAN 客户端通过 ORACLE_NET 连接 target database 在 target database 启动监听，在 client 配置 tnsnames.ora。

```
C:\Documents and Settings\timran>rman target sys/system@timran11g
```

6.5、查看 rman 的默认配置，修改 rman 的配置信息

1）查看 rman 的默认配置

```
RMAN> show all;
```

db_unique_name 为 TIMRAN11G 的数据库的 RMAN 配置参数为:

```
CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default
CONFIGURE BACKUP OPTIMIZATION OFF; # default
CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default
CONFIGURE CONTROLFILE AUTOBACKUP ON; # default
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/U01/MYRMAN/%F';
# default
CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default
CONFIGURE MAXSETSIZE TO UNLIMITED; # default
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
CONFIGURE COMPRESSION ALGORITHM 'BZIP2'; # default
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default
CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/u01/oracle/dbs/snapcf_timran11g.f'; #
default
```

2）理解以上各行 RMAN 环境变量

第一行：CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default

有两种保留策略：两者只能选一，互斥。（见 PPT47）
一个是时间策略，指定一个时间窗口，必须能恢复此时间段内任一个时刻的数据。
一个冗余策略，规定至少有几个冗余的备份。

恢复窗口备份保存策略:
例如，假如我们指定恢复窗口是 7 天，假设今天是星期一，此前存在 3 个全备及归档日志。第一个全备是 5 天前生成的，第二个全备是十天前生成的，而最早一个全备是 15 天前备份的，那么十天前生成的备份及之后的归档日志必须保留，而 15 天前的那个备份会成为废弃备份（obsolete）（见 PPT-48）。

下面的命令将恢复窗口配置为 7 天：

```
RMAN> configure retention policy to recovery window of 7 days;
```

冗余备份保存策略:
使用这种保存策略，RMAN 会从最新备份开始保留 N 个数据备份，其余的废弃(obsolete)。

例如，如果有三个备份，而冗余数是 2，那么最早的那个备份将被废弃。下面的命令将备份策略设置为 2：

```
RMAN> configure retention policy to redundancy 2;
```

设置 NONE 可以把使备份保持策略失效，Clear 将恢复默认的保持策略

```
RMAN> configure retention policy to none; //RMAN 此后不会将任何备份集或映像文件标
```

记为 obsolete 状态。
RMAN> configure retention policy clear;

例：保证至少有一个备份能恢复到 Sysdate-5 的时间点上，之前的备份将标记为 Obsolete（废弃）
RMAN> configure retention policy to recovery window of 5 days;

至少需要有三个冗余的备份存在，如果多余三个备份以上的备份将标记为冗余
RMAN> configure retention policy to redundancy 3;

第二行：CONFIGURE BACKUP OPTIMIZATION OFF; # default

设置备份优化选项（ optimization ）可以在配置中设置备份的优化，如：
RMAN> configure backup optimization on;

如果优化设置打开，它只用于归档日志，只读或脱机表空间的数据文件，因为这些文件是不会变化的，备份集运行一个优化算法。跳过重复的备份文件，比如你要备份归档日志，此参数设为 on 可以避免重复备份，可大大节省空间和时间。 (PPT103)

考点：备份优化选项依赖于 RETENTION POLICY 策略，如果启用优化，在已有足够相同文件副本的情况下(r+1)，RMAN 将不创建额外的文件副本。

第三行：CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default

设备类型有两种，可以是磁盘（DISK），或者磁带（STB），默认为磁盘。

第四行：CONFIGURE CONTROLFILE AUTOBACKUP OFF; # default

设置控制文件自动备份 (autobackup on)

RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;

off: 默认情况下，在备份 system 表空间时，会备份控制文件和 spfile
on: 在做任何文件的备份时，会自动备份控制文件和 spfile，并且数据库的物理结构发生变化时，也自动备份 controlfile。

第五行：CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F'; # default

可以用如下的配置指定控制文件的备份路径与格式，format 指明备份文件的路径和文件名
RMAN 备份文件格式 备份文件可以自定义各种各样的格式，如下：

%c 备份片的拷贝数
%D 位于该月中的第几天 (DD)

%M 位于该年中的第几月 (MM)
%F 一个基于 DBID 唯一的名称,这个格式的形式为 c-IIIIIIIIII-YYYYMMDD-QQ,
%d 数据库名称其中 IIIIIIIIII 为该数据库的 DBID，YYYYMMDD 为日期，QQ 是一个 1-256 的序列
%n 数据库名称，向右填补到最大八个字符
%u 一个八个字符的名称代表备份集与创建时间
%p 该备份集中的备份片号，从 1 开始到创建的文件数
%U 一个唯一的文件名，代表%u_%p_%c
%s 备份集的号
%t 备份集时间戳
%T 年月日格式(YYYYMMDD)

第六行：CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;

指定在以后的备份与恢复操作中并行度为 2，即同时开启 2 个通道进行备份与恢复

并行的数目决定了开启通道的个数，当然也可以在 RUN{}中指定通道来决定备份与恢复的并行程度。如果在 RUN{}中指定了通道配置，将覆盖掉配置参数中指定的默认并行度(考点)。通常一个通道对应一个备份集。

提高 RMAN 性能和吞吐量，除了通道数，还可以通过一些参数控制输入数据文件数，备份集数，备份片数。

参数影响可有全局和局部两个层次：基本上分为

PARALLELISM：保存在 RMAN 存储库。
CONFIGURE CHANNEL：更新配置，保存在 RMAN 存储库。
ALLOCATE CHANNEL：限于该通道，不保存。
BACKUP: 限于本次语句，不保存。

相关参数
FILESPERSET: 每个备份集的最大输入文件数 在 BACKUP 语句中描述
MAXOPENFILES: 在给定时间内可以打开的最大输入文件数量 在 ALLOCATE CHANNEL 或 CONFIGURE CHANNEL 中描述
MAXPIECESIZE: 每个通道的备份片大小 在 ALLOCATE CHANNEL 或 CONFIGURE CHANNEL 中描述
BACKUP DURATION: 增加或减少完成备份的时间 在 BACKUP 语句中描述
MAXSETSIZE: 限制最大备份集大小 在 BACKUP 语句中描述

1）在 RUN{}里分配通道，可以指定备份的数据文件和通道对应关系。

例 1：

RMAN> RUN {

```
2> ALLOCATE CHANNEL c1 DEVICE TYPE sbt;
3> ALLOCATE CHANNEL c2 DEVICE TYPE sbt;
4> ALLOCATE CHANNEL c3 DEVICE TYPE sbt;
5> BACKUP
6> INCREMENTAL LEVEL = 0
7> FORMAT '/disk1/backup/df_%d_%s_%p.bak'
8> (DATAFILE 1,4,5 CHANNEL c1)
9> (DATAFILE 2,3,9 CHANNEL c2)
10> (DATAFILE 6,7,8 CHANNEL c3);
11> ALTER SYSTEM ARCHIVE LOG CURRENT;
12>}
```

2）使用 FILESPERSET 参数限制备份集中的输入文件数， 指定每个备份集中一次可以包含输入文件最大数，该参数默认值为 64。例 2 中约定每个备份集中备份 8 个文件。如果数据库有 20 个文件，那将只生成 3 个备份集，得到 3 个备份片，虽然开了 4 个通道，但并行度是 3，有 1 个通道闲置。

例 2：

```
RMAN> RUN{
2> ALLOCATE CHANNEL t1 DEVICE TYPE sbt;
3> ALLOCATE CHANNEL t2 DEVICE TYPE sbt;
4> ALLOCATE CHANNEL t3 DEVICE TYPE sbt;
5> ALLOCATE CHANNEL t4 DEVICE TYPE sbt;
6> BACKUP DATAFILE FILESPERSET 8;
}
```

例 2 中假如取消 ALLOCATE CHANNEL 语句，则通道数按全局变量 PARALLELISM 指定，假定为 1，那 20 个文件，FILESPERSET=8 该是怎样的结果：一个通道干活，8 个文件组成一个备份集，每备份集对一个备份片，完成一个备份集再接下一个备份集。串行的跑三趟。

3）使用 MAXPIECESIZE 参数限制备份片大小

例 3：

```
RUN{
CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 300M;
BACKUP datafile 2 format '/u01/myrman/%U.bak';
}
```

限定了最大备份片为 300M，如果输出文件是 500M（注意不是限制输入文件），则会在一个备份集中生成两个备份片。

如果将

```
CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 300M;
```

换成

```
ALLOCATE CHANNEL c1 DEVICE TYPE DISK MAXPIECESIZE 300M;
```

就是全部和局部的区别

4）使用 maxsetsize 参数限制备份集大小:
例 4：
RMAN> backup database maxsetsize 1G format '/u01/myrman/%U.bak';

结果如何，一个通道产生了多个备份集，每个备份集对一个备份片。
注意：一个备份集只能占一个通道，一个输入数据文件不能分成多个备份集，所以这里必须满足：最大输入文件<1G.

5）使用 SECTION SIZE 子句设置多段备份（multisection backup）

通常一个数据文件只能占用一个通道，如果某个 datafile 太大，为提高备份效率，可以考虑采用多道关键字 SECTION SIZE，并行化这个大文件的备份。例 5 中的数据文件约 800M，被分成若干 file section 来备份，它启动了三个通道，每个通道备份 300M 的 file section，但三个通道产生一个备份集，包含三个压缩后的备份片。

例 5:
RMAN> RUN{
ALLOCATE CHANNEL d1 DEVICE TYPE disk;
ALLOCATE CHANNEL d2 DEVICE TYPE disk;
ALLOCATE CHANNEL d3 DEVICE TYPE disk;
BACKUP AS COMPRESSED BACKUPSET DATAFILE 2 SECTION SIZE 300M format '/u01/myrman/%U.bak';
}

注意: SECTION SIZE 隐含限定 maxpiecesize,当 maxpiecesize 全局生效时不能使用 SECTION SIZE。（考点）
所以，要使用 SECTION SIZE，maxpiecesize 只能在 allocate channel 中说明。

RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK clear;

第七行：CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default

备份集也可以有多路复用(但映像文件不可以),最大值 4,如为 2 就备份相同的 2 份(PPT101)，注意：TO STB 和 TO DISK 是独立的（PPT-102)，也就是说不能同时以 DISK 和 TAPE 多路复用（考点)。那么再问一句，如果一定要 duplex 磁盘和磁带，请使用 backup backupset.

单独使用 RMAN 命令可以覆盖这个参数

RMAN> backup copies 2 datafile 4 format '/u01/myrman/%s_dbf','/u01/myrman1/%s_dbf';

第八行：CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1; # default

归档日志的多路复用，类似数据文件多路复用

第九行：CONFIGURE MAXSETSIZE TO UNLIMITED; # default

该配置限制通道上备份集的最大尺寸，单位支持 Bytes 、KB、MB、GB，默认值是 unlimited，（前面提到过）

第十行： CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
第十一行：CONFIGURE ENCRYPTION ALGORITHM '', ''; # default

加密，Transparent encryption 透明的加密，带钱夹，Password encryption:不带钱夹，PPT-107

第十二行：CONFIGURE COMPRESSION ALGORITHM 'BZIP2'; # default

RMAN 里的压缩也叫 binary compression,这是 11g 新增的参数 PPT-II-104-106), 注意 rman 本来就不备份 Unallocated block(HWM 以上)，binary compression 是指压缩 Unused block, 另外，压缩只能是针对 backup set，有两种算法:'BZIP2'和'ZLIB'.不要将 RMAN 的压缩和外部压缩实用程序混合在一起做（考点）。

第十三行：CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default

对于归档文件，一般来说,如果你仅是单实例的话（不是 Data Guard 环境），archive file 备份完之后就没有什么用处了，可以删除，但很多人不习惯在这里设置参数，更愿意使用脚本删除归档文件。（见最后一章 Oracle 一周备份计划）

第十四行：CONFIGURE SNAPSHOT CONTROLFILE NAME TO '/u01/oracle/dbs/snapcf_timran11g.f'; # default

rman 控制文件 RMAN 存储库与 catalog 做全同步的时候,需要建立一个 controlfile 的快照，这个参数指定快照存放位置。(PPT-II-74)

6.6 rman 备份的分类：

1）backupset：不备份 datafile 里 unallocate 的块（HWM 以上）、可以实现增量备份（可以备份到 disk 和 tape）。如果只启用一个 channel，默认会把所有备份的 datafile 放入到一个 backupset 里，并且生成一个备份片（backup piece，在 OS 下看到）

2）copy（映像文件）：实际上和手工 cp 类似的，备份 datafile 所有的数据块,不能作为增量备份的基础（即 0 级增量备份），只能备份到 disk ，不能备份到 tape，因可以省略还原步骤，所有恢复速度快。

6.7 rman 的命令格式：

1）交互式 （也叫 stand alone 方式）

```
RMAN> shutdown immediate;
RMAN>                    startup                    force                    mount;
RMAN>                    alter                    database                    open;
RMAN> sql 'alter system switch logfile';

RMAN> sql 'select * from scott.emp';     //对 select 不显示结果
```

2）批处理方式（也叫 job 方式）

```
RMAN>run {
shutdown immediate;
startup mount;
allocate channel c1 type disk;
allocate channel c2 type disk;
backup database format '/u01/myrman/%d_%s.bak';
alter database open;
}
```

3)基于 EM 方式（WEB 方式）

第七章：rman 备份

7.1 归档方式下 rman 备份常用语法：

7.1.1 backupset 备份集

1）备份全库：

1.1 RMAN> backup database format='/u01/myrman/timran_%s.bak' filesperset 3;

1.2 RMAN> backup database plus archivelog delete input; //备份全库及控制文件、参数文件与所有归档日志，并删除旧的归档日志

1.3 RMAN> backup database format '/u01/myrman/%s_bak' plus archivelog delete input skip inaccessible;

注：关于 1.3 的说明

backup database 紧接 format 可以使 datafile 的备份片指定到 format 的目的地，archivelog 的备份是根据控制文件中（v$archived_log）中的内容导航的，如果控制文件中记录了而实际归档中又不存在，则会报错，skip inaccessible 的含义是跳过物理上缺失的日志文件。

当 delete input 后，控制文件相关信息（v$archived_log）也会被修改。

因为有了全备，随时可以还原备份，还原点之前的归档日志一般没有就什么用处了，如果想单独还原归档日志备份可以使用：
restore archivelog all;

考点：在使用 RMAN 时，不一定需要还原归档重做日志，RMAN 会自动根据恢复过程的需要应用归档重做日志。

2）备份表空间：
RMAN> backup tablespace users format '/u01/myrman/users_%s.bak' tag=userbak;
RMAN> backup tablespace system plus archivelog delete input; //备份指定表空间及归档日志，并删除旧的归档日志

3）备份数据文件
RMAN> backup datafile 3,5 format '/u01/myrman/%d_%s.bak'; //备份数据文件，可以多个，以“,”分开。

4）备份归档日志
RMAN> backup archivelog all delete input;

5）备份控制文件：
RMAN> backup current controlfile;

6）备份参数文件
RMAN> backup spfile;

7）备份闪回恢复区

RMAN> backup recovry area;

8) 备份备份集

RMAN> backup backupset 18; //将 disk 上的 backupset 备份一份到 tape 上，注意不产生新的备份集，类似 duplex.

RMAN> list backup; //列出 backup set

*考点：RMAN 从不备份联机重做日志文件、临时文件等。它只备份数据文件，控制文件，spfile，归档日志文件。

7.1.2 image 映像文件

1）RMAN> copy datafile 4 to '/u01/myrman/users_%s.bak';
2）RMAN> backup as copy tablespace 'TEST' format='/u01/myrman/%d_test_%s.bak';
//都用 backup 语法。统一格式

RMAN> list datafilecopy all; //列出映像集

*考点：RMAN 可在不执行 restore 情况下直接使用映像副本，而 backupset 在 recover 前必须先 restore。RMAN 映像副本备份不包括 spfile

7.2 增量备份:（见 PPT-117）（10g PPT-108）

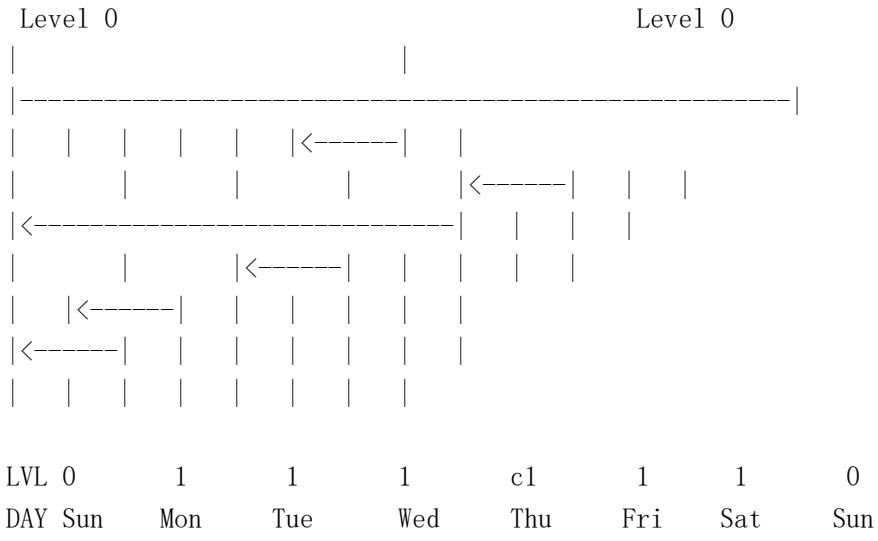
只备份自上次备份以来发生变化的 block（rman 在备份时会记录 datablock 的 scn，下次备份时通过比较前次备份数据块上的 scn 来确定该数据块是否发生变化）

1）差异增量备份（Differential incremental backup)：以上次以来同级别或低级别的备份作为基础备份

2）累积增量备份（Cumulative incremental backup)： 以上次以来比自己级别低的备份作为基础备份

提示：10g 以后，增量备份只有 0 级和 1 级，原来的 2 级以上的级别保留兼容，单不推荐使用。

7.2.1 差异和累计综合运用的一周备份计划。



*考点：
1）0 级增量备份不等同于全备(full backup),全备不能做为增量备份的基础备份，也就是说不能作为 1 级增量的父备份。
1）如果没有备份过 0 级备份，那么第一次 1 级备份（差异或累计）就当做 0 级备份。
2）非归档情况下的恢复就只有还原备份这一步，在 RMAN 下的方法是：

```
startup force mount;
restore database;
alter database open resetlogs;
```

如果非归档情况下使用了增量备份，在 restore 第二句后，加上 recover database noredo;这个命令有两个作用，1）定位所有级别为 1 的累计或差异增量备份并使用它们，2）不应用日志。(PPT-II-179)

7.3 块变更跟踪（block change tracking)（从 10G 开始引入）

增量备份总是小于完整备份，但是备份时节省的时间并不像想象的那样少，原因是增量备份默认的是要扫描整个要备份的数据文件块，以便确定哪些块需要提取，这当然有一个好处就是可以顺便检查坏块，如果想要增量备份能快些，oracle 还提供了一个机制，叫做块变更跟踪。
块变更跟踪在后台启动一个进程叫 CTWR，这个进程向变更跟踪文件中记录每个已更改的块的地址。如果启用了块跟踪，增量备份时就去检查块跟踪文件，这比遍历整个数据文件块要快的多。当然缺点是数据库维护块跟踪文件会付出性能代价，所以实际工作中此特性通常不在生产库上配置，而在 DATA GUARD 物理备库上配置则比较合适。

需要了解的是块改变跟踪使用一个文件，这个文件是不能有 RMAN 备份的，文件里记录了 8 个位图，实际上是对应 0 级增量和之后的 7 个差异增量备份，位图是循环利用的，会发生覆盖。

例：

1）使能块改变跟踪

```
SQL> alter database enable block change tracking using file '/u01/oradata/timran11g/change_tracking.dbf';
```

2）查看视图信息,确认已启动了 CTWR 进程，确认生成了块改变跟踪文件。

```
SQL> col filename for a50;
SQL> select * from v$block_change_tracking;

STATUS     FILENAME                                           BYTES
---------- -------------------------------------------------- ----------
ENABLED    /u01/oradata/timran11g/change_tracking.dbf          11599872
```

3）开始 0 级增量备份打底，用时 36 秒

```
RMAN> backup incremental level 0 format '/u01/myrman/%s.bak' datafile 2;
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:36
```

4）开始 1 级增量备份，用时 2 秒,时间已经大大缩短.

```
RMAN> backup incremental level 1 format '/u01/myrman/%s.bak' datafile 2;
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:02
```

5）关闭块改变跟踪

```
SQL> alter database disable block change tracking;
```

6) 开始一级增量备份，注意（没有 0 级备份，首次 1 级备份顶替 0 集备份,但这个备份未打底）用时 25 秒。

```
RMAN> backup incremental level 1 format '/u01/myrman/%s.bak' datafile 2;
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:25
```

7）再次使能块改变跟踪

```
SQL> alter database enable block change tracking using file
'/u01/oradata/timran11g/change_tracking.dbf';
```

8）用一级备份（打底），用时 25 秒

```
RMAN> backup incremental level 1 format '/u01/myrman/%s.bak' datafile 2;
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:25
```

9）再次一级备份，用时 1 秒,时间已经大大缩短.

```
RMAN> backup incremental level 1 format '/u01/myrman/%s.bak' datafile 2;
通道 ORA_DISK_1: 备份集已完成, 经过时间:00:00:01
```

考点：使能块改变跟踪后，需要一个相当于 incremental level 0 backup 做为基础备份（打底）。

7.4 增量更新备份(PPT-II-168-169)

增量更新特性是以一个 image copy（0 级或相当 0 级）为基础，将增量备份应用到这个 image copy，由于在做每日的 imcremental level 1 时， image copy 是不断的累积的，可以保证数据库需要还原时仅需要这个累积的 image copy 以及随后的 imcremental level 1。

```
RMAN>
run {
recover copy of database with tag 'incr_update';
backup incremental level 1 for recover of copy with tag 'incr_update' datafile 4;
}
```

这个例子每天执行一次，结果如何?

第一天,backup 命令将创建一个的相当于 level 0 的 image copy, 因为之前没有这个备份。
第二天，backup 命令创建一个 level 1 的 backup set。
第三天以及以后的每一天，recover copy 命令将把 level 1 应用到 image copy，从而不断的更新它。

//将 database 代替 datafile，便可以得到一个具有增量特性的 a whole backup。
//根据保留策略，如果 retention=1，多余的增量 level 1 将成为 obsolete。

7.5 数据恢复顾问 DRA(PPT-II-218-225)

DRA 是一个诊断和恢复数据库的工具，通过两个途径操作，一个是 RMAN 界面，另一个是 EM，它依赖于 ADR 的自动诊断信息以及 Health Monitor
例：

1）先以 RMAN 备份一个表空间

```
RMAN>backup tablespace sysaux format '/u01/myrman/%d_%s.bak';
```

2)关闭数据库，将 sysaux01.dbf 删掉。

```
SQL> shutdown abort

[oracle@timran timran11g]$ mv sysaux01.dbf sysaux01.bak

SQL> startup

ORA-01157: 无法标识/锁定数据文件 2 - 请参阅 DBWR 跟踪文件
ORA-01110: 数据文件 2: '/u01/oradata/timran11g/sysaux01.dbf'
```

3) 数据库在 mount 模式下连接 rman，看 DRA 给出的建议报告

```
[oracle@timran timran11g]$ rman target /

RMAN> list failure;

数据库故障列表
==========================

失败 ID 优先级状态      检测时间              概要
------- -------- --------- ------------------- -------
202      HIGH     OPEN      2013-10-25 19:09:09 缺失一个或多个非系统数据文件
```

优先级分为 CRITICAL | HIGH | LOW

CRITICAL 是关键文件损坏的状态，更改 CRITICAL 优先级会出现错误。只能将故障优先级从 HIGH 更改为 LOW 或从 LOW 更改为 HIGH。（从 HIGH 更改为 LOW 的一个原因是为了避免该故障在 LIST FAILURE 命令中显示出来）。

状态是 OPEN 表示打开了一个故障。待修复故障后，将隐式关闭打开的故障。

下面尝试使用 Change failure 命令可以改变 failure 的级别，

4)RMAN> change failure 202 priority low;

数据库故障列表
========================

失败 ID 优先级状态 检测时间 概要
------- -------- --------- ------------------- -------
202 HIGH OPEN 2013-10-25 19:09:09 缺失一个或多个非系统数据文件

是否确实要更改以上故障 (输入 YES 或 NO)? y
已将 1 个故障更改为 LOW 优先级

RMAN> list failure;

未找到与指定项匹配的故障

RMAN> change failure 202 priority high;

数据库故障列表
========================

失败 ID 优先级状态 检测时间 概要
------- -------- --------- ------------------- -------
202 LOW OPEN 2013-10-25 19:09:09 缺失一个或多个非系统数据文件

是否确实要更改以上故障 (输入 YES 或 NO)? y
已将 1 个故障更改为 HIGH 优先级

RMAN> list failure;

数据库故障列表
========================

失败 ID 优先级状态 检测时间 概要
------- -------- --------- ------------------- -------
202 HIGH OPEN 2013-10-25 19:09:09 缺失一个或多个非系统数据文件

5)RMAN> advise failure;
......
策略: 修复操作包括无数据丢失的完全介质恢复

修复脚本: /u01/diag/rdbms/timran11g/timran11g/hm/reco_3505864154.hm

6)RMAN> repair failure;

策略: 修复操作包括无数据丢失的完全介质恢复
修复脚本: /u01/diag/rdbms/timran11g/timran11g/hm/reco_1197260073.hm

修复脚本的内容:

```
# restore and recover datafile
restore datafile 2;
recover datafile 2;
```

是否确实要执行以上修复 (输入 YES 或 NO)? y
执行修复脚本

启动 restore 于 2013-03-12 14:19:36
使用通道 ORA_DISK_1

通道 ORA_DISK_1: 正在开始还原数据文件备份集
通道 ORA_DISK_1: 正在指定从备份集还原的数据文件
通道 ORA_DISK_1: 将数据文件 00002 还原到 /u01/oradata/timran11g/sysaux01.dbf
通道 ORA_DISK_1: 正在读取备份片段 /u01/myrman/TIMRAN11_1.bak
通道 ORA_DISK_1: 段句柄 = /u01/myrman/TIMRAN11_1.bak 标记 = TAG20130311T212425
通道 ORA_DISK_1: 已还原备份片段 1
通道 ORA_DISK_1: 还原完成, 用时: 00:00:45
完成 restore 于 2013-03-12 14:20:22

启动 recover 于 2013-03-12 14:20:22
使用通道 ORA_DISK_1

正在开始介质的恢复
介质恢复完成, 用时: 00:00:07

完成 recover 于 2013-03-12 14:20:29
修复故障已完成

是否要打开数据库 (输入 YES 或 NO)? y
数据库已打开

*考点:
1)先 list failure,然后再 advise failure.次序不能错,修复问题的顺序是 list,advise,repair
2)所有启动模式下都可以使用 DRA, nomount 下修复控制文件,mount 或 open 下修复数据文

件。
3)DRA 目前只能在单实例下运行，RAC 不可以使用它。

第八章：rman 完全恢复

8.1 recover 恢复：

1）归档 ： 完全恢复和不完全恢复
2）非归档：只能恢复到最后一次备份状态（还原）

8.2 完全恢复：

----先对数据库做一个冷备（如果是 archived 模式，热备也可以）

换一种形式，我们将下面的 run{} 写到 linux 的脚本中，叫做 /u01/oradata/timran11g/myrman.rcv

```
run {
allocate channel c1 type disk;
allocate channel c2 type disk;
backup database format '/u01/myrman/%d_%s.bak';
}
```

然后以 rman 登录时执行这个脚本。语法如下：

```
rman target / @/u01/oradata/timran11g/myrman.rcv
```

可以监控 RMAN 备份，RMAN 使用字符串 rman 和通道名称填充 v$session 中 client_info 列，每个 RMAN 通道有一行信息，备份结束后进程信息消失.
也可以使用 set command id to 'XXX'做一个标识，再结合 v$session 观察通道和进程之间的对应关系。

```
SQL>
select sid,spid,client_info
from v$process p join v$session s on (p.addr=s.paddr)
where client_info like 'rman%';

       SID SPID                     CLIENT_INFO
---------- ------------------------ ----------------------------------------------
--------------------
       113 12992                    rman channel=ORA_DISK_1
       111 12995                    rman channel=ORA_DISK_2

RMAN> list backup;
```

备份集列表
==========

BS 关键字 类型 LV 大小   设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
22   Full   635.16M   DISK   00:01:29   2013-01-15 15:49:14
        BP 关键字: 22   状态: AVAILABLE 已压缩: NO 标记: TAG20130115T154745
段名:/u01/myrman/TIMRAN11_24.bak
  备份集 22 中的数据文件列表
  文件 LV 类型 Ckp SCN   Ckp 时间   名称
  ---- -- ---- ---------- ------------------- ----
  1   Full   6634197   2013-01-15   15:47:29
/u01/oradata/timran11g/system01.dbf
  4   Full 6634197   2013-01-15 15:47:29 /u01/oradata/timran11g/user01.dbf
  6   Full 6634197   2013-01-15 15:47:29 /u01/oradata/timran11g/test01.dbf

BS 关键字 类型 LV 大小   设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
23   Full   621.61M   DISK   00:01:39   2013-01-15 15:49:27
        BP 关键字: 23   状态: AVAILABLE 已压缩: NO 标记: TAG20130115T154745
段名:/u01/myrman/TIMRAN11_25.bak
  备份集 23 中的数据文件列表
  文件 LV 类型 Ckp SCN   Ckp 时间   名称
  ---- -- ---- ---------- ------------------- ----
  2   Full   6634197   2013-01-15   15:47:29
/u01/oradata/timran11g/sysaux01.dbf
  5   Full   6634197   2013-01-15   15:47:29
/u01/oradata/timran11g/example01.dbf
  7   Full   6634197   2013-01-15   15:47:29
/u01/oradata/timran11g/undotbs01.dbf

BS 关键字 类型 LV 大小   设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
24   Full   9.67M   DISK   00:00:11   2013-01-15 15:49:44
        BP 关键字: 24   状态: AVAILABLE 已压缩: NO 标记: TAG20130115T154933
段名:/u01/myrman/c-3416564781-20130115-08
  包含的 SPFILE: 修改时间: 2013-01-15 15:47:44
  SPFILE db_unique_name: TIMRAN11G
  包括的控制文件: Ckp SCN: 6634197   Ckp 时间: 2013-01-15 15:47:29

RMAN>

说明: 两个通道 C1，C2 对应了两个备份集 22，23

范例 1：system 表空间损坏，使用 EM 的 DRA 恢复

1）环境

```
[oracle@timran ~]$ sqlplus scott/scott

SQL> create table t1 (id int);
SQL> insert into t1 values(1);
SQL> commit;
SQL> select * from t1;

        ID
----------
         1

SQL> alter system switch logfile;
SQL> /
SQL> /

SQL> insert into t1 values(2);
SQL> commit;
SQL> select * from scott.t1;

        ID
----------
         1
         2
```

3）在线删除 system01.dbf，模拟关键表空间损坏，然后启动数据库

```
[oracle@timran ~]$ rm /u01/oradata/timran11g/system01.dbf

SQL> startup froce
ORACLE instance started.

Total System Global Area  285212672 bytes
Fixed Size                  1218968 bytes
Variable Size              83887720 bytes
Database Buffers          192937984 bytes
Redo Buffers                7168000 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 1 - see DBWR trace file
```

ORA-01110: data file 1: '/u01/oradata/timran11g/system01.dbf'

使用 EM 恢复...

范例 2：恢复表空间（open 状态）。因非关键数据文件介质损坏，需要将其表空间恢复到一个新的物理位置。

1）环境

```
SQL> create table scott.emp1 as select * from scott.emp where rownum < 3;
SQL> select * from scott.emp1;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- -----
-----
      7369 SMITH      CLERK             7902 17-DEC-80        800
20
      7499 ALLEN      SALESMAN          7698 20-FEB-81       1600        300
30
```

2）模拟 users 表空间损坏，删除数据文件。

```
[oracle@timran ~]$ rm /u01/oradata/timran11g/users01.dbf
```

3) 清除 db buffer ，证实物理读失败

```
SQL> alter system flush buffer_cache;
SQL> conn /as sysdba
SQL> select * from scott.emp1;
select * from scott.emp1
                    *
ERROR at line 1:
```

SQL> alter system checkpoint　　//实验中为防备 rman 登录不正常，可以尝试先做个检查点切换

4）建个目录，假设介质损坏了，更换需要时间，先把数据文件恢复到一个新的目录下（不同的物理位置），

```
[oracle@timran timran11g]$ mkdir /u01/oradata/timran11g/dir1
```

5）使用 RMAN 恢复表空间

```
RMAN>run{
```

```
sql 'alter database datafile 4 offline';
set newname for datafile 4 to '/u01/oradata/timran11g/dir1/users01.dbf';
restore tablespace users;
switch datafile 4;
recover tablespace users;
sql 'alter database datafile 4 online';
}
```

说明：
set newname for 告诉 RMAN 还原数据文件的新位置在哪里。这个命令在 restore 前出现。
switch datafile 更新 controlfile，记录这个新位置。这个命令要在 recover 前出现。

5）验证

```
SQL> select * from scott.emp1;

     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
```

6）待介质更换完成后，可以将表空间迁移回原来位置。

```
SQL> alter tablespace users offline;

[oracle@timran       ~]$       mv       /u01/oradata/timran11g/dir1/users01.dbf
/u01/oradata/timran11g


SQL>alter          tablespace          users          rename          datafile
'/u01/oradata/timran11g/dir1/users01.dbf'                                  to
'/u01/oradata/timran11g/users01.dbf';
SQL> alter tablespace users online;
```

7）再验证

```
SQL> select * from scott.emp1;
```

范例 3：新建表空间(rman 备份没有这个表空间),datafile 被破坏

1）环境

```
SQL> create tablespace lx datafile '/u01/oradata/timran11g/lx01.dbf' size 5m;
SQL> create table scott.t2(id int) tablespace lx;
SQL> insert into scott.t2 values (1);
SQL> commit;
SQL> select * from scott.t2;

        ID
----------
         1
```

2) 删除 t2 表所在的数据文件

```
[oracle@timran ~]$rm /u01/oradata/timran11g/lx01.dbf
```

3）清除 db buffer，t2 表物理读失败。

```
SQL> alter system flush buffer_cache;
SQL> conn / as sysdba
SQL> select * from scott.t2;
select * from scott.t2
                    *
第 1 行出现错误:
ORA-01116: 打开数据库文件 3 时出错
ORA-01110: 数据文件 3: '/u01/oradata/timran11g/lx01.dbf'
ORA-27041: 无法打开文件
```

4）利用 rman 恢复数据文件（注意：rman 备份里没有 lx01.dbf）

```
RMAN>run {
sql 'alter database datafile 3 offline';
restore datafile 3;
recover datafile 3;
sql 'alter database datafile 3 online';
}
```

使用目标数据库控制文件替代恢复目录
sql 语句: alter database datafile 3 offline

启动 restore 于 2013-01-18 11:23:32
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: SID=130 设备类型=DISK

创建数据文件, 文件号 = 3 名称 = /u01/oradata/timran11g/lx01.dbf　　//对比手工恢

复，RMAN 自动建立了新的 lx01.dbf 文件
没有完成还原；所有文件均为只读或脱机文件或者已经还原
完成 restore 于 2013-01-18 11:23:35

启动 recover 于 2013-01-18 11:23:35
使用通道 ORA_DISK_1

正在开始介质的恢复
介质恢复完成，用时：00:00:01

完成 recover 于 2013-01-18 11:23:36

sql 语句：alter database datafile 3 online

5)验证
SQL> select * from scott.t2;

        ID

         1

第九章：rman 不完全恢复

9.1 rman 不完全恢复的三个标准模式：基于 time、基于 scn 和基于 sequence：

范例 1：恢复过去某个时间点误操作，一般使用基于 time 或 scn。

1）环境：做一套全备份

RMAN> show all;

CONFIGURE CONTROLFILE AUTOBACKUP ON; //控制文件自动备份，备份目的地是 flash_recovery_area

RMAN>delete backupset;
RMAN>backup database format '/u01/myrman/%s.bak'

SQL> create table scott.t1 (id int);
SQL> insert into scott.t1 values(1);
SQL> commit;
select * from scott.t1;
SQL> select * from scott.t1;

        ID

```
----------
         1

SQL> select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- -------------------
         1             1             7     52428800             1  NO   CURRENT
6689019 2013-01-18 13:45:25
         2             1             5     52428800             1  YES  INACTIVE
6689014 2013-01-18 13:45:22
         3             1             6     52428800             1  YES  INACTIVE
6689016 2013-01-18 13:45:23
```

2） 取时间

```
SQL> select sysdate from dual;

SYSDATE
-------------------
2013-01-18 13:47:04
```

3）

```
SQL> truncate table scott.t1;                //此动作记录在当前日志，即 sequence 7
里了。

SQL> alter system switch logfile;
SQL> /
SQL> select * from v$log;

    GROUP#        THREAD#    SEQUENCE#            BYTES        MEMBERS  ARC  STATUS
FIRST_CHANGE# FIRST_TIME
---------- ---------- ---------- ---------- ---------- --- ---------------- ----
--------- -------------------
         1             1             7     52428800             1  YES  ACTIVE
6689019 2013-01-18 13:45:25
         2             1             8     52428800             1  YES  ACTIVE
6689269 2013-01-18 13:49:17
         3             1             9     52428800             1  NO   CURRENT
6689271 2013-01-18 13:49:19
```

```
SQL> insert into scott.t1 values(2);              //此动作记录在当前日志，即
sequence 9里了。
SQL> commit;
SQL> select * from scott.t1;

        ID
----------
         2
```

4） RMAN按时间点做不完全恢复

```
run {
startup force mount;
allocate channel c1 type disk;
allocate channel c2 type disk;
set until time '2013-01-18 13:47:04 ';
restore database;
recover database;
alter database open resetlogs;
}

Oracle 实例已启动
数据库已装载

系统全局区域总计      422670336 字节

Fixed Size                  1300352 字节
Variable Size             331352192 字节
Database Buffers           83886080 字节
Redo Buffers                6131712 字节

使用目标数据库控制文件替代恢复目录
分配的通道: c1
通道 c1: SID=154 设备类型=DISK

分配的通道: c2
通道 c2: SID=151 设备类型=DISK

正在执行命令: SET until clause

启动 restore 于 2013-01-18 14:19:34

通道 c1: 正在开始还原数据文件备份集
通道 c1: 正在指定从备份集还原的数据文件
```

通道 c1: 将数据文件 00001 还原到 /u01/oradata/timran11g/system01.dbf
通道 c1: 将数据文件 00002 还原到 /u01/oradata/timran11g/sysaux01.dbf
通道 c1: 将数据文件 00003 还原到 /u01/oradata/timran11g/lx01.dbf
通道 c1: 将数据文件 00004 还原到 /u01/oradata/timran11g/user01.dbf
通道 c1: 将数据文件 00005 还原到 /u01/oradata/timran11g/example01.dbf
通道 c1: 将数据文件 00006 还原到 /u01/oradata/timran11g/test01.dbf
通道 c1: 将数据文件 00007 还原到 /u01/oradata/timran11g/undotbs01.dbf
通道 c1: 正在读取备份片段 /u01/myrman/51.bak
通道 c1: 段句柄 = /u01/myrman/51.bak 标记 = TAG20130118T123557
通道 c1: 已还原备份片段 1
通道 c1: 还原完成, 用时: 00:01:55
完成 restore 于 2013-01-18 14:21:30

启动 recover 于 2013-01-18 14:21:31

正在开始介质的恢复

线程 1 序列 4 的归档日志已作为文件 /u01/disk1/timran/arch_1_804846837_4.log 存在于磁盘上
线程 1 序列 5 的归档日志已作为文件 /u01/disk1/timran/arch_1_804846837_5.log 存在于磁盘上
线程 1 序列 6 的归档日志已作为文件 /u01/disk1/timran/arch_1_804846837_6.log 存在于磁盘上
线程 1 序列 7 的归档日志已作为文件 /u01/disk1/timran/arch_1_804846837_7.log 存在于磁盘上
归档日志文件名=/u01/disk1/timran/arch_1_804846837_4.log 线程=1 序列=4
归档日志文件名=/u01/disk1/timran/arch_1_804846837_5.log 线程=1 序列=5
归档日志文件名=/u01/disk1/timran/arch_1_804846837_6.log 线程=1 序列=6
归档日志文件名=/u01/disk1/timran/arch_1_804846837_7.log 线程=1 序列=7
介质恢复完成, 用时: 00:00:08
完成 recover 于 2013-01-18 14:21:41

数据库已打开
释放的通道: c1
释放的通道: c2

4）验证

```
SQL> select * from scott.t1;

        ID
----------
         1
```

//基于 scn 的方法与基于 time 相似，语法是 set until scn 6689163;

//基于日志的方法类似手工恢复的例子，语法是 set until sequence 3;

*考点：不完全恢复的手工与 RMAN 语法比较：

	手工方法	RMAN 方法
基于 time	until time XXX	set until time XXX
基于 scn	until change XXX	set until scn XXX
基于日志	until cancel	set until sequence XXX

范例 2

恢复 SPFILE 或 CONTROLFILE:

1）环境

```
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
         2 UNDOTBS1                       YES NO  YES
```

//TEST 表空间里有 T1 表

```
SQL> select owner,table_name,tablespace_name from dba_tables where tablespace_name='TEST';

OWNER                          TABLE_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------ ------------------
------------
SCOTT                          T1                             TEST
```

//T1 表里有一条记录

```
SQL> select * from scott.t1;
```

```
        ID
----------
         1
```

rman 的情况：

```
RMAN> list backup;

备份集列表
==========

BS 关键字  类型 LV 大小       设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
1       Full    1.27G      DISK        00:01:42     2013-01-16 19:36:18
        BP 关键字: 1   状态: AVAILABLE  已压缩: NO  标记: TAG20130116T193436
段名:/u01/myrman/2.bak
  备份集 1 中的数据文件列表
  文件 LV 类型 Ckp SCN    Ckp 时间             名称
  ---- -- ---- ---------- -------------------- ----
  1                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/system01.dbf
  2                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/sysaux01.dbf
  4       Full 6698790    2013-01-16 19:34:45 /u01/oradata/timran11g/user01.dbf
  5                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/example01.dbf
  6       Full 6698790    2013-01-16 19:34:45 /u01/oradata/timran11g/test01.dbf
  7                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/undotbs01.dbf

BS 关键字  类型 LV 大小       设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
2       Full    9.67M      DISK        00:00:10     2013-01-16 19:36:31
        BP 关键字: 2   状态: AVAILABLE  已压缩: NO  标记: TAG20130116T193621
段
名:/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x
9y118z_.bkp
  包含的 SPFILE: 修改时间: 2013-01-16 19:29:09
  SPFILE db_unique_name: TIMRAN11G
  包括的控制文件: Ckp SCN: 6698846       Ckp 时间: 2013-01-16 19:36:21
```

//由于控制文件自动备份的关系，spfile file 和 controle file 打包在一个备份集 2 里。

3）得到数据库唯一标识号：DBID

```
SQL> select dbid from v$database;

      DBID
----------
3416564781
```

//dbid 是你的 database 的一个唯一识别 ID，恢复 spfile 和 controlfile 时候都要用到。这个信息在 rman 加载数据库时也可以得到。

4）关闭数据库，然后让参数文件不起作用

```
SQL> shutdown abort

[oracle@timran dbs]$ mv spfiletimran.ora spfiletimran.old
[oracle@timran dbs]$ mv inittimran.ora  inittimran.old
```

//模拟 spfile 损坏，pfile 也不能起作用。

5) RMAN 恢复参数文件

```
[oracle@timran ~]$ [oracle@timran ~]$ rman target /

connected to target database (not started)

RMAN> startup nomount;     //没有了参数文件，SQL*PLUS 是无法启动实例的，但 RMAN 可以，所以 startup nomount 一定要在 RMAN 下做!!!

startup failed: ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file '/u01/oracle/dbs/inittimran.ora'

starting Oracle instance without parameter file for retrival of spfile
Oracle instance started

Total System Global Area     159383552 bytes

Fixed Size                     1218244 bytes
Variable Size                 58722620 bytes
Database Buffers              92274688 bytes
Redo Buffers                   7168000 bytes

RMAN> set dbid=3416564781;
```

executing command: SET DBID

RMAN> restore spfile from autobackup;

//执行该命令，如果没有找到的话，那可能是文件的路径发生错误.可以通过直接赋予它的物理路径及文件名

RMAN> restore spfile from '/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y118z_.bkp';

启动 restore 于 2013-01-16 14:43:42
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: SID=100 设备类型=DISK

通道 ORA_DISK_1: 正在从 AUTOBACKUP /u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y118z_.bkp 还原 spfile
通道 ORA_DISK_1: 从 AUTOBACKUP 还原 SPFILE 已完成
完成 restore 于 2013-01-16 14:43:46

//查看在 dbs/目录下已经产生 spfiletimran.ora 文件。证明 spfile 恢复好了。

控/参两文件恢复方法类似，每个数据库有一个唯一的 dbid，保存在控制文件中(不在参数文件)，恢复参/控两文件都要首先 set dbid（考点）

RMAN 的元数据保存在控制文件中，要使用 RMAN 就必须有控制文件，Oracle 设计了自动备份控制文件功能，

当 CONFIGURE CONTROLFILE AUTOBACKUP ON 时，控制文件将自动保存在指定的 flash_recovery_area 目录下，以支持恢复时使用：

RMAN> restore controlfile from autobackup 命令

范例 3 恢复误删除表空间（已备份），RMAN 必须通过备份的控制文件（与之配套的老控制文件）进行恢复。

本例要做的是 drop tablespace test，然后再通过不完全恢复，使数据库在 drop 表空间前的那一刻打开，从而恢复 test 表空间及 t1 表的内容。

1）$ tail -f /u01/diag//rdbms/timran11g/timran11g/trace/alert_timran11g.log

//打开告警日志，查看 drop tablespace 的告警信息，记下时间点

2）SQL> drop tablespace test including contents and datafiles;

//删除 test 表空间

3）查看告警有关信息：

```
Wed Jan 16 19:39:56 2013                    //这个时间是你要 until time 的时刻
drop tablespace test including contents and datafiles
Deleted file /u01/oradata/timran11g/test01.dbf
Wed Jan 16 19:40:12 2013
Completed: drop tablespace test including contents and datafiles
```

4）查看 rman 备份集信息

```
RMAN> list backup;

备份集列表
===================

BS 关键字  类型 LV 大小       设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
1       Full    1.27G      DISK        00:01:42     2013-01-16 19:36:18
        BP 关键字: 1   状态: AVAILABLE  已压缩: NO  标记: TAG20130116T193436
段名:/u01/myrman/2.bak
  备份集 1 中的数据文件列表
  文件 LV 类型 Ckp SCN    Ckp 时间             名称
  ---- -- ---- ---------- -------------------- ----
  1                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/system01.dbf
  2                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/sysaux01.dbf
  4       Full 6698790    2013-01-16 19:34:45 /u01/oradata/timran11g/user01.dbf
  5                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/example01.dbf
  6       Full 6698790    2013-01-16 19:34:45
  7                       Full    6698790              2013-01-16    19:34:45
/u01/oradata/timran11g/undotbs01.dbf

BS 关键字  类型 LV 大小       设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
2       Full    9.67M      DISK        00:00:10     2013-01-16 19:36:31
        BP 关键字: 2   状态: AVAILABLE  已压缩: NO  标记: TAG20130116T193621
```

段
名:/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y1l8z_.bkp
  包含的 SPFILE: 修改时间: 2013-01-16 19:29:09
  SPFILE db_unique_name: TIMRAN11G
  包括的控制文件: Ckp SCN: 6698846   Ckp 时间: 2013-01-16 19:36:21

BS 关键字 类型 LV 大小   设备类型 经过时间 完成时间
------- ---- -- ---------- ----------- ------------ -------------------
3   Full   9.67M   DISK   00:00:10   2013-01-16 19:40:12
        BP 关键字: 3   状态: AVAILABLE 已压缩: NO 标记: TAG20130116T194002
段
名:/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y1l9z_.bkp
  包含的 SPFILE: 修改时间: 2013-01-16 19:29:09
  SPFILE db_unique_name: TIMRAN11G
  包括的控制文件: Ckp SCN: 6698986   Ckp 时间: 2013-01-16 19:40:02

//备份集 1 中的数据文件 6 已经空了，这是删除了 test 表空间所致。又增加了备份集 3，这是自动备份控制文件设为 on 所致。

5）删除所有控制文件和数据文件

```
SQL> shutdown abort
[oracle@timran timran11g]$ rm *.ctl
[oracle@timran timran11g]$ rm *.dbf
```

6）准备对 drop tablespace test 做不完全恢复

三个要点：

1.新增了 3 号备份集，其中的控制文件备份是由于 drop tablespace 这个动作之后产生的，这个控制文件里已经没有 test 表空间的记录了。

2.在恢复控制文件语句 restore controlfile from autobackup 后面加上 until time XXX. 给出时间点，RMAN 会找备份集 2，不会使用备份集 3.也就是说 RMAN 会自动选取正确的“老”控制文件恢复。

3.当你 restore 控制文件后，下一步是进入 mount 状态（考点），然后才能做不完全恢复，使用老控制文件，比较一下手工恢复方式，RMAN 里是不使用 using backup controlfile 子句的。

```
RMAN>run{
startup nomount;
set dbid=3416564781;
restore controlfile from autobackup until time '2013-01-16 19:39:56';
alter database mount;
set until time '2013-01-16 19:39:56';
restore database;
recover database;
alter database open resetlogs;
}


Oracle 实例已启动

系统全局区域总计      422670336 字节

Fixed Size                  1300352 字节
Variable Size             352323712 字节
Database Buffers           62914560 字节
Redo Buffers                6131712 字节


正在执行命令: SET DBID

启动 restore 于 2013-01-16 19:44:47
使用目标数据库控制文件替代恢复目录
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: SID=154 设备类型=DISK

恢复区目标: /u01/flash_recovery_area
用于搜索的数据库名 (或数据库的唯一名称): TIMRAN11G
通道 ORA_DISK_1: 在恢复区中找到 AUTOBACKUP
/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y1
18z_.bkp
通道 ORA_DISK_1: 寻找以下日期的 AUTOBACKUP: 20130116
通道 ORA_DISK_1: 正在从 AUTOBACKUP
/u01/flash_recovery_area/TIMRAN11G/autobackup/2013_01_16/o1_mf_s_819894065_8x9y1
18z_.bkp 还原控制文件
通道 ORA_DISK_1: 从 AUTOBACKUP 还原控制文件已完成
输出文件名=/u01/oradata/timran11g/control01.ctl
输出文件名=/u01/oradata/timran11g/control02.ctl
输出文件名=/u01/oradata/timran11g/control03.ctl
完成 restore 于 2013-01-16 19:45:28
```

数据库已装载
释放的通道: ORA_DISK_1

正在执行命令: SET until clause

启动 restore 于 2013-01-16 19:45:33
启动 implicit crosscheck backup 于 2013-01-16 19:45:33
分配的通道: ORA_DISK_1
通道 ORA_DISK_1: SID=154 设备类型=DISK
已交叉检验的 1 对象
完成 implicit crosscheck backup 于 2013-01-16 19:45:34

启动 implicit crosscheck copy 于 2013-01-16 19:45:34
使用通道 ORA_DISK_1
完成 implicit crosscheck copy 于 2013-01-16 19:45:34

搜索恢复区中的所有文件
正在编制文件目录...
没有为文件编制目录

使用通道 ORA_DISK_1

通道 ORA_DISK_1: 正在开始还原数据文件备份集
通道 ORA_DISK_1: 正在指定从备份集还原的数据文件
通道 ORA_DISK_1: 将数据文件 00001 还原到 /u01/oradata/timran11g/system01.dbf
通道 ORA_DISK_1: 将数据文件 00002 还原到 /u01/oradata/timran11g/sysaux01.dbf
通道 ORA_DISK_1: 将数据文件 00004 还原到 /u01/oradata/timran11g/user01.dbf
通道 ORA_DISK_1: 将数据文件 00005 还原到 /u01/oradata/timran11g/example01.dbf
通道 ORA_DISK_1: 将数据文件 00006 还原到 /u01/oradata/timran11g/test01.dbf
//test 表空间已还原
通道 ORA_DISK_1: 将数据文件 00007 还原到 /u01/oradata/timran11g/undotbs01.dbf
通道 ORA_DISK_1: 正在读取备份片段 /u01/myrman/2.bak
通道 ORA_DISK_1: 段句柄 = /u01/myrman/2.bak 标记 = TAG20130116T193436
通道 ORA_DISK_1: 已还原备份片段 1
通道 ORA_DISK_1: 还原完成, 用时: 00:01:46
完成 restore 于 2013-01-16 19:47:21

启动 recover 于 2013-01-16 19:47:22
使用通道 ORA_DISK_1

正在开始介质的恢复

线程 1 序列 2 的归档日志已作为文件 /u01/oradata/timran11g/redo02.log 存在于磁盘上

归档日志文件名=/u01/oradata/timran11g/redo02.log 线程=1 序列=2
介质恢复完成，用时：00:00:02
完成 recover 于 2013-01-16 19:47:25

数据库已打开

7）验证：
SQL> select * from scott.t1;

```
        ID
----------
         1
```

范例 4 表空间时间点恢复（TableSpace Point In Time Recovery）

作为一条基本原则，不完全恢复必须应用到整个数据库，即必须还原整个数据库并运用日志一起向前滚动。TSPITR 是一种对个别表空间执行不完全恢复的技术，一般是针对用户错误的删除（或截断）了表。TSPITR 通过 RMAN 创建一个辅助库，将单个表空间在辅助库上恢复到指定的某个时刻，因为是在辅助库恢复，目标库（生产库）不用停机。

在一系列准备工作完成后（建立辅助实例，网络连接等），利用 RMAN 同时连接目标数据库和新启动的辅助实例,如：

$rman target sys/oracle@timran11g auxiliary sys/oracle@newdb

我下面的测试没有连接 auxiliary instance ，就是在 target database 上完成的，我们仅仅是了解一下这个过程。

前提：

1）有一套全库备份，因为 TSPITR 这个过程也要在辅助库上恢复目标库的 system 和 undo 表空间（考点）
2）本例设置了控制文件自动备份：RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;

1)建表空间
SQL> create tablespace abcd datafile '/u01/oradata/timran11g/abcd01.dbf' size 5m;

2)建表
```
create table scott.t2(c1 date) tablespace abcd;
insert into scott.t2 values(sysdate);
commit;
```

3)RMAN 备份表空间

```
RMAN>backup tablespace abcd format '/u01/myrman/abcd_%s.bak';
```

4）取当前时间

```
SQL>select sysdate from dual;

SYSDATE
-------------------
2013-01-16 22:02:14
```

5)删除表并 purge

```
SQL>drop table scott.t2 purge;
```

6)建立目录指定辅助库目标

```
$mkdir -p /u01/oradata/timran11g/auxdata
```

7)做 RMAN TSPITR 并指定辅助库目的地（这里还是目标库）

```
[oracle@timran ~]$ rman target /

RMAN> recover tablespace abcd until time '2013-01-16 22:02:14' auxiliary destination '/u01/oradata/timran11g/auxdata';
```

执行过程值得一读，但太长了，略了。当命令结束后，如果没有报错，接下来应该对 abcd 表空间做一个备份，因为之前 abcd 表空间备份不能再做 TSPITR 用了（略）。

8)验证

```
SQL> alter tablespace abcd online;
SQL> select * from scott.t2;

C1
-------------------
2013-01-16 21:58:11
```

考点：
1）TSPITR 是保持在线业务下表空间级的不完全恢复，恢复的表空间要自包含。使用 TS_PITR_CHECK 视图查看自包含信息。
2) TSPITR 一般用于将已有的表空间恢复到过去的某个时间点，而 11gR2 版使用可传输表空

间和数据泵技术，Oracle 声明可以恢复被删除的表空间。（PPT-II-563）
3) 通过 TS_PITR_OBJECTS_TO_BE_DROPPED 视图，查看 creation time 字段可以知道截止恢复时间之后的有哪些新建对象可能丢失了。
4）完成指定表空间的 TSPITR 后，这个表空间之前做的备份就不能再用于以后的 TSPITR。 这就是 TSPITR 后的表空间备份操作非常重要的原因。

范例 5：数据块介质恢复（BMR）（PPT-II-229-236）

如果数据文件只是出现部分数据块损坏，RMAN 可以尝试针对坏块进行恢复，就是说不需要恢复整个数据文件，只恢复损坏的数据块。为了保证数据文件的一致性，块介质恢复是一个完全恢复。

介质损坏，Block format incorrect，checksum 无效
逻辑损坏，Oracle internal error，checksum 有效

很多方法可以发现坏块(考点)：
A. ANALYZE operations
B. dbv
C. SQL queries that access the potentially corrupt block
D. DBMS_REPAIR
F. RMAN

通过设置参数，当对块读写时进行检查

DB_BLOCK_CHECKSUM 检查 disk I/O 讹误，缺省 TYPICAL。
DB_BLOCK_CHECKING 防止 memory 和 data 讹误，缺省 FALSE。
DB_LOST_WRITE_PROTECT lost write 与 standy database 的延迟永久写有关，缺省 TYPICAL。

11g 新出了一个参数，顶替了上面三个参数，它就是 DB_ULTRA_SAFE(考点)。
该参数可选项：OFF（缺省）|DATA_ONLY|DATA_AND_INDEX

RMAN 在 BACKUP 时，可以监测到损坏的数据块，并且自动将坏块记录 V$database_block_currption 中。

如果是某个查询会话击中了坏块，可能会有下面的报错：
ORA-01578: ORACLE DATA BLOCK CURRPTED(FILE #5,BLOCK #21)

对应的 rman 恢复命令应该是：

RMAN>blockrecover device type disk datafile 5 block 21;

11g 新特性增加 RMAN>recover corruption list，它和 backup validate database 配合使

用，为的是简化数据文件坏块恢复。

例：
RMAN>BACKUP VALIDATE DATABAE;
RMAN>RECOVER CORRUPTION LIST;

BACKUP VALIDATE DATABAE 不生成备份集，它只是检查坏块，并确认所有数据文件在正确的位置存在，这条命令会在 v$backup_corruption 和 v$database_block_corruption 视图中填充检测到的所有讹误数据块。

例：
RMAN> BACKUP CORRUPTION LIST UNITL TIME SYSDATE - 7;

注意：RMAN 的块恢复是完全恢复，这里的 UNTIL 表示的是要使用一周前的备份做还原，它不代表不完全恢复。

考点会围绕以下知识点：

1）BMR 是 RMAN 的一个功能，它是针对物理损坏的 block 进行的完全恢复，要求数据库是 ARCHIVELOG 模式。
2）RMAN 是块级备份，备份时自动检测坏块，缺省下碰到坏块就会中断备份，除非 set maxcorrupt
3）做 BMR 有两点前提：一个是数据库要在 mount 或 open 下，另一个是要有坏块所在的数据文件 incremental level 0 备份。
4）闪回日志也可以帮助恢复坏块。

SQL> desc V$DATABASE_BLOCK_CORRUPTION;
 名称 是否为空? 类型
 --- -------- --
 FILE# NUMBER
 BLOCK# NUMBER
 BLOCKS NUMBER
 CORRUPTION_CHANGE# NUMBER
 CORRUPTION_TYPE VARCHAR2(9)

如果 CORRUPTION_TYPE 列记录了 MEDIA_CORRUPT，可以使用下面命令尝试修复。

RMAN> recover corruption list;

范例 6：归档备份(PPT-II-127)

概念：在 Oracle 11g 中，可以使用 backup … keep 命令保留比 RMAN 备份保留策略所指定的时间更长的备份。它可以构造能保留数年的备份(不被 obsolete)，归档备份之所能使备份长久保存，关键是使用 keep 选项覆盖掉了 RMAN 中缺省的保留策略。

语法：BACKUP ... KEEP {FOREVER|UNTIL TIME 'date_expt'} [RESOTRE POINT rename];

说明：

KEEP FOREVER 永不过期（需要 catalog）在 Oracle 11g 中
KEEP UNTIL TIME 跟一个时间长度，如表示一年，'sysdate+365'

RESOTRE POINT 是将备份后的 SCN 保存在控制文件中。以便恢复时的数据一致性。

例：

```
RMAN>
backup datafile 4 format '/u01/myrman/%s_%t.bak'
keep until time 'sysdate+30'
restore point timranbak;
```

可以查看 RESOTRE POINT rename 和 scn 的对应关系。

```
RMAN> list restore point all;

SCN               RSP 时间            类型       时间                 名称
----------------- ------------------- ---------- -------------------- ----
13400758                              2014-03-13 10:10:18 TIMRANBAK
```

考点：
1）归档备份不受 retention 策略影响，也不会由 delete obsolete 命令自动删除掉。
2）归档备份是包含一切的备份，共四种类型：数据文件备份，spfile 备份，control 备份和归档日志备份。

```
RMAN>
```

第十章：目录库和辅助库

10.1 创建目录库（Catalog database）的必要性

如果没有 catalog，RMAN 的存储库（元数据）保存在目标库的控制文件里，这样可能存在如下隐患

1）目标库上的控制文件损坏，造成该目标库上 RMAN 元数据丢失。尽管你做了数据备份，但如果没有 autobackup，RMAN 就很难找那些备份集了。
2）RMAN 元数据受控制文件参数 control_file_record_keep_time 限制，缺省是 7 天，超时可能被覆盖，即造成老化的元数据丢失。

对此，我们可以在另一台服务器上建立一个 Catalog Database

优点是：
1）它多了一份目标库的 RMAN 元数据副本。
2）它不受时间限制，不会被覆盖，理论上是无限期的存储 rman 元数据历史信息。
3）使用一个 catalog 目录库就可以管理许多目标库，所谓集中管理多个目标库，这些目标库可以是不同平台，不同版本。
4）它可以使用脚本 catalog script 这是 catalog 独有的功能，使 RMAN 的备份与恢复更加灵活，自动化。

10.2 Catalog database 的配置

catalog 目录库在生产环境中一定要建在一台独立的 Oracle server 上， 不要建在同一台目标机上。

配置一个 Catalog 的简单操作：catalog :xp-oracle-orcl target:linux-oracle-timran11g

1）创建 Catalog 所用表空间

```
SQL> create tablespace rman_ts datafile 'E:\rman.dbf' size 50m;
```

2）创建 RMAN 用户并授权

```
SQL> create user rman identified by rman default tablespace rman_ts;
SQL> grant resource,recovery_catalog_owner to rman;
SQL> exit
```

3）从 RMAN 客户端上同时连接 target 库和 catalog 库

```
C:>rman target sys/system@timran11g catalog rman/rman@orcl
```

连接到目标数据库：TIMRAN(DBID=4035750304)
连接到恢复目录数据库

4）创建 catalog 目录，（建了一大堆表）

```
RMAN>create catalog tablespace rman_ts;
```

恢复目录已创建

//这一步 Oracle 到底做了些什么？你可以使用 rman 用户登录 catalog 查看 user_objects 视图，比较显示结果。

xp 下 sqlplus rman/rman
SQL> select object_name,object_type from user_objects;

5）注册目标库，（登记你连接上的那个 target，把 target 上控制文件中的 rman 信息同步到 catalog 里）
RMAN>register database;

注册在恢复目录中的数据库
正在启动全部恢复目录的 resync
完成全部 resync

至此创建 Catalog 过程完毕，可以测试一下结果。

一般而言,每次连接 target 和 catalog 时 target 库的控制文件都会自动刷新到 Catalog 库。但如果 target 库结构有改变，或在备份期间有大量 online 日志归档，Catalog 这边可能没有自动更新，必要时做一下手动同步。这种同步是整体的刷新，也叫 full resync.

RMAN> RESYNC CATALOG;

10.3 catalog 连接 targe 方法 t

有三种方法：

第一种是我们前面已经用过,在连接 catalog 时同时连接 target

第二种先进入 RMAN，再分别 connect target|catalog

第三种是先进入 catalog，然后再使用 connect target 命令连接 target

演示一下第三种方法:

现在有两个 target,一个是 timran11g(linux)， 另一个是 orcl(xp)， 看看 catalog 如何连他们（只能分别连）

1）使用 rman 连接到 catalog

C:\Documents and Settings\timran>rman catalog rman/rman@orcl

连接到恢复目录数据库

2）连接到 timran11g

```
RMAN> connect target sys/system@timran11g

连接到目标数据库: TIMRAN11 (DBID=3439065160)

RMAN> connect target sys/system@orcl;

RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-06167: 已经连接

//说明 catalog 一次只能连一个 target（考点）
RMAN> exit

3) 连接到 orcl
RMAN> connect target sys/system@orcl;

连接到目标数据库: ORCL (DBID=1335748581)

RMAN> register database;

注册在恢复目录中的数据库
正在启动全部恢复目录的 resync
完成全部 resync
```

10.4 RMAN catalog 的存储脚本

非 catalog 的 rman 脚本可以作为操作系统文件来存储，比如：将下面的两个命令保存到一个名为 timran.rcv 文件中

```
run{
backup database plus archivelog delete all input;
delete obsolet;
}
```

那么可以在操作系统下调用该脚本

```
$rman target / catalog rman/rman@orcl @timran.rcv
```

但是如果有 catalog 目录的话，可以在 rman catalog 里存储脚本

有关脚本的命令有：

```
create [global] script
```

```
replace [global] script
print [global] script
list [global] script name
exectue [global] script
delete [global] script
```

10.5 导入 catalog database(考点）(PPT-II-83)

可能有多个 catalog 的环境， 一些 catalog 针对来自不同 Oracle 版本的 target,可以使用 IMPORT CATALOG 将这些目录合并为一个目录。默认时，源 catalog 中注册的所有 target 的元数据导入到目标 catalog， 也可以指定想要导入到目标 catalog 的 target，而且默认的 RMAN 不注册从源 catalog 导入的 target，可以通过 NO UNREGISTER 子句添加到 IMPORT CATALOG 命令使这些 target 自动注册。

例：使用 IMPORT CATALOG 命令合并两个 catalog（将 10g 版本的 catalog 合并到 11g 版本 catalog）

```
$rman
```

第一步，连接到目标恢复目录，（11g 版本）

```
RMAN> connect catalog rman/rman@rman11
```

第二步，发布 import catalog 命令，将源 catalog（10g)与目标 catalog（11g)连接，并将所有源 catalog 上的 target database 导入目标 catalog

```
RMAN> import catalog rman1/rman1@rman10
```

也可以指定要导入的 target database

```
RMAN> import catalog rman1/rman1@rman10 dbid=123456,1234557;
```

10.6 Catalog command

目的是对存储库进行编目，有这样一种情形，RMAN 元数据受损，我们手里只有备份，那可以根据备份的物理路径重新编目 RMAN 元数据。这种方法可以将早期的老化的备份，重新（要有磁盘路径）注册到 target controlfile 中，也可以完成向一个空的 catalog 中注册元数据。

常见的语法形式有：

```
CATALOG DATAFILECOPY '/disk1/old_datafiles/01_01_2003/users01.dbf';
CATALOG ARCHIVELOG '/disk1/arch_logs/archive1_731.dbf',
                   '/disk1/arch_logs/archive1_732.dbf';
CATALOG BACKUPPIECE '/disk1/backups/backup_820.bkp';
```

还可以一次完成一个目录树下的所有文件的注册: 通过提供一个 OS 路径，RMAN 自动导航找到要编目的所有备份片（包括其子目录）

CATALOG START WITH '/disk1/backups/';

范例，catalog 下使用 image copy 快速恢复数据文件（不需要 restore）(PPT-II-170)

RMAN> backup as copy datafile 4 format '/u01/myrman/%s.dbf';

SQL> select file#,name from v$datafile;　　//看一下控制文件中 datafile 4 标识的物理路径

[oracle@timran timran11g]$ rm /u01/oradata/timran11g/users01.dbf

RMAN> sql 'alter database datafile 4 offline';

RMAN> switch datafile 4 to copy;

//switch 命令更改了控制文件相关信息并触发 resync catalog。

RMAN> recover datafile 4;

RMAN> sql 'alter database datafile 4 online';

如果还想恢复原状可以再使用 SQL*PLUS
SQL>alter tablespace users offline;
[oracle@timran timran11g]$ cp /u01/myrman/3.dbf /u01/oradata/timran11g/users01.dbf
SQL>alter tablespace users rename datafile '/u01/myrman/3.dbf' to '/u01/oradata/timran11g/users01.dbf';
SQL>alter tablespace users online;

考点：
1）到目标库的 RMAN 连接通常是 SYS 用户，因为一般需要在 RMAN 里发出启动或关闭数据库，但是不需要指定 AS SYSDBA。
2）创建 catalog 目录的版本必须高于或等于任何目标库的数据库版本，因此可以创建单个版本为 11.1.0.6 的目录加补丁集 6。
3)使用 create script 建立本地脚本,它只让你连接单个目标库,使用 create global script 命令创建全局存储脚本，它适用于所有目标库，　但不能同时对多个目标库执行命令。本地脚本和全局脚本都保存在 RMAN catalog 里。
4）迁移过来的 global script 如果有重名出现，RMAN 会重命名它们。
5)使用 exectue 去执行脚本,它必须包含在 RUN{}。可以参考笔记最后的附录部分,有《 Oracle

一周备份计划范例》。

10.7 虚拟专用目录（PPT85-86）

一个 catalog 可以注册你单位的所有数据库的详细信息，当 target 服务器比较多，如果让某个 DBA 只能查看 CATALOG 中某些元数据，那么可以利用 Oracle11 提供的虚拟专用目录(virtual private catalog)，你可以通过只授予 CATALOG 目录中某个元数据子集的访问权，严格限制对 CATALOG 目录的访问。

与虚拟专用目录(virtual private catalog)概念相关的是基恢复目录(base recovery catalog)。使用虚拟专用目录的前提是必须有基目录。这两种类型的目录差别只是虚拟专用目录拥用只能访问他们有访问权限的那些数据库，而基目录拥有者可以访问所有注册的数据库。（考点）

一道考题：考点是建立虚拟目录的步骤：
条件：
基目录(base catalog)下有数据库：PROD1, PROD2, PROD3
用户 CATOWNER 有基目录权限，希望让新用户 VPC1 仅仅管理 PROD1
问：如何建立虚拟目录(virtual catalog)

1.SQL> GRANT recovery_catalog_owner TO vpc1; //为 VPC1 用户授权
2.RMAN> CONNECT CATALOG catowner/password@catdb; //CATOWNER 用户登录 catalog server
3.RMAN> GRANT CATALOG FOR DATABASE prod1 TO vpc1; //为 VPC1 指定 PROD1 的管理权限
4.RMAN> CONNECT CATALOG vpc1/password@catdb; //VPC1 用户登录 catalog server
5.RMAN> CREATE VIRTUAL CATALOG; //VPC1 用户建立 virtual catalog

10.8 复制数据库 Duplicate Database

10.8.1 名词定义：
Source Database （或 source host）源数据库
Duplicate Database （或 new host）新复制的数据库
Catalog Database （或 catalog host） 目录库

10.8.2 Duplicate Database 有手工方法和 RMAN 方法，RMAN Duplicate 可以有两个用途：

1）TEST 用途的 Database

就是 source host 的一个 copy, RMAN 通过不完全恢复复制出来一个 new host，如果 source 和 new host 都在一个平台上，文件路径命名不能相同，如果是两个平台（不支持异构），文件路径命名可同也可不同。 无论如何 new host 有唯一的 DBID。

2）Standby 用途 Database
典型的就是 Data Guard, Standby Database 的主要功能是可以提供 Failover, 所以 Primary database 会将 log 输送到 Standy Database.使其不断更新数据。

10.8.3 RMAN 使用 Duplicate 命令又有两种模式：

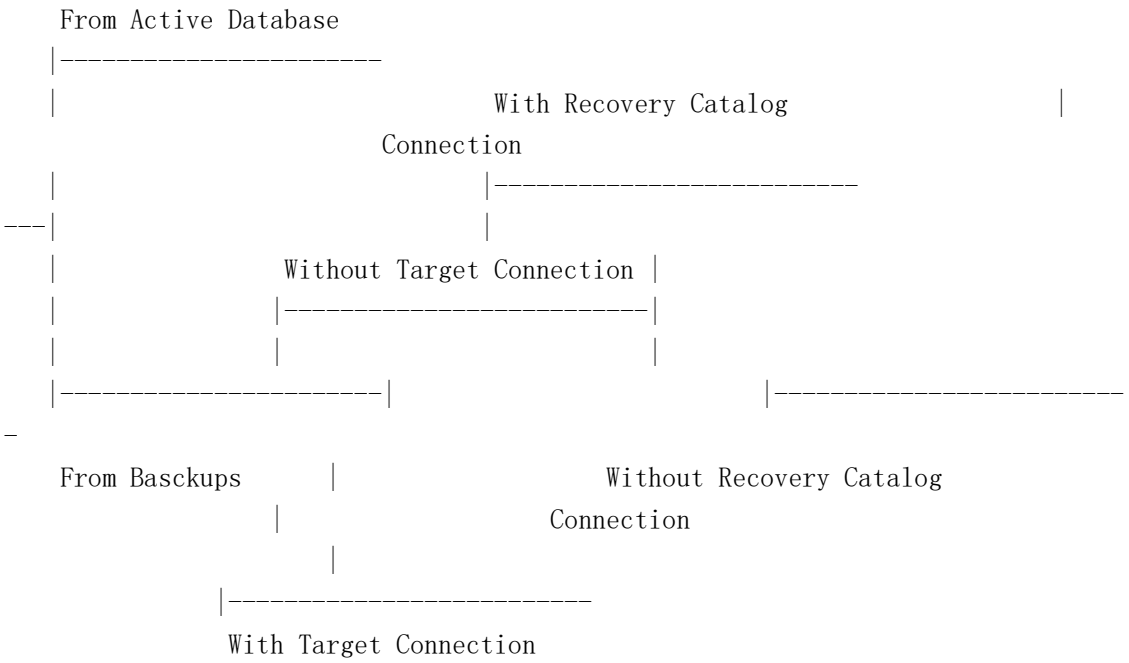
1）Backup-Based Duplication

基于 RMAN 备份的,利用 RMAN 备份创建辅助库

2）Active Database Duplication

基于 RMAN 网络的,从源数据库通过网络（不使用 RMAN 备份）直接创建辅助库。

10.8.4 RMAN Duplicate(for Database or Standby Database)的基本框架(PPT-II-535)



一般步骤：

1）将辅助库上安装数据库软件，建立 Oracle Home
2）为辅助库建立口令文件, Active 方式下与源数据库口令相同。
3）确保到辅助实例的网络连通性　（建立静态监听）
4）为辅助实例创建参数文件
5）以 nomount 方式启动辅助实例　（启动静态监听）
6）在 mount 或 open 模式下启动目标数据库

7）创建备份或将备份和归档日志文件复制到辅助库实例可以访问的某个位置，除非正在使用活动（active)数据库复制
8）如果有必要分配辅助通道
9）运行 rman duplicate 命令
10)使用 resetlogs 打开辅助数据库　（体现了 duplicate 是个不完全恢复）

例 1 Duplicating to a Host with the Same Directory Structure (Active)

```
DUPLICATE TARGET DATABASE TO dupdb
  FROM ACTIVE DATABASE
  PASSWORD FILE
  SPFILE
  NOFILENAMECHECK;
```

解析：
基于网络的复制数据库，此例是 Active 方式，source host 和 new host 需要同样的一个 PASSWORD FILE, SPFILE 是复制过去的（没有任何参数修改）。new host 和 source host 是同样的目录结构及文件命名。NOFILENAMECHECK 的意思是不检查文件是否重名，因为 new host 和 source host 不是一个平台。

例 2 Duplicating a Database Without a Target and Recovery Catalog Connection (Backup-Based)

```
DUPLICATE DATABASE TO dupdb
  UNTIL TIME "TO_DATE('11/01/2007 14:00:00', 'MM/DD/YYYY HH24:MI:SS')"
  SPFILE
  BACKUP LOCATION '/prod_backups'
  NOFILENAMECHECK;
```

解析：
基于备份的复制数据库，此例 RMAN 只连接了 new host(dupdb), source host 的备份已经放在 new host 能找的到的/prod_backups 目录下。

例 3 Duplicating a Standby Database to a Host with the Different Directory Structure (Active)

```
DUPLICATE TARGET DATABASE
  FOR STANDBY
  FROM ACTIVE DATABASE
  PASSWORD FILE
  SPFILE
    PARAMETER_VALUE_CONVERT '/disk1', '/disk2'
    SET DB_UNIQUE_NAME 'dup1'
    SET DB_FILE_NAME_CONVERT '/disk1','/disk2'
```

```
SET LOG_FILE_NAME_CONVERT '/disk1','/disk2'
SET SGA_MAX_SIZE 200M
SET SGA_TARGET 125M;
```

解析：
基于网络的复制Standby数据库，RMAN并不是简单复制source host的spfile到new host，它通过 spfile 子句做了一些参数指定及修改， new host 这边必须指定一个新的DB_UNIQUE_NAME以区别source host（因为它们的db_name是相同的），new host 使用新的spfile启动实例,然后通过网络(active方式)复制source host所有的datafile,archivelog到new host 的新的路径下。

考点：
1）duplicate 目前还不支持跨平台的复制，如windows--linux 就是两个平台。(windows32位和64位算是一个平台)。
2）必须保证RMAN的备份能使new host访问到（除非active方式）。
3）Active模式要保证source hos和new host两端有同样的sys口令文件.
4）如果是复制database，Oracle依靠不同的DBID区别source host和new host。
如果是复制 standby database，Oracle 依靠 db_unqiue_name 区分 primary host 和standby host。
5）如果是active模式，使用source host端target channel工作，要求source database是归档模式的。
6）如果是backup模式，需要有new host端的auxiliary channel工作。
7）初始化参数DB_FILE_NAME_CONVERT指定数据文件和临时文件的映射名称。
初始化参数LOG_FILE_NAME_CONVERT指定联机重做日志文件的映射名称。
8) 也可以通过duplicate database中的spfile子句指定上面两个参数。

第十一章： rman 维护

11.1 rman 使用和维护

11.2 list 命令一览

```
1）RMAN> list backup;
2）RMAN> list backup of tablespace users;
3）RMAN> list backup of datafile 2;
4）RMAN> list backup of controlfile;
5) RMAN> list backup of archivelog all;
6）RMAN> list backup of archivelog until time ='sysdate -1';
7）RMAN> list backupset 56;
8）RMAN> list datafilecopy all;
9）RMAN> list copy of tablespace users;
10）RMAN> list copy of datafile 1;
11）RMAN> list datafilecopy 67;
```

12）RMAN> list copy of controlfile;
13) RMAN> list expired backup;

11.3 crosscheck：用于检测备份状态（AVAILABLE：可用；UNAVAILABLE：不可用；EXPIRED：过期(RMAN 元数据还记录着，但 OS 上物理备份已经被删除了)。

AVAILABLE/UNAVAILABLE 的转换使用 change 命令

RMAN> CHANGE BACKUPSET 1 UNAVAILABLE;

是否 EXPIRED 要经过 crosscheck 命令才能看到。

RMAN> crosscheck backup;
RMAN> crosscheck copy;

对于 EXPIRED 状态的产生，与 crosscheck 命令是密切相关的，RMAN 通过 crosscheck 命令检查备份是否存在于备份介质上，如果不存在，则状态由 AVAILABLE 改为 EXPIRED（duplex 一个道理）。

例：可以试一下将某文件的备份在 os 下重命名，再使用 crosscheck 进行检查，然后再复原其命名，观察 AVAILABLE--EXPIRED--AVAILABLE 过程

RMAN>delete expired backup;　　//删除所有过期的备份。

元数据有而实际备份没有，这叫 expired，也可以反过来，让备份有而元数据没有。

RMAN> change backuppiece 11 uncatalog;　　//11 号备份片从 RMAN 元数据中删掉了，但物理备份 xxx.bak 还在。
RMAN> catalog backuppiece '/u01/myrman/xxx.bak';　　//将 11 号备份片又登记到 RMAN 元数据中了。

*考点：DELETE EXPIRED 命令不删除任何文件，它只更新 RMAN 存储库（元数据）， DELETE OBSOLETE 命令将删除文件并更新存储库。

11.4 report 报告命令一览

RMAN> REPORT schema;　　//查看目标库的物理结构
RMAN> REPORT schema at time|scn|sequence;　　//at 子句必须在 catalog 里使用（考点）
RMAN> REPORT need backup;　　//根据当前备份保留策略，列出哪些数据文件需要备份。
RMAN> REPORT NEED BACKUP DAYS n　　//使 N 天取代当前备份保留策略，列出哪些数据文件需要备份

RMAN> REPORT obsolete; //根据备份冗余策略来判断，那些备份是陈旧的（obsolete）
RMAN> REPORT NEED BACKUP incremental 3; //三天以来尚未进行备份的文件：
RMAN> REPORT NEED BACKUP redundancy 2; //如果不具有两个或更多个备份则需要进行备份：
RMAN> REPORT NEED BACKUP recovery window of 3 days; //需要进行备份以恢复到三天前的数据状态：
RMAN> REPORT OBSOLETE REDUNDANCY 2; //如果需要的备份副本不多于两个，列出作废的恢复文件：
RMAN> REPORT UNRECOVERABLE DATABASE; //备份文件中的表，采用了nologging 属性，那么该数据文件是不能恢复的

要理解区分这两组形式：

1）REPORT NEED BACKUP 和 REPORT NEED BACKUP DAYS n

REPORT NEED BACKUP //按照保留策略出报告
REPORT NEED BACKUP DAYS n //按照 N 天出报告

关于 REPORT NEED BACKUP DAYS n,联机文档中描述是：

“Displays files that require more than n days’ worth of archived redo log files for recovery”。

RMAN 基于这样的考虑，要保证恢复 n 天内任意一时间点的数据，如果缺少 n 天前的备份及归档日志，就将它们列出来。

2）REPORT OBSOLETE 和 REPORT UNRECOVERABLE DATABASE

REPORT OBSOLETE 根据保留策略计算出那些备份集是废弃的。

REPORT UNRECOVERABLE DATABASE

如果一个文件做过备份，其后在这个文件上创建的表采用了 nologging 属性，表插入数据没有记日志，那么是不能恢复的，这个表所在的文件就是 report unrecoverable database 要列出来的文件

11.5 delete 删除备份命令一览

RMAN> delete backup of datafile 2;
RMAN> delete backup of tablespace system;
RMAN> delete backupset ;
RMAN> delete backupset 30,32;

```
RMAN> delete backup of controlfile;
RMAN> delete noprompt backup of controlfile;      //删除 noprompt 不提示
RMAN> delete datafilecopy all;
RMAN> delete copy of  datafile 10;
RMAN> delete copy of  tablespace users;
RMAN> delete expired backup;    //删除过期的备份
RMAN> delete expired copy;
RMAN> delete obsolete;      //删除陈旧的备份
RMAN> delete noprompt obsolete; //删除不加提示
```

第二部分 优化 Oracle 数据库

第十二章 Oracle 资源管理

12.1 为什么要使用 Oracle 资源管理器

传统意义上，系统的资源分配是由 OS 来完成的，但是对于数据库资源，OS 分配资源会带来一些问题：

以 Linux 为例，最为突出的一个问题是：Linux 的资源调度是基于进程的，比如对于 CPU 的资源一般都是采用轮循的方法针对进程分时间片.也就是说 Linux 无法区分 Oracle 的后台进程和服务器进程之间谁轻谁重，也没有办法对 Oracle 用户以 session 角度考虑资源的配比。

Oracle Resource Manager 就是把原本由 OS 管理的硬件资源交给 Oracle 来管理。

12.2 Oracle 资源管理的基本概念： PPT:406

1)资源使用组：Resource consumer group

一个资源使用组由一组具有相似请求的用户组成，一个组可以包含许多用户，一个用户又可以是多个组的成员（这个很重要），实际中多个 session 使用一个用户名访问数据库是很普遍的，但是同一时刻，每个 session 只能有一个组作为这个 session 的有效使用者组。

当新创建一个 session 时，oracle 会根据你的设定自动把它分配到某个组（初始化组）。如果以某个用户登录的 session，它的用户名是属于多个组的，数据库管理员还可以手动的切换这个 session 所属的组。

下面三类特别的组是系统定义的组，它们不能被修改或删除。

```
SYS_GROUP                 //sys 和 system 属于这个组
DEFAULT_CONSUMER_GROUP      //没有指定用户属于哪个组时缺省放在这个组里
OTHER_GROUP               //在资源计划里除了指定组以外的用户都默认在这个组里
```

需要强调的一点是 OTHER_GROUPS，它是绑定在资源计划中的，所有资源计划必须要包括 OTHER_GROUPS 组，这个组的作用就是作为一个后选项，当一个没有匹配到任何资源组的 SESSION 连接到数据库的时候会自动的匹配到 OTHER_GROUPS 下面，受制于 OTHER_GROUPS 的资源限定。

2)资源计划：Resource plan

就是 Oracle 把数据库资源按一定方法来分配，可以限制每个组占用资源的比例。

在一个数据库中同一时间只能有一个资源计划 active。(也可以无任何资源计划 active)。一个资源计划还可以包含子资源计划。

查看当前被激活的资源计划的三个办法：

一是 show resoure_manager_plan 参数
二是命令 select name,is_top_plan from v$rsrc_plan;
三是通过 OEM 查看资源管理信息

3)资源计划指令：Resource plan directives

资源管理的目的是实现组或 session 对于资源分配的优先级，一部分用户在某时间内可以更多的享受资源，而另一部分则不能。

资源计划指令就是给出各种限定的条件，例如给某个组分配一定百分比的 CPU 时间，或者限制一个组内最大活动的会话数等等。

12.3 在资源管理中创建组、计划、指令的例子：

1）建三个用户，并授予 connect 角色：

```
grant connect to tim identified by tim;
grant connect to mike identified by mike;
grant connect to mgr identified by mgr;
```

2）建立两个使用者组：
sys 登录 EM-->Server-->Resource Manager-->Consumer Groups　　//查看默认组

建 OLTP 组、DSS 组，我们把新建的三个用户指定到两个自建的组中，使用 OEM 完成：

OLTP 组（tim,mgr）、DSS 组（mike,mgr）

建组结束前可以单击 Show SQL 研究一下输出，注意挂起区域的用法，单击 Enter 返回。

3）设置初始使用者组

缺省方式建立的用户会定位到 DEFAULT_CONSUMER_GROUP 组下：

```
SQL> select username,INITIAL_RSRC_CONSUMER_GROUP from dba_users;

USERNAME                       INITIAL_RSRC_CONSUMER_GROUP
------------------------------ ------------------------------
MGR                            DEFAULT_CONSUMER_GROUP
HR                             DEFAULT_CONSUMER_GROUP
TIM                            DEFAULT_CONSUMER_GROUP
MIKE                           DEFAULT_CONSUMER_GROUP
SCOTT                          DEFAULT_CONSUMER_GROUP
......
SYSTEM                         SYS_GROUP
SYS                            SYS_GROUP
......
```

将 TIM,MIKE,MGR 这三个用户定位到相应的自建组中，session 登录后便直接属于对应的自建组。

```
SQL>
exec dbms_resource_manager.set_initial_consumer_group('TIM','OLTP');
exec dbms_resource_manager.set_initial_consumer_group('MIKE','DSS');
exec dbms_resource_manager.set_initial_consumer_group('MGR','OLTP');

SQL> select username,INITIAL_RSRC_CONSUMER_GROUP  from dba_users;

USERNAME                       INITIAL_RSRC_CONSUMER_GROUP
------------------------------ ------------------------------
TIM                            OLTP
MIKE                           DSS
MGR                            OLTP
HR                             DEFAULT_CONSUMER_GROUP
SCOTT                          DEFAULT_CONSUMER_GROUP
......
```

4) 建立两个新的资源计划

首先是建立 DAYTIME--白天资源计划

4.1) 将 SYS_GROUP 组、OLTP 组、DSS 组加入 DATTIME 计划，在 OEM 中会发现 OTHER_GROUPS 组也自动加入了。
4.2) 指定指令计划，比如：CPU 使用优先级。选 advance,这个应该对应的是 MGMT 方法（8 个 cpu level).

具体设置：

```
Group/Subplan       level 1     level 2     level 3     level4  ......
------------------------------------------------------------------------------
-------
DSS                 20
OLTP                    80
OTHER_GROUPS                        100
SYS_GROUP       100
```

第二，建立 NIGHTTIME--夜晚资源计划
参考 4.1-4.2

```
Group/Subplan       level 1     level 2     level 3     level4  ......
------------------------------------------------------------------------------
-------
DSS                 90
OLTP                    10
OTHER_GROUPS                        100
SYS_GROUP       100
```

5)组的切换,MGR 用户初始组是 OLTP 组，现在切换成 DSS 组。

首先激活 DAYTIME 计划

linux 下 session1 sys;

```
SQL> alter system set resource_manager_plan='DAYTIME';
```

windows/cmd 下 session2 MGR 登录

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

linux 下 session1 sys;

```
SQL>select resource_consumer_group from v$session where username='MGR';

resource_consumer_group
------------------------
OLTP
```

session2,mgr:

```
declare old_grp varchar2(30);
```

```
begin
dbms_session.switch_current_consumer_group('DSS',old_grp,TRUE);
end;
/

session1 sys;

SQL>select resource_consumer_group from v$session where username='MGR';

resource_consumer_group
------------------------
DSS
```

6）sys 用户来切换某用户使用者组(需要 plan 切换或重新激活）

```
execute dbms_resource_manager.switch_consumer_group_for_user('MGR','OLTP');
```

12.4 其他的资源指令和阀值

1）以上介绍的 cpu 分配方法在原文中叫 EMPHASI 方法(按%分)，还有一种叫 RATIO（比例）的方法，两种性质相同，但表达方式不同，前者有 8 个 level,每个 level 中合计值不能超过 100，而后者是一种比例关系，只有一个 level，合计值可以超过 100。(PPT-II-413)

2）再回到 daytime 计划，看看还有几种指令

2.1）活动会话池：(PPT-414)
目的是限制一组同时运行的 SESSION 数量，假设 DSS 组有 8 个用户，如果就让 3 个可以运行（active session)。可将该组最大激活数设为 3，那么另外的 5 个可以连接上来，但要排队等着激活。而排队也可设延迟时间，超时后就会报错。

2.2）限制并行度：
Oracle 可通过参数设置并行度，如:parallel_max_servers,(创建一个并行执行服务器池)，但是无法阻止任何人使用它，于是我们可以通过 Resource Management 加以限制。

2.3）通过执行时间控制作业
数据库中一个大型作业会挤掉其他用户性能，Threshold 指令可以解决这个问题。到了时间阀值，按 action 的规定去做。

2.4）依据空闲时间终止 session
不做任何事情的 session 浪费服务器资源，如 PGA 白白占用，idle time 指令从两个方面限制这样的情况，比如某组的 max idle time(秒）=1800，表示空闲了 3 分钟，block another session(秒）=30,表示把别人锁了 30 秒，这两种情况都会终止该 session.

2,5）限制 undo 数据的产生

某些用户的大型事务可能填满 undo 表空间，如批处理事务不定期的提交，为了安全起见，可以对这里潜在的用户设置 undo 表空间使用上限，比如某组其 undo pool 设置为 6G，当到达这个上限值后该组中的所有 session 都会被挂起，直至事务提交后释放池中的空间。

12.5 配置测试使用者组自动切换

举例，接续 12.3 的例子，假设 MGR 要运行大作业，两个组里都有 MGR 这个用户，MGR 总能切换到那个资源优先级较高的组,那么 DBA 可以限制他的行为,通过设置 Threshold,以 DAYTIME 计划为例，OLTP 组的 Threshold 设为 10, ACTION 选 switch to DSS 组，这样一来，MGR 用户占用 10 秒时就被降级了。

1）OEM 使 DAYTIME 计划激活

2）设置上面的 Threshold 为 10 秒

3）session2 mgr：登录

4）session1 sys: 查看 MGR 当前组是 OLTP

```
SQL>select resource_consumer_group from v$session where username='MGR';
```

5) session2 mgr：做一个大查询

```
SQL>select count(*) from all_objects;
```

6) session1 sys: 查看 10 秒后降级为 DSS 组

```
SQL>select resource_consumer_group from v$session where username='MGR';
```

7) 若选择 revert after call=yes, MGR 任务结束后又返回到 OLTP 组（考点）。

12.6 考点：

1）SQL> show parameter resource

NAME	TYPE	VALUE
resource_limit	boolean	FALSE
resource_manager_cpu_allocation	integer	1
resource_manager_plan	string	NIGHTIME

激活的资源计划有一个参数：resource_manager_plan，但是 resource_limit 这个初始化参数与资源计划毫无关系。

2）赋给一个用户管理资源管理器的能力，需要系统权限 ADMINISTER RESOURCE MANAGER,而这个权限不是使用通常的 grant 方法授予的，只能使用 oracle 提供的程序包。

DBMS_RESOURCE_MANAGER 和 DBMS_RESOURCE_MANAGER_PRIVS 是有关资源管理器的 PL/SQL API 形式的最有用的两个包。

3）还是关于上面的两个程序包:

DBMS_RESOURCE_MANAGER_PRIVS 包负责：授予管理 Resource Manager 的权限，将用户放置到组中，从组中删除用户。
DBMS_SESSION 和 DBMS_RESOURCE_MANAGER 包 负责切换会话的有效组，创建使用者组，配置会话映射到组的方式。

4）在每个级别的总 CPU 使用率不能超过 100%，如果超过的话，则挂起区域将无法验证并且其资源计划不能保存到数据字典，资源计划容许在一个级别分配的资源<100%,但这样做没有什么意义。

5）每个资源计划都必须包含一条针对 OTHER_GROUPS 组的指令。

6）会话池不限制连接会话的数量，限制的是活动会话的数量。活动会话如果没有提交，该会话仍然对该组的活动会话池计数有效。

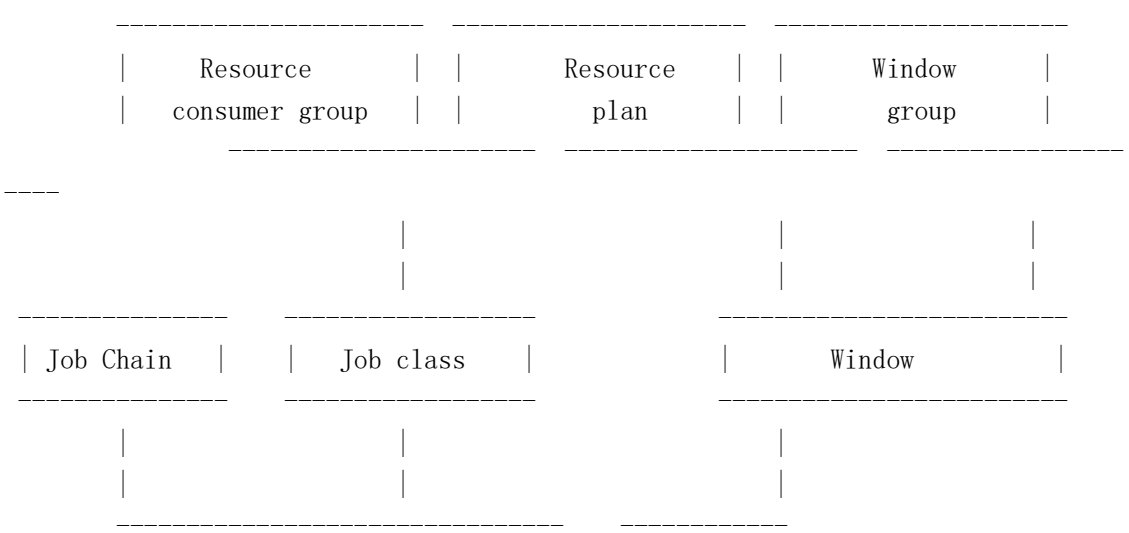
7）每个组的 undo pool 与 undo 表空间配额无关，你甚至不可能授予 undo 表空间上的配额。

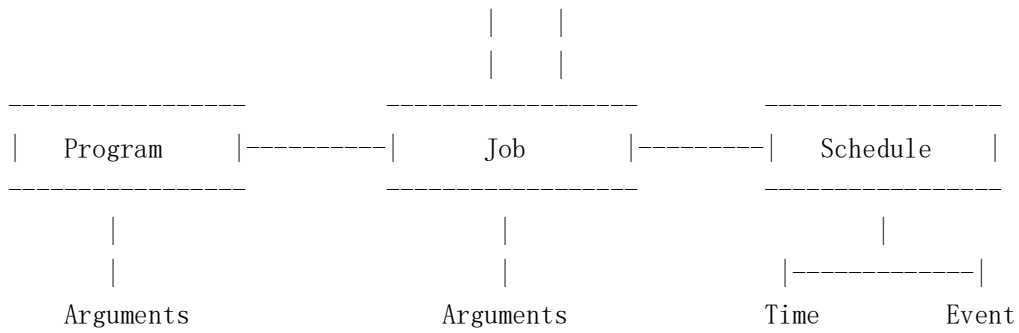
第十三章 Oracle 自动任务调度

13.1 Oracle 任务调度概述

在 Oracle 中任务调度指某一（组）执行程序在特定的时间被周期性的执行。Oracle 把任务调度称为 job（作业）。

Advanced Scheduler Concepts:(PPT:450)





13.2 理解以下概念：

1) Advanced SCHEDULER 下的 jobs

JOB 翻译为作业，一个基本的 JOB 由两方面组成：program 和 schedule。JOB 总体上可分为两大类，基于时间的 JOB 和基于事件的 JOB。在 Oracle 10g 之前，采用 dbms_job 程序包来完成任务调度的相关工作。在 Oracle 10g 之后，Oracle 推出了功能更加强大的 dbms_scheduler 来完成作业调度工作。

*考点：必须将 JOB_QUEUE_PROCESSES 实例参数设置为>0 的值，否则调度程序将无法运行，默认值为 1000。如果有任何定义的、活动的作业，那么总是运行作业队列协调器(后台进程 cjq0)。

```
SQL> show parameter job_queue_processes

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
job_queue_processes                  integer     1000

SQL> select program from v$process where program like '%J%';

PROGRAM
------------------------------------------------
oracle@timran.localdomain (J000)                 //术语：job slave 即 job 工作者，根据参数可以有 1000 个
oracle@timran.localdomain (CJQ0)                 //术语：Job Coordinator 即 job 协调员
```

2) Advanced SCHEDULER 下的 program

Program 指的是运行的程序。包括具体的要操作的执行代码，可选的是三种类型：

PL/SQL BLOCK ： 标准的 pl/sql 代码块；
STORED PROCEDURE ： 编译好的 PL/SQL 存储过程，或者 Java 存储过程，以及外部的 c 子程序；

EXECUTEABLE ： ORACLE 数据库之外的应用，比如操作系统命令等等。

3) Advanced SCHEDULER 下的 schedule

Scheduler 有的资料翻译为调度，有的翻译成时间表，简单来说指的就是作业运行周期，即时间和频率。

总结一下：如果不考虑与资源计划的结合，Schedules（调度）；Programs（程序）；Jobs（任务）这三者之间到底是个什么关系？直白地来说就是：program 负责具体做什么（左膀），schedule 负责什么时候做（右臂），job 就简单了（联手左膀右臂）。11g 的 SCHEDULER 提供了六种建立 Job 的存储过程，可以将 program、schedule、job 分立创建，也可以混合创建，相互组合非常灵活。

浏览一下 dbms_scheduler 中的内容。

```
SQL> spool /tmp/sch.txt
SQL> desc dbms_scheduler
SQL> spool off
$more /tem/sch.txt
```

4）JOB CLASSES

相当于创建了一个 job 组，可以将那些具有相同特性的 job，统统放到相同的 Job Classes 中，在 Job Classes 中的 Jobs 可以指派优先级（1-5），，默认优先级是 3，而 1 的优先级最高，然后让 Job Classes 与资源计划器结合进行管理。

5）WINDOW

窗口指定了作业运行的起始时间(START DATE)、终止时间(END DATE)，运行时间(DURATION)，以及重复间隔(REPEAT INTERVAL)等，它实际上扩展了 SCHEDULER(时间表）的概念，所谓扩展是指窗口可以激活资源管理计划，当窗口打开时，数据库自动切换到相关的 resource plan 上，此时这个 resource plan 便处于活动状态（ACTIVE),当窗口关闭时，将切换回原来的 resource plan,前提是此时没有其他窗口打开。

窗口对于 job class 也有着特殊的意义，通过 job class 和 resource plan 联系，这就使窗口下被激活的 resource plan 中所关联的 job class 中的所有 job 被激活了,这些 job 会根据优先权调度运行。

6）JOB CHAIN

CHAIN 可以被视做一组 Programs 的组合,举个简单的例子:运行 PROGRAM:A 以及 PROGRAM:B，如果成功的话继续运行 PROGRAM:C，否则的话运行 PROGRAM:D。Programs:A、B、C、D 以及执行的逻辑关系就构成了一个最简单的 CHAIN。

7) 轻型作业

轻型作业不显示在 DBA_SCHEDULER_JOBS 视图中，因为轻型作业不是模式对象，与普通作业相比创建和删除轻型作业的开销非常小。使用轻型作业有两种情形：1）作业成百上千次的批量执行，2）单个作业运行时间较短。

DBMS_SCHEDULER 程序包可以使用最少的语法来创建轻量作业。轻量作业只有很少的参数可以指定：作业参数和计划。作业的其余元数据（包括权限）都是从作业模板继承来的。轻量作业的模板必须是 PL/SQL 块或存储过程。轻量作业必须使用 DBMS_SCHEDULER 程序包在 PL/SQL 中创建。JOB_STYLE 参数在 EM 中是不可见的。

例 1，指定时间和频率的

```
dbms_scheduler.create_job(
job_name     =>'test_ltwtjob1',
program_name     =>'test_prog',
repeat_interval =>'freq=hourly',
end_date     =>to_timestamp(sysdate+1),
job_style    =>'lightweight',
```

例 2，使用预定义计划

```
dbms_scheduler.create_job(
job_name     =>'test_ltwtjob1',
program_name     =>'test_prog',
shedule_name     =>'test_sched'
job_style    =>'lightweight',
```

以上两种方法，轻型作业的名字叫 TEST_LTWTJOB1，程序 test_prog 就是它的模板，（这个模板我没有写），job_style 是轻型作业的关键字。

建立单一的轻型作业也许意义不大，另一种方法是通过 PL/SQL 块一次执行几百个轻量级作业的数组。

考点：轻型作业的指定的程序类型只能是 PLSQL_BLOCK 或者 STORED_PROCEDURE。

8）基于事件(Event)的作业

SCHEDULER 中有两种触发 EVENT 的情况：

Scheduler 触发的 Events

Scheduler 中触发的 Events，一般是说当前 schduler 中 job 的状态发生修改，类似 job 启动，或者运行结束，或者达到运行时间等诸如此类的动作，都能够抛出一个 EVENT，接收到 EVENT 的 application 就可以根据这些信息进行适当的处理。

Application 触发的 Events

外部的应用也可以触发 Events，并且由 Scheduler 来接收并处理这一类型的 Events。所谓 Scheduler 处理 EVENT 就是指 Scheduler 启动相应的 job 来执行相关操作，这类 job 在创建时专门声明了 event 的处理，这样当接收到 EVENT 时，这类 job 就会启动。

建立和管理基于事件(Event)的作业的方法超出了 OCP 考试范畴，这里就不多说了。

13.3 例：

下面通过实例来演示，如何创建 program、schedule、job,以便能初步理解三者间的关系。

我们使用分立创建三者的方法，先建一个测试表

```
SQL> create table scott.job_test1(my_date date);
```

第一步，创建一个名叫 my_pro1 的 program，(带 PL/SQL BLOCK)操作如下：

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM(
program_name    =>'my_pro1',
program_action  =>'begin
             insert into scott.job_test1 values(sysdate);
             commit;
          end;',
program_type    =>'PLSQL_BLOCK',
number_of_arguments =>0,
comments    =>'insert sysdate into table',
enabled     =>TRUE);
END;
/

PL/SQL procedure successfully completed.
```

通过上述语句,我们定义了一个program,类型是一个oracle匿名块,其动作是将输入sysdate插入到 scott.job_test 中。

前三个属性是不可以省略的，它们没有缺省值，必须指定，属性 number_of_arguments 是指定支持的参数个数，默认为 0，最多 255，这里又置为 0，强调此参数为 0，因为 program_type 是 PLSQL_BLOCK。要为 progremm 添加参数需要使用 DEFINE_PROGRAM_ARGUMENT 存储过程。

第二步：我们建立一个 schedule，规定开始 program 的操作时间，及操作频率。这里定义了 30 秒执行一次。

```
BEGIN
dbms_scheduler.create_schedule(
repeat_interval  => 'FREQ=SECONDLY;INTERVAL=30',
start_date   => sysdate,
comments     => 'Start Every 30 seconds',
schedule_name    => 'my_sch1');
END;
/
```

此处 REPEAT_INTERVAL 参数的语法结构比较复杂。其中最重要的是 FREQ 和 INTERVAL 两个关键字。

FREQ 关键字用来指定间隔的时间周期，可选参数有：YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, and SECONDLY，分别表示年、月、周、日、时、分、秒等单位。

INTERVAL 关键字用来指定间隔的频繁，可指定的值的范围从 1-99。

本例'FREQ=SECONDLY;INTERVAL=30 表示每 30 秒钟执行一次，如果是'FREQ=DAILY;INTERVAL=1';就表示每天执行一次，

第三步：创建 job，按照指定的 schedule，执行 program，创建 job 时，start_date,repeat_interval,job_action 等均无须指定，因为这些参数将由 program 和 schedule 来控制。操作如下：

```
BEGIN
dbms_scheduler.create_job(
job_name     => 'my_job1',
program_name     => 'my_pro1',
schedule_name    => 'my_sch1',
comments     => 'insert sysdate into table',
enabled      => TRUE);
END;
/
```

这样，操作完成后，ORACLE 就会自动定时(当前设置为 30 秒执行一次)program 中定义的操作。

检查结果：

```
SQL> select * from scott.job_test1;
```

```
MY_DATE
-------------------
2012-11-22 15:56:13
2012-11-22 15:56:43
2012-11-22 15:57:13
......
```

可以通过一些视图查看 JOB 状态

```
DBA_SCHEDULER_JOBS
DBA_SCHEDULER_RUNNING_JOBS;   //这两个都是查看当前 job 运行状态的(考点)
DBA_SCHEDULER_JOB_LOG
DBA_SCHEDULER_JOB_RUN_DETAILS
```

例如，查看刚刚创建的"MY_JOB1"任务的执行情况的细节，执行命令如下：

```
SQL>select log_id, log_date, status, additional_info from user_scheduler_job_run_details where job_name = 'MY_JOB1';
```

Oracle 为了创建和管理 Scheduler 下的基本部件，提供了大量的 PL/SQL API, 即 DBMS_SCHEDULER 包，通过存储过程管理作业、程序、进度表、链和事件，另外还有作业类，窗口及窗口组等。

除了作业按照时间表自动运行，你也可以手工干预它们，举几个例子：

使能作业：

```
SQL> exec dbms_scheduler.enable('my_job1');
```

禁用作业：

```
SQL> exec dbms_scheduler.disable('my_job1');
```

删除作业：

```
SQL> exec dbms_scheduler.drop_job('my_job1');
```

运行作业：

```
SQL> exec dbms_scheduler.run_job('my_job1');
```

停止作业：

```
SQL> exec dbms_scheduler.stop_job('my_job1');
```

等等，很多很多类似的存储过程，Scheduler 对它的每个组件都要关照，比如程序、时间表、JOB 类、窗口等。

从管理角度看，你能创建 JOB，也应该能修改 JOB，即修改 JOB 的属性，由于 JOB 属性众多，

Oracle 把这部分工作独立出来一个存储过程，这就是 DBMS_SCHEDULER.set_attribute， 此过程可以修改 JOB 属性，因属性太多了，这里仅举几个（也是考点）：

1）JOB_TYPE：指定 job 执行的任务的类型

有四个可选值：'PLSQL_BLOCK', 'STORED_PROCEDURE', 'EXECUTABLE', and 'CHAIN'。

2）JOB_ACTION：指定 job 执行的任务.这一参数所指定的值依赖于 JOB_TYPE 参数中的值，比如说 JOB_TYPE 设置为 'STORED_PROCEDURE'，那么本参数值中指定的可以是 ORACLE 中的过程名。

3）LOGGING_LEVEL：指定对 jobs 执行情况记录的日志信息级别。

SCHEDULER 管理的 JOB 对任务的执行情况专门进行了记录，同时用户还可以选择日志中记录信息的级别，有下列三种选择：

DBMS_SCHEDULER.LOGGING_OFF：关闭日志记录功能；
DBMS_SCHEDULER.LOGGING_RUNS：对任务的运行信息进行记录；
DBMS_SCHEDULER.LOGGING_FULL：记录任务所有相关信息，不仅有任务的运行情况，甚至连任务的创建、修改等也均将记入日志。

4）AUTO_DROP：当为 TRUE 时，一旦 job 到达停止运行的时间，该 job 就会被自动删除，否则的话 job 仍然存在，不过状态被修改为 COMPLETED。

5）RESTARTABLE：指定 jobs 运行出错后，能否适时重启，该参数默认情况下设置为 FALSE，如果设置为 TRUE，就表示当任务运行时出错，下次运行时间点到达时仍会启动，并且如果运行仍然出错，会继续重新运行，不过如果连接出错达到 6 次，该 job 就会停止。

13.5 Job Class(类）和 Window（窗口）

job class 包含着一组有相同特性的 job, 而 window 是一个时间段，当系统时间进入到窗口时间段，则窗口打开，有作业运行。

它们和资源计划的关联是个难点：一个 job 是怎样和资源计划联系起来的？

看图(PPT-II-452)，有两条线路需要理解：

1.Job class 将多个 Job 与一个资源组绑定，从而服从这个资源组的资源分配。

2.window 可以激活一个 Resource plan, 从而使得在 window 期内的 Job 得以运行。

这实际上是通过两条线路将 JOB 和资源计划关联起来的：

第一条线路：

创建一个 job class，将该 Job class 绑定某个资源用户组，则这个 Job_Class 将服从该用户组的资源分配计划。

创建 job,再通过 set_attribute 过程赋予相对优先权(1-5),然后将 job 加入 Job Class 后，会根据设定的优先值从 1-5 顺序执行的，1 首先被执行，默认是 3。

第二条线路：

建立 window 时必须绑定一个 resource plan，当系统时间进入 window 时，便激活了该 resource plan。Window 也可以设置优先权（high 和 low）

window 的优先级的作用是：在窗口重叠的情况下如何选择哪个窗口打开，假定有两个窗口，设置为同一时间打开则优先级为 high 的 window 会打开，如果是同样的优先级，window 时间长的打开，如果优先级相同，window 时间长度也相同，怎么办？则两个 window 都不打开，返还给原来活动的 window。

window 和 schedule 功能上相似，但 Oracle 设计让 window 和 resource plan 绑定，以便可以完成不同的 resource plan 的自动切换。

查看哪个窗口是活动的以及哪个资源计划与该窗口相关联，可以使用 DBA_SCHEDULER_WINDOWS 视图。

SQL>SELECT window_name,resource_plan,enabled,active FROM DBA_SCHEDULER_WINDOWS;

例；Job Class 和 Window 在 Scheduler 框架内的作用

1）运行下列查询看激活的窗口

select WINDOW_NAME,ACTIVE from dba_scheduler_windows; //ACTIVE 为 TRUE，表示这个窗口绑定的资源计划被激活。

2）看哪个资源计划是活动的

select * from v$rsrc_plan;

3）暂时清除当前活动计划

SQL> alter system set resource_manager_plan='';

下面接续刚才的例子，把那两条线路完成：

4）建立一个 Job Class，名叫 My_Class，将之前的 MY_JOB1 加入到 My_Class（MY_JOB1 默认加入的是 DEFAULT_JOB_CLASS），再将 My_Class 关联 OLTP 组。

5) 建立一个叫 DAY_WIN 的窗口，该窗口将激活 DAYTIME 计划

```
BEGIN
DBMS_SCHEDULER.CREATE_WINDOW(
window_name=>'"DAY_WIN"',
resource_plan=>'DAYTIME',
start_date=>systimestamp at time zone 'Asia/Shanghai',
duration=>numtodsinterval(1, 'minute'),
repeat_interval=>'FREQ=MINUTELY;INTERVAL=2',
end_date=>null,
window_priority=>'HIGH',
comments=>'');
END;
/
```

这将立即打开 DAY_WIN 窗口，每隔 2 分钟将再次打开该窗口，每次激活 DAYTIME 计划，duration 的含义是打开窗口的持续时间。这里的 1 分钟是窗口 Active 状态为 true 可以持续 1 分钟。

```
                              |     2 分钟      |     2 分钟     |...
窗口再次打开间隔 2 分钟       |-----------------|----------------|...
窗口持续打开时间 1 分钟       |Active |  关闭  |Active|  关闭  |...
```

也就是说：当 repeat_interval（间隔）>duration（持续时间）才有意义，Oracle 不检查这两个时间单位的逻辑关系。所以很容易晕。

至此，一个 job 关联一个 resource_plan 的框架已经搭建起来了，4）和 5）两条线路需要联动配合，实为关键。

6）
改一下 MY_JOB1 的时间表，原来它关联 MY_SCH1 这个 Schedule，现在让它关联 DAY_WIN 这个 window，
即 EDIT(MY_JOB1)-->Schedul Type-->Use_Pre_Define_Window-->DAY_WIN

改一下 MY_JOB1 的 JOB CLASS，原来它关联 DEFAULT_JOB_CLASS，现在让它关联 MY_CLASS
OK！到此，整个例子完成了。有时间去测试一下吧。

```
SQL> select * from scott.job_test1;
```

可以看到每间隔 2 分钟插入一条记录。

关闭一个窗口

SQL> exec dbms_scheduler.close_window('DAY_WIN');

*考点：
1）当 create job 过程创建作业时，无法指派优先级，必须在后面使用 API 的 set attribute 过程。
2）作业 A 在其类中的优先级是 1，作业 B 在另一个类的优先级是 5，作业 B 处于 resource plan 中有更高的优先权的使用者组内，则 B 先于 A 被执行。

第十四章 AWR 存储库及顾问框架(PPT-I-349-360)

14.1 Oracle 数据库采样 ASH 和 AWR。

1) ASH(Active Session History)

ASH 收集的是活动会话的样本数据，Oracle 的会话状态可以分为 3 种，1）on cpu,2)waiting,3)idle. 前两种都是活动会话（Active Session）而 v$session 包括了所有的（三种状态）当前会话，它每秒采样一次，那么 ASH 就以 v$session 为数据源，只记录活动会话信息，不活动的会话不会记录，记录数据在 SGA 缓冲区中。最终将 ASH 信息存入了 AWR 库。有关 ASH 数据采集和存入 AWR 库的过程都由后台进程 MMNL 来完成的。

生成 ASH 报告：

SQL> @/u01/oracle/rdbms/admin/ashrpt.sql

2) AWR（Automatic Workload Repository)

AWR 架构是从 oracle 10g 开始的，它以快照形式自动收集并保存和数据库有关性能统计数据，它的前身 Statspack，AWR 的作用是提供一个时间段内整个系统资源使用情况的报告，它存储重要的累计统计信息。通过这个报告，我们就可以了解一个系统的整个运行情况。

*考点：
AWR 工作时是由后台进程 MMON 负责，于每 1 小时生成一个内存统计的快照,并写入磁盘上的 sysaux 表空间，快照不能移动到其他位置，快照也会作为 ADDM 的原始数据，缺省情况下，Oracle 将快照保留 8 天。

生成 AWR 报告，注意是你确定的起始快照至结束快照之间的报告。

SQL> @/u01/oracle/rdbms/admin/awrrpt.sql

14.2 相关的一些概念:

1) AWR 收集数据库有关性能信息：它是新的数据库自动调优机制的核心，结果是以每小时一次快照的形式将关键数据的写入 SYSAUX 表空间。

所谓关键数据包括：

.基本统计数据，也是 v$sysstat 和 v$sesstat 视图中收集的系统和会话的统计信息；
.SQL 统计数据，分别按执行时间、cpu 时间、执行次数等标准来统计
.对象的统计信，
.时间模型统计信息，告知每个数据库活动要花多长时间。（在 v$sys_time_model 和 v$sess_time_model 视图中查看)；
.等待统计数据（来自 V$session 视图中的几个新添加的字段）
.ASH 统计信息，包含近期会话活动的历史记录
.数据库特性利用的统计数据
.各种管理顾问会话的结果，如 ADDM、Segment Advisor、Sql Access Adivisor 等
.操作系统的统计数据，如 I/O 和内存的利用率

2) AWR 度量（metric)： 两个或多个统计数据综合的结果。它是衡量累计性能统计数据变化率的统计指标。度量有两个主要作用，1）所有管理顾问都使用 metrics 诊断性能问题并给出调优建议。2）metrics 是服务器产生预警特性的基础。

3) AWR 基准线（baseline)。是一种快照集，由多个快照组成。通过把当前性能与基本阶段的性能进行比较，可以检验数据库运行的优劣。基线可以被无限期保留，除非你设定删除，因此基线总是拿来比较性能，而不像快照有 retention 限制。

Oracle 数据库中包含了三种类型的基线：

静态基准线(Fixed Baselines)

静态基准线相当于被指定的过去的一个固定的、连续的时间范围。在创建固定基线以前，要慎重考虑这个时间段，因为基线代表了一个理想状态的系统状态。之后，你可以用这个基线和其他基线或者某个时间范围内的快照来分析性能上的退化情况。

移动窗口基线(Moving Window Baseline)

Oracle 设计移动窗口基线是为了得到基于基线的度量阀值。只有一个移动窗口基线，即 Oracle 内置的 system moving baseline, 它使用 AWR 保留期内存在的所有 AWR 数据。可以自动产生并调整阀值，即自适应阈值(Adaptive Thresholds)，（见后叙)，system moving baseline 的默认窗口大小必须小于等于当前 AWR 保留的时间，默认为 8 天。如果你要使用自适应阈值，可以考虑使用更大的移动窗口，例如 30 天，可以更精确地计算出阈值。若你需要增大移动窗口的大小，首先需要增加 AWR 的保留时间。

基线模板(Baseline Templates)

你可以创建一个基线，作为未来一个时间连续的时间段可以使用的基线模板。有两种类型的基线模板：单一（single）的和重复的（Repeating） 。你可以为未来一个单独的连续时间段的（将会产生快照）创建单一基线模板。如果你要提前准备获取一个未来的时间段，这个

技术会很有用处。例如，你安排好要在周末进行一个系统测试，并准备获取 AWR 数据，这种情况下，你可以创建一个单一基线模板，用以在测试时自动获取该时间范围内的数据。你也可以使用重复基线模板来创建或者删除一个重复的时间计划，当你想自动获取一个连续的时间范围，这将很有用。例如，你可能希望在一个月里的每周一早晨获取 AWR 数据，这种情况下，你可以创建一个重复基线模板来自动为每个周一创建基线，并且在设置了过期时间(例如一个月)后，自动删除过期的基线。

建立一个 singe 基线模板的例子（考题）

```
begin
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE(
START_TIME => TO_TIMESTAMP(SYSDATE+2),
END_TIME => TO_TIMESTAMP(SYSDATE+10),
BASELINE_NAME => 'Mybase4',
TEMPLATE_NAME => 'Mytemp4',
EXPIRATION => NULL);
end;
/
```

建立一个 repeat 基线模板的例子

```
BEGIN
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
day_of_week => 'monday', hour_in_day => 17,
duration => 3, expiration => 30,
start_time => '2014-01-01 17:00:00',
end_time => '2014-12-31 20:00:00',
baseline_name_prefix => 'baseline_2014_mondays_',
template_name => 'template_2014_mondays');
END;
/
```

4) 自适应阈值(Adaptive Thresholds)

自适应阈值可以帮你以最低的开销来监控和检测性能问题。自适应阈值只能从移动窗口基线捕获到的度量值里得到的统计信息，为系统 metric 自动设置预警和关键预警(warning and critical alert)的阈值。这些统计信息每周会重新生成，并可能由于系统性能随着时间变化改变，而产生新的阈值。

打个比方，很多数据库白天是一个 OLTP 系统，而到晚上需要执行一些批量进程(例如生成报表)。每个事务响应时间的性能度量对检测 OLTP 的性能退化问题在白天可能很有用，但是这个阈值常常对于批量工作来说会太低，而频繁触发报警。自适应阈值能检测到这样的工作量模式，并自动为白天和夜里设置不同的阈值。

自适应阈值的类型有两种：

1）最大值的百分比：阈值以最大值的百分比倍数的方式设计，

2）重要性级别：阈值被设为一个统计学中的百分位来观察基于移动窗口基线数据的阈值以上的值，来体现异常程度。百分位能指定为以下几种：高(0.95)，100 个中只有 5 个能超过这个值；非常高(0.99)：100 个中只有 1 个能超过这个值；严重的(0.999)：1000 个中只有 1 个能超过这个值；极端的(0.9999)：10000 个里只有 1 个能超过这个值。

当一个系统以高峰期工作量来设计的，并且你希望在当前工作量接近或超过先前的高值时触发报警，最大值百分比阈值将非常有用。例如，每秒产生 redo 量的度量就是个典型的例子。

重要性级别阈值在以下情况很有用：当系统运行正常时表现得很稳定，但当性能变差时可能会在一个大范围内波动。例如，每个事务的响应时间的度量在一个优化过的 OLTP 系统上将表现平稳，但当性能问题凸显时，可能会波动很大。采用重要性级别阈值，当环境产生不正常的度量值和系统性能时触发报警。

5) 与 AWR 有关参数

```
SQL> show parameter statistics_level

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
statistics_level                     string      TYPICAL
SQL>
```

若参数 STATISTICS_LEVEL 设置为 TYPICAL 或 ALL 将默认启用 AWR 来采集数据库统计信息。ALL 参数收集信息最全，参数的默认值是 TYPICAL，

*考点：
如果 STATISTICS_LEVEL 设为 BASIC，将禁用收集快照和运行顾问。但你仍可以通过 DBMS_WORKLOAD_REPOSITORY 包来手动获得 AWR 统计信息。

14.3 ADDM (Automatic Database Dianostic Monitor)

14.3.1 概念

ADDM 相当于 Oracle 内部的一个顾问系统，它能够自动的完成对数据库的一些优化建议，它是根据 AWR 每小时采集的数据，看看有没有性能问题，如果有就给出建议调用各个相关的指导（Advisor），比如建议做 SQL Tuning Advisor，或 SQL Access Advisor，或者建议创建相关索引，总之， 给出建议是 ADDM 作为一个顾问的特色，就像医生给病人看病。

ADDM 的另一个特点是自动生成的 ADDM 报告，默认它会包括当前快照和前一个快照的时间段，如果想要 ADDM 跨越更长的时间段，也可以手动调用 ADDM 生成包括任意两个快照间的时间段。

与 ADDM 有关的参数：
control_management_pack_access 缺省是 DIAGNOSTIC+TUNING，如果设成 NONE，则 ADDM 关闭。

*考点：收集 AWR 快照时自动运行 ADDM，根据情况决定是否产生报告，也可以根据需要手动生成 ADDM 报告，并将其结果保存到 AWR 中。默认情况下 ADDM 报告保存 30 天。

ADDM 报告生成：

SQL> @/u01/oracle/rdbms/admin/addmrpt.sql

例：

1）在 OEM 中了解最新 AWR 快照号和 ADDM 报告

2）手工即时生成 AWR 快照
exec dbms_workload_repository.create_snapshot;

3）模拟业务高峰
create table scott.test1 as select * from all_objects;

```
begin
for i in 1..20 loop
insert into scott.test1 select * from all_objects;
update scott.test1 set object_id=1 where object_name='TEST1';
delete scott.test1;
end loop;
commit;
end;
/
```

4）再次手工即时生成 AWR 快照
exec dbms_workload_repository.create_snapshot;

在 OEM 中，查看这两次快照的 AWR 报告。

查看 ADDM

OEM：Server-->Advisor Central-->Advisor Task Results（画面下方）已经显示了一个 ADDM 结果，

我们可以尝试单独生成一个 ADDM 报告。方法：

Server-->Advisor Central--> ADDM-->Run ADDM to analyze past performance

结合图型高峰段选择 Period Start Time 和 Period End Time-->ok

可以看到报告生成的名称是 task_nn,然后有 Impact(%)，Finding，Occurrences 等信息
进入第一行，即 Finding 是 Top SQL by DB Time,里面是 Recommendations，然后一组 SQL Tuning Set(STS)...

14.3.2 其他顾问 （Advisor）

顾问就是通过分析 AWR 捕获的数据，提出改进性能的建议。ADDM 本身就是顾问，它的报告里还会建议你找其他的顾问。具体病症还要看专科。

Oracle 11g 主要的一些 Advisor：(PPT-II-382)

1)SQL Advisors 其中包括：

SQL Tuning Advisor: 对单个 SQL 语句提供调优建议,生成 sql profile，sql 语句执行路径分析，sql 语句结构分析
SQL Access Advisor: 评估 SQL 语句对数据库负荷的影响，提供建议。如 index,partition, materilizer view 等
SQL Repair Advisor: 对可能的 oracle 内部错误，如 ORA-600 需要的 patch(补丁）提出建议

2)Memory Advisors：可以对 Oraclen 内存结构（SGA+PGA）做自动调整，以适应数据库在不同时间段的工作量变化。

3)Segment Advisor: 提供段收缩命令（shrink)。释放未使用的空间。

4)Undo Advisor: 为 undo 表空间的大小提供建议，如避免快照太旧的问题。

5)MTTR Advisor：为实例恢复的时间提供建议。

14.3.3 关于 Advisor 的 API's 程序包

首先是 DBMS_ADVISOR: 不过，它只是顾问管理中一部分包的套件，有一些 Advisor 有它们自己的包，如：

Automatic Database Diagnostic Monitor (DBMS_ADDM)

SQL Performance Analyzer (DBMS_SQLPA)

SQL Repair Advisor (DBMS_SQLDIAG)

SQL Tuning Advisor (DBMS_SQLTUNE)

14.3.5 例：通过 OEM 观察：如何使 Segment Advisor 完成段重组。

接续 14.3.1 的实验,scott.vvvvvv 表产生了较高的 HWM,delete 命令使该表没有任何记录，但没有降低 HWM。

OEM：Server-->Advisor Central-->Segment Advisor-->Tablespace-->进入顾问评估的四个步骤，在第二步骤选择 add users 表空间（可以看到 users 当前的表空间 used%）。-->Submit

这时进入 Segment Advisor 的分析状态，它的数据源是 AWR，Refresh 后 Status：COMPLETED，然后点击这个任务的 name 看看是否有 Recommendations，如果有，点击 Recommendation Detail 可以进入实施（Implement）对于 users 整个表空间的段重组。

选择 Recommendation 项下的 Shrink 可有一下选择：

Compact Segments and Release Space（压缩并释放空间）

Compact Segments（压缩）

然后选择 Implement,最后观察 users 表空间 used%棒图大大降低了。说明 HWM 下降了,unused block 已经回收了。

14.4 优化器和优化器统计信息

14.4.1 概念：

优化器(Optimizer),也叫查询优化器，它的功能是为了对 DML 操作寻找最理想的执行计划。早期的 RBO（Rule_Based Optimizer)已经淘汰，现在 Oracle 使用 CBO（Cost_Based Optimizer）,基于成本的优化器。

CBO 优化器得出执行计划，依据的是优化器统计数据(optimizer statistics)，而优化器统计数据又是怎么产生的？由 DBMS_STATS 程序包派生。

如果在查询中，你访问的对象（表、索引），如果已经有了统计信息，那么正好优化器会很快的得到一个好执行计划，如果缺少统计信息，优化器不得不使用动态采样技术在运行时收集统计信息。这可能是仓促的，不准确的，即得到一个不好的执行计划。

```
SQL> show parameter optimizer_dynamic_sampling

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
optimizer_dynamic_sampling           integer     2
```

14.4.2 优化器有两种工作模式 tuning mode 和 normal mode.

tuning 模式下就是 SQL Tuning Advisor.可以有充足的时间完成下面四个方面的工作：

1）统计信息分析，2）产生 SQL profile ，3）访问路径分析，4）SQL 结构分析。

normal 模式下，对一个新的 sql（查询）语句，优化器要产生对应的执行计划，没有办法细致的分析，即可能的得到执行计划不是最优的。

14.4.3 自动和手动收集 optimizer statistics

Oracle 通过自动维护任务（Automated Maintenance Tasks）框架自动收集 optimizer statistics，所谓“自动任务”是将任务纳入资源管理的中，由 Automatic Statistics Gathering 作业来完成，好处是可以被 Window 定时激活，同时被限制过度使用资源（25%以下）。定时的收集数据库中所有的 objects 信息，表、列、索引，系统 IO 等统计信息都属于 optimizer statistics 范畴，这些信息都存入了数据字典。

Oracle 推荐 optimizer statistics 使用“自动收集”。

此外，特别情况下也可以“手动收集”，即 Manual Statistics，使用 DBMS_STATS 包随时可以进行 optimizer statistics 的收集。

测试手工收集 scott 下所有对象的统计信息

OEM->Server->Query Optimizer->Manage Optimizer Statistics->Gather Optimizer Statistics

查看信息：

SQL> select OWNER,TABLE_NAME,LAST_ANALYZED from dba_tables where owner='SCOTT';

SQL> select job_name,LAST_START_DATE from dba_scheduler_jobs;

SQL> select column_name,num_distinct from dba_tab_col_statistics where table_name='EMP';

COLUMN_NAME	NUM_DISTINCT
DEPTNO	3
COMM	4
SAL	12
HIREDATE	13
MGR	6

JOB	5
ENAME	14
EMPNO	14

一般情况下，一旦一个 object 的统计信息新版本产生，老版本统计信息就被自动保存起来了,那么同样的查询语句需要在新版本下再做解析。当然,oracle 也给你提供了灵活的选择，这就带来了下面的概念：

考点：
1)Restoring Previous Versions of Statistics 恢复老版本的统计信息
2)Locking Statistics 对于某些 schema（如 table)可以锁定,就是不产生新版本统计信息了，想想静态表（只读表）可能是需要的。
3)Pending Statistics 收集新版本信息，但是挂起统计结果，暂时不用.

例：EXEC DBMS_STATS.SET_TABLE_PREFS('SCOTT', 'EMP', 'PUBLISH', 'false'); //收集 EMP 表的新版本统计信息，但被挂起了

4)Automatic Optimizer Statistics Collection 过程正在收集所有 objects 信息，这时 Window 关闭了，会出现什么状况？

第一，job 自动停止，第二，已收集好的 objects 保存进数据字典，第三，还未收集的 objects 在下个 Window 到来时继续收集。当然也可以通过指定属性改变以上状况，使得当 Window 结束后未完成的 job 继续进行，直至 job 结束。

14.4.4 关于 Autotask 的三个 job：

通过 OEM->Server->Oracle Scheduler->Automated Maintenance Tasks 查看。

Automatic Optimizer Statistics Collection,
Automatic Segment Advisor,
Automatic SQL Tuning Advisor。

14.4.5 预定义维护窗口
有 7 种预定义窗口，通过 OEM->Server->Oracle Scheduler->Window Groups 可以查看，当一个维护窗口打开时，数据库激活 DEFAULT_MAINTENANCE_PLAN，3 个自动维护任务运行在 ORA$AUTOTASK_SUB_PLAN 子计划之下。

14.4.6 管理 Autotask(PPT-1-362-364)

Oracle 不给 3 种自动维护任务分配永久的 Scheduler 作业，所以不能用 DBMS_SCHEDULER 程序包管理这些任务。如果想修改 Autotask 必须使用 DBMS_AUTO_TASK_ADMIN 程序包。

一个反复出现的考点是关于顾问产生的 RECOMMEND，

RECOMMEND(推荐)不会自动 IMPLEMENT(实施)，但是自动维护任务中的 Automatic SQL Tuning Advisor 产生的 SQL PROFILE 可以是个例外，通过设置 IMPLEMENT ENABLE，如果 Oracle 分析出实施 SQL PROFILE 可以为 PLAN 带来 3 倍以上效率时，会自动 IMPLEMENT 这个 SQL PROFILE。其他 RECOMMEND 都必须要 DBA 手动 IMPLEMENT。

14.4.7 监控 Autotask

通过两个视图：dba_autotask_clien 和 dba_autotask_operation

SQL> select client_name,status,attributes,window_group,service_name from dba_autotask_client;

SQL> select client_name,operation_name from dba_autotask_operation;

第十五章 预警及诊断系统(PPT-I-365-371)

15.1 使用服务器生成预警

从 10g 开始，Oracle 数据库凭借警报系统一举实现了‘自我管理’。警报系统顶替了大量的单调工作，在问题发生时自动报警。预警分为有状态警报和无状态警报，

有状态警报----通过阀值配置的预警，而阀值又是基于度量（metric）的。
无状态警报----非阀值的与问题有关的预警，基于数据库中的预定义事件。

11g 有 200 多种阀值，它们存在于 AWR 中，此后，MMON 后台进程将实施监控，并将当前状态与阀值比较，比如：当表空间达到全满的 85%时发出警告性警报，达到 97 时将发出严重警报。

典型的几个服务器产生的预警

有状态警报：
表空间使用率（使用率 85%为警告预警，97%为严重预警）

无状态警报：
快照太旧
可恢复会话挂起
恢复会话运行空间出超。

例，表空间有状态预警的机制。注意两个视图：dba_outstanding_alerts 和 dba_alert_history 的关系。

1） 建立表空间，使用 unifom 是为了分配空间时可以控制超过 85%

SQL> create tablespace small datafile '/u01/oradata/timran11g/small01.dbf' size 10m extent management local uniform size 3m;

2） 建表，最小分配 3 个 extent，合计就是 3*3 个 uniform size，空间分配已经达到 9M，即超过 85%了。

SQL> create table scott.test_table(id int) tablespace small storage(minextents 3);

查看 dba_outstanding_alerts 视图，信息不是马上就有，MMON 采集需要几分钟。

SQL> select reason from dba_outstanding_alerts;

REASON

表空间 [SMALL] 已占用 [90 percent]

看 OEM->HOME 该预警也显示出来了。

3）表空间加大，

SQL> alter database datafile '/u01/oradata/timran11g/small01.dbf' resize 20m;

4）再看 dba_outstanding_alerts 视图已经没有信息了（要等 MMON 几分钟），信息被记录到 dba_alert_history 视图，OEM 预警是从 dba_outstanding_alerts 取信息，这时 HOME 下的显示信息也 clear 掉了。

SQL> select reason from dba_outstanding_alerts;

未选定行

SQL> select reason,resolution from dba_alert_history where reason like '%表空间%';
　　//cleared 表示预警已清除

REASON
RESOLUTION

表 空 间 [SMALL] 已 占 用 [45 percent]
cleared

SQL> drop tablespace small including contents and datafiles;

15.1.1 阀值的设置方法

有 API 和 OEM 两种方法、OEM 方法;HOME 主页-->Relete Link-->Metric and Policy settings

15.1.2 通知系统

需要除默认配置外其他通知方法，需要进行三个级别的配置

1）必须配置通知方法　　OEM->右上角 setup 链接
2）必须创建规则来捕获事件　OEM->右上角 Preferences 链接
3）管理员必须订阅规则　　Preferences 链接->左边 Administrators

*考点：
1）预警由 MMON 后台进程而非 EM 引发，EM 只管读取警报，自己或第三方编写的事件处理程序也同样可以。
2）有状态警报默认机制是在 OEM 主页上显示，并将它们写入了 dba_outstanding_alerts 视图，如果问题解决了，则可能将其清除，然后保存到 dba_alert_history 视图中，而无状态警报直接写入 dba_alert_history 视图。

15.2 Support Workbench

15.2.1 自动诊断知识库（ADR）(PPT-II-237)

052 课程中已经介绍过，它是基于文件的一个所有诊断信息的中心存储点，它包括了各种存储文件、跟踪文件、跟踪日志以及健康状况检测信息

查看和管理 ADR 的诊断信息可以有两种途径：OEM 的 Support Workbench，以及 ADRCI 命令行工具

15.2.2 诊断框架的几个基本概念：Problem、Incident、Packaging(PPT-I-506)

Problem(问题）:

每个问题都有一个问题键，此键是一个文本字符串，包含错误代码和参数。

问题可以分为几类：

数据库或实例中的关键错误，（如：ORA-4031 错误，没有足够共享内存）
Oracle 内部错误（如：ORA-600 的错误）
操作系统错误

Incident(事件):

也有翻译成意外事件的，它是指某个问题的单次出现，如果多个事件的问题键是相同的，说明这些事件具有相同的根源。当问题发生时，数据库会自动创建事件并在 ADR 预警日志中报告它，当然，除了数据库自动产生事件，你也可以将你认为的错误手动生成一个事件。

每个 Incident 都有自己的 Incident ID。

Packaging（打包）：

如果买了 Oracle 服务，你可能希望将一些 Problem 收集起来，打包成一个 zip 文件提交给 Oracle Support，这就引出了另一个概念：意外事件打包服务:

Support Workbench 提供的意外事件打包服务可以支持的功能有：

1）查看错误预警
2）查看问题细节
3）收集额外数据　　//除了自动诊断数据外，额外数据指的是从健康检查里得到的数据。
4）创建服务请求　　//指的是使用 MetaLink 创建服务请求，你要有账号。
5）创建意外事件包　//有 Quick Packaging 方法，和 Cusotm Packaging 方法，后者可定制自己的意外事件包，步骤稍复杂。
6）跟踪服务请求　　//提交了一个意外事件包后，还可以追加新的意外事件。
7）进行修复　　//修复指的是你可以调用链接的顾问框架去自行修复
8）关闭意外事件　　//关闭是为了结束除一个解决了的意外事件，也可以让 Oracle 清除它，期限默认是 30 天。

ADRCI 有一套命令可以完成打包任务，如：

```
ADRCI> help
ADRCI> show incident          //查看意外事件
ADRCI> ips create package incident 17000      //将事件 17000 打包
```

15.3 Health Monitor 健康监测

Health Monitor 是 11g 里新增加的特性，它也是属于数据库的诊断框架的一个模块。它在数据库遇到严重错误时执行自动诊断检查（Reactive），你也可以根据需要随时进行手动检查（Manually）。HM 可以检查：文件损坏、物理逻辑块损坏、redo 和 undo 故障、数据字典损坏等。HM 根据检查的结果可以产生一个报表，并提供解决问题的建议。

15.3.1 运行方式：

1). Reactive
　　Fault diagnosability infrastructure 能自动响应严重的错误（critical error）。

2). Manually

可以通过 DBMS_HM 系统包或 Enterprise Manager 来手工运行 HM。

15.3.2 运行模式

1). DB-ONLINE
2). DB-OFFLINE

```
SQL> select name,internal_check,offline_capable from v$hm_check where internal_check='N';

NAME                                                             INTERNAL_CHECK
OFFLINE_CAPABLE
---------------------------------------------------------------- --------------
---------------
DB Structure Integrity Check                                     N
Y
Data Block Integrity Check                                       N
Y
Redo Integrity Check                                             N
Y
Transaction Integrity Check                                      N
N
Undo Segment Integrity Check                                     N
N
Dictionary Integrity Check                                       N
N
```

15.4 模拟数据字典讹误，使用 EM 手工运行 HM，查看 Alerts、Active Incident、Problem。通过 Support Workbench 提交 service request(SR)，完成 Package。

第一步：建立一个测试表 dictcheck，定位该表的 object id

```
create table scott.dictcheck(id int);
select object_id from dba_objects where object_name='DICTCHECK';

 OBJECT_ID
----------
     71117

select obj#,cols from tab$ where obj#=71117;

      OBJ#       COLS
---------- ----------
     71117          1
```

第二步，让数据字典产生一个错误。表的列数最多支持 1000，强制超过此上限。

```
SQL> update tab$ set cols=1001 where obj#=71117;
SQL> commit;
SQL> select obj#,cols from tab$ where obj#=71117;

      OBJ#       COLS
---------- ----------
     71117       1001
```

第三步，进入 EM，检查数据字典一致性

首先进入 Home 主页，观察 Diagnostic Summary 及 Alerts。目前还没有相关的告警信息，现在做一个手动 HM，然后：

```
EM-->Advisor Central-->Checkers-->Dictionary Integrity Check-->Run Name(MyCheck)
```

查看检查结果细目，
在 Findings 中有"SQL dictionary health check: invalid column number 8 on object TAB$ failed"信息
在 Damage Translation 中有"Damaged rowid is AAAAACAABAAAS7QAAA - description: Object SCOTT.DICTCHECK is referenced"

第四步，访问 dictcheck 表，将产生一个 Active Incident

```
SQL> alter system flush shared_pool;
SQL> alter system flush buffer_cache;
SQL> exit
[oracle@timran11g ~]$ sqlplus / as sysdba

SQL> select * from scott.dictcheck;
select * from scott.dictcheck
*
第 1 行出现错误:
ORA-03113: 通信通道的文件结束
进程 ID: 19231
会话 ID: 100 序列号: 16001
```

检查 Home 主页--Damage Translation 中已经有了 1 个 Active Incident
重新连接

```
SQL> conn scott/scott
```

再次访问 dictcheck 表，

```
SQL> select * from scott.dictcheck;
select * from scott.dictcheck
```

*
第 1 行出现错误:
ORA-03113: 通信通道的文件结束
进程 ID: 19615
会话 ID: 113 序列号: 3254

检查 Home 主页--Damage Translation 中已有了 2 个 Active Incident

第五步，进入 Support Workbench，查看 Problem 和 Incident 的相关性

点击 Incident(事件)将自动链接到 Support Workbench, 看到它们都关联同一个 Problem(问题）--ORA 7445 [qcstda()+490]
点击 Incident ID,查看 incident dump 文件，

第六步，申请 SR，完成打包服务。

点击 Problem ID，查看 ORA 7445，接下来申请 SR，然后 Packge...

最后删除测试表
SQL> drop table scott.dictcheck purge;

第十六章：Oracle 性能调优

16.1 SQL 的硬解析和软解析

1）parse 分析语法语义

i) 从共享池的库缓冲区搜索，该语句是否执行过，凡是执行过的 sql 语句，oracle 会使用 HASH 函数计算，产生一个很小的文本记录， 如果是第一次执行则进入第二步。

ii) 检查语法，权限（权限的信息放在 oracle 的数据字典当中。oracle 先从共享池的数据字典缓冲区中搜素，如果没有，再从数据文件 （system 表空间的数据文件）当中读取， 然后，存放在数据字典缓冲区，以便共享。

iii)分析过程中，对访问到的表进行锁操作，目的是保护表的结构不被修改，优化器会根据数据的存储结构（表的存储结构），统计 信息，计算读取的代价，生成执行计划同时编译并存储在共享池的缓冲区中

2）BIND 变量，优化器会考虑绑定变量来确定执行计划。

3）查询优化器建立执行计划

4）执行库池里的执行计划， 返回结果（sql 硬解析从 1)开始，软解析从 4)开始）

16.2 共享游标的概念，父游标和子游标

shared cursor

父游标和子游标同属共享游标范畴，通过视图 V$sqlare 和 V$sql 可以得到具化。父游标在进行硬解析时产生，当产生父游标的同时跟随父游标会产生相应的子游标，此时 v$sql 中的 child_number 的值为 0。同样的父游标由于不同的运行环境会产生不同的子游标。如果一条 sql 语句不但与 library cache 中的父游标匹配，同时又和其下的子游标匹配，那么这个 sql 语句就称为共享游标匹配。

```
SQL> alter system flush shared_pool;

select count(*) from scott.emp where deptno=10;
select count(*) from scott.emp where deptno=10;
SELECT count(*) from scott.emp where deptno=10;


SQL> select sql_id,sql_text,child_number,executions from v$sql where sql_text
like '%deptno=10%' and sql_text not like '%v$sql%';

SQL_ID                                                                  SQL_TEXT
CHILD_NUMBER EXECUTIONS
------------- ------------------------------------------------------------------
-------------- ------------ ----------
dp4tk30fhwyp1        SELECT    count(*)    from    scott.emp    where    deptno=10
0          1
1kjj3s9qg6xg4  select count(*) from scott.emp where deptno=10
      0          2

//查看父游标，其中一个父游标（1kjj3s9qg6xg4）执行了两次。
//查看子游标，CHILD_NUMBER 为 0，说明父亲游标下只有一个子游标


SQL> alter system flush shared_pool;
SQL> alter session set optimizer_index_caching=40;      //设定此参数并执行聚合查
询

SQL> select count(*) from scott.emp where deptno=10;

SQL> alter session set optimizer_index_caching=100;     //改变此参数并执行聚合查
询
```

```
SQL> select count(*) from scott.emp where deptno=10;

SQL> select sql_id,sql_text,child_number,executions from v$sql where sql_text
like '%deptno=10%' and sql_text not like '%v$sql%';

SQL_ID                                                                    SQL_TEXT
CHILD_NUMBER EXECUTIONS
------------- ------------------------------------------------------------------
-------------- ------------ ----------
1kjj3s9qg6xg4        select    count(*)    from    scott.emp    where    deptno=10
0           1
1kjj3s9qg6xg4        select    count(*)    from    scott.emp    where    deptno=10
1           1
```

//看到一个父游标(1kjj3s9qg6xg4)下,由于不同的运行环境，产生了两个子游标。

以上操作说明了两点：

1.与父游标 SQL 文本完全一致的情形下，多个相同的 SQL 语句可以共享一个父游标。
2.当 SQL 文本，执行环境完全一致的情形下，子游标能够被共享，否则，当环境不一致时，将在父游标下生成新的子游标。

16.3 使用 BIND 变量减少硬解析：

例：

```
---------
带绑定变量的查询
create table scott.m1(x int);
create or replace procedure proc1
as
  begin
    for i in 1..10000
    loop
        execute immediate
        'insert into scott.m1 values(:x)' using i;
    end loop;
  end;
/

---------不带绑定变量的查询
create table scott.m2(x int);
create or replace procedure proc2
```

```
as
  begin
    for i in 1..10000
    loop
        execute immediate
        'insert into scott.m2 values('||i||')';
    end loop;
  end;
/
---------两项分别执行，比较效率

SQL> set timing on;
SQL> exec proc1

SQL> select sql_id,sql_text,child_number,executions from v$sql where sql_text
like '%insert into scott%' and sql_text not like '%v$sql%';


SQL_ID                                                                    SQL_TEXT
CHILD_NUMBER EXECUTIONS
------------- ------------------------------------------------------------------------
-------------- ------------ ----------
2uqs6daxb54ar          insert          into          scott.m1          values(:x)
1      10000

PL/SQL 过程已成功完成。
已用时间:  00: 00: 01.21

SQL> exec proc2

PL/SQL 过程已成功完成。
已用时间:  00: 00: 09.98
```

以上例子可以看出绑定变量效果

1，硬解析通常是由于不可共享的父游标，如经常变动的 SQL 语句，或动态 SQL 或未使用绑定变量等。
2，为了减少硬解析，通常是考虑使用绑定变量的办法。

16.4 11g 新特性，

当一个查询执行时，数据库在 cache memory 查找是否存在结果集，如果有就从内存中取出而不再执行查询，如果没有则执行查询，返回结果，RESULT CACHE 这个新特性可以看成是执行计划的一个补充内容，利用内存中保存的结果集，可以避免再次软解析。

进一步说，Result Cache 又可以分为：Server Result Cache 和 Client Result Cache。

前者通过服务器端 SGA 来缓存结果集保存 shared pool 中，后者通过客户端来缓存结果集。

客户端结果集缓存针对使用 OCI 应用程序(考点)，并不使用服务器端的内存，不会对服务器的内存使用造成影响，

注：（Oracle Call Interface，OCI）是 Oracle 数据库访问的一种底层接口，通过它可以高效地访问 Oracle 数据源，性能优于 ODBC，是开发大型 Oracle 数据库应用程序的利器。

Client Result Cache 相关的视图是：V$CLIENT_RESULT_CACHE_STATS

```
SQL> show parameter result

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
client_result_cache_lag              big integer 3000
client_result_cache_size             big integer 0
result_cache_max_result              integer     5
result_cache_max_size                big integer 1056K
result_cache_mode                    string      MANUAL
result_cache_remote_expiration       integer     0
```

前两行是 client_result_cache 的参数
后四行是 server result cache 的参数

考点多在 server result cache 上

result_cache_max_size=0 则表示禁用该特性。
result_cache_max_result 则控制单个缓存结果可以占总的 Server Result Cache 大小的百分比。

参数 result_cache_mode 用于控制 Server Result Cache 的模式，有 3 个可选设置。

result_cache_mode=AUTO：　则优化器自动判断是否将查询结果缓存。
result_cache_mode=MANUAL：　则需要通过查询提示来告诉优化器是否缓存结果。
result_cache_mode=FORCE：　则尽可能地缓存查询结果，通过提示：no_result_cache 可以拒绝缓存。

例：测试 Result Cache 的性能。

当前 result_cache_mode=MANUAL：则需要通过查询提示来告诉优化器是否缓存结果。

```
SQL> set autotrace on;

SQL> select /*+ result_cache */ count(*) from scott.emp;

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


执行计划
----------------------------------------------------------
Plan hash value: 2937609675

-------------------------------------------------------------------------------------------
--------
| Id  | Operation          | Name                         | Rows  | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT   |                              |     1 |     1   (0)|
00:00:01 |
|   1 |  RESULT CACHE      | 5cjd9qv8vs58n2baj8u7sfwmq7 |       |            |
|
|   2 |   SORT AGGREGATE   |                              |     1 |            |
|
|   3 |    INDEX FULL SCAN| PK_EMP                       |    14 |     1   (0)|
00:00:01 |
-------------------------------------------------------------------------------------------
--------

Result Cache Information (identified by operation id):
------------------------------------------------------

   1 - column-count=1; dependencies=(SCOTT.EMP); attributes=(single-row);
name="select /*+ result_cache */ count(*) from
 scott.emp"


统计信息
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          1  consistent gets
```

```
          0  physical reads
          0  redo size
        419  bytes sent via SQL*Net to client
        416  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

看第一次执行，逻辑读 consistent gets=1 说明执行了这个 SQL 的执行计划。

再做一次：

```
SQL> select /*+ result_cache */ count(*) from scott.emp;

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


执行计划
----------------------------------------------------------
Plan hash value: 2937609675

-------------------------------------------------------------------------------------
--------
| Id  | Operation          | Name                         | Rows  | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT   |                              |     1 |     1   (0)|
00:00:01 |
|   1 |  RESULT CACHE      | 5cjd9qv8vs58n2baj8u7sfwmq7 |       |            |
|
|   2 |   SORT AGGREGATE   |                              |     1 |            |
|
|   3 |    INDEX FULL SCAN| PK_EMP                       |    14 |     1   (0)|
00:00:01 |
-------------------------------------------------------------------------------------
--------

Result Cache Information (identified by operation id):
------------------------------------------------------
```

```
   1   -   column-count=1;   dependencies=(SCOTT.EMP);   attributes=(single-row);
name="select /*+ result_cache */ count(*) from
 scott.emp"
```

统计信息

```
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          0  consistent gets
          0  physical reads
          0  redo size
        419  bytes sent via SQL*Net to client
        416  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

第二次，逻辑读 consistent gets=0 了，说明没有执行该 SQL 执行计划，它从 RESULT CACHE 中直接拿到结果显示给你了。

删去/*+ result_cache */提示，比较不使用 hints 的效果。

16.4 11g 内存管理(PPT-II-335-345)

Oracle11g 比较大的改进之一，是在 Oracle 实例的内存管理方面，也就是 Oracle11g 中的新的内存管理特性--自动化内存管理(automatic memory management，AMM)。它将 SGA 和 PGA 统一分配，其作用是实现了 SGA 和 PGA 之间自动转换内存空间。

16.4.1 内存组件管理历史

手动 PGA 和 SGA 管理 （8i 之前）
自动 PGA 管理 (9i)
自动 SGA 管理 (10g)
自动 memory 管理 （11g)

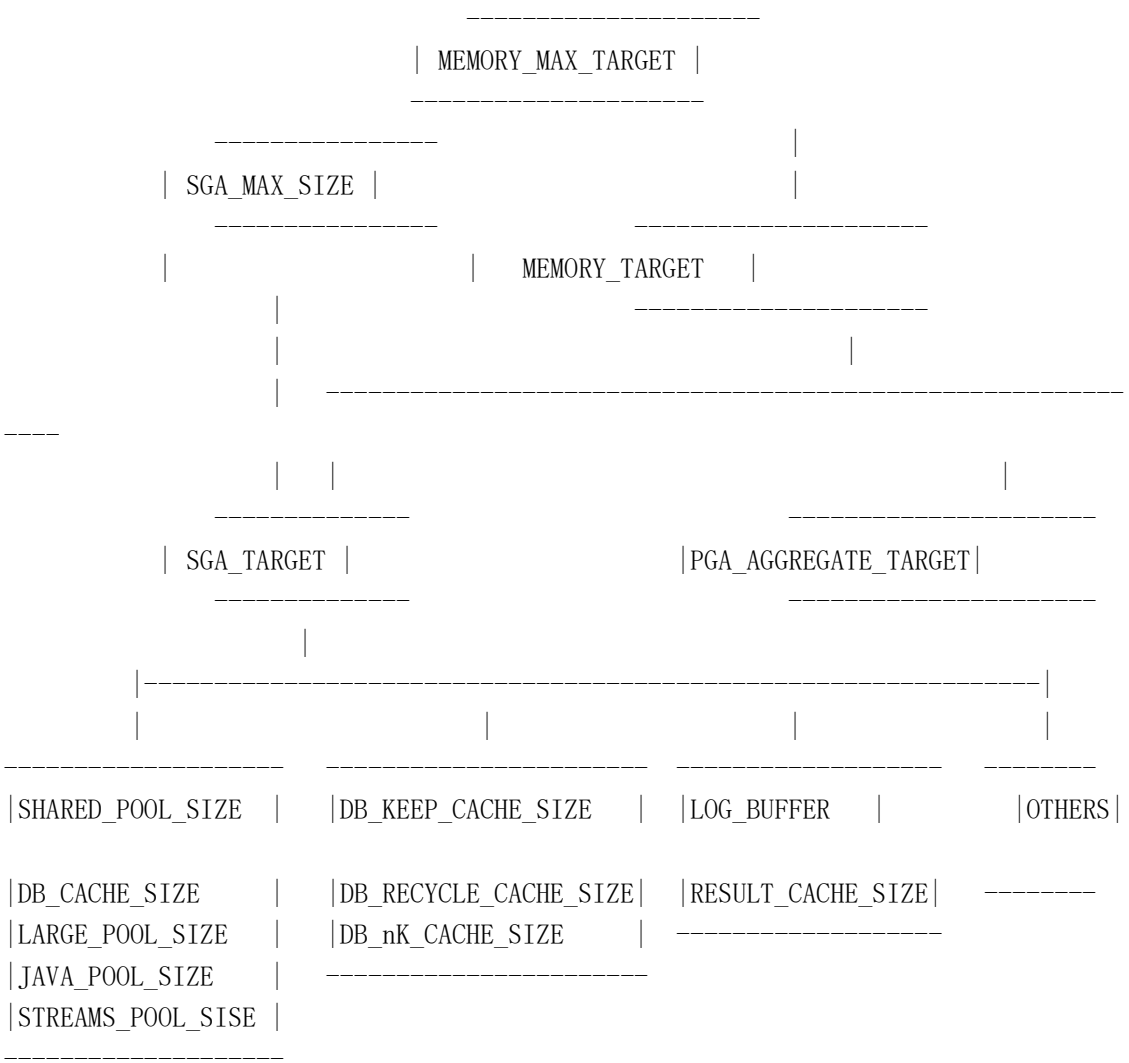
16.4.2 AMM 的新增内存参数

memory_max_target:

该参数设置 Oracle 实例可以使用的最大内存量。memory_target<=memory_max_target.这个参数是静态初始化参数。

memory_target:
该参数设置整个 oracle 数据库实例可以使用的内存量，自动的调节 SGA 与 PGA 的大小。该参数是可以动态调整的初始化参数。

考点：
1）如果你的初始化参数文件里没有指定 memory_max_target，即使用它的缺省值，那么 memory_max_target=memory_target
2）如果你不想设置 SGA 与 PGA 的最小值，可以把 sga_target 与 pag_aggregate_target 初始化参数都设置为 0。



16.4.3 从 10g 的自动 sga 和自动 pga 过渡到 11g 全自动内存管理。

1）得到 SGA 当前值

show parameter sga_target;　　//比如是 600M

2）执行以下查询，确定 PGA 给定时间内的最大值

SQL> select value/1024/1024 from v$pgastat where name='maximum PGA allocated'; //比如是 300M

如果执行了下列语句，你将得到一个错误的 PGA 估计值

SQL> show parameter pga_aggregate_target; //比如是 2G, 比 300M 大多了，其实 300M 才是数据库在单一时间里最多使用的 PGA。

3）11g 的参数 memory_target 应该设置为：memory_target=600M(SGA)+300M(PGA)

4）11g 的参数 memory_max_target 参数如果不设，数据库将自动使 memory_max_target=memory_target.

5) 最后使 sga_target=0, pga_aggregate_target=0

16.4.4 自动内存管理的相关性

SGA 中可以自动调整的只有 5 个池分别是：

DB_CACHE_SIZE
SHARED_POOL_SIZE
LARGE_POOL_SIZE
JAVA_POOL_SIZE
STREAMS_POOL_SIZE

3）还有几个池是不能自动调整的分别是：

LOG_BUFFER

DB_KEEP_CACHE_SIZE
DB_RECYCLE_CACHE_SIZE
DB_nK_CACHE_SIZE

15.4.5 检查内存组件的当前值

SQL> select * from v$memory_target_advice order by memory_size;

MEMORY_SIZE	MEMORY_SIZE_FACTOR	ESTD_DB_TIME	ESTD_DB_TIME_FACTOR	VERSION
202	.5	1508	1.1186	0
303	.75	1409	1.0452	0
404	1	1348	1	0
505	1.25	1344	.9972	0
606	1.5	1344	.9972	0
707	1.75	1344	.9972	0
808	2	1344	.9972	0

已选择 7 行。

memory_size_factor 值为 1 的值是实例分配到的 memory_target 的当前值。

查看 V$PGA_TARGET_ADVICE 可预测 PGA 的 cache hit 率（考点）。

```
SELECT                                    round(PGA_TARGET_FOR_ESTIMATE/1024/1024)
target_mb,ESTD_PGA_CACHE_HIT_PERCENTAGE cache_hit_perc,
       ESTD_OVERALLOC_COUNT FROM V$PGA_TARGET_ADVICE;
```

相关动态视图
V$MEMORY_DYNAMIC_COMPONENTS
V$MEMORY_RESIZE_OPS
V$PGA_TARGET_ADVICE
v$sga_dynamic_components
v$sga_dynamic_free_memory

v$sga
v$sgainfo
v$sgastat
v$pgastat
v$buffer_pool

总之：11g 在简化 DBA 基本工作上还是下了很大功夫。可是这样也掩盖了一些技术细节，Oracle 正在逐步把内存的管理变成一个黑盒子，当然这也是相关算法更加稳定作为基础的。总体来说，利大于弊。

16.5 查看 Oracle 的执行计划：

16.5.1 使用 AUTOTRACE

sys 用户缺省配置了 autotrace。它只能在 SQL*PLUS 里使用。作用：自动跟踪为 SQL 语句生成一个执行计划，并提供与该语句的处理有关的统计。

系统安装后 sys 用户已经有了 autotrace 功能

普通用户想使用 autotrace 需要建立授权

autotrace 的结果将放在一个叫 plan_table 的表中

```
SQL> desc plan_table;
SQL> drop table plan_table;
```

sys：重新建立这个表，执行下面脚本

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan
```

建立 plustrace 角色

```
SQL> @$ORACLE_HOME/sqlplus/admin/plustrce
```

把 plustrace 角色授给用户

```
SQL> grant plustrace to scott;
```

进入用户可以使用 autotrace 了

```
SQL> conn scott/scott
SQL> set autotrace on;
```

显示执行计划常用命令

```
SQL>set autotrace on;
或
SQL>set autotrace traceonly explain;     //仅显示执行计划策略。
SQL>set autotrace traceonly statistics; //仅显示统计信息。
```

如何解读执行计划的内容

```
SQL> create table emp1 as select * from emp;
SQL> create table dept1 as select * from dept;
```

例：查找在 NEW YORK 工作的经理是谁？

```
SQL> set autotrace traceonly explain;
```

```
SQL> select e.ename,e.job,d.loc,d.deptno from emp1 e ,dept1 d where e.deptno=d.deptno and e.job='MANAGER' and d.loc='NEW YORK';
```

执行计划

```
----------------------------------------------------------
Plan hash value: 619452140

------------------------------------------------------------------------------
| Id  | Operation          | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |       |     1 |    47 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN         |       |     1 |    47 |     7  (15)| 00:00:01 |
|*  2 |   TABLE ACCESS FULL| DEPT1 |     1 |    21 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS FULL| EMP1  |     3 |    78 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("E"."DEPTNO"="D"."DEPTNO")
   2 - filter("D"."LOC"='NEW YORK')
   3 - filter("E"."JOB"='MANAGER')

Note
-----
   - dynamic sampling used for this statement

SQL>
```

Operation 是一棵树，执行顺序并不是按照 Id 的顺序，它是先完成叶子节点，然后是分支节点可能采用并行 IO。

看到执行计划是这样的顺序

1）读 dept1 表，并使用谓词 where 过滤后剩下一行记录，
2）读 emp1 表，并使用谓词 where 过滤后剩下三行记录，
3）将过滤后的两个子集（一行和三行）做 HASH JOIN。

连接又叫二元操作，选择又叫一元操作，这个执行计划没有先将 emp1 和 dept1 来个笛卡尔乘积（二元操作），然后再过滤（一元操作），那样的话成本太高了。早期的优化器（RBO）是基于规则的，它执行启发式策略：尽早执行选择和投影的一元操作，如果执行连续的二元操作，首先执行较小的连接。而基于成本的优化器（CBO）就更有优势了，它多了一条优点：考虑数据库对象的最新统计数据。

CBO 中表的三种连接方法：

1）nested_loop join（嵌套循环连接），

指定一个表作为驱动表（driving table),也叫外部表（outer table），连接中的另一表被称为内部表（inner table),Oracle 对驱动表中的每行数据都要读取内部表的所有行，

适用：driving table 比较小，而 inner table 较大且有索引，此法效率高。使用较小内存，第一次返回结果较快。

2）hash join（散列连接）

在连接两个表时，Oracle 用其较小的表在连接键上构建一个 hash 表，然后,Oracle 对较大的表进行搜索，并从 hash 表中返回被连接的行。

适用：大数据连接，连接字段缺乏索引，hash 连接比嵌套循环有效。但需要大内存。第一次结果返回较慢

3）sort_merge join（分类合并连接）

分类连接操作将连接键上的输入值进行分类，合并连接操作将已分类的表进行合并，如果输入值已经按照连接列进行了分类，那么没必要为每个行源执行分类连接操作。

适用：两个大表，连接字段缺乏索引，但已有排序。此法为最优化吞吐量设计，并且在结果没有全部找到前不返回数据。

可以使用 hints 强制优化器使用以下连接：

/*+ use_nl（emp1,dept1）*/ 表示采用嵌套循环连接。
/*+ use_merge（emp1,dept1）*/ 表示采用排序合并连接。
/*+ use_hash（emp1,dept1）*/ 表示采用哈希连接。
/*+ leading(emp1) */ 表示选择 emp 为驱动表。
/*+ ordred */ 按照 from 列出的表顺序进行连接。

如：SELECT /*+ use_nl（e,d）*/ e.ename,e.job,d.loc,d.deptno from emp1 e ,dept1 d where e.deptno=d.deptno and e.job='MANAGER' and d.loc='NEW YORK';

需要注意的是如果 SQL 语句中表使用别名的话,那么 hints 中也必须使用表的别名,否则 hints 将不会生效。

16.6 SQL 计划管理 SPM

SQL 的一些顾问可以调优 SQL 语句，但最多是一种被动的机制，而且需要 DBA 干预。而 SPM

（SQL Plan Management)可以提供一种预防性的机制，为保持执行计划始终处于最优状态。Oracle 引入 SQL 计划基线的概念，目的是使 SQL 执行计划得以进化。

16.6.1 概念：

如果一条语句经常被使用，Oracle 为了防止因数据库环境的变化造成原执行计划出现性能退化，又会生成新的执行计划。

既然优化程序可以对一条语句生成若干执行计划，累积的这些计划组成了一部计划历史，在其中有标记为 accepted 的计划，这样的计划是比较优异的(成本较低的)，计划基线就是计划历史中那些 accepted 的执行计划。标记为 unaccepted 的计划是非计划基线中的计划。生成一条语句的最初的计划肯定是 accepted, 因为它无从比较。以后再生成的计划就可以同计划基线中的计划去比较了。

16.6.2 一个计划成为计划基线的两种办法；

1 自动捕获； 设 OPTIMIZER_CAPTURE_SQL_PLAN_BASELINE 为 TRUE，数据库自动捕获新计划，但新计划不管是否优异，暂不列入 accepted 计划行列，要以观后效,一个未认可的计划必须优于认可的计划才能进化为计划基线。

2 手动装载； 手动在数据库中装载它们，一般是升级或已经测试过的，新计划直接列入 accepted。因为你装载的计划是你自认为最优的计划。

16.6.3 管理 SPM

SMB（SQL Management Base），又叫 SQL 管理库，所有的 sql 计划基线，sql profile 等信息都保存在 SYSAUX 表空间。SBM 占用 SYSAUX 的比例缺省是 10%，可以在 1%-50%任意设置它，数据库执行每周一次的定期清除以删除不使用的 SQL 基线.

可以利用 Oracle 提供了 DBMS_SPM 程序包或 OEM 管理 SQL 计划基线。

管理视图：dba_sql_plan_baselines

16.7 真实应用测试（Total Recall）(PPT-II-371)

如何判断一个重大的系统更改(如升级到某个 Oracle 新版本)对性能产生的影响。Oracle11g 提供了一个很容易使用的工具。使得生产系统在引入这些更改前，能在测试系统环境下测试它们对系统的影响。

Total Recall 由两部分组成：数据库重放（Database Replay）和 SQL 性能分析器（SQL Performence Analyzer）

Database Replay 数据库重放）

(
软件和硬件环境会发生变化，如单一数据库升级为 RAC，升级操作系统，扩容硬件，新应用程序添加等。Database Replay 通过捕获生产服务器上的工作负荷（收集到一个文件中），然后在测试系统上重放工作负荷，以此来评估测试系统上的性能变化。

16.7.1 使用 Database Replay 时有几个步骤：

1）捕获工作负荷：在生产机上做，结果写进一个二进制文件，又叫捕获文件，它包括启动捕获以来所有外部客户端的请求，比如 SQL 语句，绑定变量，业务信息，登录或退出以及 DML/DDL 语句，但不包括后台进程及调度 job

2）预处理工作负荷：把捕获到的负荷转换成重放文件，如果版本相同这一过程最好在测试机上完成，这时的测试机只是生产机的一个克隆，把捕获文件传到测试机，预处理这个文件，得到 Replay 所需要的所有的元数据。

3）然后把你的测试机进行更改，比如升级数据库版本。

4）重放工作负荷：在测试机上把生产机捕获的工作量 Replay， 重放使用一个客户端程序，叫做重放客户端，replay client 可以一个也可以多个。
5）分析和报告，重放后会产生一个详尽的报告，报告里将捕获数据和重放数据一起做比较和分析。

Oracle 提供程序包 dbms_workload 实现 Database Replay

考点：
1）数据库重放的两个主要应用场合：一个是系统硬软件升级要测试新版本的业务压力，另一个是将数据库存储数据库存储改换为 ASM 方式时。改换为 ASM 方式。
2）一个重放客户端可以重放来自许多会话的语句。

16.7.2 使用 SQL Performence Analyzer（SPA)

测试数据库升级的 SQL 工作负荷响应时间。它分析并比较系统更改前后的 SQL 性能，提出组织性能退化的建议。

思路同 Database Replay 差不多，仅仅是针对 SQL 性能的分析。Oracle 提供程序包 dbms_splpa 实现 SQL Performence Analyzer。

1）捕捉生产机的产品 SQL 工作负荷
2）装载 SQL 调优集，将 STS 导入测试机
以下在测试机上做：
3）测试机上创建 SQL 性能分析器任务
4）分析更改前的 SQL 工作负荷
5）分析升级后的 SQL 工作负荷

6）比较 SQL 性能
7）生成分析报告

完

附录 Oracle 一周备份计划范例（Linux 的 RMAN CATALOG 备份策略_基于 9i 版本）（选）

前言

对于 Oracle 数据库的备份与恢复，尽管存在热备，冷备以及逻辑备份之外，使用最多的莫过于使用RMAN进行备份与恢复。而制定RMAN备份策略则是基于数据库丢失的容忍程度，即恢复策略来制定。在下面的备份策略中，给出的是一个通用的备份策略。在该备份策略中，使用了 catalog 方式来保持备份脚本以及备份信息。在实际环境中应根据具体的情况对此进行相应的调整。

步骤

确认备份可用空间以及备份路径，根据需要创建相应文件夹

1.对于账户的连接创建一个 connect.rcv，该文件包含连接到 target 和 catalog 信息

2.创建通用的脚本用于删除过旧的备份和备份控制文件以及备份归档日志

global_del_obso --删除过旧的备份

global_bkctl --备份控制文件

global_arch --备份归档日志

3.创建 0,1,2 级增量备份

这三个脚本中均包含第 2 步的 3 个脚本，先调用 global_del，然后做增量备份，最后备份归档日志 global_arch 和控制文件 global_bkctl

4.创建 inc0.rcv,inc1.rcv,inc2.rcv

文件内容包含调用 @@/<dir>/connect.rcv 以及 run{execute global script scriptname;}exit;

5.编辑第 4 步的三个文件分别为 inc0.sh,inc1.sh,inc2.sh

nohup $ORACLE_HOME/bin/rman cmdfile=/u03/bk/scripts/inc1.rcv

log=/u03/bk/scripts/inc0.log append &

6.使用 crontab 制定备份计划

具体实现

1.连接脚本

connect.rcv 文件内容

```
connect catalog rman/rman@david;
connect target sys/oracle@austin;
```

catalog 的通用脚本

```
rman cmdfile=/u03/bk/scripts/connect.rcv --(在 rman 中使用外部脚本)
相当于：
rman catalog rman/rman@david target sys/oracle@austin
```

2.建立通用脚本

--删除不必要的备份

```
RMAN> create global script global_del_obso  comment 'A script for obsolete backup
and delete it' {
crosscheck archivelog all;
delete noprompt expired archivelog all;
allocate channel ch1 device type disk;
delete noprompt obsolete recovery window of 7 days;
release channel ch1;
}
```

--备份控制文件脚本

```
RMAN> create global script global_bkctl comment 'A script for backup control
file'{
allocate channel ch1 device type disk;
backup as compressed backupset
current controlfile reuse
format='/u03/bk/backupctl.ctl'
tag='bkctl';
release channel ch1;
}
```

--备份归档日志脚本

```
RMAN> create global script global_arch comment "backup archivelog and then delete
it" {
allocate channel ch1 device type disk;
allocate channel ch2 device type disk;
sql "alter system archive log current";    --归档当前的联机日志
set limit channel ch1 readrate=10240;        --(读取速率 10M)
set limit channel ch1 kbytes=4096000;        --(备份片的大小)
backup as compressed backupset
format='/u03/bk/arch_%d_%U'
tag='bkarch'
archivelog all delete input;
release channel ch1;
release channel ch2;
}
```

--使用 list 查看所有的已建立的全局脚本

list global script names;　　--(列出所有的脚本)

delete globals script script_name;　　--(删除脚本)

RMAN> list global script names;

List of Stored Scripts in Recovery Catalog

Global Scripts

Script Name

Description

---

global_arch

backup archivelog and then delete it

global_bkctl

A script for backup control file

global_del_obso

A script for obsolete backup and delete it

3.创建 0，1，2 级增量备份脚本(注：每个脚本备份前会执行删除过旧的备份，脚本尾部会调用另外两个脚本来备份归档日志及控制文件)

--创建 0 级增量备份

```
RMAN> create global script global_inc0 comment "backup database as incrmental level 0"{
execute global script global_del_obso;
allocate channel ch1 device type disk;

allocate channel ch2 device type disk;
set limit channel ch1 readrate=10240;
set limit channel ch1 kbytes=4096000;
set limit channel ch2 readrate=10240;
set limit channel ch2 kbytes=4096000;
backup as compressed backupset
incremental level 0 database

format='/u03/bk/inc0_%d_%U'
tag='inc0';
release channel ch1;
release channel ch2;
execute global script global_arch;
execute global script global_bkctl;
 }
```

--创建 1 级增量备份

```
RMAN> create global script global_inc1 comment "backup database as incrmental level 1"{
execute global script global_del_obso;
allocate channel ch1 device type disk;
allocate channel ch2 device type disk;
set limit channel ch1 readrate=10240;
set limit channel ch1 kbytes=4096000;
set limit channel ch2 readrate=10240;
set limit channel ch2 kbytes=4096000;
backup as compressed backupset
incremental level 1 database
format='/u03/bk/inc1_%d_%U'
```

```
tag='inc1';
release channel ch1;
release channel ch2;
execute global script global_arch;
execute global script global_bkctl;
}
```

--创建 2 级增量备份

```
RMAN> create global script global_inc2 comment "backup database as incrmental level 2"{
execute global script global_del_obso;
allocate channel ch1 device type disk;
allocate channel ch2 device type disk;
set limit channel ch1 readrate=10240;
set limit channel ch1 kbytes=4096000;
set limit channel ch2 readrate=10240;
set limit channel ch2 kbytes=4096000;
backup as compressed backupset
incremental level 2 database
format='/u03/bk/inc2_%d_%U'
tag='inc2';
release channel ch1;
release channel ch2;
execute global script global_arch;
execute global script global_bkctl;
}
```

--在 rman 中检验在 rman 中写的脚本 global_inc0、global_inc1、global_inc2,因为 RMAN 不会自动检查,下面的语句用来执行脚本(检验)

```
RMAN> run{
execute global script global_inc0;
execute global script global_inc1;
execute global script global_inc2;
 }
```

--查看备份完成情况

```
list backupset summary;
```

4.建立 shell 脚本，让 linux 自动执行脚本

a. vi inc0.rcv，inc1.rcv ,inc2.rcv --注意不同的文件执行不同的备份脚本

@@/u03/bk/scripts/connect.rcv --(rman 下的脚本去调用其他脚本用@@符号)(调用脚本不需要分号)

```
run{
execute global script gloal_inc0;
}
exit;
```

b. 编辑 shell 文件

vi inc0.sh

nohup $ORACLE_HOME/bin/rman cmdfile=/u03/bk/scripts/inc0.rcv log=/u03/bk/scripts/inc0.log append &

vi inc1.sh

nohup $ORACLE_HOME/bin/rman cmdfile=/u03/bk/scripts/inc1.rcv log=/u03/bk/scripts/inc0.log append &

vi inc2.sh

nohup $ORACLE_HOME/bin/rman cmdfile=/u03/bk/scripts/inc2.rcv log=/u03/bk/scripts/inc0.log append &

--注意：nohup 与&表示将脚本放入后台执行

c.使用 crontab 建立一个备份计划

crontab -e

```
#min    hour    date  mon  day(星期)    command
30       1       *     *    0           /u03/bk/scripts/inc0.sh
30       1       *     *    1           /u03/bk/scripts/inc2.sh
```

```
30      1      *      *      2          /u03/bk/scripts/inc2.sh
30      1      *      *      3          /u03/bk/scripts/inc2.sh
30      1      *      *      4          /u03/bk/scripts/inc1.sh
30      1      *      *      5          /u03/bk/scripts/inc2.sh
30      1      *      *      6          /u03/bk/scripts/inc2.sh
```

d.重启 crontab 服务(如果没有启动)

```
# /sbin/service crond status  --用于检查 crontab 服务状态
# /sbin/service crond stop //关闭服务
# /sbin/service crond restart //重启服务
# /sbin/service crond reload //重新载入配置
```

使 crontab 服务在系统启动的时候自动启动：

在/etc/rc.d/rc.local 这个脚本的末尾加上：

```
/sbin/service crond start
```

e.从上面的备份策略来看，即

周日执行 0 级增量备份，相当于全备

周一，周二，周三执行 2 级增量备份

周四执行 1 级增量备份

周五，周六执行 2 级增量备份

f.编辑好的 shell 脚本测试

```
chmod 755 *.sh          --给 shell 脚本加权限
```

测试脚本　例如./inc0.sh

总结

1.backup controlfile in each scripts tail (在脚本的尾部备份控制文件)

2.Delete obsolete backupset in each scripts threshold (删除旧的备份)

3.Switch logfile before backup database; (在数据库备份以前切换日志，这样相当于备份了当前的联机重做日志)

4.Chmod u+x*.sh