**备份与恢复**

云和恩墨(北京)信息技术有限公司

技术顾问 燕鑫

http://www.enmotech.com

**文档控制：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序** | **版本号** | **更改人** | **日期** | **备注** |
| 1 | 1.0版 | 燕鑫 | 2018-09-12 | 初始版本 |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1. Recovery Manager - 6 -](#_Toc7095410)

[1.1 Recovery Manager Architecture - 6 -](#_Toc7095411)

[1.1.1 RMAN Environment - 6 -](#_Toc7095412)

[1.1.2 RMAN Channels - 8 -](#_Toc7095413)

[1.1.3 RMAN Repository - 10 -](#_Toc7095414)

[1.1.4 Media Management - 10 -](#_Toc7095415)

[1.1.5 Fast Recovery Area - 10 -](#_Toc7095416)

[1.2 Configuring the RMAN Environment - 13 -](#_Toc7095417)

[1.2.1 DEFAULT DEVICE TYPE - 14 -](#_Toc7095418)

[1.2.2 DEVICE TYPE and BACKUP TYPE - 15 -](#_Toc7095419)

[1.2.3 PARALLELISM - 17 -](#_Toc7095420)

[1.2.4 CHANNEL - 20 -](#_Toc7095421)

[1.2.5 CONTROLFILE AUTOBACKUP - 25 -](#_Toc7095422)

[1.2.6 CONTROLFILE AUTOBACKUP FORMAT - 26 -](#_Toc7095423)

[1.2.7 RETENTION POLICY - 29 -](#_Toc7095424)

[1.2.8 BACKUP OPTIMIZATION - 33 -](#_Toc7095425)

[1.2.9 ARCHIVELOG DELETION POLICY - 37 -](#_Toc7095426)

[1.2.10 MAXSETSIZE - 37 -](#_Toc7095427)

[1.2.11 DATAFILE BACKUP COPIES - 38 -](#_Toc7095428)

[1.2.12 EXCLUDE - 39 -](#_Toc7095429)

[1.3 RMAN Backup Concepts - 39 -](#_Toc7095430)

[1.3.1 Consistent and Inconsistent RMAN Backups - 39 -](#_Toc7095431)

[1.3.2 Backup Sets - 40 -](#_Toc7095432)

[1.3.3 Image Copies - 44 -](#_Toc7095433)

[1.3.4 Multiple Copies of RMAN Backups - 45 -](#_Toc7095434)

[1.3.5 Control File and Server Parameter File Autobackups - 46 -](#_Toc7095435)

[1.3.6 Incremental Backups - 46 -](#_Toc7095436)

[1.3.7 Backup Retention Policies - 60 -](#_Toc7095437)

[1.4 Backing Up the Database - 61 -](#_Toc7095438)

[1.4.1 Specifying Backup Output Options - 61 -](#_Toc7095439)

[1.4.2 Backing Up Database Files with RMAN - 71 -](#_Toc7095440)

[1.5 RMAN Data Repair - 77 -](#_Toc7095441)

[1.5.1 Incarnation - 77 -](#_Toc7095442)

[1.5.2 Validating Database Files and Backups - 77 -](#_Toc7095443)

[1.5.3 Preparing for Complete Database Recovery - 86 -](#_Toc7095444)

[1.5.4 Performing Complete Database Recovery - 93 -](#_Toc7095445)

[1.5.5 Performing Complete Recovery of CDBs - 101 -](#_Toc7095446)

[1.5.6 Performing Block Media Recovery - 103 -](#_Toc7095447)

[1.6 Tablespace Point-in-Time Recovery（TSPITR） - 105 -](#_Toc7095448)

[1.6.1 Overview of RMAN TSPITR - 105 -](#_Toc7095449)

[1.6.2 Planning and Preparing for TSPITR - 108 -](#_Toc7095450)

[1.6.3 TSPITR - 110 -](#_Toc7095451)

[1.7 Recovering Tables and Table Partitions from RMAN Backups - 127 -](#_Toc7095452)

[1.7.1 总概 - 127 -](#_Toc7095453)

[1.7.2 实验 - 130 -](#_Toc7095454)

[1.8 对RMAN的监控 - 137 -](#_Toc7095455)

[1.8.1 RMAN基本工作原理 - 137 -](#_Toc7095456)

[1.8.2 监控 - 140 -](#_Toc7095457)

[2. Flashback Database（Physical） - 145 -](#_Toc7095458)

[2.1 基础概念 - 145 -](#_Toc7095459)

[2.2 Limitations of Flashback Database - 147 -](#_Toc7095460)

[2.3 Restore Points - 147 -](#_Toc7095461)

[2.3.1 Normal Restore Points - 147 -](#_Toc7095462)

[2.3.2 Guaranteed Restore Points - 148 -](#_Toc7095463)

[2.3.3 Logging for Flashback Database and Restore Points - 148 -](#_Toc7095464)

[2.4 Flashback Database的环境配置和性能监控 - 160 -](#_Toc7095465)

[2.5 Flashback Database实验 - 160 -](#_Toc7095466)

[2.5.1 Prerequisites of Flashback Database - 160 -](#_Toc7095467)

[2.5.2 Performing a Flashback Database Operation - 160 -](#_Toc7095468)

[3. Flashback Table（Logical） - 173 -](#_Toc7095469)

[3.1 Rewinding a Table - 174 -](#_Toc7095470)

[3.1.1 Prerequisites for Flashback Table - 174 -](#_Toc7095471)

[3.1.2 Performing a Flashback Table Operation - 174 -](#_Toc7095472)

[3.2 Rewinding a DROP TABLE Operation - 180 -](#_Toc7095473)

[3.2.1 Prerequisites of Flashback Drop - 180 -](#_Toc7095474)

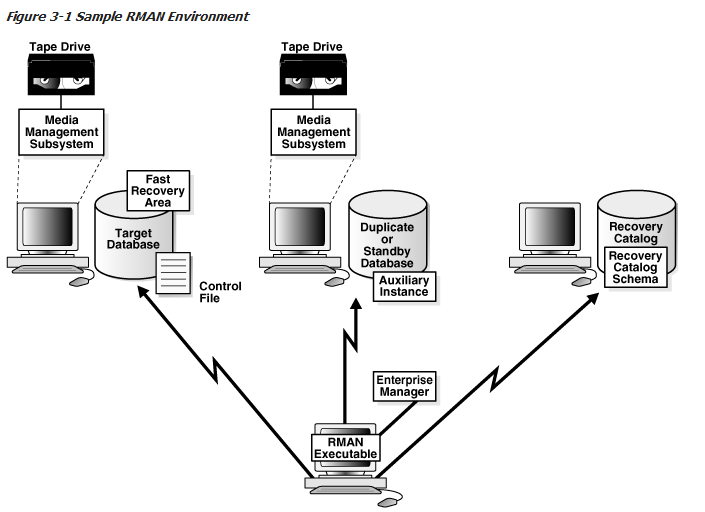
[3.2.2 Performing a Flashback Drop Operation - 180 -](#_Toc7095475)

# Recovery Manager

## Recovery Manager Architecture

### RMAN Environment

以下是RMAN的组件（components）。



#### RMAN Client

RMAN客户端是执行各种备份恢复指令的工具，该工具可以在任意一台可以通过oracle net链接到target database的服务器上。

[oracle@host01 ~]$ rman

Recovery Manager: Release 12.2.0.1.0 - Production on Wed Sep 12 10:37:06 2018

Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.

RMAN>

#### Target Database

目标数据库，就是用rman连接到的准备进行备份或者恢复的数据库。Rman会把读target database的控制文件，收集数据库相关的元数据，并把自己操作相关的信息，比如说配置啊等等，记录在target database的controlfile中。

同时，rman客户端所执行的所有备份恢复操作都只是指令，真正的工作均是跑在target database上的server session完成的。

如下是rman client中连接target database的方式：

RMAN> connect target "sys/oracle@prod4 as sysdba"

connected to target database: PROD4 (DBID=1716001866)

#### Recovery Catalog Database

拥有recovery catalog的数据库。一个Recovery catalog也是跟控制文件对于rman的作用一样，所以该组件是可有可无的，记录rman执行备份恢复所需要的元数据。

他比控制文件有俩个优势：

1. recovery catalog database可以记录多个target database的元数据
2. 控制文件丢失损坏不会导致target database进行备份恢复的相关元数据丢失，比如备份集的位置呀、有哪些可用备份集呀等等
3. 比控制文件可以保存更多更长久的历史记录

#### Recovery Catalog Schema

The user within the recovery catalog database that owns the metadata tables maintained by RMAN。

Recovery catalog数据库中拥有被rman维护的metadata tables的用户。

#### Physical Standby Database

之所以把物理备库也列为rman的一个组件，是因为物理备库和主库有相同的DBID，逻辑备库则被视为另外一个不同的数据库，因为逻辑备库可以有自己的dbid。

#### Fast Recovery Area

这是rman默认的各种备份的存放路径，该路径下还可以存放归档、flashback logs。

#### Others

media management software和media management catalog，前者是特定供应商的（vendor-specific）可以让rman把数据库备份到别的存储上（而不是本地磁盘，比如tape上）的工具软件。

后者是管理前者元数据的vendor-specific repository。

### RMAN Channels

Rman client指导数据库的server session来完成所有的备份恢复工作。对于linux来说，一个server session对应一个server process。

一个rman channel代表一个到设备的数据流（a data stream to device）。在整个备份恢复期间，channel从input device读入数据，处理后，再写入output device。

大部分的指令需要通过channel传达给server session，那么对于channel的分配，可以手动分配或者再配置中配置并行由rman自动分配。

run{

allocate channel ch00 type disk;

allocate channel ch01 type disk;

allocate channel ch02 type disk;

allocate channel ch03 type disk;

restore database;

release channel ch00;

release channel ch01;

release channel ch02;

release channel ch03;

}

或者配置：

RMAN> show all;

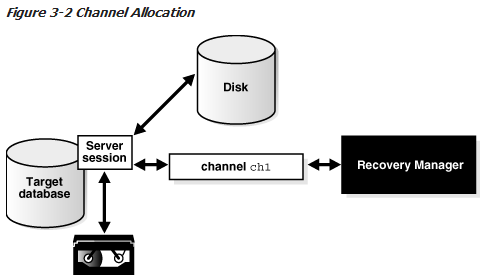
using target database control file instead of recovery catalog

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default

下图展示，channel是通过在目标数据库实例上发起一个server session，来建立rman client和target database的连接的：



#### Channels and Devices

Rman支持俩种设备：disk和SBT（system backup to tape）。

如果是用disk channel来backup，那么能放数据文件的disk device就能放备份文件（所以备份集可以直接放在ASM上）。

那么如果是想把数据库备份到non-disk media，就需要特定供应商的media management software来管理并分配该软件支持的channels。

#### Automatic and Manual Channels

### RMAN Repository

The RMAN repository is the collection of metadata about the target databases that RMAN uses for backup, recovery, and maintenance.

Rman资料库是RMAN用以进行备份、恢复、维护的target database的元数据的一个集合。

### Media Management

RMAN client和media manager不会直接通信，而是rman client连接target dabase，并指导target database与media manger通信。

### Fast Recovery Area

#### 配置FRA

Fast recovery area（FRA）可以用来存放controlfile、online redo log、archive log、rman backup。

配置FRA很简单：

SQL> show parameter db\_recovery\_file

NAME TYPE VALUE

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

db\_recovery\_file\_dest string /u01/app/oracle/fast\_recovery\_area/PROD4

db\_recovery\_file\_dest\_size big integer 8016M

FRA下的文件是OMF（Oracle Managed Files）的方式管理的。

这里有几点需要特别注意的：

1. 配置FRA数据库开着就能配置
2. 一定要先设置db\_recovery\_file\_dest\_size（FRA的大小）的值，然后再设置db\_recovery\_file\_dest（FRA的路径）。
3. db\_recovery\_file\_dest可以是文件系统，也是可以ASM，但就是不能是裸设备（raw disk）。
4. RAC环境下，各个实例FRA的配置一定要一致，并且必须配置在共享存储上。
5. db\_recovery\_file\_dest\_size的配置注意俩点：  
   其一，数据文件的block 0和operating system block header的大小是要考虑在其中的，应当为其预留百分之10的空间，总之就是要设置的够大，要按真实的文件大小计算，而不是数据库中查询出的大小。  
   其二，FRA的大小并不考虑文件系统对文件mirror的情况。比如说如果ASM一个磁盘组是normal冗余，那在该磁盘组上配置FRA的大小就不能超过磁盘组大小的2分之一，如果是high，那就不能超过3分之一。

FRA越大越好，如果多个数据库的FRA路径配置的一样，那要这些数据库中db\_unique\_name不能相同，如果参数文件中没指明db\_unique\_name的，那自然db\_name就不能相同。

FRA别跟数据库的数据文件、控制文件这些放在一个disk上，以防因为磁盘故障，数据库跟备份都没了。

#### FRA中管理的文件

FRA中管理的文件分为permanent和transient俩种类型：

1. permanent：  
   Multiplexed copies of the current control file  
   Online redo log files
2. transient：  
   Archived redo log files  
   Foreign archived redo log files  
   Image copies of data files and control files  
   Backup pieces  
   Flashback logs

那么当FRA突然不可访问时，对数据的影响又分为

1. 实例失败或hang住
2. 对实例无影响

我们就说会导致实例失败或hang住的情况，其他就是对实例正常运行无影响的。

##### 导致实例hang住

能导致实例hang住的情况是：归档日志的默认存放路径是FRA，如果此时FRA不可访问，那由于无法正常完成online redo log的归档，会导致实例hang住。

##### 导致实例失败

导致实例失败的情况如下：

1. 控制文件的多路复用，只要是有某一个控制文件的访问路径不可达，那实例就会失败。所以如果有控制文件放在FRA下，而FRA突然不可访问，当然会导致实例失败。
2. Oline redo log有部分存放于FRA的话，FRA不可访问，会导致实例失败，但是如果不是FRA，而是别的存放redolog的路径不可访问，则不会导致实例失败。
3. Flashback log一定是存放于FRA的，  
   如果没有设置guaranteed restore point，那FRA不可访问时，oracle会自动关闭database flashback，并且会在alert日志中记录一条信息。  
   如果没有设置guaranteed restore point，当FRA满了的时候，而其他备份又都在保留期内，那oracle会把flashback log先删掉。  
   如果设置了guaranteed restore point，那当FRA不可访问时，实例会失败。

#### ASM上配置FRA

这个具体怎么配置很简单，就是把db\_recovery\_file\_dest设置为ASM下的一个路径就可以了。

如果不想把备份放在FRA中，但是还是要放在ASM上，是绝对可以的，但是非常不推荐。

如果FRA配在了ASM上，备份集包括归档文件的都在FRA下，ASM会自动删除过期的备份来释放空间。而如果不把这些放在FRA下，自动删除无用备份的功能将不可用，同时数据库运行的时候ASM上删除文件又删不了，那就很awkward。

#### FRA的空间管理

配置FRA的空间大小的计算公式（假设备份策略是每天有增量备份，且冗余为1）：

Disk Quota =

Size of a copy of database +

Size of an incremental backup +

Size of (n+1) days of archived redo logs +

Size of (y+1) days of foreign archived redo logs (for logical standby) +

Size of control file +

Size of an online redo log member \* number of log groups +

Size of flashback logs (based on DB\_FLASHBACK\_RETENTION\_TARGET value)

#### 归档到FRA

SQL> alter system set log\_archive\_dest\_1='LOCATION=USE\_DB\_RECOVERY\_FILE\_DEST';

System altered.

#### 默认使用FRA路径的情况

有以下4种情况，会默认使用FRA的路径（在配置了FRA的情况下）：

1. 执行backup不指定format，同时也没配置channel的format，那默认备份到FRA
2. Control file自动备份的时候，不使用set controlfile autoback format或者不在configuration中配置controlfile autobackup format的时候，control file自动备份到FRA
3. Restore achivelog时，如果配置归档日志路径为LOCATION = USE\_DB\_RECOVERY\_FILE\_DEST，则默认把归档恢复到FRA
4. Recover xx和flashback database的时候，恢复时所需的redo log会被暂时恢复到FRA中，用完后会删除掉这些redo log。
5. 配置归档日志路径为LOCATION = USE\_DB\_RECOVERY\_FILE\_DEST，online redo log自动归档到FRA。

## Configuring the RMAN Environment

Show all展示所有的configuration。

Configure xx来进行配置。

Configure xx clear来还原配置为默认值。

### DEFAULT DEVICE TYPE

该配置是设置当执行backup时，不指定device type，所选择的默认device type。

如果默认是disk，那么disk device需要满足一个条件：disk device的物理块大小必须为要备份的数据文件的逻辑块大小的偶数倍。

一般的disk device是512bytes的物理块大小，而oracle的逻辑块最小2k，正常8k，并不会受到该限制影响。但是一些可写的DVD或者大物理块的disk device就有可能被该限制影响。

该配置的默认值时disk，可选值为SBT：

RMAN> show DEFAULT DEVICE TYPE;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE DEFAULT DEVICE TYPE TO DISK; # default

将其修改为SBT：

RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;

new RMAN configuration parameters:

CONFIGURE DEFAULT DEVICE TYPE TO 'SBT\_TAPE';

new RMAN configuration parameters are successfully stored

显示指定device type可覆盖该配置。

BACKUP DEVICE TYPE sbt DATABASE;

BACKUP DEVICE TYPE DISK DATABASE;

实验：

RMAN> backup tablespace users;

Starting backup at 12-SEP-18

RMAN-00571: ===========================================================

RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============

RMAN-00571: ===========================================================

RMAN-03002: failure of backup command at 09/12/2018 16:30:03

ORA-19554: error allocating device, device type: SBT\_TAPE, device name:

ORA-27211: Failed to load Media Management Library

Additional information: 2

这个报错很显然，因为我没配置tape，但是说明了一个问题，我之前对DEFAULT DEVICE TYPE的修改生效了。下面我们指定device type：

RMAN> backup device type disk tablespace users;

Starting backup at 12-SEP-18

中间omitted

Finished backup at 12-SEP-18

现在我们把配置取消，变回默认值：

RMAN> CONFIGURE DEFAULT DEVICE TYPE clear;

old RMAN configuration parameters:

CONFIGURE DEFAULT DEVICE TYPE TO 'SBT\_TAPE';

RMAN configuration parameters are successfully reset to default value

released channel: ORA\_DISK\_1

可以看到，我们做个配置，都要分配channel，因为要与server session通信，所以说大部分的指令都是要分配channel的。

### DEVICE TYPE and BACKUP TYPE

该设置是在backup到device的时候，不指定backup type时，默认的backup type是什么。

如果是backup type disk，那么可以有3个设置：backupset、uncompressed backupset、copy

如果设置为backupset，那就是多个数据文件备份在一个备份集。这种设置backupset是不压缩的（uncompressed）。

如果设置为compressed backupset，那rman就会对backupset进行binary compress。

如果设置为copy，那么就是一个数据文件一个image copy。注意，rman只能把image copy写到disk device上，而不能写到tape上。

如果设置为SBT，那么久没有copy选项。

原本的默认值为backupset：

RMAN> show DEVICE TYPE;

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default

可以修改的值有：copy和compressed backupset：

RMAN> CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY;

new RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY PARALLELISM 1;

new RMAN configuration parameters are successfully stored

RMAN> CONFIGURE DEVICE TYPE sbt BACKUP TYPE TO COMPRESSED BACKUPSET;

new RMAN configuration parameters:

CONFIGURE DEVICE TYPE 'SBT\_TAPE' BACKUP TYPE TO COMPRESSED BACKUPSET PARALLELISM 1;

new RMAN configuration parameters are successfully stored

现在我们还原默认值：

RMAN> CONFIGURE DEVICE TYPE DISK clear;

old RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK BACKUP TYPE TO COPY PARALLELISM 1;

RMAN configuration parameters are successfully reset to default value

还原的时候一定要指定device type，因为我们没配置device type sbt backup type to xx的时候，show all是看不见device type sbt这一条的，现在我们设置了，我们再show一下：

RMAN> show DEVICE TYPE;

CONFIGURE DEVICE TYPE 'SBT\_TAPE' BACKUP TYPE TO COMPRESSED BACKUPSET PARALLELISM 1;

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default

我们看到现在确实有俩条configuration了，一条是sbt的，一条是disk的。

我们把sbt的那条clear：

RMAN> CONFIGURE DEVICE TYPE sbt clear;

old RMAN configuration parameters:

CONFIGURE DEVICE TYPE 'SBT\_TAPE' BACKUP TYPE TO COMPRESSED BACKUPSET PARALLELISM 1;

RMAN configuration parameters are successfully reset to default value

RMAN> show DEVICE TYPE;

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET; # default

我们再show的时候，发现sbt那条已经“消失”了。

### PARALLELISM

还是上面的设置了parallelism的值大于1，那么在不手动分配通道的情况下，系统会自动分配parallelism个通道。

我们先来进行设置，我们设置俩个并行度：

RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 2;

old RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET;

new RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;

new RMAN configuration parameters are successfully stored

然后我们不手动分配通道，做一个全备：

RMAN> backup database format '/home/oracle/bak/%d\_%U.full';

Starting backup at 12-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=10 device type=DISK

allocated channel: ORA\_DISK\_2

channel ORA\_DISK\_2: SID=625 device type=DISK

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_2: starting full datafile backup set

channel ORA\_DISK\_2: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

channel ORA\_DISK\_2: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_05tcupv3\_1\_1.full tag=TAG20180912T191755 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:25

channel ORA\_DISK\_2: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_06tcupv3\_1\_1.full tag=TAG20180912T191755 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:25

Finished backup at 12-SEP-18

我们看到，通道名是rman自己取的，并且确实开了俩个通道，而且都用到了。

这里注意每个通道有自己的SID，同时每个通道对应一个backup set，而由于我的数据文件少，所以每个backup set只有一个backup piece，backup piece如果想变多可以在后面通过设置channel来变多。

但是，一个backup set的最小单位是一个数据文件。所以如果我平行设置默认为2，那么在备份一个文件时，虽然会开俩个通道，但也只会真正使用一个通道：

RMAN> backup tablespace users format '/home/oracle/bak/%d\_%U.dbf';

Starting backup at 12-SEP-18

using channel ORA\_DISK\_1

using channel ORA\_DISK\_2

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_08tcuqjs\_1\_1.dbf tag=TAG20180912T192859 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 12-SEP-18

我们看，这里确实是使用俩个通道，但是发现数据文件只有一个，就只有1通道对应的server session做了工作。

同时我们还看到，这里并不是像最开始那样分配通道了，而是直接using channel，说明此时channel并没有被释放。我们来一探究竟：

SQL> select spid from v$process p,v$session s where p.addr=s.paddr and sid in (10,625);

SPID

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

50810

50811

[oracle@host01 ~]$ ps -ef|grep 50810

oracle 50810 50510 1 19:17 ? 00:00:22 oraclePROD4 (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))

[oracle@host01 ~]$ ps -ef|grep 50811

oracle 50811 50510 1 19:17 ? 00:00:22 oraclePROD4 (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))

我们看到俩个sever session依然存在。现在我们关闭rman client：

RMAN> exit

[oracle@host01 ~]$ ps -ef|grep 50810

[oracle@host01 ~]$ ps -ef|grep 50811

我们看到，随着rman client的关闭，server session也结束了。

### CHANNEL

#### CHANNEL的基本设置

CONFIGURE CHANNEL和ALLOCATE CHANNEL是一样的，也需要指定device type。configure channel其实就是指定一个样板（template），包括每个channel中backup set中的backup piece的最大值，以及每个channel将backup set输出到哪个路径、backupset的命名规则。

我们下面来做实验。

首先我们配置channel的backup piece的最大值为500M：

RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 500M;

using target database control file instead of recovery catalog

new RMAN configuration parameters:

CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 500 M;

new RMAN configuration parameters are successfully stored

RMAN> show channel;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 500 M;

然后我们把并行开到2：

RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;

old RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK PARALLELISM 1 BACKUP TYPE TO BACKUPSET;

new RMAN configuration parameters:

CONFIGURE DEVICE TYPE DISK PARALLELISM 2 BACKUP TYPE TO BACKUPSET;

new RMAN configuration parameters are successfully stored

然后开始执行全库备份：

RMAN> backup database format '/home/oracle/bak/%d\_%U.full';

Starting backup at 12-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=871 device type=DISK

allocated channel: ORA\_DISK\_2

channel ORA\_DISK\_2: SID=130 device type=DISK

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_2: starting full datafile backup set

channel ORA\_DISK\_2: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

channel ORA\_DISK\_2: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_0dtcuui9\_1\_1.full tag=TAG20180912T203625 comment=NONE

channel ORA\_DISK\_1: starting piece 2 at 12-SEP-18

channel ORA\_DISK\_2: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_0etcuui9\_1\_1.full tag=TAG20180912T203625 comment=NONE

channel ORA\_DISK\_2: starting piece 2 at 12-SEP-18

channel ORA\_DISK\_2: finished piece 2 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_0etcuui9\_2\_1.full tag=TAG20180912T203625 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:10

channel ORA\_DISK\_1: finished piece 2 at 12-SEP-18

piece handle=/home/oracle/bak/PROD4\_0dtcuui9\_2\_1.full tag=TAG20180912T203625 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:14

Finished backup at 12-SEP-18

[oracle@host01 bak]$ ll

total 1311176

-rw-r----- 1 oracle oinstall 524288000 Sep 12 20:36 PROD4\_0dtcuui9\_1\_1.full

-rw-r----- 1 oracle oinstall 224436224 Sep 12 20:36 PROD4\_0dtcuui9\_2\_1.full

-rw-r----- 1 oracle oinstall 524288000 Sep 12 20:36 PROD4\_0etcuui9\_1\_1.full

-rw-r----- 1 oracle oinstall 69632000 Sep 12 20:36 PROD4\_0etcuui9\_2\_1.full

通过与上一小节的实验的对比，我们可以看出，我们的设置生效了，现在因为一个backup piece只能有500M，而导致现在一个backup set中有俩个backup piece。

然后我们再设置备份路径和备份集的命名：

RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/home/oracle/bak/ora\_df%t\_s%s\_s%p';

old RMAN configuration parameters:

CONFIGURE CHANNEL DEVICE TYPE DISK MAXPIECESIZE 500 M;

new RMAN configuration parameters:

CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/home/oracle/bak/ora\_df%t\_s%s\_s%p';

new RMAN configuration parameters are successfully stored

released channel: ORA\_DISK\_1

released channel: ORA\_DISK\_2

我们看到，当我们重新设置channel的时候，原本的channel被release了。

现在我们再来做全备：

RMAN> backup database;

Starting backup at 12-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=871 device type=DISK

allocated channel: ORA\_DISK\_2

channel ORA\_DISK\_2: SID=130 device type=DISK

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_2: starting full datafile backup set

channel ORA\_DISK\_2: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

channel ORA\_DISK\_2: starting piece 1 at 12-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/ora\_df986676681\_s16\_s1 tag=TAG20180912T205121 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:03

channel ORA\_DISK\_2: finished piece 1 at 12-SEP-18

piece handle=/home/oracle/bak/ora\_df986676681\_s17\_s1 tag=TAG20180912T205121 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:03

Finished backup at 12-SEP-18

这次我们并没有使用format指定路径，但是rman还是按照channel的设置将备份集输出到了指定路径。

同时我们也注意到之前backup piece最大值的设置已经失效了。这很正常，相当于重新配置了channel，所以这里就不划重点了。

现在我们来还原channel默认值（其实对于channel就是清除设置）：

RMAN> CONFIGURE CHANNEL DEVICE TYPE DISK clear;

old RMAN configuration parameters:

CONFIGURE CHANNEL DEVICE TYPE DISK FORMAT '/home/oracle/bak/ora\_df%t\_s%s\_s%p';

old RMAN configuration parameters are successfully deleted

released channel: ORA\_DISK\_1

released channel: ORA\_DISK\_2

RMAN> show channel;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

RMAN configuration has no stored or default parameters

我们看到，清除channel的设置也会release channel。

而且还是老规矩，手动的分配通道会无视device type、parallelism、channel的设置。显示指定优先级要高于configuration。

有一个小知识点就是rman对channel的自动命名：ORA\_DISK/SBT\_n。

#### CHANNEL其他的一些设置

RATE：限制I/O流量的，这个做实验的时候可以把很快就完成的一个备份变得耗时很长。

MAXPIECESIZE：这个前面说过了，就是限制每天个通道backup pieces的大小的。

PARMS：指定特定的media management software的信息的。跟SBT有关。

CONNECT：在RAC环境中，用来指定某一个节点的。

#### 配置特定的CHANNEL

我们来看，如下配置了channel 1和channel 2俩个channel的设置：

RMAN> show CHANNEL;

CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '/home/oracle/bak/ora\_df%t\_s%s\_s%p';

CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT '/home/oracle/yx/ora\_df%t\_s%s\_s%p';

现在我们来做个全备，当然我们还是要把并行开到2以上：

RMAN> CONFIGURE DEVICE TYPE DISK PARALLELISM 3 BACKUP TYPE TO BACKUPSET;

然后进行全备：

RMAN> backup database;

Starting backup at 14-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=132 device type=DISK

allocated channel: ORA\_DISK\_2

channel ORA\_DISK\_2: SID=253 device type=DISK

allocated channel: ORA\_DISK\_3

channel ORA\_DISK\_3: SID=875 device type=DISK

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_2: starting full datafile backup set

channel ORA\_DISK\_2: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

channel ORA\_DISK\_2: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 14-SEP-18

piece handle=/home/oracle/bak/ora\_df986826272\_s70\_s1 tag=TAG20180914T142431 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:03

channel ORA\_DISK\_2: finished piece 1 at 14-SEP-18

piece handle=/home/oracle/yx/ora\_df986826272\_s71\_s1 tag=TAG20180914T142431 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:03

Finished backup at 14-SEP-18

我们看到，通道确实给开了3个，第三个通道连server session都分配好了，但是最终只使用了2个，因为文件太少了。

我们配置的channel 1的路径和channel 2的路径确实生效了，俩个备份集分别放在了各自channel指定的路径。

配置specific channel后，该channel在使用时会覆盖掉generic channel的设置，其他的channel依然遵循generic channel设置。

### CONTROLFILE AUTOBACKUP

控制文件自动备份开启时，只要有备份记录增加，就会自动备份控制文件+参数文件。说白了，只要一个备份行为成功完成，那接着就会自动备份控制文件+参数文件。

当数据库处于归档模式时，记录在controlfile中的数据库结构的元数据发生改变时，也会触发控制文件自动备份。

开启关闭控制文件自动备份很简单：

RMAN> show CONTROLFILE AUTOBACKUP;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE CONTROLFILE AUTOBACKUP ON; # default

RMAN> CONFIGURE CONTROLFILE AUTOBACKUP OFF;

new RMAN configuration parameters:

CONFIGURE CONTROLFILE AUTOBACKUP OFF;

new RMAN configuration parameters are successfully stored

RMAN> CONFIGURE CONTROLFILE AUTOBACKUP clear;

old RMAN configuration parameters:

CONFIGURE CONTROLFILE AUTOBACKUP OFF;

RMAN configuration parameters are successfully reset to default value

自动备份的控制文件有一个well-known format，所以rman在使用控制文件的自动备份恢复数据库控制文件时，并不需要访问repository。换句话说，及时控制文件、recovery catalog、spfile都丢了，依然可以使用控制文件的自动备份进行spfile+control file的恢复。

### CONTROLFILE AUTOBACKUP FORMAT

这个是配置控制文件自动备份存放的路径，我们来看下默认值：

RMAN> show CONTROLFILE AUTOBACKUP FORMAT;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

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

这个%F就代表c-IIIIIIIIII-YYYYMMDD-QQ：

1. IIIIIIIIII代表DBID
2. 年月日
3. QQ代表16进制序列号—00到FF。

我们来备份users表空间：

RMAN> backup tablespace users format '/home/oracle/bak/%d\_%U.dbf';

Starting backup at 12-SEP-18

Finished backup at 12-SEP-18

Starting Control File and SPFILE Autobackup at 12-SEP-18

piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/autobackup/2018\_09\_12/o1\_mf\_s\_986686683\_fsldpvyc\_.bkp comment=NONE

Finished Control File and SPFILE Autobackup at 12-SEP-18

我们看到，当我们数据库设置了fast recovery area，如果CONTROLFILE AUTOBACKUP FORMAT是默认设置的话，那自动备份会存放在fast recovery area下，且是OMF的文件管理方式。

SQL> show parameter db\_recovery\_file

NAME TYPE VALUE

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

db\_recovery\_file\_dest string /u01/app/oracle/fast\_recovery\_area/PROD4

db\_recovery\_file\_dest\_size big integer 8016M

我们可以修改configuration使其autobakup到指定的路径，而不是FRA。

RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/home/oracle/yx/%F';

new RMAN configuration parameters:

CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/home/oracle/yx/%F';

new RMAN configuration parameters are successfully stored

RMAN> backup tablespace users;

Starting backup at 13-SEP-18

Finished backup at 13-SEP-18

Starting Control File and SPFILE Autobackup at 13-SEP-18

piece handle=/home/oracle/yx/c-1716001866-20180913-00 comment=NONE

Finished Control File and SPFILE Autobackup at 13-SEP-18

如同其他的configuration一样，我们同样可以显示指定controlfile autobackup的路径和文件命名。

使用SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE deviceSpecifier TO 'string';。这个deviceSpecifier可以是disk或者是sbt。

Controlfile自动备份路径选择的优先级如下：

1. SET CONTROLFILE AUTOBACKUP FORMAT (within a RUN block)
2. SET CONTROLFILE AUTOBACKUP FORMAT (at RMAN prompt)
3. CONFIGURE CONTROLFILE AUTOBACKUP FORMAT

用一个例子来解释这个：

RMAN> SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/home/oracle/bak/cf\_%F';

executing command: SET CONTROLFILE AUTOBACKUP FORMAT

RMAN> backup tablespace users;

Starting backup at 12-SEP-18

Finished backup at 12-SEP-18

Starting Control File and SPFILE Autobackup at 12-SEP-18

piece handle=/home/oracle/bak/cf\_c-1716001866-20180912-07 comment=NONE

Finished Control File and SPFILE Autobackup at 12-SEP-18

RMAN> run{

2> SET CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '/home/oracle/%F';

3> backup tablespace users;

4> }

executing command: SET CONTROLFILE AUTOBACKUP FORMAT

Starting backup at 12-SEP-18

Finished backup at 12-SEP-18

Starting Control File and SPFILE Autobackup at 12-SEP-18

piece handle=/home/oracle/c-1716001866-20180912-08 comment=NONE

Finished Control File and SPFILE Autobackup at 12-SEP-18

从实验，我们看到显式指定定然优先configuration，但是run block中的显示指定优先级是最高的，及时我们在run block之前执行了autobackup的路径，在run block中仍然使用了run block中指定的路径。

### RETENTION POLICY

这个参数就是设置备份保留策略的。我们看下默认值：

RMAN> show RETENTION POLICY;

CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default

默认值的意思就是备份冗余1份，也就是说有同一个备份有新的，那旧的就过期，可以被删除了。

那么什么叫同一个备份：

就是一个数据文件或控制文件的full backup或level 0 backup，那么如果对应的level 0的备份还没obsolete，那所有基于此的level 1的备份也不会被obsolete。

Retention policy可以分俩种：基于冗余的和基于时间窗口的。

#### Redundancy-Based Retention Policy

我们直接来实验默认值。我们把users连续备份俩次，然后看看效果：

RMAN> report obsolete;

RMAN retention policy will be applied to the command

RMAN retention policy is set to redundancy 1

no obsolete backups found

RMAN> backup tablespace users;

Starting backup at 13-SEP-18

piece handle =/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_13/o1\_mf\_nnndf\_TAG20180913T222053\_fsnwl62s\_.bkp tag=TAG20180913T222053 comment=NONE

Finished backup at 13-SEP-18

RMAN> backup tablespace users;

Starting backup at 13-SEP-18

Piece handle =/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_13/o1\_mf\_nnndf\_TAG20180913T222103\_fsnwlhxk\_.bkp tag=TAG20180913T222103 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 13-SEP-18

RMAN> report obsolete;

RMAN retention policy will be applied to the command

RMAN retention policy is set to redundancy 1

Report of obsolete backups and copies

Type Key Completion Time Filename/Handle

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

Backup Set 27 13-SEP-18

Backup Piece 29 13-SEP-18 /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_13/o1\_mf\_nnndf\_TAG20180913T222053\_fsnwl62s\_.bkp

我们可以很明确的看到，在report obsolete的时候，rman会根据retention policy来找出符合obsolete的备份。因为我们默认redundancy是1，所以当有俩份一样的users的备份的时候，早些时候的备份就被视为obsolete。

现在我们把策略改为redundancy 2：

RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;

new RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO REDUNDANCY 2;

new RMAN configuration parameters are successfully stored

RMAN> report obsolete;

RMAN retention policy will be applied to the command

RMAN retention policy is set to redundancy 2

no obsolete backups found

我们看到，冗余变为2后，obsolete的备份就没有了。也就是说备份是否obsolete，是根据当前的备份策略来决定的。Obsolete只是rman对其的一个判断，意思这样的备份就可以删了，该记录还会保存在v$backup\_files中，而并不代表这个备份不能用了（如果这个备份没损坏的话）。只有当执行delete obsolete才会删除这些文件，或者启用了FRA，FRA会在空间不足的时候删除obsolete的备份。

官文有个例子很好的解释了0级备份和1级备份被定义为obsolete的情况：

Assume a different case in which REDUNDANCY is 1. You run a level 0 database backup at noon on Monday, a level 1 cumulative backup at noon on Tuesday and Wednesday, and a level 0 backup at noon on Thursday. Immediately after each daily backup you run the command DELETE OBSOLETE. The Wednesday DELETE command does not remove the Tuesday level 1 backup because this backup is not redundant: the Tuesday level 1 backup could be used to recover the Monday level 0 backup to a time between noon on Tuesday and noon on Wednesday. However, the DELETE command on Thursday removes the previous level 0 and level 1 backups.

#### Window-Based Retention Policy

这个就是用时间窗口来判断备份集的有效性，这个窗口的时间是从备份集生成开始到当前时间。

我们直接来修改参数就好了，这个跟上面的冗余策略是一个道理。

RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;

old RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO REDUNDANCY 2;

new RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;

new RMAN configuration parameters are successfully stored

#### Disabling the Retention Policy

一条命令很简单：

RMAN> CONFIGURE RETENTION POLICY TO NONE;

old RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;

new RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO NONE;

new RMAN configuration parameters are successfully stored

RMAN> report obsolete;

RMAN-00571: ===========================================================

RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============

RMAN-00571: ===========================================================

RMAN-03002: failure of report command at 09/14/2018 00:18:52

RMAN-06525: RMAN retention policy is set to none

但是，如果禁用了retention policy，那么report obsolete和delete obsolete会报错，这都问题不大。

如果启用了FRA，那么FRA将不会自动删除备份文件，因为根本没有obsolete的备份文件。

#### 还原默认值

RMAN> CONFIGURE RETENTION POLICY clear;

old RMAN configuration parameters:

CONFIGURE RETENTION POLICY TO NONE;

RMAN configuration parameters are successfully reset to default value

RMAN> show RETENTION POLICY;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default

### BACKUP OPTIMIZATION

首先我们来看该配置的默认值：

RMAN> show BACKUP OPTIMIZATION;

RMAN configuration parameters for database with db\_unique\_name PROD4 are:

CONFIGURE BACKUP OPTIMIZATION OFF; # default

默认是关闭的。

将该配置开启的效果是，如果已经存在某文件等同的备份的话，就会跳过该文件不备。

那么什么是等同的文件呢？这个要求要比retention policy严格：

1. 对于datafiles的备份：必须是相同的DBID、checkpoint scn、creation scn、resetlogs scn、time（这个time不太明确），同时数据文件必须是offline-normal、 read-only或者closed normally。
2. 对于achivelogs的备份：相同的DBID、thread、sequence number、resetlogs scn和time
3. 对于backupset而言：相同的DBID、backup set record ID、stamp

现在我们将它开启：

RMAN> CONFIGURE BACKUP OPTIMIZATION ON;

new RMAN configuration parameters:

CONFIGURE BACKUP OPTIMIZATION ON;

new RMAN configuration parameters are successfully stored

#### 与retention policy的关系

我们这里就官文上的一个实验，来看下这俩个参数之间的关系。

我们先看下目前的配置：

RMAN> show all;

CONFIGURE RETENTION POLICY TO REDUNDANCY 1; # default

CONFIGURE BACKUP OPTIMIZATION ON;

然后我们把表空间users进行offline：

SQL> alter tablespace users offline;

Tablespace altered.

然后我们开始备份数据库：

RMAN> backup database;

Starting backup at 14-SEP-18

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: finished piece 1 at 14-SEP-18

piece handle= /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_14/o1\_mf\_nnndf\_TAG20180914T111844\_fspb4n8o\_.bkp tag=TAG20180914T111844 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:15

channel ORA\_DISK\_2: finished piece 1 at 14-SEP-18

piece handle= /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_14/o1\_mf\_nnndf\_TAG20180914T111844\_fspb4n9b\_.bkp tag=TAG20180914T111844 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:15

Finished backup at 14-SEP-18

RMAN> backup database;

Starting backup at 14-SEP-18

channel ORA\_DISK\_2: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_2: finished piece 1 at 14-SEP-18

piece handle= /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_14/o1\_mf\_nnndf\_TAG20180914T111913\_fspb5kns\_.bkp tag=TAG20180914T111913 comment=NONE

channel ORA\_DISK\_2: backup set complete, elapsed time: 00:00:03

channel ORA\_DISK\_1: finished piece 1 at 14-SEP-18

piece handle= /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_14/o1\_mf\_nnndf\_TAG20180914T111913\_fspb5kmo\_.bkp tag=TAG20180914T111913 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:07

Finished backup at 14-SEP-18

RMAN> backup database;

Starting backup at 14-SEP-18

using channel ORA\_DISK\_1

using channel ORA\_DISK\_2

skipping datafile 7; already backed up 2 time(s)

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

channel ORA\_DISK\_1: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_2: starting full datafile backup set

channel ORA\_DISK\_2: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

channel ORA\_DISK\_2: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 14-SEP-18

Finished backup at 14-SEP-18

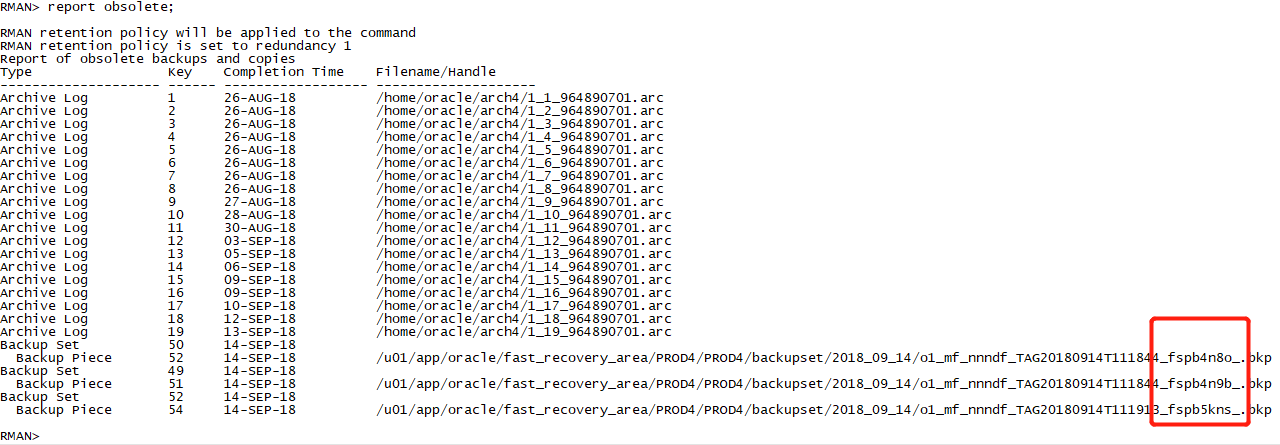
我们这里只留下能说明问题的部分。

我们从第二次全备开始说起，第二次全备仍然备份了users表空间，原因很简单，因为根据策略，第一次全备现在是obsolete的了，那么第二次全备就一定全都备份，不能跳过。

但是当第三次全备的时候，因为我们已经有了第二次的全备，那么第三次全备，可以跳过始终不会变化的已经被offline的users表空间。

所有我们看到了这样的提示：skipping datafile 7; already backed up 2 time(s)

那么我们现在来看下obsolete状态的是第一次全备+第二次全备还是只有第一次全备：



我们看到，第一次的全备是有的，而第二次的全备中，我们看到通道2产生的备份集（sysaux+undo）也被视为obsolete，而通道1产生的备份集由于是作为第三次备份的依赖，所以并不能obsolete。

那我们现在把obsolete的删除，再进行全备：

RMAN> delete obsolete;

RMAN> backup database;

Starting backup at 14-SEP-18

using channel ORA\_DISK\_1

using channel ORA\_DISK\_2

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

Finished backup at 14-SEP-18

我们看到，当没有obsolete时，users表空间又再次被重新备份。

### ARCHIVELOG DELETION POLICY

我们先来看该配置的默认值：

RMAN> show ARCHIVELOG DELETION POLICY;

CONFIGURE ARCHIVELOG DELETION POLICY TO NONE; # default

### MAXSETSIZE

一个备份集的最大值。

RMAN> configure MAXSETSIZE to 20k;

new RMAN configuration parameters:

CONFIGURE MAXSETSIZE TO 20 K;

new RMAN configuration parameters are successfully stored

RMAN> backup database;

Starting backup at 14-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=132 device type=DISK

RMAN-00571: ===========================================================

RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============

RMAN-00571: ===========================================================

RMAN-03002: failure of backup command at 09/14/2018 15:12:53

RMAN-06183: datafile or datafile copy /u01/app/oracle/oradata/PROD4/system01.dbf (file number 1) larger than MAXSETSIZE

这个实验不言而喻了。我们看到由于备份集最大值指定太小，且备份集的最小单位是一个树蕨文件，所以当备份超过该值的文件时，会报错。

也可以在备份的时候显示指定。

RMAN> backup tablespace users maxsetsize 2G;

Starting backup at 14-SEP-18

Finished backup at 14-SEP-18

清除该设置：

RMAN> CONFIGURE MAXSETSIZE clear;

### DATAFILE BACKUP COPIES

就是duplexed backupset。意思就是备份的时候，同一个备份集输出n份。

该设置只影响datafile、controlfile、archivelog的备份，对于控制文件的自动备份和image copy不受该设置影响。

我们看下默认值：

RMAN> show DATAFILE BACKUP COPIES;

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

根据我们之前的实验可以明白，这个设置的意思就是备份的时候只会有一份备份。那么我们现在把这个值该为2：

RMAN> CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 2;

RMAN> backup tablespace users;

Starting backup at 14-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 14-SEP-18

RMAN-00571: ===========================================================

RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============

RMAN-00571: ===========================================================

RMAN-03009: failure of backup command on ORA\_DISK\_1 channel at 09/14/2018 15:38:12

ORA-19806: cannot make duplex backups in recovery area

RMAN> backup tablespace users format '/home/oracle/bak/%d\_%U.dbf';

Starting backup at 14-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 14-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 14-SEP-18 with 2 copies and tag TAG20180914T153845

piece handle=/home/oracle/bak/PROD4\_2dtd3ls5\_1\_1.dbf comment=NONE

piece handle=/home/oracle/bak/PROD4\_2dtd3ls5\_1\_2.dbf comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 14-SEP-18

我们看到oracle不让把copy出来的backup set放FRA中。

我们把该配置清理掉：

RMAN> CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK clear;

### EXCLUDE

## RMAN Backup Concepts

### Consistent and Inconsistent RMAN Backups

Rman可以备份如下5种文件：

1. Datafiles
2. controlfiles
3. spfile
4. archivelogs
5. RMAN backups

所以外部表的文件是不会被备份的。

Rman备份有可以分为一致性备份和非一致性备份。

1. 一致性备份就是把数据库一致性关闭后，启动到mount状态，然后进行备份。
2. 非一致性备份就是数据库open状态时进行备份，或者数据库非一致性关闭后进行的备份。

所以这就是很明显了，为什么rman在线备份数据库一定要求数据库处于归档模式。因为在线备份就是inconsistent backup，recover的时候一定通过日志来使数据库恢复到一致状态，才能open数据库。所以非一致性备份必须开归档！

### Backup Sets

Rman可以将文件备份为俩种格式：backup set和image copy，默认是backup set。

#### Backup Pieces

Backup set一定包含一个或者多个完整的datafiles。

而在制定了section size后，一个backup set又可分为多个binary files，这每一个binary files就是一个backup piece。而backup piece就是不一定会完整的包含一个或多个数据文件了。这样的backup set叫做mutisection backup set。

如果是只要一个backup piece的backup set，restore或者recover的时候，就可以直接用这个backup piece，但是如果是mutisection backup的话，在backup的过程中，可能部分piece失败，那么这样的backupset是不能用的，并且需要手动清理成功了的backup piece。

#### Block Compression for Backup Sets

这种压缩方式是rman默认使用的，也就是说人为无需干涉的。这个压缩，其实严格意义上是对一些块的忽略备份而使备份集变小。

那么哪些块可以不备份呢？

如果数据块中未曾包含过（has never contained）任何数据，那不会被备份，这叫做null block compression。说白了就是没用过的块不备份。

如果数据块被用过了，但是现在没有被用，比如说drop了一个表，那么这个表原来所占有的块，在未被新的对象占有前，就不会被备份。这种叫做unused block compression。那么官文中举例只是举了drop的例子，也就是说高水位下的block应该还是要备份。

能够使用unused block compression的条件如下：

1. COMPATIBLE参数必须是10.2以上，比如你数据库虽然是10.2以上的，但是该参数不是，也不行。
2. 数据库中当前没有任何guaranteed restore point
3. 数据文件是本地管理的（locally managed）
4. 全备或者0级备份
5. 备份到disk上，或者media manager是Oracle Secure Backup

能够使用null block compression的条件就是：全备或者0级备份

#### Binary Compression for Backup Sets

那么binary compression就是真正意义上的压缩了。

俩种方式：

1. backup的时候指定AS COMPRESSED BACKUPSET
2. 配置configuration，CONFIGURE DEVICE TYPE <DISK | SBT> BACKUP TYPE TO COMPRESSED BACKUPSET

俩种压缩结果：

1. 没有配置Oracle Advanced Compression，那就默认是medium
2. 配置了Oracle Advanced Compression，那就可以选择压缩比例：low、medium、high

注意binary compression会消耗额外的cpu。

#### Backup Undo Optimization

如果是已经提交的事务，就不备份相应的undo了，这个也是默认的。人为更改不了，也就是说不能disable。

#### Encryption for Backup Sets

就是给备份加密：

1. SET ENCRYPTION
2. CONFIGURE ENCRYPTION

#### File Names for Backup Pieces

如果备份的时候不用format指定，那么默认在默认路径下使用%U取名。

常用的format有：

1. %d：数据库名
2. %i：DBID
3. %t：timestamp
4. %U：oracle自己取的一个unique标识

#### Number and Size of Backup Pieces

有俩种办法，能让backup set有多个backup piece：

1. CONFIGURE CHANNEL或者ALLOCATE CHANNEL指定maxpiecesize
2. Backup的时候指定section size。

第一种方式当备份到tape上时，每个tape的大小不足以放下整个backupset时，就让每个tape放一个backup piece。

第二种方式是为了备份一个大的数据文件时，可以多通道开并行加快备份速度。因为一般来说一个通道对应一个backupset，但是如果指定section size，则可以进行mutisection backup。

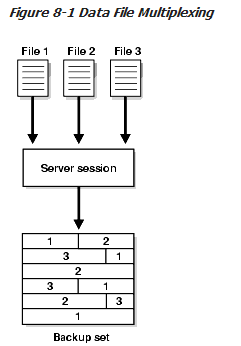
#### Number and Size of Backup Sets

可以通过configure或者backup时指定的方式，配置或指定maxsetsize，从而强制rman产生多个backupset，还可以通过BACKUP ... FILESPERSET来限定每个backupset最多包含多少个数据文件。

但是一定要记住，不管怎么配置，一个backupset至少包含一个数据文件，而且包含的每一个数据文件都是完整的。

#### Multiplexed Backup Sets

官文的一个图解释的非常明确：



就是当把几个数据文件备份到一个backupset的时候，我们看到每个backupset中每个数据文件不是连续的，这就叫multiplexing，那么对于image copies是绝对不可能存在这种情况的。

那么每个备份集一次可以同时读多少个数据文件，取决于filesperset和maxopenfiles的较小值。

专业一点的说法就是filesperset决定了Number of files in each backup set，而maxopenfiles决定了level of multiplexing。

官文举的例子贴一下就不翻译了：

Suppose that you back up 12 data files with one channel when FILEPERSET is set to 4. The level of multiplexing is the lesser of this number and 8. Thus, the channel simultaneously writes blocks from 4 data files into each backup piece.

Now suppose that you back up 50 data files with one channel. The number of files in each backup set is 50. The level of multiplexing is the lesser of this number and 8. Thus, the channel simultaneously writes blocks from 8 data files into each backup piece.

#### Proxy Copies

Media manager专属功能。所以说disk用不了。

### Image Copies

就是一个文件的copy，跟操作系统上copy一个文件没啥区别。

RMAN-Created Image Copies就是执行backup as copy或者配置了defualt device type for disk as image copies，而产生的image copy。

Rman自己生成copy的可以指定format，或者使用db\_file\_name\_convert。

Restore的时候默认还原到文件原来的路径，说白了就是repository中记录的路径，如果就想用image copies现在所在的路径，那直接switch to copy就可以了。

Rman自己生成copy是会检查坏块的。

User-Managed Image Copies就是不用rman产生的copy，比如操作系统中直接cp。

这样的image copies想让rman使用，就要用catalog把copies的metadata注册到repository中。

在注册的时候，rman是不会检查这些copy是否有坏块的。

### Multiple Copies of RMAN Backups

可以使用如下例中方法，用rman生成多份完全相同的copies：

1. backup…copies，可以往一个备份集里放多份相同的copies，每个copy一个backup piece。
2. Backup backupset或者backup copy of，就是把备份集或者image copies再备份一遍

注意，这里说的copies可不是image copies。

第一种方式仅适用于backupset，这样做的主要目的是，可以用一条指令，把想要备份的数据文件同时备份到多个路径，这些路径要么同为disk，要么同为tape。

我们看个实验：

RMAN> BACKUP DEVICE TYPE DISK COPIES 3 DATAFILE 7 FORMAT '/home/oracle/yx/%d\_%U.dbf','/home/oracle/bak/%d\_%U','/home/oracle/%d\_%U';

Starting backup at 20-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 20-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 20-SEP-18 with 3 copies and tag TAG20180920T162734

piece handle=/home/oracle/yx/PROD4\_2vtdjivm\_1\_1.dbf comment=NONE

piece handle=/home/oracle/bak/PROD4\_2vtdjivm\_1\_2 comment=NONE

piece handle=/home/oracle/PROD4\_2vtdjivm\_1\_3 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 20-SEP-18

从这个实验我们可知：

1. 仍然是一个backup set
2. 但是每份copy一个backup piece

那第二种方式就是把备份再备份到另外一个地方（spread backups among multiple media），可以是源备份可以是个backup set也可以是个image copy。

当备份image copies是，rman会选择最新的image copies做为备份源。

### Control File and Server Parameter File Autobackups

Controlfile auto backup启用时，才会自动备份控制文件。

那么什么时候自动备份控制文件呢？

做任何备份的时候都会自动备份controlfile和spfile，数据库有结构改变，比如添加表空间呀等等操作的时候，会触发controlfile的自动备份。

怎么自动备份控制文件：

1. 如果是rman备份的时候，那么所有的数据文件备份结束后，会分配第一个通道把控制文件和spfile单独备份到一个备份集中。
2. 如果是因为数据库结构改变，那么控制文件的自动备份由导致数据库结构改变的server process完成。  
   要是一个脚本中发生多个导致数据库结构改变的指令，那么等脚本执行完，才发起一次控制文件自动备份。

那么控制文件自动备份的文件名必须得有%F，意思要么就别定义备份文件名，要定义就得加上%F，不然rman就不能在控制文件丢失的情况下，找到自动备份出来的备份文件。说白了，就不用使用retore contolfile from autobackup了。

### Incremental Backups

直接进行backup，而不指定increment的话，无论是backup set还是image copies，都是full backup。虽然0级备份和full backup可以说是一样的，但是full backup是不能作为增量备份策略中的一部分的。也就是说不能用full backup当作0级备份。

#### Multilevel Incremental Backups

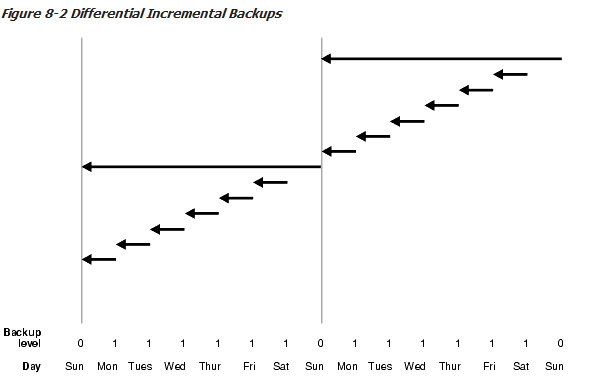
说白了就俩种level，0和1。0级备份就是个全备。

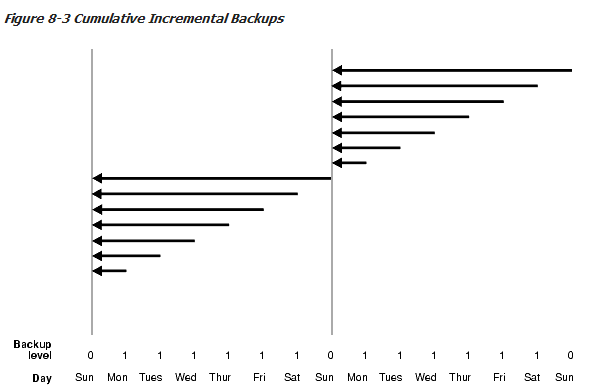
注意backup set和image copies都可以作为0级备份，但是1级备份一定是backup set。

1级备份又分为：

1. Differential Incremental Backups：  
   只备份从最近的一个1级备份到现在变化了的block  
   最近的这个1级备份可以是differential也可以是cumulative  
   如果最近的备份是0级备份，那就备份跟这个0级备份的变化的block。  
   如果最近既没有1级备份又没0级备份，那么就备份从该文件创建以来所有变化的block。
2. Cumulative Incremental Backups：  
   备份从最近一个0级备份到现在变化了的block

如下俩个图诠释的可以说是非常清楚了：





特别注意：显然cumulative的方式要比differential的方式在备份时花费更多的时间并且会占用更多的空间。但是恢复起来会更快，因为在recover的时候，只需要读一个1级备份。

#### Block Change Tracking

用一个很小的二进制文件记录数据库中发生了变化的块，这样当进行1级备份的时候，就不用遍历数据文件的所有块了。所以启用block change tracking就是为了提升1级备份的效率的。

#### \_bct\_bitmaps\_per\_file

##### 参数介绍

SQL> @yx/par

Enter value for par: \_bct\_bitmaps\_per\_file

old 6: AND x.ksppinm LIKE '%&par%'

new 6: AND x.ksppinm LIKE '%\_bct\_bitmaps\_per\_file%'

NAME VALUE DESCRIB

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

\_bct\_bitmaps\_per\_file 8 number of bitmaps to store for each datafile

来翻译一下这个解释：

为每一个数据文件而存放的位图的的数量：

1. 首先这个参数是一个数量
2. 其次这个数量是位图的数量
3. 最后，上述两点限定针对的是每一个数据文件，是each，而不是all。

说白了，就是bct文件中，在默认值的情况下，1个datafile最多有8个位图，那2个datafile，加起来最多自然就是16个位图。

那这个位图是什么呢？我们知道bct中记录的是变更的块，那么是以哪个为基准变更的块呢？显然这就取决于增量备份是cumulative还是differential，那么说白了，我们这里可以简单的推断一个位图对应该datafile的一次backup。

但是隐藏参数的这个解释有时候是不对的，所以我们不能全信，要试了才知道。

##### 官文依据

下面再来贴一个官文的解释：

这是对于bct的介绍：

If block change tracking is enabled on a primary or standby database, then RMAN uses a block change tracking file to identify changed blocks for incremental backups. By reading this small bitmap file to determine which blocks changed, RMAN avoids having to scan every block in the data file that it is backing up.

The first level 0 incremental backup scans the entire data file. Subsequent incremental backups use the block change tracking file to scan only the blocks that have been marked as changed since the last backup.

如果bct被启用了，那么RMAN会使用一个快跟踪文件来确认需要进行增量备份的变化了的block。这个bct文件是个很小的位图文件（后面说它到底有多小）。有了这个文件，RMAN在做增量备份的时候就不用再把数据文件里的每个block扫一次来确定到底哪些是跟上次相比变化了的block。

0级备份是要全扫整个数据文件的，后面的增量备份有了bct文件，就不用再这么费事儿了。

下面是说bct文件的具体工作原理：

Oracle Database automatically manages space in the change tracking file to retain block change data that covers the eight most recent backups. After the maximum of eight bitmaps is reached, the oldest bitmap is overwritten by the bitmap that tracks the current changes.

The block change tracking file maintains bitmaps that mark changes in the data files between backups. The database performs a bitmap switch before each backup.

Oracle数据库自动管理bct文件的空间（不要曲解第一句的意思，它说的是单个文件，而不是说就能维护8个备份集）。当达到8个bitmap的上限时，最老的bitmap会被覆盖（再说白点儿，超过了\_bct\_bitmaps\_per\_file，老的bitmap就会被覆盖）。

bct文件维护的bitmap是什么呢？！是这个数据文件俩次备份之间变化的block。每次备份前，oracle会像切redo一样切到新的bitmap，然后再开始备份。

这是对于bitmap的举例说明：

Consider the eight-bitmap limit when developing your incremental backup strategy. For example, if you make a level 0 database backup followed by seven differential incremental backups, then the block change tracking file now includes eight bitmaps. If you then make a cumulative level 1 incremental backup, then RMAN cannot optimize the backup, because the bitmap corresponding to the parent level 0 backup is overwritten with the bitmap that tracks the current changes.

这个例子就是说，默认是一个文件8个位图，如果我们有一个0级备份和7个差异备份，那么就把8个bitmap都占了。那如果再来一个累计增量备份，怎么办？！我们这里要注意，虽然说0级备份的bitmap还在，但是前面刚说的，备份前要switch，这一次累计备份就要先对这个0级备份的bitmap覆盖，那覆盖了不就没了吗？！对！所以这次累计备份就用不到bct文件。

我们就跟着这个延伸思考一下，那么再来一次累计备份呢，会用到bct文件么？显然不会，原因就是0级备份的bitmap没了，你后面再做多少次累计备份都用不到bct文件了。

那么要是我这时候做一次差异备份呢？显然又能用到bct文件了，因为差异备份只要求上一个备份的bitmap在就行。

官文里有一个信息不要漏掉！0级备份也有自己的bitmap！

官文的也有写错的时候，所以我们还是要实验。

所以我从小到大会提醒自己的其中一件事就是书本和先辈只是指路人，它们说的对不对要先思考再测试，如果我确实没条件证明它们错，那我才迫不得已选择信。

很简单一个道理，大家都是人，文明是人建立的，凭什么他们不会错。

不过随着不断的成长，我现在会再附带提醒自己一句：不要杠！

##### 环境准备

12c中，bct哪儿开，显然是cdb开，pdb虽然怎么看怎么像个单独的数据库，但是我一定清楚，真正的database是CDB，pdb只是下面一个组件罢了。

pdb下开就是这个效果：

SQL> alter database enable block change tracking using file '/home/oracle/yx/aa.trf';

alter database enable block change tracking using file '/home/oracle/yx/aa.trf'

\*

ERROR at line 1:

ORA-65040: operation not allowed from within a pluggable database

还有就是，我能确定的是11gR2以后开这个open状态直接开就行了，官文里当然也说了的（mount或这open状态）。

SQL> alter database enable block change tracking using file '/home/oracle/yx/test\_bct.trf';

Database altered.

SQL> COL STATUS FORMAT A8

SQL> COL FILENAME FORMAT A60

SQL> SELECT STATUS, FILENAME FROM V$BLOCK\_CHANGE\_TRACKING;

STATUS FILENAME

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

ENABLED /home/oracle/yx/test\_bct.trf

开了就能用。

那我们接着别跑题，我们就是要看看这个\_bct\_bitmaps\_per\_file参数的真实效果。

如果如官文所说，那么我们这里把它改成2更好实验。

SQL> alter system set "\_bct\_bitmaps\_per\_file"=2;

System altered.

##### 实验

###### 0级备份

我们先做一个0级备份。在一个pdb下备份pdb的users表空间：

[oracle@host01 ~]$ rman target sys/oracle@pdbprod1

RMAN> backup incremental level 0 tablespace users;

Starting backup at 25-APR-19

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting incremental level 0 datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00012 name=/u01/app/oracle/oradata/PRODCDB/PDBPROD1/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 25-APR-19

channel ORA\_DISK\_1: finished piece 1 at 25-APR-19

piece handle=/home/oracle/flash/PRODCDB/622F3C480F3F3E2FE0530A0CA8C0ED75/backupset/2019\_04\_25/o1\_mf\_nnnd0\_TAG20190425T094508\_gd2494d6\_.bkp tag=TAG20190425T094508 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 25-APR-19

后面就不会这么全部贴了。

我们来看视图：

SQL> select file#,to\_char(COMPLETION\_TIME,'hh24:mi:ss') time,USED\_CHANGE\_TRACKING,INCREMENTAL\_LEVEL,con\_id from v$backup\_datafile where COMPLETION\_TIME>trunc(sysdate);

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

可以看到，0级备份用到了bct，那也就是说它确实占一个bitmap。

###### 做2次differential备份，做1次cumulative备份

我们做俩次1级差异备份，让12号文件的bitmap超过2个：

RMAN> backup incremental level 1 tablespace users;

…

tag=TAG20190425T094751

…

RMAN> backup incremental level 1 tablespace users;

tag=TAG20190425T094910

我们来查询：

SQL> select file#,to\_char(COMPLETION\_TIME,'hh24:mi:ss') time,USED\_CHANGE\_TRACKING,INCREMENTAL\_LEVEL,con\_id from v$backup\_datafile where COMPLETION\_TIME>trunc(sysdate);

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

12 09:47:52 YES 1 3

12 09:49:10 YES 1 3

我们看到，都用到bct了的，其实这里已经佐证了只要\_bct\_bitmaps\_per\_file大于等于2，那么差异增量备份就一定能用的上bct。

然后我们做一次累计增量备份：

RMAN> backup incremental level 1 CUMULATIVE tablespace users;

tag=TAG20190425T095134

按前面官文所说，这次增量备份是用不到bct的，其实上一次增量备份如果是cumulative的，就已经用不到bct了。我们来看看是不是：

SQL> select file#,to\_char(COMPLETION\_TIME,'hh24:mi:ss') time,USED\_CHANGE\_TRACKING,INCREMENTAL\_LEVEL,con\_id from v$backup\_datafile where COMPLETION\_TIME>trunc(sysdate);

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

12 09:47:52 YES 1 3

12 09:49:10 YES 1 3

12 09:51:34 NO 1 3

###### 再做几次cumulative备份

我们再做俩次，看看是不是，cumulative方式的增量备份再也用不到bct了（因为0级的bitmap被覆盖了），我们直接看结果：

SQL> select file#,to\_char(COMPLETION\_TIME,'hh24:mi:ss') time,USED\_CHANGE\_TRACKING,INCREMENTAL\_LEVEL,con\_id from v$backup\_datafile where COMPLETION\_TIME>trunc(sysdate);

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

12 09:47:52 YES 1 3

12 09:49:10 YES 1 3

12 09:51:34 NO 1 3

12 09:52:45 NO 1 3

12 09:53:09 NO 1 3

6 rows selected.

确实如此。

###### 换个表空间做一次0级备份，并做1次cumulative

RMAN> backup incremental level 0 tablespace sysaux;

tag=TAG20190425T095400

RMAN> backup incremental level 1 CUMULATIVE tablespace sysaux;

tag=TAG20190425T095429

这一次，因为\_bct\_bitmaps\_per\_file是2，所以0级bitmap在，那么第一次cumulative一定是可以用的到bct的：

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

12 09:47:52 YES 1 3

12 09:49:10 YES 1 3

12 09:51:34 NO 1 3

12 09:52:45 NO 1 3

12 09:53:09 NO 1 3

10 09:54:02 YES 0 3

10 09:54:29 YES 1 3

确实如此。

###### 再对sysaux做几次cumulative

接着我有做了俩次sysaux的cumulative备份，我们想下，到底能不能用到bct呢？

这完全取决于oracle会覆盖哪一条bitmap，如果覆盖上一条cumulative的bitmap，那么就能接着用到bct，如果覆盖了0级备份的bitmap，那么就用不到了。

显然Oracle不会这么蠢：

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:45:08 YES 0 3

12 09:47:52 YES 1 3

12 09:49:10 YES 1 3

12 09:51:34 NO 1 3

12 09:52:45 NO 1 3

12 09:53:09 NO 1 3

10 09:54:02 YES 0 3

10 09:54:29 YES 1 3

10 09:54:52 YES 1 3

10 09:55:12 YES 1 3

###### 再对users做一次differential备份

这里我们想一下，如果\_bct\_bitmaps\_per\_file是一个bct文件中所有的bitmap数，那么此时bct文件中应当已经没有users的数据文件的bitmap了。那么这一次差异备份就不会用到bct。

如果如我们的理解以及官文的叙述以及参数的解释，那么本次差异备份就可以用到bct：

RMAN> backup incremental level 1 tablespace users;

tag=TAG20190425T095554

FILE# TIME USE INCREMENTAL\_LEVEL CON\_ID

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

12 09:55:54 YES 1 3

###### 总结

\_bct\_bitmaps\_per\_file只要大于等于2，对于单纯的differential备份或单纯的cumulative备份都没影响，但是对于混合的---就一个0级备份，一会儿differential一会儿cumulative，那么对cumulative可能会有影响（说可能，是因为如果0级备份的bitmap不被覆盖，那么也不会有影响）

我们顺带可以看下v$backup\_datafile别的列，这个视图还是挺有用的：

SQL> desc v$backup\_datafile;

Name Null? Type

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

RECID NUMBER

STAMP NUMBER

SET\_STAMP NUMBER

SET\_COUNT NUMBER

FILE# NUMBER

CREATION\_CHANGE# NUMBER

CREATION\_TIME DATE

RESETLOGS\_CHANGE# NUMBER

RESETLOGS\_TIME DATE

INCREMENTAL\_LEVEL NUMBER

INCREMENTAL\_CHANGE# NUMBER

CHECKPOINT\_CHANGE# NUMBER

CHECKPOINT\_TIME DATE

ABSOLUTE\_FUZZY\_CHANGE# NUMBER

MARKED\_CORRUPT NUMBER

MEDIA\_CORRUPT NUMBER

LOGICALLY\_CORRUPT NUMBER

DATAFILE\_BLOCKS NUMBER

BLOCKS NUMBER

BLOCK\_SIZE NUMBER

OLDEST\_OFFLINE\_RANGE NUMBER

COMPLETION\_TIME DATE

CONTROLFILE\_TYPE VARCHAR2(1)

USED\_CHANGE\_TRACKING VARCHAR2(3)

BLOCKS\_READ NUMBER

USED\_OPTIMIZATION VARCHAR2(3)

FOREIGN\_DBID NUMBER

PLUGGED\_READONLY VARCHAR2(3)

PLUGIN\_CHANGE# NUMBER

PLUGIN\_RESETLOGS\_CHANGE# NUMBER

PLUGIN\_RESETLOGS\_TIME DATE

SECTION\_SIZE NUMBER

UNDO\_OPTIMIZED VARCHAR2(3)

BLOCKS\_SKIPPED\_IN\_CELL NUMBER

CON\_ID NUMBER

BACKED\_BY\_PDB VARCHAR2(3)

SPARSE\_BACKUP VARCHAR2(3)

GUID RAW(16)

SQL> select file#,to\_char(COMPLETION\_TIME,'hh24:mi:ss') time,USED\_CHANGE\_TRACKING,INCREMENTAL\_LEVEL,BLOCKS,CHECKPOINT\_CHANGE#,BACKED\_BY\_PDB from v$backup\_datafile where COMPLETION\_TIME>trunc(sysdate);

FILE# TIME USE INCREMENTAL\_LEVEL BLOCKS CHECKPOINT\_CHANGE# BAC

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

12 09:45:08 YES 0 741 12624004 YES

12 09:47:52 YES 1 11 12624446 YES

12 09:49:10 YES 1 2 12624655 YES

12 09:51:34 NO 1 11 12625086 YES

12 09:52:45 NO 1 11 12625286 YES

12 09:53:09 NO 1 11 12625357 YES

10 09:54:02 YES 0 53306 12625501 YES

10 09:54:29 YES 1 1 12625581 YES

10 09:54:52 YES 1 1 12625680 YES

10 09:55:12 YES 1 1 12625744 YES

##### bct文件的大小

The size of the block change tracking file is proportional to the size of the database and the number of enabled threads of redo.

The size of the block change tracking file can increase and decrease as the database changes. The size is not related to the frequency of updates to the database. Typically, the space required for block change tracking for a single instance is approximately 1/30,000 the size of the data blocks to be tracked. For an Oracle RAC environment, it is 1/30,000 of the size of the database, times the number of enabled threads.

The following factors that may cause the file to be larger than this estimate suggests:

1. To avoid the overhead of allocating space as your database grows, the block change tracking file size starts at 10 megabytes. New space is allocated in 10 MB increments. Thus, for any database up to approximately 300 gigabytes, the file size is no smaller than 10 MB, for up to approximately 600 gigabytes the file size is no smaller than 20 megabytes, and so on.
2. For each data file, a minimum of 320 kilobytes of space is allocated in the block change tracking file, regardless of the size of the data file. Thus, if you have a large number of relatively small data files, the change tracking file is larger than for databases with a smaller number of larger data files containing the same data.

这个总结一下：

1. 大小跟数据库大小以及enabled redo threads数（也可以理解为rac的节点数）成比例
2. 跟数据库发生的更新频次无关！！！
3. 大小的估算公式是：size of database \* 1/30000 \* number of enabled threads
4. 300g以下的库bct文件不会小于10m，600g以下的库bct文件不会小于20m，依此类推
5. bct文件中，为每个datafile至少分配320k的空间。
6. 由第5条得出，就算是数据完全一样的数据库，数据文件多的database的bct文件要更大。

#### Incremental Backup Algorithm

有3个概念：Checkpoint SCN、Incremental start SCN、Block SCN

数据文件的checkpoint scn可以在v$datafile中的CHECKPOINT\_CHANGE#查到。那么增量备份的算法原理是这样的：0级备份把checkpoint scn之前的block都备了，下个1级备份的时候从0级备份的那个checkpoint scn开始，block scn大于或等于这个checkpoint scn的block都备份一遍，所以0级备份的checkpoint scn就是该1级备份的incremental start scn。那下一个1级备份是同理，differential的话，就以上个1级备份时的checkpoint scn为incremental start scn，如果是cumulative，那还是以0级备份的checkpoint为incremental start scn。

当block change tracking启用时，做1级备份的时候就可以跟bitmap跳过读取那些没有发生改变的block。注意，只是节省了读取的时间，并不代表block change tracking不启用，1级备份就要备份所有读了块。

#### Recovery with Incremental Backups

恢复的时候，使用1级备份比应用归档有俩个好处：

1. 将修改直接应用的block上要比应用redo快的多
2. 对于在0级备份之后做的nologging操作，只能用1级备份进行恢复

### Backup Retention Policies

#### Recovery Window

#### Backup Redundancy

#### Batch Deletes of Obsolete Backups

#### Backup Retention Policy and Fast Recovery Area Deletion Rules

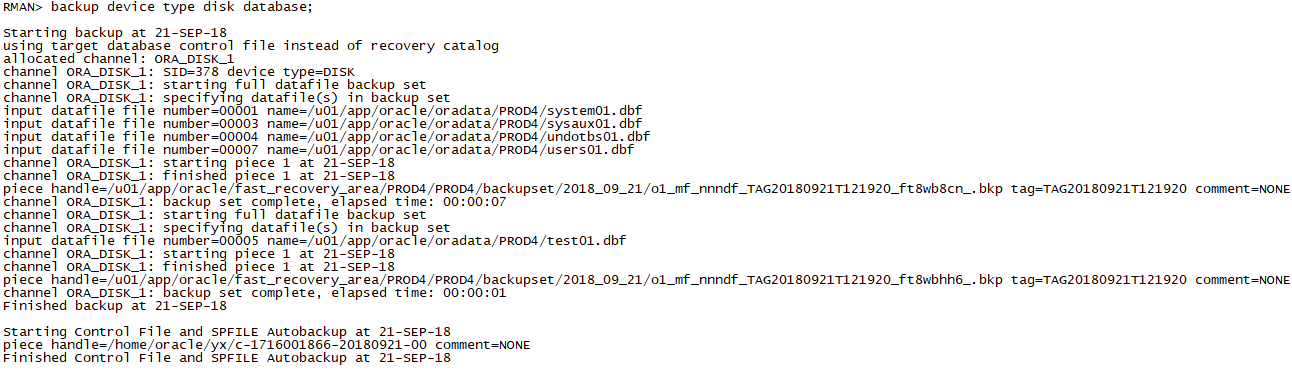
## Backing Up the Database

### Specifying Backup Output Options

#### Specifying the Device Type for an RMAN Backup

Backup的时候指定备份存放的设备：

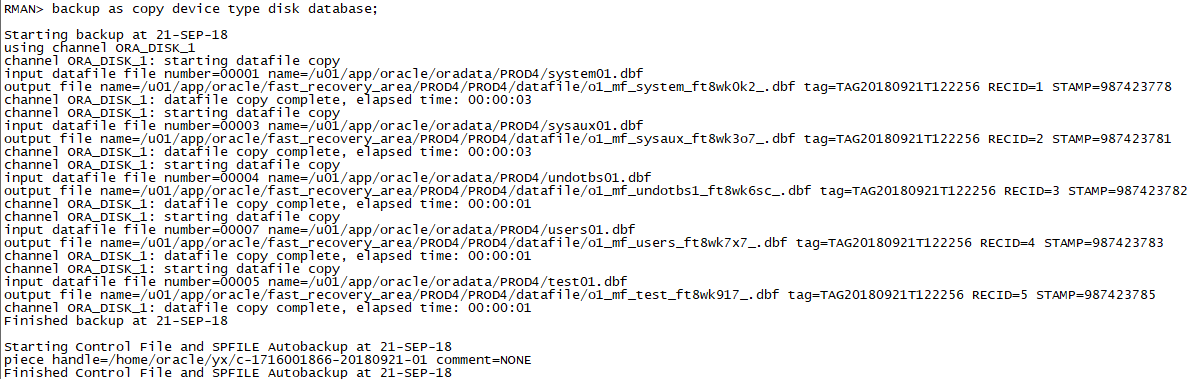
RMAN> backup device type disk database;



#### Specifying Backup Set or Copy for an RMAN Backup to Disk

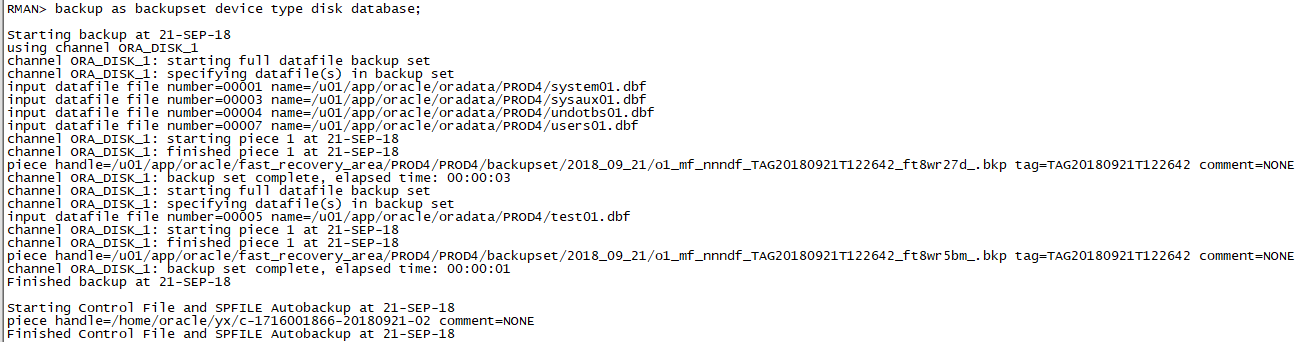
指定备份方式为image copy，该方式的device type只能是disk：

RMAN> backup as copy device type disk database;



指定备份方式为backup set（如果不在configuration中配置，默认就是该方式）：

RMAN> backup as backupset device type disk database;



#### Specifying a Format for RMAN Backups

就是指定备份路径，默认是FRA。我们看官文的一个示例：

BACKUP DATABASE

FORMAT '+dgroup1'; # specifies an ASM disk group

因为asm会自动为文件生成unique file name，所以可以不用在format中加入%U。

##### Specifying Multiple Formats for Disk Backups

也可以在channel中配置format，configure的方式前面已经说过了，我们实验手动分配通道的方式：

RUN

{

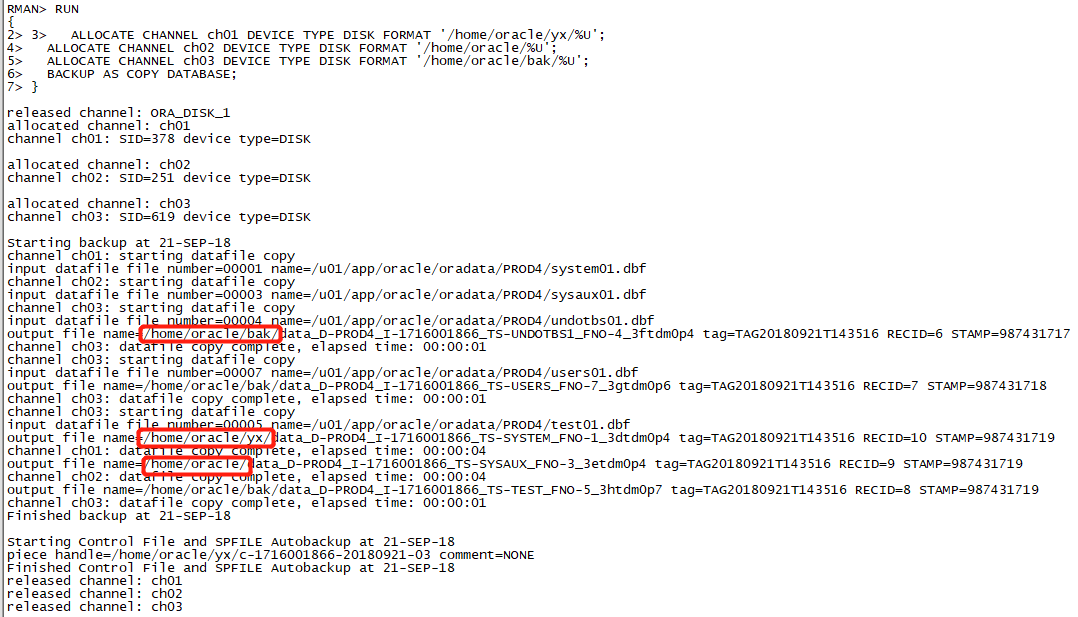
ALLOCATE CHANNEL ch01 DEVICE TYPE DISK FORMAT '/home/oracle/yx/%U';

ALLOCATE CHANNEL ch02 DEVICE TYPE DISK FORMAT '/home/oracle/%U';

ALLOCATE CHANNEL ch03 DEVICE TYPE DISK FORMAT '/home/oracle/bak/%U';

BACKUP AS COPY DATABASE;

}



#### Specifying Tags for an RMAN Backup

Tag就是个标签，如果不显式指定，那么除了控制文件自动备份，rman都会加一个默认的tag：TAGYYYYMMDDTHHMMSS，这个在前面的各种例子也是看到了的，这个时间就是backup开始的时间。

注意tag的长度是最多30bytes，命名规则就是不要使用%D、%T这些的，如果备份集是放在asm中，命名中不能带有-（hyphen），rman会把tag存成大写，但使用时不区分大小写。

如果在备份backup或者restore时，可以只指定某个tag，oracle会在所有符合条件的同名的tag中找到最新的backupset或者image copy进行使用。

RMAN> backup tablespace users tag uses\_tablespace;

Starting backup at 21-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 21-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 21-SEP-18

piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_21/o1\_mf\_nnndf\_USES\_TABLESPACE\_ft96syqq\_.bkp tag=USES\_TABLESPACE comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:01

Finished backup at 21-SEP-18

然后我们备份backup：

RMAN> backup backupset from tag uses\_tablespace tag users\_tablespace\_copy;

Starting backup at 21-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: input backup set: count=115, stamp=987434302, piece=1

channel ORA\_DISK\_1: starting piece 1 at 21-SEP-18

channel ORA\_DISK\_1: backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_21/o1\_mf\_nnndf\_USES\_TABLESPACE\_ft96syqq\_.bkp

piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_21/o1\_mf\_nnndf\_USES\_TABLESPACE\_ft971v76\_.bkp comment=NONE

channel ORA\_DISK\_1: finished piece 1 at 21-SEP-18

channel ORA\_DISK\_1: backup piece complete, elapsed time: 00:00:01

Finished backup at 21-SEP-18

我们通过tag uses\_tablespace找到备份集，并且备份这个备份集，并且给新的备份集取了个tag叫做users\_tablespace\_copy。我们list backupset可以一目了然。

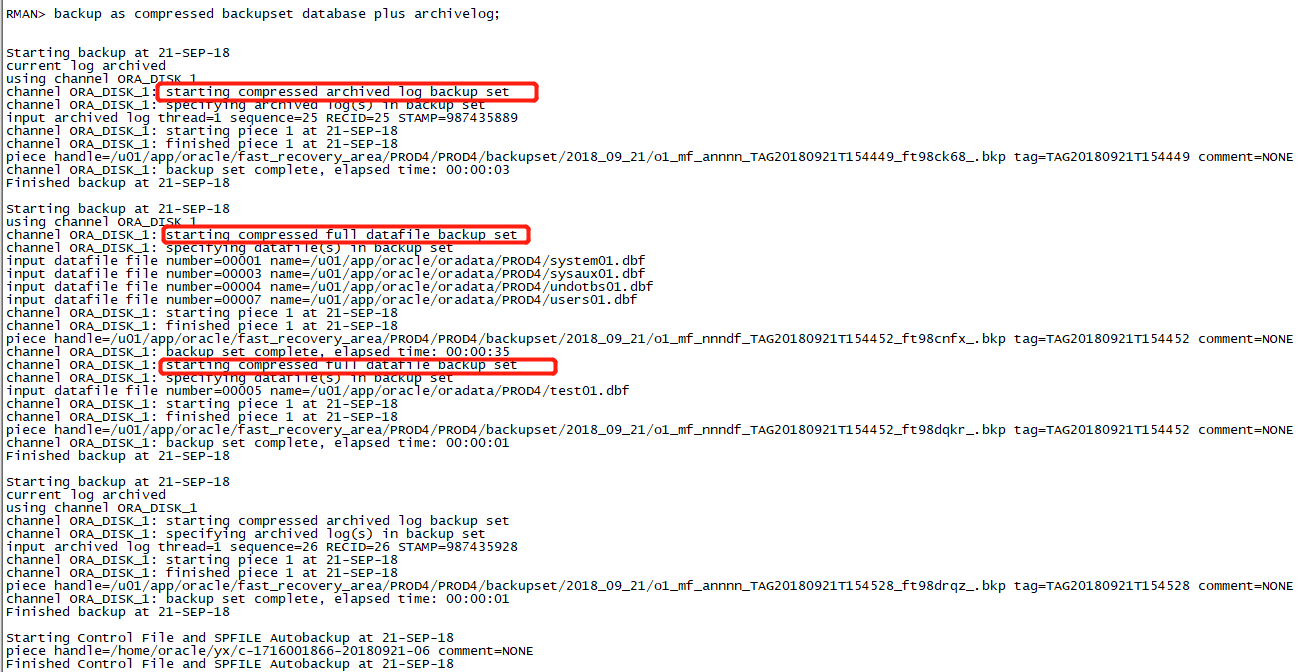


#### Making Compressed Backups

就是在把备份集写到磁盘前，rman会对其进行压缩。到底用的啥级别的（low medium high）压缩等级，都会自动记录在备份集中。压缩备份需要消耗额外的cpu，cpu忙的时候不要使用。

同时Restore的时候不需要显式解压。

RMAN> backup as compressed backupset database plus archivelog;



#### Specifying Multisection Incremental Backups

想对1级备份进行multisection backup，compatible参数必须得12.0.0以上。

如果指定的section size大于文件大小，那么rman就不用使用mutisection backup，如果指定的section size太小，导致backup pieces超过256个，那么rman会自动调整sectionsize，最多生成256个backup piece。

RMAN在进行mutilsection backup的时候，filesperset一定是1，也就是说一个文件一个backupset，每个backupset多个backup piece。

我们来看实验：

RUN

{

ALLOCATE CHANNEL ch01 device type disk;

ALLOCATE CHANNEL ch02 device type disk;

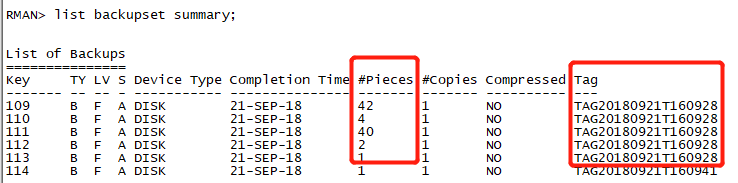
ALLOCATE CHANNEL ch03 device type disk;

BACKUP DATABASE section size 20m;

}

我们因为备份片太多了，我们使用list backupset summary:

RMAN> list backupset summary;



我们看，数据库中一共5个数据文件，所以一共产生了5个备份集，最后那个tag是controlfile自动备份的。同时可以看到最大的一个数据文件，被分了42个pieces，也有只有一个pieces的，说明该数据文件还不足20m。

我们再做个1级备份，同样道理，我们就不做0级备份了，主要为了观察multisection。

我们先看下compatible参数：

SQL> show parameter compatible

NAME TYPE VALUE

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

compatible string 12.2.0

然后我们来进行备份：

RUN

{

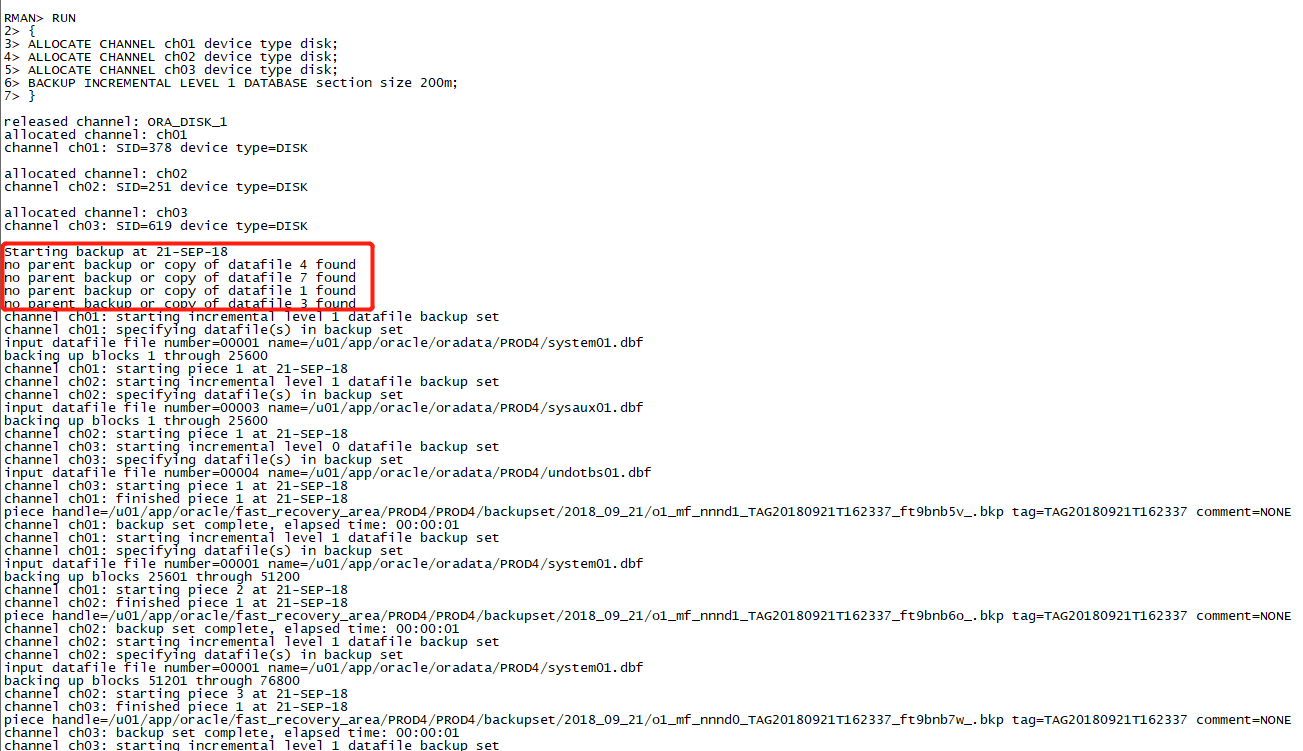
ALLOCATE CHANNEL ch01 device type disk;

ALLOCATE CHANNEL ch02 device type disk;

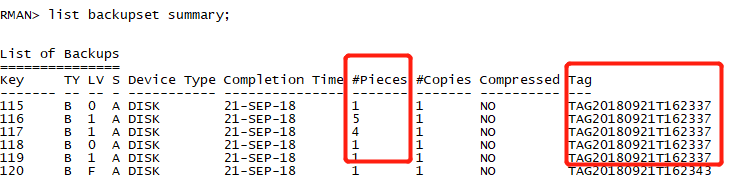
ALLOCATE CHANNEL ch03 device type disk;

BACKUP INCREMENTAL LEVEL 1 DATABASE section size 200m;

}

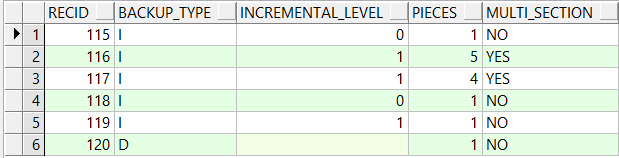


我们再list backupset summary：



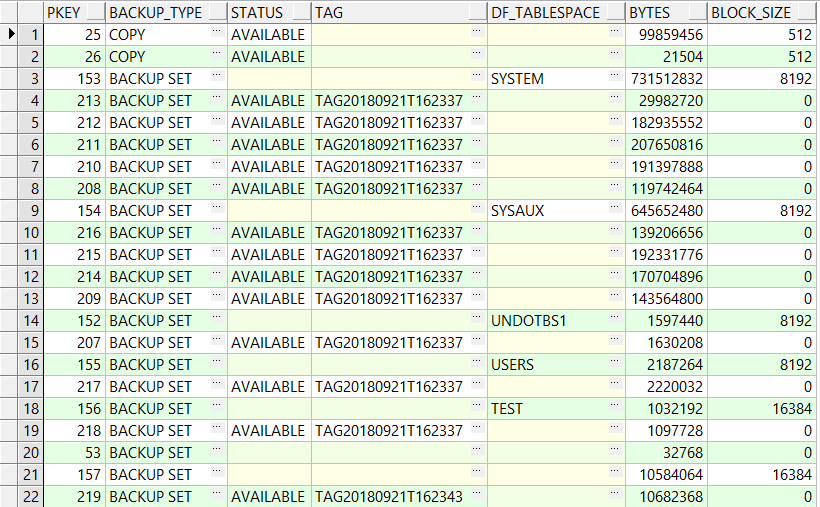
V$BACKUP\_SET的multi\_section列为yes，就说明该备份集是mutisection的。

select recid,backup\_type,incremental\_level,pieces,multi\_section from v$backup\_set;



V$BACKUP\_DATAFILE的bytes列显示每个section的大小。

select pkey,backup\_type,status,tag,df\_tablespace,bytes,block\_size from v$backup\_files;



#### Making Multisection Backups Using Image Copies

跟对1级备份multisection一样，也是12c的新特性，所以也要求compatible为12.0.0以上。

RUN

{

ALLOCATE CHANNEL ch01 device type disk;

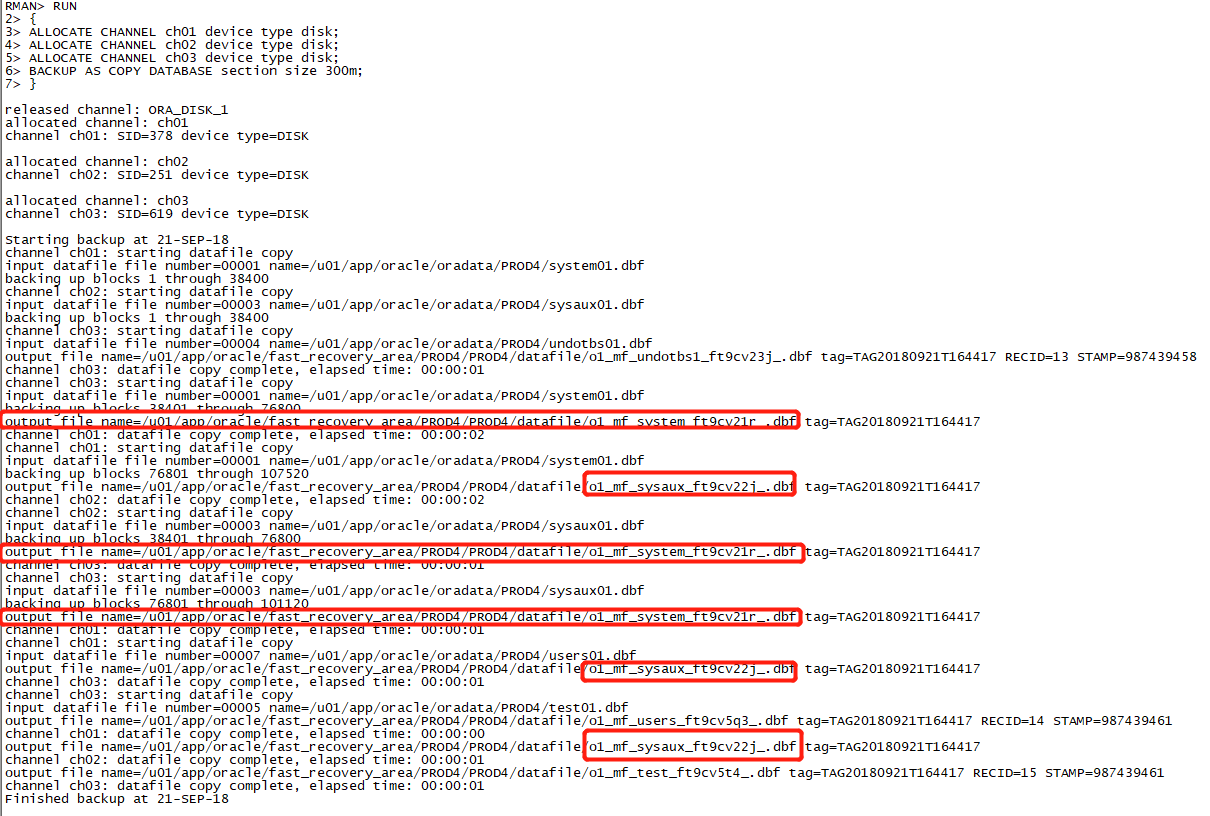
ALLOCATE CHANNEL ch02 device type disk;

ALLOCATE CHANNEL ch03 device type disk;

BACKUP AS COPY DATABASE section size 300m;

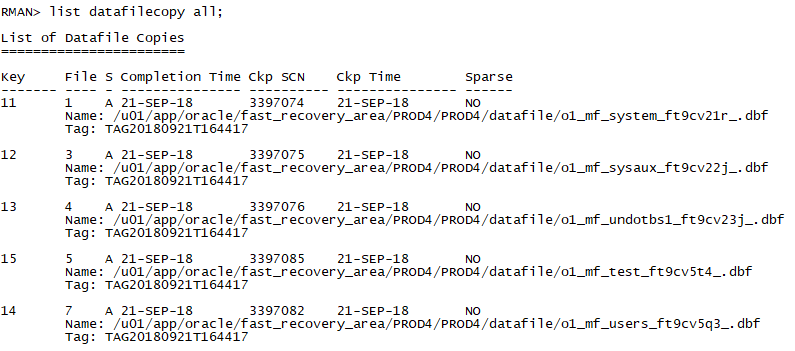
}

我们来看这个过程：



可以看到，虽然也是并行输出，但是相同的数据文件，最终还是输出到了同一个文件中，我们通过list datafilecopy all也可以看到这一点：

RMAN> list datafilecopy all;



### Backing Up Database Files with RMAN

#### Backing Up a Whole Database with RMAN

就是backup database，如果连带归档一起就是：

RMAN> BACKUP DATABASE PLUS ARCHIVELOG;

也可以通过configure exclude for tablespace xx，在备份的时候永久跳过某表空间。

#### Backing Up Tablespaces and Data Files with RMAN

Backup datafie <fileid list>，如果是backup tablespace，rman会自动将指令翻译为backup该表空间下的所有datafile。

RMAN> backup datafile 1,5,7;

#### Backing Up Control Files with RMAN

可以backup current controlfile或者include controlfile或者backup as copy current controlfile：

RMAN> backup datafile 7 include current controlfile;

RMAN> backup as copy current controlfile;

如果是backup as copy，那么不需要保证一致性的snapshot，但是该copy必须放在共享存储上。

如果不是copy，那么需要有一致性的snapshot，并且snapshot必须放在共享存储上，关于snapshot的介绍，在前面的configuration中有。

#### Backing Up Server Parameter Files with RMAN

RMAN> backup spfile;

#### Backing Up a Database in NOARCHIVELOG Mode

必须保证库是一致性时才能备份，比如shutdown immediate然后启动到mount阶段。Open阶段不可以进行备份。

#### Backing Up CDBs and PDBs

##### Backing Up a Whole CDB

登到cdb下，进行backup database：

[oracle@host01 ~]$ rman target /

Recovery Manager: Release 12.2.0.1.0 - Production on Sun Sep 23 13:00:52 2018

Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.

connected to target database: PRODCDB (DBID=2962887328)

RMAN> backup database;

该指令会把所有的数据文件，包括pdb的全都备份。

##### Backing Up the Root with RMAN

可以不备份pdbs的数据文件，那就只备份root：

RMAN> BACKUP DATABASE ROOT;

Starting backup at 23-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting full datafile backup set

channel ORA\_DISK\_1: specifying datafile(s) in backup set

input datafile file number=00003 name=/u01/app/oracle/oradata/PRODCDB/sysaux01.dbf

input datafile file number=00001 name=/u01/app/oracle/oradata/PRODCDB/system01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PRODCDB/undotbs01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PRODCDB/users01.dbf

channel ORA\_DISK\_1: starting piece 1 at 23-SEP-18

channel ORA\_DISK\_1: finished piece 1 at 23-SEP-18

piece handle=/home/oracle/flash/PRODCDB/backupset/2018\_09\_23/o1\_mf\_nnndf\_TAG20180923T130305\_ftg7n9x7\_.bkp tag=TAG20180923T130305 comment=NONE

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:07

Finished backup at 23-SEP-18

##### Backing Up PDBs with RMAN

可以使用backup pluggable database：

RMAN> BACKUP PLUGGABLE DATABASE PDBPROD1,PDBPROD2;

或者直接target到对应的pdb上，进行backup database：

[oracle@host01 ~]$ rman target sys/oracle@pdbprod1

Recovery Manager: Release 12.2.0.1.0 - Production on Sun Sep 23 14:19:20 2018

Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.

connected to target database: PRODCDB:PDBPROD1 (DBID=2212847075)

RMAN> backup database;

如果在cdb下是backup pdb的某一个表空间：

RMAN> backup tablespace pdbprod1:users;

当然，也可以直接到pdb下备份：

RMAN> backup tablespace users;

#### Backing Up Archived Redo Logs with RMAN

有如下几种备份归档的方式：

1. PLUS ARCHIVELOG
2. ARCHIVELOG ALL
3. ARCHIVELOG FROM ...

前面的俩种方式会自动进行archive log current，即一定会发生logfile switch。

RMAN> backup datafile 7 plus archivelog;

RMAN> backup archivelog all;

Starting backup at 24-SEP-18

current log archived

当我们使用from sequence a until sequence b的时候，备份的时候是包含a和b的：

RMAN> backup archivelog from sequence 29 until sequence 31;

Starting backup at 24-SEP-18

using target database control file instead of recovery catalog

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=8 device type=DISK

channel ORA\_DISK\_1: starting archived log backup set

channel ORA\_DISK\_1: specifying archived log(s) in backup set

input archived log thread=1 sequence=29 RECID=29 STAMP=987692229

input archived log thread=1 sequence=30 RECID=30 STAMP=987692230

input archived log thread=1 sequence=31 RECID=31 STAMP=987692233

##### Backing Up Only Archived Redo Logs That Need Backups

如果不想再次备份已经备份过的归档，可以配置backup optimization为on：

RMAN> backup archivelog all;

input archived log thread=1 sequence=29 RECID=29 STAMP=987692229

input archived log thread=1 sequence=30 RECID=30 STAMP=987692230

input archived log thread=1 sequence=31 RECID=31 STAMP=987692233

input archived log thread=1 sequence=32 RECID=32 STAMP=987693291

input archived log thread=1 sequence=33 RECID=33 STAMP=987693294

input archived log thread=1 sequence=34 RECID=34 STAMP=987693445

RMAN> CONFIGURE BACKUP OPTIMIZATION ON;

RMAN> backup archivelog all;

skipping archived logs of thread 1 from sequence 29 to 34; already backed up

input archived log thread=1 sequence=35 RECID=35 STAMP=987693702

RMAN> delete backupset;

RMAN> backup archivelog all;

input archived log thread=1 sequence=29 RECID=29 STAMP=987692229

input archived log thread=1 sequence=30 RECID=30 STAMP=987692230

input archived log thread=1 sequence=31 RECID=31 STAMP=987692233

input archived log thread=1 sequence=32 RECID=32 STAMP=987693291

input archived log thread=1 sequence=33 RECID=33 STAMP=987693294

input archived log thread=1 sequence=34 RECID=34 STAMP=987693445

input archived log thread=1 sequence=35 RECID=35 STAMP=987693702

input archived log thread=1 sequence=36 RECID=36 STAMP=987693847

还以使用：BACKUP ... NOT BACKED UP integer TIMES

RMAN> BACKUP ARCHIVELOG ALL NOT BACKED UP 2 TIMES;

如果想要删除已经备份了的archivelog，那就加上delete input。

RMAN> backup archivelog all delete input;

Starting backup at 24-SEP-18

current log archived

skipping archived logs of thread 1 from sequence 29 to 36; already backed up

channel ORA\_DISK\_1: starting archived log backup set

channel ORA\_DISK\_1: specifying archived log(s) in backup set

input archived log thread=1 sequence=37 RECID=37 STAMP=987693944

channel ORA\_DISK\_1: backup set complete, elapsed time: 00:00:02

channel ORA\_DISK\_1: deleting archived log(s)

archived log file name=/home/oracle/arch4/1\_37\_964890701.arc RECID=37 STAMP=987693944

channel ORA\_DISK\_1: deleting archived log(s)

archived log file name=/home/oracle/arch4/1\_29\_964890701.arc RECID=29 STAMP=987692229

archived log file name=/home/oracle/arch4/1\_30\_964890701.arc RECID=30 STAMP=987692230

archived log file name=/home/oracle/arch4/1\_31\_964890701.arc RECID=31 STAMP=987692233

archived log file name=/home/oracle/arch4/1\_32\_964890701.arc RECID=32 STAMP=987693291

archived log file name=/home/oracle/arch4/1\_33\_964890701.arc RECID=33 STAMP=987693294

archived log file name=/home/oracle/arch4/1\_34\_964890701.arc RECID=34 STAMP=987693445

archived log file name=/home/oracle/arch4/1\_35\_964890701.arc RECID=35 STAMP=987693702

archived log file name=/home/oracle/arch4/1\_36\_964890701.arc RECID=36 STAMP=987693847

#### Making Incremental Backups

##### Purpose of Incremental Backups

使用增量备份的目的主要有：

1. 备份起来快，还原起来也比应用日志快
2. 对于nologging的操作，只能用增量备份进行备份和还原，archivelog做不到
3. 可以使用在主库backup incremental from scn（备库当前的scn），然后将增量备份应用到physical standby database。

##### Making Incremental Backups

做0级备份：

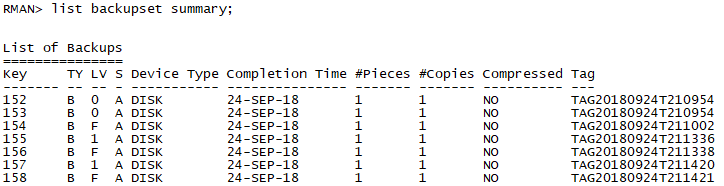
RMAN> BACKUP INCREMENTAL LEVEL 0 DATABASE;

做1级differential increamental backup：

RMAN> BACKUP INCREMENTAL LEVEL 1 TABLESPACE users;

做1级cumulative increamental backup：

RMAN> BACKUP INCREMENTAL LEVEL 1 CUMULATIVE datafile 5;



我们看LV列，0就是说0级备份，1就是1级备份，F就是full backup，这3个F都是控制文件的自动备份。即使是list backupset也看不出是differential的还是cumulative的。

##### Using Block Change Tracking to Improve Incremental Backup Performance

见1.3.6.3

## RMAN Data Repair

### Incarnation

### Validating Database Files and Backups

#### Overview of RMAN Validation

Validation的目的就是确认有没有corrupt blocks和missing files，或者backups是否可用于restore。常用命令格式如下：

VALIDATE

BACKUP ... VALIDATE

RESTORE ... VALIDATE

数据库为了阻止那些会导致备份不可用或者损坏被restore的文件的操作，会自动做如下事情：

1. 在同一时间，只能对同一个文件只进行一个restore的操作；
2. 确保增量备份在恢复的时候按顺序应用
3. 会把一些信息存到backup files，以便检测损坏
4. 当每一个block在被读取或者写入之前，检测顺坏，如果发现损坏，立马报告损坏

##### Checksums and Corrupt Blocks

数据库中有个参数：

SQL> show parameter checksum

NAME TYPE VALUE

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

db\_block\_checksum string TYPICAL

这个值如果是typical，那么数据库在把一个block写入到磁盘前会计算checksum保存在块头，读取的时候，会再次计算checksum，当发现于存储在块头的checksum不符时，就会报告该块坏损了。

这一参数只对数据文件和online redo有效，也就是说，跟backup没关系。

但是，在对一个数据文件backup的时候，backup这条指令会自动做同样的工作：计算每一个block的checksum并且把它store in backups。

言外之意，backup指令在做这件事的时候，跟数据库配没配置db\_block\_checksum一点儿关系都没有。

##### Physical and Logical Block Corruption

Pyhical block corruption就是介质损坏（media corruption），数据库干脆就不认识这个block了，checksum是无效的，block中全是0，或者块头和块尾不一致。

Logical block corruption就是比如块里面某个row piece或者index entry坏损了，就是逻辑上不一致了，这样的block会被记录在alert日志中。

##### Limits for Corrupt Blocks in RMAN Backups

设置maxcorrupt后，当备份数据文件数据文件时，发现坏块，依然不会中断备份，而是把坏块标记为corrupt，继续备份，直到发现的坏块超过了maxcorrupt的设置，备份会被中断。

有如下几点需要注意：

1. maxcorrupt默认值是0，也就是说发现坏块就会中断备份。
2. Set maxcorrupt for datafile … to …是固定句式
3. Set maxcorrupt for datafile只能在run block中使用
4. 被标记为corrupt的block所在的文件，在被restore之后，再次被backup的时候不需要再次为备份这些已被标记过的block而指定maxcorrupt，RMAN认得出来这些已经被标记为corrupt的block
5. 含有被标记为corrupt的block的备份集，可以用validate检测出来。

如下是一个使用示例：

RMAN> run{

2> set maxcorrupt for datafile 5,7 to 10;

3> backup database;

4> }

executing command: SET MAX CORRUPT

Starting backup at 27-SEP-18

##### Interblock and Intrablock Corruption

Interblock corruption是块与块之间的corruption，一定是logical corruption。

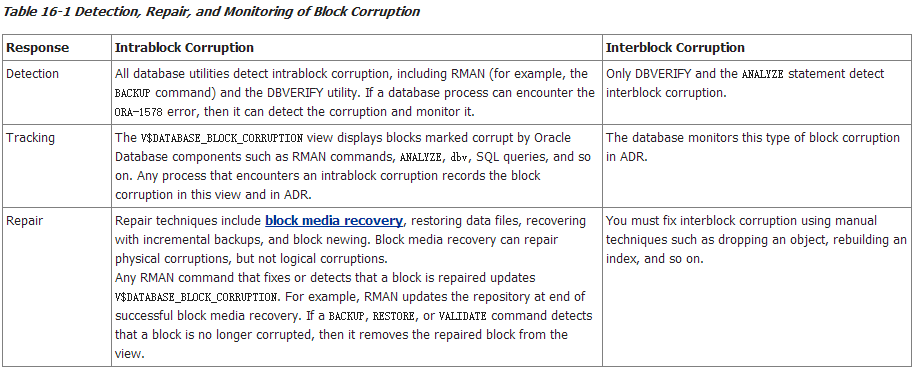
Intrablock corruption是块里面的的corruption，可以是physical和logical俩种情况。

用RMAN和DBV可以检测出Intrablock corruption，而interblock corruption只能靠dbv和analyze检测出来。

V$DATABASE\_BLOCK\_CORRUPTION会记录被标记为corrupt的intrablock corruption，这些被标记的block可能被rman、analyze、dbv或者sql查询等标记的。

对于intrablock corruption，使用介质恢复、restore等都可以进行恢复。同时，被恢复了的block，V$DATABASE\_BLOCK\_CORRUPTION中会删掉关于该block的记录。而对于interblock corruption，必须要手动恢复，比如重建索引，drop表等等。

如下是官文中的比对：



#### Checking for Block Corruption with the VALIDATE Command

Validate的过程中，遇到未被标记过的corrupt的block，会标记为corrupt block，并且更新V$DATABASE\_BLOCK\_CORRUPTION。

无论是physical或是logical block corruption，都可以通过validate确认出来。

Validate可以检查整个数据库，某个文件，或者某个备份集：

1. Validate database会检测每一个数据文件的每一个block，包括spfile和controlfile。
2. Validate datafile也是同理，validate datafile xx block xx还可以指定检查某一个block。
3. Validate backupset xx可以检测某一个备份集是否有文件缺失或者坏块，一旦发现坏块，则会终止validation。

下面我们来看实验：

RMAN> validate database;

Starting validate at 27-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile

channel ORA\_DISK\_1: specifying datafile(s) for validation

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:03

List of Datafiles

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

File Status Marked Corrupt Empty Blocks Blocks Examined High SCN

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

1 OK 0 17966 107522 3824901

File Name: /u01/app/oracle/oradata/PROD4/system01.dbf

Block Type Blocks Failing Blocks Processed

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

Data 0 71775

Index 0 13309

Other 0 4470

File Status Marked Corrupt Empty Blocks Blocks Examined High SCN

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

3 OK 0 20232 101143 3824768

File Name: /u01/app/oracle/oradata/PROD4/sysaux01.dbf

Block Type Blocks Failing Blocks Processed

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

Data 0 19119

Index 0 21498

Other 0 40271

List of Control File and SPFILE

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

File Type Status Blocks Failing Blocks Examined

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

SPFILE OK 0 2

Control File OK 0 646

也可以validate某个数据文件的某个block：

RMAN> validate datafile 1 block 520;

Starting validate at 27-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile

channel ORA\_DISK\_1: specifying datafile(s) for validation

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:01

List of Datafiles

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

File Status Marked Corrupt Empty Blocks Blocks Examined High SCN

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

1 OK 0 0 1 205

File Name: /u01/app/oracle/oradata/PROD4/system01.dbf

Block Type Blocks Failing Blocks Processed

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

Data 0 0

Index 0 0

Other 0 1

Finished validate at 27-SEP-18

还可以validate备份集：

RMAN> list backupset;

List of Backup Sets

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

BS Key Type LV Size Device Type Elapsed Time Completion Time

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

169 Full 2.12M DISK 00:00:00 27-SEP-18

BP Key: 268 Status: AVAILABLE Compressed: NO Tag: TAG20180927T171044

Piece Name: /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_27/o1\_mf\_nnndf\_TAG20180927T171044\_fts7nntw\_.bkp

List of Datafiles in backup set 169

File LV Type Ckp SCN Ckp Time Abs Fuz SCN Sparse Name

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

7 Full 3825754 27-SEP-18 NO /u01/app/oracle/oradata/PROD4/users01.dbf

RMAN> validate backupset 169;

Starting validate at 27-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile backup set

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_27/o1\_mf\_nnndf\_TAG20180927T171044\_fts7nntw\_.bkp

channel ORA\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_27/o1\_mf\_nnndf\_TAG20180927T171044\_fts7nntw\_.bkp tag=TAG20180927T171044

channel ORA\_DISK\_1: restored backup piece 1

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:01

Finished validate at 27-SEP-18

##### Make Parallel the Validation of a Data File

如果数据文件很大，可以开并行validate datafile，每个通道validate一个section：

RMAN> RUN

2> {

3> ALLOCATE CHANNEL c1 DEVICE TYPE DISK;

4> ALLOCATE CHANNEL c2 DEVICE TYPE DISK;

5> VALIDATE DATAFILE 1 SECTION SIZE 1200M;

6> }

当然还可以开多通道validate database：

RMAN> RUN

2> {

3> ALLOCATE CHANNEL c1 DEVICE TYPE DISK;

4> ALLOCATE CHANNEL c2 DEVICE TYPE DISK;

5> ALLOCATE CHANNEL c3 DEVICE TYPE DISK;

6> ALLOCATE CHANNEL c4 DEVICE TYPE DISK;

7> VALIDATE database;

8> }

#### Validating Database Files with BACKUP VALIDATE

就是连检查带备份。

下面的指令是只检查physical corruption：

RMAN> BACKUP VALIDATE DATABASE ARCHIVELOG ALL;

如果除了physical corruption，还要检查logical corruption，要使用如下指令：

BACKUP VALIDATE

CHECK LOGICAL

DATABASE ARCHIVELOG ALL;

#### Validating Backups Before Restoring Them

执行如下指令，只是简单的检查，就是restore前的确认，并不真的restore，如果没有报错，说明接下来真的执行restore是没有问题的。

该指令必须在数据库mount或者open状态下进行：

RESTORE DATABASE VALIDATE;

RESTORE ARCHIVELOG ALL VALIDATE;

做个小实验：

RMAN> restore datafile 7 validate;

Starting restore at 27-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile backup set

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_27/o1\_mf\_nnndf\_TAG20180927T171044\_fts7nntw\_.bkp

channel ORA\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_27/o1\_mf\_nnndf\_TAG20180927T171044\_fts7nntw\_.bkp tag=TAG20180927T171044

channel ORA\_DISK\_1: restored backup piece 1

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:01

Finished restore at 27-SEP-18

我们看到，并没有真正执行restore，而只是对备份集进行了validation。

#### Validating CDBs and PDBs

跟backup一个道理，如果是直接用rman连到cdb，那么执行如下指令确认cdb和root：

VALIDATE DATABASE;

VALIDATE DATABASE ROOT;

使用如下指令确认pdb：

RMAN> VALIDATE PLUGGABLE DATABASE pdbprod1,pdbprod2;

还可以执行如下指令：

RESTORE PLUGGABLE DATABASE VALIDATE

如果用rman连到某个pdb，就直接validate database就是确认的本pdb。

### Preparing for Complete Database Recovery

#### Identifying the Database Files to Restore or Recover

其实就是使用validate进行一遍扫描确认。

比如我这里把7号文件进行破坏：

BBED> copy file 4 block 10 to file 7 block 300;

这里我们同时看一下V$DATABASE\_BLOCK\_CORRUPTION：

SQL> select \* from V$DATABASE\_BLOCK\_CORRUPTION;

no rows selected

说明此时还没有发现坏块。

然后我们validate datafile 7或者validate database：

RMAN> validate database;

Starting validate at 28-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile

channel ORA\_DISK\_1: specifying datafile(s) for validation

input datafile file number=00001 name=/u01/app/oracle/oradata/PROD4/system01.dbf

input datafile file number=00003 name=/u01/app/oracle/oradata/PROD4/sysaux01.dbf

input datafile file number=00004 name=/u01/app/oracle/oradata/PROD4/undotbs01.dbf

input datafile file number=00007 name=/u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:03

List of Datafiles

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

File Status Marked Corrupt Empty Blocks Blocks Examined High SCN

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

1 OK 0 17961 107522 3887288

File Name: /u01/app/oracle/oradata/PROD4/system01.dbf

Block Type Blocks Failing Blocks Processed

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

Data 0 71775

Index 0 13314

Other 0 4470

File Status Marked Corrupt Empty Blocks Blocks Examined High SCN

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

7 FAILED 0 1631 2885 3401205

File Name: /u01/app/oracle/oradata/PROD4/users01.dbf

Block Type Blocks Failing Blocks Processed

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

Data 0 72

Index 0 0

Other 1 1177

同时我们可以在告警日志中看到报错：

2018-09-28T14:56:51.823037+08:00

Hex dump of (file 7, block 300) in trace file /u01/app/oracle/diag/rdbms/prod4/PROD4/trace/PROD4\_ora\_69650.trc

Corrupt block relative dba: 0x01c0012c (file 7, block 300)

Bad header found during validation

Data in bad block:

type: 30 format: 2 rdba: 0x0100000a

last change scn: 0x0000.0000.00157694 seq: 0x2 flg: 0x04

spare3: 0x0

consistency value in tail: 0x76941e02

check value in block header: 0x1b3

computed block checksum: 0x0

Reread of blocknum=300, file=/u01/app/oracle/oradata/PROD4/users01.dbf. found same corrupt data

Reread of blocknum=300, file=/u01/app/oracle/oradata/PROD4/users01.dbf. found same corrupt data

Reread of blocknum=300, file=/u01/app/oracle/oradata/PROD4/users01.dbf. found same corrupt data

Reread of blocknum=300, file=/u01/app/oracle/oradata/PROD4/users01.dbf. found same corrupt data

Reread of blocknum=300, file=/u01/app/oracle/oradata/PROD4/users01.dbf. found same corrupt data

我们看到，告警日志中很明确的指出是哪个block损坏了。

现在我们再来看V$DATABASE\_BLOCK\_CORRUPTION：

SQL> select \* from V$DATABASE\_BLOCK\_CORRUPTION;

FILE# BLOCK# BLOCKS CORRUPTION\_CHANGE# CORRUPTIO CON\_ID

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

7 300 1 0 CORRUPT 0

我们看到，现在视图中有内容了，跟我们在validate的使用中所说的一致。

#### Determining the DBID of the Database

当需要从自动备份中恢复spfile和controlfile时需要明确dbid。

[oracle@host01 ~]$ rman target /

Recovery Manager: Release 12.2.0.1.0 - Production on Fri Sep 28 13:03:43 2018

Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.

connected to target database: PROD4 (DBID=1716001866)

#### Previewing Backups Used in Restore Operations

You can apply RESTORE ... PREVIEW to any RESTORE operation to create a detailed list of every backup to be used in the requested RESTORE operation, and the necessary target SCN for recovery after the RESTORE operation is complete. This command accesses the RMAN repository to query the backup metadata, but does not actually read the backup files to ensure that they can be restored.

可以使用restore…preview来列出restore时所需要的所有备份集（就是所有的restore命令最后加个preview就行），并且列出在之后recover时要恢复到的target scn。该命令是通过从rman repository中查出备份的metadata来实现其功能的，但是该命令并不能确认这些备份是不是真的能用来进行恢复，也就是说备份集有损坏，该命令检查不出来。

RMAN> restore datafile 7 preview;

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

List of Backup Sets

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

BS Key Type LV Size Device Type Elapsed Time Completion Time

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

171 Full 1.33G DISK 00:00:03 28-SEP-18

BP Key: 270 Status: AVAILABLE Compressed: NO Tag: TAG20180928T133152

Piece Name: /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp

List of Datafiles in backup set 171

File LV Type Ckp SCN Ckp Time Abs Fuz SCN Sparse Name

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

7 Full 3876829 28-SEP-18 NO /u01/app/oracle/oradata/PROD4/users01.dbf

archived logs generated after SCN 3876829 not found in repository

recovery will be done up to SCN 3876829

Media recovery start SCN is 3876829

Recovery must be done beyond SCN 3876829 to clear datafile fuzziness

Finished restore at 28-SEP-18

这里我们看到开始的scn跟结束的scn一致，是因为我在破坏之前，没有对users表空间上的对象进行dml。

还可以只列个概要的，就是在preview后面加个summary：

RMAN> restore datafile 7 preview summary;

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

List of Backups

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

Key TY LV S Device Type Completion Time #Pieces #Copies Compressed Tag

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

171 B F A DISK 28-SEP-18 1 1 NO TAG20180928T133152

archived logs generated after SCN 3876829 not found in repository

recovery will be done up to SCN 3876829

Media recovery start SCN is 3876829

Recovery must be done beyond SCN 3876829 to clear datafile fuzziness

Finished restore at 28-SEP-18

##### Recalling Off-site Backups

跟SBT有关，不说了先。

#### Validating Backups Before Restoring Them

这个在之前的validate部分说过，这里概括下，就是该指令只确认备份集的可用性，并不真的实施restore。

RMAN> restore datafile 7 validate;

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting validation of datafile backup set

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp

channel ORA\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp tag=TAG20180928T133152

channel ORA\_DISK\_1: restored backup piece 1

channel ORA\_DISK\_1: validation complete, elapsed time: 00:00:03

Finished restore at 28-SEP-18

这里可以更明确的看出这一点，如果真的会同时实施restore，那么这里会报错，因为我现在的datafile 7还是online状态。

#### Restoring Archived Redo Logs Needed for Recovery

这个意义不大，在recover的时候，rman会自动从备份集恢复所需要的归档日志。

这里需要说的是，可以使用set命令，指定归档日志恢复的路径，下面是官文的示例：

RUN

{

SET ARCHIVELOG DESTINATION TO '/oracle/temp\_restore';

RESTORE ARCHIVELOG ALL;

# restore and recover data files as needed

}

开头也说了，recover的时候会自动恢复media recovery需要的归档，所以还可以这样操作

还是官文的示例，就是还是说的是把归档恢复到指定路径，但是不用非得先把归档恢复出来：

RUN

{

SET ARCHIVELOG DESTINATION TO '/oracle/temp\_restore';

RESTORE DATABASE;

RECOVER DATABASE; # restores and recovers logs automatically

}

注意，该set archivelog destination可以多次指定，指定到多个路径，还是看下官文的示例：

RUN

{

# Set a new location for logs 1 through 100.

SET ARCHIVELOG DESTINATION TO '/fs1/tmp';

RESTORE ARCHIVELOG FROM SEQUENCE 1 UNTIL SEQUENCE 100;

# Set a new location for logs 101 through 200.

SET ARCHIVELOG DESTINATION TO '/fs2/tmp';

RESTORE ARCHIVELOG FROM SEQUENCE 101 UNTIL SEQUENCE 200;

# Set a new location for logs 201 through 300.

SET ARCHIVELOG DESTINATION TO '/fs3/tmp';

RESTORE ARCHIVELOG FROM SEQUENCE 201 UNTIL SEQUENCE 300;

# restore and recover data files as needed

}

我们看到，set跟restore得是一组一组的写一起。

其实set archivelog destination无非就是默认路径可能不够大，放不下recover是所需要的归档日志。

但是如果是默认路径是fra的话，归档日志应用完了，oracle会自动删除它们，所以，如果数据库默认的归档路径不是fra的话，一方面要考虑剩余空间大小，二方面如果想删掉已经应用过的归档日志，需要在recover的时候加上delete archivelog：

RECOVER DATABASE DELETE ARCHIVELOG;

### Performing Complete Database Recovery

#### For the Whole Database

这里主要提一下temp表空间。因为tempfile既然不会被备份，那么也就不会被从备份集中恢复回来。但是当数据库open的时候，oracle会根据控制文件中记录的temp表空间的信息原封不动的创建出来，包括大小，autoextend、maxsize等属性都不会变。

如果tempfile是OMF的，那么就会被自动创建到db\_create\_file\_dest，如果不是，那么就会被创建到原来的路径下。

但是，只会创建丢失了的临时表空间，没丢的不会进行重建。同时，如果创建失败，alert日志中会记录。

#### For a Tablespace/ a Datafile

这里就把tablespace和datafile一起说了。操作是一样的，恢复tablespace，可以看作是恢复多个datafile的一个总体操作。

我们之前破坏了datafile 7，也就破坏了tablespace users。

因为该表空间只是个普通表空间，不是system和sysaux，那么我们并不需要关闭数据库，就可以进行恢复。

首先我们要先将数据文件offline，online的状态会报错：

RMAN> restore datafile 7;

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

ORA-19573: cannot obtain exclusive enqueue for datafile 7

ORA-45909: restore, recover or block media recovery may be in progress

所以我们来将数据文件offline，如果是offline一个tablespace，那么就相当于把该tablespace的所有数据文件全部offline，所以是同理的，这里只offline datafile 7：

SQL> alter database datafile 7 offline;

Database altered.

然后我们来恢复数据文件，当然也可以retore tablespace users：

RMAN> restore tablespace users;

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: starting datafile backup set restore

channel ORA\_DISK\_1: specifying datafile(s) to restore from backup set

channel ORA\_DISK\_1: restoring datafile 00007 to /u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp

channel ORA\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp tag=TAG20180928T133152

channel ORA\_DISK\_1: restored backup piece 1

channel ORA\_DISK\_1: restore complete, elapsed time: 00:00:01

Finished restore at 28-SEP-18

我们来看告警日志：

2018-09-28T16:39:42.873745+08:00

Full restore complete of datafile 7 /u01/app/oracle/oradata/PROD4/users01.dbf. Elapsed time: 0:00:00

checkpoint is 3876829

last deallocation scn is 3

现在我们来试着把数据文件online：

SQL> alter tablespace users online;

alter tablespace users online

\*

ERROR at line 1:

ORA-01113: file 7 needs media recovery

ORA-01110: data file 7: '/u01/app/oracle/oradata/PROD4/users01.dbf'

告警日志如下：

2018-09-28T16:42:08.981721+08:00

alter tablespace users online

ORA-1113 signalled during: alter tablespace users online...

2018-09-28T16:42:09.010954+08:00

Errors in file /u01/app/oracle/diag/rdbms/prod4/PROD4/trace/PROD4\_m000\_76054.trc:

ORA-01110: data file 7: '/u01/app/oracle/oradata/PROD4/users01.dbf'

ORA-01208: data file is an old version - not accessing current version

我们看到数据文件7需要实例恢复，说datafile 7版本太旧了。为什么呢？！明明我什么都没改变。

原因就是，我们的backup的时候是在数据库open时进行的备份，本来就是非一致性的备份集，而此时数据库又已经是open状态了。所以一定要使用recover，应用日志。这就跟你一个数据库啥都没干，shutdown abort然后open，照样要进行media recovery。

现在我们进行recover：

RMAN> recover datafile 7;

Starting recover at 28-SEP-18

using channel ORA\_DISK\_1

starting media recovery

media recovery complete, elapsed time: 00:00:00

Finished recover at 28-SEP-18

我们再看告警日志：

2018-09-28T16:46:27.772265+08:00

alter database recover datafile list clear

Completed: alter database recover datafile list clear

alter database recover

if needed datafile 7

2018-09-28T16:46:27.777857+08:00

Media Recovery Start

2018-09-28T16:46:27.778306+08:00

Serial Media Recovery started

2018-09-28T16:46:27.806053+08:00

Recovery of Online Redo Log: Thread 1 Group 2 Seq 44 Reading mem 0

Mem# 0: /u01/app/oracle/oradata/PROD4/redo02.log

2018-09-28T16:46:27.839919+08:00

Media Recovery Complete (PROD4)

Completed: alter database recover

if needed datafile 7

我们看到，其实相当于再数据库中执行了俩条sql指令。

现在我们再来online datafile 7:

SQL> alter database datafile 7 online;

Database altered.

我们再来查V$DATABASE\_BLOCK\_CORRUPTION：

SQL> select \* from V$DATABASE\_BLOCK\_CORRUPTION;

no rows selected

至此datafile 7恢复完了。当然如果是datafile 7丢失，也是一样的恢复方式，我们在下面介绍将数据文件恢复到新路径的时候来模拟datafile丢失的情况。

#### Set Newname and Switch to a Copy

这里我们删除datafile 7，然后做俩次恢复，一次使用set newname重新为数据文件指定路径，一次使用不是用set newname。

我们先把数据文件删了：

SQL> select name from v$datafile where file#=7;

NAME

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

/u01/app/oracle/oradata/PROD4/users01.dbf

SQL> !rm -f /u01/app/oracle/oradata/PROD4/users01.dbf

然后我们进行恢复：

我们先直接恢复，仍然是要将datafile 7 offline：

RMAN> alter database datafile 7 offline;

using target database control file instead of recovery catalog

Statement processed

然后来restore：

RMAN> restore datafile 7;

Starting restore at 28-SEP-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: restoring datafile 00007 to /u01/app/oracle/oradata/PROD4/users01.dbf

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp

Finished restore at 28-SEP-18

告警日志如下：

2018-09-28T20:03:36.132513+08:00

Full restore complete of datafile 7 /u01/app/oracle/oradata/PROD4/users01.dbf. Elapsed time: 0:00:00

checkpoint is 3876829

last deallocation scn is 3

现在，数据文件被恢复回去了，接下来就可以recover了，但是我们这里不急着recover，我们再做一次restore。

这一次我们把datafile 7指定到新的路径：

RUN

{

ALLOCATE CHANNEL c1 DEVICE TYPE DISK;

SET NEWNAME FOR datafile 7 to '/home/oracle/user\_7.dbf';

RESTORE DATAFILE 7;

}

allocated channel: c1

channel c1: SID=747 device type=DISK

executing command: SET NEWNAME

Starting restore at 28-SEP-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: restoring datafile 00007 to /home/oracle/user\_7.dbf

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_09\_28/o1\_mf\_nnndf\_TAG20180928T133152\_ftvh693m\_.bkp

Finished restore at 28-SEP-18

告警日志如下：

2018-09-28T20:15:19.937685+08:00

Full restore complete of datafile 7 to datafile copy /home/oracle/user\_7.dbf. Elapsed time: 0:00:00

checkpoint is 3876829

last deallocation scn is 3

现在我们来看控制文件是怎么记录的：

DATA FILE #7:

name #4: /u01/app/oracle/oradata/PROD4/users01.dbf

creation size=0 block size=8192 status=0x1c flg=0x0 head=4 tail=4 dup=1

pdb\_id 0, tablespace 4, index=4 krfil=7 prev\_file\_in\_ts=0 prev\_file\_in\_pdb=0

unrecoverable scn: 0x0000000000000000 01/01/1988 00:00:00

Checkpoint cnt:171 scn: 0x00000000003b9024 09/28/2018 16:50:36

Stop scn: 0x00000000003b9a76 09/28/2018 20:02:33

Creation Checkpointed at scn: 0x000000000000752f 01/26/2017 13:54:50

thread:0 rba:(0x0.0.0)

enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000

Offline scn: 0x00000000002af08a prev\_range: 1

Online Checkpointed at scn: 0x0000000000330dcf 09/19/2018 16:44:08

thread:1 rba:(0x19.ebd1.10)

enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000

也就是说，现在控制文件中记录的路径还是原路径。

那么我们想让控制文件知道我们该路径了，那就需要执行switch datafile to copy：

RMAN> switch datafile 7 to copy;

datafile 7 switched to datafile copy "/home/oracle/user\_7.dbf"

告警日志如下：

2018-09-28T20:28:13.460946+08:00

Switch of datafile 7 complete to datafile copy

checkpoint is 3876829

现在我们再来看controlfile中的记录：

DATA FILE #7:

name #4: /home/oracle/user\_7.dbf

creation size=0 block size=8192 status=0x1c flg=0x0 head=4 tail=4 dup=1

pdb\_id 0, tablespace 4, index=4 krfil=7 prev\_file\_in\_ts=0 prev\_file\_in\_pdb=0

其实从v$datafile就可以看到：

SQL> select name from v$datafile where file#=7;

NAME

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

/home/oracle/user\_7.dbf

##### Alter Database Rename File 'xx' to 'xx'

还有一种办法，与switch datafile的作用一模一样，但是是用sql完成的。

但是不像switch datafile xx to copy，默认switch最新restore出来的“copy”，rename file可以直接指定某一个数据文件注册到控制文件中：

SQL> alter database rename file '/home/oracle/user\_7.dbf' to '/u01/app/oracle/oradata/PROD4/users01.dbf';

Database altered.

然后我们查v$datafile：

SQL> select name from v$datafile where file#=7;

NAME

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

/u01/app/oracle/oradata/PROD4/users01.dbf

最后我们把datafile 7 recover并且online：

RMAN> recover datafile 7;

Finished recover at 28-SEP-18

RMAN> alter database datafile 7 online;

Statement processed

##### Run Block中并行Switch Datafile：

注意，switch datafile也可以开并行，如下是俩种用法，第一种是指定datafile进行switch，第二种是所有datafile进行switch：

RUN

{

ALLOCATE CHANNEL c1 DEVICE TYPE DISK;

ALLOCATE CHANNEL c2 DEVICE TYPE DISK;

SWITCH datafile xx,xx,xx,xx,xx;

SWITCH datafile all;

}

注意，如果是在run block中，swith datafile的命令就没有to copy。

### Performing Complete Recovery of CDBs

跟non-cdb的restore database一毛一样。也要在mount下进行。

#### For ROOT

用如下指令，也要在mount下进行：

RESTORE DATABASE ROOT;

RECOVER DATABASE ROOT;

#### For PDBs

##### Restore Datafile

这个我们直接做个实验，我们先在cdb下做个全备，然后我们把pdbprod 2的users表空间对应的数据文件删掉：

RMAN> backup database;

[oracle@host01 ~]$ rm -f /u01/app/oracle/oradata/PRODCDB/PDBPROD2/users01.dbf

我们看下是几号文件：

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

FILE# NAME

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

16 /u01/app/oracle/oradata/PRODCDB/PDBPROD2/users01.dbf

我们看到是16号文件。

同样道理，我们要先把数据文件offline：

SQL> alter database datafile 16 offline;

alter database datafile 16 offline

\*

ERROR at line 1:

ORA-01516: nonexistent log file, data file, or temporary file "16" in the current container

我们看到，16号文件不属于cdb，所以不能alter database，而要alter pluggable database：

SQL> alter pluggable database datafile 16 offline;

alter pluggable database datafile 16 offline

\*

ERROR at line 1:

ORA-65046: operation not allowed from outside a pluggable database

我们看到，cdb下是不允许做次操作的。所以我们一定要到对应的pdb下offline：

SQL> alter session set container=pdbprod2;

Session altered.

SQL> alter pluggable database datafile 16 offline;

Pluggable database altered.

现在我们来进行恢复：

RMAN> restore datafile 16;

Finished restore at 28-SEP-18

RMAN> recover datafile 16;

Finished recover at 28-SEP-18

SQL> alter pluggable database datafile 16 online;

Pluggable database altered.

##### Resotre Pluggable Database

这次我们还是把16号文件删掉：

[oracle@host01 ~]$ rm -f /u01/app/oracle/oradata/PRODCDB/PDBPROD2/users01.dbf

这次我们直接retore pluggable database。

那么执行该操作，首先要将pdbprod2 close。

SQL> alter pluggable database pdbprod2 close;

Pluggable database altered.

然后执行恢复：

RMAN> restore pluggable database pdbprod2;

Finished restore at 28-SEP-18

RMAN> recover pluggable database pdbprod2;

Finished recover at 28-SEP-18

RMAN> alter pluggable database pdbprod2 open;

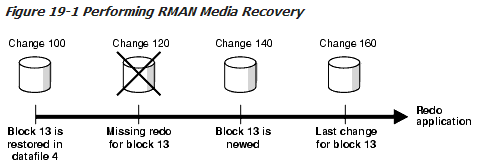
Statement processed

### Performing Block Media Recovery

如果某些block坏了，那么可以使用rman只恢复这些坏损的block，前提条件如下：

1. 数据库是归档模式的
2. 包含对应block的备份集必须是全备或者0级备份
3. Rman只能应用归档日志来进行recover，当然还有redo，但是绝不能应用1级备份。
4. 归档日志不能断，也就是说recover所需要的归档日志不能损坏或丢失
5. 在一些情况下允许redo存在丢失

我们来看允许存在redo丢失的情况：



我们来看这个示例的意思。示例就是说block13在recover的过程中，开始应用redo了，但是发现block 13相关change 120的redo丢失了，那么recover不会停止，会继续读redo，当发现发生change 140的操作是一个重新分配block 13的操作。

举个例子，比如block 13原本属于a表，但是a表被删了或者truncate了，在change 140的时候block 13被重新分配给了b表。

那么此时recover就可以继续进行，因为此时block 13已经被重新分配，之前做了什么并不重要了。

相关指令：

RECOVER

DATAFILE 8 BLOCK 13

DATAFILE 2 BLOCK 19;

如果是想一次性recover V$DATABASE\_BLOCK\_CORRUPTION中记录的所有block：

RECOVER CORRUPTION LIST;

我们做个实验，先破坏block：

BBED> copy file 1 block 3 to file 7 block 132;

BBED> copy file 1 block 4 to file 7 block 133;

然后我们直接不validate，直接RECOVER CORRUPTION LIST：

RMAN> RECOVER CORRUPTION LIST;

Starting recover at 02-OCT-18

using channel ORA\_DISK\_1

starting media recovery

media recovery complete, elapsed time: 00:00:00

Finished recover at 02-OCT-18

我们看到，其实rman果然不会恢复v$database\_block\_corruption中没有记录的block。所以我们要先validate一下，然后再recover：

RMAN> validate database;

SQL> select \* from V$DATABASE\_BLOCK\_CORRUPTION;

FILE# BLOCK# BLOCKS CORRUPTION\_CHANGE# CORRUPTIO CON\_ID

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

7 132 2 0 CORRUPT 0

RMAN> RECOVER CORRUPTION LIST;

Starting recover at 02-OCT-18

using channel ORA\_DISK\_1

channel ORA\_DISK\_1: restoring block(s)

channel ORA\_DISK\_1: specifying block(s) to restore from backup set

restoring blocks of datafile 00007

channel ORA\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ovzkh\_.bkp

channel ORA\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ovzkh\_.bkp tag=TAG20181002T193319

channel ORA\_DISK\_1: restored block(s) from backup piece 1

channel ORA\_DISK\_1: block restore complete, elapsed time: 00:00:01

starting media recovery

media recovery complete, elapsed time: 00:00:01

Finished recover at 02-OCT-18

SQL> select \* from V$DATABASE\_BLOCK\_CORRUPTION;

no rows selected

ok，现在实验完毕。

## Tablespace Point-in-Time Recovery（TSPITR）

### Overview of RMAN TSPITR

#### 目的

实施tspitr的目的就是为了恢复一些误操作：

1. 对表的误操作，包括truncate、drop purge这些使用flashback table无法进行还原的操作
2. 对表空间的误操作，比如把表空间删除了，这是flashback database无法进行还原的操作
3. 只对某个表空间进行不完全恢复，这是单纯recover until做不到的。
4. 对数据库其他部分的正常运行没有任何影响，这优势flashback database做不到的。

#### 相关术语

Target database：包含要恢复的表空间的数据库，说白了就是源库

Target time：表空间要恢复到的时间点或者scn

Auxiliary database：辅助数据库，就是为了做tspitr时，临时建的一个database

Auxiliary destination：可选选项，auxiliary set存放的路径，只对rman-managed auxiliary database有用，对user-managed auxiliary database指定该参数会报错

Recovery set：就是要恢复的表空间包含的数据文件们

Auxiliary set：就是除了recovery set以外，auxiliary database启动所需要的其他文件，这包括：

1. system和sysaux表空间
2. control file
3. undo表空间
4. 临时表空间
5. 用来将表空间恢复到指定时间点的源库的归档日志
6. 在线日志，这个在线日志时auxiliary database在open resetlogs的时候产生的online redo log，跟源库没关系
7. Auxiliary set不包含参数文件、密码文件和相关的network files

#### 模式

分为3中模式：

1. Fully Automated (the default) 全自动
2. Automated：RMAN-Managed Auxiliary Database with User Settings 半自动
3. Non-Automated：TSPITR and User-Managed Auxiliary Database 手动

#### 工作原理

TSTIPR工作步骤如下：

1. 如果要被恢复的表空间（recovery set中的表空间）还没被删，就会经过DBMS\_TTS.TRANSPORT\_SET\_CHECK检查，来检查表空间否是自包含，如果表空间不是自包含，那么检查后结果会放在TRANSPORT\_SET\_VIOLATIONS视图中，也就是说，oracle查询该视图时发现返回不为空，则会终止TSTIPR。
2. 会检查是否可以连接到一个user-managed auxiliary database，如果没有，rman就自己创建一个auxiliary database并且连接它。
3. 如果要恢复的表空间在target database还没被drop，那么就会先被offline。
4. 为auxiliary database restore一个过去时间点的controlfile（就是从controlfile的一个老备份恢复controlfile）
5. 为auxiliary database restore recovery set和auxiliary set
6. 然后把auxiliary database recover到target time
7. 把auxiliary database open resetlogs
8. 把要恢复的表空间在auxiliary database中置为readonly
9. 使用data pump工具导出要恢复的表空间，生成一个transportable tablespace dump file，说白了就是要使用表空间传输技术了。
10. 关闭auxiliary database
11. 从target database删除原来的表空间
12. 然后传输表空间到target database上
13. 传输表空间完成后，把表空间online到read write，然后再立马offline
14. 最后，删除auxiliary database

RMAN把恢复出来的表空间online到read write是为了看看传输成功没，然后立马offline，是为了让用户赶紧备份，因为之前的被恢复的表空间的备份已经不能再使用了。也就是说，最后要自己手动把恢复出来的表空间online。

#### 限制和特殊情况

限制有如下：

1. 要想使用tspitr，必须有源库的归档日志。
2. 不能使用tspitr来恢复current default tablespace。也就是说如果一个表空间是某个用户的default tablespace，不能使用tspitr恢复。
3. 不能使用tspitr恢复undo 表空间和sys用户下的对象。
4. Tspitr不恢复表空间上相应的query optimizer statistics。
5. Tspitr一旦进行完，也就是被恢复的表空间一旦online，那么之前的备份就不能用当前的控制文件进行恢复了。
6. 因为tspitr是基于表空间传输（xtts）的，所以xtts的限制也就是tspitr的限制

如下是一些特殊情况：

1. 如果有一个tablespace被rename了，想要恢复到rename之前的状态，在恢复时，必须使用rename操作之前的名字，也就是老名字。Tspitr完成后，源库恢复出老名字的表空间，但不会删除新名字的表空间，也就是说tspitr结束后，会有俩个“一样”的表空间。
2. 如果tbs1不是自包含（self-contained），tbs1上有表与tbs2上的某个表存在约束关系，那么要想恢复tbs1，就必须在recovery set中同时包含tbs2。
3. CONTROL\_FILE\_RECORD\_KEEP\_TIME如果设置的不够长，其中有些记录被覆盖重用，那么target time又设置的比较老，就会导致tspitr失败，所以CONTROL\_FILE\_RECORD\_KEEP\_TIME一定要设置的足够长，来保证tspitr的成功进行。
4. 如果要再次tspitr，如果没有使用recovery catalog的话，而是只使用controlfile，那么必须要先把已经恢复的表空间们删除掉，才能再次执行tspitr。

### Planning and Preparing for TSPITR

第一步，确定合适的target time，可以通过flashback query等方式，确定过去某一时间点的数据状态是否是需要的数据状态，或者确定表空间删除前的某一个时间点。

第二步，确定recovery set。如果不是自包含的表空间，要么就把关联关系都先处理掉，要么就把相关联对象所在的表空间也包含到recovery set中。如果表空间在tspitr之前已经被删除了，那么rman就不会执行DBMS\_TTS.TRANSPORT\_SET\_CHECK，而是在transport tablespace的时候才做。

BEGIN

DBMS\_TTS.TRANSPORT\_SET\_CHECK('USERS,TEST', TRUE,TRUE);

END;

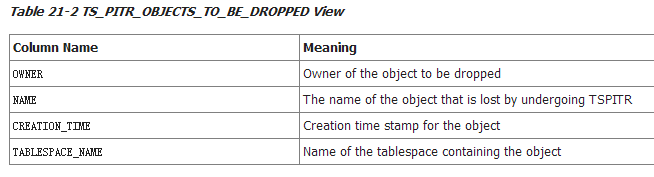
/

SQL> SELECT \* FROM TRANSPORT\_SET\_VIOLATIONS;

no rows selected

第三步，确定和恢复因tspitr而丢失的对象。可以在tspitr之前将需要的对象dump出来，在tspitr之后，再impdp回去。

在tspitr之后，可以通过视图TS\_PITR\_OBJECTS\_TO\_BE\_DROPPED来查询tspitr导致丢失的对象。



可以通过时间和scn俩种方式来查：

SELECT OWNER, NAME, TABLESPACE\_NAME,

TO\_CHAR(CREATION\_TIME, 'YYYY-MM-DD:HH24:MI:SS')

FROM TS\_PITR\_OBJECTS\_TO\_BE\_DROPPED

WHERE TABLESPACE\_NAME IN ('USERS','TOOLS')

AND CREATION\_TIME > TO\_DATE('02-NOV-07:07:03:11','YY-MON-DD:HH24:MI:SS')

ORDER BY TABLESPACE\_NAME, CREATION\_TIME;

SELECT OWNER, NAME, TABLESPACE\_NAME,

TO\_CHAR(CREATION\_TIME,'YYYY-MM-DD:HH24:MI:SS')

FROM TS\_PITR\_OBJECTS\_TO\_BE\_DROPPED

WHERE TABLESPACE\_NAME IN ('USERS','TOOLS')

AND CREATION\_TIME > TO\_DATE(TO\_CHAR(SCN\_TO\_TIMESTAMP(1645870),'MM/DD/YYYY HH24:MI:SS'),'MM/DD/YYYY HH24:MI:SS')

ORDER BY TABLESPACE\_NAME, CREATION\_TIME;

### TSPITR

#### 实验环境准备

我们先来准备实验环境。

SQL> select sequence#,to\_char(SEQUENCE#,'xxxxxxxxxxxxxxxxxx') seq\_hex,to\_char(FIRST\_CHANGE#,'xxxxxxxxxxxxxxxxxxxxxxxx') first\_change\_hex from v$archived\_log ;

SEQUENCE# SEQ\_HEX FIRST\_CHANGE\_HEX

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

1 1 3d61cb

2 2 3da79e

同时，在sequence 1的时候就已经有了个一个全备：

RMAN> list backupset summary;

List of Backups

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

Key TY LV S Device Type Completion Time #Pieces #Copies Compressed Tag

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

174 B F A DISK 02-OCT-18 1 1 NO TAG20181002T193319

175 B F A DISK 02-OCT-18 1 1 NO TAG20181002T193319

176 B F A DISK 02-OCT-18 1 1 NO TAG20181002T193319

然后，我们开始准备实验用户实验表实验数据：

SQL> create user test identified by test default tablespace test;

User created.

SQL> grant dba to test;

Grant succeeded.

SQL> conn test/test

Connected.

SQL> create table test(id number,name varchar2(10));

Table created.

SQL> insert into test values(1,'AAAAA');

1 row created.

SQL> commit;

Commit complete.

做完这一切，并没有发生日志切换：

SQL> select sequence#,to\_char(SEQUENCE#,'xxxxxxxxxxxxxxxxxx') seq\_hex,to\_char(FIRST\_CHANGE#,'xxxxxxxxxxxxxxxxxxxxxxxx') first\_change\_hex from v$archived\_log ;

SEQUENCE# SEQ\_HEX FIRST\_CHANGE\_HEX

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

1 1 3d61cb

2 2 3da79e

现在我们来进行日志切换：

SQL> alter system switch logfile;

System altered.

SQL> alter system switch logfile;

System altered.

SQL> alter system switch logfile;

System altered.

SQL> alter system switch logfile;

System altered.

SQL> select sequence#,to\_char(SEQUENCE#,'xxxxxxxxxxxxxxxxxx') seq\_hex,to\_char(FIRST\_CHANGE#,'xxxxxxxxxxxxxxxxxxxxxxxx') first\_change\_hex from v$archived\_log ;

SEQUENCE# SEQ\_HEX FIRST\_CHANGE\_HEX

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

1 1 3d61cb

2 2 3da79e

3 3 3da821

4 4 3dbcf9

5 5 3dbd3a

6 6 3dbd51

现在我们看到，产生了3、4、5、6归档。那么显然，我们所做的事务一定是在3中。

#### Fully Automated RMAN TSPITR（全自动）

下面我们就来把test表空间恢复到log sequence 2，也就是相当于recover 该表空间时，应用日志只应用到log sequence 2，那么sequence 3及以后的内容将会丢失，所以，我们这里一定要写until logseq 3而不是until logseq 2。

选择test表空间的原因，还有我们实验中看到，test表空间是test用户的default tablespace，我们看看能不能进行tspitr操作，因为对于前面所说的current default tablespace不能进行tspitr操作这个并没有理解的很清楚。

直接打开rman，用rman target / 连接就ok，不用指定auxiliary database，rman看到我们没指定就会自动创建，这就是automated TSPITR：

[oracle@host01 haha]$ rman target /

connected to target database: PROD4 (DBID=1716001866)

RMAN> RECOVER TABLESPACE 'TEST' UNTIL LOGSEQ 3 THREAD 1 AUXILIARY DESTINATION '/home/oracle/tspitr';

Starting recover at 05-OCT-18

using channel ORA\_DISK\_1

RMAN-05026: warning: presuming following set of tablespaces applies to specified point-in-time

List of tablespaces expected to have UNDO segments

Tablespace SYSTEM

Tablespace UNDOTBS1

我们看到，这里rman先找了一下有undo段的表空间。

Creating automatic instance, with SID='cFBF'

initialization parameters used for automatic instance:

db\_name=PROD4

db\_unique\_name=cFBF\_pitr\_PROD4

compatible=12.2.0

db\_block\_size=8192

db\_files=200

diagnostic\_dest=/u01/app/oracle

\_system\_trig\_enabled=FALSE

sga\_target=2400M

processes=200

db\_16k\_cache\_size=32M

db\_create\_file\_dest=/home/oracle/tspitr

log\_archive\_dest\_1='location=/home/oracle/tspitr'

#No auxiliary parameter file used

starting up automatic instance PROD4

Oracle instance started

Total System Global Area 2516582400 bytes

Fixed Size 8623832 bytes

Variable Size 603982120 bytes

Database Buffers 1895825408 bytes

Redo Buffers 8151040 bytes

Automatic instance created

然后我们看到，这一步是创建auxiliary database的过程，rman自己生成了一个SID和参数文件。

Running TRANSPORT\_SET\_CHECK on recovery set tablespaces

TRANSPORT\_SET\_CHECK completed successfully

我们看到，这里开始检查要恢复表空间的自包含，这里检查过关。

contents of Memory Script:

{

# set requested point in time

set until logseq 3 thread 1;

# restore the controlfile

restore clone controlfile;

# mount the controlfile

sql clone 'alter database mount clone database';

# archive current online log

sql 'alter system archive log current';

# avoid unnecessary autobackups for structural changes during TSPITR

sql 'begin dbms\_backup\_restore.AutoBackupFlag(FALSE); end;';

}

executing Memory Script

executing command: SET until clause

Starting restore at 05-OCT-18

allocated channel: ORA\_AUX\_DISK\_1

channel ORA\_AUX\_DISK\_1: SID=1 device type=DISK

channel ORA\_AUX\_DISK\_1: starting datafile backup set restore

channel ORA\_AUX\_DISK\_1: restoring control file

channel ORA\_AUX\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_ncsnf\_TAG20181002T193319\_fv6ow7ph\_.bkp

channel ORA\_AUX\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_ncsnf\_TAG20181002T193319\_fv6ow7ph\_.bkp tag=TAG20181002T193319

channel ORA\_AUX\_DISK\_1: restored backup piece 1

channel ORA\_AUX\_DISK\_1: restore complete, elapsed time: 00:00:01

output file name=/home/oracle/tspitr/PROD4/controlfile/o1\_mf\_fvfh8052\_.ctl

Finished restore at 05-OCT-18

这里就是为auxiliary database还原目标时间点时的控制文件。我们看到，是rman自己生成的脚本，先是set until，然后就是restore controlfile。

sql statement: alter database mount clone database

sql statement: alter system archive log current

sql statement: begin dbms\_backup\_restore.AutoBackupFlag(FALSE); end;

然后，这里把auxiliary database mount起来。

contents of Memory Script:

{

# set requested point in time

set until logseq 3 thread 1;

plsql <<<--

declare

sqlstatement varchar2(512);

pdbname varchar2(128);

offline\_not\_needed exception;

pragma exception\_init(offline\_not\_needed, -01539);

begin

pdbname := null; -- pdbname

sqlstatement := 'alter tablespace '|| '"TEST"' ||' offline immediate';

krmicd.writeMsg(6162, sqlstatement);

krmicd.execSql(sqlstatement, 0, pdbname);

exception

when offline\_not\_needed then

null;

end; >>>;

# set destinations for recovery set and auxiliary set datafiles

set newname for clone datafile 1 to new;

set newname for clone datafile 4 to new;

set newname for clone datafile 3 to new;

set newname for clone tempfile 1 to new;

set newname for datafile 5 to

"/u01/app/oracle/oradata/PROD4/test01.dbf";

# switch all tempfiles

switch clone tempfile all;

# restore the tablespaces in the recovery set and the auxiliary set

restore clone datafile 1, 4, 3, 5;

switch clone datafile all;

}

我们看到，这里有生成了一个脚本，脚本中先定义一个匿名存储过程，意思就是把target database的test表空间进行offline。

然后主要就是进行1、4、3、5数据文件的restore，1、4、3分别是system、undotbs1、sysaux，这是属于auxiliary set的，5就是test的数据文件，这是recovery set的。

我们看到，这里就有set newname，在set newname for clone datafile xx to new中，这个to new就是to我们执行的指令中定义的auxiliary destination。

我们看到，这里把临时表空间也恢复到auxiliary database上了。Switch clone tempfiles all就是因为前面有set newname for clone tempfile 1 to new。

但我们也注意到，这里并没有set newname for clone datafile 5，而是set newname for datafile 5。同时指向的路径正是原数据文件所在的路径。

这里datafile 5没有加clone，因为datafile 5并不需要克隆，而是直接要从备份集中restore出来的。

而前面的set newname for clone datafile，是因为我们指定了auxiliary destination，也就是auxiliary set恢复的位置，而我们并没有指定recovery set的路径，所以默认把recovery set的路径就指定为了原路径。

executing Memory Script

executing command: SET until clause

sql statement: alter tablespace "TEST" offline immediate

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

executing command: SET NEWNAME

renamed tempfile 1 to /home/oracle/tspitr/PROD4/datafile/o1\_mf\_temp\_%u\_.tmp in control file

Starting restore at 05-OCT-18

using channel ORA\_AUX\_DISK\_1

channel ORA\_AUX\_DISK\_1: starting datafile backup set restore

channel ORA\_AUX\_DISK\_1: specifying datafile(s) to restore from backup set

channel ORA\_AUX\_DISK\_1: restoring datafile 00001 to /home/oracle/tspitr/PROD4/datafile/o1\_mf\_system\_%u\_.dbf

channel ORA\_AUX\_DISK\_1: restoring datafile 00004 to /home/oracle/tspitr/PROD4/datafile/o1\_mf\_undotbs1\_%u\_.dbf

channel ORA\_AUX\_DISK\_1: restoring datafile 00003 to /home/oracle/tspitr/PROD4/datafile/o1\_mf\_sysaux\_%u\_.dbf

channel ORA\_AUX\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ovzkh\_.bkp

channel ORA\_AUX\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ovzkh\_.bkp tag=TAG20181002T193319

channel ORA\_AUX\_DISK\_1: restored backup piece 1

channel ORA\_AUX\_DISK\_1: restore complete, elapsed time: 00:00:07

channel ORA\_AUX\_DISK\_1: starting datafile backup set restore

channel ORA\_AUX\_DISK\_1: specifying datafile(s) to restore from backup set

channel ORA\_AUX\_DISK\_1: restoring datafile 00005 to /u01/app/oracle/oradata/PROD4/test01.dbf

channel ORA\_AUX\_DISK\_1: reading from backup piece /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ow8t5\_.bkp

channel ORA\_AUX\_DISK\_1: piece handle=/u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_02/o1\_mf\_nnndf\_TAG20181002T193319\_fv6ow8t5\_.bkp tag=TAG20181002T193319

channel ORA\_AUX\_DISK\_1: restored backup piece 1

channel ORA\_AUX\_DISK\_1: restore complete, elapsed time: 00:00:01

Finished restore at 05-OCT-18

datafile 1 switched to datafile copy

input datafile copy RECID=22 STAMP=988709006 file name=/home/oracle/tspitr/PROD4/datafile/o1\_mf\_system\_fvfh86gv\_.dbf

datafile 4 switched to datafile copy

input datafile copy RECID=23 STAMP=988709006 file name=/home/oracle/tspitr/PROD4/datafile/o1\_mf\_undotbs1\_fvfh86hp\_.dbf

datafile 3 switched to datafile copy

input datafile copy RECID=24 STAMP=988709006 file name=/home/oracle/tspitr/PROD4/datafile/o1\_mf\_sysaux\_fvfh86hd\_.dbf

这里便是整个retore file脚本的执行过成，我们看到auxiliary set恢复到了我们指定的auxiliary dest，而recovery set还原到了原路径。

contents of Memory Script:

{

# set requested point in time

set until logseq 3 thread 1;

# online the datafiles restored or switched

sql clone "alter database datafile 1 online";

sql clone "alter database datafile 4 online";

sql clone "alter database datafile 3 online";

sql clone "alter database datafile 5 online";

# recover and open resetlogs

recover clone database tablespace "TEST", "SYSTEM", "UNDOTBS1", "SYSAUX" delete archivelog;

alter clone database open resetlogs;

}

接下来rman生成的这个脚本，就是把auxiliary database进行recover然后open resetlogs的过程，我们看到，在脚本的最开始，通过set until指定了recover进行到的位置。

executing Memory Script

executing command: SET until clause

sql statement: alter database datafile 1 online

sql statement: alter database datafile 4 online

sql statement: alter database datafile 3 online

sql statement: alter database datafile 5 online

Starting recover at 05-OCT-18

using channel ORA\_AUX\_DISK\_1

starting media recovery

archived log for thread 1 with sequence 1 is already on disk as file /home/oracle/arch4/1\_1\_988482062.arc

archived log for thread 1 with sequence 2 is already on disk as file /home/oracle/arch4/1\_2\_988482062.arc

archived log file name=/home/oracle/arch4/1\_1\_988482062.arc thread=1 sequence=1

archived log file name=/home/oracle/arch4/1\_2\_988482062.arc thread=1 sequence=2

media recovery complete, elapsed time: 00:00:01

Finished recover at 05-OCT-18

database opened

ok，到这里，auxiliary database打开了，并且已经拥有了我们需要的时间点的数据状态的test表空间。接下来就要进行表空间传输了。

contents of Memory Script:

{

# make read only the tablespace that will be exported

sql clone 'alter tablespace "TEST" read only';

# create directory for datapump import

sql "create or replace directory TSPITR\_DIROBJ\_DPDIR as ''

/home/oracle/tspitr''";

# create directory for datapump export

sql clone "create or replace directory TSPITR\_DIROBJ\_DPDIR as ''

/home/oracle/tspitr''";

}

我们看这个脚本，就是用我们之前指定的auxiliary dest在target database和auxiliary database上分别创建directory，并且，我们看到，是先把auxiliary database的test表空间置为read only。

executing Memory Script

sql statement: alter tablespace "TEST" read only

sql statement: create or replace directory TSPITR\_DIROBJ\_DPDIR as ''/home/oracle/tspitr''

sql statement: create or replace directory TSPITR\_DIROBJ\_DPDIR as ''/home/oracle/tspitr''

Performing export of metadata...

EXPDP> Starting "SYS"."TSPITR\_EXP\_cFBF\_csiC":

EXPDP> Processing object type TRANSPORTABLE\_EXPORT/PLUGTS\_BLK

EXPDP> Processing object type TRANSPORTABLE\_EXPORT/POST\_INSTANCE/PLUGTS\_BLK

EXPDP> Master table "SYS"."TSPITR\_EXP\_cFBF\_csiC" successfully loaded/unloaded

EXPDP> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXPDP> Dump file set for SYS.TSPITR\_EXP\_cFBF\_csiC is:

EXPDP> /home/oracle/tspitr/tspitr\_cFBF\_86204.dmp

EXPDP> \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

EXPDP> Datafiles required for transportable tablespace TEST:

EXPDP> /u01/app/oracle/oradata/PROD4/test01.dbf

EXPDP> Job "SYS"."TSPITR\_EXP\_cFBF\_csiC" successfully completed at Fri Oct 5 09:23:50 2018 elapsed 0 00:00:17

Export completed

上面就是脚本执行的过程和expdp的过程，我们看到expdp是oracle自己自动完成的。

contents of Memory Script:

{

# shutdown clone before import

shutdown clone abort

# drop target tablespaces before importing them back

sql 'drop tablespace "TEST" including contents keep datafiles cascade constraints';

}

这个脚本就是把auxiliary database关掉，并且将target database的test表空间彻底删掉。

executing Memory Script

Oracle instance shut down

sql statement: drop tablespace "TEST" including contents keep datafiles cascade constraints

Performing import of metadata...

IMPDP> Master table "SYS"."TSPITR\_IMP\_cFBF\_Cqhf" successfully loaded/unloaded

IMPDP> Starting "SYS"."TSPITR\_IMP\_cFBF\_Cqhf":

IMPDP> Processing object type TRANSPORTABLE\_EXPORT/PLUGTS\_BLK

IMPDP> Processing object type TRANSPORTABLE\_EXPORT/POST\_INSTANCE/PLUGTS\_BLK

IMPDP> Job "SYS"."TSPITR\_IMP\_cFBF\_Cqhf" successfully completed at Fri Oct 5 09:24:00 2018 elapsed 0 00:00:02

Import completed

我们看到，上面依然是执行了脚本，并且执行impdp。

contents of Memory Script:

{

# make read write and offline the imported tablespaces

sql 'alter tablespace "TEST" read write';

sql 'alter tablespace "TEST" offline';

# enable autobackups after TSPITR is finished

sql 'begin dbms\_backup\_restore.AutoBackupFlag(TRUE); end;';

}

这个脚本就是很简单的把test表空间在target database上先online测试一下，然后在offline。

executing Memory Script

sql statement: alter tablespace "TEST" read write

sql statement: alter tablespace "TEST" offline

sql statement: begin dbms\_backup\_restore.AutoBackupFlag(TRUE); end;

Removing automatic instance

Automatic instance removed

auxiliary instance file /home/oracle/tspitr/PROD4/datafile/o1\_mf\_temp\_fvfh8m7v\_.tmp deleted

auxiliary instance file /home/oracle/tspitr/PROD4/onlinelog/o1\_mf\_3\_fvfh8jn2\_.log deleted

auxiliary instance file /home/oracle/tspitr/PROD4/onlinelog/o1\_mf\_2\_fvfh8jmo\_.log deleted

auxiliary instance file /home/oracle/tspitr/PROD4/onlinelog/o1\_mf\_1\_fvfh8jm0\_.log deleted

auxiliary instance file /home/oracle/tspitr/PROD4/datafile/o1\_mf\_sysaux\_fvfh86hd\_.dbf deleted

auxiliary instance file /home/oracle/tspitr/PROD4/datafile/o1\_mf\_undotbs1\_fvfh86hp\_.dbf deleted

auxiliary instance file /home/oracle/tspitr/PROD4/datafile/o1\_mf\_system\_fvfh86gv\_.dbf deleted

auxiliary instance file /home/oracle/tspitr/PROD4/controlfile/o1\_mf\_fvfh8052\_.ctl deleted

auxiliary instance file tspitr\_cFBF\_86204.dmp deleted

Finished recover at 05-OCT-18

我们看到，在执行了最后的脚本后，rman自动把之前所有auxiliary database的文件全部删除了。

如下是alert日志中的内容：



然后我们到数据库里看一下数据文件的状态：

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

FILE# NAME STATUS

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

1 /u01/app/oracle/oradata/PROD4/system01.dbf SYSTEM

3 /u01/app/oracle/oradata/PROD4/sysaux01.dbf ONLINE

4 /u01/app/oracle/oradata/PROD4/undotbs01.dbf ONLINE

5 /u01/app/oracle/oradata/PROD4/test01.dbf OFFLINE

7 /u01/app/oracle/oradata/PROD4/users01.dbf ONLINE

现在我们把数据文件online，来验证数据：

SQL> alter tablespace test online;

Tablespace altered.

SQL> conn test/test

Connected.

SQL> select \* from test;

select \* from test

\*

ERROR at line 1:

ORA-00942: table or view does not exist

我们看到，只是表没了，但是用户还在，注意用户还在！我们创建用户的操作，一定是在sequence 3中的，但是用户还在！

原因很简单，创建用户的行为，并不是对test表空间操作的行为，而我们仅仅是对test表空间做了不完全恢复。

所以这就是为什么不能单独对system表空间做不完全恢复！

我们还可以从视图TS\_PITR\_OBJECTS\_TO\_BE\_DROPPED检查我们本次tspitr丢失的对象：

SQL> SELECT OWNER, NAME, TABLESPACE\_NAME,

2 TO\_CHAR(CREATION\_TIME, 'YYYY-MM-DD:HH24:MI:SS')

3 FROM TS\_PITR\_OBJECTS\_TO\_BE\_DROPPED

4 WHERE TABLESPACE\_NAME IN ('TEST')

5 ORDER BY TABLESPACE\_NAME, CREATION\_TIME;

OWNER NAME TABLESPACE\_NAME TO\_CHAR(CREATION\_TI

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

TEST TEST TEST 2018-10-05:12:17:52

#### Overriding Defaults for RMAN TSPITR（半自动）

半自动就是：

1. 我们自己用set newname把auxiliary dest指定了；
2. 用set newname把recovery dest指定了；
3. 把参数文件自己配置好。

如下是官文上的示例。

RUN

{

SET NEWNAME FOR DATAFILE '?/oradata/prod/system01.dbf'

TO '/disk1/auxdest/system01.dbf';

SET NEWNAME FOR DATAFILE '?/oradata/prod/sysaux01.dbf'

TO '/disk1/auxdest/sysaux01.dbf';

SET NEWNAME FOR DATAFILE '?/oradata/prod/undotbs01.dbf'

TO '/disk1/auxdest/undotbs01.dbf';

RECOVER TABLESPACE users, tools

UNTIL LOGSEQ 1300 THREAD 1

AUXILIARY DESTINATION '/disk1/auxdest';

}

上面就是用set newname把auxiliary set指定到指定路径。

RUN

{

SET NEWNAME FOR DATAFILE 'ORACLE\_HOME/oradata/trgt/users01.dbf'

TO '/newfs/users01.dbf';

...other SET NEWNAME commands...

RECOVER TABLESPACE users, tools UNTIL SEQUENCE 1300 THREAD 1;

}

标黄的，那就是用set newname把recovery set指定到指定路径。

#### 纯手工

就是手动把前面的脚本自己执行一遍。自己创建auxiliary database是必须的。

官文示例如下：

准备参数文件：

DB\_NAME=PROD

DB\_UNIQUE\_NAME=tspitr\_PROD

CONTROL\_FILES=/bigtmp/tspitr\_cntrl.dbf

DB\_CREATE\_FILE\_DEST=/bigtmp

COMPATIBLE=11.0.0

BLOCK\_SIZE=8192

REMOTE\_LOGIN\_PASSWORD=exclusive

创建service name：pitprod

启动auxiliary database到nomount：

SQL> STARTUP NOMOUNT PFILE=/bigtmp/init\_tspitr\_prod.ora

用rman连接：

rman target / auxiliary '"sbu@pitprod AS SYSBACKUP"'

执行脚本：

RUN

{

# Specify NEWNAME for recovery set data files

SET NEWNAME FOR TABLESPACE clients

TO '?/oradata/prod/rec/%b';

# Specify NEWNAMES for some auxiliary set

# data files that have a valid image copy to avoid restores:

SET NEWNAME FOR DATAFILE '?/oradata/prod/system01.dbf'

TO '/backups/prod/system01\_monday\_noon.dbf';

SET NEWNAME FOR DATAFILE '?/oradata/prod/system02.dbf'

TO '/backups/prod/system02\_monday\_noon.dbf';

SET NEWNAME FOR DATAFILE '?/oradata/prod/sysaux01.dbf'

TO '/backups/prod/sysaux01\_monday\_noon.dbf';

SET NEWNAME FOR DATAFILE '?/oradata/prod/undo01.dbf'

TO '/backups/prod/undo01\_monday\_noon.dbf';

# Specify the types of channels to use

ALLOCATE AUXILIARY CHANNEL c1 DEVICE TYPE DISK;

ALLOCATE AUXILIARY CHANNEL t1 DEVICE TYPE sbt;

# Recover the clients tablespace to 24 hours ago:

RECOVER TABLESPACE clients UNTIL TIME 'sysdate-1';

}

## Recovering Tables and Table Partitions from RMAN Backups

### 总概

跟tspitr原理一样，也是利用auxiliary database，把表恢复出来，然后再用dump导出导入。

#### 目的

说白了就是在不影响其他表对象的正常使用的情况下，把表还原到一个过去的时间点。

那么flashback table有俩个局限：一个就是表结构不能变化，比如有truncate这样的操作就不行，二个是对应的undo能找的到，所以要求的时间点太久远也不行。

那么该技术就没有这些局限。因为它与flashback table基于的原理不一样，该技术同tspitr原理一样。

#### 前提条件

必须有SYSTEM, SYSAUX, SYSEXT（如果有的话）,undo表空间的full backup。

当然要恢复的表或者表分区所在的表空间的full backup肯定也是必须的。

Target database需要满足如下条件：

1. read write
2. 归档模式
3. 想要恢复表分区，compatible必须是11.1以上

执行命令的时候，必须提供如下三个信息给rman：

1. 要恢复的表名或者分区名
2. Target time
3. 是否让rman自动执行impdp操作

第三条如果不告诉rman的话，rman默认自己会把恢复出来的表导入进target database，如果告诉rman不需要自动导入的话（NOTABLEIMPORT），那么就需要自己手动把oracle生成的dump文件导入到target database中。

#### 步骤

RMAN会执行如下如下步骤：

1. 选择合适的备份集
2. 创建auxiliary database
3. 创建一个export dump file
4. （Optional）用impdp导入到target database中
5. （Optional）重命名恢复后的表或者分区

#### 参数

Auxiliary Database Files的路径：

1. auxiliary destination
2. set newname for：如果要使用set newname，那就必须把所有需要的数据文件都set newname，漏一个就会导致恢复失败

expdp的文件名和位置：

1. DUMP FILE：指定expdp导出的文件的名字，如果路径下已存在同名文件，则恢复失败，如果未指定，那么系统默认取名为：tspitr\_SID-of-clone\_n.dmp，其中sid是auxiliary的sid，n是随机数
2. DATAPUMP DESTINATION：指定expdp导出文件的路径，注意，指定该参数为绝对路径，而不是一个directory，如果未指定该参数，则使用auxiliary destination，如果未指定auxiliary destination，则恢复到$ORACLE\_HOME/dbs下

是否自动进行impdp：如果不需要rman自动进行，则指定：NOTABLEIMPORT

重命名表或分区名字：

1. REMAP TABLESPACE：将恢复的表或者表分区重新导入到一个新的表空间
2. REMAP TABLE：重命名子句，对于表分区也是一样，因为每一个表分区会被分别作为一个独立的表导入target database中，如果恢复表分区时，没有指定remap table，则使用的默认表名为：表分区所属表名+表分区原名，即tablename\_partitionname。如果有重名，RMAN就会再补个\_1，如果还重，就是\_2，以此类推。

#### 限制

该技术有如下限制：

1. sys用户的表不能恢复
2. system和sysaux表空间上的表不能恢复
3. 11.1以后才成恢复表分区
4. 备库的表不能恢复
5. 有not null约束的表，恢复的时候不能用remap

### 实验

#### 官文示例

恢复pdb下的表并重命名：

RECOVER TABLE HR.PDB\_EMP OF PLUGGABLE DATABASE HR\_PDB

UNTIL TIME 'SYSDATE-4'

AUXILIARY DESTINATION '/tmp/backups'

REMAP TABLE 'HR'.'PDB\_EMP':'EMP\_RECVR'

恢复俩个表，只产生dump文件，不导入：

RECOVER TABLE SCOTT.EMP, SCOTT.DEPT

UNTIL TIME 'SYSDATE-1'

AUXILIARY DESTINATION '/tmp/oracle/recover'

DATAPUMP DESTINATION '/tmp/recover/dumpfiles'

DUMP FILE 'emp\_dept\_exp\_dump.dat'

NOTABLEIMPORT;

恢复俩个表分区，并且重命名，并且重新放到新的表空间上：

RECOVER TABLE SH.SALES:SALES\_1998, SH.SALES:SALES\_1999

UNTIL SEQUENCE 354

AUXILIARY DESTINATION '/tmp/oracle/recover'

REMAP TABLE 'SH'.'SALES':'SALES\_1998':'HISTORIC\_SALES\_1998',

'SH'.'SALES':'SALES\_1999':'HISTORIC\_SALES\_1999'

REMAP TABLESPACE 'SALES\_TS':'SALES\_PRE\_2000\_TS';

#### 自己实验

我们就演示一个pdb下test表基于scn的恢复，让它导入，但给它重命名，放到新的表空间上。

Rman做备份，做一个pdbprod1的全备和root的全备就够了：

[oracle@host01 flash]$ rman target /

Recovery Manager: Release 12.2.0.1.0 - Production on Fri Oct 5 16:50:07 2018

Copyright (c) 1982, 2017, Oracle and/or its affiliates. All rights reserved.

connected to target database: PRODCDB (DBID=2962887328)

RMAN> backup pluggable database pdbprod1;

RMAN> backup database root;

准备实验表：

SQL> create table test(id number);

Table created.

SQL> insert into test select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> insert into test select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> insert into test select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from test;

ID

----------

7204756

7204770

7204783

SQL> select owner,table\_name,tablespace\_name from dba\_tables where owner='TEST';

OWNER TABLE\_NAME TABLESPACE\_NAME

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

TEST TEST TEST

然后我们来把数据恢复到scn 7204771之前：

RECOVER TABLE 'TEST'.'TEST' OF PLUGGABLE DATABASE PDBPROD1

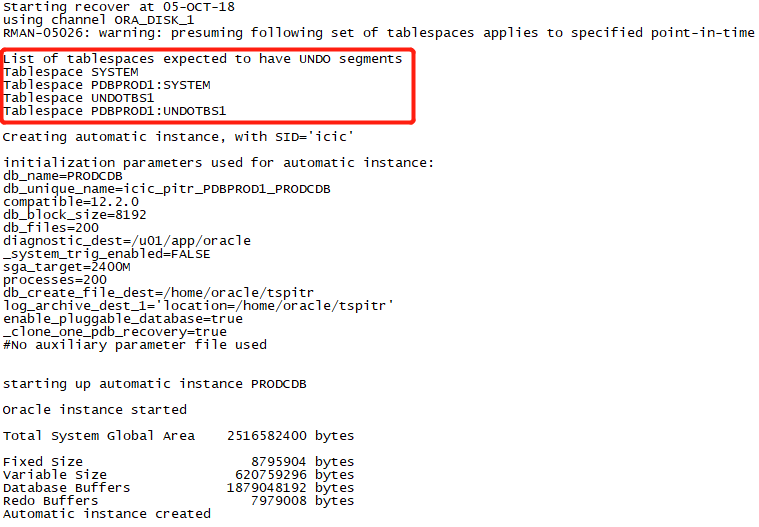
UNTIL SCN 7204771

AUXILIARY DESTINATION '/home/oracle/tspitr'

REMAP TABLE 'TEST'.'TEST':'TEST\_HIS'

REMAP TABLESPACE 'TEST':'USERS';

这里步骤只贴截图了，跟tspitr大同小异：

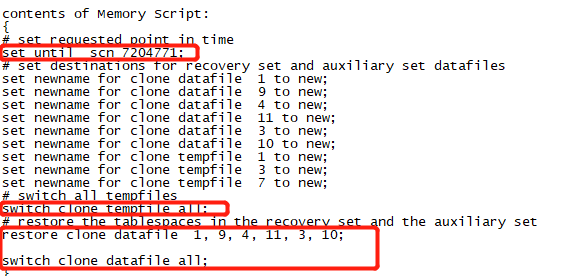


注意，一定要有root的备份，因为auxiliary database也一定是个CDB。这一步就是在做auxiliary database的创建。

然后就是把auxiliary database启动到mount：



再然后，就是恢复数据文件和临时表空间的文件：



这些文件就是root+pdbprod1的system、sysaux、undo、temp和要恢复表所在的表空间：

SQL> select file#,name from v$datafile where file# in(1,9,4,11,3,10);

FILE# NAME

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

1 /u01/app/oracle/oradata/PRODCDB/system01.dbf

3 /u01/app/oracle/oradata/PRODCDB/sysaux01.dbf

4 /u01/app/oracle/oradata/PRODCDB/undotbs01.dbf

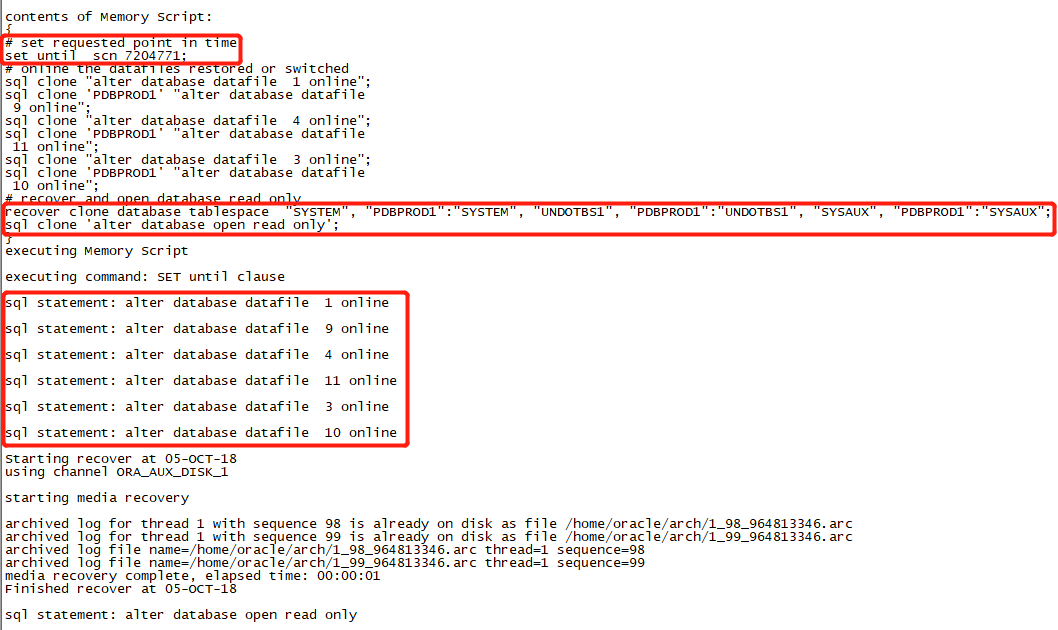
9 /u01/app/oracle/oradata/PRODCDB/PDBPROD1/system01.dbf

10 /u01/app/oracle/oradata/PRODCDB/PDBPROD1/sysaux01.dbf

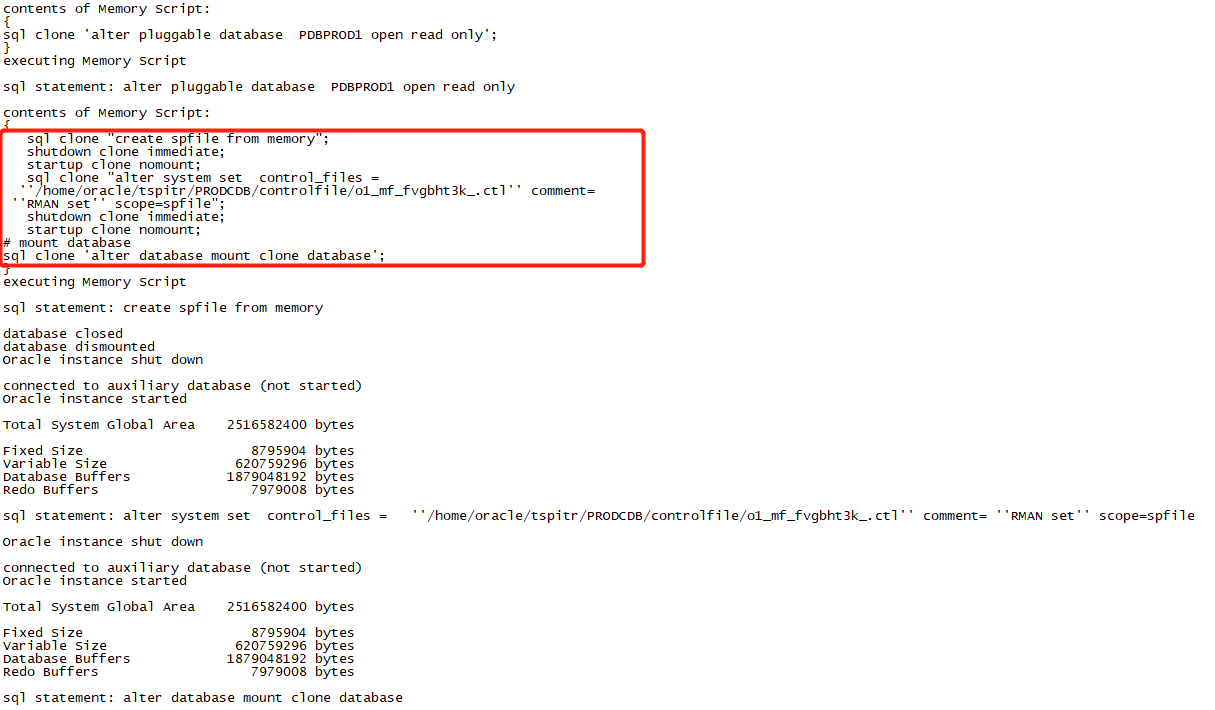
11 /u01/app/oracle/oradata/PRODCDB/PDBPROD1/undotbs01.dbf

6 rows selected.

然后就是要先把auxiliary set recover到target time，并把数据库启动到read only模式：

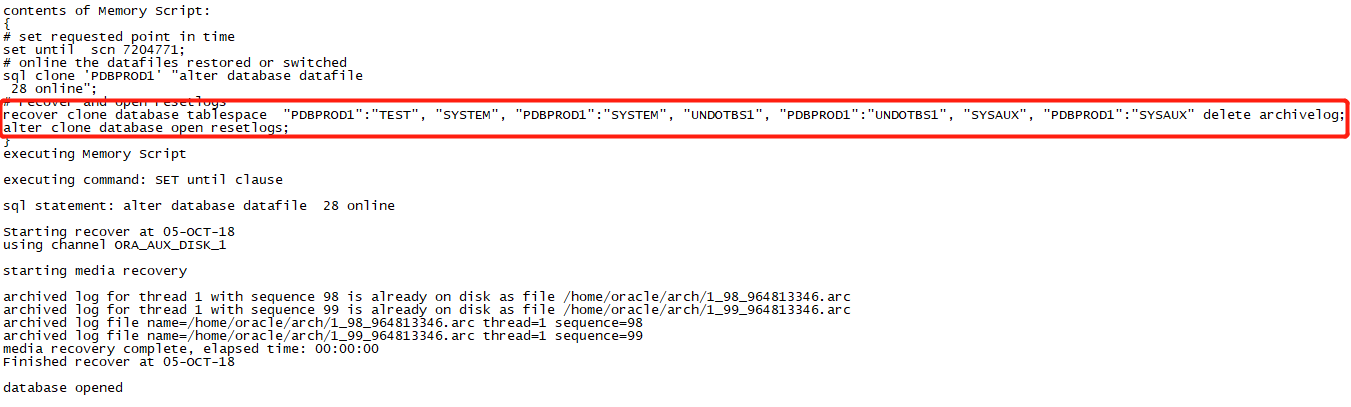


接着为auxiliary database创建spfile并且重启到mount状态：



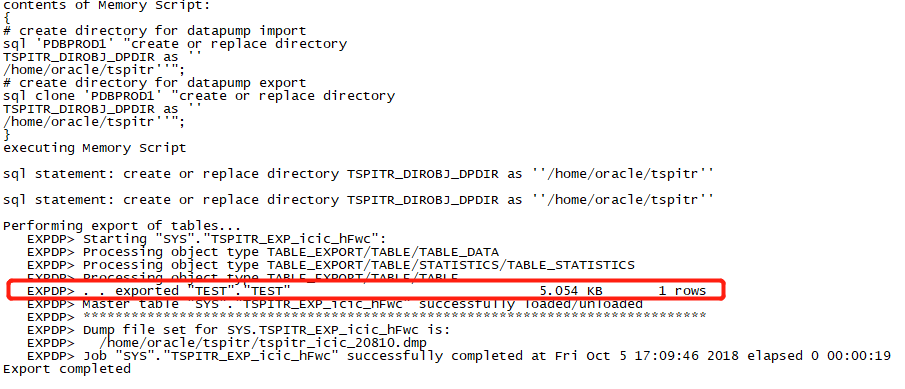
然后，接下来才是recovery set的restore和recover，并且把auxiliary database使用resetlogs的方式打开：



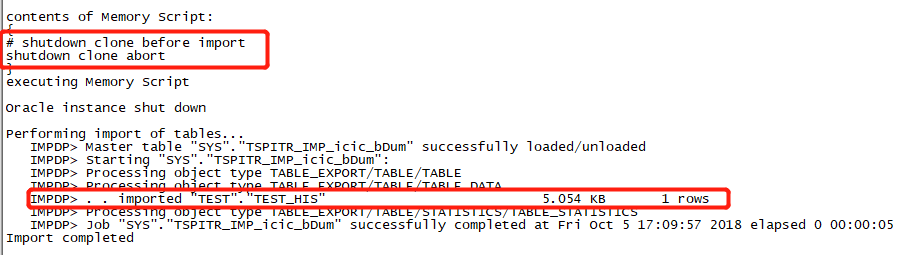


datafile 28正是我们的test表空间，也就是本次恢复的recovery set。

至此，auxiliary database彻底恢复完成，接下来就该导出我们要恢复的表了：



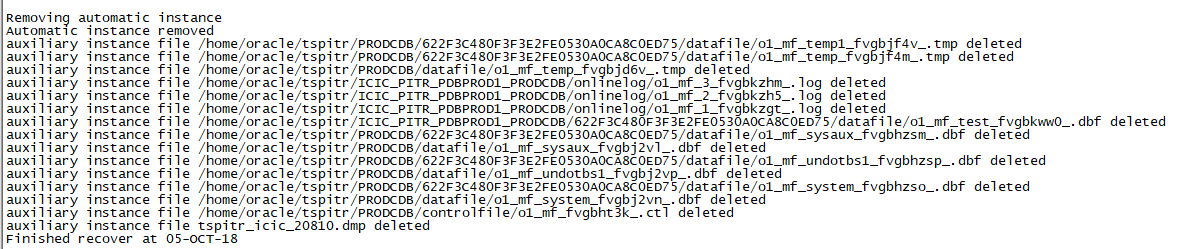
我们看到，表中只有一行数据，因为我们没有指定NOTABLEIMPORT，所以rman接着会自行导入该表到target database中：



我们看到，还是跟tspitr一样，导入之前先把auxiliary database关闭掉，然后导入表。

我们看到表名确实被remap了。

最后就是把auxiliary database相关的文件都删掉：



我们看到最后一行，dmp文件也被删了。如果impdp导入失败的话，该文件才不会被删。

然后我们到数据库中验证一下：

[oracle@host01 tspitr]$ sqlplus test/test@pdbprod1

SQL\*Plus: Release 12.2.0.1.0 Production on Fri Oct 5 17:59:57 2018

Copyright (c) 1982, 2016, Oracle. All rights reserved.

Last Successful login time: Fri Oct 05 2018 16:55:56 +08:00

Connected to:

Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> select \* from test;

ID

----------

7204756

7204770

7204783

SQL> select \* from test\_his;

ID

----------

7204756

SQL> select owner,table\_name,tablespace\_name from dba\_tables where owner='TEST';

OWNER TABLE\_NAME TABLESPACE\_NAME

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

TEST TEST TEST

TEST TEST\_HIS USERS

Ok，实验成功。

我们看到，虽然我们指定unitl到的scn是第二行数据+1，但是仍然只恢复了第一个scn的事务结果。说明即使依据scn进行recover，依然是要遵循一致性的原则，如果我们的scn再稍微指定的大一点点，比如不是+1而是+6，可能就把第二行数据也恢复回来了。

## 对RMAN的监控

### RMAN基本工作原理

不论backup还是restore，一共分3个阶段：read、copy、write。只是backup是从datafile到backupset，而restore是从backupset到datafile。

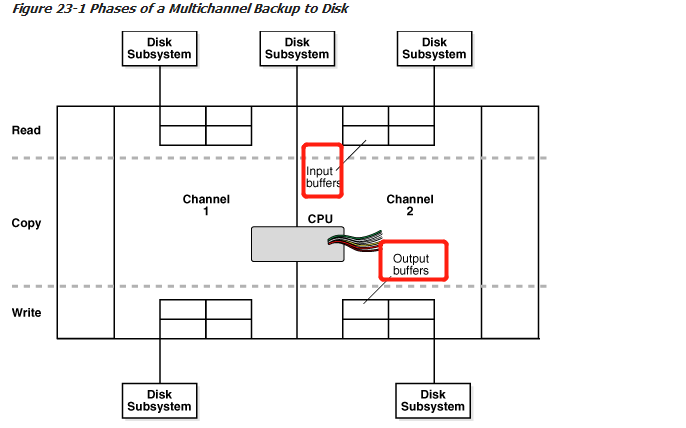
我们知道一个channel就是一个bytes stream，所以我们以一个channel为单位，简述3个phase。

Read phase：一个channel从disk读blocks到input buffer。

Copy phase：一个channel从input buffer将blocks copy到output buffer，并进行一些额外的处理，这些处理包括：validation、compression、encryption。

Write phase：一个channel从output buffer将block写到存储上。

这个流程可以用下图表示：

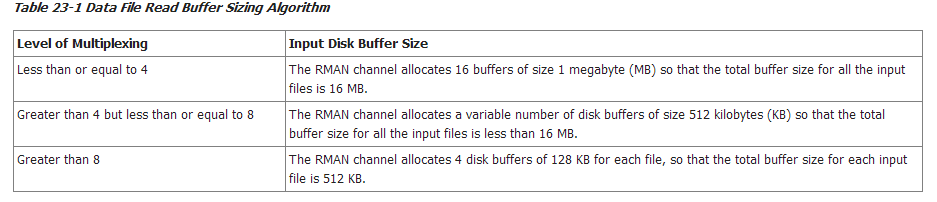


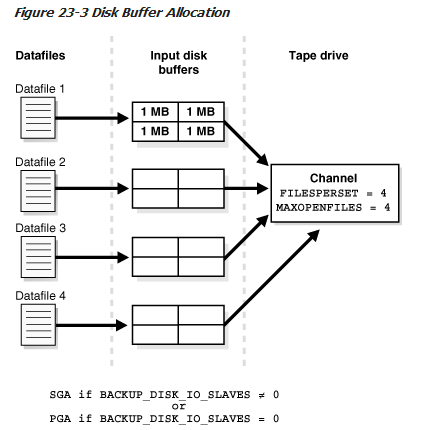
然后我们稍微详细的说一下read phase。

#### Read phase

##### Inupt buffer的分配

根据channel的分配数量，对于input buffer有不同的分配算法：





##### 异步IO

同步IO就是，一个server process就等一个读或写操作彻底完成，那么异步IO就是，等读写操作完成的这个时间段，server process还可以做别的事情，而不是干等。

有的操作系统本身就能实现异步IO，有的不行。对于不能实现异步IO的操作系统来说，oracle有办法。

Disk I/O slaves就是用来解决这个问题的，想要使用该功能，需要dbwr\_io\_slaves非0。

SQL> show parameter dbwr\_io\_slaves

NAME TYPE VALUE

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

dbwr\_io\_slaves integer 0

我们看到，默认是0。

如果该参数非0，那么RMAN就会自动分配4个backup disk I/O slaves。

那么当要为I/O slaves获得shared buffers时，数据库就要做如下工作：

如果large\_pool\_size不为0，那就会从large pool里分配buffer，否则就要从shared pool中分配buffer，如果shared pool也没有足够的memory，那么就从PGA中获取，并且会写入alert日志。

SQL> show parameter large\_pool

NAME TYPE VALUE

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

large\_pool\_size big integer 0

我们知道，large pool的主要作用是为shared server、parallel query和RMAN I/O slave buffers分配内存，说白了，就是为了分配一块较大的连续的内存。

如果没有large pool，那么这个需求就需要shared pool来完成，这就加重了内存的争用。同时，large pool中没有LRU，也就是说，数据库不会主动去把large pool中的buffers age out。

所以，配置large pool是很有必要的。

##### Channel的RATE参数

就是用来限流的。限制rman一个channel在I/O时获得的带宽，单位时bytes/s。

run{

allocate channel ch01 device type disk RATE 1000k;

allocate channel ch02 device type disk RATE 1000k;

backup datafile 1;

}

### 监控

我们就使用如下备份脚本进行实验：

run{

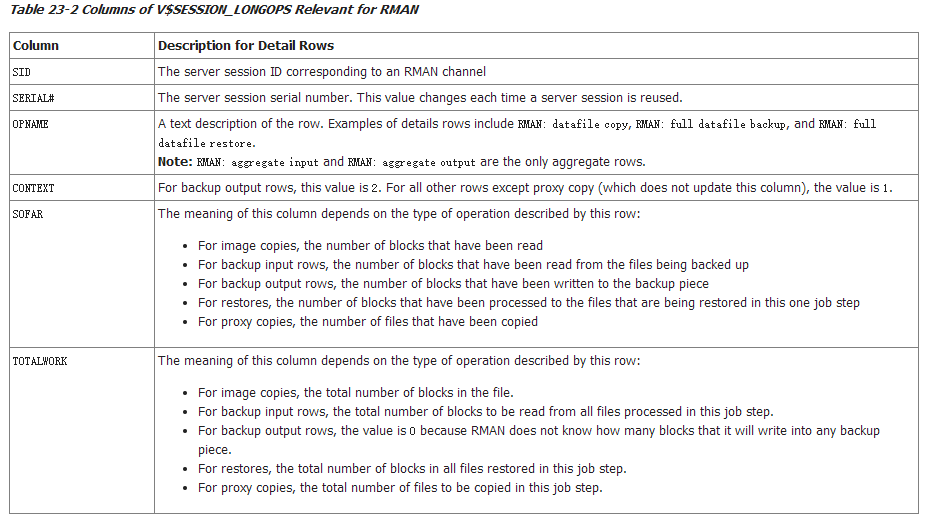
allocate channel ch01 device type disk RATE 1000k;

allocate channel ch02 device type disk RATE 1000k;

backup datafile 1;

}

#### 脚本一（V$SESSION\_LONGOPS）



SELECT SID,

SERIAL#,OPNAME,

CONTEXT,

SOFAR,

TOTALWORK,

ROUND(SOFAR / TOTALWORK \* 100, 2) "%\_COMPLETE"

FROM V$SESSION\_LONGOPS

WHERE OPNAME LIKE 'RMAN%'

AND OPNAME NOT LIKE '%aggregate%'

AND TOTALWORK != 0

AND SOFAR <> TOTALWORK;

SID SERIAL# OPNAME CONTEXT SOFAR TOTALWORK %\_COMPLETE

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

747 57119 RMAN: full datafile backup 1 62 652 9.51

622 14731 RMAN: full datafile backup 1 126 107520 .12

622 14731 RMAN: full datafile backup 1 14334 107520 13.33

#### 脚本二（V$BACKUP\_ASYNC\_IO和V$BACKUP\_SYNC\_IO;）

SELECT LONG\_WAITS/IO\_COUNT, FILENAME

FROM V$BACKUP\_ASYNC\_IO

WHERE LONG\_WAITS/IO\_COUNT > 0

ORDER BY LONG\_WAITS/IO\_COUNT DESC;

LONG\_WAITS/IO\_COUNT FILENAME

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

.248333333 /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_07/o1\_mf\_nnndf\_TAG20181007T164019\_fvmkmmdk\_.bkp

.181818182 /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/backupset/2018\_10\_07/o1\_mf\_ncnnf\_TAG20181007T164019\_fvmkmnfc\_.bkp

.062049062

.062049062 /u01/app/oracle/oradata/PROD4/system01.dbf

#### 脚本三（查找channel的SPID）

如下脚本可以查询到每个channel对应的spid。

COLUMN CLIENT\_INFO FORMAT a30

COLUMN SID FORMAT 999

COLUMN SPID FORMAT 999

set lines 200

COLUMN EVENT FORMAT a40

COLUMN SEC\_WAIT FORMAT 999

COLUMN STATE FORMAT a10

COLUMN CLIENT\_INFO FORMAT a30

SELECT s.sid,

p.SPID,

s.EVENT,

s.SECONDS\_IN\_WAIT AS SEC\_WAIT,

sw.STATE,

s.CLIENT\_INFO

FROM V$SESSION\_WAIT sw, V$SESSION s, V$PROCESS p

WHERE s.SID = sw.SID

AND s.PADDR = p.ADDR

AND CLIENT\_INFO LIKE '%rman%';

SID SPID EVENT SEC\_WAIT STATE CLIENT\_INFO

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

748 724 Backup Restore Throttle sleep 1 WAITING rman channel=ch01

869 725 SQL\*Net message from client 291 WAITING rman channel=ch02

我们来看一下进程：

[oracle@host01 ~]$ ps -ef|grep 724

oracle 724 662 0 10:01 ? 00:00:00 oraclePROD4 (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))

[oracle@host01 ~]$ ps -ef|grep 725

oracle 725 662 0 10:01 ? 00:00:00 oraclePROD4 (DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))

我们在开始备份的时候，也可以看到每个channel分配的sid，这个在configuration的channel部分也讲述过：

RMAN> run{

2> allocate channel ch01 device type disk RATE 1000k;

3> allocate channel ch02 device type disk RATE 1000k;

4> backup datafile 1;

5> }

using target database control file instead of recovery catalog

allocated channel: ch01

channel ch01: SID=748 device type=DISK

allocated channel: ch02

channel ch02: SID=869 device type=DISK

当多个rman任务同时进行的时候，为了区分那个channel对应的是哪个rman任务，可以使用SET COMMAND ID TO来做标记：

run{

allocate channel ch01 device type disk RATE 1000k;

allocate channel ch02 device type disk RATE 1000k;

SET COMMAND ID TO 'sess1';

backup datafile 1;

}

如下是查询结果：

SELECT s.sid,

p.SPID,

s.EVENT,

s.SECONDS\_IN\_WAIT AS SEC\_WAIT,

sw.STATE,

s.CLIENT\_INFO

FROM V$SESSION\_WAIT sw, V$SESSION s, V$PROCESS p

WHERE s.SID = sw.SID

AND s.PADDR = p.ADDR

AND CLIENT\_INFO LIKE '%rman%';

SID SPID EVENT SEC\_WAIT STATE CLIENT\_INFO

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

748 724 Backup Restore Throttle sleep 1 WAITING id=sess1,rman channel=ch01

869 725 SQL\*Net message from client 291 WAITING id=sess1,rman channel=ch02

#### 终止rman

使用脚本三查出来的spid，然后直接kill -9：

[oracle@host01 ~]$ kill -9 724

[oracle@host01 ~]$ kill -9 725

效果如下：

RMAN-03009: failure of backup command on ch01 channel at 10/08/2018 10:27:36

RMAN-10038: database session for channel ch01 terminated unexpectedly

channel ch01 disabled, job failed on it will be run on another channel

RMAN-00571: ===========================================================

RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============

RMAN-00571: ===========================================================

RMAN-03009: failure of backup command on ch02 channel at 10/08/2018 10:27:36

RMAN-10038: database session for channel ch02 terminated unexpectedly

# Flashback Database（Physical）

## 基础概念

Flashback database可以还原窗口期内的逻辑修改，比如说用户对数据的误操作啊，但一定要保证数据文件的完好无损（intact）。这个功能类似于database point-in-time recovery (DBPITR)，但是flashback database更快，因为flashback不需要restore datafile，并且比dbpitr会应用更少量的archive redo log。

Flashback database的技术一般情况不常用，在数据库升级的时候很常用，升级前先做个retore point，万一升级失败了，再flashback回来。

Restore point跟flashback database并不是一定要一起用的，这俩各有各的用途，可以分开用。值得一提的是guaranteed restore point建立时的scn，再flashback database的时候，是可以保证回到这个scn时的状态的。

Flashback database并不会rewind删除或者收缩表空间的行为。但是可以回滚alter database open restlogs的操作，也就是说，可以把数据库flashback到上一个incarnation。

Flashback database 用的是自己的一套logging mechanism。就是存储再FRA中的flash logs。Flash logs只能放在FRA中，所以要想使用flashback database，就一定要配置FRA。

此外还要配置flashback retention target，这个就是窗口。

SQL> show parameter flashback

NAME TYPE VALUE

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

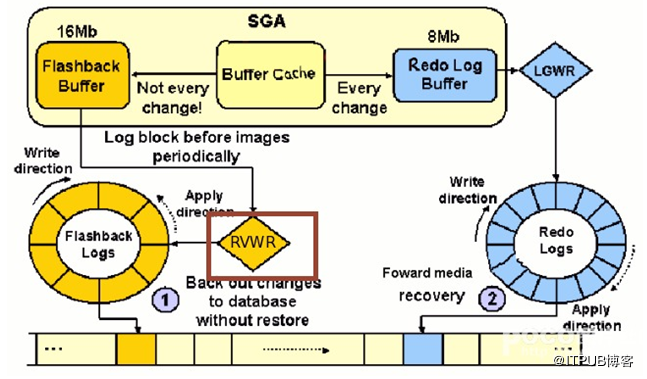
db\_flashback\_retention\_target integer 1440

但是如果建立的是普通的restore point，当FRA空间不足时，会优先删除最早的flash logs，所以flashback retention target对于普通的resotre point就只是一个目标，而不是保证。

如果建立的是guarantee restore point，那么这个窗口就可以保证。

Oracle只会把数据文件中发生了修改的块的image copy到flash logs。

在flashback的时候不仅要应用flash logs，还要应用这个期间所有的redo和归档，所以想要恢复到一个恢复点，这期间的归档绝不能丢。



## Limitations of Flashback Database

闪回数据库有如下限制：

1. flashback database不能修复media failure或者数据文件删除操作。
2. 不能撤销shrink datafile的操作，但是，可以先把做了shrink操作的datafile离线，flashback database其他部分，然后再单独restore和recover离线的datafile。
3. 控制文件不能被restore或者re-create，否则flash logs全都会被discard。
4. 不能有nologging操作，如果再target time期间有nologging操作，flashback database还是可以完成，但是完成之后，发了nologging操作的数据文件会有坏块。

## Restore Points

### Normal Restore Points

创建一个normal restore point后，控制文件会记录restore point name和对应的SCN。当实施flashback或者point-in-time recovery时，可以直接使用restore point name，而不需要指定scn或time。

Restore point可以应用在如下操作：

1. RMAN中执行recover database和flashback database操作
2. sql语句执行flashback database的操作

创建normal restore point就不用记录scn，或者当使用flashback query时也不用去找要查询的点的正确的scn。

Normal restore point是轻量级的，就是在控制文件中做个记录，控制文件中可以记录上千个normal restore point，如果不手动删除normal restore points，controlfile也会自动清除掉它们。所以这类restore point不需要一个持续的维护。所以其相应的db\_flashback\_retention\_target也不会有绝对的保证。

### Guaranteed Restore Points

Guaranteed restore points与normal restore points的区别就是：guaranteed restore point除非显式删除，否则不会自动被从控制文件清除掉。

即使不开flashback log或者flashback window中发生nologging的操作，依然能将数据库闪回到guaranteed restore point的scn。

但是只有当启用了flashback log并且没有nologging操作的时候，可以将database flashback到guaranteed restore point之后的任意一点。

guaranteed restore point这个有一个极大的用途就是，在做危险操作时，比如升级、打补丁、验证某些业务的时候，可以先做一个guaranteed store point。

### Logging for Flashback Database and Restore Points

#### Logging for Flashback Database

##### Enable

打开database flashback的前提条件如下：

1. 归档模式，因为flashback的时候要用归档
2. 配置FRA，因为flashback logs只放在FRA中
3. RAC环境中，FRA必须配置在共享存储上

我们先来看一下是否满足前提条件：

SQL> archive log list

Database log mode Archive Mode

Automatic archival Enabled

Archive destination /home/oracle/arch4

Oldest online log sequence 23

Next log sequence to archive 25

Current log sequence 25

SQL> show parameter recovery

NAME TYPE VALUE

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

db\_recovery\_file\_dest string /u01/app/oracle/fast\_recovery\_area/PROD4

db\_recovery\_file\_dest\_size big integer 8016M

SQL> show parameter flashback

NAME TYPE VALUE

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

db\_flashback\_retention\_target integer 1440

注意，db\_flashback\_retention\_target的值的单位是分钟，默认值是1440分钟，也就是1天。

开始我们并没有打开数据库闪回，FRA下也没有flashback logs：

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

NO

[oracle@host01 PROD4]$ ls

autobackup backupset onlinelog

然后我们打开闪回：

SQL> alter database flashback on;

Database altered.

[oracle@host01 PROD4]$ ls

autobackup backupset flashback onlinelog

[oracle@host01 PROD4]$ cd flashback/

[oracle@host01 flashback]$ ls

o1\_mf\_ft3lq67x\_.flb o1\_mf\_ft3lqb58\_.flb

我们看到出现了flashback log。

我们来看看alert日志：

2018-09-19T12:04:54.230315+08:00

alter database flashback on

Starting background process RVWR

2018-09-19T12:04:54.249405+08:00

RVWR started with pid=79, OS id=129611

2018-09-19T12:04:55.314763+08:00

Allocated 15937344 bytes in shared pool for flashback generation buffer

Flashback Database Enabled at SCN 3308768

Completed: alter database flashback on

2018-09-19T12:11:43.573360+08:00

我们看到，oracle产生了一个叫RVWR的进程，同时在shared pool中分配了flashback generation buffer，同时也记下了SCN。

数据库open的状态下是可以把database flashback打开的，但是有可能会没有获得flashback generation buffer，这种情况下，要么重试，要么就把数据库起到mount，然后打开database flashback。

##### Disable

而当我们关闭database flashback logging，oracle会把RVWR结束掉，并且将flashback logs删除掉。

SQL> alter database flashback off;

Database altered.

我们看alert日志：

2018-09-19T12:18:17.756660+08:00

alter database flashback off

Stopping background process RVWR

2018-09-19T12:18:18.792132+08:00

Deleted Oracle managed file /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3lq67x\_.flb

Deleted Oracle managed file /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3lqb58\_.flb

Flashback Database Disabled

Completed: alter database flashback off

还可以单独关闭某个表空间的flashback，默认如果开启了database flashback，那么所有永久表空间都会开启flashback，如果执行如下sql：

SQL> alter tablespace users flashback off;

Tablespace altered.

那么users表空间的块的修改就不会记录在flashback logs中。

同时切记！在进行flashback之前，一定要将该表空间离线！

SQL> select name,flashback\_on from v$tablespace;

NAME FLA

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

SYSAUX YES

SYSTEM YES

UNDOTBS1 YES

USERS NO

TEMP YES

TEST YES

6 rows selected.

#### 创建和删除Normal还原点

对于，normal restore points，flash logs有如下特点：

1. 只要fra空间富余，那么就会创建flash logs，当然是在alter database flashback on的前提下。
2. 如果flashback logs已经过于陈旧，超过了db\_flashback\_retention\_target，那么这些flashback logs可以被重用
3. 如果数据库必须要创建一个flashback log，但是fra没有额外的空间，那么最旧的flashback log会被重用，所以此时db\_flashback\_retention\_target并不能保证。
4. 如果fra自动删除了obsolete的或者已经备份到磁盘的归档日志，那么相应的在flashback时需要与这些归档配合的flashback log也会被删除。
5. 如果手动删除了归档日志，比如backup … delete input，那么在flashback时，如果需要这些归档，这些归档会自动恢复回来。

创建normal restore point：

SQL> CREATE RESTORE POINT before\_upgrade;

Restore point created.

我们dump controlfile看一下：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NORMAL RESTORE POINT RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(size = 256, compat size = 256, section max = 2108, section in-use = 2,

last-recid= 2, old-recno = 1, last-recno = 2)

(extent = 1, blkno = 199, numrecs = 2108)

Earliest record:

RECID #1 Recno 1 Record timestamp 09/19/18 12:25:35

RESTORE POINT #1:

Freed

Latest record:

RECID #2 Recno 2 Record timestamp 09/19/18 15:09:05

RESTORE POINT #2:

restore point name: BEFORE\_UPGRADE guarantee flag: 0 incarnation: 2next record 0

restore point scn: 0x000000000032dd65 09/19/2018 15:09:05

注意，创建normal restore point，alert日志都懒得记录，因为这个并不需要严格保障，而且能无限创建。

然后删除restore point：

SQL> drop RESTORE POINT before\_upgrade;

Restore point dropped.

对于normal restore point可以不用显式删除，控制文件会根据俩个判断原则自动删除normal restore points：

1. 最多只保留最近创建的2048个normal restore points
2. 超过CONTROL\_FILE\_RECORD\_KEEP\_TIME的normal restore points都会被删除。

#### 创建和删除Guaranteed还原点

根据database flashback的开启状态，可分为如下俩种情况：

如果在开启了flashback database logging的情况下，同normal唯一的区别就是，如果归档和flashback log还要被用于闪回操作，那么都不会被FRA自动删除，所以，该情况下，一定要监控好FRA的空间大小。很有可能被撑满，而导致数据库hang住。

在这种情况下，可以将database flashback至现在到guaranteed restore point中间的任何一个时间点的数据状态。

如果没开启flashback database logging，那么依然可以创建guaranteed restore point，同时也会生成flashback logs，但是这些flashback logs只记录建立restore point后，block的第一次修改后的前镜像，之后的修改将不再记录，也就是说，在闪回的时候也只能闪回到restore point时数据库的数据状态。

这样设计的优势是，如果在restore point后的大量操作都是针对一部分块的反复修改，那么这样做所产生的flashback logs就不会很大，但又足够将数据库闪回到restore point。

但是如果restore point之后的大量insert或批量insert，那在不开启database flashback的情况，那这样做就不太好了。

我们在不开启flashback database logging的情况下，创建guaranteed restore point：

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

NO

SQL> CREATE RESTORE POINT before\_upgrade GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

RESTORE POINT ONLY

[oracle@host01 flashback]$ ls

o1\_mf\_ft3v2j85\_.flb o1\_mf\_ft3v2lz7\_.flb

我们看到v$database的flashback\_on列变为了RESTORE POINT ONLY，而且FRA中生成了俩个flashback logs，所以并不是一定要打开database flashback才会产生flashback logs。

同时我们来看告警日志：

2018-09-19T14:27:28.247920+08:00

Starting background process RVWR

2018-09-19T14:27:28.257236+08:00

RVWR started with pid=85, OS id=1658

2018-09-19T14:27:28.586902+08:00

Created guaranteed restore point BEFORE\_UPGRADE

Oracle又一次打开了RVWR进程。

我们dump controlfile来找一下restore point和flashback log的记录。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RESTORE POINT RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(size = 256, compat size = 256, section max = 2048, section in-use = 1,

last-recid= 3, old-recno = 0, last-recno = 0)

(extent = 1, blkno = 166, numrecs = 2048)

RESTORE POINT #1:

restore point name: BEFORE\_UPGRADE guarantee flag: 1 incarnation: 2next record 0

restore point scn: 0x000000000032c779 09/19/2018 14:27:28

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FLASHBACK LOGFILE RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(size = 84, compat size = 84, section max = 2048, section in-use = 2,

last-recid= 0, old-recno = 0, last-recno = 0)

(extent = 1, blkno = 144, numrecs = 2048)

FLASHBACK LOG FILE #1:

name #10: /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3v2j85\_.flb

Thread 1 flashback log links: forward: 1 backward: 1

size: 25600 seq: 1 bsz: 8192 nab: 0xffffffff flg: 0x0 magic: 2 dup: 1

Low scn: 0x000000000032c779 09/19/2018 14:27:28

High scn: 0xffffffffffffffff 09/20/2018 14:27:28

FLASHBACK LOG FILE #2: RESERVED

name #11: /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3v2lz7\_.flb

Thread 1 flashback log links: forward: 2 backward: 2

size: 25600 seq: 1 bsz: 8192 nab: 0x2 flg: 0x1 magic: 1 dup: 1

Low scn: 0x0000000000000000 01/01/1988 00:00:00

High scn: 0x0000000000000000 01/01/1988 00:00:00

我们再来创建俩个guaranteed restore point：

SQL> CREATE RESTORE POINT before\_upgrade01 GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> CREATE RESTORE POINT before\_upgrade02 GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

NO

我们看到flashback\_on列又变回了no，然后dump controlfile：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RESTORE POINT RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(size = 256, compat size = 256, section max = 2048, section in-use = 3,

last-recid= 5, old-recno = 0, last-recno = 0)

(extent = 1, blkno = 166, numrecs = 2048)

RESTORE POINT #1:

restore point name: BEFORE\_UPGRADE guarantee flag: 1 incarnation: 2next record 2

restore point scn: 0x000000000032c779 09/19/2018 14:27:28

RESTORE POINT #2:

restore point name: BEFORE\_UPGRADE01 guarantee flag: 1 incarnation: 2next record 3

restore point scn: 0x000000000032cd67 09/19/2018 14:39:33

RESTORE POINT #3:

restore point name: BEFORE\_UPGRADE02 guarantee flag: 1 incarnation: 2next record 0

restore point scn: 0x000000000032cf9d 09/19/2018 14:44:08

我们看到，第一建的restore point的next record 变成了2（指向下一个restore point），而第二个创建的restore point的next point变为了3，最后一个创建的restore point的next point是0。

Flashback logs这次并没有增加！

然后我们将该restore point删掉，我们先删第二个：

SQL> drop RESTORE POINT before\_upgrade01;

Restore point dropped.

然后dump控制文件：

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RESTORE POINT RECORDS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(size = 256, compat size = 256, section max = 2048, section in-use = 3,

last-recid= 6, old-recno = 0, last-recno = 0)

(extent = 1, blkno = 166, numrecs = 2048)

RESTORE POINT #1:

restore point name: BEFORE\_UPGRADE guarantee flag: 1 incarnation: 2next record 3

restore point scn: 0x000000000032c779 09/19/2018 14:27:28

RESTORE POINT #2:

Freed

RESTORE POINT #3:

restore point name: BEFORE\_UPGRADE02 guarantee flag: 1 incarnation: 2next record 0

restore point scn: 0x000000000032cf9d 09/19/2018 14:44:08

我们看到原来的record 2变成了free，record 1的next record变成了3。

现在我们依次把剩下俩个都删了，同时看alert日志，先删第一个，再删第三个：

SQL> drop RESTORE POINT before\_upgrade;

Restore point dropped.

SQL> drop RESTORE POINT before\_upgrade02;

Restore point dropped.

我们来看alert日志：

2018-09-19T15:03:36.329514+08:00

Drop guaranteed restore point BEFORE\_UPGRADE

Guaranteed restore point BEFORE\_UPGRADE dropped

2018-09-19T15:03:48.514792+08:00

Drop guaranteed restore point BEFORE\_UPGRADE02

Stopping background process RVWR

2018-09-19T15:03:49.551240+08:00

Deleted Oracle managed file /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3v2j85\_.flb

Deleted Oracle managed file /u01/app/oracle/fast\_recovery\_area/PROD4/PROD4/flashback/o1\_mf\_ft3v2lz7\_.flb

Guaranteed restore point BEFORE\_UPGRADE02 dropped

我们可以清晰的看到，再删就没有guaranteed restore point的时候，oracle会先停RVWR，然后删掉flashback logs，最后才完成最后一个restore point的删除。

注意，guaranteed restore point绝不会自动删除，必须手动删除。

#### 查询还原点

有俩种方式查询restore point：

1. RMAN中LIST RESTORE POINT
2. SQLPLUS中V$RESTORE\_POINT

List restore point可以只查看某个还原点，也可以同时查看所有的还原点：

LIST RESTORE POINT restore\_point\_name;

LIST RESTORE POINT ALL;

查询v$restore\_point：

Set lines 200

col name for a20

col time for a40

SELECT NAME, SCN, TIME, DATABASE\_INCARNATION#,GUARANTEE\_FLASHBACK\_DATABASE,STORAGE\_SIZE FROM V$RESTORE\_POINT;

做实验看一下：

SQL> create RESTORE POINT NORMAL\_RS;

Restore point created.

SQL> create RESTORE POINT GUARANTEED\_RS GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> SELECT NAME, SCN, TIME, DATABASE\_INCARNATION#,GUARANTEE\_FLASHBACK\_DATABASE,STORAGE\_SIZE FROM V$RESTORE\_POINT;

NAME SCN TIME DATABASE\_INCARNATION# GUA STORAGE\_SIZE

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

GUARANTEED\_RS 3335435 19-SEP-18 03.24.30.000000000 PM 2 YES 209715200

NORMAL\_RS 3335396 19-SEP-18 03.24.13.000000000 PM 2 NO 0

我们看到storage\_size这一列，对于normal restore points，这里就是0，对于guaranteed restore points，这就是保证能flashback到还原点所用的flashback logs占用的fra中的磁盘大小。

[oracle@host01 flashback]$ ll

total 409616

-rw-r----- 1 oracle oinstall 209723392 Sep 19 15:40 o1\_mf\_ft3yfgmm\_.flb

-rw-r----- 1 oracle oinstall 209723392 Sep 19 15:24 o1\_mf\_ft3yfhoj\_.flb

我们再到rman中查：

RMAN> LIST RESTORE POINT NORMAL\_RS;

SCN RSP Time Type Time Name

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

3335396 19-SEP-18 NORMAL\_RS

RMAN> LIST RESTORE POINT ALL;

SCN RSP Time Type Time Name

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

3335396 19-SEP-18 NORMAL\_RS

3335435 GUARANTEED 19-SEP-18 GUARANTEED\_RS

#### Flashback Writer (RVWR)

我们前面也看到了，如果开启了database flashback，或者创建了guaranteed restore point。那么oracle就会打开该进程，将buffer中的block的改变，写到flashback logs中。

那如果RVWR出现了IO错误，会对实例有什么影响？！主要分俩种情况：

1. 有guaranteed restore point，实例失败
2. 没有guaranteed restore point，实例不受影响：  
   对于primary database，database flashback会自动关闭  
   对于物理或者逻辑备库，RVWR会周期性的尝试IO，这可能会导致managed recovery延迟，所以，如果备库上出现该情况，建议shutdown abort，然后重启到mount，手动关闭database flashback。

## Flashback Database的环境配置和性能监控

FRA所在的存储一定要快！

性能监控：AWR，V$FLASHBACK\_DATABASE\_STAT

如果发现等待事件flashback buf free by RVWR居高，说明写flashback logs的IO性能有问题。

## Flashback Database实验

### Prerequisites of Flashback Database

这里我们再总结下，在当前时间点到想要flashback的时间点期间：

1. 数据文件不能有丢失或者损坏
2. 不能有数据文件被删除，或者shrink datafile，或者修改dbname
3. 不能发生控制文件restore或者重做，一旦发生，所有曾记录的flashback logs就会被弃用。
4. 不能有compatibility change（这个没太懂）

### Performing a Flashback Database Operation

这里我们说的flashback database，是在current database incarnation中的flashback database，也就是说，期间未曾发生过open resetlogs。

#### 第一次实验（失败，users表空间flashback off）

首先我们要把flashback database打开，并创建一个guaranteed restore point。当然如果是创建guaranteed restore point，我们可以不做打开flashback这一步：

SQL> alter database flashback on;

Database altered.

SQL> CREATE RESTORE POINT before\_upgrade GUARANTEE FLASHBACK DATABASE;

Restore point created.

然后我们来检查一下：

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

YES

SQL> SELECT OLDEST\_FLASHBACK\_SCN, OLDEST\_FLASHBACK\_TIME FROM V$FLASHBACK\_DATABASE\_LOG;

OLDEST\_FLASHBACK\_SCN OLDEST\_FL

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

3971547 30-SEP-18

SQL> SELECT NAME, SCN, TIME, DATABASE\_INCARNATION#,GUARANTEE\_FLASHBACK\_DATABASE,STORAGE\_SIZE FROM V$RESTORE\_POINT;

NAME SCN TIME DATABASE\_INCARNATION# GUA STORAGE\_SIZE

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

BEFORE\_UPGRADE 3971794 30-SEP-18 12.40.00.000000000 PM 2 YES 209715200

现在我们来创建实验表并删除实验表：

SQL> create table yx.test\_flashback\_database(id number);

Table created.

SQL> insert into yx.test\_flashback\_database select current\_scn from v$database;

1 row created.

SQL> create index yx.idx\_fd\_id on yx.test\_flashback\_database(id);

Index created.

SQL> insert into yx.test\_flashback\_database select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from yx.test\_flashback\_database;

ID

----------

3974140

3974370

SQL> drop table yx.test\_flashback\_database purge;

Table dropped.

注意，这里我们使用purge删除的表。

现在，我想要还原到索引创建后，第二条数据插入前时的数据库的状态。

也就是还原到scn为3974370-1时数据库的状态。

然后我们关闭数据库并启动到mount状态：

SQL> SHUTDOWN IMMEDIATE;

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup mount;

ORACLE instance started.

Database mounted.

然后我们来flashback database：

SQL> FLASHBACK DATABASE TO SCN 3974369;

FLASHBACK DATABASE TO SCN 3974369

\*

ERROR at line 1:

ORA-38753: Cannot flashback data file 7; no flashback log data.

ORA-01110: data file 7: '/u01/app/oracle/oradata/PROD4/users01.dbf'

SQL> FLASHBACK DATABASE TO RESTORE POINT BEFORE\_UPGRADE;

FLASHBACK DATABASE TO RESTORE POINT BEFORE\_UPGRADE

\*

ERROR at line 1:

ORA-38753: Cannot flashback data file 7; no flashback log data.

ORA-01110: data file 7: '/u01/app/oracle/oradata/PROD4/users01.dbf'

我们看到，说datafile 7没有flashback log data，所以不能进行flashback。当时没想清楚为什么，但是为了能够flashback成功，我们可以把datafile 7先offline：

SQL> select status from v$datafile where file#=7;

STATUS

-------

ONLINE

SQL> alter database datafile 7 offline;

Database altered.

然后我们再来执行：

SQL> FLASHBACK DATABASE TO SCN 3974369;

Flashback complete.

此时我们有俩种选择，将数据库打开并resetlogs正常使用，或者将数据库打开到readonly：

此时我们尝试将数据库直接打开：

SQL> alter database open;

alter database open

\*

ERROR at line 1:

ORA-01589: must use RESETLOGS or NORESETLOGS option for database open

为什么一定要resetlogs呢？前面其实也说过了。因为如果正常打开的话，oracle会应用redo，把数据库实例恢复到最新状态，说白了，就是flashback白做了。

那么我们现在需要把数据库还原到最新的状态，来看一下问题到底出在哪儿了，也就是说，我们准备把本次flashback操作“取消掉”。

那么我们必须要先把datafile 7 online，然后recover database，然后open数据库：

SQL> alter database datafile 7 online;

Database altered.

SQL> recover database;

Media recovery complete.

SQL> alter database open;

Database altered.

然后，突然想起来，上把做实验，把users表空间的flashback给off了，我们来查一下：

SQL> select name,flashback\_on from v$tablespace;

NAME FLA

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

SYSAUX YES

SYSTEM YES

UNDOTBS1 YES

USERS NO

TEMP YES

TEST YES

6 rows selected.

还真是，难怪报错说没有它的flashback data。

那么通过这个实验，我们也看到了这种情况的处理方法，就是把对应表空间offline了。

同时，我们看到，一旦将表空间flashback关闭，无论是重新开启database的flashback，或者是把表空间重新online，都不会自动将表空间的flashback打开。

现在我们来手动打开它：

SQL> alter tablespace users flashback on;

alter tablespace users flashback on

\*

ERROR at line 1:

ORA-01126: database must be mounted in this instance and not open in any instance

那我们启动到mount打开它：

SQL> shutdown immediate;

SQL> startup mount

SQL> alter tablespace users flashback on;

Tablespace altered.

SQL> alter database open;

Database altered.

SQL> select name,flashback\_on from v$tablespace;

NAME FLA

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

SYSAUX YES

SYSTEM YES

UNDOTBS1 YES

USERS YES

TEMP YES

TEST YES

6 rows selected.

下面我们把上面的这次实验再做一次，同时这次我们观察alert日志的记录。

#### 第二次实验（成功）

我们先删了刚才那个还原点，再次创建它：

SQL> drop restore point before\_upgrade;

Restore point dropped.

SQL> CREATE RESTORE POINT before\_upgrade GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

YES

SQL> SELECT OLDEST\_FLASHBACK\_SCN, OLDEST\_FLASHBACK\_TIME FROM V$FLASHBACK\_DATABASE\_LOG;

OLDEST\_FLASHBACK\_SCN OLDEST\_FL

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

3971547 30-SEP-18

SQL> SELECT NAME, SCN, TIME, DATABASE\_INCARNATION#,GUARANTEE\_FLASHBACK\_DATABASE,STORAGE\_SIZE FROM V$RESTORE\_POINT;

NAME SCN TIME DATABASE\_INCARNATION# GUA STORAGE\_SIZE

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

BEFORE\_UPGRADE 3997785 30-SEP-18 02.59.37.000000000 PM 2 YES 209715200

现在我们重新实验：

SQL> create table yx.test\_flashback\_database(id number);

Table created.

SQL> insert into yx.test\_flashback\_database select current\_scn from v$database;

1 row created.

SQL> create index yx.idx\_fd\_id on yx.test\_flashback\_database(id);

Index created.

SQL> insert into yx.test\_flashback\_database select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from yx.test\_flashback\_database;

ID

----------

3999341

3999410

SQL> drop table yx.test\_flashback\_database purge;

Table dropped.

然后我们将数据库flashback到scn 3999409：

SQL> shutdown immediate

SQL> startup mount;

SQL> FLASHBACK DATABASE TO SCN 3999409;

Flashback complete.

告警日志如下：

2018-09-30T15:38:30.209357+08:00

FLASHBACK DATABASE TO SCN 3999409

2018-09-30T15:38:30.231399+08:00

Flashback Restore Start

Flashback Restore Complete

Flashback Media Recovery Start

Started logmerger process

2018-09-30T15:38:30.479573+08:00

Parallel Media Recovery started with 8 slaves

2018-09-30T15:38:30.718347+08:00

Recovery of Online Redo Log: Thread 1 Group 3 Seq 45 Reading mem 0

Mem# 0: /u01/app/oracle/oradata/PROD4/redo03.log

2018-09-30T15:38:30.810346+08:00

Incomplete Recovery applied until change 3999410 time 09/30/2018 15:03:59

2018-09-30T15:38:30.812791+08:00

Flashback Media Recovery Complete

Completed: FLASHBACK DATABASE TO SCN 3999409

我们看到，其实就是做了个不完全恢复。

数据库是先restore到最早的flashback logs中记录的点（restore phase），然后再应用redo（recover phase），做recover database unitl scn。

然后我们打开数据库，一定要使用resetlogs或者read only，这次我们readonly打开：

SQL> select \* from yx.test\_flashback\_database;

ID

----------

3999341

SQL> select index\_name from dba\_indexes where index\_name='IDX\_FD\_ID';

INDEX\_NAME

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

IDX\_FD\_ID

SQL> select current\_scn from v$database;

CURRENT\_SCN

-----------

3999409

我们看到，数据库状态确实还原到了3999409之前，current\_scn也是3999409。

我们通过read only模式检查了数据库，那么现在我们可以把数据库使用resetlogs启动了。：

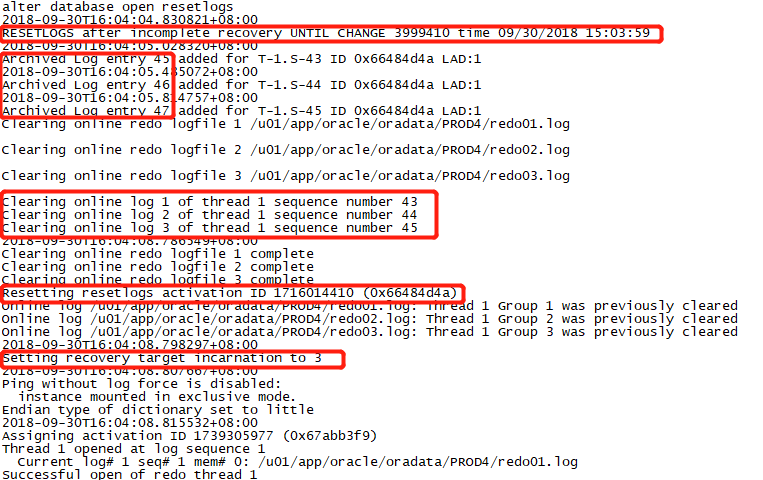
SQL> shutdown immediate

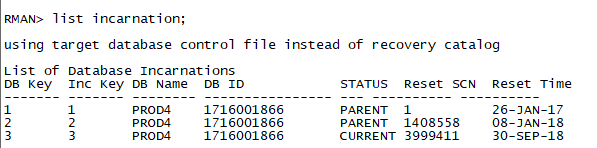
SQL> startup mount;

SQL> alter database open resetlogs;

Database altered.

Alert日志如下：





这就是resetlogs的过程。我们看到有了新的incarnation。这些就是顺带看一下。下面回归重点。

如果resetlogs打开了数据库，那么从还原的scn号也就是3999409以后的flashback logs就全部都被abandon了。

#### Rewinding open resetlogs操作

再次做一遍实验，这次我们open resetlogs，然后再把该操作“回退掉”，再让数据库回到最新的状态。

SQL> alter database flashback on;

Database altered.

SQL> CREATE RESTORE POINT before\_upgrade GUARANTEE FLASHBACK DATABASE;

Restore point created.

SQL> SELECT OLDEST\_FLASHBACK\_SCN, OLDEST\_FLASHBACK\_TIME FROM V$FLASHBACK\_DATABASE\_LOG;

OLDEST\_FLASHBACK\_SCN OLDEST\_FL

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

4019917 02-OCT-18

SQL> SELECT NAME, SCN, TIME, DATABASE\_INCARNATION#,GUARANTEE\_FLASHBACK\_DATABASE,STORAGE\_SIZE FROM V$RESTORE\_POINT;

NAME SCN TIME DATABASE\_INCARNATION# GUA STORAGE\_SIZE

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

BEFORE\_UPGRADE 4020731 02-OCT-18 05.35.48.000000000 PM 4 YES 209715200

SQL> create table yx.haha(id number);

Table created.

SQL> insert into yx.haha select current\_scn from v$database;

1 row created.

SQL> create index yx.idx\_haha on yx.haha(id);

Index created.

SQL> insert into yx.haha select current\_scn from v$database;

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from yx.haha;

ID

----------

4022341

4022554

SQL> shutdown immediate

SQL> startup mount

SQL> FLASHBACK DATABASE TO SCN 4022341;

Flashback complete.

SQL> alter database open resetlogs;

Database altered.

SQL> select \* from yx.haha;

no rows selected

现在我们发现我们其实是想要第一行数据的，但是我们没有read only检查就open resetlogs了，现在想重新回到执行flashback之前的状态怎么办？

SQL> SELECT RESETLOGS\_CHANGE# FROM V$DATABASE;

RESETLOGS\_CHANGE#

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

4022343

SQL> select OLDEST\_FLASHBACK\_SCN from V$FLASHBACK\_DATABASE\_LOG;

OLDEST\_FLASHBACK\_SCN

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

4019917

我们看到，只要resetlogs的scn比flashback log的scn大的话，我们就可以把resetlogs的操作“回滚”掉。

SQL> shutdown immediate

SQL> startup mount

SQL> FLASHBACK DATABASE TO BEFORE RESETLOGS;

Flashback complete.

现在我们就把数据库回滚到了resetlog之前。

然后我们要把数据库再次recover到第一次做flashback之前，那么我们就要把数据库重置到parent incarnation，然后recover：

RMAN> list incarnation;

using target database control file instead of recovery catalog

List of Database Incarnations

DB Key Inc Key DB Name DB ID STATUS Reset SCN Reset Time

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

1 1 PROD4 1716001866 PARENT 1 26-JAN-17

2 2 PROD4 1716001866 PARENT 1408558 08-JAN-18

3 3 PROD4 1716001866 PARENT 3999411 30-SEP-18

4 4 PROD4 1716001866 PARENT 4006600 30-SEP-18

5 5 PROD4 1716001866 CURRENT 4022343 02-OCT-18

RMAN> reset database to incarnation 4;

database reset to incarnation 4

RMAN> recover database;

Starting recover at 02-OCT-18

allocated channel: ORA\_DISK\_1

channel ORA\_DISK\_1: SID=622 device type=DISK

starting media recovery

archived log for thread 1 with sequence 1 is already on disk as file /home/oracle/arch4/1\_1\_988216566.arc

archived log file name=/home/oracle/arch4/1\_1\_988216566.arc thread=1 sequence=1

media recovery complete, elapsed time: 00:00:00

Finished recover at 02-OCT-18

然后我们把数据库启动到read only，检查一下数据：

SQL> alter database open read only;

Database altered.

SQL> select \* from yx.haha;

ID

----------

4022341

4022554

我们看到，数据库状态还原回来了。

#### Monitoring Flashback Database

从前面的实验我们也看到了，flashback的过程分为restore phase和recover phase。

Restore phase可以通过视图v$session\_longops监控：

SQL> SELECT sofar, totalwork, units FROM v$session\_longops WHERE opname = 'Flashback Database';

SOFAR TOTALWORK UNITS

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

17 60 Megabytes

Sofar是已经使用的flashback logs的大小，totalwork本次flashback一共需要使用的flashback logs的大小。Units是单位。也就是说sofar和totalwork的单位都是M。

对于recover phase的监控可以通过视图V$RECOVERY\_PROGRESS来观察。

# Flashback Table（Logical）

为什么要跟flashback database分开呢？因为俩者依赖的原理不一样。Flashback database是物理上的，依赖的是flashback logs，直接通过flashback logs里block修改前的images将block还原到过去的一个状态。

而对于table的flashback是逻辑上的，也就是说并不依赖flashback logs。对于table的flashback分俩种，一种是将表的数据flashback到之前的一个数据状态，这种依赖的是undo data，另一种是表被drop，使用flashback将其还原回来，这种依赖的是recyclebin。

如果一个表被flashback，那么它曾经拥有的索引，只要也还在回收站的，也会被flashback回来，但是表曾经拥有的外键，需要重新手动创建。如果部分索引的空间已经被重用，那么这部分索引也需要手动创建。

## Rewinding a Table

Flash table依赖于undo，效果是把表的数据还原到之前的一个状态。

原理就是delete新行，重新insert老行。

Flash table会自动维护表相关的索引、触发器和约束，就是flashback完了，并不需要手动去恢复这些属性。

### Prerequisites for Flashback Table

想使用该功能对用户和对象都有前提要求。

对于用户而言，主要就是权限：

1. 用户需要具备flashback any table的权限，或者对某个表具有flashback的权限
2. 对要flashback的表要有insert、delete、select、alter的权限，根据flashback table的原理这很显然，并不需要update的权限。
3. 要是想flashback table到一个还原点的话，还需要有SELECT ANY DICTIONARY或者flashback any table的权限，或者是SELECT\_CATALOG\_ROLE的role

对于对象而言：

1. 对象不能使如下类型：tables that are part of a cluster, materialized views, Advanced Queuing (AQ) tables, static data dictionary tables, system tables, remote tables, object tables, nested tables, or individual table partitions or subpartitions.说白了就是普通表。
2. 在current time到要flashback到的时间点之间，表不能发生结构变化，就是不能有如下的ddl在表上执行过：upgrading, moving, or truncating a table; adding a constraint to a table, adding a table to a cluster; modifying or dropping a column; adding, dropping, merging, splitting, coalescing, or truncating a partition or subpartition (except adding a range partition).
3. Row movement必须开启，也就是说flashback table会导致rowid发生变化
4. Undo\_retention要设置的足够保证从当前时间将表flashback到目标时间点。

### Performing a Flashback Table Operation

首先我直接给了yx用户一个dba role。那权限肯定是够了。

然后，我们把undo\_retention改成60秒：

SQL> alter system set undo\_retention=60;

System altered.

SQL> show parameter undo\_retention

NAME TYPE VALUE

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

undo\_retention integer 60

SQL> conn yx/yx

我们再看此时flashback database并没有开启，因为跟这个一点儿关系都没有：

SQL> select flashback\_on from v$database;

FLASHBACK\_ON

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

NO

做这俩步是为了什么，把undo retention变小，是为了证明，即使undo段已经是expired了，但只要还没被重用，那么就能查到对应的undo数据，能查到就能flashback。

确认flashback database是想再次强调，normal restore point和flashback database并不是固定组合。是可以组合使用也可以分别单独使用的。

下面我们进入正题。我们先看实验表：

SQL> select \* from test\_flash;

ID

----------

1

然后创建还原点：

SQL> create restore point before\_update;

Restore point created.

SQL> col name for a20

SQL> col time for a50

SQL> set lines 200

SQL> SELECT NAME, SCN, TIME FROM V$RESTORE\_POINT;

NAME SCN TIME

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

BEFORE\_UPDATE 3928867 29-SEP-18 04.21.36.000000000 PM

然后我们建个索引，并且完成一个事务：

SQL> create index idx\_flash\_id on test\_flash(id);

Index created.

SQL> insert into test\_flash values(2);

1 row created.

SQL> commit;

Commit complete.

SQL> select \* from test\_flash;

ID

----------

1

2

然后我们等个2分钟，开始做flashback：

SQL> ALTER TABLE test\_flash ENABLE ROW MOVEMENT;

Table altered.

SQL> FLASHBACK TABLE test\_flash to RESTORE POINT BEFORE\_UPDATE;

Flashback complete.

SQL> ALTER TABLE test\_flash disable ROW MOVEMENT;

Table altered.

千万记得，开了row movement，还要记得关。

然后我们来查数据：

SQL> set autot on

SQL> select id from test\_flash where id=1;

ID

----------

1

Execution Plan

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

Plan hash value: 3011311283

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

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

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

| 0 | SELECT STATEMENT | | 1 | 3 | 1 (0)| 00:00:01 |

|\* 1 | INDEX RANGE SCAN| IDX\_FLASH\_ID | 1 | 3 | 1 (0)| 00:00:01 |

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

Predicate Information (identified by operation id):

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

1 - access("ID"=1)

我们看到索引的使用并没有收到flashback的影响，同时数据确实还原了。

我们指定scn和timestamp：

SQL> FLASHBACK TABLE test\_flash to scn 3928866;

Flashback complete.

SQL> select \* from test\_flash;

ID

----------

1

我们再看个timestamp的，使用timestamp前后误差大概3秒，所以想要精确flashback到一个时间点，要使用scn，不建议使用timestamp。

可以使用to\_timestamp也可以使用sysdate-xx：

SQL> select ora\_rowscn,id from test\_flash;

ORA\_ROWSCN ID

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

3933955 1

SQL> insert into test\_flash values(2);

1 row created.

SQL> commit;

Commit complete.

SQL> select ora\_rowscn,id from test\_flash;

ORA\_ROWSCN ID

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

3934677 1

3934677 2

SQL> FLASHBACK TABLE test\_flash to TIMESTAMP sysdate-40/1440;

Flashback complete.

SQL> select ora\_rowscn,id from test\_flash;

ORA\_ROWSCN ID

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

3934945 1

同时我们也看到，scn一直在推进，所以为什么说这样的flashback是logical。

还有，默认是在flashback的时候，把表涉及的trigger会disable掉，然后flashback完了，在enable，我们也可以在flashback的时候指定让oracle一直保持trigger enable（我们做这个实验顺带观察下rowid的变化）：

SQL> FLASHBACK TABLE test\_flash to TIMESTAMP to\_timestamp('2018-09-29 16:21:00','yyyy-mm-dd hh24:mi:ss') enable triggers;

FLASHBACK TABLE test\_flash to TIMESTAMP to\_timestamp('2018-09-29 16:21:00','yyyy-mm-dd hh24:mi:ss') enable triggers

\*

ERROR at line 1:

ORA-08189: cannot flashback the table because row movement is not enabled

SQL> select rowid,id from test\_flash;

ROWID ID

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

AAASd+AAHAAAAC/AAA 1

SQL> insert into test\_flash values(2);

1 row created.

SQL> commit;

Commit complete.

SQL> ALTER TABLE test\_flash ENABLE ROW MOVEMENT;

Table altered.

SQL> select rowid,id from test\_flash;

ROWID ID

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

AAASd+AAHAAAAC/AAA 1

AAASd+AAHAAAAC/AAB 2

SQL> FLASHBACK TABLE test\_flash to TIMESTAMP to\_timestamp('2018-09-29 16:21:00','yyyy-mm-dd hh24:mi:ss') enable triggers;

Flashback complete.

SQL> select rowid,id from test\_flash;

ROWID ID

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

AAASd+AAHAAAAC/AAC 1

我们看，rowid确实发生了改变。

## Rewinding a DROP TABLE Operation

把一个被drop了的表进行flashback，依据的原理是，如果一个表被drop了，那么这个表只是被oracle rename了一下，表所占用的空间并不会立即释放，直到所在的tablespace处于space pressure的时候，这部分空间才会被reclaim，所以在只要这部分空间不被重用，说白了，就是在recyclebin中还能查到，就能flashback。

### Prerequisites of Flashback Drop

对权限上的要求就俩个，用户要有对表的drop权限和select权限，有了drop一个表的权限，就有flashback table…to before drop的权限，也就有了purge的权限，而在表drop前可以select，那在表drop后，依然可以从回收站中队该表select。

对于对象的要求：

1. The recycle bin is only available for non-system, locally managed tablespaces。就是说表只要不在system表空间上，且所在的表空间是locally managed，那么就能被recyclebin保护。
2. 表开了FGA（fine-grained auditing）和VPD（Virtual Private Database）策略的，是不受recyclebin保护的。
3. 分区索引组织表（Partitioned index-organized）不受recyclebin保护。
4. 表不能被purge，无论是手动的或者是oracle自己的space reclamation操作。

### Performing a Flashback Drop Operation

我们直接做实验，先建个表，并且为其建个索引：

SQL> create table test\_drop(id number);

Table created.

SQL> create index idx\_drop\_id on test\_drop(id);

Index created.

SQL> insert into test\_drop values(1);

1 row created.

SQL> commit;

Commit complete.

然后我们来把表drop了：

SQL> show recyclebin;

SQL> drop table test\_drop;

Table dropped.

SQL> show recyclebin;

ORIGINAL NAME RECYCLEBIN NAME OBJECT TYPE DROP TIME

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

TEST\_DROP BIN$dw7seqg5RFbgUwoMqMABTg==$0 TABLE 2018-09-30:11:02:49

我们看到，回收站里有表了，但是索引呢？索引其实也放进了回收站，但是show recyclebin只会显示table，所以我们要从视图recyclebin、USER\_RECYCLEBIN、DBA\_RECYCLEBIN中查：

SQL> col recycle\_name for a60

SQL> col original\_name for a30

SQL> col type for a10

SQL> set lines 200

SQL> SELECT object\_name AS recycle\_name, original\_name, type FROM recyclebin;

RECYCLE\_NAME ORIGINAL\_NAME TYPE

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

BIN$dw7seqg4RFbgUwoMqMABTg==$0 IDX\_DROP\_ID INDEX

BIN$dw7seqg5RFbgUwoMqMABTg==$0 TEST\_DROP TABLE

我们看到索引确实也放到了回收站，现在我们来通过原表名进行flashback：

SQL> flashback table test\_drop to before drop;

Flashback complete.

SQL> SELECT object\_name AS recycle\_name, original\_name, type FROM recyclebin;

no rows selected

我们看到，回收站中的索引和表都没了，那么我们再删一遍，就能知道索引是否也跟着还原了。

SQL> drop table test\_drop;

Table dropped.

SQL> SELECT object\_name AS recycle\_name, original\_name, type FROM recyclebin;

RECYCLE\_NAME ORIGINAL\_NAME TYPE

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

BIN$dw7seqg6RFbgUwoMqMABTg==$1 BIN$dw7seqg4RFbgUwoMqMABTg==$0 INDEX

BIN$dw7seqg7RFbgUwoMqMABTg==$0 TEST\_DROP TABLE

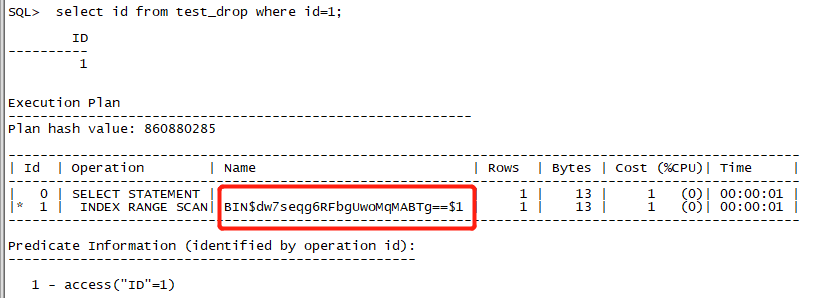
我们看到，索引是跟着还原了，但是名字变了，或者说没有变回去，original\_name竟然是上一次它在回收站里的名字。

我们现在通过回收站的名字进行对表的flashback：

SQL> flashback table "BIN$dw7seqg7RFbgUwoMqMABTg==$0" to before drop;

Flashback complete.

这次我们来验证一下索引，是不是仍然使用的是这一次在回收站中的名字：



我们看到，果然是。

其实这很好理解，因为oracle没有flashback index这个功能，如果flashback索引到原来的名字的话，万一这个名字已经被人使用了呢？！因为索引的名字几乎不用在意，除非sql中hint是指定索引名而不是索引列。所以oracle的这个做法可以说是很明智的选择。

我们再次来drop，但是这一次我们flashback加上rename to子句，看看索引还能否跟着回来：

SQL> drop table test\_drop;

Table dropped.

SQL> SELECT object\_name AS recycle\_name, original\_name, type FROM recyclebin;

RECYCLE\_NAME ORIGINAL\_NAME TYPE

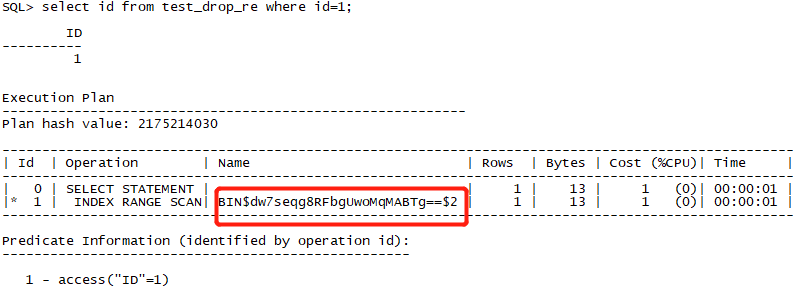
---------------------------------------- ------------------------------ ----------

BIN$dw7seqg8RFbgUwoMqMABTg==$2 BIN$dw7seqg6RFbgUwoMqMABTg==$1 INDEX

BIN$dw7seqg9RFbgUwoMqMABTg==$0 TEST\_DROP TABLE

SQL> flashback table "BIN$dw7seqg9RFbgUwoMqMABTg==$0" to before drop rename to test\_drop\_re;

Flashback complete.



我们看到索引还是回来了。

这里我们可以通过手动重命名index，来把index的名字改回去：

SQL> alter index "BIN$dw7seqg8RFbgUwoMqMABTg==$2" rename to idx\_drop\_id;

Index altered.

当然，如果删了一个表，那么该表名就可以被再次使用，如果回收站中有多个original\_name相同的表的话，如果是原表名恢复，并且不使用rename，那么最后进回收站的，会最先被选择用来flashback，其他的则仍留在回收站中。

这个就是个注意的点，其实最好是在回收站中查询好，确认好，然后再还原，所以我们这个跟在回收站中直接进行查询，一并实验了：

SQL> create table test\_drop(name varchar(2));

Table created.

SQL> insert into test\_drop values('a');

1 row created.

SQL> commit;

Commit complete.

SQL> drop table test\_drop;

Table dropped.

SQL> alter table test\_drop\_re rename to test\_drop;

Table altered.

SQL> drop table test\_drop;

Table dropped.

SQL> show recyclebin;

ORIGINAL NAME RECYCLEBIN NAME OBJECT TYPE DROP TIME

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

TEST\_DROP BIN$dw7seqhARFbgUwoMqMABTg==$0 TABLE 2018-09-30:11:44:06

TEST\_DROP BIN$dw7seqg+RFbgUwoMqMABTg==$0 TABLE 2018-09-30:11:43:08

我们现在来看这俩个表，使用回收站的名字，不仅可以desc而且还可以select：

SQL> select \* from "BIN$dw7seqhARFbgUwoMqMABTg==$0";

ID

----------

1

SQL> select \* from "BIN$dw7seqg+RFbgUwoMqMABTg==$0";

NA

--

a

SQL> desc "BIN$dw7seqhARFbgUwoMqMABTg==$0";

Name Null? Type

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

ID NUMBER

SQL> desc "BIN$dw7seqg+RFbgUwoMqMABTg==$0";

Name Null? Type

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

NAME VARCHAR2(2)

注意，我们最后删的带有id列的test\_drop，那么我们现在使用原表名flashback，那么还原出来的表就是带id列的test\_drop：

SQL> flashback table test\_drop to before drop;

Flashback complete.

SQL> select \* from test\_drop;

ID

----------

1