GDUL 用户使用手册 V2025.1.u2

更新日期: 2025-01-25

目录

产品新特性 7
2025.1.2 版本 7
2024.1.9 版本 7
2024.1.8 版本 7
2024.1.5 版本 7
2024.1.3 版本 7
2024.1.2 版本 7
2023.1.13 版本 7
2023.1.12 版本 8
2023.1.10 版本 8
2023.1.7 版本 8
2023.1.6 版本 8
2022.1.1 版本 8
5.8.0.1 版本 8
5.7.0.35 版本 8
5.7.0.33 版本 9
5.7.0.32 版本 9
5.7.0.31 版本 9
5.7.0.28 版本 9
5.7.0.24 版本 9
5.7.0.21 版本 9
5.7.0.19 版本 9
5.7.0.12 版本 9
5.7.0.7 版本 10
5.7.0.6 版本 10
5.7.0.1 版本 10
5.6.0.13 版本 10
5.5.0.10 版本 10
5.4.0.9 版本 10
5.3.0.6 版本 10
5.2 版本 10

5.1 版本 11
5.0.4.3 版本 11
5.0 版本 11
产品支持列表 12
支持的表类型 12
支持的列类型 12
支持的导出格式 13
支持的文件系统 14
文件说明 16
GDUL 介质列表 16
GDUL 目录结构 16
gdul.ini 文件 16
最佳实践 18
1 生成配置文件 18
1) 自动生成配置文件(Linux/Unix) 18
2) 自动生成配置文件(Windows) 19
3) 手工设置配置文件 19
4) ASM 磁盘注意事项 22
5) CDB 数据库注意事项 23
2 初始化数据字典，导出数据 24
1) 执行 GDUL 命令，初始化数据字典 24
2) 显示数据库信息 24
3) 导出表数据 26
4) 导出 delete 的表数据 27
5) 导出 truncated 的表数据 28
6) 导出 drop 的表数据 30
7) 导出对象定义 31
8) 表有坏块场景恢复 32
3 drop user 恢复(从字典表数据块中恢复被删除行) 36
4 drop user 及 drop tablespace 恢复(测试库中有表结构) 37
1) 测试库建立用户及表结构 37
2) 扫描删除的表空间 38
3) 获取被删除的表对应的 data_object_id 38
4) unload 采样出的数据段 38
5 查找表对应的 data_object_id 39

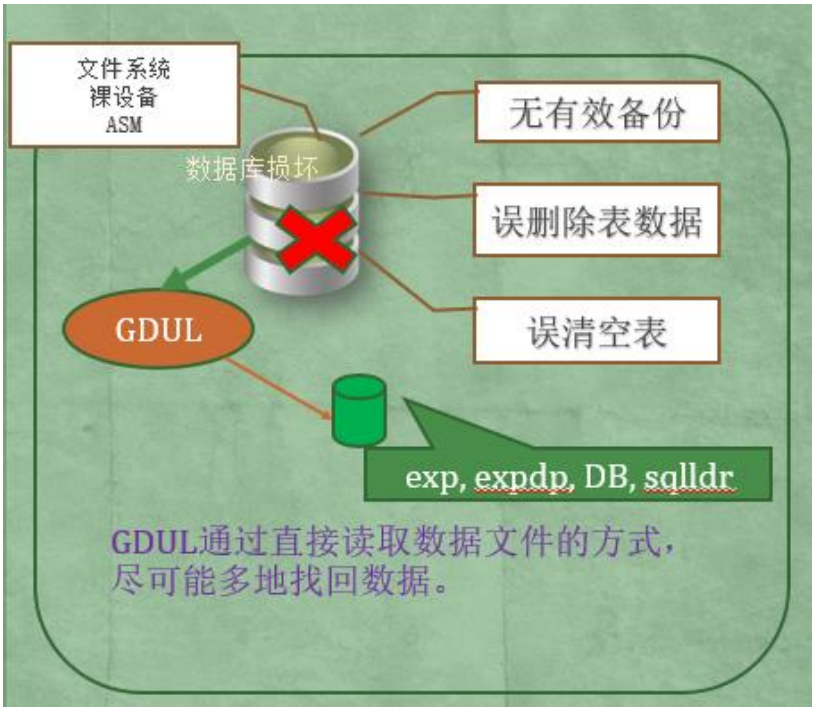
6 特殊表及列类型 41
1) LOB 列类型 41
2) XMLTYPE 列类型 42
3) UDT/VARRAY 列类型 44
4) 关于 12C guard-column 列 44
5) 导出带中文表名 46
直接导出到新库 47
1 生成配置文件 47
2 启动 gdul.sh，初始化字典 47
3 新库创建表结构 48
4 导出用户或表到新库 49
5 导入时常见错误 51
6 常用查询脚本 52
7 限制 53
SYSTEM 文件损坏处理 54
1. 文件头损坏时，手工填写 ts#, rfile#. 54
1. 设置 gdul.ini 54
2. 设置 datafile.ini 54
3. 扫描数据文件，获取 ts#, rfile# 55
2. SYSTEM 文件头损坏场景 56
1) 扫描 system 表空间，查找 rootdba 56
2) bootstrap 时，指定 rootdba 的文件#和块# 56
3. SYSTEM 文件内容部分损坏场景 57
1) 扫描 SYSTEM 表空间 57
2) 扫描字典表 57
3) 扫描单个字典表 57
4. 无 SYSTEM 表空间恢复(测试库中有表结构) 62
1) 使用测试库的 system 导出数据字典。 62
2) 尝试从 SYS.wrh$_seg_stat_obj 表获取<表, data_obj#>对应关系。 62
3) 扫描业务表所在表空间 62
4) 采样已扫描的业务表空间 62
5) unload 采样出的数据段 63
5. 无 SYSTEM 表空间恢复(无任何备份) 63
1) 设置字符集等参数(gdul.ini) 63
2) 扫描表所在表空间 63

3) 采样已扫描的表空间 63
4) unload 采样出的数据段 64
6. 勒索程序处理 65
1) PL/SQL 勒索病毒删除 TAB$表数据恢复 65
2) PL/SQL 勒索病毒删除 BOOTSTRAP$表数据恢复 72
3) 文件勒索病毒恢复 73
ASM 磁盘损坏处理 75
1. 磁盘组损坏，元数据正常 75
1) 配置 asmdisk.ini 文件 75
2) 扫描 ASM 磁盘组 77
3) 更正 asmdisk.ini 文件 77
4) 更改 datafile.ini 文件，指向 ASM 里的数据文件 78
5) 运行 gdul，导出数据或拷贝出数据文件 79
2. 磁盘组损坏，元数据部分损坏 79
3. 磁盘组损坏，元数据彻底损坏 79
US7ASCII 字符集中文处理 80
EXP/IMP 导出导入(适用于非 CLOB 表) 80
GDUL/SQLLDR 导出导入（适用于 CLOB 表） 81
实用程序 82
1 内置 asmcmd 82
1) ls、cd 82
2) copy 82
3) cpdir 83
4) scan diskgroup 83
5) extract asmfile# 84
6) scan asmdisk 84
2 EXPDP 损坏导出工具 84
1) 设置导出格式为 DB 84
2) 扫描 dump 文件 85
3) 创建表结构 85
4) 导出 DP 中的表数据 86
3 EXP 损坏导出工具 86
1) 设置导出格式为 DB 86
2) 扫描 dump 文件 87
3) 创建表结构 87

4) 导出 exp 文件中的表数据 87
4 磁盘扫描工具(beta) 88
5 其它命令 89
1) file 89
2) rowid 90
3) rdba 90
4) copyscn 90
5) scan filelist 91
6) scan undo 92
7) oradump 92
6 ORACLE 导入命令 92
1) impdp 92
2) imp 94
3) sqlldr 94

产品介绍

当 ORACLE 数据库由于某种原因无法打开时，或者误 truncate 表数据时，可以利用 GDUL 直接读取数据文件，然后把表数据读取出来。



功能特点：

1. ORACLE 数据库损坏时导出各类对象定义及表数据。
2. 误 truncated 表、删除行、drop 表恢复。
3. 无 SYSTEM 表空间恢复。
4. 从 ASM 拷贝文件至本地文件系统。
5. 多种导出方式，可以导出成 datapump 文件，旧 exp 文件，新数据库，SQLLDR 数据文件。

产品新特性

2025.1.2 版本

1. 修复 Windows 2003 下无法运行的问题。
2. 修复 Solaris 10 早期 update 版本无法运行的问题。
3. 修复 GDUL 命令行无法输入中文字符的问题，Windows 下已禁用 TAB 键命令完成功能。
4. 行链接涉及到损坏的 LONG 列值时，更改策略，由跳过该行，改为 LONG 列置为 NULL，保留该行。
5. 新增 exp_incomplete_row 参数，控制是否导出不完整的链接行(chained-row).

2024.1.9 版本

1. unload object 命令增强，支持导出表 Oracle interval 类型分区选项。
2. 新增 reload config 命令，重新加载各个配置文件，不重新加载 dict 文件。
3. ASM 配置文件增强，支持 1M(1048576)大小的磁盘头 raw_offset。适用于某些 ASM 环境，/dev/mapper/mpathX 磁盘具有磁盘头偏移量（如 1MB）的场景。
4. SQLLDR 导出格式增强，支持导出 timestamp with time zone, interval 列类型。

2024.1.8 版本

1. gdul.ini 中加入 export_dir 参数，指定导出目录。
2. GDUL> scan filelist 命令增强，支持 ASM 环境，应对 ASM 数据文件头损坏情况。
3. 新增 GDUL> ddl XXX 命令，显示表的基础 DDL 语句。
4. GDUL> unload user XXX 命令增强，执行结束，打印成功，失败，跳过，忽略的表数。
5. 支持 ANYDATA 列类型，导出至 EXPDP 文件格式。

2024.1.5 版本

1. 支持 21C JSON 列类型，导出至 EXPDP 文件格式。
2. 支持 23AI VECTOR，BOOLEAN 列类型，导出至 EXPDP/DB 格式。

2024.1.3 版本

1. oradump 命令增强，打印段头块内容。
2. 导出为 expdp 格式时，磁盘空间不足时打印错误信息。

2024.1.2 版本

1. 命令行增强, 提供类似 bash 的 TAB 键命令补全、命令行编辑，及上下箭头显示历史命令。

2023.1.13 版本

1. bootstrap scan 命令更新，执行完自动加载新生成的字典文件。

2. 新增加 reload dict 命令，在 scan col, scan obj 等扫描命令执行完毕后，重新加载新生成的字典文件。

2023.1.12 版本

1. AIX 环境, ASM 存储获取磁盘大小失败 bug 修复。
2. 导出格式为 DB 时，大量空表的情况，导致创建连接失败 bug 修复。

2023.1.10 版本

1. 增加 forcecopy 命令，处理磁盘物理损坏，无法拷贝时的情况。
2. Bootstrap 时，如果 USER$损坏，使用 USER_ID 作为临时的用户名称。
3. sample segment crash 修复。
4. GDUL for Windows，添加 osetup.bat 脚本，数据库 mount 时，收集 datafile.ini 和 asmdisk.ini。

2023.1.7 版本

1. Bootstrap 命令加入 scan 选项，用于处理 SYSTEM 数据文件头部若干块损坏的情况，如勒索加密病毒。

2023.1.6 版本

1. unload object DDL 导出命令增强，支持导出 job, scheduler job DDL。
2. unload object, sequence DDL 增量为负时，bug 修复。
3. unload object, index DDL 增强，排除已创建的主键约束索引。
4. 2022.1 版本之后引入的 UDT 类型表导入 bug，错误修复。
5. list table 命令增强，显示表创建时间。

2022.1.1 版本

1. 版本改为年份发布版本。
2. `unload segment all` 问题修复。
3. 跳过 ET$表。

5.8.0.1 版本

1. 增加 `merge dict` 命令，当表对象名正常，但丢失列定义时，从测试库的同名用户中并入列定义。
2. 当字典丢失 LOB 信息时，在加载字典过程中，通过表信息，补上 LOB 信息。
3. 增加 `useLobIndex` 参数，用来控制导出时，是否读取 LOB 索引内容。当数据异常，导致读取 LOB 索引卡住时，设置为 `false`。

5.7.0.35 版本

1. 增加 `scan filelist` 命令，扫描数据文件对应的`<ts#, rfile#>`。
2. 增加 `scan undo` 命令，扫描 UNDO$中的活动回滚段。

5.7.0.33 版本

1. 增加 unload database 命令，导出除 SYS/Example 用户的所有用户数据。

5.7.0.32 版本

1. Binary XMLTYPE 表，impdp 导入时，ORA-00600: [qmcxeReplaceNspid]，ORA-00600: [qmcxeReplaceQnameids:1]错误修复。

5.7.0.31 版本

1. 由于 12C 的 DDL 优化功能，在 unload object 生成表完整 DDL 时，临时添加 RAW 隐含列定义，导完数据后，请手工 drop 该 RAW 类型列。
2. Unload table 获取 Extent 列表时，如果缺失数据文件，可能的死循环问题修复。
3. Unload table 单表并行导出时，scan 选项未生效问题修复。

5.7.0.28 版本

1. Binary XMLTYPE 列值，中文标签 bug 修复。
2. Expdp 导出格式，支持 12C 长表名(>30 字符)。

5.7.0.24 版本

1. list/unload user 过滤掉域索引相关的 DR$,DRC$内部表。
2. AIX 平台 ASM 环境，如果使用 RLV 作为 ASM 磁盘，自动检测磁盘头 offset。

5.7.0.21 版本

1. Linux 环境, scan database/tablespace 时，coredump bug 修复。
2. unload table LOB 分区表 scan，功能增强。
3. untrunc table LOB 分区表，功能增强。

5.7.0.19 版本

1. 支持导出物化视图定义。
2. 压缩 xmltype/lob， bug 修复。
3. clob 存储格式 xmltype，impdp 导入到 12c 时， bug 修复。
4. 扫描 lob 分区表， bug 修复。

5.7.0.12 版本

1. 函数索引 DDL 中包含"||"符号时，问题修复。
2. GDUL 导出损坏的 EXPDP 文件时，已支持 Basic 压缩格式。

5.7.0.7 版本

1. LOB 表> 255 列时，EXPDP 导出格式 bug 修复。

5.7.0.6 版本

1. SecureFile LOB 支持去重，以及去重+压缩存储格式。

5.7.0.1 版本

1. 自动检测 AIX 环境，数据文件使用裸设备(/dev/rlv*)时的偏移量。
2. 加入 offset 实用程序，手工验证 AIX 下裸设备(/dev/rlv*)的偏移量。

5.6.0.13 版本

1. DPCMD 实用程序，支持高级压缩 LOW，MEDIUM，HIGH 三种压缩。不支持旧的 BASIC 压缩。
2. 支持 12C 扩展数据类型，varchar2，raw 最大长度 32767 字节。

5.5.0.10 版本

1. undelete dicttab 命令，完善块校验算法，dbv 命令通过。
2. 导出为 EXP/EXPDP/SQLLDR 格式时，生成导入脚本文件 run_<用户名>.sh。
3. 导出为 DB 格式时，检查数据库表是否存在，不存在的话，给出错误提示。
4. 导出 xmltype 表时，如果导出格式非 EXPDP 格式，给出提示，并跳过表。
5. 支持导出损坏的 EXP 文件。
6. EXPDP 导出增强，支持 BFILE 列类型。

5.4.0.9 版本

1. unload object 增强，支持导出目录，同义词，数据库链接，UDT，VARRAY，用户及角色定义。
2. unload table 增强，支持 UDT 及 VARRAY 列类型。

5.3.0.6 版本

1. DROP USER 处理，恢复字典表中的删除行。
2. US7ASCII 字符集，导出 GBK 中文处理。
3. 导出损坏的 DATAPUMP 文件，此功能仅提供服务，不对外开放。

5.2 版本

1. 内置 asmcmd 增加 scan diskgroup, extract 命令，当 ASM 元数据未完全损坏时，扫描 ASM 元数据。
2. 内置 asmcmd 增加 scan asmdisk 命令，当 ASM 磁盘组彻底损坏时，直接读取 ASM 磁盘，生成数据文件。
3. 增加 scan disk 命令，当文件系统损坏时，直接读取磁盘分区或逻辑卷，生成数据文件。

5.1 版本

1. BasicFile LOB 导出性能增强，单块读改为多块读。
2. 加入 SCAN <dict 表>, 用于处理 SYSTEM 数据文件部分损坏的情况。
3. unload/undelete/untrunc 单表并行导出功能增强, 支持分区表或非分区表并行导出。
4. 增加 copyscn 实用程序，可以用来更改某个数据文件的检查点 SCN。

5.0.4.3 版本

1. LOB 列值读取错误时，错误日志写入 unload_table.log。
2. 存储过程或触发器文本末尾, 有时出现"/"未换行的处理。
3. XMLTYPE store as binary 列类型, expdp 文件错误修复。
4. SecureFile 压缩 LOB 算法完善。
5. undelete dicttab 命令块校验完善，dbv 验证块通过。
6. 18C, 19C 版本, EXPDP 格式导入 bug 修复。
7. 性能优化，导出为 EXPDP,DB 格式时，导出性能明显提升。

5.0 版本

1. 支持完整的对象定义（DDL）导出，包括表/分区表、约束、索引/分区索引、视图等对象。
2. 支持以双引号包含的小写表名。
3. 支持 Binary 存储的 XMLTYPE 列类型，XMLTYPE 只支持导出为 EXPDP 格式。
4. 数据文件和 ASM 磁盘读取改为直接 IO 方式。

产品支持列表

支持的表类型

GDUL 支持所有常见表类型。

表类型	是否支持	导出格式			
		DB	EXPDP	EXP	SQLLDR
常规表(Heap)	支持	✓	✓	✓	✓
索引组织表(IOT)	支持	✓	✓	✓	✓
聚簇表(Cluster)	支持	✕	✓	✓	✓
分区表	支持	✓	✓	✓	✓
压缩表	支持	✓	✓	✓	✓
XMLTYPE 表(CLOB)	支持	✕	✓	✓	✕
XMLTYPE 表(BLOB)	支持	✕	✓	✕	✕
嵌套表(Nested)	不支持	✕	✕	✕	✕
对象表(Object)	不支持	✕	✕	✕	✕

支持的列类型

GDUL 支持所有常见列类型。

列类型	是否支持	导出格式			
		DB	EXPDP	EXP	SQLLDR
CHAR NCHAR	支持	✓ ✓	✓ ✓	✓ ✓	✓ ✓
VARCHAR2 NVARCHAR2	支持 *12C Extended Type	✓ ✓	✓ ✓	✓ ✕	✓ ✓
NUMBER FLOAT	支持	✓ ✓	✓ ✓	✓ ✓	✓ ✓
BINARY_FLOAT BINRAY_DOUBLE	支持	✓ ✓	✓ ✓	✓ ✓	✓ ✓
DATE	支持	✓	✓	✓	✓
TIMESTAMP	支持	✓	✓	✓	✓
TIMESTAMP WITH TIME ZONE	支持	✓	✓	✓	✓

TIMESTAMP WITH LOCAL TIME ZONE	支持	✓	✓	✓	✓
INTERVAL YEAR TO MONTH	支持	✓	✓	✓	✓
INTERVAL DAY TO SECOND	支持	✓	✓	✓	✓
RAW	支持 *12C Extended Type	✓ ✓	✓ ✓	✓ ✗	✓ ✓
LONG LONG RAW	支持	✓	✓	✓	✗
ROWID	支持	✓	✓	✓	✓
UROWID	支持	✓	✓	✗	✓
BLOB CLOB	支持 *支持 SecureFile 压缩和去重存储，不支持加密。	✓	✓	✓	✓
NCLOB	支持	✓	✓	✗	✓
BFILE	支持	✗	✓	✓	✗
XMLTYPE(CLOB)	支持	✗	✓	✓	✗
XMLTYPE(BLOB)	支持 *需要 system 和 sysaux 表空间均未损坏	✗	✓	✗	✗
UDT	支持 SDO_GEOMETRY 及用户定义 UDT 类型 *impdp 需要 TRANSFORM=oid:n 选项	✓	✓	✗	✗
VARRAY	支持 *impdp 需要 TRANSFORM=oid:n 选项	✓	✓	✗	✗
ANYDATA	支持	✗	✓	✗	✗
NESTED TABLE	不支持	✗	✗	✗	✗
JSON	支持	✗	✓	✗	✗
VECTOR	支持	✓	✓	✗	✗
BOOLEAN	支持	✓	✓	✗	✗

支持的导出格式

GDUL 支持 EXPDP, EXP, DB, TEXT 四种格式。

首选格式为 DB，其次为 EXPDP 格式，EXP 仅为兼容 9i 版本。

导出类型	说明	ORACLE 版本	限制
EXPDP	导出成数据泵(datapump)格式 dump 文件。	>=10g	无
EXP	导出成旧 exp 格式 dump 文件。	>=9i	1. 不支持 NCLOB 2. 不支持 XMLTYPE 3. 不支持 UDT 和 VARRAY 4. 不支持 12C Extended VARCHAR2/RAW 类型 5. 不支持 JSON 6. 不支持 VECTOR
DB	高速直接路径(Direct Path Load)导出至新数据库。 *需要 ORACLE 客户端(OCI)可用，如果不可用，需要安装 Instant Client，或者安装完整客户端。	>=10g	直接路径导入限制： 1. 不支持 Cluster 表 2. 不支持 BFILE 3. 不支持 XMLTYPE 4. 不支持 JSON 5. 不支持 ANYDATA
SQLLDR	导出成 SQLLDR 加载的纯文本文件。	>=9i	1. 不支持 XMLTYPE 2. 不支持 JSON 3. 不支持 VECTOR

支持的文件系统

GDUL 支持所有常见主流硬件平台(HP-UX, AIX, Solaris, Linux, Windows)和文件系统。

文件系统	示例	是否支持
常规文件系统	NTFS EXT3/EXT4 JFS2	支持
集群文件系统	OCFS2 GPFS	支持
裸设备	/dev/raw*	支持
ASM 存储	N/A	支持 *不支持 Exadata *内置 asmcmd 命令

文件说明

GDUL 介质列表

安装介质	说明
gdul<版本号>.tar.gz	GDUL for Unix/Linux 版本 解压命令： $gunzip gdul<版本号>.tar.gz $tar -xvf gdul<版本号>.tar
gdul<版本号>_win64.zip	GDUL for Windows 软件介质。
GDUL 用户使用手册.pdf	用户操作手册。

GDUL 目录结构

目录	文件	说明
根目录	osetup	连接数据库，配置 ASM 磁盘、数据文件列表
	gdul.sh	GDUL 启动脚本，包含 LD_LIBRARY_PATH 设置
conf	gdul.ini	参数文件
	datafile.ini	数据文件列表
	asmdisk.ini	asm 磁盘列表
dict	*.dat	存放 SYS 用户下的字典表数据
ddl	*.sql	存放 unload object 导出的对象定义
dump	*.dmp, *.dat	存放导出 dmp、text 文件
log	gdul.log	主日志文件
	unload_table.log	表导出日志
sample	*.dict, *.dat	无 system 表空间时存放行数据采样文件
bin_file	*	各平台的 gdul 可执行文件，由 gdul.sh 调用。

gdul.ini 文件

参数	描述	取值范围	默认值
db_compat_version	数据库版本	sqlplus -v 输出	11.2.0.1
db_block_size	默认数据块大小	4096, 8192, 16384, 32768	8192
file_raw_offset	默认文件头偏移量，仅适用于 AIX	0 或 4096	0

reverse_byte	读取不同 CPU 平台的数据文件。 <windows, linux>与<AIX, HP-UX, SUN-SPARC>为不同 CPU 平台。	false, true	false
export_format	导出数据格式	SQLLDR EXP EXPDP DB	EXPDP
ldr_enclose_char	导出到文本文件时，字段分隔符	ascii 字符	|
trace_block	导出时跟踪块	false, true	False
export_db. db_conn	导出到数据库时，需要指定目标库连接字符串	连接字符串	无
export_db.buffer_size	导出到数据库时，直接路径 buffer 大小	524288, 1024000, 16777216	16777216

注意：需要明确指定 db_compat_version 的值，即 sqlplus -v 的输出。

最佳实践

运行 GDUL 程序之前，首先需要生成配置文件，包括 gdul.ini，datafile.ini，asmdisk.ini 文件，然后就可以初始化数据字典，并导出表数据。

提示：如果数据库处于打开状态，请执行 alter system checkpoint; 以便刷新数据块到数据文件。

1 生成配置文件

✓ 如果数据库处于 mount 或 open 状态，可使用 osetup 命令，自动配置参数及文件列表。

Unix/Linux 平台：osetup 能够自动配置参数文件(gdul.ini)、ASM 磁盘列表(asmdisk.ini)、数据文件列表(datafile.ini)。

Windows 平台：osetup.bat 能够自动配置 ASM 磁盘列表(asmdisk.ini)、数据文件列表(datafile.ini)。必须手工设置 gdul.ini 中的兼容性版本参数 db_compat_version。

✓ 当数据库无法 mount 时，则需要手工配置上述文件。

1) 自动生成配置文件(Linux/Unix)

$su - oracle

$ ls -rlt

```
-rwxr-xr-x 1 andy andy 4732 Feb 18 14:38 osetup*
drwxrwxr-x 2 andy andy 4096 Feb 18 14:45 bin_file/
```

1) $ *./osetup*

```
GDUL setup program, version 5.2.0.1
----------------------------------------------------------------------
ORACLE_HOME:      /u01/app/oracle/product/12.1.0
ORACLE_SID:       db12101
ORACLE_VERSION: 12.1.0.1.0
----------------------------------------------------------------------
...
setup has completed sucessfully.
```

2) 设置导出格式，默认导出成 EXPDP(conf/gdul.ini)

```
export_format:     EXPDP
```

2) 自动生成配置文件(Windows)

1) CMD> osetup.bat

```
d:\gdul_win64> osetup.bat
Please enter password for SYS (Press Enter for OS auth):
ORACLE version: [11.2.0.4.0]
ORACLE_SID:     [db11204]
Now is going to setup datafile.ini and asmdisk.ini...
```

说明：如果有多个 Oracle 实例，需要手工更改 osetup.bat，指定当前的 ORACLE_SID.

```
@echo off
@REM set ORACLE_SID if there are multiple instances.
set ORACLE_SID=test
sqlplus -S /nolog @osetup.sql
```

2) 设置 Oracle 兼容性版本，默认 11.2.0.1(conf/gdul.ini)

```
<db_compat_version>11.2.0.4</db_compat_version>
```

说明：必须指定正确的 ORACLE 版本，后续 bootstrap 加载数据字典时，需要用到 ORACLE 版本信息。

3) 设置导出格式，默认导出成 EXPDP(conf/gdul.ini)

```
export_format:     EXPDP
```

3) 手工设置配置文件

1. 更改 gdul.ini 文件

注意：需要设置 db_compat_version 为当前数据库版本，注意 12c 及以上版本需要指定详细版本号，如 12.1.0.1，请参见 sqlplus -v 输出。

```
<!--DB compatiable version, 9, 10, 11, 12-->
<db_compat_version>12.1.0.1</db_compat_version>

<!--export to SQLLDR, EXP, EXPDP-->
```

```
<export_format>EXPDP</export_format>
```

2. 添加数据文件至 datafile.ini 文件

SQL 语句：

```
set linesize 200
set pagesize 0
select ' <row> '            ||
            '<file#>'         ||file#          ||'</file#> '          ||
            '<ts#>'           ||ts#            ||'</ts#> '            ||
            '<rfile#>'        ||rfile#         ||'</rfile#> '         ||
            '<blocks>'        ||blocks         ||'</blocks> '         ||
            '<block_size>' ||block_size ||'</block_size> ' ||
            '<name>'          ||name           ||'</name> '           ||
            '<raw_offset>' ||0                 ||'</raw_offset> ' ||
            '</row>'
  from v$datafile;
```

文件最终格式：

```
<datafile>
 <row> <file#>1</file#> <ts#>0</ts#> <rfile#>1</rfile#> <blocks>256000</blocks>
<block_size>8192</block_size> <name>/datafiles/a110a64f/data/system01.dbf</name>
<raw_offset>0</raw_offset> </row>
 <row> <file#>2</file#> <ts#>1</ts#> <rfile#>2</rfile#> <blocks>211744</blocks>
<block_size>8192</block_size> <name>/datafiles/a110a64f/data/sysaux01.dbf</name>
<raw_offset>0</raw_offset> </row>
</datafile>
```

说明：

1. 文件头正常时，可以指定 file#, ts#, rfile#, blocks 项为 0，GDUL 自动从文件头中获取具体值。

2. 文件头损坏，但控制文件正常时，可以直接通过上述 SQL 获取数据文件列表。

3. 如果文件头损坏，同时控制文件也损坏时(无法 mount)，必须手工找出并指定 file#, ts#, rfile#, blocks 项，此时需要使用 scan filelist 命令，具体步骤请参考《SYSTEM 文件损坏处理》章节。

11G 版本还可以通过文本格式导出表 SYS.WRH$_DATAFILE 内容，但 12.2 版本未收集 SYS.WRH$_DATAFILE 内容，原因：Bug 25416731 TABLESPACE IO STATISTICS MISSING FROM AWR REPORT, Support 文档：Awr Report in 12.2 Missing Tablespace IO Stats, FIle IO Stats (Doc ID 2333859.1)。

4. blocks 可以设置为 0，GDUL 打开数据文件时，自动根据 block_size 获取块数。

AIX 平台，数据文件存放在裸设备时的偏移量

AIX 平台，如果数据文件存放在设备(/dev/r*)，GDUL 会自动检测数据文件偏移量，由参数 auto_raw_offset 控制。该参数默认是启用状态，此时，datafile.ini 中的 offset 值会忽略。

如果遇到极端情况，无法判断裸设备数据文件的偏移量时，可以设置 auto_raw_offset 为 false，然后手工在 datafile.ini 中指定偏移量的值为 0 或 4096。

手工判断方法：使用 dbfsize 命令可看是否有 4k 头。

下面是包含 offset 的裸设备

$dbfsize /dev/<RLV_DATAFILE_NAME>

```
Database file: /dev/<RLV_DATAFILE_NAME>
Database file type:raw device   具有 4k 头时显示
Database file type:raw device without 4K starting offset   无 4k 头时显示
Database file size: ##### #### byte blocks
```

3. 添加 ASM 磁盘至 asmdisk.ini 文件

SQL 语句：

```
set linesize 200
select /*+RULE*/
        '<row> '                ||
        '<group_number>'   ||g.group_number ||'</group_number> '     ||
          '<group_name>'   ||g.name              ||'</group_name> '       ||
          '<block_size>'     ||g.block_size     ||'</block_size> '       ||
          '<au_size>'        ||g.allocation_unit_size ||'</au_size> '||
        '<disk_number>'     ||d.disk_number    ||'</disk_number> '      ||
        '<path>'                ||d.path              ||'</path> '               ||
          '<f1b1>'              ||0                   ||'</f1b1> '               ||
        '<raw_offset>'   ||0                   ||'</raw_offset> '       ||
        '</row>'
  from v$asm_disk d
```

```
 inner join v$asm_diskgroup g
      on d.group_number = g.group_number
   order by g.group_number, d.disk_number;
```

文件最终格式：

```
<asmdisk>
asm disk 列表，非 asm 环境为空行。
</asmdisk>
```

4) ASM 磁盘注意事项

1. ASM 磁盘权限问题

GDUL 一般在 ORACLE 用户下执行，但是 ASM 磁盘文件的属主通常情况下为 grid:asmadmin，所以大多数情况下，ORACLE 用户无读取 ASM 磁盘文件的权限。

因此运行 GDUL 前，建议先查看 ASM 磁盘文件的权限，并把 asmadmin 组赋给 ORACLE 用户。

#vi /etc/group，在 asmadmin 组名后添加 oracle 用户名。

2. 使用 ASMLIB 或 AFD 的磁盘路径

如果 ASM 使用 ASMLIB 或 12C 的 AFD，则需要删除 asmdisk.ini 中的虚拟磁盘，并添加/保留真实的磁盘路径。

查看所有 ASM 磁盘路径：

$ kfod disks=all

$ asmcmd dsget

$ asmcmd afd_lsdsk

$ /sbin/blkid |grep oracleasm

虚拟磁盘对应真实磁盘路径：

ASM 类型	虚拟磁盘名	真实磁盘路径
ASMLIB	ORCL:磁盘名	/dev/oracleasm/disks/磁盘文件
AFD(12c)	AFD:磁盘名	/dev/oracleasm/asm-disk*

注意：

1. ASMLIB 环境，/dev/oracleasm/disks/下无法查看 drop 掉的 disk，/dev/mapper/下对应的的磁盘可能会有 1M 或其它 offset 的磁盘头，可用 kefd 确认。

   可以读取 aun=0, 1, 2, 3 的块 0, 确认是否有偏移量。

   ```
   # /u01/app/11.2.0/grid_1/bin/kfed read /dev/mapper/mpathX aun=0 blkn=0
   ```

   如果磁盘有 offset，则需要在 asmdisk.ini 里手工配置 raw_offset 值。

3. AIX 平台使用 VG 中的 RLV 作为 ASM 磁盘

如果 ASM 磁盘使用 VG 中的裸 LV，可能会有 4k 的 LV 头。GDUL 会自动检测 RLV 的偏移量，由参数 asm.auto_raw_offset 控制。该参数默认是启用状态，此时，asmdisk.ini 中的 offset 值会忽略。

如果遇到极端情况，无法判断裸设备的偏移量时，可以设置 asm.auto_raw_offset 为 false，然后手工在 asmdisk.ini 中 offset 的值为 0 或 4096。

手工判断方法：查看 lslv 命令的 DEVICESUBTYPE 确认。

$lslv -L <LV_NAME>

DEVICESUBTYPE 属性值	**是否有 4k 头**
DS_LVZ	无
DS_LV	有
未显示 DEVICESUBTYPE	有

4. Windows 平台 ASM 磁盘

Windows 平台，ASM 磁盘为虚拟路径，先查询《添加 ASM 磁盘至 asmdisk.ini 文件》章节中的 SQL，获取 ASM 磁盘列表，asmdisk.ini 中的磁盘路径格式如下：

\\.\ORCLDISKDATA0

\\.\ORCLDISKDATA1

… …

以下命令查看磁盘列表:

CMD> asmtool -list

5) CDB 数据库注意事项

对于 12C 及以上版本，如果数据库为 CDB，则需要在配置完后，把 datafile.ini 配置文件中只保留一个 PDB 的数据文件列表。

2 初始化数据字典，导出数据

1) 执行 GDUL 命令，初始化数据字典

$./gdul.sh

GDUL> bootstrap

```
Bootstrap finish.
```

注意：

1. bootstrap 读取 SYSTEM 表空间中的数据字典，如果数据库处于 OPEN 状态，可能需要执行 alter system checkpoint;命令来同步数据文件，然后再执行该命令。

2. ASM 环境，如果 GDUL 以 oracle 执行，则需要把 grid 用户下的 asmadmin 组赋给 oracle 用户，重新登陆 oracle 用户，再运行 gdul.sh。

2) 显示数据库信息

显示数据文件列表

GDUL> info

FILE#	TS#	RFILE#	BIGFILE	SIZE(GB)	NAME
1	0	1	FALSE	0.68	D:\APP\ORADATA\TEST32\SYSTEM01.DBF
2	1	2	FALSE	0.59	D:\APP\ORADATA\TEST32\SYSAUX01.DBF
3	2	3	FALSE	1.01	D:\APP\ORADATA\TEST32\UNDOTBS01.DBF
4	4	4	FALSE	8.88	D:\APP\ORADATA\TEST32\USERS01.DBF
5	5	5	FALSE	0.13	D:\APP\ORADATA\TEST32\TBS01.DBF
6	6	6	FALSE	0.01	D:\APP\ORADATA\TEST32\DMP01.DBF
7	4	7	FALSE	4.00	D:\APP\ORADATA\TEST32\USERS02.DBF
8	7	9	FALSE	0.01	D:\APP\ORADATA\TEST32\TBSFNO01.DBF
… …					

显示数据库用户列表

GDUL> list user

```
ID          NAME                TABLE_CNT
---------- ------------------- -----------
```

```
0           SYS                    1258
7           AUDSYS                  1
8           SYSTEM                  178
…
109         OE                     14
110         SCOTT                  4
…
```

显示当前用户下各类对象

GDUL> list object

```
OBJECT_TYPE                CNT
-------------- ----------
TABLE                     1051
VIEW                      3877
SEQUENCE                   117
TRIGGER                     5
PROCEDURE                  111
FUNCTION                    99
PACKAGE                    605
PACKAGE BODY               580
```

显示当前用户下的表

GDUL>set user andy

GDUL>list table

```
ID          NAME                          DICT_ROWS    DICT_BLOCKS
---------- ----------------------------- ---------- ----------
15727       T_OBJ1                        0            0
16814       T_TEST_TYPE                   2            13
16824       T_PERSON                       2            5
17065       T_ROWID                        2            5
17070       T_RAW                          5            9
17252       T_HELLO                       0            0
```

显示表结构

GDUL>desc andy.t_test

```
object_id: 18969, dataobj#: 35558, cluster tab#: 0
segment header: (ts#: 4, rfile#: 4, block#: 138))

Seg Column#   Column#      Name                    Null?          Type
------------ ---------- -------------------- --------------- --------------
1                1        OWNER                                  VARCHAR2(30)
2                2        OBJECT_NAME                            VARCHAR2(128)
3                3        SUBOBJECT_NAME                         VARCHAR2(30)
4                4        OBJECT_ID                             NUMBER
5                5        DATA_OBJECT_ID                         NUMBER
```

说明：可以用 FULL 选项显示分区。

3) 导出表数据

说明：

1. 如果数据库处于打开状态，建议先执行 SQL>alter system checkpoint;以便刷新磁盘数据块。
2. 导出的表位于 dump 目录，然后可以根据导出类型(expdp, exp, sqlldr)，再导入到表中。

导出单张表

示例 1：导出单张表：

GDUL> unload table andy.t_compart

```
2017-05-04 18:01:09 unloading table "ANDY"."T_COMPART"...
2017-05-04 18:01:09 unloaded 3 rows.
```

示例 2：导出单个表分区，分区名可以为复合分区或子分区：

GDUL> unload table andy. t_compart:p1

```
2017-05-04 17:59:37 unloading table "ANDY"."T_COMPART":"P1"...
2017-05-04 17:59:37 unloaded 2 rows.
```

示例 3：并行导出分区表，每个线程导出一个文件

GDUL> unload table t_compart parallel 5

```
2017-05-04 17:58:35 unloading table "ANDY"."T_COMPART" in parallel 5...
[00] 2017-05-04 17:58:35 unloading table T_COMPART:SP1...
[01] 2017-05-04 17:58:35 unloading table T_COMPART:SP2...
[02] 2017-05-04 17:58:35 unloading table T_COMPART:SP3...
[03] 2017-05-04 17:58:35 unloading table T_COMPART:SP4...
[00] 2017-05-04 17:58:35 table T_COMPART:SP1 unloaded 1 rows.
[01] 2017-05-04 17:58:35 table T_COMPART:SP2 unloaded 1 rows.
[02] 2017-05-04 17:58:35 table T_COMPART:SP3 unloaded 0 rows.
[03] 2017-05-04 17:58:35 table T_COMPART:SP4 unloaded 1 rows.
2017-05-04 17:58:35 unloaded 3 rows.
```

导出用户下所有表

GDUL>unload user andy [parallel #]

```
About to unload ANDY's tables, total cnt: 43
2016-02-18 14:55:37 unloading table DEPT...
2016-02-18 14:55:37 unloaded 4 rows.
2016-02-18 14:55:37 unloading table EMP...
2016-02-18 14:55:37 unloaded 14 rows.
2016-02-18 14:55:37 unloading table DEMO_TAGS...
2016-02-18 14:55:37 unloaded 6 rows.
2016-02-18 14:55:37 unloading table DEMO_TAGS_TYPE_SUM...
2016-02-18 14:55:37 unloaded 3 rows.
```

说明：导出整个用户下的所有表时，可以用 parallel [并行数]来加快导出速度。

4) 导出 delete 的表数据

当表数据被 delete 后，如果数据块未被覆盖，则可以用 undelete 命令恢复被删除的数据。

限制如下：

1. 对于链接行，如果单列(如 LONG, VARCHAR)也出现链接，由于已删除的行片中无法判断列链接标记，会导致问题。
2. 对于 OLTP 压缩表，如果已删除的行片对应的符号表行已被删除，将无法导出。
3. LOB 表需要先扫描 LOB 列所在表空间#。

示例 1：导出单张表。

GDUL> undelete table andy.t_part2

```
2017-05-05 16:25:29 undeleting table "ANDY"."T_PART2"...
2017-05-05 16:25:29 undeleted 7 rows.
```

示例 2：导出单个表分区，分区名可以为复合分区或子分区。

GDUL> undelete table andy.t_part2:PART1_1

```
2017-05-05 16:26:47 undeleting table partition "ANDY"."T_PART2":"PART1_1"...
2017-05-05 16:26:47 undeleted 7 rows.
```

示例 3：并行导出表，每个线程导出一个文件。

GDUL> undelete table t_compart parallel 5

```
GDUL> undelete table andy.t_part2 parallel 5
2017-05-05 16:30:38 undeleting table "ANDY"."T_PART2" in parallel 5...
[00] 2017-05-05 16:30:38 undeleting table T_PART2:PART1_1...
[01] 2017-05-05 16:30:38 undeleting table T_PART2:PART1_2...
[02] 2017-05-05 16:30:38 undeleting table T_PART2:PMAX...
[00] 2017-05-05 16:30:38 table T_PART2:PART1_1 undeleted 1479 rows.
[02] 2017-05-05 16:30:38 table T_PART2:PMAX undeleted 0 rows.
[01] 2017-05-05 16:30:38 table T_PART2:PART1_2 undeleted 8527 rows.
2017-05-05 16:30:38 undeleted 10006 rows.
```

示例 4：导出 LOB 表。

GDUL>scan tablespace 10

GDUL> undelete table andy.t_lob

```
2017-05-05 16:25:29 undeleting table "ANDY"."T_LOB"...
2017-05-05 16:25:29 undeleted 7 rows.
```

说明：LOB 表需要先扫描 LOB 列所在表空间#。

5) 导出 truncated 的表数据

1. Truncate 后无新数据 insert 进来

如果表被 truncate 后，无新数据 insert，可以直接使用 untrunc 命令来恢复。

1) 扫描表空间

*查看表所在的表空间，分区表可以用 FULL 选项查看

GDUL> desc andy.t_test [full]

```
object_id: 108775, dataobj#: 112379, cluster tab#: 0
segment header: (ts#: 7, rfile#: 5, block#: 562))
Seg Column#    Column#       Name                      Null?              Type
-----------  ----------  -------------------  --------------  --------------
1                  1              OWNER                                    VARCHAR2(30)
…
```

*扫描表空间

GDUL> scan tablespace 7

```
start scan tablespace 7...
scan tablespace completed.
```

说明：如果磁盘性能较好，且表空间下有多个数据文件，可以采用并行方式。

GDUL> scan tablespace 7 parallel 10

2) 确认 unload 结果为 0

GDUL> unload table andy.t_test

```
2016-02-18 15:03:39...unloading table T_TEST 0 rows unloaded.
```

3) 导出 truncated 的行

GDUL> untrunc table andy.t_test

```
2016-02-18 15:04:03...untruncating table T_TEST 99998 rows unloaded.
```

说明：untrunc 和 unload/undelete 类似，可以按分区及并行导出。

2. Truncate 后有新数据 insert 进来

如果 truncated 后，有新数据进来，将无法自动判断出旧的 data_object_id。需要先找到 truncate 前的 data_object_id，再用 unload table 恢复。

1) 扫描表空间

*查看表所在的表空间，分区表可以用 FULL 选项查看

GDUL> desc andy.t_test [full]

GDUL> scan tablespace 7

2) 查找 truncate 前的 data_object_id。
步骤请参考《查找表对应的 data_object_id》章节。

3) 找到 data_object_id 后，便可以恢复

GDUL>unload table andy.t_test object_id <data_object_id>

6) 导出 drop 的表数据

1) 创建一张临时的表，表结构和 drop 掉的表相同。
SQL>create table t_test2... ... tablespace system;
SQL>alter system checkpoint;

说明：需要借助上述新建表的表结构来恢复 drop 的表。
1. 表所在表空间设置为非 drop 表所在的表空间，以免数据块被覆盖。
2. 如果有 LOB 列，必须把 LOB 表空间指向原来的表的 LOB 所在的表空间。

```
CREATE TABLE SYS.T_BASIC_CLOB_INLINE (ID NUMBER, NAME CLOB) TABLESPACE
SYSTEM
 LOB (NAME) STORE AS BASICFILE (TABLESPACE "TEST" ENABLE STORAGE IN ROW);
```

如果 LOB 表空间和原来的表不在同一个表空间，则需要在 bootstrap 后，手工更改 lob.dat, lobpart.dat 中的 ts#，更改为 drop 前 LOB 所在的表空间。

2) 初始化数据字典，以识别新创建的表。
GDUL>bootstrap

3) 扫描 drop 掉的表所在表空间，得到表空间内所有 data_object_id 及对应的数据块。
GDUL> scan tablespace 7
说明：该步骤建立指定表空间下所有 data_object_id 和数据块的对应关系，在接下来的步骤 4，步骤 5 都需要用到。
说明 2：如果磁盘性能较好，且表空间下有多个数据文件，可以采用并行方式。
GDUL> scan tablespace 7 parallel 10

4) 查找 drop 表当时的 data_object_id。
步骤请参考《查找表对应的 data_object_id》章节。

5) 借助步骤 1 创建的表结构，用步骤 4 找到的 drop 掉的表的 data_object_id 导出表数据
GDUL> unload table andy.t_part1 object_id 79435

```
2017-05-05 17:02:47 unloading table "ANDY"."T_PART1" with data_object_id 79435...
2017-05-05 17:02:47 unloaded 1480 rows.
```

7) 导出对象定义

GDUL 支持导出多种对象定义，目前支持表，索引，触发器，存储过程，函数，包，视图，目录，同义词，数据库链接，用户及角色定义。

1. 显示对象列表

GDUL>set user ANDY

GDUL>list object

OBJECT_TYPE	CNT
TABLE	1053
INDEX	1733
VIEW	3877
SEQUENCE	117
TRIGGER	5
CLUSTER	11
PROCEDURE	111
FUNCTION	99
PACKAGE	605
PACKAGE BODY	580

GDUL> list proc

ID	NAME
45469	P_TEST
45564	P_TEST3
131498	P_2

2. 导出用户对象定义

GDUL>unload object ANDY

```
2018-09-07 12:24:35 unloading ANDY's TABLE DDL(total 396) to
"dump/ANDY_TABLE_DDL.sql"...
2018-09-07 12:24:35 unloading ANDY's CLUSTER DDL(total 8) to
"dump/ANDY_CLUSTER_DDL.sql"...
  2018-09-07 12:24:35 unloading ANDY's TABLE FULL DDL(tot al 396) to
"dump/ANDY_TABLE_FULL_DDL.sql"...
```

```
 2018-09-07 12:24:35 unloading ANDY's TABLE CONSTRAINT DDL to "dump/ANDY_TABLE_CONS_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's TABLE INDEX DDL to "dump/ANDY_TABLE_INDEX_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's VIEW DDL(total 11) to "dump/ANDY_VIEW_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's SEQUENCE DDL(total 33) to "dump/ANDY_SEQ_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's TRIGGER DDL(total 2) to "dump/ANDY_TRIGGER_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's PROCEDURE DDL(total 4) to "dump/ANDY_PROC_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's FUNCTION DDL(total 5) to "dump/ANDY_FUNC_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's PACKAGE DDL(total 47) to "dump/ANDY_PACKAGE_DDL.sql"...
 2018-09-07 12:24:35 unloading ANDY's PACKAGE_BODY DDL(total 47) to "dump/ANDY_PACKAGE_BODY_DDL.sql"...
finish.
```

3. 导出数据库级对象定义

说明：导出数据库级对象定义，如角色，目录等。

GDUL>unload object DB

8) 表有坏块场景恢复

导出单张表，字典正常，但段头损坏

段头损坏症状：

GDUL> unload table andy.t_test_corrupt_header

```
 2016-05-09 11:13:19 unloading table "ANDY"."T_TEST_CORRUPT_HEADER"...
   unload table error: segment header parse error, object_id: 36399, dba[6, 10, 130]: block tail checksum error, consistency value in tail: 0xc9312301
 2016-05-09 11:13:19 unloaded 0 rows.
```

扫描表空间#：

GDUL> desc andy.t_test_corrupt_header

```
object_id: 36399, dataobj#: 36399, cluster tab#: 0
```

```
segment header: (ts#: 6, rfile#: 10, block#: 130))

Seg Column#    Column#      Name                    Null?          Type
------------ ---------- -------------------- --------------- --------------
1              1            OWNER                                VARCHAR2(30)
2              2            OBJECT_NAME                          VARCHAR2(128)
3              3            SUBOBJECT_NAME                       VARCHAR2(30)
4              4            OBJECT_ID                            NUMBER
…
```

GDUL> scan tablespace 6

```
start scan tablespace 6...
scan tablespace completed.
```

使用 **scan** 选项导出表：

GDUL> unload table andy.t_test_corrupt_header scan

```
2016-05-09 11:13:51 unloading table "ANDY"."T_TEST_CORRUPT_HEADER"...
2016-05-09 11:13:51 unloaded 124 rows.
```

导出单张表，列定义不完整(字典中出现坏块引起)

列定义不完整现象：

GDUL> desc andy.t_test_corrupt_header

```
object_id: 36399, dataobj#: 36399, cluster tab#: 0
segment header: (ts#: 0, rfile#: 0, block#: 0))
Seg Column#    Column#      Name                    Null?          Type
------------ ---------- -------------------- --------------- --------------
```

1. 由于列定义不完整，需要找到表定义，并在正常库创建测试表，然后拷贝 **dict/col$**中对应的行至损坏库的 **GDUL dict/col$**中。

GDUL> desc andy.t_test_corrupt_header

```
object_id: 36399, dataobj#: 36399, cluster tab#: 0
segment header: (ts#: 0, rfile#: 0, block#: 0))
Seg Column#    Column#      Name                    Null?          Type
------------ ---------- -------------------- --------------- --------------
1              1            OWNER                                VARCHAR2(30)
2              2            OBJECT_NAME                          VARCHAR2(128)
```

3	3	SUBOBJECT_NAME	VARCHAR2(30)
4	4	OBJECT_ID	NUMBER
…			

2. 扫描表空间#：

GDUL> scan tablespace 6

```
start scan tablespace 6...
scan tablespace completed.
```

3. 使用 object_id 选项导出表：

GDUL> unload table andy.t_test_corrupt_header object_id 36399

```
2016-05-09 11:13:51 unloading table "ANDY"."T_TEST_CORRUPT_HEADER"...
2016-05-09 11:13:51 unloaded 124 rows.
```

LOB 表，LOB 索引损坏

LOB 索引损坏症状：

GDUL> unload table andy.t_test_outline_lob

```
2018-09-12 19:20:18 unloading table "ANDY"."T_TEST_OUTLINE_LOB"...
2018-09-12 19:20:18 unloaded 1 rows.
```

日志显示：

```
20180912 19:20:18 [obj#: 133812] table "ANDY"."T_TEST_OUTLINE_LOB" unloading...
20180912 19:20:18 [obj#: 133812, dataobj#: 133812, tab#: 0] use extent map: 1
20180912 19:20:18 [data_object_id: 133812, column# 3] start init LOB Index (column DEMO).
20180912 19:20:18 [data_object_id: 133812, column# 3] Failed to init LOB Index, msg: segment header parse error, object_id: 0, dba[22, 6, 170]: Invalid segment header block type dba:[22, 6, 170], block_type:238.
20180912 19:20:18 [data_object_id: 133812, column# 3] Warning: LOB value with "STORE AS BASICFILE" may incomplete!
```

说明：如果 LOB 类型为 BasicFile 格式，并且需要用到 LOB 索引，将无法获取完整的 LOB 值。

使用 scan 扫描 LOB 表空间，再导出表：

GDUL> scan tablespace 22

```
start scan tablespace 22...
scan tablespace completed.
```

GDUL> unload table andy.t_test_outline_lob

```
2018-09-12 19:25:11 unloading table "ANDY"."T_TEST_OUTLINE_LOB"...
2018-09-12 19:25:11 unloaded 1 rows.
```

3 drop user 恢复(从字典表数据块中恢复被删除行)

理想状态下，drop user 执行完，字典表被删除的行还未被清除，此时可以通过恢复数据块中的被删除行，再 bootstrap 获取新的字典信息。

1. 拷贝 SYSTEM.dbf 文件，并更改 datafile.ini 文件中的 system 文件指向该拷贝文件。请勿直接更改生产库中的 SYSTEM 数据文件。

2. GDUL>bootstrap

GDUL>list user

由于用户 X 已经被删除，此时无法显示。

3. 恢复字典表中的被删除行，此步骤直接更改数据块，请勿直接更改生产库中的 SYSTEM 数据文件。

GDUL>undelete block USER$

GDUL>undelete block TAB$

GDUL>undelete block COL$

GDUL>undelete block OBJ$

GDUL>undelete block CLU$

GDUL>undelete block SEQ$

GDUL>undelete block SOURCE$

GDUL>undelete block TRIGGER$

GDUL>undelete block VIEW$

GDUL>undelete block CON$

GDUL>undelete block CDEF$

GDUL>undelete block CCOL$

GDUL>undelete block TABPART$

GDUL>undelete block TABCOMPART$

GDUL>undelete block TABSUBPART$

GDUL>undelete block LOB$

GDUL>undelete block LOBCOMPPART$

GDUL>undelete block LOBFRAG$

GDUL>undelete block IND$

GDUL>undelete block ICOL$

GDUL>undelete block INDPART$

GDUL>undelete block INDCOMPART$

GDUL>undelete block INDSUBPART$

GDUL>undelete block PARTOBJ$

GDUL>undelete block PARTLOB$

GDUL>undelete block PARTCOL$

GDUL>undelete block SUBPARTCOL$

4. GDUL>bootstrap

GDUL>list user

用户已经可以正常显示。

5. GDUL>set user X

GDUL>list table

GDUL>unload object X --生成用户下所有对象的 DDL。

6. 导出用户表数据

GDUL>scan database parallel 2

GDUL>unload user X scan parallel 2

4 drop user 及 drop tablespace 恢复(测试库中有表结构)

说明：drop user 和 drop tablespace(数据文件还在)和无 SYSTEM 表空间恢复方法类似，都需要新建表结构，然后导出数据文件中被删除表的数据。

以下步骤使用测试库中的字典数据做恢复。

1) 测试库建立用户及表结构

create user XXX;

create table XXX;

alter system checkpoint;

GDUL>bootstrap

GDUL>list user 等确认读取字典。

GDUL>exit

注意：如果表中存在 LOB 列，则需要在 bootstrap 后，手工更改 lob.dat, lobfrag.dat 中的 ts# 为之前 LOB 所在的表空间。

2) 扫描删除的表空间

1. 添加被删除用户或表空间的数据文件到 conf/datafile.ini。

2. 扫描被删除用户或表空间对应的表空间

GDUL> info，确认 ts#

GDUL> scan tablespace <ts#>

说明：该操作会输出到 dict/scan_*.dat 文件中。

1. 如果涉及多个表空间，把 datafile.ini 中无关的数据文件先去掉，再 scan database 来扫描整个数据库。

2. 默认扫描线程数为 1，如果磁盘性能较好，且表空间下有多个数据文件，可以采用并行方式，以加快扫描速度。

示例：*GDUL> scan tablespace 7 parallel 10*

3) 获取被删除的表对应的 data_object_id

1. 尝试从**生产库**的 SYS.wrh$_seg_stat_obj 表获取<表, data_obj#>对应关系。

详细步骤请参考《查找表对应的data_object_id》章节。
理想状态可以从中找到表名和 data_object_id 的对应关系，如果该表中没有相关信息，则需要采样表空间，然后手工确认。

2. 手工采样方法

GDUL>sample segment all

$cd sample

说明：该操作执行采样，并在 sample 目录下生成 segment 列定义及采样数据。

进入 sample 目录，查看采样数据，然后对照测试库，查找是哪张表的数据。

4) unload 采样出的数据段

GDUL>unload table 用户名.表名 object_id <分析出的 data object id>

5 查找表对应的 data_object_id

在数据字典损坏，表被 truncate（新数据进来），表被 drop，表空间被 drop 等操作后，往往需要先找到 data_object_id，再通过 unload table object_id <data_object#>来恢复表数据。

以下为查找 data_object_id 方法：

方法 1：使用闪回查询

如果 SYS UNDO 段未被覆盖的话，可以闪回查找旧的信息。

```
select obj#, dataobj# from sys.obj$
  as of timestamp to_timestamp('2017-02-08 19:00:00','YYYY-MM-DD
HH24:MI:SS')
  where owner# = (select user_id from dba_users where username='ANDY')
  and name='T_TEST';
```

此外，还可以获取 drop 掉的表结构：

```
select column_name, data_type, data_length
  from dba_tab_columns as of timestamp to_timestamp('2017-02-08
19:00:00','YYYY-MM-DD HH24:MI:SS')
 where owner = 'ANDY'
  and table_name='T_TEST';
```

方法 2：AWR 表中可能会记录有旧的 data_object_id.

```
select s.snap_id, s.begin_interval_time, o.obj#, o.ts#, o.dataobj#
 from sys.wrh$_seg_stat_obj o
inner join sys.wrm$_snapshot s
   on o.snap_id = s.snap_id
where owner='ANDY'
  and object_name = 'T_TEST'
order by s.snap_id;
```

```
select snap_id, ts#, obj#, dataobj#, object_type
  from sys.wrh$_seg_stat_obj
 where owner='ANDY'
   and object_name = 'T_TEST'
 order by snap_id;
```

方法 3：手工 sample 表空间，查找

GDUL>sample segment all

$cd sample，手工分辨所有的采样数据，最后确认 data_object_id。

方法 4：使用 logminer

先找到 truncate 时的归档或在线日志，然后从 logminer 输出中查找 truncate 的 DDL 语句，truncate 后几行记录 update "SYS"."TAB$"和 update "SYS"."OBJ$"中会记录有旧的 data_object_id。

```
select * from v$log;
select * from v$logfile;
select thread#, sequence#, name, first_time, next_time from
v$archived_log
where to_date('2017-02-08 19:00:00','YYYY-MM-DD HH24:MI:SS') between
first_time and next_time;

exec
dbms_logmnr.add_logfile('/oradata/db10205/redo03.log',dbms_logmnr.new);
exec
dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);
drop table test_log;
create table test_log tablespace sysaux
as select * from v$logmnr_contents;
execute dbms_logmnr.end_logmnr;
```

6 特殊表及列类型

1) LOB 列类型

ORACLE 支持两类 LOB，CLOB 和 BLOB。

11G 以后版本还支持 SecureFile 存储格式的 LOB，以提高读写性能。

LOB 列类型支持导出为 EXPDP，EXP，DB，SQLLDR 格式。

当读取 LOB 出现坏块时，会报如下错误：

[lob id: 000100000c4f4add] tab_obj#: 89453, col#: 19, error: can't parse Lob chunk 0, datablock <5,16,1891759>, msg:Not a valid LOB block。

可通过以下两种方式，确认损坏的 LOB 所在的行：

1. 数据库处于打开状态时，可以通过以下方式查询涉及 LOB 坏块的行：

```
set serverout on
declare
  val clob;
begin
  for rec in(select rowid from 表名)
  loop
    begin
      select demo
      into val
      from 表名
      where rowid = rec.rowid;
    exception
      when others then
        dbms_output.put_line(rec.rowid);
    end;
  end loop;
end;
/
```

2. GDUL 导入至新数据库表后，查询 EMPTY 的值，确认损坏的 LOB 位置：

GDUL 把未能获取的 LOB 值，设置为 EMPTY，因此可通过以下查询定位。

```
  select * from t_clob_table
   where (lob_col is not null)
     and (dbms_lob.getlength(lob_col) = 0);
```

2) XMLTYPE 列类型

ORACLE 支持两类 XMLTYPE 存储格式，CLOB 和 Binary(BLOB)。

GDUL 处理时，导出库和导入库表结构中的 xmltype 列存储类型**必须**一致，即同时为 CLOB 或 Binary 存储方式。

支持导出格式：EXPDP 格式。

XMLTYPE (store as CLOB)

对于 CLOB 存储格式中的 XML 内容，ORACLE 不做校验。

处理步骤：

1. 通过 unload object 用户名，导出对象定义。
2. 搜索含 xmltype 类型的字符串，进而得知表名。
   对于 CLOB 存储格式，DDL 选项为：`XMLTYPE COLUMN "列名" STORE AS SECUREFILE CLOB;`
3. 针对 xmltype 表，设置导出为 expdp 格式。
4. 数据库中建立表结构，然后 impdp 导入。
5. 校验 xmltype 列是否有损坏的列值：

当获取内容失败(即列值损坏)时，GDUL 把 XMLTYPE 长度设置为 0，该存储值为非法的 XMLTYPE 值。

请通过以下查询未能导出的 XMLTYPE 值，然后设置为 NULL。

```
select * from t_xmltype_clob
 where (xml_col is not null)
   and (dbms_lob.getlength(XMLTYPE.getclobval(xml_col)) = 0);
```

等手工确认有问题的 CLOB 列值后，再设置为 NULL。

```
update t_xmltype_clob set xml_col = null
where (xml_col is not null)
   and (dbms_lob.getlength(XMLTYPE.getclobval(xml_col)) = 0);
```

如果由 CLOB 格式的 XMLTYPE，导入到 BinaryXML 格式，当 GDUL 生成的内容含有 0 长度的非法 xmltype 值，将会报以下错误：

<

Processing object type TABLE_EXPORT/TABLE/TABLE_DATA

ORA-31693: Table data object "ANDY"."CON_ITEM" failed to load/unload and is being skipped due to error:

ORA-29913: error in executing ODCIEXTTABLEFETCH callout

ORA-31011: XML parsing failed

ORA-19202: Error occurred in XML processing

LPX-00229: input source is empty

>

XMLTYPE (store as BINARY XML)

由于 Binary 存储格式的特殊性，该格式有如下限制：

需要 system 和 sysaux 表空间均未损坏。

导出前，请确认 bootstrap 已成功获取 XMLDB 元数据

1) log/gdul.log 日志具有以下输出：

..XDB.X$QNXXXX table, rows ###.

..XDB.X$NMXXXX table, rows ###.

2)dict/xdb_nmspc_token.dat 和 dict/xdb_qname_token.dat 文件均有数据。

注意：如果无法获取 XMLDB 元数据，即使 impdp 成功导入 xmltype 列，当查询或者 expdp 导出时，也极可能会报 ORA-600[qmcxdGetQNameInfo2]错误。

处理步骤：

1. 通过 unload object 用户名，导出对象定义。
2. 搜索含 xmltype 类型的字符串，进而得知表名。
   对于 BLOB 存储格式，DDL 选项为：`XMLTYPE COLUMN "列名" STORE AS SECUREFILE BINARY XML;`
3. 针对 xmltype 表，导出表为 expdp 格式。
4. 数据库中建立表结构，然后 impdp 导入。
5. 校验 xmltype 列是否有损坏的列值：

```
declare
  len number;
begin
   for i in(select rowid from 用户名.表名)
   loop
     begin
       select dbms_lob.getlength(XMLTYPE.getclobval(xml列)) into len
         from 用户名.表名
        where rowid = i.rowid;
     exception
       when others
         then
           dbms_output.put_line(i.rowid || ' ' || SQLERRM);
```

```
        end;
      end loop;
  end;
  /
```

说明：如果列值损坏，SQLERRM 错误消息可能为 ORA-600[qmcxdGetQNameInfo2]错误，该错误会影响表的 expdp 导出。一旦发现这些列值，需要 update 为 NULL，再进行 expdp 命令导出。

错误信息：
说明：GDUL 生成的 EXPDP 文件中包含有 XMLTYPE，导出表和导入表格式(CLOB, BINARY XML)必须相同，否则可能会报以下错误。

Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
ORA-31693: Table data object "ANDY"."CON_ITEM" failed to load/unload and is being skipped due to error:
ORA-29913: error in executing ODCIEXTTABLEFETCH callout
ORA-31011: XML parsing failed
ORA-19213: error occurred in XML processing at lines 1
LPX-00210: expected '<' instead of

Job "ANDY"."SYS_IMPORT_TABLE_01" completed with 1 error(s) at Mon May 31 16:37:13 2021 elapsed 0 00:00:19

3) UDT/VARRAY 列类型

UDT 和 VARRAY 列类型支持导出为 EXPDP 和 DB 格式。

*impdp 导入语法如下：

```
impdp andy/andy directory=GDUL_DIR tables=t_test_udt dumpfile=andy_t_test_udt.dmp
TRANSFORM=oid:n
```

说明：IMPDP时，需要加入TRANSFORM=oid:n选项。

4) 关于 12C guard-column 列

12C 版本，当表上新增带默认值的、且允许为 NULL 的列时，会进行 DDL 优化，添加系统生成的隐含列 SYS_NC0*，类型为 RAW(126)。

示例：

```
create table t_test1(id number, name varchar2(20));
insert into t_test1(id , name) values(1, '11');
commit;
```

```
--add two columns with allow NULL and default value
alter table t_test1 add val1 NUMBER (1) DEFAULT 0;
alter table t_test1 add UPDATE_TIME TIMESTAMP (6) DEFAULT CURRENT_TIMESTAMP;
insert into t_test1(id , name) values(2, '22');
commit;
alter system checkpoint;
```

GDUL> desc andy.t_test1 full

Column#	Int Column#	Seg Column#	Name	Null?	Type	Property
1	1	1	ID		NUMBER	0
2	2	2	NAME		VARCHAR2(20)	0
0	3	3	SYS_NC00003$		RAW(126)	288
3	4	4	VAL1		NUMBER(1)	1073741824
4	5	5	UPDATE_TIME		TIMESTAMP(6)	1073741824

导入时会失败：

```
$ impdp andy/andy directory=GDUL_DIR tables=andy.t_test1 dumpfile=ANDY_T_TEST1.dmp
REMAP_TABLE=ANDY.t_test1:t_test5

Starting "ANDY"."SYS_IMPORT_TABLE_02":  andy/******** directory=GDUL_DIR
tables=andy.t_test1 dumpfile=ANDY_T_TEST1.dmp REMAP_TABLE=ANDY.t_test1:t_test5
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
ORA-31693: Table data object "ANDY"."T_TEST5" failed to load/unload and is being
skipped due to error:
ORA-02354: error in exporting/importing data
ORA-02373: Error parsing insert statement for table "ANDY"."T_TEST5".
ORA-00904: SYS_NC00003$: invalid identifier
```

处理方法：

为表临时添加该列，导完数据后再DROP掉。

```
ALTER TABLE "ANDY"."T_TEST5" ADD "SYS_NC00003$" RAW(126);
ALTER TABLE "ANDY"."T_TEST5" DROP COLUMN "SYS_NC00003$";
```

参考文档：

Oracle DDL优化涉及以下隐含参数：

11g:

_ADD_COL_OPTIM_ENABLED，默认true

alter table t_test9 add val1 NUMBER (1) DEFAULT 0 NOT NULL;

12c:

_ADD_NULLABLE_COLUMN_WITH_DEFAULT_OPTIM，默认为true

alter table t_test9 add val2 NUMBER (1) DEFAULT 0;

Metalink: Doc ID:(2277937.1) Hidden Column "SYS_NC000XX$" Being Added When A New Column Is Added To The Table.
https://asktom.oracle.com/pls/apex/f?p=100:11:::NO::P11_QUESTION_ID:9538764600346809504
https://stackoverflow.com/questions/45756882/why-does-oracle-add-a-hidden-column-here
https://www.jianshu.com/p/5558d8329c93

5) 导出带中文表名

Linux/Unix 环境下，当表名带中文字符时，可能会出现表名显示为乱码的情况。

说明：以下测试使用 SecureCRT 工具，并且数据库字符集为 ZHS16GBK。

1. 确认 Secure CRT 设置的 Character Encoding 为 Default 或 GBK 兼容字符集。
2. 导出格式为 EXPDP 时，必须设置 LANG 或 LC_ALL 为中文，才能显示正确的中文文件名。

处理方法如下：

GDUL> unload table andy.T_测试表
dump/目录下，.dmp 和.sql 文件显示为乱码。

$ cd <GDUL>/dump
$ ls -rlt
$ locale -a
$ export LANG=zh_CN.GBK
或
$ export LC_ALL=zh_CN.GBK
$ cd <GDUL>/dump
$ ls -rlt

3. 导出格式为 DB 时，必须设置 NLS_LANG 为损坏库的字符集。

$ export NLS_LANG=.zhs16gbk
$ ./gdul.sh
GDUL> unload table andy.T_测试表

直接导出到新库

当数据字典正常时，建议导出数据到新建数据库，这在大多数情况下是最省时省力的方式。

1 生成配置文件

1) $ *./osetup*

说明：数据库需要为 mount 或 open 状态。

2) 设置导出格式为 DB(conf/gdul.ini)

参数	设置值
export_format	DB
export_db.db_conn	目标库的连接字符串， 格式：username/password@ip:port/service_name
export_db.buffer_size	16777216 提示：如果导入时报ORA-03146: invalid buffer length for TTC field错误，需要减小buffer值为524288。

2 启动 gdul.sh，初始化字典

$./gdul.sh

GDUL> bootstrap

```
Bootstrap finish.
```

说明：gdul.sh 包含 LD_LIBRARY_PATH 等 ORACLE 所需要环境变量。

*如果服务器端 ORACLE 软件已损坏或者不可用，可以配置 instant client。

Unix/Linux 环境：

1. 从 oracle 官网下载 instant client 文件，以 instantclient-basic-linux-12.2.0.1.0.zip 为例。
2. 解压文件，并配置符号链接。
   cd /u02/
   unzip instantclient-basic-linux-12.2.0.1.0.zip
   cd /u02/ instantclient_12_2
   ln -s libclntsh.so.12.1 libclntsh.so
3. 更改 gdul.sh
   设置 ORACLE_HOME
   ORACLE_HOME=/u02/ instantclient_12_2
4. 启动 gdul.sh

Windows 环境：

1. 从 oracle 官网下载 instant client 文件，以 instantclient-basic-windows.x64-12.2.0.1.0.zip 为例。
2. 解压文件。
3. 把 Instant Client 路径添加置系统 PATH 环境变量。
   系统=>高级系统设置->高级->环境变量->系统变量中找到 Path，添加 d:\instantclient_12_2
   或者打开 CMD 命令行窗口，启动 GDUL 前，临时设置 PATH 变量。
   CMD> set PATH=d:\instantclient_12_2;%PATH%

   确认 oci.dll 能够找到，并且是正确的 64 位版本。
   CMD> where oci.dll
   CMD> gdul

3 新库创建表结构

说明：新库需要创建同名用户和表结构，如果没有表结构 SQL，需要用 GDUL 导出基本表结构定义。

1) GDUL 导出表及其它对象 DDL 语句。
GDUL>list user
GDUL>unload object <用户名>
上述命令在 ddl 目录下创建若干 SQL 文件，包含表和其它对象 DDL 定义。
@ANDY_CLUSTER_DDL.sql
@ANDY_TABLE_FULL_DDL.sql
@ANDY_TABLE_CONS_DDL.sql
@ANDY_TABLE_INDEX_DDL.sql
@ANDY_TABLE_TRIG_DDL.sql
@ANDY_VIEW_DDL.sql
@ANDY_SEQ_DDL.sql
@ANDY_PROC_DDL.sql
@ANDY_FUNC_DDL.sql
@ANDY_PACKAGE_DDL.sql
@ANDY_PACKAGE_BODY_DDL.sql
@ANDY_TABLE_DDL.sql

2) 目标库创建数据库用户和表结构
SQL>drop user ANDY cascade;
SQL>create user ANDY identified by ANDY;
SQL>grant dba to ANDY;
SQL>connect ANDY/ANDY

GDUL 正式版支持完整表 DDL 语句：
@ANDY_CLUSTER_DDL.sql
@ANDY_TABLE_FULL_DDL.sql

GDUL 试用版支持基本的表 DDL 语句：
@ANDY_TABLE_DDL.sql

4 导出用户或表到新库

GDUL> unload table andy.t_compart [parallel 5]

GDUL>unload user andy [parallel 5]

建议：

1) 导入前只建立表结构，不包含索引、约束、触发器等对象，导入完成后再建立这些对象。
2) 如果数据库位于阵列，可以采用并行导出，如 parallel 5。

导入前重要说明：

1) 如果目标库表中已有数据，需要先 truncate 掉数据，再导入。
2) 如果目标库表上已建立触发器，外键，作业等，需要先禁用掉。

--查询外键约束

```
select 'alter table '||owner||'.'||table_name||' disable constraint
'||constraint_name||';'
  from dba_constraints t
 where constraint_type = 'R'
   and status = 'ENABLED'
   and owner = 'ANDY';
```

--查询触发器

```
select 'alter trigger '||t.owner||'."'||t.trigger_name||'" disable;'
  from dba_triggers t
 where table_owner = 'ANDY'
   and status = 'ENABLED';
```

导入后重要说明：

1) 导入完成后，请查看 log/unload_table.log 日志，确认正常导入至数据库，无 ORA-错误，并且无 skipped 表。
2) 由于高速导入采用的直接路径+NOLOGGING+禁用索引的方式，导入完成后，如果表上建有索引，需要 rebuild 索引。

后续工作：

数据导入完成后，创建约束、索引、触发器 DDL 等对象。
@ANDY_TABLE_CONS_DDL.sql
@ANDY_TABLE_INDEX_DDL.sql
@ANDY_TABLE_TRIG_DDL.sql

注意：CONS_DDL.sql 中脚本创建约束时，加入了 novalidate 选项，以便创建约束时不校验已有数据，立即创建完成。
也可以手工去除 novalidate 选项，但此时创建约束时，会校验所有数据，导致全表扫描，花费时间较长。

如果目标库表上已建立触发器，外键，作业等，需要重建索引，并启用触发器，外键对象。

--重建索引

```
select 'alter index '||t.owner||'."'||t.index_name||'" rebuild;'
  from dba_indexes t
 where table_owner = 'ANDY'
   and status<>'VALID';
```

```
select 'alter index ' || index_owner || '.' ||index_name ||' rebuild partition ' || PARTITION_NAME
|| ';'
from dba_ind_partitions
where index_owner='ANDY';
```

```
select 'alter index ' || index_owner || '.' ||index_name ||' rebuild subpartition ' ||
subpartition_name || ';'
from dba_ind_subpartitions
where index_owner='ANDY';
```

--启用触发器

```
select 'alter trigger '||t.owner||'."'||t.trigger_name||'" enable;'
   from dba_triggers t
 where table_owner = 'ANDY'
    and status = 'DISABLED';
```

清空用户下所有表脚本：

```
set serverout on size unlimited
declare
   table_owner varchar2(128);
begin
  table_owner := 'ANDY';
  for tab in (select table_name from dba_tables where owner = table_owner)
  loop
    begin
      execute immediate 'truncate table '||table_owner||'."'||tab.table_name ||
'"';
    exception
      when others then
        dbms_output.put_line('Failed to truncate table
'||table_owner||'.'||tab.table_name);
    end;
```

```
  end loop;
end;
/
```

查询非空表：

```
set serverout on size unlimited
declare
   table_owner varchar2(128);
   cnt number;
begin
  table_owner := 'ANDY';
  for tab in (select table_name from dba_tables where owner = table_owner)
  loop
    begin
      execute immediate 'select count(*) from
'||table_owner||'."'||tab.table_name || '"' into cnt;
      if (cnt > 0) then
       dbms_output.put_line(table_owner||'.'||tab.table_name||' is not empty');
      end if;
    exception
      when others then
        dbms_output.put_line('Failed to count(*) table
'||table_owner||'.'||tab.table_name);
    end;
  end loop;
end;
/
```

5 导入时常见错误

1. ORA-26014: table object XXXX 上出现意外错误(在检查 table object number 时)

或 ORA-39826: Direct path load of view or synonym (OWNER.TABLE_NAME) could not be resolved
原因：目标数据库中未建立表结构。
处理方法：目标库建立表结构。

2. Begin unload table error: ORA-00600: internal error code, arguments: [kpodpals:parsefailure]
或 ORA-39774: parse of metadata stream failed with the following error: LPX-00210: expected '<' instead of '?'
原因：GDUL 程序运行时的 NLS_LANG 设置和目标库字符集不兼容。
处理方法：更改 gdul.sh 脚本，设置 NLS_LANG 为目标库字符集，如 NLS_LANG=.UTF8 或 NLS_LANG=.ZHS16GBK。

3. ASM 下可能会出现如下错误
GDUL> unload table andy.t_test2

2021-06-03 09:36:16 unloading table "ANDY"."T_TEST2" in parallel 1...
unload table error: ORA-39776: fatal Direct Path API error loading table "ANDY"."T_TEST2"
ORA-01565: error in identifying file '+DATADG/testdb/datafile/users.259.960661583'
ORA-17503: ksfdopn:2 Failed to open file +DATADG/testdb/datafile/users.259.960661583
ORA-15055: unable to connect to ASM instance
ORA-01017: invalid username/password; logon denied
Finish unload table "ANDY"."T_TEST2" error: ORA-39780: Direct path context operations are not allowed after the context is aborted or finished
2021-06-03 09:36:19 unloaded 0 rows.

原因：ORACLE 操作系统用户下，缺失 asmdba 组。

处理方法：

$id -a

确认 asmdba, asmadmin 组都位于 ORACLE 操作系统用户下，然后重新登录 telnet。

6 常用查询脚本

--查询 GDUL 导入会话状态

```
col username for a30
col program for a30
col event for a30
set linesize 200
select sid, username, program, status, event, seconds_in_wait
  from gv$session
where program like 'gdul%';
```

--查询当前表大小

```
select segment_name, partition_name, blocks, bytes/1024/1024 as mb
  from dba_segments
 where owner = 'ANDY'
   and segment_name = 'T_PART';
```

--查询表内 LOB 段大小

```
select segment_name, bytes, blocks
  from dba_segments
 where segment_name in(select segment_name from dba_lobs where table_name
= 'T_XML_CLOB');
```

--大表进度查询

1. 新建 gdul 窗口，获取旧库中字典里的表块数

./gdul.sh

GDUL>set user 用户名

GDUL>list table 表名

2. 新库查询表的当前块数

```
select blocks
  from dba_segments
 where owner='用户名'
   and segment_name = '表名';
```

3. 计算进度

```
select <2获取的块数> * 100 / <1获取的块数>
  from dual;
```

分区表说明：
如果是分区表的话，每个分区的导出都会记录在 log/unload_table.log 文件中。

7 限制

导出到新库采用直接路径加载方式，直接路径加载限制如下：

1. 不支持导出到 cluster 表。
2. 不支持 XMLTYPE，BFILE 类型，请导出到 EXPDP 格式，再 impdp 导入。
3. CLOB 表的并行导出，需要新库>=11G，10G 不支持 LOB 并行。

SYSTEM 文件损坏处理

当无法正常 bootstrap 时，说明 SYSTEM 文件发生损坏，或者数据字典异常。根据严重程度，涉及以下几种场景：

1) SYSTEM 数据文件头损坏，但内容正常。

2)数据文件部分损坏，大部分字典正常。

3)SYSTEM 表空间丢失，测试库有表结构。

4)SYSTEM 表空间丢失，无测试库。

1. 文件头损坏时，手工填写 ts#, rfile#.

文件头一旦损坏，需要手工填写 datafile.ini 中 file#, ts#, rfile#, blocks, block_size, name 信息。

如果可以到 mount 状态，可以通过查询 v$datafile 获取，否则需要 scan filelist 命令协助填写。

scan filelist 命令，扫描 SYS.FILE$和数据文件，并通过 rfile#对应关系生成 scan_datafile.ini 文件，并据此更改 conf/datafile.ini 内容。注意，rfile#重复时，将会生成不正确的结果。可通过尝试导出 WRH$_DATAFILE 内容，来查找 file#, ts#, rfile#内容。

1. 设置 gdul.ini

设置详细的 ORACLE 数据库版本，可以通过 sqlplus -v 获取。

2. 设置 datafile.ini

如果可以到 mount 状态，可以通过查询 v$datafile 获取。如果无法 mount，需要先设置 system, sysaux 数据文件的必要信息，再扫描数据文件。

datafile.ini 设置

Name	file#	ts#	rfile#	blocks	block_size
system01.dbf	1	0	1	0	8192
其它数据文件	1	1	1	0	8192

说明：非 system 表空间文件 ts#设置为非 0 值，比如 1。

完整示例：

```
<datafile>
 <row> <file#>1</file#> <ts#>0</ts#> <rfile#>1</rfile#> <blocks>0</blocks>
<block_size>8192</block_size> <name>/oradata/system01.dbf</name>
<raw_offset>0</raw_offset> </row>
```

```
 <row> <file#>1</file#> <ts#>1</ts#> <rfile#>1</rfile#> <blocks>0</blocks>
<block_size>8192</block_size> <name>/oradata/sysaux01.dbf</name>
<raw_offset>0</raw_offset> </row>
</datafile>
```

3. 扫描数据文件，获取 ts#, rfile#.

GDUL> info

显示文件列表.

GDUL> scan filelist

扫描数据文件内容，获取 ts#, rfile#.

```
stage 1: scan datafiles...
   rfile#      blocks name
-------- -------- --------------------------------------------
          1     152321 /oradata/system01.dbf
          2      81921 /oradata/sysaux01.dbf
scan datafiles completed, 3 files.
stage 2: scan filelist info...
scan tablespace #0...
scan FILE$ with data_obj# 17(DB).
    file#         ts#     rfile#     blocks
-------- -------- -------- --------
          1            0          1      64000
          2            1          2      51200
          3            2          3       3200
          4            4          4        640
... ...
scan filelist completed, 11 rows.
stage 3: report datafile list...
    file#         ts#     rfile#     blocks name
-------- -------- -------- -------- --------------------------------------------
          1            0          1     152321 /oradata/system01.dbf
          2            1          2      81921 /oradata/sysaux01.dbf
Please check "log/scan_datafile.ini", and modify "conf/datafile.ini" when necessary.
```

说明：检查生成的 log/scan_datafile.ini 文件，确认无误后更改 conf/datafile.ini 文件。

2. SYSTEM 文件头损坏场景

SYSTEM 数据文件头损坏时，将无法直接得到 rootdba，进而无法获取字典信息，此时需要手工查找 rootdba 位置。

1) 扫描 system 表空间，查找 rootdba

GDUL> scan tablespace 0

```
start scan tablespace 0 in parallel 1...
scan tablespace completed.
```

GDUL> scan bootstrap

```
scan BOOTSTRAP$ with data_obj# 59(DB).
ROOT DBA: 1, 520, get 60 rows.
TABLE/CLUSTER          DATA_OBJ#    TAB#
C_OBJ#                 2            0
COL$                   2            5
TAB$                   2            1
OBJ$                   18           0
C_USER#                10           0
USER$                  10           1
```

2) bootstrap 时，指定 rootdba 的文件#和块#

语法：GDUL> bootstrap file <rfile#> block <block#>

GDUL> bootstrap file 1 block 520

```
Starting bootstrap at rfile# 1, block# 520
Bootstrap finish.
```

3. SYSTEM 文件内容部分损坏场景

SYSTEM 数据文件部分损坏时，无法 bootstrap，只能尽量尝试逐个恢复字典表数据。

1) 扫描 SYSTEM 表空间

GDUL> scan database 0

```
start scan tablespace 0 in parallel 1...
scan tablespace completed.
```

2) 扫描字典表

GDUL> bootstrap scan

说明：bootstrap scan 会尝试扫描所有必要的字典表，生成数据字典文件。

如果 bootstrap scan 失败，可能需要手工扫描单个字典表。

3) 扫描单个字典表

1. 核心字典表

序号	命令	语法
0	扫描 bootstrap	**语法：scan bootstrap [object <data_obj#>]** **说明：**显示核心字典表对应的 data_object#和 tab#， 供 scan col, scan tab, scan obj 等参考。 **示例：**GDUL> scan bootstrap
1	扫描 COL$表	**语法：scan col [object <data_obj#> tab <tab#>]** **说明：**导出 COL$表中的数据，存入 dict/col.dat 文件。 **示例：**GDUL> scan col
2	扫描 TAB$表	**语法：scan tab [object <data_obj#> tab <tab#>]** **说明：**导出 TAB$表中的数据，存入 dict/tab.dat 文件。 **示例：**GDUL> scan tab
3	扫描 OBJ$表	**语法：scan obj [object <data_obj#>]** **说明：**导出 OBJ$表中的数据，存入 dict/obj.dat 文件。 **示例：**GDUL> scan obj
4	扫描 USER$表	**语法：scan user [object <data_obj#> tab <tab#>]** **说明：**导出 USER$表中的数据，存入 dict/user.dat 文件。

		示例：GDUL> scan user
5	扫描 PROP$表	**语法：scan props [object <data_obj#>]** **说明：**导出 PROP$表中的数据，存入 dict/props.dat 文件。 **注意：**如果无法导出该表，可以从正常库 bootstrap 后拷贝 prop.dat 即可，此时需要注意 prop.dat 中的字符集。 **示例：**GDUL> scan props
6	扫描 KOPM$表	**语法：scan kopm [object <data_obj#>]** **说明：**导出 KOPM$表中的数据，存入 dict/kopm.dat 文件。 **注意：**如果无法导出该表，可以从正常库 bootstrap 后拷贝 kopm.dat 即可。 **示例：**GDUL> scan kopm

上述扫描命令成功执行完后，即可以查看及导出普通表。

2. 扫描其它字典表

扫描表分区数据

序号	命令	语法
1	扫描 TABPART$表	**语法：scan tabpart [object <data_obj#>]** **说明：**导出 TABPART$表中的数据，存入 dict/tabpart.dat 文件。 **示例：**GDUL> scan tabpart
2	扫描 TABCOMPART$表	**语法：scan tabcompart [object <data_obj#>]** **说明：**导出 TABCOMPART$表中的数据，存入 dict/tabcompart.dat 文件。 **示例：**GDUL> scan tabcompart
3	扫描 TABSUBPART$表	**语法：scan tabsubpart [object <data_obj#>]** **说明：**导出 TABSUBPART$.表中的数据，存入 dict/tabsubpart.dat 文件。 **示例：**GDUL> scan tabsubpart

扫描 LOB 数据

序号	命令	语法

1	扫描 LOB$表	**语法：scan lob [object <data_obj#> tab <tab#>]** **说明：**导出 LOB$表中的数据，存入 dict/lob.dat 文件。 **示例：**GDUL> scan lob
2	扫描 LOBCOMPPART$表	**语法：scan lobcompart [object <data_obj#>]** **说明：**导出 LOBCOMPPART$表中的数据，存入 dict/lobcompart.dat 文件。 **示例：**GDUL> scan lobcompart
3	扫描 LOBFRAG$表	**语法：scan lobfrag [object <data_obj#>]** **说明：**导出 LOBFRAG$表中的数据，存入 dict/lobfrag.dat 文件。 **示例：**GDUL> scan lobfrag

扫描索引数据

序号	命令	语法
1	扫描 IND$表	**语法：scan ind [object <data_obj#> tab <tab#>]** **说明：**导出 IND$表中的数据，存入 dict/ind.dat 文件。 **示例：**GDUL> scan ind
2	扫描 ICOL$表	**语法：scan icol [object <data_obj#> tab <tab#>]** **说明：**导出 ICOL$表中的数据，存入 dict/indcol.dat 文件。 **示例：**GDUL> scan icol
3	扫描 INDPART$表	**语法：scan indpart [object <data_obj#>]** **说明：**导出 INDPART$表中的数据，存入 dict/indpart.dat 文件。 **示例：**GDUL> scan indpart
4	扫描 INDCOMPART$表	**语法：scan indcompart [object <data_obj#>]** **说明：**导出 INDCOMPART$表中的数据，存入 dict/indcompart.dat 文件。 **示例：**GDUL> scan indcompart
5	扫描 INDSUBPART$表	**语法：scan indsubpart [object <data_obj#>]** **说明：**导出 INDSUBPART$表中的数据，存入 dict/indsubpart.dat 文件。 **示例：**GDUL> scan indsubpart

扫描分区对象数据

序号	命令	语法
1	扫描 PARTOBJ$表	**语法：scan partobj [object <data_obj#>]** **说明：**导出 PARTOBJ$表中的数据，存入 dict/partobj.dat 文件。 **示例：**GDUL> scan partobj
2	扫描 PARTLOB$表	**语法：scan partlob [object <data_obj#>]** **说明：**导出 PARTLOB$表中的数据，存入 dict/partlob.dat 文件。 **示例：**GDUL> scan partlob
3	扫描 PARTCOL$表	**语法：scan partcol [object <data_obj#>]** **说明：**导出 PARTCOL$表中的数据，存入 dict/partcol.dat 文件。 **示例：**GDUL> scan partcol
4	扫描 SUBPARTCOL$表	**语法：scan subpartcol [object <data_obj#>]** **说明：**导出 SUBPARTCOL$表中的数据，存入 dict/subpartcol.dat 文件。 **示例：**GDUL> scan subpartcol

扫描其它数据

序号	命令	语法
1	扫描 CLU$表	**语法：scan clu [object <data_obj#> tab <tab#>]** **说明：**导出 CLU$表中的数据，存入 dict/clu.dat 文件。 **示例：**GDUL> scan clu
2	扫描 TS$表	**语法：scan ts [object <data_obj#> tab <tab#>]** **说明：**导出 TS$表中的数据，存入 dict/ts.dat 文件。 **示例：**GDUL> scan ts
3	扫描 COLTYPE$表	**语法：scan coltype [object <data_obj#> tab <tab#>]** **说明：**导出 COLTYPE$表中的数据，存入 dict/coltype.dat 文件。

		示例：GDUL> scan coltype
4	扫描 OPQTYPE$表	**语法：scan opqtype [object <data_obj#> tab <tab#>]** **说明：**导出 OPQTYPE$表中的数据，存入 dict/opqtype.dat 文件。 **示例：**GDUL> scan opqtype
5	扫描 X$QN*表	**语法：scan xdbqn [object <data_obj#>]** **说明：**导出 X$QN*表中的数据，存入 dict/xdb_qname_token.dat 文件。 **示例：** GDUL> set user XDB GDUL> list table X$QN GDUL> scan xdbqn object <data_obj#>
6	扫描 X$NM*表	**语法：scan xdbnm [object <data_obj#>]** **说明：**导出 X$NM*表中的数据，存入 dict/xdb_nmspc_token.dat 文件。 **示例：** GDUL> set user XDB GDUL> list table X$NM GDUL> scan xdbnm object <data_obj#>

3. 重新加载数据字典

说明：扫描单个字典表后，需要使用以下方式，重新加载新的数据字典文件。

1. GDUL> reload dict 命令。
2. 或者重新进入 GDUL。

   GDUL> exit

   $ ./gdul.sh

   GDUL> list user

4. 无 SYSTEM 表空间恢复(测试库中有表结构)

当测试库中有表结构时，可以借助测试库中的表结构，指定生产库的 data_object_id，从而导出数据。

1) 使用测试库的 system 导出数据字典。

1. 把测试库 system 文件拷贝过来。

2. $./osetup

3. $./gdul.sh

GDUL> bootstrap

GDUL> info　–确认所有数据文件正常

2) 尝试从 SYS.wrh$_seg_stat_obj 表获取<表, data_obj#>对应关系。

先尝试获取 wrh$_seg_stat_obj 表的数据，理想状态可以从中找到 表名和 data_object_id 的对应关系。

GDUL>scan tablespace 1　--SYSAUX

GDUL>sample segment all

$cd sample

从 sample 目录搜索几个表名字符串， 找到 wrh$_seg_stat_obj 对应的内容，据此可以找到若干 <表名, data_object_id>对应关系。

如果本步操作未找到<表, data_object_id>对应关系，就需要执行第 4 步，手工分析表和 data_obj#的对应关系。

3) 扫描业务表所在表空间

GDUL> scan tablespace ###

说明：该操作会输出到 dict/scan_*.dat 文件中。

默认扫描线程数为 1，如果磁盘性能较好，且表空间下有多个数据文件，可以采用并行方式，以加快扫描速度。

示例：*GDUL> scan tablespace 7 parallel 10*

4) 采样已扫描的业务表空间

GDUL>sample segment all

$cd sample

说明：该操作执行采样，并在 sample 目录下生成 segment 列定义及采样数据。

进入 sample 目录，查看采样数据，然后对照测试库，查找是哪张表的数据。

5) unload 采样出的数据段

GDUL>unload table 用户名.表名 object_id <分析出的 data object id>

5. 无 SYSTEM 表空间恢复(无任何备份)

无任何表结构信息的话，只能通过采样，猜测表结构，并导出扫描出的数据块内容。

1) 设置字符集等参数(gdul.ini)

参数	值
db_charset	ZHS16GBK
db_ncharset	AL16UTF16
db_timezone	+08:00

说明：可以查询其它相似库来获取参数值：

```
select name, value$
  from sys.props$
 where name in ('NLS_CHARACTERSET',
'NLS_NCHAR_CHARACTERSET',
'DBTIMEZONE');
```

2) 扫描表所在表空间

GDUL> scan tablespace ###

说明：该操作会输出到 dict/scan_*.dat 文件中。

默认扫描线程数为 1，如果磁盘性能较好，且表空间下有多个数据文件，可以采用并行方式，以加快扫描速度。

示例：*GDUL> scan tablespace 7 parallel 10*

3) 采样已扫描的表空间

GDUL>sample segment all

$cd sample

说明：该操作执行采样，并在 sample 目录下生成 segment 列定义及采样数据。

注意：

采样的列类型可能不准确，如果有测试库可以获取表结构，可以替换掉 sample/seg_<data_object_id>.dict 中<column_def></column_def>内的采样列定义，再执行 unload

从其它库获取表定义语句：

```
SQL>select '<row>'
       || '<COL#>'               || col#              || '</COL#>'
       || '<SEGCOL#>'            || segcol#           || '</SEGCOL#>'
       || '<INTCOL#>'            || intcol#           || '</INTCOL#>'
       || '<TYPE#>'              || type#             || '</TYPE#>'
       || '<NAME>'               || name              || '</NAME>'
       || '<LENGTH>'             || length            || '</LENGTH>'
       || '<SEGCOLLENGTH>' || segcollength || '</SEGCOLLENGTH>'
       || '<PRECISION#>'      || precision#      || '</PRECISION#>'
       || '<SCALE>'              || scale             || '</SCALE>'
       || '<CHARSETID>'       || charsetid       || '</CHARSETID>'
       || '<CHARSETFORM>'    || charsetform    || '</CHARSETFORM>'
       || '</row>'
  from sys.col$
 where obj# = (select object_id from dba_objects where owner = '<OWNER>' and object_name =
'<TABLE_NAME>')
 order by col#;
```

4) unload 采样出的数据段

GDUL>unload segment all | <data_object_id>

说明：用指定的 data_object_id unload 表数据，指定 all 会恢复所有 segment

6. 勒索程序处理

1) PL/SQL 勒索病毒删除 TAB$表数据恢复

tab$表数据被删除后，数据库重启时无法启动，alert 文件中会出现类似如下错误信息：

ORA-00704: bootstrap process failure

ORA-00704: bootstrap process failure

ORA-00600: internal error code, arguments: [16703], [1403], [20], [], [], [], [], [], [], [], [], []

如果执行 GDUL 的 bootstrap 命令，gdul.log 中会显示以下错误内容：

20200925 07:41:52 Loading dictionary from DB...

20200925 07:41:52 ..ROOT dba: 0, 1, 520.

20200925 07:41:52 ..COBJ# OBJ#: 2, dba: 1, 144.

20200925 07:41:52 ..COL$ [21, 5], TAB$ [4, 1], OBJ$ [18]

20200925 07:41:54 ..COL$ table, rows 93866.

20200925 07:41:54 ..TAB$ table, rows 0.

20200925 07:41:54 ..OBJ$ table, rows 0.

20200925 07:41:54 ..USER$ table error, can't find table.

处理方法:

使用 undelete dicttab 命令，恢复被删除的 tab$行，尝试打开数据库，然后 expdp/exp 导出数据。

如果数据库打开（或强制打开）失败，建议使用 GDUL 导出到新数据库，参见《导出到新 DB(SYSTEM 表空间正常)》章节。

1. 还原 TAB$表被删除行

GDUL>bootstrap

GDUL>list user --由于 tab$无内容，bootstrap 失败

查看 log/gdul.log 文件，tab$表导出 0 行。

更改 conf/gdul.ini 中的 db_file.read_only 参数，设置为 false。

GDUL>unload dictcol --导出列定义

GDUL>undelete dicttab –直接更改 tab$表的数据块，去除行删除标记

GDUL>exit

GDUL>bootstrap --确认 GDUL 可以正常读取。

GDUL>list user　　--用户列表显示正常

GDUL>exit

$dbv file=SYSTEM01.DBF　--dbv 命令确认 undelete 后的块校验正确

说明：undelete dicttab 命令直接更新 system 表空间中的块，移除了 TAB$行的删除标记位。

注意：undelete dict*命令不支持 ASM。

ASM 下处理步骤：

1. 手工拷贝 ASM 中 system 数据文件至文件系统，并做好备份。

```
$rman target /
RMAN> backup as copy datafile '+DATA/TESTDB/DATAFILE/system.257.982567557' format
'/home/oracle/system01.dbf';
RMAN> list copy;
```

或者:

```
RMAN> copy datafile '+DATA/TESTDB/DATAFILE/system.257.982567557' to
'/home/oracle/system01.dbf';
```

2. 把 conf/datafile.ini 中的 SYSTEM 数据文件指向本地 system01.dbf 数据文件。

3. GDUL 下执行 undelete dicttab，恢复被删除的 tab$数据。

4. 把恢复后的 system 数据文件拷贝至 ASM 下。

```
$rman target /
RMAN> backup as copy datafilecopy '/home/oracle/system01.dbf' format '+DATA';
```

或:

```
ASM> cp /home/oracle/system01.dbf +DATA/TESTDB/DATAFILE/
ASM> cd +DATA/TESTDB/DATAFILE/
ASM> ls
```

2. 打开数据库，删除恶意触发器

步骤：

1. 数据库打开前，需要禁用所有系统级触发器和作业。

SQL>select status from v$instance; 确认实例处于 MOUNT 状态。

SQL>ALTER SYSTEM SET "_system_trig_enabled"=FALSE SCOPE=memory;

SQL>ALTER SYSTEM SET job_queue_processes=0 SCOPE=memory;

SQL>alter database open upgrade;

2. 数据库打开过程中，可能会出现由于 tab$数据不完整，导致打开失败。

案例：

SQL> alter database open upgrade;

ORA-01092: ORACLE Instance terminated, Disconnection forced.

ORA-00913: 值过多

处理过程：

1) 打开 sql trace:

startup mount

select * from v$diag_info;

alter session set sql_trace=true;

alter database open upgrade;

2) 从.trc 中找到报错的 SQL:

PARSE ERROR #644971544:len=70 dep=1 uid=0 oct=2 lid=0 tim=1669154855940 err=913

insert into SYS_FBA_TRACKEDTABLES values (-1, -1, 0, '', '', 1, NULL)

Flashback Archive: Error ORA-913 in SQL insert into SYS_FBA_TRACKEDTABLES values (-1, -1, 0, '', '', 1, NULL)

ORA-00913: 值过多

ORA-00913: 值过多

上述 SQL 涉及到 SYS_FBA_TRACKEDTABLES 表，而该表未能从 tab$中恢复。

3) 解决办法：

直接更改 oracle.exe 中，上述 SQL 文本执行所依赖的 SQL，

原 SQL 文本: "select count(OBJ#) from SYS_FBA_TRACKEDTABLES where bitand(FLAGS,%d)!=0";

临时新 SQL 文本: "select 1 from dual".

使用新 oracle.exe 启动实例，Windows 环境需要临时把注册表中的 ORA_<实例名>_AUTOSTART 键改为 FALSE。

CDB/PDB 库:

alter session set container=ORACLERCU;

alter pluggable database open upgrade;

3. 数据库打开后，删除恶意代码相关的触发器及存储过程。

1) 还原 tab$数据

select object_name from dba_objects where object_name like '%ORACHK%';

select count(*) from ORACHK458480EB64A41C6D9650681

where obj# not in(select obj# from sys.tab$);

insert into sys.tab$

```
select * from ORACHK458480EB64A41C6D9650681
  where obj# not in(select obj# from sys.tab$);
commit;
```

2) 删除触发器和过程

以下 SQL 查询恶意触发器列表。

```
SQL>select 'DROP TRIGGER '||owner||'."'||TRIGGER_NAME||'";'
  from dba_triggers
 where TRIGGER_NAME like   'DBMS_%_INTERNAL% ';
select 'DROP TRIGGER '||owner||'."'||TRIGGER_NAME||'";'
  from dba_triggers
 where TRIGGER_NAME like   'DBMS_%_MONITOR%';
select 'DROP TRIGGER '||owner||'."'||TRIGGER_NAME||'";'
  from dba_triggers
 where TRIGGER_NAME like   'DBMS_%_SUPPORT% ';

 select 'DROP PROCEDURE '||owner||'."'||object_name||'";'
  from dba_procedures
 where object_name like 'DBMS_%_INTERNAL% '
     or object_name like 'DBMS_%_DBMONITOR%';
select 'DROP PROCEDURE '||owner||'."'||object_name||'";'
  from dba_procedures
 where object_name like 'DBMS_STANDARD_FUN9%';
```

3) 删除正在执行的恶意 JOB

说明：以**业务用户**连接 sqlplus，删除恶意 JOB，该作业会执行 truncate table 操作。

以下查询 JOB 时间范围:

```
select min(last_date), min(next_date), max(last_date), max(next_date)
from dba_jobs
where schema_user='SXLSUSER' and what like 'DBMS_STANDARD_FUN9(%';
```

以下删除恶意作业:

```
begin
  for rec in(select job from dba_jobs where schema_user='ANDY' and what
like 'DBMS_STANDARD_FUN9(%')
  loop
    execute immediate 'call dbms_job.remove(' || rec.job || ')';
```

```
  end loop;
end;
/
commit;
```

以下观察进度:

```
select executions from v$sqlarea where sql_id='4n17wsu9m9nfq';
```

如果 JOB 量太大(比如>100 万)的话，dbms_job.remove 运行缓慢，建议直接删除:

SQL> select bytes/1024/1024/1024 from dba_segments where segment_name='JOB$';

BYTES/1024/1024/1024

24.8925781

--删除作业

```
delete from sys.job$
 where powner ='SXLSUSER'
   and what like 'DBMS_STANDARD_FUN9(%';
commit;
```

--move 表

```
alter table sys.job$ move;
alter index sys.I_JOB_JOB rebuild;
alter index sys.I_JOB_NEXT rebuild;
```

--查看进度:

```
col opname for a20
select opname, sofar, totalwork, time_remaining
  from v$session_longops
 where time_remaindding> 0;
```

```
OPNAME                SOFAR  TOTALWORK TIME_REMAINING
--------------------- ---------- ---------- --------------
Table Scan             16914    3262718         17079
```

--查看 delete 事务状态

```
select xid, used_ublk, used_urec, log_io, phy_io, phy_io, cr_get,
cr_change
 from v$transaction;
```

4) 查看业务用户是否被 truncate

```
col object_type a15
col object_name for a30
set pagesize 200
select object_type, object_name, last_ddl_time
  from dba_objects
 where object_type in('TABLE', 'TABLE PARTITION')
   and owner = 'ANDY'
 order by 3;
```

说明：

1. 有一类勒索代码存在于含勒索代码的 ORACLE 安装程序中，建议从 ORACLE 官网下载正版 ORACLE 软件。

文件名：?/rdbms/admin/prvtsupp.plb

触发器名为 DBMS_SUPPORT_DBMONITOR，调用的存储过程名为 DBMS_SUPPORT_DBMONITORP。

2. 另外一类勒索代码存在于含勒索代码的绿色版 PLSQL Developer 程序中，建议使用安装版 PLSQL Developer 软件。

文件名: AfterConnect.sql, Login.sql

触发器名分别为"DBMS_SUPPORT_INTERNAL ", "DBMS_SYSTEM_INTERNAL ", "DBMS_CORE_INTERNAL "，名称末尾为空格。

3. 如果由于 ORA-600 错误无法正常打开，可能需要设置以下参数：

_allow_resetlogs_corruption=TRUE

_corrupted_rollback_segments=(_SYSSMU1_1880814008$, _SYSSMU2_237578013$,...)

再打开

SQL>recover database until cancel;

SQL>alter database open resetlogs;

3. Exp/expdp 导出数据

正常打开数据库后，就可以 expdp/exp 导出用户。

SQL> create directory EXPDP_DIR as '<expdp_path>';

$ expdp '/ as sysdba' directory=EXPDP_DIR dumpfile=test.dmp logfile=test.log schemas=<owner_name>

注意：

如果字典表中有坏块，expdp 或 exp 可能会导出失败，需要跳过坏块。

1. exp 导出需要 10231 事件，以便跳过坏块：

SQL>alter system set events='10231 trace name context forever,level 10';

export NLS_LANG=.ZHS16GBK

exp system/manager owner=<ower_name> file=d:\oracle_db\test.dmp log=test.log buffer=1000000 rows=n compress=n STATISTICS=none

导入：

imp system/manager full=y file=d:\oracle_db\test.dmp log=imp_test.log buffer=1000000

2. expdp 可以尝试新建库 B，创建连接到坏库 A 的 dblink(如 link_a)，在 B 库执行 expdp，导出 A 库数据。

B 库执行 expdp：

expdp '/ as sysdba' directory=EXPDP_DIR dumpfile=test.dmp logfile=test.log schemas=<owner_name> network_link=link_a

除 10231 跳过坏块事件外，也可以尝试 dbms_repair 包，跳过单张表中的坏块。

```
SQL> exec sys.dbms_repair.skip_corrupt_blocks('ANDY', 'T_TEST');
SQL> select skip_corrupt from dba_tables where owner = 'ANDY' and
table_name = 'T_TEST';
```

~~备用方案~~

~~1. 查找 ORACHK*表的段头块#~~

~~1. 设置 gdul.ini。~~

~~更改导出类型为 SQLLDR，供步骤 4 使用。~~

~~2. 扫描 SYSTEM 表空间。~~

~~*GDUL> scan tablespace 0*~~

~~说明：该操作会输出到 dict/scan_*.dat 文件中。~~

~~3. 采样 system 表空间中的段，查找 obj$表的 obj#。~~

~~*GDUL>sample segment all*~~

~~*$cd sample*~~

~~说明：一般 obj$的 data_object_id 为 18，如不确定，可以使用本步骤 sample 所有段，查找其 obj#。~~

~~4. 导出 obj$表，找到 ORACHK*表的 obj#。~~

~~*GDUL>unload segment <obj$表的 data_object_id>*~~

~~说明：导出该 segment 后，查看 dump/GDUL_SEG_00000018.dat 文件，搜索 ORACHK 字符串，据此找到该 ORACHK*表的 obj#。~~

~~5. 根据上一步查到的 obj#，查找 dict/scan_segment.dat 文件，据此找到段头 file#, block#。~~

~~2. 导出数据字典~~

~~GDUL>unload dictcol~~

~~GDUL>unload dicttab2 1 151904 --手工指定 ORACHK*表的段头地址~~

~~GDUL>unload dictobj~~

~~3. 导出表数据~~

~~GDUL>list user~~

~~GDUL>set user XXX~~

~~GDUL>unload user XXX~~

2) PL/SQL 勒索病毒删除 BOOTSTRAP$表数据恢复

bootstrap$表数据被删除后，数据库重启时无法启动，trace 文件中会出现以下错误信息：

*** 2019-08-05 14:49:45.668

ORA-00704: bootstrap process failure

ORA-00702: bootstrap verison '' inconsistent with version '8.0.0.0.0'

如果执行 GDUL 的 bootstrap 命令，gdul.log 中会显示以下错误内容：

20190806 11:49:09 Loading dictionary from DB...

20190806 11:49:09 Failed to read BOOTSTRAP$ table.

处理方法：使用 undelete bootstrap 命令，恢复被删除的 bootstrap$行，然后打开数据库。

1. 还原 BOOTSTRAP$表被删除行

GDUL>undelete bootstrap –直接更改 boostrap$表的数据块，去除行删除标记

GDUL>exit

GDUL>bootstrap --确认 GDUL 可以正常读取。

GDUL>exit

$dbv file=SYSTEM01.DBF --dbv 命令确认 undelete 后的块校验正确

注意：undelete dict*命令不支持 ASM。

2. 打开数据库

SQL>ALTER SYSTEM SET "_system_trig_enabled"=FALSE SCOPE=memory;

SQL>alter database open;

数据库打开后，查询 dba_triggers，确认是否有恶意触发器。

3) 文件勒索病毒恢复

说明：某些文件级勒索病毒会加密数据文件头部若干块，导致启动失败。

处理方法如下：

首先设置 db_compat_version 参数为正确的 Oracle 版本。

1. 扫描数据文件的 file#, ts#, rfile#

datafile.ini 中添加所有数据文件，SYSTEM 表空间对应数据文件的 ts#设置为 0，其它表空间数据文件 ts#临时设置为 1。

GDUL> scan filelist

扫描出数据文件对应的 file#, ts#, rfile#后，更改 conf/datafile.ini 内容。

注意：

2. 启动 GDUL，扫描数据字典

尝试 1: 使用常规方式，扫描数据字典。

GDUL> bootstrap

GDUL> list user

尝试 2：如果常规方式失败，尝试 scan 方式扫描。

GDUL> scan tablespace 0

GDUL> bootstrap scan

GDUL> list user

尝试 3: 如果 scan 选项失败，尝试单独扫描字典表。

GDUL> scan bootstrap

GDUL> scan col

GDUL> scan tab

GDUL> scan obj

GDUL> scan user

...

其它字典表 scan 命令，请参考《SYSTEM 文件内容部分损坏场景》章节。

所有字典表手工扫描完毕后，重新加载扫描生成的数据字典。

GDUL> reload dict

GDUL> list user

3. 导出表数据

GDUL> list user

GDUL> unload user XXX

请查看导出日志，如果坏块较多，尝试 scan 选项导出数据。

GDUL> scan database parallel 2

GDUL> unload user XXX scan

ASM 磁盘损坏处理

ASM 磁盘头出现损坏时，GDUL 将无法正常获取数据文件内容，需要手工配置磁盘文件 asmdisk.ini 中的磁盘头信息，并做特殊处理。

建议只把损坏磁盘组对应的所有磁盘加入到磁盘配置文件中。

1. 磁盘组损坏，元数据正常

当 ASM 磁盘头损坏（比如前 1M 字节损坏），但元数据未损坏时，可以直接扫描磁盘组，获取磁盘头信息，然后更正 asmdisk.ini 文件项，再启动 GDUL。

1) 配置 asmdisk.ini 文件

可通过查询 v$asm_disk 视图获取磁盘列表及对应的磁盘组。

如果 ASM 实例无法启动，或者无法从 v$asm_disk 中获取 ASM 磁盘列表，可以从 asm alert 日志中获取磁盘组名称和磁盘路径。

通过 grid 用户，使用 sqlplus / as sysasm 连接 ASM 实例，执行以下 SQL：

```
set verify    off
set heading   off
set feedback  off
set echo      off
set term      off
set trimout   on
set trimspool on
set pagesize  0
set define    off
set linesize  2000
set timing    off
set time      off
select /*+RULE*/
       ' <row> '       ||
       '<group_number>'|| g.group_number ||'</group_number> '||
       '<group_name>'  ||g.name         ||'</group_name> '     ||
       '<block_size>'  ||g.block_size   ||'</block_size> '     ||
       '<au_size>'     ||g.allocation_unit_size ||'</au_size> '||
       '<disk_number>' ||d.disk_number  ||'</disk_number> '    ||
       '<path>'        ||d.path         ||'</path> '      ||
       '<f1b1>'        ||0              ||'</f1b1> '           ||
       '<raw_offset>'  ||0              ||'</raw_offset> '     ||
       '</row>'
  from v$asm_disk d
 inner join v$asm_diskgroup g
    on d.group_number = g.group_number
```

```
  order by g.group_number, d.disk_number;
```

或

```
select /*+RULE*/
       ' <row> '       ||
       '<group_number>'||d.group_number ||'</group_number> '  ||
       '<group_name>'  ||'DATA'         ||'</group_name> '    ||
       '<block_size>'  ||'4096'         ||'</block_size> '    ||
       '<au_size>'     ||'1048576'      ||'</au_size> '       ||
       '<disk_number>' ||d.disk_number  ||'</disk_number> '   ||
       '<path>'        ||d.path         ||'</path> '      ||
       '<f1b1>'        ||0              ||'</f1b1> '          ||
       '<raw_offset>'  ||0              ||'</raw_offset> '    ||
       '</row>'
  from v$asm_disk d
  order by d.group_number, d.disk_number;
```

asmdisk.ini 文件格式为：

```
<asmdisk>
  <row>
  <group_number>1</group_number>
  <group_name>DATA</group_name>
  <block_size>4096</block_size>
  <au_size>1048576</au_size>
  <disk_number>0</disk_number>
  <path>/dev/asm-diskc</path>
  <f1b1>0</f1b1>
  <raw_offset>0</raw_offset>
  </row>
</asmdisk>
```

说明：

按上述格式配置损坏的 ASM 磁盘组中对应的所有磁盘文件，

group_number 为磁盘组编号，以 1 开始。

group_name 为磁盘组名称。

f1b1 为文件目录块位置，如果磁盘具有 f1b1 值，默认为 2。

disk_number 为磁盘编号，以 0 开始，相同磁盘组内不能重复。损坏磁盘的 disk_number 必须和 ASM 磁盘头中一致，否则无法生成正确的数据文件。

disk_path 为 ASM 磁盘全路径。

2) 扫描 ASM 磁盘组

GDUL> asmcmd

ASMCMD> lsdg -l

```
Group#       Name                    Type         Sector      Block       AU          Total_MB    Compat      DB Compat
---------- -------------------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
0            TEST                    EXTERNAL   512         4096        1048576     0           10.0.0.0    10.0.0.0
```

ASMCMD>lsdsk -l

```
DiskNum    Name                 Path                                        DiskGroup          AuSize       BlockSize
RedunType    F1B1          Status        DiskSize_MB
-------- --------------- ---------------------------------------- --------------- ---------- ---------- ---------- ---------- ---------- ------------
0                              E:\db\asm_old\asm_diskb                     TEST               1048576      4096
1              2              INVALID       0
```

说明：

对于损坏磁盘，上述输出内容为 asmdisk.ini 中为手工配置的磁盘信息，如果不准确，需要后续步骤获取正确信息后再调整。

对于正常磁盘，上述显示内容为正确数据，可以作为参考，更正损坏磁盘的 asmdisk.ini 内容。

ASMCMD> scan diskgroup TEST

格式：scan diskgroup <dg_name> [parallel <#>]

```
scan diskgroup "TEST" in parallel 1...
f1b1 loc
   disk#          f1b1#
   0              2
scan diskgroup "TEST" completed.
```

说明：扫描磁盘组中 ASM 元数据信息，打印 1#文件的 f1b1 位置，并生成 ASM 文件映射。

上述扫描磁盘组 TEST 时，发现 disk#0 包含 f1b1 位置信息，为 2。

3) 更正 asmdisk.ini 文件

退出 GDUL，更正 asmdisk.ini 文件中损坏磁盘所有对应的项的内容。

从上述 lsdsk -l 和 scan asmdg 命令已经获取到了 f1b1 值和其它正常磁盘的内容。

设置 asmdisk.ini 中的 disk#0 的 f1b1 值为 2。

对损坏磁盘的 disk#，磁盘组名，AU_SIZE，BlockSize 内容，可以参考正常磁盘的 lsdsk -l 输出。

重新进入 GDUL 内置的 asmcmd 命令：

GDUL>asmcmd

```
ASMCMD> ls
ASM/
TEST/
```

ASMCMD> cd test

ASMCMD> ls -l

```
Type          Redund      Striped     StripSize   StripWidth  Sys  Size(GB)  BlockSize  Name
                                                                 Y             CONTROLFILE/
                                                                 Y             ONLINELOG/
                                                                 Y             DATAFILE/
                                                                 Y             TEMPFILE/
                                                                 Y             PARAMETERFILE/
                                                                 N             spfiletest.ora => TEST.265
```

ASMCMD> pwd

```
+TEST/TEST/DATAFILE/
```

ASMCMD> ls -l

```
Type          Redund      Striped     StripSize   StripWidth  Sys  Size(GB)  BlockSize  Name
DATAFILE      UNPROT      COARSE      1048576     1           Y    0.684     8192       SYSTEM.260.912401703
DATAFILE      UNPROT      COARSE      1048576     1           Y    0.586     8192       SYSAUX.261.912401709
DATAFILE      UNPROT      COARSE      1048576     1           Y    0.435     8192       UNDOTBS1.262.912401713
DATAFILE      UNPROT      COARSE      1048576     1           Y    0.005     8192       USERS.264.912401727
```

4) 更改 datafile.ini 文件，指向 ASM 里的数据文件

从上述输出可以看到各数据文件名及块大小为 8192，添加到 datafile.ini 中。file#, ts#, rfile#, blocks 在数据文件未损坏时，可以设置为 0。

```
<datafile>
<row> <file#>1</file#> <ts#>0</ts#> <rfile#>1</rfile#> <blocks>0</blocks> <block_size>8192</block_size>
<name>+TEST/TEST/DATAFILE/SYSTEM.260.912401703</name> <raw_offset>0</raw_offset> </row>
</datafile>
```

5) 运行 gdul，导出数据或拷贝出数据文件

GDUL> bootstrap

GDUL> list user

或

GDUL> asmcmd

ASMCMD> cd +TEST/TEST/

ASMCMD> cpdir datafile d:\datafile

2. 磁盘组损坏，元数据部分损坏

如果按《磁盘头损坏，元数据正常》处理后，发现磁盘组部分元数据损坏，需要手工导出数据库文件。

ASMCMD> scan diskgroup TEST

ASMCMD>extract dbfile

说明：上述命令会在<GDUL>/datafile/生成提取的数据库文件，包括数据文件，日志文件，控制文件。

3. 磁盘组损坏，元数据彻底损坏

当 ASM 磁盘组中的元数据彻底损坏时，可以直接扫描磁盘组中的所有磁盘，导出数据文件。

使用限制：不支持多数据库情况，不支持 bigfile 表空间，同时只支持一个块大小（默认 8k)的数据文件导出。

格式：scan asmdisk <dg_name> [parallel <#>]

说明：扫描 ASM 磁盘，导出数据文件至 datafile 目录。

示例：

1. 配置 gdul.ini 中默认的数据块大小

```
<db_block_size>8192</db_block_size>
```

注意：如果有非默认的 8K 表空间，需要设置多次，并执行多次 scan asmdisk。

2. 配置 asmdisk.ini 文件

略

3. 扫描 ASM 磁盘，导出数据文件

GDUL> scan asmdisk DATA

start scan all asm disk in parallel 1...

find datafile rfile# 1, block# 1, DB "TEST", TBS "SYSTEM", blocks: 89600

find datafile rfile# 2, block# 1, DB "TEST", TBS "SYSAUX", blocks: 76800

find datafile rfile# 3, block# 1, DB "TEST", TBS "UNDOTBS1", blocks: 56960

find datafile rfile# 4, block# 1, DB "TEST", TBS "USERS", blocks: 640

scan asmdisk completed.

US7ASCII 字符集中文处理

由于历史原因，有一些遗留库为 US7ASCII 字符集，但存储的为 GBK 编码字符。客户端使用时 NLS_LANG 设置为.US7ASCII，中文可以正常显示和存储。

当迁移至 ZHS16GBK 或 UTF8 字符集数据库时，中文字符导入时会乱码。此时需要特殊处理，才能正常导入中文。

测试环境：

源端字符集：US7ASCII

目标端字符集：AL32UTF8

EXP/IMP 导出导入(适用于非 CLOB 表)

1. 源端 exp 导出用户或表

--导出, direct=y 可能会触发 bug.

set NLS_LANG=.us7ascii

--按用户导出

exp cstest/cstest owner=cstest file=D:\temp\cstest.dmp log=exp_cstest.log

--按表名导出

exp cstest/cstest tables=(t_varchar, t_clob) file=D:\temp\cstest.dmp log=exp_cstest.log

2. GDUL 更改 exp 文件字符集为 GBK 编码

GDUL> expcmd

EXPCMD> open D:\temp\cstest.dmp

EXPCMD> set csid 852

EXPCMD> open D:\temp\cstest.dmp

说明：

852 为 ZHS16GBK 字符集 ID。

```
SELECT TO_CHAR(NLS_CHARSET_ID('ZHS16GBK'), '9999') ZHS16GBK_ID
  FROM DUAL;
```

3. 目标端导入用户或表

set NLS_LANG=.zhs16gbk

--按用户导入

imp andy/andy fromuser= cstest touser= cstest ignore=y buffer=20480000 commit=y file=D:\temp\ cstest.dmp log=imp_ cstest.log

--按表名导入

imp cstest/cstest ignore=y fromuser=cstest touser=cstest tables=(t_varchar, t_clob) buffer=20480000 commit=y file=D:\temp\cstest.dmp

GDUL/SQLLDR 导出导入（适用于 CLOB 表）

1. 源端 GDUL 导出 CLOB 表

设置 GDUL 导出格式为 SQLLDR。

GDUL>bootstrap

GDUL>unload table CSTEST.T_CLOB

2. SQLLDR 导入

set NLS_LANG=.zhs16gbk

cd <GDUL_DIR>\dump

sqlldr userid=cstest/cstest control=CSTEST_T_CLOB.ctl

实用程序

1 内置 asmcmd

说明：内置 asmcmd 可以在 ASM 实例未打开，但文件头和目录块完整时，查看 ASM 文件信息，并且可以 copy 到本地文件系统。

GDUL> asmcmd

```
Enter internal asmcmd.
```

1) ls、cd

ASMCMD> ls

```
NORMAL_DATA/
DATA/
```

ASMCMD> cd data

ASMCMD> ls

```
test-cluster/
TEST/
```

ASMCMD> cd test

ASMCMD> ls -l

```
Type          Redund       Striped      Sys             Name
---------- ---------- ---------- ---------- ----------
                                        Y               DATAFILE/
                                        Y               CONTROLFILE/
                                        Y               ONLINELOG/
                                        Y               TEMPFILE/
                                        Y               PARAMETERFILE/
                                        N               spfiletest1.ora => DATA.265
```

2) copy

说明：copy 命令拷贝单个 ASM 文件至本地文件系统。

ASMCMD> cp spfiletest1.ora d:\test.ora

```
copy SPFILETEST1.ORA to d:\test.ora sucessfully, bytes: 2560
```

3) cpdir

说明：cpdir 命令可以拷贝目录下的所有文件到本地文件系统。

ASMCMD>cd +DATA/TEST/

需要先 cd 至拷贝目录的上一层目录。

ASMCMD> ls

```
CONTROLFILE/
ONLINELOG/
DATAFILE/
TEMPFILE/
PARAMETERFILE/
spfiletest.ora
```

ASMCMD> cpdir datafile d:\datafile

```
copy +DATA/TEST/DATAFILE/SYSTEM.260.912401703 to
d:\datafile/SYSTEM.260.912401703 successfully, 700.01 MB
copy +DATA/TEST/DATAFILE/SYSAUX.261.912401709 to d:\datafile/SYSAUX.261.912401709
successfully, 600.01 MB
copy +DATA/TEST/DATAFILE/UNDOTBS1.262.912401713 to
d:\datafile/UNDOTBS1.262.912401713 successfully, 445.01 MB
copy +DATA/TEST/DATAFILE/USERS.264.912401727 to d:\datafile/USERS.264.912401727
successfully, 5.01 MB
```

4) scan diskgroup

格式：scan diskgroup <dg_name> [parallel <#>]

说明：scan diskgroup 命令扫描 ASM 磁盘组中的元数据信息。

扫描出的元文件信息位于 dict/scan_asmdg*.dat 文件中。

扫描出的 ASM 文件信息位于 log/scan_asmdg_file.log 中。

ASMCMD> scan diskgroup *TEST*

```
scan diskgroup "TEST" in parallel 1...
f1b1 loc
   disk#         f1b1#
```

```
  0              2
scan diskgroup "TEST" completed.
```

5) extract asmfile#

格式：extract <dbfile|<asmfile#>>

说明：由 scan diskgroup 命令扫描出的元数据，生成数据库文件，文件位于 datafile 目录。

extract dbfile 抽取所有数据库文件，包括数据文件，日志文件，控制文件。

ASMCMD> extract 256

```
extract asmfile "256"...
   local_file: datafile/current.256.1025636943, size: 18497536
extract asmfile "256" completed.
```

6) scan asmdisk

格式：scan asmdisk <dg_name> [parallel <#>]

说明：scan asmdisk 命令扫描 ASM 磁盘组中的所有数据文件块，并生成对应的数据文件，文件位于 datafile 目录。该命令仅在磁盘组元数据彻底损坏时使用。

ASMCMD> scan asmdisk TEST parallel 20

```
scan asmdisks of diskgroup "DATADG" in parallel 20...
 find datafile rfile#      1, block# 1, DB "DB1", TBS "SYSTEM", blocks: 230400
 find datafile rfile#      2, block# 1, DB "DB1TBS "SYSAUX", blocks: 556800
```

2 EXPDP 损坏导出工具

EXPDP dump 文件损坏时，GDUL 内置的 dpcmd 工具可以扫描其中的有效数据，直接导入到数据库中。

目前已支持 Basic, Low, Medium, High 等压缩格式。

1) 设置导出格式为 DB

配置 conf/gdul.ini 参数文件，设置导出格式为 DB。

参数	设置值
export_format	DB
export_db.db_conn	目标库的连接字符串， 格式：username/password@ip:port/service_name
export_db.buffer_size	16777216 提示：如果导入时报ORA-03146: invalid buffer length for

	TTC field错误，需要减小buffer值为524288。

2) 扫描 dump 文件

GDUL>dpcmd

DPCMD> scan E:\db\dump\T_VARCHAR.dmp

```
Scanned 2 segments.

Simple table DDL created in file log/scan_dp_file_ddl.log.
```

说明：扫描损坏的 datapump 文件，生成表数据映射。

DPCMD> list table

```
DP File: E:\db\dump\T_VARCHAR.dmp

TABLE_NAME                                POSITION    META_LENGTH      DATA_LENGTH
----------------------------------------- ----------- ---------------  ---------------
ANDY.T_VARCHAR                            8192        2238              173
ANDY.SYS_EXPORT_TABLE_01                  16384       40363             40419
```

格式：list table <table_substr>

说明：列出扫描出的表数据映射。

3) 创建表结构

1) 导入 master 表

由 scan_dp_file_ddl.log 获取 SYS_EXPORT_TABLE_01 或 SYS_EXPORT_SCHEMA_01 表定义，然后数据库中创建该表结构。

DPCMD> list table EXPORT

SQL>create table ANDY. SYS_EXPORT_TABLE_01...;

DPCMD> unload table ANDY.SYS_EXPORT_TABLE_01

2) 由 master 表获取表结构

3) 创建完整的业务表结构

4) 查看损坏 dmp 中导出的数据行数

4) 导出 DP 中的表数据

--导出单张表到数据库

DPCMD> unload table t_varchar

--导出所有表到数据库

DPCMD> unload table all

格式：unload table ALL|<OWNER.TABLE_NAME>

说明：导出表数据至数据库。

--导出单个用户下的所有表到数据库

DPCMD>unload user <user_name>

清空所有表脚本：

```
declare
begin
  for i in (select table_name from user_tables)
  loop
    execute immediate 'truncate table "' || i.table_name ||'"';
  end loop;
end;
```

```
begin
  for i in (select table_name from dba_tables where owner = 'ANDY')
  loop
    execute immediate 'truncate table ANDY."' || i.table_name ||'"';
  end loop;
end;
```

3 EXP 损坏导出工具

Exp dump 文件损坏时，GDUL 内置的 expcmd 工具可以扫描其中的有效数据，直接导入到数据库中。

1) 设置导出格式为 DB

配置 conf/gdul.ini 参数文件，设置导出格式为 DB。

参数	设置值

export_format	DB
export_db.db_conn	目标库的连接字符串，用户名必须为表所在用户。 格式：username/password@ip:port/service_name
export_db.buffer_size	16777216 提示：如果导入时报ORA-03146: invalid buffer length for TTC field错误，需要减小buffer值为524288。

2) 扫描 dump 文件

GDUL>expcmd

EXPCMD> scan C:\Users\Andy\exp_t_varchar.dmp
Scan exp file C:\Users\Andy\exp_t_varchar.dmp, FILE KEY [179370974]...
tab_pos, 2335
Table DDL created in file log/scan_exp_file_ddl.log.

EXPCMD > list table

```
Exp File: C:\Users\Andy\exp_t_varchar.dmp
TABLE_NAME                                DDL_POS    INSERT_POS DATA_POS
----------------------------------------- ---------- ---------- ----------
T_VARCHAR                                 2335       2579       2634
```

注意：扫描表结构及数据时，无法获取对应的用户信息，请自行判断所属用户。

3) 创建表结构

scan 命令从 dmp 文件中获取建表语句，并把基本的建表语句写入 log/scan_exp_file_ddl.log 文件中，可据此创建表结构。

4) 导出 exp 文件中的表数据

说明：创建完表结构后，即可导入表到数据库当前用户下。

--查看所有表

EXPCMD>list table [表名子串]

--导出单张表到数据库

EXPCMD> unload table t_varchar

--导出所有表到数据库

EXPCMD> unload table all

示例：

EXPCMD> unload table t_varchar

```
unload table T_VARCHAR...
table T_VARCHAR unloaded 6 rows.
```

限制:

1. 有限支持 LONG 列

导入前必须设置正确的 NLS_LANG，如 NLS_LANG=.zhs16gbk，并且数据库表中的 LONG 列类型需要改为 CLOB。

2. 不支持 ROWID/UDT/XMLTYPE/VARRAY 等特殊列类型。

清空所有表脚本：

```
declare
begin
  for i in (select table_name from user_tables)
  loop
    execute immediate 'truncate table "' || i.table_name ||'"';
  end loop;
end;


begin
  for i in (select table_name from dba_tables where owner = 'ANDY')
  loop
    execute immediate 'truncate table ANDY."' || i.table_name ||'"';
  end loop;
end;
```

4 磁盘扫描工具(beta)

当文件系统损坏时，可以直接扫描磁盘分区或逻辑卷，导出数据文件。

使用限制：不支持多数据库情况，不支持 bigfile 表空间，同时只支持一个块大小（默认 8k)的数据文件导出。

格式：scan disk <disk partition path>

说明：扫描磁盘分区或逻辑卷，导出数据文件至 datafile 目录。

示例：

1. 配置 gdul.ini 中默认的数据块大小

```
<db_block_size>8192</db_block_size>
```

注意：如果有非默认的 8K 表空间，需要设置多次，并执行多次 scan disk。

2. 扫描磁盘分区

```
GDUL> scan disk /dev/sdb1
   find datafile rfile#       3, block# 1, DB "D3210205", TBS "SYSAUX", blocks: 16640
   find datafile rfile#       4, block# 1, DB "D3210205", TBS "USERS", blocks: 640
   find datafile rfile#       1, block# 1, DB "D3210205", TBS "SYSTEM", blocks: 38400
   find datafile rfile#       2, block# 1, DB "D3210205", TBS "UNDOTBS1", blocks: 25600
scan disk completed.
```

3. 配置 GDUL 的 datafile.ini，指向<GDUL>/datafile 目录下的数据文件，并 bootstrap 测试。

5 其它命令

1) file

格式：file <absfile#>

说明：显示数据文件头信息

示例：

```
GDUL> file 1
NAME:                  D:\ORADATA\TEST32\SYSTEM01.DBF
FILE#:                 1
TS#:                   0
RFILE#:                1
DB_NAME:               TEST32
TABLESPACE_NAME:       SYSTEM
BLOCKS:                99840
BLOCK_SIZE:            8192
BYTES:                 780.00 MB
SERVER_VERSION:        00.00.00.00
CREATION_CHANGE#:      7.0
CREATION_TIME:         2013-10-11 09:45:31
CHECKPOINT_CHANGE#: 10146469.0
```

CHECKPOINT_TIME: 2019-08-30 16:08:54

2) rowid

格式：rowid <rowid>

说明：解析 rowid

示例：

GDUL> rowid AAAXocAAEAAAs5LAAH

data object id:96796

  datafile id:4

    block id:183883

      row no:7

3) rdba

格式：rdba <0xnnnnnnnn>

说明：解析 16 进制 rdba

示例：

GDUL> rdba 0x12345678

datafile id:72

  block id:3430008

4) copyscn

格式：copyscn from <abs_file#1> to <abs_file#2>

说明：更改数据文件 abs_file#2 的检查点 SCN 和检查点时间，设置为 abs_file#1 的值。

示例：

GDUL> info

FILE#	TS#	RFILE#	BIGFILE	CHK SCN#	CHK TIME	SIZE(GB)	NAME
1	0	1	No	10147836	1017678412	0.76	D:\ORADATA\TEST\SYSTEM01.DBF
2	1	2	No	10147836	1017678412	1.75	D:\ORADATA\TEST\SYSAUX01.DBF

```
8           12          8           No          10146466              1017677108 0.00
D:\ORADATA\TEST\TEST_SCN01_old.DBF
```

GDUL> copyscn from 1 to 8

```
File#: 8
File Name: D:\ORADATA\TEST\TEST_SCN01_old.DBF
Old Check SCN#:      10146466, Old Check SCN Time: 1017677108
New Check SCN#:      10147836, New Check SCN Time: 1017678412
Are you going to set new SCN#/Time(N/Y) y
copyscn finished, new scn: 10147836, new scn time: 1017678412
```

GDUL> info

```
FILE#       TS#         RFILE#      BIGFILE    CHK SCN#              CHK TIME     SIZE(GB)     NAME
-------- -------- -------- -------- ---------- ---------- ------------------------------
1           0           1           No          10147836              1017678412 0.76
D:\ORADATA\TEST\SYSTEM01.DBF
2           1           2           No          10147836              1017678412 1.75
D:\ORADATA\TEST32\SYSAUX01.DBF
8           12          8           No          10147836              1017678412 0.00
D:\ORADATA\TEST\TEST_SCN01_old.DBF
GDUL>
```

后续可能的命令：

SQL>recover datafile 8;

SQL>alter database open;

5) scan filelist

当数据库无法 mount，且数据文件头损坏时，无法填写 datafile.ini 中的数据文件头信息，此时需要 scan filelist 命令来协助生成 ts#, rfile#等必要内容。

具体使用方法，参见《SYSTEM 文件损坏处理》章节。

6) scan undo

数据库强制打开时，往往需要 offline 活动的 undo 段，此时需要 scan undo 命令来查找需要 UNDO 段列表。

7) oradump

用法：oradump <file#> <block#>

说明：dump 段头或数据块内容。

6 ORACLE 导入命令

1) impdp

```
impdp andy/andy directory=GDUL_DIR tables=t_secure_clob
dumpfile=ANDY_t_secure_clob.dmp remap_schema=andy:andy2 table_exists_action=truncate
```

指定查询条件导入：

```
impdp andy/andy directory=GDUL_DIR3 full=y dumpfile=用户名_表名.dmp remap_schema=用户名:
用户名2 table_exists_action=truncate query=用户名.表名:\"where 列名 in ('1', '2')\"
```

其它命令：

只导入旧expdp中的元数据：

```
impdp andy/andy DIRECTORY=GDUL_DIR DUMPFILE=expdp_andy.dmp SCHEMAS=ANDY
CONTENT=METADATA_ONLY exclude=STATISTICS LOGFILE=imp.log
```

只导入索引：

```
impdp andy/andy DIRECTORY=DUMP_DIR DUMPFILE=expdp_andy.dmp SCHEMAS=ANDY
CONTENT=METADATA_ONLY INCLUDE=INDEX IGNORE=y LOGFILE=imp.log
```

只导入触发器：

```
impdp andy/andy DIRECTORY=GDUL_DIR DUMPFILE=expdp_andy.dmp SCHEMAS=ANDY
CONTENT=METADATA_ONLY INCLUDE=TRIGGER LOGFILE=imp.log
```

只导入视图：

```
impdp andy/andy DIRECTORY=GDUL_DIR DUMPFILE=expdp_andy.dmp SCHEMAS=ANDY
CONTENT=METADATA_ONLY INCLUDE=VIEW LOGFILE=imp.log
```

只生成旧expdp中的元数据，保存至imp.sql文件：

```
impdp andy/andy DIRECTORY=GDUL_DIR DUMPFILE=expdp_andy.dmp SCHEMAS=ANDY
CONTENT=METADATA_ONLY exclude=STATISTICS LOGFILE=imp.log SQLFILE=imp.sql
```

提示:

查看impdp导入进度：

```
col opname for a20
```

```
col target for a20
col start_time for a20
 SELECT sl.sid,
       sl.serial#,
       sl.opname,
       sl.target,
       to_char(sl.start_time, 'YYYY/MM/DD HH24:MI:SS') start_time,
       sl.sofar,
       sl.message,
       sl.totalwork,
       round(sl.SOFAR * 100 / sl.TOTALWORK, 0) || '%' as "%DONE",
       sl.TIME_REMAINING,
       dp.state,
       dp.job_mode
  FROM v$session_longops sl, v$datapump_job dp
 WHERE sl.opname = dp.job_name
   AND sl.sofar < sl.totalwork;
```

注意：如果 impdp 导入时遇到 ORACLE bug，可能需要重建 datapump 组件。

请参考 ORACLE 文档 How To Reload Datapump Utility EXPDP/IMPDP (文档 ID 430221.1)。

重建 datapump 组件方法：

10G:

<

@?/rdbms/admin/catdph.sql

@?/rdbms/admin/catmetx.sql

@?/rdbms/admin/prvtdtde.plb

@?/rdbms/admin/catdpb.sql

@?/rdbms/admin/dbmspump.sql

@?/rdbms/admin/utlrp.sql

>

11G:

@?/rdbms/admin/catproc.sql

@?/rdbms/admin/utlrp.sql

12C：

sql> @?/rdbms/admin/dpload.sql

sql> @?/rdbms/admin/utlrp.sql

如果出现 Datapump Import Fails With ORA-39006, ORA-39213: "Metadata processing is not available" (Doc ID 801337.1)

可能需要重新运行以下过程:

SQL> execute dbms_metadata_util.load_stylesheets;

2) imp

1. 设置NLS_LANG正确的字符集

```
SQL>
select property_value
from database_properties
where property_name='NLS_CHARACTERSET';
$export NLS_LANG=.ZHS16GBK
```

2.把buffer设置成100m, commit=y，再imp

```
imp andy/andy TABLES=t_secure_clob file=<GDUL_DIR>\dump\ANDY_t_secure_clob.dmp
FROMUSER=andy TOUSER=andy2 ignore=y buffer=104857600 commit=y
```

3) sqlldr

```
cd <GDUL_DIR>\dump
sqlldr userid=andy/andy control=ANDY_t_secure_clob.ctl
```