ODU 使用指南

第一版

2011.4

目录

第一章 ODU 软件介绍

什么是 ODU

ODU 全称为 Oracle Database Unloader，是用于 Oracle 数据库紧急恢复的软件，在各种原因造成的数据库不能打开或数据删除后没有备份时，使用 ODU 抢救数据，最大限度地减少数据丢失。

现实中总会有很多的意外，数据被意外删除、硬件问题导致数据库损坏、错误地格式化了 ASM 磁盘等等，在没有备份的情况下，ODU 能够通过直接访问 Oracle 数据库数据文件或直接访问 ASM 磁盘，恢复出所有完好的数据，以避免所有数据丢失造成的损失。

ODU 的功能特点

- 不需要运行 Oracle 数据库软件，ODU 直接读取数据库文件和 ASM 磁盘进行数据解析。在由于硬件问题或人为误操作引起数据损坏时，Oracle 数据库软件为遵行数据完整性和一致性原则，往往不能打开数据库对数据库进行访问，而 ODU 可以绕过 Oracle 数据库软件，直接从 Oracle 数据库文件和 ASM 磁盘中解析数据，从而恢复数据。
- 支持 ASM，能够直接从 ASM 磁盘中解析数据。即使 Oracle 数据文件存储在 ASM 磁盘中，ODU 仍然能够直接从 ASM 磁盘中读取数据，即使由于硬件或错误地 FORMAT 导致 ASM 磁盘头部数据损坏而导致 ASM 磁盘组不能加载时，ODU 仍然能够从 ASM 中恢复数据。
- 支持从 ASM 磁盘组中直接抽取出数据文件和其他任意存储在 ASM 磁盘组中的文件，并保存为文件。比如可以将 ASM 磁盘组中的在线日志文件、控制文件等文件抽取出来保存到文件系统中。
- 多版本 Oracle 支持，支持的 Oracle 数据库版本包括 7,8i,9i,10g,11g。
- 自动检测数据文件的表空间号和文件号，在只知道数据文件名而不知道数据文件所属表空间等信息时，ODU 能够自动获取这些信息。
- 在有 SYSTEM 表空间的情况下，自动解析和获取数据字典信息。这样使用 ODU 恢复的数据能够生成与原来一样的用户名以及列类型。
- 支持各种类型的表，包括普通的 HEAP 表，IOT 表和聚簇（CLUSTER）表。
- 支持类似于 SQLPLUS 的 DESC 命令，以显示表的列定义。
- 支持 10g 及以上的大文件(BigFile)表空间。
- 可以列出表的分区和子分区，并能够指定只需要恢复的分区和子分区。
- 支持表被 Truncate 后的数据恢复。由于表被 Truncate 之后，只能使用备份或 database 级的 flashback(10g 以上的新功能）才能恢复 Truncate 的数据。在没有备份、备份不可用或 Database Flashback 没有开启时，将不能使用正常的方法来恢复数据。ODU 可以在 Truncate 的表的空间没有被占用的情况下恢复出 Truncate 之前的数据。
- 支持表被 Drop 之后的数据恢复，如果表被 Drop 同时 Drop 的表之前的空间没有被占用，则可以使用 ODU 恢复被 Drop 的表之前的数据。

- 支持压缩表，ODU 能够恢复 Oracle 压缩表的数据。
- 支持在没有 SYSTEM 表空间和数据字典损坏的情况下恢复数据，在没有数据字典可用时，ODU 能够自动判断数据的类型。
- 支持 IOT 表的恢复：
  - ✓ 支持普通 IOT 表
  - ✓ 支持压缩 IOT 表
  - ✓ 支持 IOT 表溢出段
  - ✓ 支持 IOT 表分区（包括子分区）
- 支持多种平台的数据库和跨平台数据恢复，包括 AIX、LINUX、HPUX、SOLARIS、WINDOWS 所有主流的操作系统和平台。这里包括有两种含义，第一是 ODU 有多个平台的版本，能够在所有主流的操作系统上运行；其二是能够在一个平台上恢复其他平台上的 Oracle 数据库数据，比如在可以将 AIX 系统上的 Oracle 数据文件复制到 Linux 系统上，然后使用 Linux 系统的 ODU 来恢复，反之亦然。
- 支持的数据类型包括：NUMBER, CHAR, VARCHAR2, NCHAR,NVARHCAR2, LONG, DATE, RAW, LONG RAW, BLOB, CLOB, TIMESTAMP (9i+) , BINARY FLOAT, BINARY DOUBLE (10g+) 。
- 恢复的数据能够以纯文本和 DMP 文件两种格式存储。以纯文本导出时，能够自动生成建表所需的 SQL 语句和 SQLLDR 导入所需的 CONTROL 文件（.CTL)，以 DMP 文件格式存储的数据，能够使用 Oracle 自带的 imp 命令导入到数据库中。
- 支持同一个库中不同的块大小的数据文件。
- 全面支持 LOB 字段：
  - ✓ 支持 CLOB、NCLOB 和 BLOB
  - ✓ CLOB 支持 Big Endian 和 Little Endian 字节序
  - ✓ 支持 LOB 分区，子分区
  - ✓ 支持同一个表中，不同 LOB 列使用不同 CHUNK SIZE 的情况
  - ✓ CLOB 数据可以导出到与其他列相同的文件中，或存储到单独的文件中
  - ✓ LOB 列在没有 SYSTEM 表空间的情况下仍然能够导出
- 支持多种字符集之间的转换，能够正确的转换 CLOB、NCLOB、NVARCHAR2 列类型的数据到与数据库相同的字符集。
- ODU 全面支持 64 位系统，支持超过 4G 大小的数据文件。
- 支持复制操作系统命令不能复制的坏文件。由于存储硬件问题导致数据库中数据文件部分损坏，并且此时操作系统的相关操作不能正常读取该文件时，可以使用 ODU 复制这样的文件到其他位置，以替换掉损坏的文件或存储阵列。

ODU 的其他特点

除了上述功能上的特点之外，ODU 还具有如下特点：

- **数据恢复速度快、占用内存少。**

ODU 由在所有主流开发语言中速度最快、内存使用最有效率的 C 语言所写，恢复速度快，经测试可以达到与 Oracle 软件自身解析数据相同的速度。同时 ODU 的一些功能能够并行执行，以提高恢复速度；ODU 支持命令文件，通过使用不同的命令文件多并发执行 ODU 提高数据恢复速度。这样在进行数据恢复时，能够大大减少恢复所需要的时间，从而减少因为

数据丢失和损坏引起的业务停止时间。

- **使用简单。**

使用 ODU 恢复数据非常简单，只需要简单的配置加上两三条命令即可恢复数据。这样在数据损坏或丢失时，不需要经过专门的培训和长时间的学习，就能使用 ODU 进行恢复。

- **运行稳定**

ODU 能够自动检测并跳过数据库中的坏块，避免了坏块解析时引起的异常，保持数据恢复的稳定不出错。

目前不支持的功能

ODU 目前暂时不支持下列特性或数据类型的恢复：

- Oracle 11g 的 SecureFiles。
- 使用 Oracle TDE 加密的数据。
- BFILE 数据类型。
- NESTED TABLE、用户自定义的数据类型。

由于这几类数据用得极少，基本不影响数据的恢复。

第二章　安装 ODU 和使用 ODU

安装 ODU，只需要简单的 3 步即可完成。

下载 ODU 软件

访问http://www.oracleodu.com/cn/download可以下载最新的ODU软件。下载ODU软件时，需要确认要使用的ODU软件所运行的平台。ODU软件安装包的命名如下：

```
odu_<版本号>_<平台>.zip
```

或

```
odu_<版本号>_<平台>.tar.gz
```

版本号中带有 trial 表示试用版。

例如：

- odu_413_hp_ia64.tar.gz 表示安腾处理器平台 HP-UX 操作系统上的 ODU，版本号为 4.1.3。
- odu_trial_411_hp_ia64.tar.gz 表示安腾处理器平台 HP-UX 操作系统上的 ODU 试用版，版本号为 4.1.1。
- odu_413_linux_x86.tar.gz 表示 X86 处理器平台 Linux 操作系统上的 ODU，版本号为 4.1.3。
- odu_413_win32.zip 表示 32 位 Windows 操作系统上的 ODU，版本号为 4.1.3。

提示：

大多数新版本操作系统能够兼容低版本操作系统上的软件，同时高位数（64 位）平台能够运行低位数(32 位）平台的软件。比如 odu_413_win32 能够运行于所有的 Windows 2000/xp/2003/Win7 操作系统，不管操作系统是 32 位还是 64 位。odu_413_linux_x86 同样能够运行于主要的 Linux 发行版上，不管操作系统是 32 位还是 64 位。

提示：

在使用IE浏览器下载ODU软件时，IE浏览器可能会将安装包的tar.gz扩展名自动改为tar.tar，对于这样的改变，请将文件的扩展名改回为 tar.gz。

提示：

ODU 正式版和试用版的区别：试用版仅用于测试、学习和验证，只能恢复 SYSTEM 表空间下的数据，对于其他表空间的数据，仅恢复少量的数据以验证数据可恢复。而正式版在获取 LICENSE 后能够恢复所有能够恢复的数据。

创建目录和上传 ODU 软件

在 ODU 将要运行的系统上，创建将要安装 ODU 软件的目录。下载的 ODU 软件安装包通过 ftp、sftp 等方式上传到准备安装 ODU 的目录。如果是 Windows 系统或是直接使用 wget 命令直接下载的安装包，则将安装包复制或移动到准备安装 ODU 的目录。

建议使用 Oracle 软件的拥有者(owner）用户来安装 ODU 软件。用其他用户安装可能使 ODU 不能读取 Oracle 的数据文件。

ODU 安装目录不包括 odu 这一层子目录。比如计划将 ODU 安装到/oracle/odu 目录，则需要有/oracle 目录并将 ODU 软件安装包复制或移动到/oracle 目录下即可。

ODU 软件本身需要的硬盘空间并不多，10MB 空间已经足够。但是 ODU 运行过程中产生的数据需要的空间主要是数据字典，与数据库大小有关。但通常 200MB 已经足够。

解压 ODU 软件

对于以 zip 结尾的安装包，在 Windows 系统上使用 winzip、winrar 或 7zip 等软件将 ODU 软件包解压到指定目录；在 Linux、UNIX 系统上使用 unzip 命令解压 ODU 软件安装包。
对于以 tar.gz 结尾的安装包，使用 gunzip 和 tar 命令解压 ODU 软件包，例如在 Linux 系统上，计划将 ODU 安装到/oracle/odu 目录，而安装包已经复制到/oracle 目录下：

```
cd /oracle
gunzip xf odu_413_linux_x86.tar.gz
tar xf odu_413_linux_x86.tar
```

ODU 软件解压后，通常有下列文件和目录：

- ✧ odu ODU 可执行文件，这是 ODU 主程序。
- ✧ config.txt ODU 配置文件。
- ✧ control.txt ODU 控制文件，用于在数据恢复时指定使用的 Oracle 数据库文件。
- ✧ asmdisk.txt ASM 磁盘信息设置文件，用于在数据恢复时指定使用的 ASM 磁盘组磁盘。
- ✧ lib ODU 软件库文件目录，在某些平台上没有此目录。
- ✧ data ODU 默认的数据目录，用于存储 ODU 恢复出来的数据。

ODU 软件包解压后 odu 主程序可能没有可执行属性（权限），在这种情况下，使用下面的命令进行设置（假设 ODU 安装在/oracle/odu 目录下）：

```
cd /oracle/odu
ls -l odu
-rwxr-xr-x 1 oracle oinstall 2677388 Apr    1 23:24 odu
如果上述显示结果中，odu 没有"x"属性，则需要执行下面的命令：
chown u+x odu
```

使用 ODU 软件

在安装完成后，进入 ODU 安装目录，对于 Windows 系统直接运行 odu.exe 文件即可进入 ODU 界面，而对于 Linux/Unix 系统，输入命令./odu，通常也能直接进入到 ODU 界面。但在部分操作系统下可能会报类似于下面的错误：

```
./odu: error while loading shared libraries: libiconv.so.2: cannot open shared object file: No such file or directory
```

这是由于需要的库文件没有在搜索路径中。执行下面的命令：export LD_LIBRARY_PATH=<ODU 目录>/lib:$LD_LIBRARY_PATH，再运行命令：./odu 则会正常运行（这里请将<ODU 目录>替换为实际的 ODU 安装目录）。

在 Solaris 和 HP-UX 操作系统下，这个环境变量也是 LD_LIBRARY_PATH，但在 AIX 下，这个环境变量则是 LIBPATH。

odu 压缩包里面所包含的库文件或其不同的版本可能已经存在于系统中。为避免不同版本库文件引起问题，可在设置环境变量时将 odu 所在的 lib 目录放到环境变量中的前面部分。

为避免反复设置 LD_LIBRARY_PATH 和 LIBPATH 环境变量，可将此环境变量的设置，加入到用户的 profile 文件中。

进入 ODU 后，ODU 会显示 ODU 的命令提示符 ODU>，这样就可以输入命令来执行操作：

```
ODU>
```

第三章　使用 ODU 恢复数据

ODU 数据恢复快速上手

以下以 ODU for Linux 为例，介绍如何使用 ODU 恢复使用了 ASM 磁盘组的数据库的数据。假设数据库不能打开，需要恢复重要的数据 SYS.T1 表。

1.1. 修改 ODU 软件 ASM 磁盘设置文件 asmdisk.txt

进入 ODU 软件所在目录，并使用 vi 修改 asmdisk.txt 文件：

```
cd /oracle/odu
vi asmdisk.txt
```

在 asmdisk.txt 中加入 ASM 磁盘组的所有 ASM 磁盘的设备文件路径及名称：

```
0  /oradata/asm/disk1.dbf
0  /oradata/asm/disk2.dbf
0  /oradata/asm/disk3.dbf
```

1.2. 修改 ODU 软件控制文件 control.txt

使用 vi 命令修改 control.txt 文件，将数据库中所有的数据文件路径及名称加入到 control.txt 文件中：

```
      0        0        0    +DGDATA/xty/datafile/system.260.745630773
      0        0        0    +DGDATA/xty/datafile/undotbs1.261.745630805
      0        0        0    +DGDATA/xty/datafile/sysaux.262.745630817
      0        0        0    +DGDATA/xty/datafile/users.264.745630833
```

1.3. 执行 odu 命令，进入 ODU 软件

```
$ ./odu

Oracle Data Unloader:Release 4.1.3

Copyright (c) 2008,2009,2010,2011 XiongJun. All rights reserved.

Web: http://www.oracleodu.com
Email: magic007cn@gmail.com
loading default config.......

byte_order little
```

```
block_size  8192
block_buffers 1024
db_timezone -7
client_timezone 8
asmfile_extract_path   /asmfile
data_path   data
lob_path    /odu/data/lob
charset_name US7ASCII
ncharset_name AL16UTF16
output_format text
lob_storage infile
clob_byte_order big
trace_level 1
delimiter |
unload_deleted no
file_header_offset 0
is_tru64 no
record_row_addr no
convert_clob_charset yes
use_scanned_lob  yes
trim_scanned_blob yes
lob_switch_dir_rows 20000
db_block_checksum yes
db_block_checking yes
rdba_file_bits 10
compatible 10
load config file 'config.txt' successful
loading default asm disk file ......


grp# dsk# bsize ausize disksize diskname        groupname       path
---- ---- ----- ------ -------- --------------- --------------- ----------------------
   1    1  4096  1024K     1024 DGDATA_0001     DGDATA          /oradata/asm/disk1.dbf
   1    0  4096  1024K     1024 DGDATA_0000     DGDATA          /oradata/asm/disk2.dbf
   1    2  4096  1024K     1024 DGDATA_0002     DGDATA          /oradata/asm/disk3.dbf

load asm disk file 'asmdisk.txt' successful
loading default control file ......


 ts#   fn  rfn bsize   blocks bf offset filename
---- ---- ---- ----- -------- -- ------ ----------------------------------------
   0    1    1  8192    44800 N       0 +DGDATA/xty/datafile/system.260.745630773
   1    2    2  8192    25600 N       0 +DGDATA/xty/datafile/undotbs1.261.745630805
```

```
   2     3     3  8192    15360 N       0 +DGDATA/xty/datafile/sysaux.262.745630817
   4     4     4  8192      800 N       0 +DGDATA/xty/datafile/users.264.745630833
load control file 'control.txt' successful
loading dictionary data......done

loading scanned data......done
```

1.4. 运行 save control 命令：

```
ODU> save control

The file write completed.
ODU> exit
```

save control 命令会在 ODU 软件目录下生成一个名为 oductl.txt 文件，然后使用 exit 命令退出 ODU 软件界面。

1.5. 获取 LICENSE

然后将将文件oductl.txt发给http://www.oracleodu.com/cn/support中指定的技术支持邮箱，然后获得一个名为oductl.dat的LICENSE文件，然后将此文件复制到ODU所在目录。

提示：

一份 LICENSE 只能恢复一个数据库中的数据，同时 LICENSE 有 30 天的使用期限，这是为了避免误用 ODU，绕过 Oracle 直接获取 Oracle 数据库中的敏感数据，以保证数据安全。

一个库在恢复之前，只需要获取一次 LICENSE 即可，不需要多次获取 LICENSE。

对于试用版本，不需要执行上述第 4 和第 5 步骤。

1.6. 获取数据字典信息

重新进入 ODU，执行 unload dict 命令，获取数据字典信息。

```
ODU> unload dict
CLUSTER C_USER# file_no: 1 block_no: 89
TABLE OBJ$ file_no: 1 block_no: 121
CLUSTER C_OBJ# file_no: 1 block_no: 25
CLUSTER C_OBJ# file_no: 1 block_no: 25
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
found TABPART$'s obj# 266
found TABPART$'s dataobj#:266,ts#:0,file#:1,block#:2121,tab#:0
found INDPART$'s obj# 271
found INDPART$'s dataobj#:271,ts#:0,file#:1,block#:2161,tab#:0
```

```
found TABSUBPART$'s obj# 278
found TABSUBPART$'s dataobj#:278,ts#:0,file#:1,block#:2217,tab#:0
found INDSUBPART$'s obj# 283
found INDSUBPART$'s dataobj#:283,ts#:0,file#:1,block#:2257,tab#:0
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
found LOB$'s obj# 151
found LOB$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:6
found LOBFRAG$'s obj# 299
found LOBFRAG$'s dataobj#:299,ts#:0,file#:1,block#:2393,tab#:0
```

1.7. 使用 Unload 命令恢复表

```
ODU> unload table sys.t1

Unloading table: T1,object ID: 42138 at 2011-04-07 13:58:41
Unloading segment,storage(Obj#=42138 DataObj#=42138 TS#=4 File#=4 Block#=11 Cluster=0)
41161 rows unloaded
At 2011-04-07 13:58:43
```

可以看到只需要 2 秒钟就恢复了行数为 41161 的表，为 SYS 用户下的 T1 表。

1.8. 将数据导入到新的数据库中

在 ODU 所在目录的 data 子目录下，可以看到有 3 个文件：

```
[oracle@xty data]$ ls -l
total 4800
-rw-r--r-- 1 oracle oinstall     597 Apr  7 13:58 SYS_T1.ctl
-rw-r--r-- 1 oracle oinstall     409 Apr  7 13:58 SYS_T1.sql
-rw-r--r-- 1 oracle oinstall 4893152 Apr  7 13:58 SYS_T1.txt

[oracle@xty data]$ cat SYS_T1.sql
CREATE TABLE "SYS"."T1"
(
    "OWNER" VARCHAR2(30) ,
    "OBJECT_NAME" VARCHAR2(128) ,
    "SUBOBJECT_NAME" VARCHAR2(30) ,
    "OBJECT_ID" NUMBER ,
    "DATA_OBJECT_ID" NUMBER ,
    "OBJECT_TYPE" VARCHAR2(19) ,
    "CREATED" DATE ,
    "LAST_DDL_TIME" DATE ,
    "TIMESTAMP" VARCHAR2(19) ,
```

```
    "STATUS" VARCHAR2(7) ,
    "TEMPORARY" VARCHAR2(1) ,
    "GENERATED" VARCHAR2(1) ,
    "SECONDARY" VARCHAR2(1)
);
[oracle@xty data]$ cat SYS_T1.ctl
--
--Generated by ODU,for table "SYS"."T1"
--
OPTIONS(BINDSIZE=8388608,READSIZE=8388608,ERRORS=-1,ROWS=50000)
LOAD DATA
INFILE 'SYS_T1.txt' "STR X'0a'"
APPEND INTO TABLE "SYS"."T1"
FIELDS TERMINATED BY X'7c' TRAILING NULLCOLS
(
    "OWNER" CHAR(30),
    "OBJECT_NAME" CHAR(128),
    "SUBOBJECT_NAME" CHAR(30),
    "OBJECT_ID" ,
    "DATA_OBJECT_ID" ,
    "OBJECT_TYPE" CHAR(19),
    "CREATED" DATE "yyyy-mm-dd hh24:mi:ss",
    "LAST_DDL_TIME" DATE "yyyy-mm-dd hh24:mi:ss",
    "TIMESTAMP" CHAR(19),
    "STATUS" CHAR(7),
    "TEMPORARY" CHAR(1),
    "GENERATED" CHAR(1),
    "SECONDARY" CHAR(1)
)
```

SYS_T1.sql 是创建 SYS.T1 表的 SQL 语句，而 SYS_T1.ctl 是用于将恢复的数据使用 SQL*Loader 导入到新数据库中的控制文件。而 SYS_T1.txt 则是实际存储了 SYS.T1 表数据的文本文件。

使用 Oracle 的 SQL*Loader（sqlldr）工具将 SYS_T1.txt 文件导入到新的数据库中即完成了 SYS 用户下 T1 表的数据恢复。

使用 ODU 恢复数据完整步骤

虽然利用 ODU 来恢复 Oracle 数据库的数据比较简单，但是为了更顺利、更迅速地完成数据恢复，建议按以下步骤来完成恢复：

1) 确定数据库数据丢失程度和恢复范围。

在恢复之前，确定数据丢失程度，是需要恢复少量被 DROP/TRUNCATE 的表，或者是少量存在数据损坏的表，或者是数据库用户被删除需要恢复一个用户下的所有数据，或者是数据库完全不能打开需要恢复全库。

2) 确认数据库损坏程度。

需要确认数据库数据文件是否齐备，特别是 SYSTEM 表空间是否完好。在使用了 ASM 的情况下 ASM 磁盘是否齐备、ASM 磁盘组是否能 MOUNT。如果 SYSTEM 表空间丢失或损坏，或者是表或用户被 DROP，需要使用基于无数据字典情况下的恢复。在这种情况下，建议请非常熟悉系统数据和数据库结构的维护人员或开发人员参与数据恢复。

3) 估算需要的存储空间以及准备足够的空间来保存恢复的数据。
在前一步确定了恢复范围的情况下，估算需要的存储空间，如果没有足够的存储空间，应该寻求系统管理员和存储管理员的帮助，以分配到足够的空间。

4) 收集足够的信息
这些信息包括 ASM 磁盘的路径、数据文件的名称、数据库的平台、数据库的版本、数据库在使用裸设备的情况下确认裸设备是否存在头部偏移。

5) 确定数据恢复保存的格式
ODU 支持两种类型的保存格式，一种为可以使用 SQL*Loader（SQLLDR）导入数据的文本文件格式，另一种为可以使用 IMP 工具导入的 DMP 文件格式。如果数据比较复杂，比如有 LONG 类型、LOB 类型、带有特殊字符的 VARCHAR2 类型，建议使用 DMP 格式保存。

6) 配置 ODU
在完成前面所述步骤之后，根据收集到的信息修改 ODU 配置参数、修改 ODU 控制文件和 ASM 磁盘信息文件（如果使用了 ASM 磁盘组）。
关于 ODU 配置参数，请参见后面关于 ODU 配置参数参考章节。
关于 ODU 控制文件和磁盘信息文件，请参见后面关于 ODU 控制文件和磁盘信息文件参考章节。

7) 获得 ODU 正式版 License
根据http://www.oracleodu.com/cn/buy的步骤，获取ODU正式版License。

8) 使用 ODU 恢复数据
使用 ODU 的 unload 命令恢复数据。关于 ODU 命令，请参见后面关于 ODU 命令参考章节。

9) 导入恢复的数据并确认数据
使用 ODU 将数据恢复为文件后，使用 SQL*Loader（SQLLDR）或 IMP 工具将数据导入到新的完好的数据库中。建议不要导入到原来的数据库中，这是为了保护现场，避免导入数据时覆盖了原来的空间。只要在确认数据没有问题后，再将数据导入到生产库中。

10) 创建其他对象
数据恢复之后，还应该创建其他的对象，才能真正用于生产系统。这些对象包括索引、主外

键约束、触发器。而在用户或数据库全库恢复时，还需要创建存储过程等对象。

以下的章节将详细描述如何使用 ODU 提供的功能来恢复数据。

ODU 恢复数据的几种场景

本节将介绍几种场景下使用 ODU 进行数据恢复时，使用的命令序列。

场景 1. 数据库不能启动，但是 SYSTEM 表空间中的数据字典是完整的

- 生成数据字典：unload dict
- 列出用户：list user
- 列出用户下的所有表：list table username
- 恢复每张表：unload table username.tablename
- 也可以按用户恢复：unload user username

场景 2. 表被 TRUNCATE

- OFFLINE 表所在的表空间
- 生成数据字典：unload dict
- 扫描数据：scan extent
- 恢复表：unload table username.tablename object truncate

如果上述步骤不成功，继续执行下面的步骤：

- 显示表的段头：desc username.tablename
- 找到实际的 data object id： dump datafile file# block block#
- 恢复表：unload table username.tablename object <data_object_id>

场景 3. 表被 DROP

- OFFLINE 表所在的表空间
- 使用 logminer 从日志里面挖掘被 drop 掉的表的 data object id，或使用闪回查询 SYS.OBJ$表得到 DROP 表的 data object id，如果不能得到 data object id，按下面的场景 4 进行恢复。
- 扫描数据：scan extent
- 如果没有表结构信息，需要自动来判断：unload object data_object_id sample
- 恢复表：unload object data_object_id column coltype coltype…

场景 4. 系统表空间丢失或损坏

- 扫描数据：scan extent
- 搜索数据：unload object all sample
- 从结果文件 sample.txt 查找需要的数据
- 恢复需要的表：unload object data_object_id column coltype coltype…

场景 5. 表中数据被 DELETE

- 将参数 unload_deleted 设置为 YES
- 生成数据字典：unload dict
- 恢复表: unload table username.tablename

场景 6. 表中存在坏块

- 生成数据字典：unload dict
- 扫描数据：scan extent
- 恢复表: unload table username.tablename object scanned

使用 ODU 恢复被 Truncate 的表

意外 Truncate 表的事情时有发生，ODU 提供了方便的恢复 Truncate 表的功能。被 Truncate 的表，只要原来的空间没有被重用（即数据被覆盖），则数据都是可以恢复的。

如果发现一个表被意外地 Truncate，而需要马上恢复。首先要做的就是关闭数据库，或者 OFFLINE 那个表所在的表空间，或者关闭所有应用。目的只有一个，确保空间不会被重用，数据不会被覆盖。

ODU 恢复被 Truncate 掉的表，有 2 种方式，一种是使用 unload table <schema.tablename> object truncate 命令来恢复，另一种是需要手工找到表被 Truncate 之前的 data object id，然后使用 unload table <schema.tablename> object <data_object_id>命令来恢复。

下面举例说明用第 1 种方式恢复被 Truncate 掉的表。

1) 建立两个测试的表 T1 和 T2，这两个表的数据完全一样。建两个数据完全一样的表的目的在于方便在恢复后对比数据。然后 Truncate T1 表。

```
SQL> conn test/test
Connected.
SQL> create table t1 as select * from dba_objects where rownum<=1000;
```

```
Table created.

SQL> create table t2 as select * from t1;

Table created

SQL> truncate table t1;

Table truncated.
```

2) OFFLINE 掉 T1 表的表空间（在实际的系统中，如果有比较多的活动事务，则表空间不容易被 OFFLINE 下来）。然后做一个 Checkpoint，让 ODU 能够读到最新的数据字典数据。

```
SQL> select tablespace_name from user_tables where table_name='T1';

TABLESPACE_NAME
------------------------------
TBS_TEST

SQL> alter tablespace tbs_test offline;

SQL> alter system checkpoint;
```

3) 运行 ODU，并 unload 数据字典。

```
ODU> unload dict
CLUSTER C_USER# file_no: 1 block_no: 89
TABLE OBJ$ file_no: 1 block_no: 121
CLUSTER C_OBJ# file_no: 1 block_no: 25
CLUSTER C_OBJ# file_no: 1 block_no: 25
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
found TABPART$'s obj# 266
found TABPART$'s dataobj#:266,ts#:0,file#:1,block#:2121,tab#:0
found INDPART$'s obj# 271
found INDPART$'s dataobj#:271,ts#:0,file#:1,block#:2161,tab#:0
found TABSUBPART$'s obj# 278
found TABSUBPART$'s dataobj#:278,ts#:0,file#:1,block#:2217,tab#:0
found INDSUBPART$'s obj# 283
found INDSUBPART$'s dataobj#:283,ts#:0,file#:1,block#:2257,tab#:0
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
```

```
found LOB$'s obj# 151
found LOB$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:6
found LOBFRAG$'s obj# 299
found LOBFRAG$'s dataobj#:299,ts#:0,file#:1,block#:2393,tab#:0
```

4) 获取 TEST 用户下的 T1 表，也就是我们要恢复的表的信息：

```
ODU> desc test.t1



Object ID:42252
Storage(Obj#=42252 DataObj#=42255 TS#=6 File#=20 Block#=11 Cluster=0)

NO.  SEG INT Column Name                    Null?     Type
---- --- --- ------------------------------ --------- ------------------------------
  1    1   1 OWNER                                    VARCHAR2(30)
  2    2   2 OBJECT_NAME                              VARCHAR2(128)
  3    3   3 SUBOBJECT_NAME                           VARCHAR2(30)
  4    4   4 OBJECT_ID                                NUMBER
  5    5   5 DATA_OBJECT_ID                           NUMBER
  6    6   6 OBJECT_TYPE                              VARCHAR2(19)
  7    7   7 CREATED                                  DATE
  8    8   8 LAST_DDL_TIME                            DATE
  9    9   9 TIMESTAMP                                VARCHAR2(19)
 10   10  10 STATUS                                   VARCHAR2(7)
 11   11  11 TEMPORARY                                VARCHAR2(1)
 12   12  12 GENERATED                                VARCHAR2(1)
 13   13  13 SECONDARY                                VARCHAR2(1)
```

从上面的输出中，可以看到，TEST.T1 表所在的表空间号为 6。

5) 用 ODU 扫描表空间的 extent：

```
ODU> scan extent tablespace 6

scan extent start: 2011-04-09 11:50:07
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-09 11:50:25
```

6) 使用 ODU 来 unload 数据：

```
ODU> unload table test.t1 object truncate
Auto mode truncated table.
```

```
Unloading table: T1,object ID: 42252 at 2011-04-09 11:51:02
Unloading segment,storage(Obj#=42252 DataObj#=42252 TS#=6 File#=20 Block#=11 Cluster=0)
1000 rows unloaded
At 2011-04-09 11:51:03
```

7) 将 TBS_TEST 表空间 ONLINE：

```
SQL> alter tablespace tbs_test online;
```

8) 使用 SQL*Loader（sqlldr）导入恢复的数据：

```
[oracle@xty data]$ sqlldr test/test control=TEST_T1.ctl

SQL*Loader: Release 10.2.0.5.0 - Production on Sat Apr 9 11:51:56 2011

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

Commit point reached - logical record count 1000
```

至此，恢复数据的步骤已经完成。对比一下数据，看看数据是否和被 Truncate 前的数据完全一样：

```
SQL> select * from t2 minus select * from t1;

no rows selected

SQL> select * from t1 minus select * from t2;

no rows selected
```

可以看到，数据已经完全恢复。

下面举例说明在第 1 种方式不可行时，用第 2 种方式来恢复被 Truncate 掉的表。

1) 建立两个测试的表 T1 和 T2，这两个表的数据完全一样。建两个数据完全一样的表的目的在于方便在恢复后对比数据。

```
SQL> connect test/test

SQL> create table t1 as select * from dba_objects;

SQL> create table t2 as select * from t1;
```

```
SQL> truncate table t1;
```

2) OFFLINE 掉 T1 表的表空间（在实际的系统中，如果有比较多的活动事务，则表空间不容易被 OFFLINE 下来）。然后做一个 Checkpoint，让 ODU 能够读到最新的数据字典数据。

```
SQL> select tablespace_name from user_tables where table_name=' T1' ;

TABLESPACE_NAME
------------------------------
TEST

SQL> alter tablespace test offline;

SQL> alter system checkpoint;
```

3) 运行 ODU，并 unload 数据字典。

```
ODU> unload dict
get_bootstrap_dba: compat header size:12
CLUSTER C_USER# file_no: 1 block_no: 177
TABLE OBJ$ file_no: 1 block_no: 241
CLUSTER C_OBJ# file_no: 1 block_no: 49
CLUSTER C_OBJ# file_no: 1 block_no: 49
found IND$' s obj# 19
found IND$' s dataobj#:2,ts#:0,file#:1,block#:49,tab#:3
found TABPART$' s obj# 230
found TABPART$' s dataobj#:230,ts#:0,file#:1,block#:3313,tab#:0
found INDPART$' s obj# 234
found INDPART$' s dataobj#:234,ts#:0,file#:1,block#:3377,tab#:0
found TABSUBPART$' s obj# 240
found TABSUBPART$' s dataobj#:240,ts#:0,file#:1,block#:3473,tab#:0
found INDSUBPART$' s obj# 245
found INDSUBPART$' s dataobj#:245,ts#:0,file#:1,block#:3553,tab#:0
found IND$' s obj# 19
found IND$' s dataobj#:2,ts#:0,file#:1,block#:49,tab#:3
found LOB$' s obj# 156
found LOB$' s dataobj#:2,ts#:0,file#:1,block#:49,tab#:6
found LOBFRAG$' s obj# 258
found LOBFRAG$' s dataobj#:258,ts#:0,file#:1,block#:3761,tab#:0
```

4) 获取 TEST 用户下的 T1 表，也就是我们要恢复的表的信息：

```
ODU> desc test.t1

Object ID:33547
Storage(Obj#=33547 DataObj#=33549 TS#=11 File#=10 Block#=1400 Cluster=0)

NO.  SEG INT Column Name                    Null?     Type
---- --- --- ------------------------------ --------- ------------------------------
  1    1   1 OWNER                                    VARCHAR2(30)
  2    2   2 OBJECT_NAME                              VARCHAR2(128)
  3    3   3 SUBOBJECT_NAME                           VARCHAR2(30)
  4    4   4 OBJECT_ID                                NUMBER
  5    5   5 DATA_OBJECT_ID                           NUMBER
  6    6   6 OBJECT_TYPE                              VARCHAR2(18)
  7    7   7 CREATED                                  DATE
  8    8   8 LAST_DDL_TIME                            DATE
  9    9   9 TIMESTAMP                                VARCHAR2(19)
 10   10  10 STATUS                                   VARCHAR2(7)
 11   11  11 TEMPORARY                                VARCHAR2(1)
 12   12  12 GENERATED                                VARCHAR2(1)
 13   13  13 SECONDARY                                VARCHAR2(1)
```

从上面的输出中，可以看到，TEST.T1 表所在的表空间号为 11，数据段头部为 10 号文件的 1400 号块。

5) 接下来用 ODU 扫描表空间的 extent：

```
ODU> scan extent tablespace 11

scanning extent…
scanning extent finished.
```

6) 我们使用 ODU 来确定 T1 表原来的 data object id。一般来说，数据段的数据块，一般是在段头后面相邻的块中。我们可以从段头的 dump 信息中来确认：

```
ODU> dump datafile 10 block 1400
Block Header:
block type=0x23 (ASSM segment header block)
block format=0x02 (oracle 8 or 9)
block rdba=0x02800578 (file#=10, block#=1400)
scn=0x0000.00286f2d, seq=4, tail=0x6f2d2304
block checksum value=0x0=0, flag=0
Data Segment Header:
  Extent Control Header
  -------------------------------------------------------------
```

```
Extent Header:: extents: 1  blocks: 5
                last map: 0x00000000  #maps: 0  offset: 668
    Highwater:: 0x02800579  (rfile#=10,block#=1401)
                ext#: 0  blk#: 3   ext size:5
    #blocks in seg. hdr's freelists: 0
    #blocks below: 0
    mapblk: 0x00000000   offset: 0
--------------------------------------------------------
Low HighWater Mark :
    Highwater::  0x02800579  ext#: 0      blk#: 3      ext size: 5
#blocks in seg. hdr's freelists: 0
#blocks below: 0
mapblk  0x00000000  offset: 0
Level 1 BMB for High HWM block: 0x02800576
Level 1 BMB for Low HWM block: 0x02800576
--------------------------------------------------------
Segment Type: 1 nl2: 1      blksz: 2048   fbsz: 0
L2 Array start offset:  0x00000434
First Level 3 BMB:  0x00000000
L2 Hint for inserts:  0x02800577
Last Level 1 BMB:  0x02800576
Last Level 1I BMB:  0x02800577
Last Level 1II BMB:  0x00000000
   Map Header:: next  0x00000000  #extents: 1    obj#: 33549  flag: 0x220000000
Extent Map
------------------------------------------------------------

 0x02800576  length: 5

Auxillary Map
------------------------------------------------------------
 Extent 0      :  L1 dba:  0x02800576 Data dba:  0x02800579
------------------------------------------------------------

 Second Level Bitmap block DBAs
------------------------------------------------------------
 DBA 1:   0x02800577
```

从上面的输出中的“Extent 0 : L1 dba: 0x02800576 Data dba: 0x02800579”可以看到，段的第 1 个数据块的 RDBA 为 0x02800579，也就是 10 号文件的 1401 块。

dump 第 10 号文件的 1401 块头，来得到表 T1 原来的 data object id：

```
ODU> dump datafile 10 block 1401 header
Block Header:
```

```
block type=0x06 (table/index/cluster segment data block)
block format=0x02 (oracle 8 or 9)
block rdba=0x02800579 (file#=10, block#=1401)
scn=0x0000.00285f2b, seq=2, tail=0x5f2b0602
block checksum value=0x0=0, flag=0
Data Block Header Dump:
 Object id on Block? Y
 seg/obj: 0x830b=33547  csc: 0x00.285f21  itc: 3  flg: E  typ: 1 (data)
     brn: 0  bdba: 0x2800576 ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0xffff.000.00000000  0x00000000.0000.00  C---    0  scn 0x0000.00285f21
0x02   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
0x03   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
Data Block Dump:
================
flag=0x0 --------
ntab=1
nrow=16
frre=-1
fsbo=0x32
ffeo=0x145
avsp=0x113
tosp=0x113
```

可以看到，T1 表原来的 data object id 就是 33547。

7) 使用 ODU 来 unload 数据：

```
ODU> unload table test.t1 object 33547

Unloading table: T1,object ID: 33547
Unloading segment,storage(Obj#=33547 DataObj#=33547 TS#=11 File#=10 Block#=1400 Cluster=0)
```

8) 将 TEST 表空间 ONLINE：

```
SQL> alter tablespace test online;
```

9) 使用 SQL*Loader（sqlldr）导入恢复的数据：

```
E:\ODU\data>sqlldr test/test control=TEST_T1.ctl
```

至此，恢复数据的步骤已经完成。对比一下数据，看看数据是否和被 Truncate 前的数据完全一样：

```
SQL> select * from t2 minus select * from t1;

no rows selected

SQL> select * from t1 minus select * from t2;

no rows selected
```

可以看到，数据已经完全恢复。

使用 ODU 恢复被 Drop 掉的表

虽然 10g 及以上版本的 Oracle 数据库，提供了 recyclebin（回收站）功能，可以找回被 drop 的表。但是仍然存在着很多 8i、9i 的库以及没有开启 recyclebin 功能、drop 时直接 purge 操作等，这样的情况下，如果想找回被意外 drop 的表，常规的手段是通过备份来恢复。如果没有备份或者没有有效备份，那就没有办法恢复了。不过 ODU 提供了相应的功能，在没有备份的情况下，恢复被 drop 表的数据。

下面通过一个示例来演示如何使用 ODU 来恢复被 drop 的表。

1) 首先创建一个测试表：

```
SQL> create table odu_test ( a number,b varchar2(10),c nvarchar2(30),d varchar2(20),e date,f timestamp,g binary_float,h binary_double);

Table created.

SQL> insert into odu_test select rownum,lpad('x',10),'NC 测试' || rownum, 'ZHS 测试' || rownum,sysdate+dbms_random.value(0,100),systimestamp+dbms_random.value(0,100),rownum+dbms_random.value(0,10000),rownum+dbms_random.value(0,10000) from dba_objects where rownum<=10000;

10000 rows created.

SQL> commit;

Commit complete.

SQL> create table t1 as select * from odu_test;

Table created.
SQL> drop table odu_test purge;

Table dropped.
```

2) OFFLINE 被 DROP 表所在的表空间。

在发现重要的表被意外 drop 掉的时候，应该立即停止应用，offline 那个表所在的表空间或关闭数据库。这里 odu_test 表是建在 users 表空间下，先将 users 表空间 offline：

```
SQL> alter tablespace users offline;

Tablespace altered.
```

3) 然后需要使用 logminer 来查找被 drop 表的 data object id：

```
SQL> select group#,status from v$log;

    GROUP# STATUS
---------- ----------------
         1 INACTIVE
         2 INACTIVE
         3 CURRENT

SQL> col member for a50
SQL> select member from v$logfile where group#=3;

MEMBER
--------------------------------------------------
/u01/app/oracle/oradata/xty/redo03.log

SQL> exec sys.dbms_logmnr.add_logfile(logfilename=>'/u01/app/oracle/oradata/xty/redo03.log');

PL/SQL procedure successfully completed.

SQL> exec sys.dbms_logmnr.start_logmnr(options=>sys.dbms_logmnr.dict_from_online_catalog);

PL/SQL procedure successfully completed.

SQL> select scn,timestamp,sql_redo from v$logmnr_contents where operation='DDL' and sql_redo
like '%odu_test%' order by 2 ;

SCN TIMESTAMP SQL_REDO
---------- ------------------- --------------------------------------------------------------------
    681455 2009-05-08 11:20:50 create table odu_test ( a number,b varchar2(10),c nvarchar2(30),d
varc
                               har2(20),e date,f timestamp,g binary_float,h binary_double);
```

```
    681521 2009-05-08 11:21:17 create table t1 as select * from odu_test;
    681567 2009-05-08 11:21:34 drop table odu_test purge;

SQL> select scn,timestamp,sql_redo from v$logmnr_contents where timestamp=to_date('2009-05-08
11:21:34','yyyy-mm-dd hh24:mi:ss') order by 1;

       SCN  SQL_REDO
----------  --------------------------------------------------------------------
    681566  set transaction read write;
    681567  drop table odu_test purge;
    681569  Unsupported
    681570  Unsupported
    681570
    681570
    681570
    681570  Unsupported
    681570
    681570
    681570
    681570  Unsupported
    681570
    681570
    681570
    681570  Unsupported
    681570
    681570
    681570
    681570
    681570
    681570
    681570  Unsupported
    681570  Unsupported
    681570
    681570
    681570
    681570  Unsupported
    681570
    681570
    681570
    681570  Unsupported
    681570
    681570
    681570
    681571  Unsupported
```

```
    681572
    681572  delete from "SYS"."OBJ$" where "OBJ#" = '52230' and "DATAOBJ#" = '5223
            0' and "OWNER#" = '57' and "NAME" = 'ODU_TEST' and "NAMESPACE" = '1' a
            nd "SUBNAME" IS NULL and "TYPE#" = '2' and "CTIME" = TO_DATE('2009-05-
            08 11:20:46', 'yyyy-mm-dd hh24:mi:ss') and "MTIME" = TO_DATE('2009-05-
            08 11:20:46', 'yyyy-mm-dd hh24:mi:ss') and "STIME" = TO_DATE('2009-05-
            08 11:20:46', 'yyyy-mm-dd hh24:mi:ss') and "STATUS" = '1' and "REMOTEO
            WNER" IS NULL and "LINKNAME" IS NULL and "FLAGS" = '0' and "OID$" IS N
            ULL and "SPARE1" = '6' and "SPARE2" = '1' and "SPARE3" IS NULL and "SP
            ARE4" IS NULL and "SPARE5" IS NULL and "SPARE6" IS NULL and ROWID = 'A
            AAAASAABAAAMzdAAS';

    681572
    681573  commit;
    681574  set transaction read write;
    681574  Unsupported
    681576  commit;
    681577  set transaction read write;
    681579  Unsupported
    681581  commit;

SQL> exec sys.dbms_logmnr.end_logmnr;

PL/SQL procedure successfully completed.
```

从 SCN 为 681572 的几行中，delete from "SYS"."OBJ$" where "OBJ#" = '52230' and "DATAOBJ#" = '52230' 可以看到被 drop 表的 data object id 为 52230。

4) 下面使用 ODU 来恢复这个被删除的表：

```
[oracle@xty odu]$ ./odu

Oracle Data Unloader:Release 2.6.0

Copyright (c) 2008,2009 XiongJun. All rights reserved.

Web: http://www.laoxiong.net
Email: magic007cn@gmail.com

loading default config…….

 ts#   fn  rfn bsize   blocks bf offset filename
---- ---- ---- ----- -------- -- ------ ----------------------------------------
   0    1    1  8192    62720 N       0 /u01/oradata/xty/system01.dbf
```

```
   1    2    2  8192    26240 N       0 /u01/oradata/xty/undotbs01.dbf
   2    3    3  8192    32000 N       0 /u01/oradata/xty/sysaux01.dbf
   4    4    4  8192      800 N       0 /u01/oradata/xty/users01.dbf
load control file ‘control.txt’ successful
loading dictionary data……
```

这里假设不知道这个表有多少列，每个列的数据类型，我们可以通过 ODU 的抽样来自动判断数据的类型：

```
ODU> scan extent tablespace 4;

scanning extent…
scanning extent finished.

ODU> unload object 52230 sample

Unloading Object,object ID: 52230, Cluster: 0
output data is in file : ‘data/ODU_ODU_0000052230.txt’

Sample result:
  object id: 52230
  tablespace no: 4
  sampled 1056 rows
  column count: 8
  column    1  type: NUMBER
  column    2  type: VARCHAR2
  column    3  type: NVARCHAR2
  column    4  type: VARCHAR2
  column    5  type: DATE
  column    6  type: DATE
  column    7  type: BINARY_FLOAT
  column    8  type: BINARY_DOUBLE

COMMAND:
unload object 52230 tablespace 4 column NUMBER VARCHAR2 NVARCHAR2 VARCHAR2 DATE DATE BINARY_FLOAT
BINARY_DOUBLE
```

可以看到，ODU 比较准确地判断出了列类型，甚至连 NVARCHAR 类型都判断出来了。只是由于测试数据的原因，TIMESTAMP 那一列按 DATE 类型进行了存储（只有 7 字节长），所以被判断成了 DATE 类型，但是在这里不影响数据的恢复。从输出的内容可以看到，可以在 'data/ODU_ODU_0000052230.txt' 中看到抽样的数据，同时可以在'data/sample.txt'中看到更详细的抽样输出。
现在我们用 ODU 来恢复数据：

```
ODU> unload object 52230 tablespace 4 column NUMBER VARCHAR2 NVARCHAR2 VARCHAR2 DATE DATE
BINARY_FLOAT BINARY_DOUBLE

Unloading Object,object ID: 52230, Cluster: 0
```

5) ONLINE USERS 表空间，导入恢复的数据。

首先修改一下生成的 SQL 文件'ODU_ODU_0000052230.sql'，并创建表：

```
SQL> CREATE TABLE "TEST"."T2"
  2  (
  3      "C0001" NUMBER ,
  4      "C0002" VARCHAR2(4000) ,
  5      "C0003" NVARCHAR2(2000) ,
  6      "C0004" VARCHAR2(4000) ,
  7      "C0005" DATE ,
  8      "C0006" DATE ,
  9      "C0007" BINARY_FLOAT ,
 10      "C0008" BINARY_DOUBLE
 11  );

Table created.
```

修改一下生成的 ODU_ODU_0000052230.ctl 文件中导入数据的用户名和表名，然后使用 SQL*Loader（sqlldr）导入数据：

```
export NLS_LANG=american_america.zhs16gbk
[oracle@xty data]$ sqlldr test/test control=ODU_ODU_0000052230.ctl

SQL*Loader: Release 10.2.0.4.0 - Production on Fri May 8 12:19:34 2009

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

Commit point reached - logical record count 630
Commit point reached - logical record count 1260
Commit point reached - logical record count 1890
Commit point reached - logical record count 2520
Commit point reached - logical record count 3150
Commit point reached - logical record count 3780
Commit point reached - logical record count 4410
Commit point reached - logical record count 5040
Commit point reached - logical record count 5670
Commit point reached - logical record count 6300
Commit point reached - logical record count 6930
Commit point reached - logical record count 7560
```

```
Commit point reached - logical record count 8190
Commit point reached - logical record count 8820
Commit point reached - logical record count 9450
Commit point reached - logical record count 10000
```

对比数据，数据完全匹配。如果数据导出为 DMP 文件格式，则不会受精度影响。至此数据已经完全恢复。

第四章　ODU 命令参考

本章将介绍 ODU 支持的命令，并详细描述 ODU 的命令格式及用法。

- ODU 采用 CLI（Command Line Interface）用于各种操作。ODU 支持多个不同模块的命令，目前支持的模块为 ODU、BBED 和 ASMCMD，这三种模块显示的命令行提示符分别为 ODU>、BBED>和 ASMCMD>。而大多数恢复数据的命令是在 ODU>提示符下执行，而其他的命令模块和提示符主要是一些辅助功能。
- ODU 的命令以回车为结束，而不是像 Sqlplus 那样的使用“分号”，即“;”
- ODU 的命令不区分大小写，只有在使用 unload 命令恢复数据时，如果用户名或表名使用了双引号，则用户名和表名会严格使用双引号内的文字，包括空格。比如 unload table sys.t2 表示恢复 SYS 用户下的 T2 表，而没有使用双引号时用户名及表名全部会转换为大写；unload table sys."t2"表示恢复 SYS 用户下的 t2 表。
- ODU 的命令由一个命令名加 0 个或多个关键字及参数组成。在命令格式中，尖括号中内容表示需要根据实际情况输入的参数，方括号中的关键字和参数表示可选。比如 unload table <schema.tablename> [object truncate] [partition <partition_name>]，<schema.tablename>表示必须的参数，object truncate 表示可选的关键字，而 partiion <partition_name>表示可选的关键字和参数，其中 partition 是关键字，partition_name 是参数。

unload 命令

unload 命令是 ODU 中命令格式最复杂的命令，这是由于其强大的功能和灵活性所决定的，但它也是恢复数据最关键的命令。命令的格式如下：

```
unload dict [block <bootstrap block#>]
unload table <schema.tablename> [object truncate] [partition <partition_name>]
unload table <schema.tablename> [object scanned] [partition <partition_name>]
unload table <schema.tablename> object <data_obj_id> [tablespace <ts_no>]
unload table <schema.tablename> datafile <rfile#> block <block#> [blocks <blocks>] [partition
<partition_name>]
unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>]  column <type [ type
[ type......]>
    type: VARCHAR2 VARCHAR CHAR NUMBER SKIP LONG RAW
          DATE LONG_RAW TIMESTAMP TIMESTAMP_TZ TIMESTAMP_LTZ
          BINARY_FLOAT BINARY_DOUBLE NVARCHAR2 NCHAR
          CLOB NCLOB BLOB
unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>]  sample
unload object all [tablespace <ts_no>] sample
unload user <schema name>
```

unload 命令恢复的数据保存在 ODU 参数 data_path 所指定的目录下，缺省为 ODU 软件目录

的 data 子目录下。
unload 命令恢复出来的数据，分为 2 种格式，由 ODU 参数 output_format 指定，该参数值为 dmp 时，以 Oracle 传统的 exp 工具生成的 dmp 格式保存，dmp 版本为 Oracle 8.0，高版本的 imp 命令能够导入低版本的 dmp 文件。output_format 参数值为 text 时，恢复的数据以文本文件格式保存，同时生成对应的建表 SQL 文件和 SQL*Loader（sqlldr）工具导入数据所需的控制文件。每一个表恢复出来保存为单独的文件。
unload 命令在恢复数据时，分为有数据字典和没有数据字典两种情况。在有数据字典时，恢复时保存数据的文件命名为"<用户名>_<表名>"，而没有数据字典时，保存数据的文件命名为"ODU_<data_object_id>"。
unload table 和 unload user 用于恢复有数据字典情况下的数据恢复，而 unload object 用于没有数据字典情况下的数据恢复。

以下是这些命令的详细说明：

unload dict

unload dict 用于从 SYSTEM 表空间解析数据字典，并将 ODU 所需要的数据字典数据保存到 user.odu,tab.odu,obj.odu,col.odu,ind.odu,lob.odu,lobfrag.odu 等文件中。这样，ODU 就可以利用数据字典信息恢复数据，能够自动生成与原表相同的创建表的 SQL 语句以及 SQL*Loader（sqlldr）控制文件和 dmp 文件。对于在有数据字典的情况下，ODU 的功能能够得到最大程度地发挥，同时也能大大减轻数据恢复的工作量。

这条命令的示例如下：

```
ODU> unload dict
CLUSTER C_USER# file_no: 1 block_no: 89
TABLE OBJ$ file_no: 1 block_no: 121
CLUSTER C_OBJ# file_no: 1 block_no: 25
CLUSTER C_OBJ# file_no: 1 block_no: 25
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
found TABPART$'s obj# 266
found TABPART$'s dataobj#:266,ts#:0,file#:1,block#:2121,tab#:0
found INDPART$'s obj# 271
found INDPART$'s dataobj#:271,ts#:0,file#:1,block#:2161,tab#:0
found TABSUBPART$'s obj# 278
found TABSUBPART$'s dataobj#:278,ts#:0,file#:1,block#:2217,tab#:0
found INDSUBPART$'s obj# 283
found INDSUBPART$'s dataobj#:283,ts#:0,file#:1,block#:2257,tab#:0
found IND$'s obj# 19
found IND$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:3
found LOB$'s obj# 151
found LOB$'s dataobj#:2,ts#:0,file#:1,block#:25,tab#:6
found LOBFRAG$'s obj# 299
```

```
found LOBFRAG$'s dataobj#:299,ts#:0,file#:1,block#:2393,tab#:0
```

值得注意的是，ODU 并不是从文件号为 1 的数据文件中得到 bootstrap$地址进而得到数据字典，而是从 ODU 控制文件的第一行指定的文件中得到 bootstrap$地址。所以，需要将 SYSTEM 表空间中的第 1 个数据文件放置于 control.txt 中的第 1 行。否则会得到如下的错误：

```
can not get bootstrap$ address from SYSTEM tablespace
```

得到数据字典后，可以使用 list user, list table 等命令查看数据库中的用户及用户下的表、视图等对象。例如：

```
ODU> list user

     USER#   USERNAME
----------   ------------------------------
        17   GLOBAL_AQ_USER_ROLE
         6   SELECT_CATALOG_ROLE
        35   EXFSYS
         0   SYS
        11   OUTLN
        19   DIP
        25   ORACLE_OCM
        27   WM_ADMIN_ROLE
        34   JAVA_DEPLOY
        36   XDB
        41   TEST
         2   CONNECT
        38   XDBADMIN
        12   RECOVERY_CATALOG_OWNER
         3   RESOURCE
         1   PUBLIC
        37   ANONYMOUS
        20   HS_ADMIN_ROLE
        30   JAVASYSPRIV
        22   OEM_ADVISOR
         5   SYSTEM
        10   IMP_FULL_DATABASE
        40   XDBWEBSERVICES
        29   JAVAIDPRIV
        23   OEM_MONITOR
        18   SCHEDULER_ADMIN
         7   EXECUTE_CATALOG_ROLE
        33   JAVA_ADMIN
         4   DBA
        24   DBSNMP
```

```
        16   AQ_USER_ROLE
        28   JAVAUSERPRIV
        39   AUTHENTICATEDUSER
        32   EJBCLIENT
        21   TSMSYS
        15   AQ_ADMINISTRATOR_ROLE
        14   LOGSTDBY_ADMINISTRATOR
        42   _NEXT_USER
        13   GATHER_SYSTEM_STATISTICS
        31   JAVADEBUGPRIV
         9   EXP_FULL_DATABASE
        26   WMSYS
         8   DELETE_CATALOG_ROLE

ODU> list table outln

      OBJ#   OBJECT_NAME
----------   ------------------------------
       452   OL$
       453   OL$HINTS
       456   OL$NODES
```

unload table <schema .tablename> [partition <partition_name>]

这个命令格式用于导出某个用户下的某张表，如果指定了 partition，则只导出分区表中的那个分区。这里 partition_name 对于简单分区表为分区表，而对于复合分区表，只能为子分区名。
这条命令的示例如下：

```
ODU> unload table sys.t1

Unloading table: T1,object ID: 42138 at 2011-04-08 01:08:03
Unloading segment,storage(Obj#=42138 DataObj#=42138 TS#=4 File#=4 Block#=11 Cluster=0)
41161 rows unloaded
At 2011-04-08 01:08:05
```

导出表时，会显示表名，对象 ID(Object ID)，以及导出段的段头、开始和结束时间等信息。

如果用户名或表名使用了双引号，则用户名和表名会严格使用双引号内的文字，包括空格。比如 unload table sys.t2 表示恢复 SYS 用户下的 T2 表；unload table sys."t2"表示恢复 SYS 用户下的 t2 表。而没有使用双引号时用户名及表名全部会转换为大写。下面一个示例可以对比出差距：

```
ODU> unload table sys.t2
table 'sys.t2' does not exist.

ODU> unload table sys."t2"

Unloading table: t2,object ID: 42149 at 2011-04-08 01:13:43
Unloading segment,storage(Obj#=42149 DataObj#=42149 TS#=0 File#=1 Block#=43001 Cluster=0)
100 rows unloaded
At 2011-04-08 01:13:43
```

可以看到使用 unload table sys.t2 时不能恢复数据，这是由于数据库中 SYS 用户下不存在 T2 这张表，而只存在 t2 这张表（注意表名大小写区别）。

unload table <schema.tablename> [object truncate] [partition <partition_name>]

此条命令与上一命令格式相比，多了 object truncate 这个选项。这条命令用于恢复被 Truncate 的表。执行这条命令前需要用 scan extent 命令扫描数据文件。例如 SYS.T3 表被 Truncate，现在需要恢复这张表：

```
ODU> unload table sys.t3

Unloading table: T3,object ID: 42150 at 2011-04-08 01:28:22
Unloading segment,storage(Obj#=42150 DataObj#=42151 TS#=0 File#=1 Block#=43009 Cluster=0)
0 rows unloaded
At 2011-04-08 01:28:22
```

可以看到恢复的数据为 0 行，即没有成功恢复数据。

```
ODU> scan extent tablespace 0

scan extent start: 2011-04-08 01:29:11
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 01:29:34

ODU> unload table sys.t3 object truncate
Auto mode truncated table.

Unloading table: T3,object ID: 42150 at 2011-04-08 01:29:51
Unloading segment,storage(Obj#=42150 DataObj#=42150 TS#=0 File#=1 Block#=43009 Cluster=0)
5000 rows unloaded
At 2011-04-08 01:29:51
```

而这次则成功地恢复出了被 Truncate 的表 SYS.T3 的数据，数据量共 5000 行。


提示：
使用这条命令恢复被 Truncate 的表数据时，ODU 会自动搜索表的段头开始的几个数据块并判断是否是 Truncate 之前的数据，如果判断是则会恢复出搜索到的块以及之后的数据，否则便会认为 Truncate 的表已经被部分覆盖。在这种情况下，需要使用下一条命令来恢复。

unload table <schema.tablename> object <data_obj_id> [tablespace <ts_no>]

此命令格式与命令格式 unload table <schema.tablename> [object truncate] [partition <partition_name>]的差别在于，在导出的表名之后指定了数据对象 ID（data object id)，这用于导出表的实际 data object id 与数据字典中表的 data object id 不一致时的表数据。简单点说，就是被 Truncate 的表不能使用 unload table <schema.tablename> [object truncate] [partition <partition_name>]命令格式恢复时，使用这一命令来恢复。执行这条命令前需要先用 scan extent 命令扫描数据文件。

在同一数据库中，在使用了传输表空间时，不同段对象的 data object id 可能相同，在这种情况下需要指定要恢复的数据所在的表空间号（注意这里是表空间号而不是表空间名）。同一表空间下不同的数据段不可能存在相同的 data object id。
这一命令的使用示例如下：

```
ODU> scan extent tablespace 0

scan extent start: 2011-04-08 01:44:12
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 01:44:12

ODU> unload table sys.t3 object 42150

Unloading table: T3,object ID: 42150
Unloading segment,storage(Obj#=42150 DataObj#=42150 TS#=0 File#=1 Block#=43009 Cluster=0) at
2011-04-08 01:44:29
5000 rows unloaded
At 2011-04-08 01:44:29
```

这条命令与 unload table <schema.tablename> [object truncate] [partition <partition_name>]命令格式虽然都能恢复 Truncate 表的数据，但是值得注意的是，对于分区表，每一个分区或子分区都有一个数据段，即每一个分区或子分区都有一个 data object id，如果使用这条命令恢复 Truncate 的分区表，需要对每一个分区或子分区执行这一格式的命令，只是每条命令的 data object id 有所不同。


提示：
表被 Truncate 之后，表的数据实际上并没有被删除，只是高水位线被缩回到了段头，空间被回收，同时 data object id 增加。使用此命令格式恢复 Truncate 表的数据，关键是需要知道 Truncate 之前的 data object id，如果表之前没有被 Truncate 或 Move 过，其之前的 data object id 应该与 data object id 相同，否则需要通过闪回查询或日志挖掘（logminer）来获取 Truncate 之前的 data object id。

unload table <schema.tablename> [object scanned] [partition <partition_name>]

这条命令与 1.2 格式的命令一样，都是用于恢复某一用户下某一张表，或表下的分区和子分区。二者之间的区别在于，这条命令用于恢复表的关键数据块有损坏的情况下的数据。这些关键数据块包括段头、Extent Map Block。这几类数据块损坏时，Oracle 不能正确地扫描到表的所有数据。
在使用这条命令恢复数据之前，需要先使用 scan extent 命令扫描数据文件。
使用举例：
假如 SYS.T3 表的段头或 Extent Map Block 损坏，需要恢复这张表：

```
ODU> scan extent tablespace 0

scan extent start: 2011-04-08 13:19:29
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 13:19:53

ODU> unload table sys.t3 object scanned
Using scanned extent.

Unloading table: T3,object ID: 42158 at 2011-04-08 13:20:09
Unloading segment,storage(Obj#=42158 DataObj#=42158 TS#=0 File#=1 Block#=43009 Cluster=0)
20000 rows unloaded
At 2011-04-08 13:20:10
```

unload table <schema.tablename> datafile <rfile#> block <block#> [blocks <blocks>] [partition <partition_name>]

这条命令用于恢复一张表下指定位置的指定数量的块。主要用于当部分块异常时只恢复这部分数据块的数据。

datafile <rfile#> block <block#>参数指定了要恢复的起始块。
blocks <blocks>指定要恢复的块数量，如果不指定则为 1。

例如：

```
ODU> unload table sys.t1 datafile 4 block 651 blocks 10
unload specific block mode.

Unloading table: T1,object ID: 42138 at 2011-04-08 18:01:23
Unloading segment,storage(Obj#=42138 DataObj#=42138 TS#=4 File#=4 Block#=11 Cluster=0)
752 rows unloaded
At 2011-04-08 18:01:23
```

unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>] column <type [type [type.....]>

这条命令简单点说，就是用于没有 SYSTEM 表空间或者没有数据字典时的数据恢复，或用于恢复被 DROP 了的表。

这条命令中，列类型为以下：

```
VARCHAR2 VARCHAR CHAR NUMBER SKIP LONG RAW
 DATE LONGRAW TIMESTAMP TIMESTAMPTZ TIMESTAMPLTZ
 BINARY_FLOAT BINARY_DOUBLE NVARCHAR2 NCHAR
 CLOB NCLOB BLOB
```

SKIP 表示不恢复那个列。恢复前需要先用 scan extent 命令扫描数据文件。

在同一数据库中，在使用了传输表空间时，不同段对象的 data object id 可能相同，在这种情况下需要指定要恢复的数据所在的表空间号（注意这里是表空间号而不是表空间名）。同一表空间下不同的数据段不可能存在相同的 data object id。

cluster <cluster_no>参数用于在恢复 Cluster 表时，指定需要恢复的表在其 Cluster 中所属的表编号。


提示：
使用这条命令恢复数据时，所指定的列顺序以数据存储在数据块上的顺序为准，在通常情况下二者是一致的，但是在以下两种情况中可能会存在不一致的情况，一是 Cluster 表，Cluster 列总是存储在最前面；二是表中有 LONG 或 LONG RAW 类型的列，这两种类型的列总是存储在最后一列。

下面是一个示例：

首先创建一个测试表，并插入一行数据，最后将表删除：

```
SQL> create table t4 (a number, b long,c varchar2(100));

Table created.

SQL> insert into t4 values (1,'test long','test varchar2');

1 row created.

SQL> commit;
SQL> select object_name,data_object_id,object_id from dba_objects where owner=user and object_name='T4';
OBJECT_NAME                    DATA_OBJECT_ID  OBJECT_ID
------------------------------ -------------- ----------
T4                                      42168      42168
SQL> drop table t4;
```

然后使用 ODU 来恢复这张已经被删除的表：

```
ODU> scan extent tablespace 0

scan extent start: 2011-04-08 17:24:52
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 17:25:15

ODU> unload object 42168 column number varchar2 long

Unloading Object,object ID: 42168,  Cluster: 0 at 2011-04-08 17:26:12
1 rows unloaded
At 2011-04-08 17:26:12

[oracle@xty data]$ cat ODU_0000042168.txt
1|test varchar2|test long
```

注意恢复时指定的列类型顺序与建表时有所不同，这是由于 LONG 类型的列在存储时总是会调整成最后一列。

提示：

使用这条命令恢复被 DROP 掉的表时，需要知道表的 data object id 以及列类型。data object id 可以通过闪回查询数据字典或通过日志挖掘（logminer）来获取。列类型可以从开发及测试库、冷备份等来源得到，如果不能得到，可以通过下面将要介绍的 unload object sample 命令来让 ODU 自动判断。

unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>] sample

这一格式的命令与上一条命令比较接近，主要用于没有数据字典信息时（比如表被删除）通过采样分析来自动判断表的列类型。通过得到的列类型，再使用上一条命令来真正用于恢复数据，这条命令是一条辅助型命令，在使用前需要使用 scan extent 扫描数据文件。示例如下：

```
ODU> unload object 42168 sample

Unloading Object,object ID: 42168,  Cluster: 0
output data is in file :  'data/ODU_0000042168.txt'

Sample result:
  object id: 42168
  tablespace no: 0
  sampled 1 rows
  column count: 3
  column    1  type: NUMBER
  column    2  type: VARCHAR2
  column    3  type: VARCHAR2

COMMAND:
unload object 42168 tablespace 0 column NUMBER VARCHAR2 VARCHAR2
```

从输出可以看到，这条命令产生了如下的结果：

- ✧ 抽样得到的表数据存储在 ODU 软件目录下的 data/ODU_0000042168.txt 文件中。检查这个文件的内容可以确认其数据是否正是需要恢复的数据，同时确认 ODU 自动识别的列类型是否正确。
- ✧ 抽样得到的数据有 1 条，列数有 3 列。并分别列出了三列的列类型。
- ✧ 列出了恢复这张表的数据时所需要使用的命令。
- ✧ 上面显示的输出内容同时保存在了 ODU 软件目录的 data/sample.txt 下（这个路径由 ODU 参数 data_path 控制）。在 sample.txt 中还记录了每个表抽样产生的前 5 条数据。

这里第 3 列实际应该为 LONG 类型，产生这样的结果的原因在于，由于数据量太少，同时该列的长度在 VARCHAR2 型最大长度（4000 字节）之内。当然如果所有数据的该列长度在 4000 字节以内时，使用 VARCHAR2 类型仍然能够恢复出数据来。LONG 类型可以理解为更长的 VARCHAR2 型。

unload object all [tablespace <ts_no>] sample

这条命令用于在系统表空间丢失或损坏时，也就是没有数据字典信息时，抽样数据库中的所有数据，以自动判断所有表的列个数及列类型。这也是一条辅助型命令，使用前需要使用 scan extent 扫描数据文件。

tablespace <ts_no>参数限制要抽样的表空间。

这条命令的使用示例如下：

```
ODU> scan extent tablespace 4

scan extent start: 2011-04-08 17:50:25
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 17:50:25

ODU> unload object all tablespace 4 sample

Unloading Object,object ID: 42169,  Cluster: 0
output data is in file :  'data/ODU_0000042169.txt'

Sample result:
  object id: 42169
  tablespace no: 4
  sampled 1056 rows
  column count: 13
  column    1  type: VARCHAR2
  column    2  type: VARCHAR2
  column    3  type: RAW
  column    4  type: NUMBER
  column    5  type: NUMBER
  column    6  type: VARCHAR2
  column    7  type: DATE
  column    8  type: DATE
  column    9  type: VARCHAR2
  column   10  type: VARCHAR2
  column   11  type: VARCHAR2
  column   12  type: VARCHAR2
  column   13  type: VARCHAR2

COMMAND:
unload object 42169 tablespace 4 column VARCHAR2 VARCHAR2 RAW NUMBER NUMBER VARCHAR2 DATE DATE
VARCHAR2 VARCHAR2 VARCHAR2 VARCHAR2 VARCHAR2



Unloading Object,object ID: 42170,  Cluster: 0
output data is in file :  'data/ODU_0000042170.txt'

Sample result:
  object id: 42170
  tablespace no: 4
  sampled 1053 rows
```

```
  column count: 13
  column    1  type: VARCHAR2
  column    2  type: VARCHAR2
  column    3  type: RAW
  column    4  type: NUMBER
  column    5  type: NUMBER
  column    6  type: VARCHAR2
  column    7  type: DATE
  column    8  type: DATE
  column    9  type: VARCHAR2
  column   10  type: VARCHAR2
  column   11  type: VARCHAR2
  column   12  type: VARCHAR2
  column   13  type: VARCHAR2

COMMAND:
unload object 42170 tablespace 4 column VARCHAR2 VARCHAR2 RAW NUMBER NUMBER VARCHAR2 DATE DATE
VARCHAR2 VARCHAR2 VARCHAR2 VARCHAR2 VARCHAR2
```

这条命令的输出与上一条命令相同，只是会输出多个表的抽样结果。

提示：
使用这条命令自动判断和发现数据库中的数据时，需要非常熟悉系统的维护人员或开发人员参与恢复，由于数据库中存在大量的垃圾数据，需要由维护人员或开发人员辨别真正需要的数据。

unload user <schema name>

此命令格式用于导出指定用户下的所有表。此命令在有数据字典时才支持。这样简化了导出一个用户下所有的表的操作。

help 命令

help 命令，显示 ODU 支持的命令列表，每个命令后面有简短描述：

```
ODU> help

help      ----   get command list
spool     ----   spool information to file
host      ----   enter os terminal
rowid     ----   decode rowid components
```

```
rdba      ----   decode RDBA to rfile# and block#
time      ----   convert number to timestamp
exit      ----   exit from odu
load config ----   load config information from file
open      ----   load database filename and asm disk list from file
hexdump    ----   dump file format hex
dump      ----   dump oracle datafile block
unload    ----   unload data
scan extent ----   scan extent
scan disk ----   scan asm disk or any disk or disk partition
list      ----   list schema object,partition,datafile
charset   ----   get or list supported charset name
```

在执行一个命令时，如果输入的命令格式不对，将会提示那个命令的正确命令格式。例如：

```
ODU> unload sys.t1
unload dict [block <bootstrap block#>]
unload table <schema.tablename> [object truncate] [partition <partition_name>]
unload table <schema.tablename> [object scanned] [partition <partition_name>]
unload table <schema.tablename> object <data_obj_id> [tablespace <ts_no>]
unload table <schema.tablename> datafile <rfile#> block <block#> [blocks <blocks>] [partition
<partition_name>]
unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>]  column <type [ type
[ type......]>
    type: VARCHAR2 VARCHAR CHAR NUMBER SKIP LONG RAW
          DATE LONG_RAW TIMESTAMP TIMESTAMP_TZ TIMESTAMP_LTZ
          BINARY_FLOAT BINARY_DOUBLE NVARCHAR2 NCHAR
          CLOB NCLOB BLOB
unload object <data_obj_id> [tablespace <ts_no>] [cluster <cluster_no>]  sample
unload object all [tablespace <ts_no>] sample
unload user <schema name>
```

不同的模块，有不同的命令支持列表，比如在 ASMCMD 模块中，使用 help 的结果如下：

```
ODU> asmcmd

Entering asmcmd module.

ASMCMD> help

help      ----   get command list
spool     ----   spool information to file
host      ----   enter os terminal
rowid     ----   decode rowid components
rdba      ----   decode RDBA to rfile# and block#
time      ----   convert number to timestamp
```

```
exit      ----   exit from asmcmd module
dump      ----   dump asm disk block
extract   ----   extract file from asm disk
```

load config 命令

load config 命令用于载入 ODU 的配置文件。命令格式如下：

```
load config [filename]
```

缺省的文件名是 config.txt
ODU 在启动时，会自动载入名为 config.txt 的缺省配置文件。后面的章节中会专门介绍 ODU 的配置文件。

ODU 启动后，也可以重新载入配置文件，载入新配置后，会立即生效。

下面是一个示例：

```
ODU> load config
byte_order little
block_size  8192
block_buffers 1024
db_timezone -7
client_timezone 8
asmfile_extract_path   /asmfile
data_path   data
lob_path    /odu/data/lob
charset_name US7ASCII
ncharset_name AL16UTF16
output_format text
lob_storage infile
clob_byte_order big
trace_level 1
delimiter |
unload_deleted no
file_header_offset 0
is_tru64 no
record_row_addr no
convert_clob_charset yes
use_scanned_lob  yes
trim_scanned_blob yes
lob_switch_dir_rows 20000
db_block_checksum yes
```

```
db_block_checking yes
rdba_file_bits 10
compatible 10
load config file 'config.txt' successful
```


提示：
在执行了 load config 命令之后，建议再执行一次 open 命令，因为部分参数会影响到 open 命令执行时 Oracle 数据文件的自动识别。例如 Oracle 数据文件的块大小为 16384（即 16K）时，使用默认的块大小配置（8K）将使 ODU 不能自动识别数据文件格式，在修改了配置文件中 block_size 为 16384 并使用 load config 命令使配置生效后，需要执行 open 命令使 ODU 能够利用新的配置自动识别数据文件。

open 命令

open 命令打开控制文件和 ASM 磁盘信息文件，这两个文件类似于 Oracle 的控制文件。ODU 控制文件存储了 Oracle 数据库的数据文件的文件名、块大小、数据文件大小、所属表空间、文件号等非常重要的信息，而 ASM 磁盘信息文件存储了磁盘设备文件路径、磁盘组名称、AU 大小等重要信息。这样 ODU 在恢复数据时才能知道从哪个文件或哪个磁盘读取数据。

缺省的控制文件名是 control.txt 和 oductl.dat。control.txt 是在获取 LICENSE 之前使用的控制文件，主要用于生成获取 LICENSE 时所需要提供的数据。而在正式用于数据恢复时，ODU 会使用包含了 LICENSE 信息的控制文件 oductl.dat，这个文件名是固定的，不能通过配置文件和命令更改。

缺省的 ASM 磁盘信息文件是 asmdisk.txt。

open 命令的格式如下：

```
open [control filename [asmdisk filename]]
```

参数 control filename 指的是 ODU 控制文件名，不指定时缺省为 control.txt。
参数 asmdisk filename 指的是 ASM DISK 磁盘信息文件，不指定时缺省为 asmdisk.txt。


提示：
为了简单，通常只需要使用 open 命令即可。

对于控制文件和 ASM 磁盘信息文件的格式，请参见后面的章节。

ODU 启动时，自动加载缺省的控制文件 control.txt 和 oductl.dat 以及缺省的 ASM 磁盘信息文件 asmdisk.txt。ODU 启动后，如果修改了 control.txt、重新获得了 LICENSE 或修改了 asmdisk.txt，则可以使用此命令再次打开控制文件和 ASM 磁盘信息文件以生效。

下面是一个示例：

```
ODU> open

grp# dsk# bsize ausize disksize diskname        groupname       path
---- ---- ----- ------ -------- --------------- --------------- ------------------------------
   1    1  4096  1024K     1024 DGDATA_0001     DGDATA          /oradata/asm/disk1.dbf
   1    0  4096  1024K     1024 DGDATA_0000     DGDATA          /oradata/asm/disk2.dbf
   1    2  4096  1024K     1024 DGDATA_0002     DGDATA          /oradata/asm/disk3.dbf

load asm disk file 'asmdisk.txt' successful

 ts#   fn  rfn bsize   blocks bf offset filename
---- ---- ---- ----- -------- -- ------ ------------------------------------------
   0    1    1  8192    44800 N       0 +DGDATA/xty/datafile/system.260.745630773
   1    2    2  8192    25600 N       0 +DGDATA/xty/datafile/undotbs1.261.745630805
   2    3    3  8192    15360 N       0 +DGDATA/xty/datafile/sysaux.262.745630817
   4    4    4  8192      800 N       0 +DGDATA/xty/datafile/users.264.745630833
load control file 'oductl.dat' successful
```

- 关于 open 命令，同时可以参考 save control 命令。
- 关于 open 命令可能会出现的问题，请参考后面关于 Trouble Shooting 章节。

list 命令

这条命令用于列出数据库中的对象，包括 USER、TABLE、VIEW、表的分区等，能够列出 ODU 使用的数据文件列表。命令格式如下：

```
list user
list <table | view | procedure | function | index | package | sequence> <user_name>
list partition <user_name.table_name>
list datafile
```

例如：

1) 列出数据库中的用户

```
ODU> list user

     USER#   USERNAME
----------   ------------------------------
        17   GLOBAL_AQ_USER_ROLE
         6   SELECT_CATALOG_ROLE
        35   EXFSYS
```

```
         0   SYS
        11   OUTLN
        19   DIP
        25   ORACLE_OCM
        27   WM_ADMIN_ROLE
        34   JAVA_DEPLOY
        36   XDB
        41   TEST
         2   CONNECT
        38   XDBADMIN
        12   RECOVERY_CATALOG_OWNER
         3   RESOURCE
         1   PUBLIC
        37   ANONYMOUS
        20   HS_ADMIN_ROLE
        30   JAVASYSPRIV
        22   OEM_ADVISOR
         5   SYSTEM
        10   IMP_FULL_DATABASE
        40   XDBWEBSERVICES
        29   JAVAIDPRIV
        23   OEM_MONITOR
        18   SCHEDULER_ADMIN
         7   EXECUTE_CATALOG_ROLE
        33   JAVA_ADMIN
         4   DBA
        24   DBSNMP
        16   AQ_USER_ROLE
        28   JAVAUSERPRIV
        39   AUTHENTICATEDUSER
        32   EJBCLIENT
        21   TSMSYS
        15   AQ_ADMINISTRATOR_ROLE
        14   LOGSTDBY_ADMINISTRATOR
        42   _NEXT_USER
        13   GATHER_SYSTEM_STATISTICS
        31   JAVADEBUGPRIV
         9   EXP_FULL_DATABASE
        26   WMSYS
         8   DELETE_CATALOG_ROLE
```

2) 列出 TEST 用户下的所有表：

```
ODU> list table test
```

```
      OBJ#   OBJECT_NAME
----------   ------------------------------
     42252   T1
     42253   T2
```

3) 列出 ODU 所使用的数据文件列表：

```
ODU> list datafile

 ts#   fn  rfn bsize   blocks bf offset filename
---- ---- ---- ----- -------- -- ------ ----------------------------------------
   0    1    1  8192    44800 N       0 +DGDATA/xty/datafile/system.260.745630773
   1    2    2  8192    25600 N       0 +DGDATA/xty/datafile/undotbs1.261.745630805
   2    3    3  8192    16640 N       0 +DGDATA/xty/datafile/sysaux.262.745630817
   4    4    4  8192     1440 N       0 +DGDATA/xty/datafile/users.264.745630833
   6    5    5  8192      100 N       0 +DGDATA/xty/datafile/tbs_test.270.746469239
```

scan 命令

scan 命令用于扫描数据文件中的 segment 以及 extent。主要作用在于没有 SYSTEM 表空间或 SYSTEM 表空间损坏时的数据恢复，以及 TRUNCATE 表和 DROP 表之后的数据恢复。命令格式如下：

```
scan extent [tablespace <ts#> [datafile <rfile#>] ] [object <data_object_id>] [parallel
<parallel_degree>]
```

扫描时可以指定表空间（只能指定表空间号），表空间中的 1 个文件；也可以扫描指定的 data_object_id。扫描完成后，将在 ODU 的运行目录下生成 ext.odu、lobpage.odu 和 lobind.odu 文件以及 segment.txt 文件，后者包含了扫描所找到的段头。

parallel <parallel_degree>指定并行扫描的并行度，在数据库比较大时，使用并行扫描将大大提高扫描速度。目前在 Windows 下运行的版本不支持并行扫描。

下面是一个示例：

```
ODU> scan extent tablespace 4

scan extent start: 2011-04-08 19:03:41
scanning extent...
scanning extent finished.
scan extent completed: 2011-04-08 19:03:41
```

在恢复没有 SYSTEM 表空间、SYSTEM 表空间损坏和 DROP，TRUNCATE 表时，需要先运行 scan extent 命令。

asmcmd 命令

asmcmd 命令进入 ASMCMD 模块，进入 ASMCMD 模块后，使用 exit 命令退出到 ODU 主模块。

```
ODU> asmcmd

Entering asmcmd module.

ASMCMD> exit

Exiting asmcmd module.
```

在 ASMCMD 模块中，同样可以使用 HELP 命令显示可以使用的命令列表。

extract 命令

extract 命令是 ASMCMD 模块中的命令，用于将 ASM 磁盘组中的文件提取到文件系统中。命令格式如下：

```
extract asmfile <src filename> to <dst file_name> [force]
```

force 表示会覆盖已有的文件。
这里 src filename 指的是 ASM 磁盘组中的任意文件。其格式为+<磁盘组名称>.<文件号>或+<磁盘组名称/路径/文件名>。
例如：

```
ASMCMD> extract asmfile +DGDATA.260 to /tmp/260.dbf

starting extract asm file '+DGDATA.260' to '/tmp/260.dbf',file size is 367009792
asm file extract completed.

ASMCMD> extract asmfile +DGDATA/xty/datafile/system.260.745630773 to /tmp/260.dbf

file exists. you can not overwrite it.

ASMCMD> extract asmfile +DGDATA/xty/datafile/system.260.745630773 to /tmp/260.dbf force

starting extract asm file '+DGDATA/xty/datafile/system.260.745630773' to '/tmp/260.dbf',file
size is 367009792
asm file extract completed.
```

可以使用 dbv 工具来检查提取出来的文件：

```
[oracle@xty odu]$ dbv file=/tmp/260.dbf blocksize=8192

DBVERIFY: Release 10.2.0.5.0 - Production on Sat Apr 9 12:59:58 2011

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

DBVERIFY - Verification starting : FILE = /tmp/260.dbf



DBVERIFY - Verification complete

Total Pages Examined         : 44800
Total Pages Processed (Data) : 28974
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 4639
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 1820
Total Pages Processed (Seg)  : 1
Total Pages Failing   (Seg)  : 0
Total Pages Empty            : 9367
Total Pages Marked Corrupt   : 0
Total Pages Influx           : 0
Highest block SCN            : 348723 (0.348723)
```

dump 命令

dump 命令模仿 Oracle 的”alter system dump datafile”命令，解析并显示 oracle 的块格式。目前支持的块包括文件头（只有部分信息）、数据段头、表和索引数据块，其他的块只显示块头。这已经足够大部分情况的使用。ODU 的升级版将解析并显示 UNDO 的段头和数据块。

dump 命令用于帮助分析块格式，在 Oracle 不能启动时（比如在关键的数据字典对象上有物理或逻辑坏块），用于分析块。dump 命令的格式如下：

```
dump datafile <file#> block <block#> [header]
```

header 参数指定只对块头（Block Header) dump。

这个命令格式与 Oracle 的”alter system dump datafile“类似，这里的文件号是绝对文件号。下面是几个示例输出：

dump 数据文件头：

```
ODU> dump datafile 1 block 1
Block Header:
block type=0x0b (file header)
block format=0x02 (oracle 8 or 9)
block rdba=0x00400001 (file#=1, block#=1)
scn=0x0000.00000000, seq=1, tail=0x00000b01
block checksum value=0xcfe5=53221, flag=4
File Header:
Db Id=0xb0f1f85c=2968647772, Db Name=XJ, Root Dba=0x400341
Software vsn=0x9200000, Compatibility Vsn=0x8000000, File Size=0x1f400=128000 Blocks
File Type=0x3 (data file), File Number=1, Block Size=4096
Tablespace #0 - SYSTEM rel_fn:1
get_bootstrap_dba: compat header size:12
bootstrap rdba 0x004002f1 rfile#=1 block#=753
```

dump 数据段头：

```
ODU> dump datafile 10 block 45
Block Header:
block type=0x23 (ASSM segment header block)
block format=0x02 (oracle 8 or 9)
block rdba=0x0280002d (file#=10, block#=45)
scn=0x0000.00209d2c, seq=3, tail=0x9d2c2303
block checksum value=0x7247=29255, flag=4
Data Segment Header:
  Extent Control Header
  -------------------------------------------------------------
  Extent Header:: extents: 1  blocks: 5
                  last map: 0x00000000  #maps: 0  offset: 668
      Highwater:: 0x02800030  (rfile#=10,block#=48)
                  ext#: 0  blk#: 5   ext size:5
      #blocks in seg. hdr's freelists: 0
      #blocks below: 2
      mapblk: 0x00000000   offset: 0
  -----------------------------------------------------
  Low HighWater Mark :
      Highwater::  0x02800030  ext#: 0      blk#: 5      ext size: 5
  #blocks in seg. hdr's freelists: 0
  #blocks below: 2
  mapblk  0x00000000  offset: 0
  Level 1 BMB for High HWM block: 0x0280002b
  Level 1 BMB for Low HWM block: 0x0280002b
  -----------------------------------------------------
  Segment Type: 1 nl2: 1      blksz: 2048   fbsz: 0
```

```
 L2 Array start offset:  0x00000434
 First Level 3 BMB:  0x00000000
 L2 Hint for inserts:  0x0280002c
 Last Level 1 BMB:  0x0280002b
 Last Level 1I BMB:  0x0280002c
 Last Level 1II BMB:  0x00000000
    Map Header:: next  0x00000000  #extents: 1    obj#: 31208  flag: 0x220000000
 Extent Map
 -------------------------------------------------------------

  0x0280002b  length: 5

 Auxillary Map
 -------------------------------------------------------------
  Extent 0      :  L1 dba:  0x0280002b Data dba:  0x0280002e
 -------------------------------------------------------------

  Second Level Bitmap block DBAs
 -------------------------------------------------------------
  DBA 1:   0x0280002c
```

dump 数据块：

```
ODU> dump datafile 10 block 47
Block Header:
block type=0x06 (table/index/cluster segment data block)
block format=0x02 (oracle 8 or 9)
block rdba=0x0280002f (file#=10, block#=47)
scn=0x0000.00209f21, seq=1, tail=0x9f210601
block checksum value=0xc8d=3213, flag=6
Data Block Header Dump:
 Object id on Block? Y
 seg/obj: 0x79e8=31208  csc: 0x00.209d2c  itc: 2  flg: E  typ: 1 (data)
     brn: 0  bdba: 0x280002b ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0x000a.008.000006a4  0x008018c1.00e0.13  --U-    2  fsc 0x0000.00209f21
0x02   0x0000.000.00000000  0x00000000.0000.00  ----    0  fsc 0x0000.00000000
Data Block Dump:
===============
flag=0x0 --------
ntab=1
nrow=2
frre=-1
fsbo=0x16
```

```
ffeo=0x6e7
avsp=0x6fc
tosp=0x6fc
0xe:pti[0] nrow=2  offs=0
0x12:pri[0]  offs=0x76d
0x14:pri[1]  offs=0x6e7
Block Rows Dump:
tab 0, row 0, @0x76d
fb: --H-FL-- lb: 0x1  cc: 2
col   0: [  2]  c1 02
col   1: [ 36]  00 54 00 01 01 0c 00 00 00 01 00 00 00 01 00 00 00 00 d9 95 00 10 09 00 00 00
00 00 00 00 00 00 00 00 00
tab 0, row 1, @0x6e7
fb: --H-FL-- lb: 0x1  cc: 2
col   0: [  2]  c1 03
col   1: [ 84]  00 54 00 01 01 0c 00 00 00 01 00 00 00 01 00 00 00 00 d9 96 00 40 05 00 00 00
03 b6 01 a1 00 00 00 00 00 03 02 80 00 a7 02 80 00 a6 02
 80 00 b1 02 80 00 ae 02 80 00 af 02 80 00 b0 02 80 00 b6 02 80 00 b3 02 80 00 b4 02 80 00 b5
```

hexdump 命令

hexdump 命令用 16 进制显示数据文件中的数据，帮助用于分析块格式。其命令格式如下：

```
hexdump datafile <file#> block <block#> [offset <offset>]
```

这里的偏移(offset)指的是从块头开始的偏移量。
下面是一个示例输出：

```
ODU> hexdump datafile 10 block 47
                 -0--1--2--3--4--5--6--7--8--9--a--b--c--d--e--f-
0000000000017800   06 02 00 00 2f 00 80 02 21 9f 20 00 00 00 01 06
0000000000017810   8d 0c 00 00 01 00 00 00 e8 79 00 00 2c 9d 20 00
0000000000017820   00 00 00 00 02 00 32 00 2b 00 80 02 0a 00 08 00
0000000000017830   a4 06 00 00 c1 18 80 00 e0 00 13 00 02 20 00 00
0000000000017840   21 9f 20 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000017850   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000000000017860   00 00 00 00 00 01 02 00 ff ff 16 00 e7 06 fc 06
0000000000017870   fc 06 00 00 02 00 6d 07 e7 06 00 00 a3 00 80 02
0000000000017880   a3 00 80 02 00 00 00 00 00 00 00 00 00 00 00 00
0000000000017890   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000000178a0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000000178b0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000000000178c0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
00000000000178d0   03 00 00 00 00 08 00 00 01 00 00 00 34 04 00 00
00000000000178e0   00 00 00 00 a4 00 80 02 01 00 00 00 a3 00 80 02
00000000000178f0   a4 00 80 02 00 00 00 00 00 00 00 00 00 00 00 00
0000000000017900   00 00 00 00 00 00 00 00 33 00 00 00 a8 01 80 02
0000000000017910   e4 79 00 00 00 00 00 20 a3 00 80 02 05 00 00 00
0000000000017920   ad 00 80 02 05 00 00 00 b2 00 80 02 05 00 00 00
0000000000017930   b7 00 80 02 05 00 00 00 bc 00 80 02 05 00 00 00
0000000000017940   c1 00 80 02 05 00 00 00 c6 00 80 02 05 00 00 00
0000000000017950   cb 00 80 02 05 00 00 00 d0 00 80 02 05 00 00 00
0000000000017960   d5 00 80 02 05 00 00 00 da 00 80 02 05 00 00 00
```

spool 命令

这条命令与 Oracle 的 sqlplus 中的 spool 命令类似，可以将显示输出记录到一个文件中。命令格式如下：

```
spool <on | off | filename>
```

spool on 是开启将显示输出记录到文件中，文件名使用缺省的 odu_spool.txt
spool off 是关闭此功能。
spool <filename>是将显示输出记录到 filename 指定的文件中。

charset 命令

charset 命令列出 ODU 支持的字符集列表。命令格式如下：

```
charset list
charset name <charset name>
charset id <charset id> --id must greater than 0
```

charset list 列出 ODU 支持的字符集列表：

```
ODU> charset list
CHARSET_NAME                             CHARSET_ID
---------------------------------------- ----------
US7ASCII                                          1
ZHS16GBK                                        852
UTF8                                            871
AL16UTF16                                      2000
ZHS16CGB231280                                  850
ISO2022-KR                                     9997
ISO2022-CN                                     9998
ISO2022-JP                                     9999
```

```
...省略部分显示内容...
```

charset name 命令根据字符集名称得到对应的数字 ID：

```
ODU> charset name US7ASCII
CHARSET_NAME                             CHARSET_ID
---------------------------------------- ----------
US7ASCII                                          1
```

charset id 命令根据数字 ID 得到对应的字符集名称：

```
ODU> charset id 2000
CHARSET_NAME                             CHARSET_ID
---------------------------------------- ----------
AL16UTF16                                       2000
```

start 命令

start 命令类似于 sqlplus 中的 start 命令，能够依次执行文件中的命令序列，命令格式如下：

```
@<filename>
start <filename>
```

start 命令关键字可以使用@符号来代替。
例如：

```
[oracle@xty odu]$ cat unloads.txt
unload table test.t1
unload table test.t2
```

在 ODU 中：

```
ODU> @unloads.txt
ODU> unload table test.t1

Unloading table: T1,object ID: 42252 at 2011-04-09 13:32:35
Unloading segment,storage(Obj#=42252 DataObj#=42255 TS#=6 File#=20 Block#=11 Cluster=0)
1000 rows unloaded
At 2011-04-09 13:32:35

ODU> unload table test.t2

Unloading table: T2,object ID: 42253 at 2011-04-09 13:32:35
Unloading segment,storage(Obj#=42253 DataObj#=42253 TS#=6 File#=20 Block#=27 Cluster=0)
1000 rows unloaded
At 2011-04-09 13:32:35
```

第五章　ODU 配置参数参考

ODU 通过不同的参数配置，以实现其灵活、强大、跨平台的数据恢复功能。一些重要的参数将会影响到数据恢复，而本章将详细介绍 ODU 的各个配置参数。

ODU 默认的配置文件是 config.txt，在启动 ODU 时，会自动打开这个配置文件，在进入 ODU 后，仍然可以通过“load config [config filename]”命令来重新载入配置文件，这个命令中的 config filename（配置文件名）是可选的，如果省略此项，将载入默认的配置文件 config.txt。

而不同平台的 ODU 软件，其安装包中包含的参数配置文件已经为该平台设置了合理的初始值，在使用 ODU 时，只需要设置少量的参数甚至不需要设置参数就可以进行数据恢复。

配置文件是一个纯文本文件，每行为一个配置参数，为“参数名”和“值”，之间以若干空格分隔。参数名不区分大小写,与路径相关的参数值和字符集名称相关的参数值区分大小写，其他参数值不区分大小写。

下面是运行在 x86 平台 Linux 下 ODU 的默认配置文件内容：

```
byte_order little
block_size  8192
block_buffers 1024
db_timezone -7
client_timezone 8
asmfile_extract_path   /asmfile
data_path   data
lob_path    /odu/data/lob
charset_name US7ASCII
ncharset_name AL16UTF16
output_format text
lob_storage infile
clob_byte_order big
trace_level 1
delimiter |
unload_deleted no
file_header_offset 0
is_tru64 no
record_row_addr no
convert_clob_charset yes
use_scanned_lob  yes
trim_scanned_blob yes
lob_switch_dir_rows 20000
```

```
db_block_checksum yes
db_block_checking yes
rdba_file_bits 10
compatible 10
```

下面将详细介绍每一项配置参数的用法以及作用：

BYTE_ORDER

这个参数，指示数据库平台字节序。可选值为“LITTLE”和”BIG”，默认值为“LITTLE”。

这个参数跟 ODU 运行在哪个平台无关，而只与要恢复的数据库的平台有关。比如，数据库是 AIX 平台上的,那么这个值就是“BIG”,而数据文件是 x86 平台上的,那么这个值就是”LITTLE“。ODU 能够跨平台恢复数据，比如可以将 AIX 上的 Oracle 数据文件复制到 Windows 上，由 Windows 版本上的 ODU 进行恢复，反之亦然。只要通过 BYTE_ORDER 这个参数正确地指示数据库的平台，即可实现跨平台恢复。

BLOCK_SIZE

这个参数设置数据文件缺省的块大小。ODU 支持同一数据库中具有不同块大小的数据文件，如果在 ODU 的 control 文件中没有指定块大小（参见后面关于 ODU 控制文件参考章节），则使用配置文件中的 BLOCK_SIZE 指定的大小。这个参数可选的值有 2048、4096、8192、16384、32768，默认值是 8192。

BLOCK_BUFFERS

这个参数指定 ODU 用于缓存数据块的缓存区数量。为提高数据恢复时的性能，ODU 会对部分类型的数据块进行缓存，这部分可以缓存的数据块其块大小与 ODU 的 BLOCK_SIZE 大小一致同时必须为 LOB 列的索引块。而缓存其他类型的数据块对性能影响不大，因此此值不需要设置过大，缺省值 1024 已经满足大部分的需求。

DATA_PATH

DATA_PATH 指定恢复的数据所存储的目录，如果需要恢复的数量量非常大，可以用这个参数值指定一个与 ODU 软件所在目录不同的路径。注意这个参数指定的目录必须是已经存在的，ODU 不会自动创建这个目录。
可以使用相对路径，也可以使用绝对路径。默认值为"data"，表示恢复的数据缺省保存在 ODU 软件所在目录的 data 子目录中。

在数据恢复时，应该首先估算需要的存储空间用于存储恢复的数据。建议将 DATA_PATH 设置为单独的容量足够大的文件系统。

LOB_PATH

LOB_PATH 指定恢复的 LOB 类型的数据所存储的目录。由于 LOB 数据一般比较消耗存储空间，因此可以通过这个参数把 LOB 数据放到与普通数据不同的目录。默认值为空，表示将 LOB 数据与普通数据放置在同一目录下，也就是 DATA_PATH 参数指定的目录。这个参数只有以恢复数据保存为文本文件(text)格式时才起作用，而以 dmp 格式时不会起作用。另外，即使是文本格式，如果 LOB_STORAGE 参数设置为 INFILE，那么 CLOB 类型的数据会与普通的数据保存在同一个文件中，不会单独存储。值得注意的是，这个参数值不为空时，其设置的值应为“绝对路径”，否则在用 SQL*Loader（SQLLDR）导入恢复的数据时，将不能正确地找到存储 LOB 数据的文件。

ASMFILE_EXTRACT_PATH

这个参数用于指定当使用 extract asmfile 命令从 ASM 磁盘组中将文件保存到文件系统上时所使用的路径，默认值为空，表示与普通数据放置在同一目录下，也就是 DATA_PATH 参数指定的目录。

OUTPUT_FORMAT

OUTPUT_FORMAT 参数指定恢复的数据保存的文件格式，目前支持的格式有 TEXT 和 DMP 两种，保存为 TEXT 格式时，ODU 同时为数据生成建表的 SQL 和供 SQL*Loader（SQLLDR）导入数据需要的 CONTROL 文件。如果保存为 DMP 格式，则 ODU 使用与 Oracle 8i 兼容的 DMP 文件格式，可以使用 imp 工具导入恢复的数据。恢复的每一个表都会生成一个 TEXT 文件或 DMP 文件，不支持多个表存储在同一个 DMP 文件中，也不支持一个表分成几个部分保存在多个 DMP 文件中。此参数默认值为“TEXT”。

LOB_STORAGE

LOB_STORAGE 参数指定 LOB 数据的存储位置以及是否要恢复 LOB 数据。这个参数可选的值有：FILE、INFILE、NONE，默认值为 INFILE。值为 NONE 表示在恢复数据时，不恢复 LOB 数据。INFILE 表示在以文本(TEXT)文件存储恢复数据时，CLOB 列的数据与普通数据存储在同一文件中，此时 BLOB 数据不受此参数影响。FILE 表示在以文本文件存储恢复数据时，CLOB 列的数据存储在单独的文件中，在普通的数据文件中只存储了 LOB 数据所存储的文件名。BLOB 列始终存储在单独的文件中，除非 LOB_STORAGE 设置为 NONE，在这种情况下将不恢复 LOB 数据。将 LOB 列存储为单独的文件时，每一行的每个 LOB 列数据，存储为一个文件。

CLOB_BYTE_ORDER

CLOB_BYTE_ORDER 参数指定 CLOB 列数据的字节序。由于 CLOB 列数据在数据块中以 UNICODE 编码存储,需要正确设置此参数才能将 CLOB 列数据转换为客户端数据。可选的值有“LITTLE”和“BIG”，默认值为“BIG”。从 Oracle 10g 开始，CLOB 数据全部以 big endian 字节序存储，在 10g 以及以上版本，ODU 会自动将 CLOB 数据字节序处理为 big endian。但是可能存在 9i 数据库升级到 10g 的情况，针对这样的情况，仍然需要将 CLOB_BYTE_ORDER 设为与数据库相同的平台字节序。9i 及以下版本的数据文件，根据数据库平台，设置为不同的值，比如 x86 平台上的应设置为”LITTLE”。注意，此参数与 BYTE_ORDER 一样，与 ODU 本身运行的平台无关，而只与数据库的平台有关系。

CONVERT_CLOB_CHARSET

在数据库字符集为多字节字符集时，在数据库中 CLOB 会存储为 UNICODE 字符，但是对于单字节字符集，CLOB 会存储为与数据库相同的单字节字符集。对于这样的数据库，应该将 CONVERT_CLOB_CHARSET 设为 FALSE，表示不转换 CLOB 的字符集，默认值为 TRUE。

LOB_SWITCH_DIR_ROWS

在将数据恢复为 TEXT 文件保存同时 LOB 数据设置为存储在单独的文件中时（参见 LOB_STORAGE 参数说明），如果表的数据量比较大，将会生成非常多的文件，这可能引起文件操作性能缓慢。为避免这一情况，ODU 会根据 LOB_SWITCH_DIR_ROWS 设置，在 LOB 文件数量到达这一设置的数量之后，会生成新的子目录来存储 LOB 数据。这一参数的默认值为 50000。

CHARSET_NAME 和 NCHARSET_NAME

这两个参数主要用于标明要恢复的数据，其字符集与国家字符集，也就是数据库的 NLS_CHARACTERSET 和 NLS_NCHAR_CHARACTERSET 两个参数的设置。ODU 在恢复数据时，将 NCHAR、NVARHCAR2 类型的列数据从 NCHARSET_NAME 指定的字符集编码转换为 CHARSET_NAME 所指定的字符集编码，同时将 CLOB 转换为 CHARSET_NAME 所指定的字符集编码。不正确的设置将导致以上几种类型的列数据在恢复后是乱码。这两个参数的取值，其值名称与 Oracle 数据库中的一致，CHARSET_NAME 默认值为“US7ASCII”，NCHARSET_NAME 默认值为“AL16UTF16”。在 ODU 中可以使用命令“CHARSET LIST”命令查看 ODU 当前版本所支持的字符集编码。

DELIMETER

DELIMITER 指定以 TEXT 格式恢复数据时，列之间的分隔符。默认值为“|”（竖线）。由于字符型数据可能包含这个字符，可以将这个参数设为一个其他不会在数据出现的字符（比如某个特殊的符号）。分隔符可以指定为字符串，而不仅仅是一个单字符。比如，可以指定为“||”等。

UNLOAD_DELETED

UNLOAD_DELETED 参数指定是否恢复已经被 DELETE 的行。可选的值为“YES”和“NO”，默认值为“NO”。这个参数用于在意外 DELETE 重要数据之后，并且不能够利用闪回查询，备份和 logminer 进行恢复，同时被 DELETE 的数据原来所在的空间没有被重用（覆盖）的情况下，恢复被 DELETE 掉的数据。由于不能判断 DELETE 的时间，因此利用这个功能恢复数据时，会恢复所有曾经 DELETE 的数据，另外，在某些情况下可能会导致 ODU 异常。因此，这个参数在大多数情况下应该设置为“NO”。

COMPATIBLE

用于指定数据库的版本。默认值为 10，即 10g。这个参数的有效值为 Oracle 的主版本号，从 7 至 12。

FILE_HEADER_OFFSET

这个参数设置裸设备数据文件缺省的文件头偏移量。ODU 支持同一数据库具有不同文件头偏移量的裸设备数据文件，如果在 ODU 的 control 文件中指定文件头偏移量（参见后面关于 ODU 控制文件参考章节），则使用配置文件中的 FILE_HEADER_OFFSET 指定的偏移量。默认值是 0。

在使用裸设备作为数据库数据文件时，裸设备的头部可能被操作系统所使用，而 Oracle 使用的部分在这个头部之后。头部的大小，在不同的操作系统上有所不同。在 AIX 系统上，普通的逻辑卷（lv 即裸设备）的头部大小是 4KB，在这种情况下 FILE_HEADER_OFFSET 为 4096，而新的 Scalable VG 的逻辑卷则不再有头部，此时 FILE_HEADER_OFFSET 应该设置为 0。在 HP TRU64 系统上，FILE_HEADER_OFFSET 应该设置为 65536。

DB_BLOCK_CHECKSUM

这个参数用于设置 ODU 在读取数据块时，是否会对数据块的 CHECKSUM 值进行校验。默认值

为 TRUE。

DB_BLOCK_CHECKING

这个参数用于设置 ODU 在读取数据块时，是否会对数据块进行逻辑校验。默认值为 TRUE。

RDBA_FILE_BITS

这个参数用于设置 ODU 在解析数据块中的 RDBA（相对块地址）时，文件号应该使用的位数。通常情况下，RDBA 由 10 位文件号和 22 位块号组成，因此这个参数的默认值 10 能够满足大多数情况下的需求。在一些特定的 Oracle 版本和平台上，这个值可能有所不同，比如在 Oracle 7 版本中，这个参数值设为 8。

USE_SCANNED_LOB

这个参数用于设置 ODU 在恢复 LOB 数据时，是否会在不能从数据字典获取到 LOB 信息或 LOB 索引数据损坏时，从扫描的数据中获取 LOB 信息并恢复 LOB 数据。默认值为 YES。

TRIM_SCANNED_BLOB

这个参数用于设置 ODU 在使用扫描的 LOB 信息恢复 BLOB 数据时，是否会去掉 BLOB 数据最后部的全 0 数据。因为这个时候 ODU 不能获得 BLOB 列准确的长度。

第六章　ODU控制文件和ASM磁盘信息文件参考

ODU 控制文件

ODU 控制文件是纯文本（TEXT）类型的文件，用于指定 ODU 在进行数据恢复时需要使用的数据文件，缺省为 control.txt 文件。文件中每一行代表一个数据文件，如果行以#号开头，表示这一行是注释。每一行由 8 列组成，列之间用 1 个或多个空格分隔。

下面分别描述控制文件中 8 列的作用。

✧ **表空间号（TS#)**
用于指定数据文件的表空间号。

✧ **相对文件号（RELATIVE FILE#)**
用于指定数据文件的相对文件号。

✧ **绝对文件号（FILE ID)**
用于指定数据文件的绝对文件号。

✧ **文件名称（FILE NAME)**
用于指定数据文件的文件名。
对于使用 ASM 的数据文件名，可以使用标准的 ASM 数据文件名称，比如+DGDATA/xty/datafile/system.260.745630773，也可以使用“+<DG 名称>.<DG 文件号>”这样的简化形式，比如上面的文件名可以简化为+DGDATA.260


提示：
对于 Linux 下的 RAW 文件，ODU 不直接支持，但是 ODU 支持 RAW 文件对应的实际设备文件，比如数据库中使用的是/dev/raw/raw1，其对应的实际设备文件为/dev/sdc1，则需要在 ODU 控制文件的文件名称一列（栏）填上/dev/sdc1。如果运行 ODU 的用户没有访问权限，则对用户赋予可读权限，或以有权限的用户比如 root 用户来运行 ODU。

✧ **块大小（BLOCK SIZE)**
当数据文件的块大小(BLOCK_SIZE)与 ODU 配置参数中指定的 BLOCK_SIZE 不一致时，可以在此列填入实际的块大小。主要用于数据库中存在多种块大小的数据文件的情况。

✧ **是否 BIG FILE（IS BIG FILE？)**
用于指定数据文件是否是 Oracle 10g 中的大文件表空间（Big File Tablespace）的数据文

件。

✧ 文件头部偏移（FILE HEAER OFFSET)

当数据文件的头部偏移(FILE HEADER OFFSET)与 ODU 配置参数中指定的 FILE_HEADER_OFFSET 不一致时，可以在此列填入实际的偏移大小。主要用于数据库中存在多种头部偏移的数据文件的情况。比如在 AIX 系统中混用了 NORMAL VG 和 SCALABLE VG 的情况。

✧ 以块为单位的文件大小（BLOCKS)

指定数据文件的大小，以块为单位。

在向 ODU 控制文件填写数据文件信息时，不必填入所有的列（栏），如果数据文件头部完好，ODU 能够自动识别出数据文件的信息。在这种情况下，只需要填入数据文件名即可。

在 ODU 控制文件填入一列信息时，这一列之前的其他列也必须填入。比如填入 FILE HEADER OFFSET 列时，必须也要填入 BLOCK SIZE 等其他列，但是 BLOCKS 列则不必填入。

通常情况下只需要填入数据文件名即可，但是正如上面所述，数据文件名前 3 列也必须要填入，ODU 能够自动识别数据文件信息的情况下，前 3 列简单地填入 0 即可。

下面是 ODU 控制文件的 2 个示例，第 1 个是使用 ASM 的数据文件，第 2 个是普通的文件系统数据文件。

示例 1：

```
        0          0          0  +DGDATA/xty/datafile/system.260.745630773
        0          0          0  +DGDATA/xty/datafile/undotbs1.261.745630805
        0          0          0  +DGDATA/xty/datafile/sysaux.262.745630817
        0          0          0  +DGDATA/xty/datafile/users.264.745630833
```

示例 2：

```
        0          0          0  D:\ORACLE\ORADATA\XJ\SYSTEM01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\UNDOTBS01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\INDX01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\TOOLS01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\USERS01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\TEST_8K.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\SYSAUX01.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\TT_TEST1.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\TT_TEST2.DBF
        0          0          0  D:\ORACLE\ORADATA\XJ\OEM.DBF
```

对于能够启动到 MOUNT 状态的数据库，可以使用下面的 SQL 来获取填入 ODU 控制文件的数据：

```
SELECT 0,0,0,NAME FROM V$DATAFILE ORDER BY FILE#
```

ASM 磁盘信息文件

ODU 使用的 ASM 磁盘信息文件是纯文本（TEXT）类型的文件，用于指定 ODU 在进行数据恢复时需要使用的 ASM 磁盘信息（如果数据库使用了 ASM），缺省为 asmdisk.txt 文件。文件中每一行代表一个 ASM 磁盘，如果行以#号开头，表示这一行是注释。每一行由 7 列组成，列之间用 1 个或多个空格分隔。

下面分别描述 ASM 磁盘信息文件中 7 列的作用。

✧ **磁盘编号（DISK NO)**

指定 ASM 磁盘组中 ASM 磁盘的编号，通常最早创建到磁盘组中的 ASM 磁盘，编号最小。磁盘编号在同一个磁盘组内不能重复。

✧ **磁盘路径（DISK PATH)**

指定 ASM 磁盘的设备文件名称。

提示：

对于 Linux 下的 RAW 磁盘（或 RAW 分区），ODU 不直接支持，但是 ODU 支持 RAW 磁盘所对应的实际设备文件，比如数据库中使用的是/dev/raw/raw1，其对应的实际设备文件为/dev/sdc1，则需要在 ODU 的 ASM 磁盘信息文件的磁盘路径列（栏）填上/dev/sdc1。如果运行 ODU 的用户没有访问权限，则对用户赋予可读权限，或以有权限的用户比如 root 用户来运行 ODU。

如下是一个 Linux 6 下的 11gR2 的 ASM 磁盘信息文件具体配置内容,这里/dev/raw/raw[i]对应的实际设备名是/dev/sda[i]：

如果直接用 RAW 磁盘的名称，则 ODU 将不能识别出正确的磁盘信息：

```
[oracle@bspdev odu]$ cat asmdisk.txt
# disk_no   disk_path          group_name  meta_block_size   ausize  disk_size
header_offset
0 /dev/raw/raw3 DATA 4096 1048576
1 /dev/raw/raw5 DATA 4096 1048576
2 /dev/raw/raw6 DATA 4096 1048576
0 /dev/raw/raw7 RECO 4096 1048576
1 /dev/raw/raw8 RECO 4096 1048576
```

```
[oracle@bspdev odu]$ ./odu

Oracle Data Unloader:Release 4.1.3
```

```
Copyright (c) 2008,2009,2010,2011 XiongJun. All rights reserved.

Web: http://www.oracleodu.com
Email: magic007cn@gmail.com

loading default config.......

byte_order little
block_size  8192
block_buffers 1024
db_timezone -7
client_timezone 8
asmfile_extract_path   /odu/asmfile
data_path   data
lob_path    /odu/data/lob
charset_name AL32UTF8
ncharset_name AL16UTF16
output_format text
lob_storage infile
clob_byte_order big
trace_level 1
delimiter |
unload_deleted no
file_header_offset 0
is_tru64 no
record_row_addr no
convert_clob_charset yes
use_scanned_lob  yes
trim_scanned_blob yes
lob_switch_dir_rows 20000
db_block_checksum yes
db_block_checking yes
rdba_file_bits 10
compatible 10
load config file 'config.txt' successful
loading default asm disk file ......

read data error from asm disk '/dev/raw/raw3'.error message:Invalid argument
read data error from asm disk '/dev/raw/raw5'.error message:Invalid argument
read data error from asm disk '/dev/raw/raw6'.error message:Invalid argument
read data error from asm disk '/dev/raw/raw7'.error message:Invalid argument
read data error from asm disk '/dev/raw/raw8'.error message:Invalid argument

grp# dsk# bsize ausize disksize diskname        groupname       path
```

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


load asm disk file 'asmdisk.txt' successful
loading default control file ......

can not found diskgroup for file +DATA/ora11g/datafile/system.256.747310449.
can not found diskgroup for file +DATA/ora11g/datafile/sysaux.257.747310449.
can not found diskgroup for file +DATA/ora11g/datafile/undotbs1.258.747310451.
can not found diskgroup for file +DATA/ora11g/datafile/users.259.747310451.

 ts#   fn  rfn bsize   blocks bf offset filename
----        ----        ----        -----        --------        --        ------
-----------------------------------------
load control file 'oductl.dat' successful
loading dictionary data......done

loading scanned data......done
```

从提示信息里我们可以看到 ODU 并不能直接识别 RAW 类型的磁盘。

此时，我们应该将 asmdisk.txt 中的磁盘路径列改成 RAW 磁盘实际对应的设备文件名，并对用户赋予可读权限，或以有权限的用户比如 root 用户来运行 ODU，如下所示：

```
[oracle@bspdev odu]$ su
Password:
[root@bspdev odu]# cat asmdisk.txt
# disk_no   disk_path           group_name  meta_block_size   ausize  disk_size
header_offset
0 /dev/sda3 DATA 4096 1048576
1 /dev/sda5 DATA 4096 1048576
2 /dev/sda6 DATA 4096 1048576
0 /dev/sda7 RECO 4096 1048576
1 /dev/sda8 RECO 4096 1048576
[root@bspdev odu]# ./odu

Oracle Data Unloader:Release 4.1.3

Copyright (c) 2008,2009,2010,2011 XiongJun. All rights reserved.

Web: http://www.oracleodu.com
Email: magic007cn@gmail.com

loading default config.......

byte_order little
```

```
block_size  8192
block_buffers 1024
db_timezone -7
client_timezone 8
asmfile_extract_path   /odu/asmfile
data_path   data
lob_path    /odu/data/lob
charset_name AL32UTF8
ncharset_name AL16UTF16
output_format text
lob_storage infile
clob_byte_order big
trace_level 1
delimiter |
unload_deleted no
file_header_offset 0
is_tru64 no
record_row_addr no
convert_clob_charset yes
use_scanned_lob  yes
trim_scanned_blob yes
lob_switch_dir_rows 20000
db_block_checksum yes
db_block_checking yes
rdba_file_bits 10
compatible 10
load config file 'config.txt' successful
loading default asm disk file ......


grp# dsk# bsize ausize disksize diskname        groupname       path
---- ---- ----- ------ -------- --------------- ---------------
----------------------------------------
   1    0  4096  1024K     9000 DATA_0000       DATA            /dev/sda3
   1    1  4096  1024K     9000 DATA_0001       DATA            /dev/sda5
   1    2  4096  1024K     9000 DATA_0002       DATA            /dev/sda6
   2    0  4096  1024K     9000 RECO_0000       RECO            /dev/sda7
   2    1  4096  1024K     7288 RECO_0001       RECO            /dev/sda8

load asm disk file 'asmdisk.txt' successful
loading default control file ......


 ts#   fn  rfn bsize   blocks bf offset filename
```

```
----        ----        ----        -----       --------        --        ------
-----------------------------------------
   0            1            1       8192            88320    N                    0
+DATA/ora11g/datafile/system.256.747310449
   1            2            2       8192            89600    N                    0
+DATA/ora11g/datafile/sysaux.257.747310449
   2            3            3       8192            12160    N                    0
+DATA/ora11g/datafile/undotbs1.258.747310451
   4            4            4       8192              640    N                    0
+DATA/ora11g/datafile/users.259.747310451
load control file 'oductl.dat' successful
loading dictionary data......done

loading scanned data......done
```

从结果里我们可以看到，现在 ODU 已经能够正确的识别出所有的 ASM 磁盘。

✧ **磁盘组名（DISK GROUP NAME)**

指定 ASM 磁盘组的名称

✧ **元数据块大小（META BLOCK SIZE)**

指定 ASM 磁盘组中元数据块的大小，通常为 4KB 大小。

✧ **AU 大小（AU SIZE)**

指定 ASM 磁盘组的分配单元大小（Allocation Unit Size），通常为 1MB。

✧ **磁盘大小（DISK SIZE)**

以 AU 为单位的磁盘大小，比如 AU 大小为 1MB 时，而磁盘大小为 5000，表示磁盘实际的大小为 5000MB。

✧ **磁盘头部偏移（HEADER OFFSET)**

磁盘设备头部可能被操作系统使用，使用此偏移指定 ASM 磁盘实际开始使用的起始偏移位置。

在向 ASM 磁盘信息文件填写 ASM 磁盘信息时，不必填入所有的列（栏），如果 ASM 磁盘头部完好，ODU 能够自动识别出 ASM 磁盘及磁盘组的信息。在这种情况下，只需要填入 ASM 磁盘路径即可。

在向 ASM 磁盘信息文件填入一列信息时，这一列之前的其他列也必须填入。比如填入 DISK SIZE 列时，必须也要填入 AU SIZE 等其他列，但是 HEADER OFFSET 列则不必填入。
通常情况下只需要填入磁盘路径即可，但是正如上面所述，磁盘路径前 1 列也必须要填入，ODU 能够自动识别 ASM 磁盘信息的情况下，第 1 列简单地填入 0 即可。

asmdisk.txt 示例：

```
# disk#  path       group_name meta_block_size  ausize disk_size header_offset
```

```
0  /oradata/asm/disk1.dbf
0  /oradata/asm/disk2.dbf
0  /oradata/asm/disk3.dbf
```

注：在这个示例中，是在测试环境中用文件来模拟的 ASM 磁盘。

第七章　Trouble Shooting

本章将描述在使用 ODU 恢复时最常遇到的几种问题。

不能自动识别数据文件

在进入 ODU 后或使用 open 打开 ODU 控制文件时，不能自动识别数据文件信息。出现这种问题通常有 4 种原因：

✧　BLOCK_SIZE 不正确。
不正确的 BLOCK_SIZE 配置参数或 ODU 控制文件中指定了错误的 BLOCK_SIZE，使得 ODU 不能够正确地识别数据文件信息。这种情况下应该将 BLOCK_SIZE 改为与数据文件一致的 BLOCK_SIZE 再试试。如果不能确定 BLOCK_SIZE,可以分别尝试 2048,4096,8192,16384,32768 这五个不同的值。

✧　字节序不正确。
即 ODU 配置参数 BYTE_ORDER 不正确，对于 HP、AIX、Solaris SPARC 平台，BYTE_ORDER 为 BIG,而其他平台通常为 LITTLE。对于不能确定的 BYTE_ORDER，可以将 BYTE_ORDER 从当前值改为另一个值再试试。即将 LITTLE 改为 BIG 或将 BIG 改为 LITTLE 试试。

✧　数据文件头部偏移不正确。
不正确的数据文件头部偏移或 ODU 控制文件中指定了错误的头部偏移，使得 ODU 不能够正确地识别出数据文件信息。不种情况只会出现在使用了裸设备的数据文件中，而文件系统数据文件不会出现这种情况。在这种情况下，可以分别尝试 0,4096,65536 这三个不同的值。

✧　数据文件头部损坏。
数据文件本身存在损坏，这种情况下只能人工确定数据文件信息并填入到 ODU 控制文件中。

CLOB 数据是乱码

如果恢复出来的数据是乱码，这通常是由于与 CLOB 相关的参数没能正确配置有关，这些参数是 CLOB_BYTE_ORDER 和 CONVERT_CLOB_CHARSET。
CLOB_BYTE_ORDER 通常应该设置为与数据库所在平台一致的字节序。即应该设置为 BYTE_ORDER 相同的值。而 CONVERT_CLOB_CHARSET 参数，在数据库使用单字节字符集时，应该为 FALSE，在使用多字节字符集时，应该为 TRUE。

其他问题

对于本章没有列出的问题，请访问 ODU 技术支持网站 http://www.oracleodu.com/cn/support，获取支持。