

Oracle DBA 手记 4

数据安全 警示录



电子工业出版社

Publishing House of Electronics Industry

北京•BEIJING

未雨绸缪，防患未然

《Oracle DBA 手记 4》序言

在数据库领域十几年，我发现在国内技术人员往往一直在充当救火员的角色，企业常常也认为只有能够力挽狂澜、起死回生的技术人员，才是好的技术人员。而实际上，能够不犯错误、少犯错误，提前预防、规避灾难的技术人员才是企业技术环境的最有力保障，能够未雨绸缪，防患于未然才是更好的技术实践。

我们每年都帮助很多企业挽救数据、拯救危机，触发这本书写作的契机正是来自 2011 年 12 月 30 日和 31 日，连续的两个整天，从凌晨再到凌晨，连续挽救了几个用户的数据库灾难，这些灾难发生的那么简单，那么不可思议，在迈入 2012 这个神秘年份的一刻，深深的触动了我，我想，如果将这些案例描述出来，就可能帮助一些用户警醒借鉴，避免再陷入这样的困境。而从别人的挫折中学习、借鉴，进而在自我的环境中未雨绸缪，防患于未然，是每个数据库管理人员和企业数据环境管理者应该具备的素质。

写作这本书还和 2011 年底众多席卷而来的密码泄露事件有关，当我注视着最常用的几个密码都在互联网上被公开时，除了手忙脚乱的在各大网站修改密码，剩下的就是深深的遗憾。几乎所有从事 IT 行业的人，都深知安全的重要性，可是放在实际执行中，大家又往往习惯性失明，忽视了自己周围本应能够达到的力所能及之安全，很多专业人士就以这样或者那样的侥幸心理放任了风险的存在，并一步一步走向了安全危机。

对于数据库安全来说，通常我们认为缺乏的并非技术手段，更多的是缺乏规范和安全认知，如果用户都能够严格的遵循安全守则并应用现有的安全技术手段，数据库的安全性就能够大幅增强，我们的安全事故率也会大大降低。

所以我决定动笔，写下自己多年来所遭遇到的安全案例以及对于数据安全的思考，如果本书中的内容能够帮助一些企业规避错误，保全数据，挽救一些技术人员的时间，那么我将感到无比欣喜，于我们的生命中，最为宝贵的就是时间，寸金难买寸光阴。

信息安全

在传统的信息安全领域，存在三个基本的安全要素，这三个要素分别是：

保密性(Confidentiality)、完整性(Integrity)和可用性(Availability)，缩写为 CIA。

这三个要素的基本定义如下：

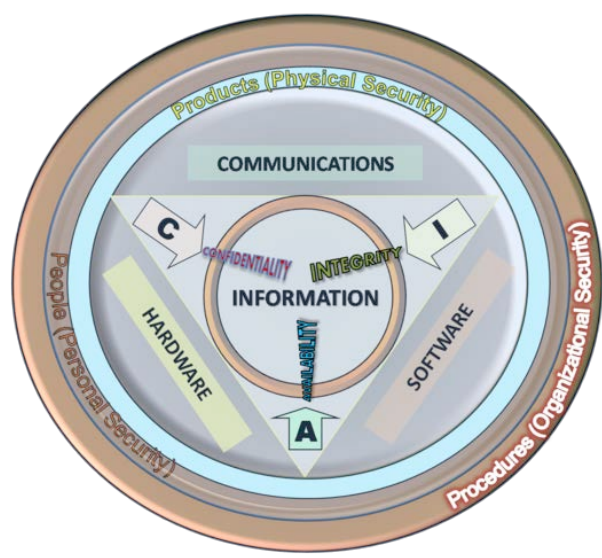
保密性 (Confidentiality): 指信息在存储、使用和传输过程中的保密性, 要确保信息在这些环节中不会泄漏给非授权方;

完整性 (Integrity): 指信息在存储、使用和传输过程中不会被非授权用户篡改、变更, 同时需要防止授权用户对系统及信息进行非授权篡改, 保持信息在整个过程中内外的一致性;

可用性 (Availability): 信息系统因其服务使命, 必须在用户需要时, 可以被正常访问, 授权用户或实体对信息系统的正常使用不应被异常拒绝或中断, 应当允许其可靠、及时地访问和获取信息及资源。高可用系统要求所有时间可用, 要确保系统不因电源故障、硬件故障和系统升级等因素影响服务的可用性。

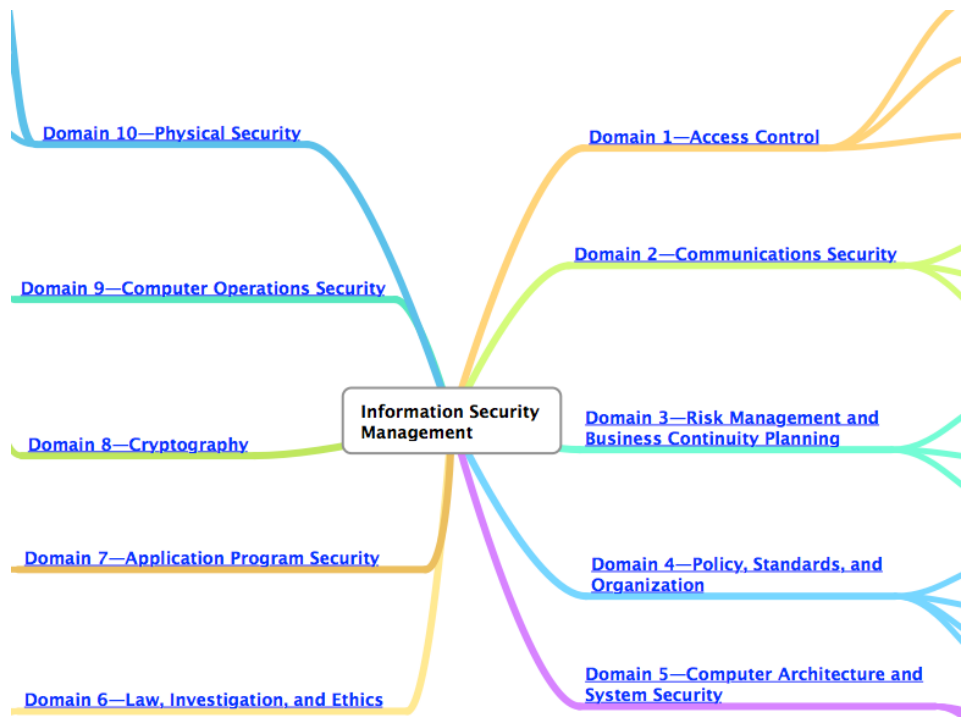
信息安全的三要素是对安全的概括和提炼, 不同机构和组织, 因为需求不同, 对 CIA 的侧重也会有所不同, 随着信息安全的发展, CIA 经过细化和补充, 增加了许多新的内容, 包括可追溯性 (Accountability)、抗抵赖性 (Non-repudiation)、真实性 (Authenticity)、可控性 (Controllable) 等; 与 CIA 三元组相反的一个概念是 DAD 三元组, 即泄漏 (Disclosure)、篡改 (Alteration) 和破坏 (Destruction), 实际上 DAD 就是信息安全面临的最普遍的三类风险, 是信息安全实践活动最终应该解决的问题。

CIA 的核心三要素涉及软件 (Software)、硬件 (Hardware) 和通信 (Communications) 三个方面, 下图清晰的描述了信息安全三要素及相关领域范畴:



信息安全的三要素 (Information security From Wikipedia)

从 CIA 理念出发，通过对信息安全范畴所有相关主题精炼整理得到了一个标准化的知识体系，被称为公共知识体系—CBK（Common Body of Knowledge），CBK 包括 10 个知识范畴（Domain），对安全进行了全面的概括，具有极强的指导意义，下图对 10 领域进行了简单的列举（不同出版物描述略有不同）：



CBK 10 Doamin（参考 Handbook of Information Security Management）

以上 10 个范畴分别为：访问控制，电信、网络和互联网安全，风险管理和商务连续性计划，策略、标准和组织，计算机架构和系统安全，法律、调查和道德，应用程序安全，密码学，计算机操作安全，物理安全。

数据安全

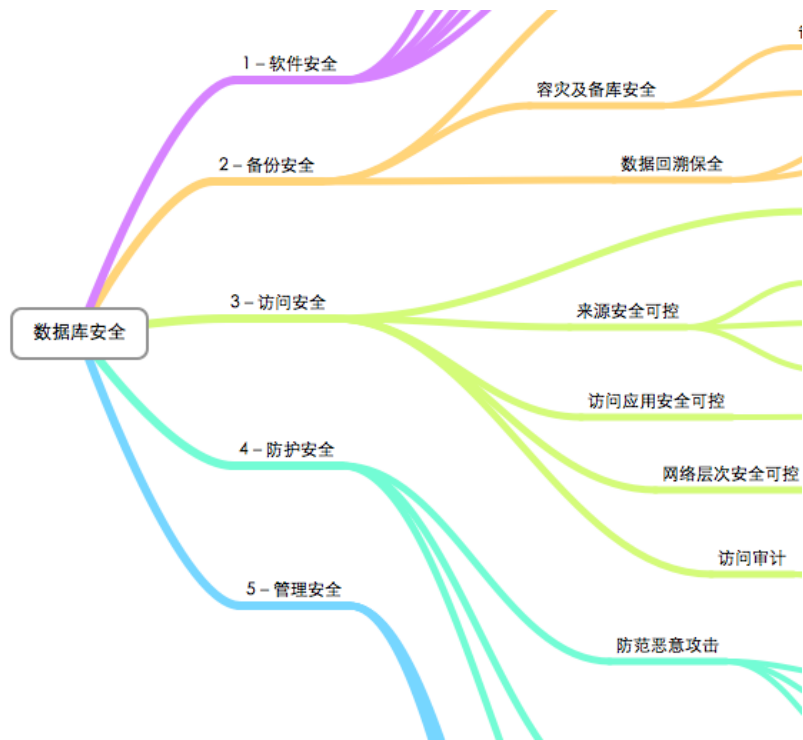
信息安全的核心是数据安全。

在数据安全的范畴内，也包含信息安全的诸多方面，根据多年的服务经验与思考，我们将安全划分为五大

方面，分别是：

软件安全、备份安全、访问安全、防护安全、管理安全

这五大方面是信息安全在数据领域的引申和映射,在企业数据安全中,这五大方面是相辅相成、互有交叉、共同存在的，下图是关于安全的一张思维导图，本书内容的案例涉及到了这五大方面：



在这五大安全方向中，可能出现两种性质的安全问题，第一，由于内部管理不善而导致的数据安全问题；第二，由于外部恶意攻击入侵所带来的安全问题。通常我们把安全问题狭义化为后者，这实际上是片面的，在数据安全问题上，前者造成的数据损失、数据损毁，其发生率和影响度都远远超过后者。

下面我们对数据安全的五大方面做一下简要的分析和探讨：

1.软件安全是指我们选择的数据库产品、版本是否稳定安全；厂商所能提供的补丁集和 BUG 修正是否及时、基础硬件与操作系统是否经过认证。很多用户在部署数据库软件时，仅仅选择了最容易获得的初始发布版本（如 Oracle Database 10.2.0.1 或者 Oracle Database 11.2.0.1 等），遗漏了可能已经存在的补丁修正，并且在运行维护中并不能够及时跟踪软件更新，也无法获得 BUG 信息、补丁修正和安全告警，这就使得软件本身的很多风险隐患得不到修正。如果软件安全无法保证，数据库安全的基础也就丧失了。

2.备份安全是指用户数据能否得到及时有效的备份保全,能否在故障灾难之后获得及时的恢复和挽救。在数据库运行期,最为重要的就是备份安全,如果没有可靠的备份,将数据集中起来就只能是等待数据灾难,所以我们将备份安全提升到核心地位,备份以及随之衍生的容灾安全等,都是企业整体数据架构应该考虑的因素。很多企业在数据灾难之后因为缺乏有效备份而一蹶不振,根据 Gartner 2007 年的一份调查报告显示,在经历了数据完全丢失而导致系统停运的企业中,有 2/5 再也没能恢复运营,余下的企业也有 1/3 在两年内宣告破产,由此可见,由于备份安全问题导致的企业伤害可能远远大于黑客攻击。

3.访问安全是指用户数据库的访问来源和访问方式是否安全可控。通常数据库系统处于 IT 系统的核心,其安全架构涉及主机、系统、存储、网络等诸多方面,如果没有明确的访问控制,缺乏足够的访问分析与管理,那么数据库的安全将是混乱和无法控制的。在应用软件使用和访问数据库时,要正确设置权限,控制可靠的访问来源,保证数据库的访问安全,唯有保证访问安全才能够确保数据不被越权使用、不被误操作所损害,通常最基本的访问安全要实现程序控制、网络隔离、来源约束等。

4.安全防范是指通过主动的安全手段对数据库通讯、传输等进行增强、监控、防护、屏蔽或阻断,诸如数据加密、审计、数据防火墙等技术都在这一范畴之内。我们必须认识到,在 IT 技术高度发展的今天,风险是无处不在、层出不穷的,可能我们从未思考过的安全问题,每天都在不断涌现,所以在数据库环境中采取主动式防护,可以帮助我们监控分析和屏蔽很多未知风险,已经有很多成熟的产品和技术可以用于安全防范。

5.管理安全是指在企业数据的日常管理维护范畴内,能否充分保证数据安全以及服务的高可用连续提供。诸如 DBA 的维护、文件的管理、参数或数据结构的变更等等都可能引入数据风险,管理安全要求我们通过规范、制度以及技术手段去确保维护管理安全;另外,基于硬件、电力等基础平台的故障都可能影响数据库服务的高可用性,在管理中要通过监控手段及时预警,通过集群、备库等切换与服务分担保障服务的连续性。

这就是数据安全的五大方面。

业界安全事故

在 2011 年的新闻报道中,我们注意到很多企业遭受到了严重的安全事故,影响深远,以下摘录了几起广为人知的数据安全事故,让我们一起看一看安全问题都出现在何处:

1. 陕西移动近 1400 万条个人信息遭泄露

根据新闻报道(案件发生大约时间为 2011 年 3 月),犯罪嫌疑人所在的某技术公司承担着为陕西某电信企业提供手机资费计算系统软件平台开发、运行、维护、咨询、防毒等多项工作,可以获取该电信运营商手机号码、姓名、年龄、性别、身份证号、住址、每月通讯费用等资料。

犯罪嫌疑人为了个人利益,窃取用户信息并出售:

“朋友。。。向我要西安、榆林、延安、渭南等六七个地市的移动手机每个月话费消费 20 元以上的信息，内容包括手机号码、月话费消费情况、办卡区域、机主性别、出生年月等，我同意了。第二天我在单位将电脑连接到省移动公司数据库中，提取了 1000 余万条信息，每个地市建立一个文件夹，存储到我的笔记本电脑中。。。 ”

2. 高阳捷迅工程师利用支付宝漏洞盗取 11 万

2009 年间，支付宝公司开通话费支付业务，用户都可以通过购买手机充值卡、充入支付宝账户后进行网上购物，高阳公司负责这一话费充值系统的运行维护。即在支付宝与移动、联通、电信之间搭建平台，负责将支付宝用户购买的手机充值卡转变为支付宝账户的存款。

犯罪嫌疑人是负责这一系统维护的工程师，在 2010 年 1 月至 3 月间，他利用了这个系统的漏洞，多次通过互联网进入高阳公司系统数据库，调取用户充值失败而暂存于此的充值卡信息，然后将其转入自己在支付宝设立的 48 个账户和快钱设立的 31 个账户，共计 111650 元。

3. CSDN 600 用户密码泄露事件

2011 年 12 月 21 日，一组安全事件在国内引发了轰动，黑客在网上公开了 CSDN 网站用户数据库，包括 600 余万个明文的注册邮箱账号和密码可能遭集中曝光。事件发生之后，CSDN 相关网页更一度紧急关闭，以升级为暂时关闭。

随后又暴露了一系列的密码安全事故，多家大型网站的用户信息遭泄露。

4. Putty 中文版后门事件

据 2012 年 2 月 1 日消息，中文版 putty 等 SSH 远程管理工具被曝出存在后门，该后门会自动窃取管理员所输入的 SSH 用户名与口令，并将其发送至指定服务器上。根据分析，此次事件涉及到来自 putty.org.cn、putty.ws、winscp.cc 和 sshsecure.com 站点的中文版 putty、WinSCP、SSHSecure 和 sftp 等软件，而这些软件的英文版本不受影响。

5. 赛门铁克遭黑客“破门” 千万用户信息安全存疑

北京时间 2012 年 02 月 07 日，一个容量达 1.2GB、标题为“赛门铁克的 pcAnywhere 源代码遭泄露”的文件出现在了 BT 网站，并开放提供下载。

赛门铁克确认，pcAnywhere 源代码已被公开发布。这是黑客组织 Anonymous 在过去几周中所声称已获取的 2006 版本产品源代码的一部分。黑客曾经向赛门铁克索取 5 万美元。

赛门铁克在与黑客的电子邮件中表示，“我们将向你支付 5 万美元。但是，我们需要确保你在收到钱后不将源代码发布到互联网上。在起初的三个月中，我们将每月支付 2500 美元。我们将从下周开始向你支付这笔费用。在三个月结束之后，在我们支付余额前，你要让我们相信你已销毁了源代码。我们相信你不会没完没

了地讨价还价。”

在经过了数周有关源代码证据及如何转账的谈判后，双方谈判破裂，交易未能完成。

很快，在微博上出现了这样的“段子”：悲剧 - 安全软件源代码被黑客偷了；喜剧 - 人家只勒索 5 万美元；悲剧 - 赛门铁克要求分期付款，谈崩了；喜剧 - 只好报警；悲剧 - 黑客发布源代码……

我们可以注意到，数据安全问题无处不在，从软件到数据库，从维护人员到黑客，损害安全的因素不是越来越少，而是越来越多。这些事件更警示我们，要不断提升数据安全，防止安全事故的发生。

Oracle 数据库安全

其实 Oracle 数据库自 1977 年肇始之初，就一直将安全置于首位，从强大的数据恢复机制，到不断增强的加密以及安全防范措施。“Oracle”这个名字就是来自于美国中央情报局投资的项目代码，而 CIA 也正是 Oracle 最早期的用户之一。

接触过 Oracle 数据库的人都应当熟悉一个类似如下图所示的错误“ORA-00942：表或视图不存在”，这个简单的错误提示，最初就是在 CIA 的要求之下作为一项安全防范设定的，这个提示的安全意义在于：**避免提供任何具体的实质性提示性信息，以预防黑客的攻击性尝试**。由此可见，安全防范可以从每一个细节入手，安全是一项全面整体的技术实现，并非孤立的存在。

```
SQL> select * from emp;  
select * from emp  
      *  
ERROR at line 1:  
ORA-00942: table or view does not exist
```

受密码事件影响，我们首先在此从 Oracle 数据库的密码机制上来稍微深入了解一下 Oracle 的加密机制。虽然我们知道早在 Oracle 数据库版本 8 的年代，就已经提供了强大丰富的数据库加密功能，但是直至今日，恐怕半数以上的数据库中，仍然存放着用户的明文密码，并且未采用任何数据库安全增强机制。**这也就是我认为最重要的安全问题：我们并不缺乏安全防范手段，但是缺乏对于安全风险的认知。**

诚然，我们对于安全的认识是随着不断出现的安全事故逐步增强的，但是希望大家都能够有计划的逐步增强对于数据库的安全防范，主动规划推进数据安全与从挫折中学习提高实有天壤之别。对于 2011 年底的密码泄露事件，如果各大网站能够采取基本的技术手段对用户密码进行一定的加密，那么这次密码泄露的安全事件就不会显得那么初级和惹人恐慌，想一想明文密码和 MD5 加密串的区别？前者基本上意味着数据库从未从安全角度进行过任何思考和增强。

Oracle 数据库的用户信息及密码存储于一个名为 `USER$` 的数据表中（所有者为 `SYS` 用户），我们可以通过基于 `USER$` 表建立的 `DBA_USERS` 视图来查询和获得这些信息，包括加密的口令串。

在 Oracle Database 11g 之前，用户口令通过 DES 加密算法进行加密，使用用户名作为“Salt”，密码最长为 30 个字符，所有字母被强制转换为大写。从 Oracle 7 至 Oracle 10g，加密一直使用 `username` 和 `password` 串连之后进行 HASH 运算，例如 `sys/temp1` 和 `system/p1` 将会获得相同的 HASH 加密输出。

从 Oracle Database 11g 开始，Oracle 允许最多使用 30 个字符、大小写混合方式作为密码，同时支持 DES 和 SHA-1 算法进行加密（SHA-1 算法支持大小写混合，通过初始化参数 `SEC_CASE_SENSITIVE_LOGON` 开关），使用 `password||salt` 的方式进行 HASH 加密。

以下是 Oracle 9i 数据库中口令的加密形式，`DBA_USERS` 视图的 `PASSWORD` 字段显示了加密后的密键：

```
SQL> select username,password from dba_users
2  where username in ('SYS','SYSTEM','EYGLE');
USERNAME                                PASSWORD
-----                                -
EYGLE                                  B726E09FE21F8E83
SYSTEM                                A204A4CEB2C2F2FB
SYS                                   7ABF43E21B3DD9A3
```

在 Oracle 11g 中，密码从 `DBA_USERS` 视图中隐藏起来，这进一步的增强了安全性，即便具有访问视图权限的用户，也无法获得口令的加密串，由此我们也可以看出 Oracle 数据库软件的安全增强历程：

```
SQL> select username,password from dba_users
2  where username in ('SYS','SYSTEM','EYGLE');
USERNAME                                PASSWORD
-----                                -
EYGLE
SYS
SYSTEM
```

口令的加密内容存储在底层的核心表（`USER$` 是 Oracle 数据库的元数据表之一）中，以下 `PASSWORD` 字段存储的是 DES 加密值，`SPARE4` 存储的是 SHA-1 加密信息：

```
SQL> select * from v$version where rownum <2;
```

BANNER

Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production

```
SQL> select name,password,spare4 from user$  
2 where name in ('SYS','SYSTEM','EYGLE');
```

NAME	PASSWORD	SPARE4
SYS	8A8F025737A9097A	S:BBEFCB886319E6A40372B9584DBCCA6B015BFE0C7DDF5B9593FB618E0D80
SYSTEM	2D594E86F93B17A1	S:C576FB5A54D009440AC047827392215C673528067BC06659EC56E3178BAB
EYGLE	B726E09FE21F8E83	S:65857F36842AEE4470828E9BE630FEED90A67CEF0D2B40C9FE98558F6B49

关于口令的维护，Oracle 支持各种约束性限制（通过 utlpwdmg.sql 脚本启用），诸如复杂程度、长度、有效期、失败登陆次数等等，通过这些增强，Oracle 的口令限制可以定制出非常稳固的安全解决方案，如果你从未接触和研究过这些手段，那么可能就说明你的数据库还缺乏足够的第一层的安全防守。

如果我们能够从 Oracle 的安全策略入手，学习一下 Oracle 的口令安全解决方案，那么就能够构建一套较为完善的基本安全解决方案。从 Oracle 的第一个 Internet 版本 Oracle 8i（1998 年发布）开始，Oracle 就提供了一个加密包 DBMS_OBFUSCATION_TOOLKIT 用于数据安全防护，这个加密包支持 DES，3DES 和 MD5 加密算法。

通过非常简单的封装调用，DBMS_OBFUSCATION_TOOLKIT 包就能够实现数据加密，以下是一个简单的示例输出，对于给定字符串进行 MD5 加密，以 RAW 方式返回加密结果（通过创建稳固的函数，可以实现用户登陆时的即时加密、比较、认证）：

```
SQL> set serveroutput on  
SQL> BEGIN  
2 dbms_output.put_line(utl_raw.cast_to_raw(  
3 DBMS_OBFUSCATION_TOOLKIT.MD5(INPUT_STRING =>'EYGLE')));  
4 END;  
5 /  
F7FB70F1E13721034765EEC4EA1A3832  
PL/SQL procedure successfully completed.
```

从 Oracle Database 10g 开始，DBMS_CCRYPTO 包被引入到数据库中，该程序包支持更广泛的加密算法，并用于替代 DBMS_OBFUSCATION_TOOLKIT 包，在新的版本中，诸如 DES，3DES，AES，RC4，MD5，SHA-1，MD4，HMAC_MD5，HMAC_SH1 等等加密算法和加密方式都被支持。

通过选定的加密算法和加密方式，可以对重要数据进行加密和解密，我们不仅可以实现对于密码或数值、字符数据的加密，甚至可以对类似 LOB 等非结构化数据进行加密。以下范例是使用 DES 算法 CBC 模式和 PKCS5 补码规则的加密解密实现，示例模拟对于信用卡卡号的处理过程，金融类企业数据的安全性更为突出，需要进

行安全加密的类型更为丰富：

```
SQL> set serveroutput on
SQL> DECLARE
2   vcardno VARCHAR2(19) := '8610-1391-1812-8031';
3   vcardraw RAW(128) := utl_raw.cast_to_raw(vcardno);
4   crawkey RAW(128) := utl_raw.cast_to_raw('oracle10g');
5   encyraw RAW(2048);
6   decyraw RAW(2048);
7 BEGIN
8   dbms_output.put_line('CardNo : ' || vcardno);
9   encyraw := dbms_crypto.encrypt(vcardraw, dbms_crypto.des_cbc_pkcs5, crawkey);
10  dbms_output.put_line('EncdNo : ' || RAWTOHEX(utl_raw.cast_to_raw(encyraw)));
11  decyraw := dbms_crypto.decrypt(src => encyraw, typ => dbms_crypto.des_cbc_pkcs5, KEY => crawkey);
12  dbms_output.put_line('DecyNo : ' || utl_raw.cast_to_varchar2(decyraw));
13 END;
14 /
CardNo : 8610-1391-1812-8031
EncdNo : 423132463130413430303245383032343945463632454537344533413738303632454230394337454346454346314234
DecyNo : 8610-1391-1812-8031

PL/SQL procedure successfully completed.
```

我想重申的是，对于不同的数据库产品，都存在足够成熟的安全实现手段，应用这些安全手段就能够实现对于数据的基本保护，对于我们技术人最重要的是：**认识和重视数据安全问题，并逐步推动企业或组织应用安全手段进行数据安全增强。重视数据，保护数据，这是每一位技术人的共同使命！**

本书使命

本书所描述的所有恢复以及安全案例全部确有其事，但是基于用户隐私，我们隐去了所有客户相关的信息，摘录的内容涉及用户判断的，全部进行了处理，但是时间、故障内容一切属实。多年来的灾难挽救为我积累了很多素材，所以写作这本书是一次回顾的旅程，以一条主线，将众多的灾难挽救过程串联起来。这本书中的很多案例，尚属首次批露，而你也许还会注意到，书中的某些技术细节和恢复方法至今你从未在其他地方看到。

本书中的很多案例恢复非常艰难，拯救过程也花费了大量的人力、物力和时间，我们也因此赢得了数百万的商业合同，但是从内心上，我们永远不希望用户陷入到这样的境地，所以我写作了这本书，以使得这些曾经惨痛的教训可以具备更广泛的借鉴意义。

基于这些想法，我对每个案例的发生过程进行了描述，并且提出了供大家警示的规避法则。所以，我希望这是一本写给大家看的数据安全之书，不仅仅是给技术人员，更重要的是给企业数据管理者，如果不看这些案例，你也许永远不会理解数据库如何会遭遇到灭顶之灾，你也许永远无法理解为何千里之堤一朝溃于蚁穴。

当然，这仍然是一本相当深入的技术之书，我将很多案例的详细拯救过程记录下来，包括一些相当深入的技术探讨，这些技术探讨一方面可以帮助读者加深对于 Oracle 数据库技术的认知，一方面又可以在你遇到类似案例时，做出同样的营救工作。

对于我自身来说，年纪越长，就越是认识到这个世界上最为宝贵的就是时间，而如果我的分享，从警示到技术，能够帮助大家规避错误，少犯错误，在恢复时不走弯路，快速拯救数据，节省工程师们和用户的时间，那么这就是我最深刻之所愿，拯救时间，即为功德，这世界上，寸金永远也买不到一寸光阴。

这本书是用他人的灾难，为大家做警示，但愿我们都能够从中吸取教训，永远不要遇到本书所描述的种种灾难。

致谢

感谢我的朋友们，他们对我的帮助和支持使得本书的很多内容得以成型，刘磊在 ACOUG 上的演讲《猜测的力量》帮助我深入了关于 REDO 分析的案例；本书还引用了杨廷琨分析解决的一个 ASM 故障案例，此外，附录中老杨提供的函数对我们非常有用；感谢用户，是他们的信赖使得我们能够接触种种艰难的案例，并帮助他们挽回数据；感谢支持帮助过我的朋友们以及一贯支持我的读者们，这书中真挚的内容就是我最好的回报。

感谢我的好友丁晓强同学，他在支付宝公司进行了多年的安全与运维体系的建设与管理工 作，他基于实践而来的对于安全和运维的真知灼见给予了本书很多肯綮的建议，这些建议让我对于安全有了更加系统全面的理解，从而也对本书的架构做出了调整，虽然有很多想法没能在本书中体现，但是我相信，我仍然有机会在未来和大家继续分享我从他那里学来的宝贵经验。

再次感谢杨廷琨，他在本书出版之前，通读了全书书稿，细致到帮我修改一个敲错的字母，一个写错的汉字，这份细致认真让我钦佩得无以复加；他还帮助我增加了两个警示条目，来自于他感触最为深刻的技术经历。老杨的工位和我相邻，在工作中我随时 的请教可以即刻得到他绝妙的提示，让我节省了大量的工作时间，这是达成本书的另外一个重要助力。

我还要感谢我的太太 Julia 和儿子，以及我的家人，我为写作这本书，牺牲了很多陪伴他们的时间，也正因为有他们的支持，我才能够不断的写作下去。

最后，我但愿书中这些看起来似乎遥远的故事，能够警醒某些你曾经似曾相识的操作，并且永远不要面对这样的灾难。

盖国强（Eygle）

2012-01-20 初稿

2012-03-01 定稿于北京

目 录

靡不有初，鲜克有终	1
以空间之由——误操作删除数据文件恢复案例两则	3
灾难描述	3
案例警示	4
技术回放	5
恢复过程	7
通过文件描述符进行数据恢复	7
技术难点	21
通过 BBED 获取文件号信息	21
通过 od 命令获得文件号信息	24
以拯救之因——强制恢复导致 ORA-600 4000 错误案例	29
灾难描述	29
案例警示	30
技术回放	31
恢复过程	35
ORA-600 4000 错误揭秘	36
通过 _minimum_giga_scn 消除 SCN 异常	41
ORA-600 4194 错误 UNDO 故障消除	45
以优化之名——存储优化导致表空间误删除案例	49
灾难描述	49
案例警示	50
技术回放	51
以安全之期	59

VALIDATE 实现备份验证	59
数据库备份加密	62
口令模式	63
透明模式	65
混合模式	68
透明加密（TDE）技术	68
合抱之木，起于毫末	75
Oracle 数据库软件发布序列	77
一个逻辑坏块引发的灾难	81
案例警示	81
技术回放	82
一个硬盘坏块引发的灾难	83
灾难描述	83
案例警示	83
技术回放	85
AIX 系统 ODM 简介	85
ASM 头块备份机制	86
kfed 工具编译与使用	89
手工修复 ASM 案例一则	93
灾难描述	93
技术回放	93
PROVISIONED 磁盘状态分析	94
使用 kfed 修改 ASM 磁盘头信息	96
ASM 数据抽取恢复	103
通过 AMDU 恢复数据案例一则	103
灾难描述	103
案例警示	103

技术回放	104
AMDU 工具	104
文件分析	107
AMDU 文件恢复	108
未雨绸缪，防患未然	111
DBA 四大守则	113
DBA 守则外两则	115
各种惨痛的案例	119
系统级误删除案例	119
数据库误删除案例	125
通过触发器实现 DDL 监控	126
主备环境错误案例	136
业务高峰误操作案例	141
备份级误操作案例	146
进程级别误操作案例	149
数据文件误操作案例	151
误关闭生产库案例	153
系统存储级误删除案例	157
亡羊补牢，未为迟也	161
数据篡改案例解析	163
案例描述	163
案例警示	163
技术回放	164
故障分析的过程	165
日志文件的转储	167
LOGMNR 解析	172
案例之深入解析	174

技术难点	186
密码安全与加密	193
明察秋毫，见微知著	215
一次碰撞引发的灾难——ASM 保护式文件离线引发故障	217
灾难描述	217
案例警示	217
技术回放	218
恢复过程	222
又一次碰撞引发的灾难——文件离线与归档缺失案例	225
灾难描述	225
案例警示	225
技术回放	227
恢复过程	232
空间与文件离线——离线表空间加载修复	247
灾难描述	247
案例警示	247
技术回放	248
恢复过程	256
技术提示	262
关于归档空间的设置	262
关于检查点的一致性调整	267
心存目想，三思后行	273
Truncate 导致的灾难——核心字典表误操作 TRUNCATE	275
灾难描述	275
案例警示	275
技术回放	276

恢复过程.....	282
脚本错误导致的灾难——数据库整体被删除故障.....	289
灾难描述.....	289
案例警示.....	289
技术回放.....	290
恢复过程.....	291
千里之堤，溃于蚁穴.....	299
一个字符引发的灾难——大小写字符疏忽导致的维护故障.....	301
灾难描述.....	301
案例警示.....	301
案情解析.....	302
技术回放.....	310
一个盘符引发的灾难——判断失误导致的误格式化故障.....	325
灾难描述.....	325
案例警示.....	325
技术回放.....	326
物尽其用，人尽其才.....	329
关库与关机——强制关机导致的写丢失故障.....	331
灾难描述.....	331
案例警示.....	331
恢复过程.....	332
技术提示.....	361
从小恙到灾难——重建控制文件失误导致的故障.....	363
灾难描述.....	363
案例警示.....	363
技术回放.....	364

尺有所短，物有不足——硬件故障导致的灾难一则 375

 灾难描述 375

 案例警示 375

 技术回放 376

附录一 BBED 的说明 379

附录二 函数 f_get_from_dump..... 382

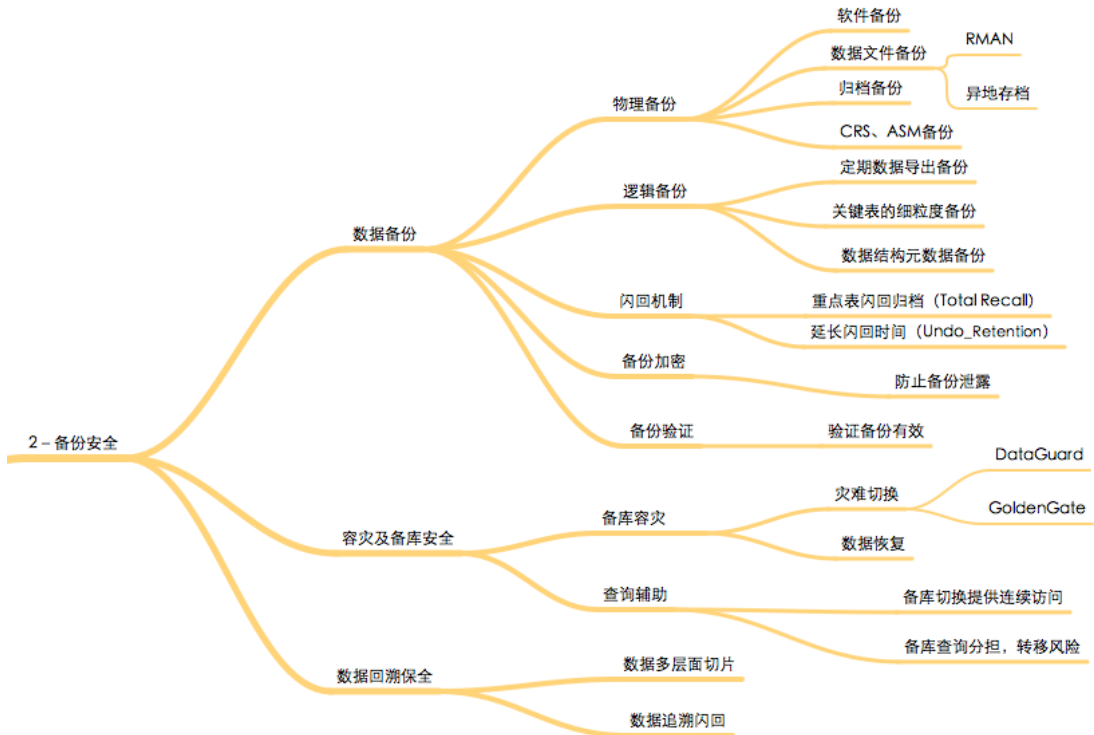
参考资料 387

靡不有初，鲜克有终

荡荡上帝，下民之辟。疾威上帝，其命多辟。

天生烝民，其命匪谌。靡不有初，鲜克有终。

——《诗经·大雅·荡》



“靡不有初，鲜克有终”，这句话的意思大致是说，人们做事大都能有一个良好的开端，但是很少有人能够善始善终。

每年岁末，我们都能够接触到大量的数据库安全相关案例，一个疏忽、一个原本为了安全而执行的操作，最终使数据库陷于绝境。而这些陷入绝境的数据库，多半都没能进行及时有效的备份，当灾难发生时，处理起来就非常棘手。

对于很多企业，备份缺失并非说明数据不够重要，很多时候客户会因为长期以来数据库一直稳定运行而麻痹大意，忽略或者忽视备份检查，直至出现故障时才发现备份不足，或已有的备份不可用，却已为时太晚。

而对于某些海量数据企业，常规的备份恢复时间已经不能满足业务连续性要求，所以一旦遭遇故障，某些时候可能仍然需要采取一些极端手段进行数据恢复和强制修复。

篇首图是关于备份安全的思考和总结，企业应当充分保障数据备份，并考虑容灾与备库的安全保障，对于特定的企业要求，可能还需要保证数据的可回溯性。所有这些都是在备份安全中应当考虑的。

关于图中内容，还需要着重强调以下几点。

1. 注意备份加密的重要性。如果不能保护备份和备份集，那么一旦发生丢失、泄露就可能导致严重的安全问题。
2. 在数据库管理维护中，应当明确闪回机制的功能。在出现常规数据误操作如删除等问题时，可以优先尝试使用闪回功能进行恢复。
3. 定期的备份检查和备份验证必不可少。如果没有确认和验证，那么备份的有效性就无法保证，在故障发生时就很有可能遭遇无法恢复的灾难。
4. 备库是非常有力的数据保障机制。如果环境许可，对生产数据库构建一个 DataGuard 备库容灾环境，可以在危急关头挽救数据。

我们应当深入思考备份安全问题，如果能够构建有效的备份保全策略，那么企业的数据安全就有了基本的保障。

下面我们就来一起审视几则数据库安全事故，这些数据库在 2011/2012 岁末未能善终。

以空间之由

误操作删除数据文件恢复案例两则

我们都应当熟悉以下这段诗句，从一颗钉子，到一个国家。

少了一颗钉子，丢了一块蹄铁；
少了一块蹄铁，丢了一匹战马；
少了一匹战马，丢了一个骑手；
少了一个骑手，丢了一场胜利；
少了一场胜利，丢了一个国家。

——詹姆斯·格莱克《混沌学》

我们见过很多客户，因为空间紧张而东拼西凑，最终导致难以挽回的数据灾难，回头看来是那么得不偿失！因为一块硬盘，损失整个数据库，看起来是多么的荒诞。然而，这样荒诞的事情还在不断发生。

灾难描述

2011 年 12 月 30 日，某运营商客户，在遭受数据损失之后请求我们协助进行数据恢复。整个数据灾难的过程如下。

1. 凌晨，数据库归档日志写满磁盘，因而无法继续归档，数据库服务中断。
2. 在进行空间释放时，删除了一个认为不再需要的目录。
3. 删除目录之后，发现数据库服务受到影响。
4. 经确认，该目录包含 7 个 16GB 大小的在用数据文件。
5. 在试图通过备份来恢复时，发现之前的备份是不成功的。
6. 灾难形成。

这是一个由删除引发的严重故障，如果数据不可恢复，灾难影响将是非常严重的。

案例警示

分析整个灾难的形成过程后，我们总结出了如下教训。

1. 数据库需要全面的系统规划和监控。

首先，客户的数据库系统缺乏监控，直到归档路径空间使用率达到 100%，出现错误影响了业务应用，客户才意识到数据库出现了问题。按照常规运维要求，系统应当部署必要的监控手段，在空间达到一定阈值时即报警，比如空间使用率达到 80%。

我曾经反复警告：不要轻信归档模式的好处，对于没有良好备份、运维的数据库系统，启用归档模式往往是个灾难，只会反复带来麻烦。而且，归档模式不可避免地要牺牲一定的性能。

此外，在规划数据库系统时，也应当尽可能预留足够的磁盘空间。如今磁盘的成本已经越来越低，因为磁盘空间而导致灾难实属不该。

2. 对数据库的破坏性操作需要谨慎。

在出现空间告警后，立即清理空间显然过于草率。在操作系统上的空间清理应当和数据库操作遵循同样的守则，即：在任何破坏性操作之前，应当进行反复确认并根据需要进行有效备份。

相应的操作人员应当铭记：情况越紧迫，越应冷静，避免犯下无法挽回的低级错误。

在我们的职业生涯中，关于删除文件养成的一个习惯是：以转移（MOVE）替代删除（DELETE/RM），经过一段时间观察确保无影响后，再进行最终的文件移除。

删除，这个简单的操作导致了数不清的灾难，因此应当极为谨慎地对待。

3. 数据环境运维必须遵守一定的安全守则。

在数据环境的运维中，应当制订制度性的规范并严格执行。规范和制度会增加工作的复杂性，但也会极大地避免出现误操作的概率。

无约束的技术自由，会带来无法预料的后果；良好的规范和制度，其良性输出是可以预期和预知的。

在本例的用户环境中，显然没有明确的维护规则，维护人员并不清楚空间的分配使用情况，也不清楚特定目录存储的内容和含义，这才发生了误删除操作。如果系统能够维护一份部署文档，明确各目录存储的用途，那么错误也就不会这么容易发生了。

按照正常的规则，数据库服务器应当维护一份文件部署分布文档，明确指出各目录结构及用途，然后制订空间清理规则；在发生空间紧张时，需要通过文档来确定哪些目录可以删除，哪些目录必须保留。这样就可以

规避掉误删除的风险。

4. 数据备份应进行必要的检查和确认。

很多用户在部署了数据备份手段之后就认为可以高枕无忧了，而很少去检验备份的有效性和完整性，一旦出现问题需要恢复，才发现备份无效，就很难挽回了。我们曾经多次遇到客户在恢复时发现磁带不可用的数据安全故障。

所以提醒大家：在进行了数据备份之后，要定期检查备份成功与否，以及备份的完好完整性，如果使用了备份介质，还要检查备份介质的完好性。很多用户在进行了磁带备份之后，恢复时却发现磁带不可读取，因而造成无法挽救的灾难，这是非常令人感到惋惜的。

5. 避免在疲劳或不清醒状态独自做出重要判断。

这次故障首次发生和处置是在凌晨。人在疲劳、熬夜或者睡梦中被叫起的半梦半醒状态下，所有的判断都可能考虑不周。所以，如同拒绝疲劳驾驶一样，应当拒绝疲劳运维，或者至少避免在不清醒的状态下独自做出重要判断，必要时，至少获得一个额外的指导或确认。

6. 在对故障做出清晰判断之前不要贸然采取措施。

这个故障的恢复得益于用户的正确判断。在问题出现后，用户并未关闭数据库，而是将其保持在打开状态下，然后进行故障分析处理，这就为简便恢复提供了可能。

所以当遇到数据库故障时，在得出清晰的分析结论和处理方法之前，不要贸然采取措施，草率的决策可能导致问题复杂化，进而引发更加复杂的级联故障。

遵循一定的守则和规范，将显著提高数据安全性，保障数据环境的稳定运行。

技术回放

接下来让我们通过技术回放，来看一看数据库是如何陷入这场灾难的。

从告警日志看最初数据库出现的问题是归档日志无法写出，出现归档错误，归档停滞导致数据库的所有 DML 事务无法进行，在线业务受到影响。

以下是日志摘录信息，注意时间，故障出现在 2011 年 12 月 30 日凌晨 3 点 34 分。发生在夜间的故障通常影响的业务范围小，但是其处理响应也可能会相对缓慢，处理上也可能出现误操作，应当警惕。

Fri Dec 30 03:34:33 2011

```

ARC0: Encountered disk I/O error 19502
ARC0: Closing local archive destination LOG_ARCHIVE_DEST_1:
'/archlog/1_6578_765575017.dbf' (error 19502) (com1)
ARC0: I/O error 19502 archiving log 5 to '/archlog/1_6578_765575017.dbf'
ARCH: Archival stopped, error occurred. Will continue retrying
ORACLE Instance com1 - Archival Error
ORA-16038: log 5 sequence# 6578 cannot be archived
ORA-19502: write error on file "", block number (block size=512)
ORA-00312: online log 5 thread 1: '/oradata2/redolog5_1.dbf'
ORA-00312: online log 5 thread 1: '/oradata3/redolog5_2.dbf'
ARCH: Archival stopped, error occurred. Will continue retrying

```

经过处理，在凌晨 5 点 22 分左右，归档得以继续，归档进程从失败中得到释放。此次的故障处理时间大约是 50 分钟。

```

Fri Dec 30 05:22:51 2011
Archived Log entry 15601 added for thread 1 sequence 6578 ID 0xffffffffcf4313f7 dest
1: krse_arc_driver_core: Successful archiving of previously failed ORL
Archiver process freed from errors. No longer stopped

```

那么，这里的空间是如何释放出来的呢？

显然是操作系统级别的行为，通过删除归档、删除文件释放空间，但是注意，这位在凌晨被吵醒的工程师很可能草率地做出了错误的判断。

后果在日志中表露无疑，十几分钟后，一系列的数据文件出现无法访问、无法打开的读/写错误。

```

Fri Dec 30 05:38:22 2011
Errors in file /opt/oracle/app/diag/rdbms/com/com1/trace/com1_m000_4496.trc:
ORA-01116: error in opening database file 229
ORA-01110: data file 229: '/oradata4/LTYT_DATA55.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
Errors in file /opt/oracle/app/diag/rdbms/com/com1/trace/com1_m000_4496.trc:
ORA-01116: error in opening database file 235
ORA-01110: data file 235: '/oradata4/LTYT_DATA56.dbf'

```

```
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
Errors in file /opt/oracle/app/diag/rdbms/com/com1/trace/com1_m000_4496.trc:
ORA-01116: error in opening database file 232
ORA-01110: data file 232: '/oradata4/SMS_DATA47.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
```

经过检查，最终用户确认，oradata4 整个目录在操作系统上被删除，其中的所有数据文件荡然无存。以下是丢失的文件列表和状态，从 v\$datafile 中可以查询到这些信息。这些都是极为重要的业务数据文件，必须要恢复出来。

TS#	FILE#	NAME	BYTES	STATUS
5	229	/oradata4/YDDY_DATA55.dbf	0	ONLINE
19	230	/oradata4/EFB01.dbf	0	ONLINE
18	231	/oradata4/undotbs203.dbf	0	RECOVER
9	232	/oradata4/SMS_DATA47.dbf	0	ONLINE
9	233	/oradata4/SMS_DATA48.dbf	0	ONLINE
9	234	/oradata4/SMS_DATA49.dbf	0	ONLINE
5	235	/oradata4/YDDY_DATA56.dbf	0	ONLINE

而当用户尝试从备份开始恢复时，发现备份无法还原出来，此前的几次备份都失败了，不可用。现在问题就相当严重了。

恢复过程

通过文件描述符进行数据恢复

这种情况下的恢复有多种可能（后面的章节还会介绍），这个用户的恢复是最简单的一种情况。恢复过程主要利用了 UNIX 系统下的“文件描述符”，通过文件描述符的指向映射恢复数据文件。

1. 通过文件描述符进行恢复

在操作系统中，内核（kernel）利用文件描述符（file descriptor）来访问文件。文件描述符是非负整数。打开现存文件或新建文件时，内核会返回一个文件描述符。读/写文件也需要使用文件描述符来指定对应的文件。

实际上文件描述符是进程态的一种组织，进程通过文件描述符将文件纳入其地址空间，然后文件描述符指向具体的文件结构。图 1 是进程与文件描述符的示意图，众多的进程都通过文件描述符进行文件的空间和地址映射。

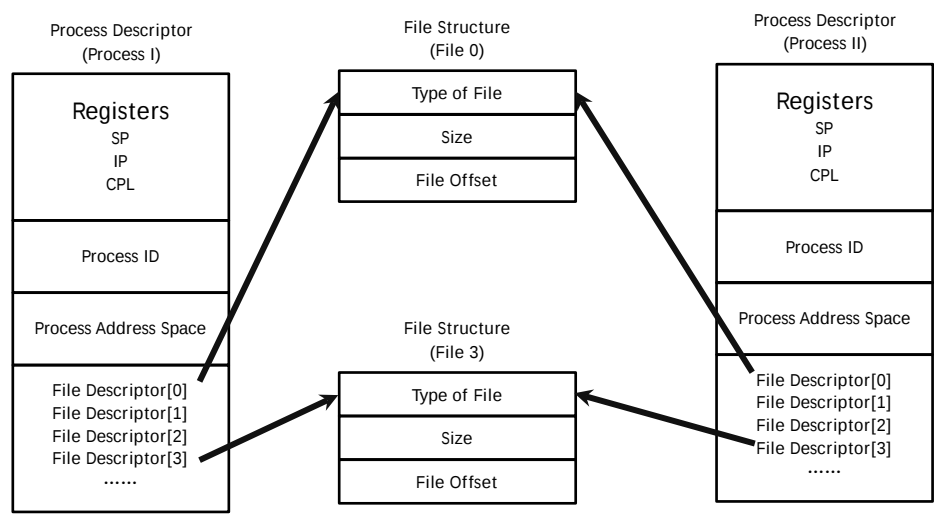


图 1 进程与文件描述符示意图

在 UNIX、Linux 系统中，误删除数据文件后，虽然该文件已从操作系统中删除，但是其文件句柄仍由数据库进程打开持有，所以在数据库层面仍然不会释放其链表信息，因而也就能从进程的地址信息中，通过复制将其直接恢复。但是请注意，这要求数据库不能中途关闭，如果关闭了数据库，则所有文件句柄均被释放，文件就真的难以找回了。

再次重申这则案例给我们的一个重要教训：如果系统出现问题，在做出准确判断之前，绝不要贸然采取应对措施，更不能贸然关闭或者重启数据库或主机操作系统。因为在一些特殊情况下，有可能主机在关闭后无法启动，或者数据库在关闭后无法启动。

还有一个特殊的地方需要注意，在前面的文件列表输出中，18 号 UNDO 文件的状态已经由 ONLINE 变成了 RECOVER，这说明这个文件还遇到了其他问题。从 RAC 环境的节点 2 上，我们找到了相关的日志信息。

首先，用户尝试删除丢失的 UNDO 文件，遇到了错误而失败。

```
Fri Dec 30 10:12:39 2011
alter tablespace UNDOTBS2 drop datafile '/oradata4/undotbs203.dbf'
ORA-3262 signalled during:
alter tablespace UNDOTBS2 drop datafile '/oradata4/undotbs203.dbf'...
Fri Dec 30 10:13:40 2011
Errors in file /opt/oracle/app/diag/rdbms/com/com2/trace/com2_j000_7662.trc:
ORA-12012: error on auto execute of job 41
ORA-01116: error in opening database file 231
ORA-01110: data file 231: '/oradata4/undotbs203.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
```

接下来，用户尝试创建一个新的 UNDO 表空间，结果失败。这个尝试执行了多次，这在实践中也应当尽量避免，反复尝试一个出错的命令是极其不慎重的。

```
Fri Dec 30 10:15:07 2011
create tablespace undotbs3 datafile '/oradata4/undotbs301.dbf' size 16374m
ORA-604 signalled during:
create tablespace undotbs3 datafile '/oradata4/undotbs301.dbf' size 16374m...
Fri Dec 30 10:15:33 2011
create tablespace undotbs3 datafile '/oradata4/undotbs301.dbf' size 16374m
Fri Dec 30 10:15:51 2011
Errors in file /opt/oracle/app/diag/rdbms/com/com2/trace/com2_j000_8575.trc:
ORA-12012: error on auto execute of job 41
ORA-01116: error in opening database file 231
ORA-01110: data file 231: '/oradata4/undotbs203.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
Fri Dec 30 10:17:25 2011
ORA-604 signalled during:
create tablespace undotbs3 datafile '/oradata4/undotbs301.dbf' size 16374m...
```

再往后，用户发出 OFFLINE 命令，成功地将 UNDO 文件离线。这里的成功离线，也就意味着存储上的文件句柄被释放，文件被删除。此后需要通过这个 UNDO 进行回滚或一致性读的操作都会出现“无法读取文件”错误。

```
Fri Dec 30 10:23:31 2011
alter database datafile '/oradata4/undotbs203.dbf' offline
Completed: alter database datafile '/oradata4/undotbs203.dbf' offline
Fri Dec 30 10:23:39 2011
Errors in file /opt/oracle/app/diag/rdbms/com/com2/trace/com2_ora_7511.trc:
ORA-00376: 此时无法读取文件 231
ORA-01110: 数据文件 231: '/oradata4/undotbs203.dbf'
ORA-00376: 此时无法读取文件 231
ORA-01110: 数据文件 231: '/oradata4/undotbs203.dbf'
Fri Dec 30 10:23:52 2011
Errors in file /opt/oracle/app/diag/rdbms/com/com2/trace/com2_ora_7511.trc:
ORA-00376: 此时无法读取文件 231
ORA-01110: 数据文件 231: '/oradata4/undotbs203.dbf'
ORA-00376: 此时无法读取文件 231
ORA-01110: 数据文件 231: '/oradata4/undotbs203.dbf'
```

2. 关于文件状态的说明

关于数据文件的离线和状态，在此需要做一个简要的说明。前面我们注意到了文件的两种状态：ONLINE 和 RECOVER。对于数据文件，还有两种状态，分别是 SYSTEM 和 OFFLINE，系统表空间的状态为 SYSTEM。

下面是一个正常的数据库。

```
SQL> select name,status from v$datafile;
NAME                                STATUS
-----
/home/ora11g/oradata/orcl11g/system01.dbf    SYSTEM
/home/ora11g/oradata/orcl11g/sysaux01.dbf    ONLINE
/home/ora11g/oradata/orcl11g/undotbs01.dbf   ONLINE
/home/ora11g/oradata/orcl11g/users01.dbf     ONLINE
```

在归档模式下，对于文件的离线操作，会将文件状态标记为 RECOVER（正因为存在归档，才存在恢复的

可能性)，文件的离线不会执行表空间级别的检查点。为了维持表空间级别的一致性，在对文件 ONLINE 时必须执行恢复。而如果是基于表空间进行，则表空间检查点会被执行，在文件进一步 ONLINE 时，就不需要进行 RECOVER。

下面对文件执行文件级别的 OFFLINE 操作。

```
SQL> archive log list;
Database log mode          Archive Mode
Automatic archival         Enabled
Archive destination        USE_DB_RECOVERY_FILE_DEST
Oldest online log sequence 120
Next log sequence to archive 122
Current log sequence        122

SQL> alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' offline;
Database altered.

SQL> select name,status from v$datafile;
NAME                                STATUS
-----
/home/ora11g/oradata/orcl11g/system01.dbf    SYSTEM
/home/ora11g/oradata/orcl11g/sysaux01.dbf     ONLINE
/home/ora11g/oradata/orcl11g/undotbs01.dbf    ONLINE
/home/ora11g/oradata/orcl11g/users01.dbf      RECOVER

SQL> alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' online;
alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' online
*

ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: '/home/ora11g/oradata/orcl11g/users01.dbf'

SQL> recover datafile 4;
Media recovery complete.

SQL> alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' online;
Database altered.
```

对于表空间级别的离线和在线加载，则可以顺利地进行。

```
SQL> alter tablespace users offline;
```

Tablespace altered.

```
SQL> select name,status from v$datafile;
```

NAME	STATUS
-----	-----
/home/ora11g/oradata/orcl11g/system01.dbf	SYSTEM
/home/ora11g/oradata/orcl11g/sysaux01.dbf	ONLINE
/home/ora11g/oradata/orcl11g/undotbs01.dbf	ONLINE
/home/ora11g/oradata/orcl11g/users01.dbf	OFFLINE

```
SQL> alter tablespace users online;
```

Tablespace altered.

而在非归档模式下，是不允许对文件级别执行离线操作的，可以改用 **OFFLINE DROP** 对文件进行离线操作。但如果后期的日志不可用，则这个离线的文件可能永远无法正常 **ONLINE** 加载。

```
SQL> alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' offline;
```

```
alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' offline
```

```
*
```

```
ERROR at line 1:
```

```
ORA-01145: offline immediate disallowed unless media recovery enabled
```

```
SQL> alter database datafile '/home/ora11g/oradata/orcl11g/users01.dbf' offline drop;
```

Database altered.

```
SQL> select name,status from v$datafile;
```

NAME	STATUS
-----	-----
/home/ora11g/oradata/orcl11g/system01.dbf	SYSTEM
/home/ora11g/oradata/orcl11g/sysaux01.dbf	ONLINE
/home/ora11g/oradata/orcl11g/undotbs01.dbf	ONLINE
/home/ora11g/oradata/orcl11g/users01.dbf	RECOVER

至于 `v$datafile` 中的状态定义，来自以下 SQL 查询，从中可以看到 Oracle 对于状态的标示与定义。通过视图的底层创建语句分析 Oracle 上层体现，是学习 Oracle 技术的重要方法之一。这些视图的定义可以通过 `v$fixed_view_definition` 来获得。

```
select fe.inst_id,
       fe.fenum,
       to_number(fe.fecrc_scn),
```

```

to_date(fe.fecrc_tim,
        'MM/DD/RR HH24:MI:SS',
        'NLS_CALENDAR=Gregorian'),
fe.fetsn,
fe.ferfn,
-- 此处即文件状态的定义信息，其内容来自 x$kccfe 结构
-- X$KCCFE - File [E]ntries ( from control file )
decode(fe.fetsn,
        0,
        decode(bitand(fe.festa, 2), 0, 'SYSOFF', 'SYSTEM'),
        decode(bitand(fe.festa, 18),
                0,'OFFLINE', 2, 'ONLINE','RECOVER')),
decode(fe.fedor, 2, 'READ ONLY',
        decode(bitand(fe.festa, 12),0,'DISABLED',4,'READ ONLY',
                12,'READ WRITE','UNKNOWN')),
to_number(fe.fecps),
to_date(fe.fecpt, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
to_number(fe.feurs),
to_date(fe.feurt, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
to_number(fe.fests),
decode(fe.fests,
        NULL, to_date(NULL),
        to_date(fe.festt,'MM/DD/RR HH24:MI:SS','NLS_CALENDAR=Gregorian')),
to_number(fe.feofs),
to_number(fe.feonc_scn),
to_date(fe.feonc_tim, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
fh.fhfsz * fe.febsz,
fh.fhfsz,
fe.fecsz * fe.febsz,
fe.febsz,
fn.fnnam,
fe.fefdb,
fn.fnbof,

```

```

decode(fe.fepax, 0, 'UNKNOWN', 65535, 'NONE', fnaux.fnnam),
to_number(fh.fhfirstunrecscn),
to_date(fh.fhfirstunrectime, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
fe.fepdi,
fe.fefcrs,
fe.fefcrt,
decode(fe.fefdb, 1, 'YES', 'NO'),
fe.feplus,
fe.feprls,
fe.feprlt
from x$kccfe fe, x$kccfn fn, x$kccfn fnaux, x$kcvm fh
where ((fe.fepax != 65535 and fe.fepax != 0 and fe.fepax = fnaux.fnnam) or
      ((fe.fepax = 65535 or fe.fepax = 0) and fe.fenum = fnaux.fnfno and
      fnaux.fntyp = 4 and fnaux.fnnam is not null and
      bitand(fnaux.fnflg, 4) != 4 and fe.fefnh = fnaux.fnnam))
and fn.fnfno = fe.fenum and fn.fnfno = fh.hxfil
and fe.fefnh = fn.fnnam and fe.fedup != 0
and fn.fntyp = 4 and fn.fnnam is not null
and bitand(fn.fnflg, 4) != 4
order by fe.fenum

```

3. 具体案例的恢复过程

几乎同样的案例，2011 年 12 月 18 日我们为用户处理过另外一则，这里一并介绍一下。这一案例同样是归档日志由于空间不足无法写出，同样是在接近凌晨时出现问题，也同样是因为误操作删除了整个目录。

以下日志摘录显示故障时间是在 23 : 41，午夜时分，错误提示为空间满，无法写归档日志，错误由归档进程抛出。

```

Sun Dec 18 23:41:15 2011
Errors in file /u01/admin/sxnms/bdump/sxnms_arc0_784.trc:
ORA-19504: failed to create file "/ora_backup/arch/1_14289.dbf"
ORA-27044: unable to write the header block of file
SVR4 Error: 28: No space left on device

```

```

Additional information: 1
ARC0: Archiving not possible: error count exceeded
Sun Dec 18 23:41:16 2011
ARC1: I/O error 19502 archiving log 4 to '/ora_backup/arch/1_14288.dbf'
Sun Dec 18 23:41:16 2011
Errors in file /u01/admin/sxnms/bdump/sxnms_arc1_788.trc:
ORA-19502:
write error on file "/ora_backup/arch/1_14288.dbf", blockno 63489 (blocksize=512)
ORA-27063: skgfsopo: number of bytes read/written is incorrect
Additional information: 7680
Additional information: 1048576
ORA-19502:
write error on file "/ora_backup/arch/1_14288.dbf", blockno 57345 (blocksize=512)
ARC1: Archiving not possible: error count exceeded

```

在以上错误提示中，有一个关于日志块大小的说明，Oracle 的日志块大小是 512 字节，与数据库文件块大小不同。注意，DBV 工具不能用于检查日志文件。

以上问题出现后，经过一系列处理（删文件），日志最终可以继续归档，这显然也是得益于空间释放。

```

Mon Dec 19 01:07:35 2011
ARC1: Evaluating archive log 4 thread 1 sequence 14288
ARC1: Beginning to archive log 4 thread 1 sequence 14288
Creating archive destination LOG_ARCHIVE_DEST_1: '/ora_backup/arch/1_14288.dbf'
Mon Dec 19 01:07:36 2011
ARC0: Evaluating archive log 4 thread 1 sequence 14288
ARC0: Unable to archive log 4 thread 1 sequence 14288
Log actively being archived by another process
ARC0: Evaluating archive log 5 thread 1 sequence 14289
ARC0: Beginning to archive log 5 thread 1 sequence 14289
Creating archive destination LOG_ARCHIVE_DEST_1: '/ora_backup/arch/1_14289.dbf'
Mon Dec 19 01:07:39 2011
ARC1: Completed archiving log 4 thread 1 sequence 14288
Archiver process freed from errors. No longer stopped

```

但是紧急处理及后续处理出现了误操作，大量文件被误删除，具体的删除时间不确定，不过日志中出现的

提示是在中午 12 点左右。首先报告的错误是日志文件无法访问，文件状态不能获得。

```
Mon Dec 19 11:52:11 2011
ARCH: Evaluating archive    log 5 thread 1 sequence 14294
Mon Dec 19 11:52:11 2011
Errors in file /u01/admin/cinms/udump/cinms_ora_16062.trc:
ORA-00313: open failed for members of log group 5 of thread 1
ORA-00312: online log 5 thread 1: '/u02/oradata/cinms_redo05.log'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
ARCH: Error 313 opening/verifying online redo log 5
Mon Dec 19 11:52:11 2011
Errors in file /u01/admin/cinms/udump/cinms_ora_16062.trc:
ORA-00313: open failed for members of log group 5 of thread 1
ORA-00312: online log 5 thread 1: '/u02/oradata/cinms_redo05.log'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
Mon Dec 19 11:52:11 2011
ARC1: Evaluating archive    log 5 thread 1 sequence 14294
ARC1: Unable to archive log 5 thread 1 sequence 14294
      Log actively being archived by another process
```

随后用户确认 u02 目录下的文件全被删除。此次事件中共有 24 个数据文件受到影响，都是极其重要的用户数据。由于日志文件也存储在 u02 目录中，因而在进行切换写入时，首先由日志文件报告了写错误。

以下是丢失的数据文件列表。

FILE_ID	FILE_NAME	TABLESPACE_NAME	MB
8	/u02/oradata/sxnms_user01.dbf	SXNMS_USER	1000
9	/u02/oradata/sxnms_user02.dbf	SXNMS_USER	1000
10	/u02/oradata/sxnms_user03.dbf	SXNMS_USER	1000
11	/u02/oradata/sxnms_user04.dbf	SXNMS_USER	1000
27	/u02/oradata/sxnms_user07.dbf	SXNMS_USER	1000

28	/u02/oradata/sxnms_user08.dbf	SXNMS_USER	1000
29	/u02/oradata/sxnms_user09.dbf	SXNMS_USER	1024
30	/u02/oradata/sxnms_user10.dbf	SXNMS_USER	1024
31	/u02/oradata/sxnms_user11.dbf	SXNMS_USER	1024
36	/u02/oradata/sxnms_user15.dbf	SXNMS_USER	1000
37	/u02/oradata/sxnms_user16.dbf	SXNMS_USER	1000
50	/u02/oradata_res/data_res09.dbf	DATA_RES	1000
51	/u02/oradata_res/data_res10.dbf	DATA_RES	1000
52	/u02/oradata_res/data_res11.dbf	DATA_RES	1000
53	/u02/oradata_res/data_res12.dbf	DATA_RES	1000
55	/u02/oradata/sxnms_backup03	SXNMS_BACKUP	1314
56	/u02/oradata/sxnms_backup04	SXNMS_BACKUP	1313.5
61	/u02/oradata/sxnms_indexes05.dbf	SXNMS_INDEXES	1000
63	/u02/oradata/sxnms_user25.dbf	SXNMS_USER	1000
67	/u02/oradata/sxnms_indexes09.dbf	SXNMS_INDEXES	1000
68	/u02/oradata/sxnms_user27.dbf	SXNMS_USER	1000
86	/u02/oradata/sxnms_indexes10.dbf	SXNMS_INDEXES	1000
87	/u02/oradata/sxnms_indexes11.dbf	SXNMS_INDEXES	1024
88	/u02/oradata/sxnms_indexes12.dbf	SXNMS_INDEXES	1024

24 rows selected.

用户在这种情况下做出了正确的判断，未关闭数据库，并且和我们取得了联系。这种情况，是完全可以利用文件描述符的方式来进行恢复的。

首先找到一个后台进程（如 DBWR 进程），通过其进程地址找到文件句柄（/proc/<proc_id>/fd）。以下就是数据库文件的句柄显示信息，复制这些文件即可恢复那些被删除但尚未消失的数据文件。

```
bash-2.05$ ps -ef | grep dbw | grep -v grep
  oracle   762      1  0   Jun 10 ?        217:30 ora_dbw0_sxnms

bash-2.05$ ls /proc/762/fd
0      12    256  260  264  268  272  276  280  284  288  292  296  3    303  307  311
315    319    323  327  331  335  339  343  347  61   13   257  261  265  269  273  277
281    285    289  293  297  300  304  308  312  316  320  324  328  332  336  340  344
348    710    14   258  262  266  270  274  278  282  286  290  294  298  301  305  309
```

```

313 317 321 325 329 333 337 341 345 4      811 2      259 263 267 271 275
279 283 287 291 295 299 302 306 310 314 318 322 326 330 334 338 342
346 5      9

```

```
db2% ls -l | grep oracle
```

```

-r--r--r-- 1 oracle dba      657920 Apr 26 2002 11
-rw-r----- 1 oracle dba      923 Jun 10 2011 12
-rw-rw---- 1 oracle dba      24 Jun 10 2011 13
-rw-r----- 1 oracle dba    1859584 Jan 5 16:42 256
-rw-r----- 1 oracle dba    1859584 Jan 5 16:42 257
-rw-r----- 1 oracle dba    1859584 Jan 5 16:42 258
-rw-r----- 1 oracle dba    414195712 Jan 5 16:41 259
-rw-r----- 1 oracle dba    6291464192 Jan 5 16:42 260
-rw-r----- 1 oracle dba    161488896 Jan 5 16:18 261
-rw-r----- 1 oracle dba     20979712 Jan 5 16:18 262
-rw-r----- 1 oracle dba     26222592 Jan 5 16:18 263
-rw-r----- 1 oracle dba     10493952 Jan 5 16:18 264
-rw-r----- 1 oracle dba     473178112 Jan 5 16:18 265
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:40 266
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:18 267
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:40 268
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:40 269
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:41 270
-rw-r----- 1 oracle dba    1048584192 Jan 5 16:42 271
-rw-r----- 1 oracle dba    1384652800 Jan 5 16:18 272

```

本例中，我们在复制文件恢复之后，创建了一个新的目录（保留原来的目录结构不动），随后通过 offline、rename、recover、online 四个步骤恢复这些文件，将其加载到数据库中。以下是简要步骤记录。

首先复制文件到新分配的目录空间。

```

cp /proc/762/fd/266 /new_u02/oradata/cinms_user01.dbf
cp /proc/762/fd/267 /new_u02/oradata/cinms_user02.dbf
cp /proc/762/fd/268 /new_u02/oradata/cinms_user03.dbf
cp /proc/762/fd/269 /new_u02/oradata/cinms_user04.dbf
cp /proc/762/fd/285 /new_u02/oradata/cinms_user07.dbf

```

```
cp /proc/762/fd/286 /new_u02/oradata/cinms_user08.dbf
```

将相应的文件离线。

```
alter database datafile 8 offline;  
alter database datafile 9 offline;  
alter database datafile 10 offline;  
alter database datafile 11 offline;  
alter database datafile 27 offline;  
alter database datafile 28 offline;
```

通过更名（RENAME）的方式对文件进行重定向。

```
alter database rename file  
'/u02/oradata/cinms_user01.dbf' to '/new_u02/oradata/cinms_user01.dbf';  
alter database rename file  
'/u02/oradata/cinms_user02.dbf' to '/new_u02/oradata/cinms_user02.dbf';  
alter database rename file  
'/u02/oradata/cinms_user03.dbf' to '/new_u02/oradata/cinms_user03.dbf';  
alter database rename file  
'/u02/oradata/cinms_user04.dbf' to '/new_u02/oradata/cinms_user04.dbf';  
alter database rename file  
'/u02/oradata/cinms_user07.dbf' to '/new_u02/oradata/cinms_user07.dbf';  
alter database rename file  
'/u02/oradata/cinms_user08.dbf' to '/new_u02/oradata/cinms_user08.dbf';
```

然后执行恢复。

```
recover datafile 8;  
recover datafile 9;  
recover datafile 10;  
recover datafile 11;  
recover datafile 27;  
recover datafile 28;
```

最后将文件 Online 加载。

```
alter database datafile 8 online;  
alter database datafile 9 online;
```

```
alter database datafile 10 online;
alter database datafile 11 online;
alter database datafile 27 online;
alter database datafile 28 online;
```

以下是日志中记录的操作信息。

```
Mon Dec 19 18:17:38 2011
alter database datafile 8 offline
Mon Dec 19 18:17:39 2011
Completed: alter database datafile 8 offline
Mon Dec 19 18:18:04 2011
alter database rename file '/u02/oradata/sxnms_user01.dbf'
to '/new_u02/oradata/sxnms_user01.dbf'
Mon Dec 19 18:18:20 2011
ALTER DATABASE RECOVER datafile 8
Media Recovery Datafile: 8
Media Recovery Start
Starting datafile 8 recovery in thread 1 sequence 14295
Datafile 8: '/new_u02/oradata/sxnms_user01.dbf'
Media Recovery Log
Recovery of Online Redo Log: Thread 1 Group 1 Seq 14295 Reading mem 0
  Mem# 0 errs 0: /u01/oradata/sxnms/redo01.log
Media Recovery Complete
Completed: ALTER DATABASE RECOVER datafile 8

Mon Dec 19 18:19:00 2011
alter database datafile 8 online
Completed: alter database datafile 8 online
```

这两则案例，处境相同，处理相同，结果也大致相同。由于数据库保持在启动状态，因而能够较为快速地恢复出被删除的数据文件，实在是一种幸运。

技术难点

在这两则案例中，还有一个关键的技术点，那就是如何从大量的 fd 中找到需要的文件。在 Oracle 数据库文件的第一个块（文件头块）上，存有数据文件号信息，只要找到这个文件号，就能和数据库建立起对应关系。

通过 BBED 获取文件号信息

通过 Oracle 的 BBED（Block Browse/EDit）工具，可以很容易获得这个信息。在 UNIX、Linux 系统上，需要通过 make 进行简单编译，生成对应的可执行文件。

```
[oracle@hpserver2 ~]$ cd $ORACLE_HOME/rdbms/lib
[oracle@hpserver2 lib]$ make -f ins_rdbms.mk $ORACLE_HOME/rdbms/lib/bbed
Linking BBED utility (bbed)
rm -f /u01/app/oracle/product/10.2.0/db_1/rdbms/lib/bbed
gcc -o
/u01/app/oracle/product/10.2.0/db_1/rdbms/lib/bbed
-L/u01/app/oracle/product/10.2.0/db_1/rdbms/lib/
-L/u01/app/oracle/product/10.2.0/db_1/lib/
-L/u01/app/oracle/product/10.2.0/db_1/lib/stubs/
.....
```

然后就可以通过该工具（密码是 blockedit）来获得文件号。以下是一个示范。

```
[oracle@hpserver2 fd]$ ls
0  10  12  14  16  18  2   21  23  3   5   7   9
1  11  13  15  17  19  20  22  24  4   6   8

[oracle@hpserver2 ~]$ ./bbed
Password: blockedit
BBED: Release 2.0.0.0.0 - Limited Production on Thu Jan 5 16:54:36 2012
Copyright (c) 1982, 2007, Oracle. All rights reserved.
***** !!! For Oracle Internal Use only !!! *****
BBED> set filename '/proc/23330/fd/18'
        FILENAME                /proc/23330/fd/18
BBED> set blocksize 8192
        BLOCKSIZE                8192
```

```

BBED> p kcvfh.kcvfhrfn
ub4 kcvfhrfn                                @368      0x00000001
BBED> set filename '/proc/23330/fd/19'
      FILENAME                               /proc/23330/fd/19
BBED> p kcvfh.kcvfhrfn
ub4 kcvfhrfn                                @368      0x00000002

```

这里的几个数据结构需要说明一下，kcvfh 表示 Kernel Cache recoVery component File Header。前三个字母 kcv 表示是 Oracle 的内核层次代码，是恢复相关的组件，具体内容就是文件头信息。Kcvfhrfn 表示相对文件号。

关于文件头的信息，在数据库层面的展示来自视图 v\$datafile_header，以下是该视图的字段信息。

```
SQL> desc v$datafile_header
```

Name	Null?	Type
FILE#		NUMBER
STATUS		VARCHAR2(7)
ERROR		VARCHAR2(18)
FORMAT		NUMBER
RECOVER		VARCHAR2(3)
FUZZY		VARCHAR2(3)
CREATION_CHANGE#		NUMBER
CREATION_TIME		DATE
TABLESPACE_NAME		VARCHAR2(30)
TS#		NUMBER
RFILE#		NUMBER
RESETLOGS_CHANGE#		NUMBER
RESETLOGS_TIME		DATE
CHECKPOINT_CHANGE#		NUMBER
CHECKPOINT_TIME		DATE
CHECKPOINT_COUNT		NUMBER
BYTES		NUMBER
BLOCKS		NUMBER
NAME		VARCHAR2(513)
SPACE_HEADER		VARCHAR2(40)

LAST_DEALLOC_CHANGE#	VARCHAR2(16)
UNDO_OPT_CURRENT_CHANGE#	VARCHAR2(40)

该视图（v\$fixed_view_definition 视图是可以获得其他视图结构定义的字典）经由 X\$KCVFH 创建，创建语句如下，X\$KCVFH 正是来自文件头的结构体内容。

```
SQL> select view_definition
       2 from v$fixed_view_definition where view_name='GV$DATAFILE_HEADER';
VIEW_DEFINITION
-----
SELECT inst_id,
       hxfil,
       DECODE(hxons, 0, 'OFFLINE', 'ONLINE'),
       DECODE(hxerr, 0, NULL, 1, 'FILE MISSING', 2, 'OFFLINE NORMAL', 3, 'NOT VERIFIED',
4, 'FILE NOT FOUND', 5, 'CANNOT OPEN FILE', 6, 'CANNOT READ HEADER', 7, 'CORRUPT
HEADER', 8, 'WRONG FILE TYPE', 9, 'WRONG DATABASE', 10, 'WRONG FILE NUMBER', 11, 'WRONG
FILE CREATE', 12, 'WRONG FILE CREATE', 16, 'DELAYED OPEN', 14, 'WRONG RESETLOGS',
15, 'OLD CONTROLFILE', 'UNKNOWN ERROR'),
       hxver,
       DECODE(hxnrcv, 0, 'NO', 1, 'YES', NULL),
       DECODE(hxifz, 0, 'NO', 1, 'YES', NULL),
       to_number(fhcrs),
       to_date(fhcrtr, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
       fhtnm,
       fhtsn,
       fhrrfn,
       to_number(fhrls),
       to_date(fhrlc, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
       to_number(fhscn),
       to_date(fhtim, 'MM/DD/RR HH24:MI:SS', 'NLS_CALENDAR=Gregorian'),
       fhpcp,
       fhfsz*fhbsz,
       fhfsz,
       hxfnm,
       DECODE(hxlmdba, 0, NULL, hxlmdba),
```

```

        DECODE(hx1mld_scn, to_number('0'), NULL, hx1mld_scn),
        DECODE(hxuopc_scn, 0, NULL, hxuopc_scn)
FROM x$kcvfh

```

通过 v\$datafile_header 获得的信息来自数据文件头，其中至关重要的有两个：数据文件的创建 SCN 和创建时间。

```

SQL> select file#,creation_change#,creation_time from v$datafile_header;
        FILE# CREATION_CHANGE# CREATION_TIME

```

```

-----
1              7 2012-01-06 23:31:51
2            1957 2012-01-06 23:32:14
3            3103 2012-01-06 23:32:24
4            3500 2012-01-06 23:32:32

```

文件的创建 SCN 和时间还记录在 file\$视图中，file\$中的信息在启动时用于和数据文件头进行比对校验。

```

SQL> select file#,crscnbas,status$,blocks,spare1 from file$;

```

```

        FILE#   CRSCNBAS   STATUS$   BLOCKS   SPARE1
-----
1              7         2      51200   4194306
2            1957         2     102400   8388610
3            3103         2      25600   12582914
4            3500         2      12800   16777218
5         241976         1       1280   20971522

```

在后面的章节我们还会接触到这些信息。

在 BBED 工具中，首先应用到了 PRINT 命令，该命令可以缩写为 p，用于打印显示数据块中的指定数据结构、子结构及偏移量信息等，也可用于打印具体的存储数据行等。前面使用的 p kcvfh.kcvfhrfn 就用于显示相对文件号信息。

关于 BBED 的进一步说明，参见本书附录。

通过 od 命令获得文件号信息

另外一个简单的方式是通过操作系统的 od 命令读出文件的指定位置，获得文件的文件号。

首先使用 BBED 取得一个参照输出。

```
dtdb2_oracle%bbed
Password:
BBED: Release 2.0.0.0.0 - Limited Production on Thu Jan 5 18:05:30 2012
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.
***** !!! For Oracle Internal Use only !!! *****
BBED> set filename '/proc/762/fd/340'
BBED-00307: incorrect blocksize (2048) or truncated file
BBED> set blocksize 8192
      BLOCKSIZE      8192
BBED> set filename '/proc/762/fd/340'
      FILENAME
BBED> p kcvfh.kcvfhfn
ub4 kcvfhfn                @280      0x00000052
BBED> exit
```

注意 Oracle 9i 文件号的偏移量是 280，再加上数据文件头上一个操作系统块，od 跳过 8472 即可获得文件号。以下是 16 进制输出。

```
dtdb2_oracle%od -j 8472 -t x1 340 | head -1
00000000 00 00 00 52 00 00 00 00 00 00 00 00 00 00 00 00
```

以下是 10 进制输出，文件号为 82。

```
dtdb2_oracle%od -j 8472 -t d2 340 | head -1
00000000 000000 00082 000000 000000 000000 000000 000000 000000
```

在 Oracle 10g/11g 中，文件号的偏移量为 368。以下是在 Linux 平台上 od 命令的 10 进制输出，考察的 3 个文件文件号分别为 1、2 和 3。

```
[oracle@hpserver2 fd]$ od -j 8560 -t x1 18 | head -1
0020560 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[oracle@hpserver2 fd]$ od -j 8560 -t d2 18 | head -1
0020560      1      0      0      0      0      0      0      0
[oracle@hpserver2 fd]$ od -j 8560 -t d2 19 | head -1
0020560      2      0      0      0      0      0      0      0
[oracle@hpserver2 fd]$ od -j 8560 -t d2 20 | head -1
```

```
0020560      3      0      0      0      0      0      0      0
```

如果不熟悉 BBED 的用法，像上面这样使用操作系统的 od 命令，也可以非常简单、快速地获得文件号。

除 BBED 外，在 Linux 和 UNIX 上，还可以通过 lsof 工具查看进程的 FD 等信息。举个例子，可以首先选择一个后台进程（如 dbwr 进程），获取其进程号。

```
root@db2 # ps -ef | grep dbw
      root 14348  9859  0 10:15:50 pts/3      0:00 grep dbw
      oracle  762      1  0   Jun 10 ?        202:01 ora_dbw0_nms
```

然后通过 lsof 工具列举进程打开的文件，输出的内容中就包含了 FD 一列。

```
root@db2 # lsof -p 762
COMMAND PID  USER  FD  TYPE          DEVICE    SIZE/OFF      NODE NAME
oracle  762 oracle 259uW VREG          85,8272   414195712 110986 /u01/nms/system01.dbf
oracle  762 oracle 260uW VREG          85,8272  6291464192 110993 /u01/nms/undotbs01.dbf
oracle  762 oracle 277uW VREG          85,8242  1048584192      9
/u04/res/data_res02.dbf
oracle  762 oracle 287uW VREG          118,0   1048584192 192538 /u02/cnnc_user01.dbf
oracle  762 oracle 288uW VREG          118,0   1048584192 192539 /u02/cnnc_user02.dbf
oracle  762 oracle 289uW VREG          118,0   1048584192 192540 /u02/cnnc_user03.dbf
oracle  762 oracle 290uW VREG          85,8252  1048584192     13 /u03/cnnc_user12.dbf
oracle  762 oracle 291uW VREG          85,8252  1048584192     14 /u03/cnnc_user13.dbf
oracle  762 oracle 325uW VREG          118,0   1048584192 192557 / (/dev/dsk/c1t0d0s0)
oracle  762 oracle 326uW VREG          118,0   1048584192 192558 /u02/cnnc_user27.dbf
oracle  762 oracle 327uW VREG          85,8252  1048584192     27 /u03/cnnc_user28.dbf
oracle  762 oracle 328uW VREG          85,8242  1048584192     24 /u04/cnnc_user29.dbf
oracle  762 oracle 329uW VREG          85,8242  1048584192     25 /u04/cnnc30.dbf
oracle  762 oracle 340uW VREG          85,8242  1048584192     26 /u04/oradata/res13.dbf
oracle  762 oracle 341uW VREG          85,8242  1048584192     27 /u04 (/dev/md/dsk/d50)
oracle  762 oracle 342uW VREG          85,8252  1048584192     28
/u03/oradata/idx_res02.dbf
oracle  762 oracle 343uW VREG          85,8272  1409294336 111241 /u01 (/dev/md/dsk/d80)
oracle  762 oracle 345uW VREG          118,0   1073750016 192560
/u02/cnnc_indexes11.dbf
oracle  762 oracle 346uW VREG          118,0   1073750016 192561
```

```
/u02/cnnc_indexes12.dbf
```

通过 proc 下的文件目录，可以找到文件句柄信息，这就是 Oracle 的数据文件。

```
root@db2 # ls -l /proc/762/fd/329
```

```
-rw-r----- 1 oracle dba 1048584192 Dec 20 10:16 /proc/762/fd/329
```

可以通过复制来恢复文件。以下是 Linux 上的一个测试范例，误删除的文件显示为 **deleted**。

```
$ cd /proc/2879/fd
```

```
$ ls -l
```

```
lr-x----- 1 oracle dba 64 Dec 19 21:50 12 ->
```

```
/oracle/10.2.0/db_1/rdbms/mesg/oraus.msb
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 13 -> /oracle/10.2.0/db_1/dbs/hc_orcl.dat
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 14 -> /oracle/10.2.0/db_1/dbs/lkORCL
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 15 ->
```

```
/oradata/controlfile/o1_mf_555wq3ng_.ctl
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 16 ->
```

```
/oradata/datafile/o1_mf_system_555wqbnk_.dbf
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 17 -> /oradata/datafile/o1_mf_undotbs1.dbf
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 18 -> /oradata/datafile/sysaux_555wr.dbf
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 19 -> /oradata/datafile/users.dbf (deleted)
```

```
lr-x----- 1 oracle dba 64 Dec 19 21:50 2 -> /dev/null
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 20 ->
```

```
/oradata/datafile/o1_mf_temp_555wrbnz_.tmp
```

```
lr-x----- 1 oracle dba 64 Dec 19 21:50 21 ->
```

```
/oracle/10.2.0/db_1/rdbms/mesg/oraus.msb
```

```
l-wx----- 1 oracle dba 64 Dec 19 21:50 5 -> /admin/udump/orcl_ora_2871.trc
```

```
l-wx----- 1 oracle dba 64 Dec 19 21:50 6 -> /admin/bdump/alert_orcl.log
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 7 -> /oracle/10.2.0/db_1/dbs/lkinstorcl  
(deleted)
```

```
l-wx----- 1 oracle dba 64 Dec 19 21:50 8 -> /admin/bdump/alert_orcl.log
```

```
lrwx----- 1 oracle dba 64 Dec 19 21:50 9 -> /oracle/10.2.0/db_1/dbs/hc_orcl.dat
```

lsf 是一个非常直观和方便的工具，在很多情况下可以给我们提供直接的帮助。

以拯救之因

强制恢复导致 ORA-600 4000 错误案例

我曾在一本书上写道：

一知半解比无知更可怕。

一知半解的草率行事可能使数据库遭受意想不到的灾难，所以在接触到一个数据库时，如果没有相当的把握，请谨慎操作，不要让自己和数据库陷入到未知的危险境地。最基本的，如果不可避免地要进行某些危险的维护性操作，应当在之前做好备份。

在面对数据时，无知者不能无畏。

灾难描述

2011 年 12 月 30 日，一个运营商客户的核心数据库发生故障，无法启动。随后在工程师的草率介入后，数据库遭遇无法启动的灾难。

整个过程是这样的。

1. 客户误操作删除了一个数据文件，数据库报错。
2. 第三方服务商介入，试图清除该缺失文件。
3. 工程师选择重建控制文件，在当前数据库下执行不完全恢复。
4. 通过内部隐含参数强制重置日志打开数据库。
5. 数据库出现一系列 600 内部错误，无法启动。
6. 检查备份，发现近期末进行备份，无从备份恢复。
7. 灾难形成。

这则案例原本并不复杂，丢失了一个数据文件，如果没有备份，可以进行下列操作。

1. 在关闭数据库之前，类似前一节的情况，从文件描述符中进行恢复。
2. 如果数据库关闭不可恢复，并且可以接受损失，则离线抛弃该文件即可。
3. 如果该文件非常重要，可以通过存储级别的恢复，找回该文件。

具体选择何种措施，取决于数据的重要程度，但是贸然采取行动则会消灭一些可能性。比如，关闭数据库，措施 1 就无效了；如果在存储级别又执行了文件复制，覆盖了存储内容，则措施 3 也就无效了。

所以，针对不同的数据灾难，做出正确的判断，找到合适的技术支持尤为重要。而对于这个案例，以上三种可能性都被掩盖，数据库走向了一个错误的方向，并且在错误的方向上走得很远。

案例警示

分析这个案例的整个过程，我们总结出了如下教训。

1. 一知半解比无知更可怕。

在这个案例中，技术人员显然对 Oracle 的技术认知不足，草率地采取了错误的手段和步骤，最终导致数据库状况和用户初衷相差了十万八千里。

前面提到的三种可能性在一系列的误操作面前变得无效，数据库必须接受不一致的灾难性考验。

通过这个案例，技术人员需要铭记，在着手处理故障之前，要遵循这样一个重要的守则：**保护现场，至少不要使情况变得更坏。**

在保护好现场之后，再进行破坏性或者把握性不大的恢复尝试；对于某些海量数据的情况，如果无法进行及时的备份，不得不在当前环境下进行恢复，那么就必须要要求工程师具有极高的素质和精准的判断，明确知道每一个命令和步骤可能带来的后果和后续的处理方式。

每个工程师都要想一想：如果一个恢复尝试之后，数据库彻底无法启动，我们还能如何做？多思多想是对用户和自己负责，无知无畏不应当在生产环境中尝试。

2. 不要超越自己的能力范围。

在技术领域，如果你采用的技术手段和方法超越了自己的能力范围，可能出现不可预知的后果，那么最好不要做这样的冒险，冒险意味着对用户的不负责任，同时也是对自己的折磨。如果决定冒险，那么至少在自己的测试环境中进行同类测试，明确可能会出现的情况，这样的测试并不会耗费太多的时间。

在这个案例中，Oracle 数据库的内部参数使用是不恰当的，导致了问题变得更加糟糕。实际上，内部参数的使用要求非常清晰地了解其含义，以及可能引致的后果，并且对随之而来的后果有应对方案。

对于用户，应当能够对第三方服务商进行适当的监督、确认，确保不可预期的后果不被贸然引入到数据库中。

3. 在不可逆操作之前执行备份。

在需要执行不可逆操作之前，执行备份，要确保可以回退到之前的状态，以便尝试恢复失败之后，别人还有机会从头开始，这也是保护现场的概念。

除了对数据库的备份，实际上还应当在一些修改操作之前执行备份。比如对于特定数据块、文件头、ASM 磁盘组信息、日志文件等的备份，这些备份可以在关键时候帮助我们。

这里必须着重提一下日志文件，因为在强制 `Resetlogs` 时，日志文件会被清空刷新，如果不进行备份，其中的内容将永远丢失，而我们知道，有时候通过日志解析可以最大限度地找回数据，所以日志文件的备份也非常重要。

4. 管理者需要参与决策。

对于重要的数据环境，在执行重要的操作之前，管理者即便不了解详细的技术细节，也应当和技术人员进行沟通，听取技术方案、操作计划、实施步骤、现场保全、回退方案等。

管理者虽然可能不了解技术细节，但是其大局观和缜密思考应当为技术决策提供保障。管理者也应当在交流中感知技术人员是否了解详细情况，是否对方案有清晰把握、对执行自信无误。交流、提问和质疑也是对技术人员完善方案、认真思考的一种促进。

有了这样一个环节，就可以规避很多风险，而管理者的判断和决策也应当成为数据管理中重要的组成部分。

保护数据，保护现场，在处理故障时认真思考，谨慎决策，是用户和工程师们共同的职责。只有大家共同努力才能保障数据环境的持久安全。

技术回放

后来我们得知，这个数据库系统的数据量大约为 600GB，是个极其重要的业务系统，其近期备份也不能用于恢复。

```
SQL> select sum(bytes)/1024/1024/1024 from v$datafile;
SUM(BYTES)/1024/1024/1024
-----
617.027023
```

以下是从日志中获得的一些操作信息，从中可以看到第三方工程师的恢复操作。

首先尝试的是使用备份控制文件执行不完全恢复，恢复没有成功，未能打开数据库，日志信息如下，最后

一个文件号是 180，意味着数据库中有 180 个数据文件。

```
Fri Dec 30 14:10:12 2011
ALTER DATABASE RECOVER database using backup controlfile until cancel
Fri Dec 30 14:10:12 2011
Media Recovery Start
WARNING! Recovering data file 1 from a fuzzy file. If not the current file
it might be an online backup taken without entering the begin backup command.
.....
WARNING! Recovering data file 180 from a fuzzy file. If not the current file
it might be an online backup taken without entering the begin backup command.
parallel recovery started with 11 processes
ORA-279 signalled during:
ALTER DATABASE RECOVER database using backup controlfile until cancel ...
Fri Dec 30 14:10:20 2011
ALTER DATABASE RECOVER CONTINUE DEFAULT
Fri Dec 30 14:10:20 2011
Media Recovery Log /archive/DMA/archivelog/2011_12_30/o1_mf_1_53868_%u_.arc
Errors with log /archive/DMA/archivelog/2011_12_30/o1_mf_1_53868_%u_.arc
ORA-308 signalled during: ALTER DATABASE RECOVER CONTINUE DEFAULT ...
```

这里的 ORA-308 错误是指无法找到需要的归档日志用于恢复。

```
Fri Dec 30 14:10:20 2011
ALTER DATABASE RECOVER CANCEL
ORA-1547 signalled during: ALTER DATABASE RECOVER CANCEL ...
Fri Dec 30 14:10:59 2011
ALTER DATABASE OPEN RESETLOGS
Fri Dec 30 14:10:59 2011
ORA-1194 signalled during: ALTER DATABASE OPEN RESETLOGS...
```

这里的 ORA-1194 是指文件不一致仍需恢复，所以无法使用 Resetlogs 方式打开数据库。通常这里会报出第一个 SYSTEM 文件需要进一步恢复。

在接下来的实例启动中，一个重要的隐含参数（_allow_resetlogs_corruption）被加入进来，用于强制打开数据库，这个参数的使用是有一系列后续影响的。

ksdpec: called for event 13740 prior to event group initialization

Starting up ORACLE RDBMS Version: 10.2.0.1.0.

System parameters with non-default values:

```

processes                = 500
sessions                 = 885
__shared_pool_size       = 1811939328
__large_pool_size        = 16777216
__java_pool_size         = 16777216
__streams_pool_size      = 33554432
nls_language             = SIMPLIFIED CHINESE
nls_territory            = CHINA
sga_target               = 13639876608
db_block_size            = 8192
__db_cache_size          = 11744051200
compatible               = 10.2.0.1.0
db_file_multiblock_read_count= 16
db_recovery_file_dest_size= 1572864000000
__allow_resetlogs_corruption= TRUE
undo_management          = AUTO
undo_tablespace          = UNDOTBS1
remote_login_passwordfile= EXCLUSIVE
db_domain                =
job_queue_processes      = 10
db_name                  = dma
open_cursors             = 300
pga_aggregate_target     = 2552233984

```

PMON started with pid=2, OS id=4487

设置后，数据库可以抛弃一致性，强制执行日志刷新，尝试打开数据库。当用户再次强制打开数据库时，遇到 ORA-600 4000 号错误，日志如下所示。

Fri Dec 30 17:54:22 2011

SMON: enabling cache recovery

Fri Dec 30 17:54:22 2011

Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_9074.trc:

```

ORA-00600: internal error code, arguments: [4000], [28], [], [], [], [], [], []
Fri Dec 30 17:54:27 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_9074.trc:
ORA-00704: 引导程序进程失败
ORA-00704: 引导程序进程失败
ORA-00600: internal error code, arguments: [4000], [28], [], [], [], [], [], []

```

然后工程师重建了控制文件。

```

Fri Dec 30 14:22:34 2011
CREATE CONTROLFILE REUSE DATABASE "DMA" RESETLOGS NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 5
    MAXDATAFILES 1024
    MAXINSTANCES 8
    MAXLOGHISTORY 1168
.....

```

当然，这样的步骤仍然解决不了问题，于是工程师又加入 ROLLBACK 参数，强制将 28 号回滚段离线标记为损坏。

```

    _allow_resetlogs_corruption= TRUE
    _offline_rollback_segments= (_SYSSMU28$)
    _corrupted_rollback_segments= (_SYSSMU28$)

```

这些尝试无助于故障的解决，最终的错误停留在 ORA-600 4000 上，其中第二个参数 28 的确代表的是回滚段号，也就是在 28 号回滚段上存在不一致事务。

```

Fri Dec 30 19:22:18 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_10809.trc:
ORA-00600: 内部错误代码, 参数: [4000], [28], [], [], [], [], [], []
Fri Dec 30 19:22:22 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_10809.trc:
ORA-00704: 引导程序进程失败
ORA-00704: 引导程序进程失败
ORA-00600: 内部错误代码, 参数: [4000], [28], [], [], [], [], [], []

```

该数据库在重建控制文件后显示共有 180 个数据库文件，以下是查询 v\$datafile 视图后的末端输出。

NAME

```
-----
/data2/userdata/d_wireless_dbs_2.dbf
/data2/userdata/d_wireless_dbs_18.dbf
/data2/userdata/d_wireless_dbs_17.dbf
/data2/userdata/d_wireless_dbs_15.dbf
```

已选择 180 行。

数据库最后完成的检查点如下。

```
SQL> col CHECKPOINT_CHANGE# for 999999999999999999
SQL> select file#,checkpoint_change# from v$datafile where rownum <10;
      FILE# CHECKPOINT_CHANGE#
-----
1      12506717382696
2      12506717382696
3      12506717382696
4      12506717382696
5      12506717382696
6      12506717382696
7      12506717382696
8      12506717382696
9      12506717382696
```

已选择 9 行。

这个案例原本不应该走到这个地步，但由于工程师对于技术判断的一知半解，无知无畏，最终将数据库带入了一个灾难的境地。而且工程师处理之前未进行备份，无法从头开始，因此只能从这个 ORA-600 的错误状态上进行进一步的分析处理。

恢复过程

ORA-600 的 4000 号错误是指，在数据库启动检测中，发现某些数据块的 SCN(具体与 CSC 和 Commit SCN

有关) 大于系统当前的 SCN, 违反了数据库的一致性, 出现内部错误异常。

Oracle 对这个错误的描述如下。

This has the potential to be a very serious error. It means that Oracle has tried to find an undo segment number in the dictionary cache and failed.

这个错误和潜在的严重错误有关, 它意味着 Oracle 试图在字典缓存中查找一个回滚段号, 但是失败了。实际上也就是在执行回退时读取回滚段遇到了错误。

ORA-600 4000 错误揭秘

我们可以根据出现错误时抛出的跟踪文件进行进一步的分析。

```
*** 2011-12-30 16:14:02.081
ksedmp: internal or fatal error
ORA-00600: 内部错误代码, 参数: [4000], [28], [], [], [], [], [], []
Current SQL statement for this session:
select ctime, mtime, stime from obj$ where obj# = :1
----- Call Stack Trace -----
calling      call      entry      argument values in hex
location     type      point      (? means dubious value)
-----
ksedst()+64   call      _etext_f()+23058430 000000000 ? 000000001 ?
              09017233136
ksedmp()+1680 call      _etext_f()+23058430 000000000 ?
              09017233136              C0000000000000D20 ?
              400000000052C8DF0 ?
              000000000 ? 000000000 ?
              000000000 ?
ksfdmp()+48   call      _etext_f()+23058430 000000003 ?
              09017233136
```

错误出现在对于 OBJ\$ 的访问上, OBJ\$ 是数据库初始化启动时必须访问的一张数据表。在跟踪文件中还包含存在不一致事务的数据块信息, 以下是第一个抛出异常的数据块。

注意这里的两个内容。

1. XID: 0x001c.01d.00069f98。

这里 XID 的第一部分为 UNDO 段号，1C 即 28，也就是 ORA-600 4000 错误抛出的第一个参数。

2. CSC: 最后一次完成的块清除 SCN。

CSC 即 **SCN of the last block cleanout**，这个 SCN 是 0xb5f.f28cc1fd，即 12 506 719 109 629。注意这个 SCN 大于前面查询到的数据文件最后 SCN（12 506 717 382 696），两者相差 1 726 933。

这两者就是“ORA-00600: 内部错误代码，参数: [4000], [28]……”的成因。

Block header dump: 0x0040007a

Object id on Block? Y

seg/obj: 0x12 csc: 0xb5f.f28cc1fd itc: 1 flg: - typ: 1 - DATA

fsl: 0 fnx: 0x0 ver: 0x01

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x001c.01d.00069f98	0x008092df.75bc.07	--U-	1	fsc 0x0000.f28cc1fe

data_block_dump,data header at 0xc00000025d526044

=====

tsiz: 0x1fb8

hsiz: 0xea

pbl: 0xc00000025d526044

bdba: 0x0040007a

76543210

flag=-----

ntab=1

nrow=108

frre=-1

fsbo=0xea

fseo=0x40c

avsp=0x368

tosp=0x368

0xe:pti[0] nrow=108 offs=0

0x12:pri[0] offs=0x1f7c

```

0x14:pri[1]      offs=0x1f3c
0x16:pri[2]      offs=0x1efb
0x18:pri[3]      offs=0x1ebc
0x1a:pri[4]      offs=0x1e7c
0x1c:pri[5]      offs=0x1e3c
0x1e:pri[6]      offs=0x1e00
0x20:pri[7]      offs=0x1dbe

```

这个 ITL 事务槽锁定了一行记录，可以从后面的转储中找到这条记录，第 30 行（块内行记录由 0 开始）。

```

tab 0, row 29, @0x40c
tl: 71 fb: --H-FL-- lb: 0x1  cc: 17
col 0: [ 2]  c1 02                      --这是 1
col 1: [ 5]  c4 02 4d 1e 2f
col 2: [ 1]  80                          --这是 0
col 3: [12]  5f 4e 45 58 54 5f 4f 42 4a 45 43 54
col 4: [ 2]  c1 02
col 5: *NULL*
col 6: [ 1]  80
col 7: [ 7]  78 6a 01 05 0b 0f 11        --这三个字段都是时间，长度为 7
col 8: [ 7]  78 6f 0c 1e 0c 17 2a
col 9: [ 7]  78 6a 01 05 0b 0f 11
col 10: [ 1]  80
col 11: *NULL*
col 12: *NULL*
col 13: [ 1]  80
col 14: *NULL*
col 15: [ 1]  80
col 16: [ 4]  c3 07 38 24

```

那么这行记录是什么数据呢？

通过下列函数可以对字符型数据做一个简单的转换，帮助进一步分析。

```

create or replace function hextostr(hexstr varchar2) return varchar2 is
    i    number;
    s    char;

```

```

str1 varchar2(20);
rst  varchar2(200);
begin
  i   := 1;
  rst := '';
  Loop
    str1 := substr(replace(hexstr, ' '), i, 2);
    select chr(to_number(str1, 'xxxxxxxx')) into s from dual;
    rst := rst || s;
    exit when i > lengthb(hexstr);
    i := i + 2;
  end loop;
  return rst;
end;
/

```

经过转换可以得出，这个记录行是_NEXT_OBJECT。

```
SQL> select hextostr('5f 4e 45 58 54 5f 4f 42 4a 45 43 54') colname from dual;
```

COLNAME

```

-----
_NEXT_OBJECT

```

以下通过杨廷琨提供的一个函数（参见本书附录）进行全面的数据类型转换。

```
SQL> select F_GET_FROM_DUMP(replace('5f 4e 45 58 54 5f 4f 42 4a 45 43 54',' ',''), 'VARCHAR2')
```

```
OUTPUT from dual;
```

OUTPUT

```

-----
_NEXT_OBJECT

```

```
SQL> select F_GET_FROM_DUMP(replace('78 6a 01 05 0b 0f 11 ',' ',''), 'DATE') OUTPUT from dual;
```

OUTPUT

```

-----
2006-1-5 10:14:16

```

```
SQL> select F_GET_FROM_DUMP(replace('c4 02 4d 1e 2f',' ',''), 'NUMBER') OUTPUT from dual;
```

OUTPUT

1762946

注意这里的NEXT_OBJECT 是一个数据库中隐藏的特殊对象，实际上是为了生成下一个对象的 object_id 或 dba_object_id 而设定的，在数据库中的对象被 Truncate 或者生成一个新的对象时使用。所以这个对象上常常会持有锁定，如果数据库异常，其锁定通常难以正常释放。

对比以下信息，可以确认这条记录的正确性。

SQL> desc obj\$

Name	Null?	Type
-----	-----	-----
OBJ#	NOT NULL	NUMBER
DATAOBJ#		NUMBER
OWNER#	NOT NULL	NUMBER
NAME	NOT NULL	
VARCHAR2(30)		
NAMESPACE	NOT NULL	NUMBER
SUBNAME		
VARCHAR2(30)		
TYPE#	NOT NULL	NUMBER
CTIME	NOT NULL	DATE
MTIME	NOT NULL	DATE
STIME	NOT NULL	DATE
STATUS	NOT NULL	NUMBER
REMOTEOWNER		
VARCHAR2(30)		
LINKNAME		
VARCHAR2(128)		
FLAGS		NUMBER
OID\$		RAW(16)
SPARE1		NUMBER
SPARE2		NUMBER
SPARE3		NUMBER

```
SPARE4
VARCHAR2(1000)
SPARE5
VARCHAR2(1000)
SPARE6
DATE
SQL> select obj#,dataobj#,owner#,name from obj$ where name='_NEXT_OBJECT';
      OBJ#      DATAOBJ#      OWNER# NAME
-----
      1         198226          0 _NEXT_OBJECT
```

检查整个跟踪文件，可以发现 0xb5f.f28cc1fd 是最大的一个 CSC 号。

```
eygle$ grep seg/obj dma_ora_6991.trc
seg/obj: 0x12  csc: 0xb5f.f28cc1fd  itc: 1  flg: -  typ: 1 - DATA
seg/obj: 0x38  csc: 0x00.141  itc: 1  flg: 0  typ: 1 - DATA
seg/obj: 0x24  csc: 0xb57.571f621  itc: 1  flg: -  typ: 2 - INDEX
seg/obj: 0x24  csc: 0xb5a.c4e1f05b  itc: 1  flg: -  typ: 2 - INDEX
seg/obj: 0x24  csc: 0x00.279a  itc: 2  flg: -  typ: 2 - INDEX
seg/obj: 0x12  csc: 0xb5f.f28cc1fd  itc: 1  flg: -  typ: 1 - DATA
```

由于是通过这个 CSC 号进行回滚段的分析，因而这个非法的 SCN 号导致了最终故障。

找到这个原因之后，我们只需要将数据库的 SCN 整体向前推进消除掉这个 SCN 差异，数据库就会通过延迟提交清除这个事务信息，错误即可解除。

通过_minimum_giga_scn 消除 SCN 异常

在 Oracle 10g 中，可以通过隐含参数_minimum_giga_scn 来推进 SCN，每个 1 将 SCN 推进到 1G。

```
SQL> select ksppinm,ksppdesc from x$ksppi
      2  where ksppinm like '%giga%'
      3  /
KSPPINM                                KSPPDESC
-----
_minimum_giga_scn                      Minimum SCN to start with in 2^30 units
```

以下将 SCN 0xb5f.f28cc1fd 进行一个转换运算，获得十进制值，再转换成以_minimum_giga_scn 为单位。

```

SQL> col scn for 9999999999999999
SQL> select to_number('b5ff28cc1fd','xxxxxxxxxxx') scn from dual;
          SCN
-----
12506719109629
SQL> select 12506719109629/1024/1024/1024 giga from dual;
          GIGA
-----
11647.7898

```

对于这个数据库，将该参数设置为 11649，稍微多推进一点，数据库的错误就可以被成功消除了。我们首先在初始化参数文件中增加如下两个参数。

```

*._allow_resetlogs_corruption=true
*._minimum_giga_scn=11649

```

然后尝试打开数据库。

```

SQL> startup mount pfile=initdma_eygle.ora
ORACLE 例程已经启动。

```

```

Total System Global Area 1.3640E+10 bytes
Fixed Size                  2115392 bytes
Variable Size              1889450176 bytes
Database Buffers           1.1744E+10 bytes
Redo Buffers                4259840 bytes

```

数据库装载完毕。

```

SQL> recover database using backup controlfile until cancel;
ORA-00279: 更改 12506717382696 (在 12/30/2011 19:22:18 生成) 对于线程 1 是必需的
ORA-00289: 建议:/archive/DMA/archivelog/2011_12_31/o1_mf_1_5_%u_.arc
ORA-00280: 更改 12506717382696 (用于线程 1) 在序列 #5 中

```

```

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

```

ORA-01547: 警告: RECOVER 成功但 OPEN RESETLOGS 将出现如下错误

ORA-01194: 文件 1 需要更多的恢复来保持一致性

ORA-01110: 数据文件 1: '/dma_sys/sysdata/dma/system01.dbf'

ORA-01112: 未启动介质恢复

SQL> alter database open resetlogs;

alter database open resetlogs

*

第 1 行出现错误:

ORA-00603: ORACLE 服务器会话因致命错误而终止

此时观察告警日志文件，可以发现 4000 错误已经不再出现，但是由于 UNDO 的 4194 错误导致数据库再次崩溃。4194 错误的解决就容易很多了。

在以上尝试启动时，告警日志的提示如下。

Sat Dec 31 16:06:30 2011

alter database open resetlogs

Sat Dec 31 16:06:30 2011

RESETLOGS is being done without consistency checks. This may result in a corrupted database. The database should be recreated.

RESETLOGS after incomplete recovery UNTIL CHANGE 12506717382696

Resetting resetlogs activation ID 984777556 (0x3ab28354)

Sat Dec 31 16:07:12 2011

Setting recovery target incarnation to 3

Sat Dec 31 16:07:12 2011

Advancing SCN to 12508018507776 according to _minimum_giga_scn

注意以上日志提示，因为 _minimum_giga_scn 参数的设置，将 SCN 增进了 12508018507776。

继续看打开数据库接下来的日志信息。

Sat Dec 31 16:07:12 2011

Assigning activation ID 984841183 (0x3ab37bdf)

Thread 1 opened at log sequence 1

Current log# 4 seq# 1 mem# 0: /dma_sys/sysdata/dma/redo04a.log

Current log# 4 seq# 1 mem# 1: /dma_sys/sysdata/dma/redo04b.log

Successful open of redo thread 1

```

Sat Dec 31 16:07:12 2011
MTTR advisory is disabled because FAST_START_MTTR_TARGET is not set
Sat Dec 31 16:07:12 2011
SMON: enabling cache recovery
Sat Dec 31 16:07:13 2011
Successfully onlined Undo Tablespace 1.
Dictionary check beginning
Tablespace 'TEMP' #3 found in data dictionary, but not in the controlfile. Adding to
controlfile.
File #181 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00181' in the controlfile.
This file can no longer be recovered so it must be dropped.
Dictionary check complete

```

注意这里提示说 181 号文件在数据字典（file\$）中存在，但是在控制文件中不存在，现在被添加回控制文件，命名为 MISSING00181。这也告诉我们，试图通过重建控制文件以去掉某个文件的做法是根本错误的。

这里的 181 号文件就是被误操作删除掉的文件。提示也告知用户，当数据库被 resetlogs 打开，这个文件将不能够再通过恢复加入到数据库中，只能被删除。

接下来的下列提示说应当添加临时表空间。

```

*****
Sat Dec 31 16:07:13 2011
SMON: enabling tx recovery
WARNING: The following temporary tablespaces contain no files.
    This condition can occur when a backup controlfile has
    been restored. It may be necessary to add files to these
    tablespaces. That can be done using the SQL statement:

    ALTER TABLESPACE <tablespace_name> ADD TEMPFILE

Alternatively, if these temporary tablespaces are no longer
needed, then they can be dropped.
Sat Dec 31 16:07:13 2011
    Empty temporary tablespace: TEMP

```

```
*****
```

```
Database Characterset is ZHS16GBK
```

ORA-600 4194 错误 UNDO 故障消除

接下来是因为一致性被破坏而出现的 ORA-600 4194 数据库内部错误，这与 UNDO 回滚有关。

```
Sat Dec 31 16:07:14 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_4379.trc:
ORA-00600: 内部错误代码, 参数: [4194], [42], [30], [], [], [], [], []
Doing block recovery for file 2 block 16237
Block recovery from logseq 1, block 111 to scn 12508018507983
Sat Dec 31 16:07:18 2011
Recovery of Online Redo Log: Thread 1 Group 4 Seq 1 Reading mem 0
  Mem# 0 errs 0: /dma_sys/sysdata/dma/redo04a.log
  Mem# 1 errs 0: /dma_sys/sysdata/dma/redo04b.log
Block recovery stopped at EOT rba 1.1008.16
Block recovery completed at rba 1.1008.16, scn 2912.1073742029
Doing block recovery for file 2 block 441
Block recovery from logseq 1, block 111 to scn 12508018507979
Sat Dec 31 16:07:18 2011
Recovery of Online Redo Log: Thread 1 Group 4 Seq 1 Reading mem 0
  Mem# 0 errs 0: /dma_sys/sysdata/dma/redo04a.log
  Mem# 1 errs 0: /dma_sys/sysdata/dma/redo04b.log
Block recovery completed at rba 1.112.16, scn 2912.1073742028
```

以上日志提示了恢复到达的 SCN 位置和 RBA 信息。

以下提示说明在恢复文件 2 block 45224 时出现了错误，这是对于 UNDO 的恢复尝试。

```
Sat Dec 31 16:08:25 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_4379.trc:
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
Sat Dec 31 16:08:27 2011
DEBUG: Replaying xcb 0xc00000046193d230, pmd 0xc0000000a79a88d0 for failed op 8
Doing block recovery for file 2 block 45224
```

```

No block recovery was needed
Sat Dec 31 16:09:35 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_4379.trc:
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
Sat Dec 31 16:09:37 2011
DEBUG: Replaying xcb 0xc00000046193d230, pmd 0xc0000000a79a88d0 for failed op 8
Sat Dec 31 16:09:37 2011
Errors in file /opt/app/oracle/admin/dma/udump/dma_ora_4379.trc:
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
Doing block recovery for file 2 block 45224
No block recovery was needed
Sat Dec 31 16:10:44 2011
Errors in file /opt/app/oracle/admin/dma/bdump/dma_smon_4365.trc:
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
Sat Dec 31 16:10:46 2011
DEBUG: Replaying xcb 0xc00000046193d230, pmd 0xc0000000a79a88d0 for failed op 8
Doing block recovery for file 2 block 45224
No block recovery was needed
Sat Dec 31 16:10:46 2011
Fatal internal error happened while SMON was doing active transaction recovery.
Sat Dec 31 16:10:46 2011
Errors in file /opt/app/oracle/admin/dma/bdump/dma_smon_4365.trc:
ORA-00600: 内部错误代码, 参数: [4194], [66], [60], [], [], [], [], []
SMON: terminating instance due to error 474
Instance terminated by SMON, pid = 4365

```

最后数据库因为 4914 错误实例崩溃。通过内部参数强制打开数据库，就破坏了数据库的一致性，不可避免地会遇到一系列的内部错误。

ORA-600 [4194]错误的官方解释是：“Undo Record Number Mismatch While Adding Undo Record”，当数据库通过 REDO 恢复来增加 UNDO 记录时，发现 UNDO 记录的号码不匹配，也就是出现了不一致。

这可以通过重建 UNDO 表空间来解决。设置如下初始化参数。

```
*.undo_management='MANUAL'
```

然后启动数据库重建 UNDO 表空间。

```
SQL> startup pfile=initdma_eygle.ora
```

ORACLE 例程已经启动。

```
Total System Global Area 1.3640E+10 bytes
```

```
Fixed Size                2115392 bytes
```

```
Variable Size            1889450176 bytes
```

```
Database Buffers        1.1744E+10 bytes
```

```
Redo Buffers            4259840 bytes
```

数据库装载完毕。

数据库已经打开。

```
SQL> drop tablespace undotbs1;
```

表空间已删除。

```
SQL> create undo tablespace undotbs1
```

```
2 datafile '/dma_sys/sysdata/dma/undotbs202.dbf' size 500M;
```

表空间已创建。

在这之后可以修改 undo_management 参数。

```
*.undo_management='AUTO'
```

```
*.undo_tablespace='UNDOTBS1'
```

最后重新启动数据库，此时数据库通常就可以正常无误地启动。

```
[oracle] % sqlplus "/ as sysdba"
```

```
SQL*Plus: Release 10.2.0.1.0 - Production on 星期六 12月 31 16:41:01 2011
```

```
Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

已连接到空闲例程。

```
SQL> startup
```

ORACLE 例程已经启动。

```
Total System Global Area 1.3640E+10 bytes
```

```
Fixed Size                2115392 bytes
```

```
Variable Size          8264792256 bytes
Database Buffers      5368709120 bytes
Redo Buffers          4259840 bytes
```

数据库装载完毕。

数据库已经打开。

```
SQL> ! ls -l /dma_sys/sysdata/dma/ |grep temp
```

```
-rw-r----- 1 oracle oinstall 16106135552 12月31日 16:41 perftemp01.dbf
-rw-r----- 1 oracle oinstall 16106135552 12月30日 11:41 temp01.dbf
```

```
SQL> ALTER TABLESPACE temp ADD TEMPFILE '/dma_sys/sysdata/dma/temp02.dbf' size 16G ;
```

表空间已更改。

```
SQL> create table test as select * from obj$;
```

表已创建。

```
SQL> drop table test purge;
```

表已删除。

```
SQL> alter database datafile '/dma_sys/sysdata/dma/undotbs202.dbf'
```

```
2 autoextend on next 200m maxsize 16G;
```

数据库已更改。

随后查询得到的文件名列表大致如下（这里的 MISSING00181 即丢失的文件，数据库默认以 MISSING 命名）。

NAME

```
-----
/data2/userdata/d_wireless_dbs_2.dbf
/data2/userdata/d_wireless_dbs_18.dbf
/data2/userdata/d_wireless_dbs_17.dbf
/data2/userdata/d_wireless_dbs_15.dbf
/opt/app/oracle/product/10.2.0/dbs/MISSING00181
```

已选择 181 行。

这样就最终完成了恢复。在通过这些手段恢复之后，由于数据库的一致性和完整性被破坏，因此通常建议导出数据，重建数据库。在消除 4194 错误之后，数据库还可能遇到一些其他的 ORA-600 错误，不过这些错误通常都是可以解决和消除的。

以优化之名

存储优化导致表空间误删除案例

一人蛇先成，引酒且饮之，乃左手持卮，右手画蛇曰：“吾能为之足！”为蛇足者，终亡其酒。

——《战国策·齐二》

嘉招欲覆杯中渌，丽唱仍添锦上花。

——宋·王安石《即事》

宁锦上添花，勿画蛇添足。

在很多数据灾难中，我们看到很多原本是可有可无的操作，或者是锦上添花的工作，最后却由于处置不当、准备不足或认知不够，导致了数据灾难，当事人后悔不迭。

所谓数据无小事，在和数据打交道时，我们一定要谨而慎之，切勿一失足而至追悔莫及。

灾难描述

2011 年 12 月 31 日，我们接到一个紧急的数据救援请求。据粗略了解，事情大致是这样的。

1. 年底，开发商帮助用户进行数据库性能优化。
2. 优化内容之一是存储优化。
3. 存储优化的方法是进行表空间重建。
4. 开放商先删除表空间（成功），然后重建表空间（失败）。
5. 用户发现删除的表空间中的数据未备份。
6. 灾难形成。

对于这则案例，如果数据库不进行优化，那么稳定运行的问题还是不大的。然而，准备不足的这些维护操作，则使数据库彻底陷入了瘫痪的困境。

同样在 2011 年底，我们还遇到了另外一则完全类似的案例，情形大致如下。

1. 客户数据库的安全性要求极高。

2. 客户要定期接受上级单位的例行检查。
3. 为了满足安全评估要求，客户决定对系统进行全新重构。
4. 在导出备份之后，将主机格式化并重建数据库。
5. 在重建后恢复时发现，导出的备份文件有错误，无法导入数据库中。
6. 灾难形成。

现场分析发现，导出文件大小是正确的，但是通过十六进制模式查看发现文件末尾全是空白，没有数据，推测可能是通过移动硬盘备份时出现了问题，未正常退出或插拔移动硬盘导致数据损失。这个导出文件是不足以用来恢复了。后来通过存储级别的恢复，在格式化后的硬盘上找到了之前的历史备份，最终总算恢复了大部分数据。

有时一个好的设想因为执行不当可能成为灾难的源头，所以在完成了基本的保障之后，无为而治在有些情况下却能保证数据库的安全稳定运行，折腾是数据灾难之源。

案例警示

这两个数据灾难带给了我们如下教训。

1. 在任何破坏性操作之前，必须严格验证备份的有效性。

这里所说的破坏性操作包括删除表空间、数据文件、数据表等。

严格验证备份的有效性，如果可能，应该进一步演进到，**操作之前进行全面有效的备份**。有时候不要相信别人传达的诸如已经备份、存在备份、有人备份之类的信息，因为不同方式的备份可能不适合你的操作恢复，别人的备份也可能存在你所不知道的问题，一旦需要的数据未成功备份，灾难就会出现。

所以，重要的环节需要亲自确认，避免传递错误带来的不确定性。

2. 在可能情况下保留多份备份介质。

很多事实显示，有时候单纯一份备份是不可靠的，磁带、移动硬盘等介质的备份更不可靠。所以，如果可能，对于重要数据的备份，最好保留多份介质，尤其是要对原有环境进行破坏、迁移、重构等情况时。

通常的备份策略，还应当结合物理备份和逻辑备份、表结构备份等来实施。对于极其重要的核心表，要保持经常性备份，多一份备份就多一份安全。

3. 避免使用无法把握的新技术。

前述案例的客户使用了自动存储管理技术（Oracle ASM）。ASM 使用裸设备，这造成了用户无法直接看到

数据文件，也就无从直接进行文件级别的复制备份，因而增加了备份的复杂性和难度。

显然用户和开发商运维人员都不能够深入了解这一技术。从这个意义上说，并非先进的技术就适合任何环境，好的技术要和好的运维结合起来才能为用户创造价值。客户在选择 Oracle 数据库的同时，必须要同时接受因此而可能支付的运行维护成本。

当然，系统架构人员也应当从用户的实际出发，为用户建议适合的系统架构，只有真正适合用户的技术才是最好的技术。

4. 必须有人能够把握技术全局。

在这个案例中，显然没有人能够从整体方案上把握全局。如果在核心的数据运维中没有人能把握全局，那么任何操作都应当极其慎重，或者干脆选择放弃进行破坏性数据维护操作。

典型地，如果没有回退方案，任何破坏性的数据操作都不应当进行。

5. 制订方案并且按照方案的步骤执行。

重要的维护工作应当制订方案，并且列出明确的操作步骤和命令。这些步骤和命令应当是通过测试验证的。在执行过程中，遵循方案的步骤来进行相关操作，一旦遇到异常必须停下来进行分析或者回退。

严格对数据负责，不因主观的重要和非重要判断行事。

6. 数据库维护操作应当通过命令行完成，避免使用图形化工具。

虽然图形化工具会为工作带来便利，但是其背后隐藏着不确定性和风险。通常用户很难确定图形后面默认和非默认的选项，任何一个错误的勾选都可能使情况变得复杂。

鉴于已经有很多用户在图形工具上遭遇挫折，我们建议用户通过命令行完成对数据库的维护操作，比如使用 SQL*Plus 工具。命令行会迫使你明确所发出的每一个指令，无形中可减少风险的出现。

明确工具的风险和用途，这也是对 DBA 的一个专业素质要求。

充足的数据备份和良好的维护计划是数据安全的守护者，在数据运维中要时刻注意。

技术回放

接下来我们看一看第一个案例的具体情形。

2011 年 12 月 31 日 12 时 34 分 53 秒，简简单单的一句 DROP TABLESPACE，删除了一个表空间中的三个

数据文件，用时不过 3 秒钟，可是后来的恢复工作却用时数天。

```
Sat Dec 31 12:34:53 2011
```

```
DROP TABLESPACE XF_PRODUCT INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS
```

```
Sat Dec 31 12:34:56 2011
```

```
Deleted file +DATA/orcl/datafile/xf_product01.dbf
```

```
Deleted file +DATA/orcl/datafile/xf_product02.dbf
```

```
Deleted file +DATA/orcl/datafile/xf_product03.dbf
```

```
Completed:
```

```
DROP TABLESPACE XF_PRODUCT INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS
```

在这里可以看到，Oracle 的 ASM 自动将数据文件删除了。在用户发出的命令中，还包含了删除数据文件和内容的指令，结果非常彻底地消灭了数据。

注意，这显然是通过工具发出的命令，将一条命令书写得如此完备对 DBA 的要求是非常高的。所以我们建议，重要的维护操作要在命令行完成，避免使用图形化的工具软件。工具虽然可带来方便，但是也会带来隐藏的风险和不确定性。

用户随后还删除了一系列的索引文件和临时文件。从为表空间规划了独立的索引文件和临时文件来看，这个数据库的早期规划是相当专业和细致的，只是后期的管理和运维显然没有跟上规划的步伐。

```
Sat Dec 31 12:34:56 2011
```

```
DROP TABLESPACE XF_PRODUCT_INDEX INCLUDING CONTENTS AND DATAFILES CASCADE  
CONSTRAINTS
```

```
Deleted file +DATA/orcl/datafile/xf_product_index01.dbf
```

```
Completed:
```

```
DROP TABLESPACE XF_PRODUCT_INDEX INCLUDING CONTENTS AND DATAFILES CASCADE  
CONSTRAINTS
```

```
Sat Dec 31 12:34:56 2011
```

```
DROP TABLESPACE XF_PRODUCT_TEMP INCLUDING CONTENTS AND DATAFILES CASCADE  
CONSTRAINTS
```

```
Deleted file +DATA/orcl/tempfile/xf_product_temp03.dbf
```

```
Deleted file +DATA/orcl/tempfile/xf_product_temp02.dbf
```

```
Deleted file +DATA/orcl/tempfile/xf_product_temp01.dbf
```

Completed:

```
DROP TABLESPACE XF_PRODUCT_TEMP INCLUDING CONTENTS AND DATAFILES CASCADE
CONSTRAINTS
```

接下来的日志输出显示操作人员完全不熟悉 ASM 技术。用户试图在 E 分区上创建数据文件，而该分区实际上是裸分区，未创建文件系统。这也说明未能有人审核相应的脚本。从下面的命令输出来看，这些命令语句应该是通过工具发出来的。也就是说这次维护操作根本没有明确的计划和步骤，临时发挥的现场操作导致了一片混乱。

以下创建尝试失败了。

Sat Dec 31 12:34:57 2011

```
CREATE SMALLFILE
    TABLESPACE "XF_PRODUCT"
    LOGGING    DATAFILE
    'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\XF_PRODUCT01.DBF' SIZE 100M
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

ORA-1119 signalled during:

```
CREATE SMALLFILE
    TABLESPACE "XF_PRODUCT"
    LOGGING    DATAFILE
    'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\XF_PRODUCT01.DBF' SIZE 100M
    EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

..

这里的 ORA-1119 错误的含义是文件创建失败，可能的原因是空间不足，本例中是因为该分区为裸分区。

```
[ora11g@ENMO ~]$ oerr ora 1119
```

```
01119, 00000, "error in creating database file '%s'"
```

```
// *Cause: Usually due to not having enough space on the device.
```

```
// *Action:
```

接下来还有很多出错的创建命令被执行，比如以下创建表空间的脚本遇到了 ORA-1543 错误，该错误提示要创建的表空间已经存在。在用户的数据库中，这个表空间确实存在，不曾删除（这其实是幸运），所以也就无法再次创建。

```
CREATE SMALLFILE
```

```
TABLESPACE "XF_PRODUCT_HISTORY"
LOGGING    DATAFILE
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\XF_PRODUCT_HISTORY01.DBF' SIZE 100M
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

ORA-1543 signalled during:

```
CREATE SMALLFILE
TABLESPACE "XF_PRODUCT_HISTORY"
LOGGING    DATAFILE
'E:\ORACLE\PRODUCT\10.2.0\ORADATA\ORCL\XF_PRODUCT_HISTORY01.DBF' SIZE 100M
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

..

这些操作也充分说明了用户及现场操作人员并不清楚自己在做什么，以及这样做可能产生的严重后果。

用户最后的正确选择是没有继续执行其他操作，而是直接寻求技术支持。对于这种情况，虽然文件从数据库中被删除，在 ASM 上也不可见，但是在存储级别的块级仍然存在，只要没有被覆盖，就仍然可以恢复出来。

ASM 默认以 1M 的单元（AU，ASM Unit）为单位进行数据分片存储。于是我们通过存储恢复软件扫描存储，找到这些 1M 大小的数据片，再进行整合拼接，恢复了 3 个数据库文件。然后结合用户之前备份的表结构信息，通过 Oracle 的内部数据抽取工具 DUL 将文件中的数据提取出来，加回到数据库之中，完成了数据恢复。

在 ASM 中创建表空间，可以极其简单。

```
SQL> create tablespace XF_PRODUCT datafile size 500M ;
```

表空间已创建。

这样才算利用到了 ASM 的便利之处。

在数据导入时，我们首先找到用户及口令信息，修改一个临时口令。

```
SQL> select username,password from dba_users where username='YNXF_PRODUCT';
```

USERNAME	PASSWORD
YNXF_PRODUCT	58C18F637CFB1EED

```
SQL> alter user ynxf_product identified by yn;
```

用户已更改。

此后，利用之前的备份导入数据结构。

```
C:\>imp ynxf_product/yn file=ynxf_product100111.dmp rows=n ignore=yes full=yes log=prod.log
连接到: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
```

经由常规路由 EXPORT:V10.02.01 创建的导出文件

已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入

. 正在将 YNXF_PRODUCT 的对象导入到 YNXF_PRODUCT

IMP-00017: 由于 ORACLE 错误 27477, 以下语句失败:

```
"BEGIN "
"dbms_scheduler.create_job('"JOB_CLEAR_PRODUCT_DATA"', "
"job_type=>'STORED_PROCEDURE', job_action=>"
```

.....

IMP-00091: 下列函数和对象出现以上错误: CREATE JOB_CLEAR_PRODUCT_DATA。将跳过此对象的其余 PL/SQL 块。

即将启用约束条件...

成功终止导入, 但出现警告。

然后通过恢复的 DMP 文件导入数据。由于存在依赖关系, 需要按照单表方式分先后进行导入, 以下是个别的导入范例。

```
C:\>imp ynxf_product/yn file=ynxf10.dmp ignore=yes log=imp.log statistics=none tables=T_CHECKITEM
commit=yes
```

经由常规路由 EXPORT:V10.02.01 创建的导出文件

警告: 这些对象由 YNXF 导出, 而不是当前用户

已经完成 ZHS16GBK 字符集和 AL16UTF16 NCHAR 字符集中的导入

. 正在将 YNXF 的对象导入到 YNXF_PRODUCT

. 正在将 YNXF 的对象导入到 YNXF_PRODUCT

. . 正在导入表 "T_CHECKITEM"导入了 362418 行

成功终止导入, 没有出现警告。

最后将用户口令修改为原始加密串, 完成恢复。

```
SQL> alter user ynxf_product identified by values '58C18F637CFB1EED';
```

用户已更改。

```
SQL> select username,password from dba_users where username='YNXF_PRODUCT';
```

USERNAME	PASSWORD
-----	-----
YNXF_PRODUCT	58C18F637CFB1EED

这个案例的恢复得益于最后创建的表空间未写入 ASM 磁盘组。如果写入则原有被删除的文件就可能被覆盖，那样就不可能实现完全恢复了。

对于前面提到的第二个案例，导入文件损坏，在导入时遇到如下错误。

```
IMP-00009: abnormal end of export file
```

中文版提示大致如下。

```
IMP-00009: 导出文件异常结束
```

```
IMP-00028: 上一个表的部分导入已回退: 回退 29050 行
```

这种情况通常是因为导出文件异常，当导入最后发现异常时，会回退所有操作。

回退在数据库处理是正常的，因为导出文件异常可能导致局部数据丢失，使数据的一致性和完整性无法确保。如果想导入部分完好的数据，可以指定 `commit=yes` 参数，如此即可提交这些残存记录。

以安全之期

基于对数据安全、备份安全的期望，我们认为对于备份有效性的验证、备份加密都非常重要，以下简单介绍一下与此相关的技术内容。

VALIDATE 实现备份验证

RMAN 工具提供了 VALIDATE 命令，可用于校验备份集的有效性，常用命令如下。

```
restore validate controlfile;
restore validate spfile;
restore validate database;
restore validate archivelog between sequence_low and sequence_high;
```

验证命令会检查备份的存在性、完好性和可恢复性，帮助我们确认备份有效与否，但是并不会进行实际的恢复动作。

比如验证控制文件和参数文件。

```
RMAN> restore validate controlfile;
Starting restore at 25-JAN-10
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=388 devtype=DISK

channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from
    backup piece /opt/oracle/product/db10g/dbs/c-1341966532-20100125-01
channel ORA_DISK_1: restored backup piece 1
piece handle=/opt/oracle/product/db10g/dbs/c-1341966532-20100125-01
channel ORA_DISK_1: validation complete, elapsed time: 00:00:02
```

Finished restore at 25-JAN-10

RMAN> restore validate spfile;

Starting restore at 25-JAN-10

using channel ORA_DISK_1

channel ORA_DISK_1: starting validation of datafile backupset

channel ORA_DISK_1: reading from

backup piece /opt/oracle/product/db10g/dbs/c-1341966532-20100116-00

channel ORA_DISK_1: restored backup piece 1

piece handle=/opt/oracle/product/db10g/dbs/c-1341966532-20100116-00

channel ORA_DISK_1: validation complete, elapsed time: 00:00:02

Finished restore at 25-JAN-10

根据备份集的大小,验证全备份的过程会有所不同,通常需要较长的时间。以下是一个全备份的验证过程。

RMAN> restore validate database;

Starting restore at 25-JAN-10

using channel ORA_DISK_1

data file 22 will be created automatically during restore operation

channel ORA_DISK_1: starting validation of datafile backupset

channel ORA_DISK_1: reading from

backup piece /data3/oradbak/orderfullback_order_20100124_4691

channel ORA_DISK_1: restored backup piece 1

piece handle=/data3/oradbak/orderfullback_order_20100124_4691 tag=order

channel ORA_DISK_1: validation complete, elapsed time: 00:02:36

channel ORA_DISK_1: starting validation of datafile backupset

channel ORA_DISK_1: reading from

backup piece /data3/oradbak/orderfullback_order_20100124_4692

channel ORA_DISK_1: restored backup piece 1

piece handle=/data3/oradbak/orderfullback_order_20100124_4692 tag=order

channel ORA_DISK_1: validation complete, elapsed time: 00:01:45

channel ORA_DISK_1: starting validation of datafile backupset

channel ORA_DISK_1: reading from

backup piece /data3/oradbak/orderfullback_order_20100124_4693

```

channel ORA_DISK_1: restored backup piece 1
piece handle=/data3/orclbak/orderfullback_order_20100124_4693 tag=order
channel ORA_DISK_1: validation complete, elapsed time: 00:00:26
channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from
        backup piece /data3/orclbak/orderfullback_order_20100124_4694
channel ORA_DISK_1: restored backup piece 1
piece handle=/data3/orclbak/orderfullback_order_20100124_4694 tag=order
channel ORA_DISK_1: validation complete, elapsed time: 00:00:56
channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from
        backup piece /data3/orclbak/orderfullback_order_20100124_4695
channel ORA_DISK_1: restored backup piece 1
piece handle=/data3/orclbak/orderfullback_order_20100124_4695 tag=order
channel ORA_DISK_1: validation complete, elapsed time: 00:00:07
channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from
        backup piece /data3/orclbak/orderfullback_order_20100124_4696
channel ORA_DISK_1: restored backup piece 1
piece handle=/data3/orclbak/orderfullback_order_20100124_4696 tag=order
channel ORA_DISK_1: validation complete, elapsed time: 00:00:04
channel ORA_DISK_1: starting validation of datafile backupset
channel ORA_DISK_1: reading from
        backup piece /data3/orclbak/orderfullback_order_20100124_4697
channel ORA_DISK_1: restored backup piece 1
piece handle=/data3/orclbak/orderfullback_order_20100124_4697 tag=order
channel ORA_DISK_1: validation complete, elapsed time: 00:00:07
failover to previous backup

data file 22 will be created automatically during restore operation
Finished restore at 25-JAN-10

```

验证命令并不会真正执行恢复，所以可以免去异机测试的麻烦，同时又可以检查备份的有效性，是应当定期执行的操作。

数据库备份加密

从 10g R2 开始，Oracle 数据库提供了备份加密功能。通过加密，可以保护备份文件，防止因为备份泄露导致的数据安全问题。

在 RMAN 中，可以查询数据库的备份加密算法和备份加密设置。以下显示数据库的加密算法为 AES128，当前未启用备份加密。

```
RMAN> show encryption for database;
RMAN configuration parameters for database with db_unique_name ORCL are:
CONFIGURE ENCRYPTION FOR DATABASE OFF; # default
RMAN> show encryption algorithm;
RMAN configuration parameters for database with db_unique_name ORCL are:
CONFIGURE ENCRYPTION ALGORITHM 'AES128'; # default
```

通过视图 `v$RMAN_ENCRYPTION_ALGORITHMS` 可以查询数据库支持的加密算法。以下是来自 Oracle Database 11g R2 的查询输出，Oracle 10g 支持的加密算法与此相同。

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

ALGORITHM_ID	ALGORITHM_NAME	ALGORITHM_DESCRIPTION	IS_	RES
1	AES128	AES 128-bit key	YES	NO
2	AES192	AES 192-bit key	NO	NO
3	AES256	AES 256-bit key	NO	NO

RMAN 的这些选项可通过 `CONFIGURE` 命令进行修改。如以下命令修改了加密算法。

```
RMAN> configure encryption algorithm 'AES256';
new RMAN configuration parameters:
CONFIGURE ENCRYPTION ALGORITHM 'AES256';
new RMAN configuration parameters are successfully stored
```

RMAN 的加密支持三种模式，即口令模式、透明模式（通过 Wallet 本地钱包加密）和混合模式。通常基于 Wallet 的加密方式因其透明性而更加安全，口令模式因引入口令而增加了额外的风险，所以如果不需要转移备份，通常不建议使用口令加密模式。

以下分别介绍这三种加密模式。

□ 令模式

口令模式是最为简单的备份加密模式。备份的时候通过以下语句设置备份密码，然后备份数据库或对应的表空间、数据文件等。

```
solaris*orcl-/home/oracle$ rman target /
Recovery Manager: Release 11.2.0.3.0 - Production on Wed Feb 15 10:58:19 2012
Copyright (c) 1982, 2011, Oracle and/or its affiliates. All rights reserved.

connected to target database: ORCL (DBID=1299676637)
RMAN> set encryption on identified by "eygle" only;
executing command: SET encryption
using target database control file instead of recovery catalog
RMAN> backup database;
Starting backup at 15-FEB-12
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=179 device type=DISK
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00004 name=/u01/app/oracle/oradata/ORCL/users01.dbf
input datafile file number=00003 name=/u01/app/oracle/oradata/ORCL/undotbs01.dbf
input datafile file number=00002 name=/u01/app/oracle/oradata/ORCL/sysaux.dbf
input datafile file number=00001 name=/u01/app/oracle/oradata/ORCL/system01.dbf
channel ORA_DISK_1: starting piece 1 at 15-FEB-12
channel ORA_DISK_1: finished piece 1 at 15-FEB-12
piece handle=/ORCL/backupset/2012_02_15/o1_mf_nnndf_TAG20120215T105827_7mp7tnfh_.bkp
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:55
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
including current control file in backup set
including current SPFILE in backup set
channel ORA_DISK_1: starting piece 1 at 15-FEB-12
channel ORA_DISK_1: finished piece 1 at 15-FEB-12
piece handle /ORCL/backupset/2012_02_15/o1_mf_ncsnf_TAG20120215T105827_7mp7wd9y_.bkp
```

```
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
Finished backup at 15-FEB-12
```

如果恢复时不提供密码，则会提示错误，不能解密，恢复不能执行。

```
RMAN> startup mount;
Oracle instance started
database mounted
```

```
Total System Global Area      835104768 bytes
```

```
Fixed Size                     2230592 bytes
```

```
Variable Size                  620758720 bytes
```

```
Database Buffers               209715200 bytes
```

```
Redo Buffers                   2400256 bytes
```

```
RMAN> restore database;
```

```
Starting restore at 15-FEB-12
```

```
using target database control file instead of recovery catalog
```

```
allocated channel: ORA_DISK_1
```

```
channel ORA_DISK_1: SID=170 device type=DISK
```

```
channel ORA_DISK_1: starting datafile backup set restore
```

```
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
```

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

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

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

```
RMAN-03002: failure of restore command at 02/15/2012 11:03:42
```

```
ORA-19870: error while restoring backup piece
```

```
      /ORCL/backupset/2012_02_15/o1_mf_nnndf_TAG20120215T105827_7mp7tnfh_.bkp
```

```
ORA-19913: unable to decrypt backup
```

```
ORA-28365: wallet is not open
```

恢复时，需要指定解密的密码才可以恢复。以下是指定密码之后的恢复过程。

```
RMAN> set decryption identified by "eygle";
executing command: SET decryption
```

```

RMAN> restore database;
Starting restore at 15-FEB-12
using channel ORA_DISK_1

channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
channel ORA_DISK_1: piece
handle=backupset/2012_02_15/o1_mf_nnndf_TAG20120215T105827_7mp7tnfh_.bkp
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:01:55
Finished restore at 15-FEB-12

```

透明模式

透明模式依赖于本地配置的 Wallet (钱包), 无须设置密码。如果备份仅在本地进行维护, 建议采用这样的加密方法, 更安全可靠, 更简便。这种方式需要额外配置 Oracle Encryption Wallet。

Oracle Encryption Wallet 配置的简要步骤如下。

配置 SQLNET.ORA 参数文件

设置 Wallet 地址信息, 目录决定了创建 Wallet 钱包的地址。

```

solaris*orcl-/home/oracle$ cd $ORACLE_HOME/network/admin
solaris*orcl-/u01/app/oracle/product/11.2.0/db_1/network/admin$ cat sqlnet.ora
ENCRYPTION_WALLET_LOCATION=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/u01/app/oracle)))

```

创建 Wallet

在 SYS 用户下创建 Wallet。

```

SQL> alter system set encryption key authenticated BY "eygle";
System altered.
SQL> ! ls -l /u01/app/oracle/ewallet*

```

```
-rw-r--r-- 1 oracle dba 1573 2月 15日 12:37 /u01/app/oracle/ewallet.p12
```

创建之后 Wallet 自动打开，但在此后重启数据库后需要手工开启。

关闭和开启 Wallet 的命令大致如下。

```
SQL> alter system set encryption wallet close identified by "eygle";
System altered.
```

```
SQL> alter system set wallet open identified by "eygle";
System altered.
```

接下来就可以使用 Wallet 实现透明的数据加密。以下是加密备份的过程示范。

```
RMAN> configure encryption for database on;
old RMAN configuration parameters:
CONFIGURE ENCRYPTION FOR DATABASE OFF;
new RMAN configuration parameters:
CONFIGURE ENCRYPTION FOR DATABASE ON;
new RMAN configuration parameters are successfully stored
RMAN> set encryption on;
executing command: SET encryption
RMAN> backup database;
Starting backup at 15-FEB-12
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=180 device type=DISK
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
input datafile file number=00004 name=/u01/app/oracle/oradata/ORCL/users01.dbf
input datafile file number=00003 name=/u01/app/oracle/oradata/ORCL/undotbs01.dbf
input datafile file number=00002 name=/u01/app/oracle/oradata/ORCL/sysaux.dbf
input datafile file number=00001 name=/u01/app/oracle/oradata/ORCL/system01.dbf
channel ORA_DISK_1: starting piece 1 at 15-FEB-12
channel ORA_DISK_1: finished piece 1 at 15-FEB-12
piece handle=/backupset/2012_02_15/o1_mf_nnndf_TAG20120215T124518_7mpg2z1f_.bkp
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:35
channel ORA_DISK_1: starting full datafile backup set
channel ORA_DISK_1: specifying datafile(s) in backup set
```

```

including current control file in backup set
including current SPFILE in backup set
channel ORA_DISK_1: starting piece 1 at 15-FEB-12
channel ORA_DISK_1: finished piece 1 at 15-FEB-12
piece handle=/backupset/2012_02_15/o1_mf_ncsnf_TAG20120215T124518_7mpg4334_.bkp
channel ORA_DISK_1: backup set complete, elapsed time: 00:00:01
Finished backup at 15-FEB-12

```

如果恢复时 Wallet 关闭，则恢复不可进行，会报告不可解密的错误。

```

SQL> alter system set encryption wallet close identified by "eygle";
System altered.

RMAN> restore database;
Starting restore at 15-FEB-12
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=170 device type=DISK

channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
ORA-19913: unable to decrypt backup
ORA-28365: wallet is not open

```

在打开 Wallet 的情形下，恢复可以正确执行。

```

RMAN> startup mount;
connected to target database (not started)
Oracle instance started
database mounted

RMAN> sql 'alter system set wallet open identified by "eygle"';
sql statement: alter system set wallet open identified by "eygle"

RMAN> restore database;
Starting restore at 15-FEB-12
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=170 device type=DISK

```

```
channel ORA_DISK_1: starting datafile backup set restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
channel ORA_DISK_1: restoring datafile 00001 to /oradata/ORCL/system01.dbf
channel ORA_DISK_1: restoring datafile 00002 to /oradata/ORCL/sysaux.dbf
channel ORA_DISK_1: restoring datafile 00003 to /oradata/ORCL/undotbs01.dbf
channel ORA_DISK_1: restoring datafile 00004 to /oradata/ORCL/users01.dbf
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: restore complete, elapsed time: 00:01:45
Finished restore at 15-FEB-12
```

混合模式

混合模式是同时启用 Wallet 和密码：在本地模式下，不需要密码恢复，实现透明；在异地异机恢复时，则需要提供口令恢复。

混合模式的密码设置如下，省去了 only 关键字。

```
RMAN> set encryption on identified by "mypass";
```

在异机恢复时，只需要指定口令就可以恢复，不需要 Wallet 的配合。

```
RMAN> set decryption on identified by "mypass";
```

```
RMAN> restore database;
```

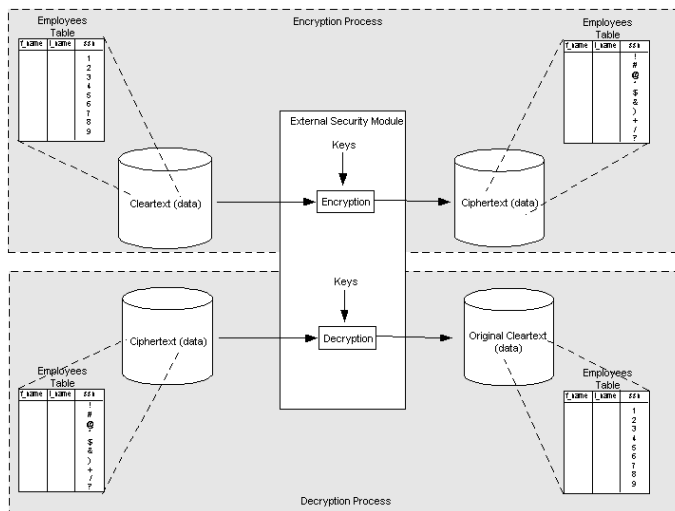
上述三种加密方式，可以根据需要选择使用。

透明加密（TDE）技术

RMAN 的透明加密，使用到了 Wallet 模式，这实际上是 Oracle 透明加密技术(Transparent Data Encryption)的一部分。

Oracle 的透明数据加密，是 Oracle 高级安全选项中的一个部分，需要额外支付软件费用。

这一选项，可以结合多种手段进行加密，包括使用 Wallet (PKCS#12 标准)，以及支持 PKCS#11 RAS 硬件设备。在 Oracle 10g 中，透明加密支持基于列级的加密，而在 Oracle 11g R2 中，增加了基于表空间的透明加密。以下是官方文档中关于加密/解密的流程图。



以下是 Windows 上基于 TDE 的一个简单测试范例。

首先，在 SQLNET.ora 文件中增加如下一段。

```
ENCRYPTION_WALLET_LOCATION=
  (SOURCE=(METHOD=FILE)(METHOD_DATA=
    (DIRECTORY=D:\Oracle\11.2.0\NETWORK\ADMIN\encryption_wallet)))
```

然后，在 SQL*Plus 中创建 Wallet 密钥。

```
SQL> connect / as sysdba
Connected.
```

```
SQL> alter system set encryption key authenticated by "eygle";
System altered.
```

关闭和打开 Wallet。

```
SQL> alter system set encryption wallet close;
alter system set encryption wallet close
*
```

```
ERROR at line 1:
```

```
ORA-28390: auto login wallet not open
```

```
SQL> alter system set encryption wallet close identified by "eygle";
System altered.
```

```
SQL> alter system set wallet open identified by "eygle";
System altered.
```

在创建数据表时可以指定加密。

```
SQL> connect eygle/eygle
Connected.

SQL> create table tde (id number(10),data varchar2(50) encrypt);
Table created.

SQL> insert into tde select user_id,username from dba_users;
9 rows created.

SQL> commit;
Commit complete.

SQL> connect / as sysdba
Connected.

SQL> select * from eygle.tde;
```

ID	DATA
0	SYS
5	SYSTEM
34	EYGLE
9	OUTLN
31	APPQOSSYS
30	DBSNMP
32	WMSYS
14	DIP
21	ORACLE_OCM

加密和解密是自动进行的。查询 dba_encrypted_columns 视图可以找到加密列。

```
SQL> select * from dba_encrypted_columns;
```

OWNER	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SAL
INTEGRITY_AL				

OWNER	TABLE_NAME	COLUMN_NAME	ENCRYPTION_ALG	SAL
EYGLE	TDE	DATA	AES 192 bits key	YES SHA-1

如果关闭 Wallet，则加密列不可访问。

```
SQL> select * from eygle.tde;
select * from eygle.tde
```

*

ERROR at line 1:

ORA-28365: wallet is not open

```
SQL> alter system set encryption wallet close identified by "eygle";
```

System altered.

```
SQL> select * from eygle.tde;
select * from eygle.tde
```

*

ERROR at line 1:

ORA-28365: wallet is not open

```
SQL> desc eygle.tde
```

Name	Null?	Type

ID		NUMBER(10)
DATA		VARCHAR2(50) ENCRYPT

```
SQL> select id from eygle.tde;
```

ID

0

5

34

9

31

30

32

14

21

9 rows selected.

在加密列时，存在两个选项：Salt 和 No Salt。

Salt 在加密前会对数据增加随机字符串,以增加破解的难度,使同样的字符串加密结果不同;而对于 No Salt,则同样字符串可以获得同样的加密输出，其安全性相对略低。

在加密列上，如果使用 Salt 方式，则不能创建索引。Salt 加密和索引两种属性互斥，不能同时设置。

```
SQL> create index idx01 on tde(data);
create index idx01 on tde(data)
```

*

ERROR at line 1:

ORA-28338: Column(s) cannot be both indexed and encrypted with salt

而如果使用默认 Salt 方式加密，则允许对于加密列创建索引。

```
SQL> create table tde2 (id number(10) encrypt no salt,data varchar2(50) );
```

Table created.

```
SQL> insert into tde2 select user_id,username from dba_users;
```

9 rows created.

```
SQL> select * from tde2;
```

ID DATA

0 SYS

5 SYSTEM

34 EYGLE

9 OUTLN

31 APPQOSSYS

30 DBSNMP

32 WMSYS

14 DIP

21 ORACLE_OCM

9 rows selected.

```
SQL> commit;
```

Commit complete.

```
SQL> create index idx1 on tde2(id);
```

Index created.

当执行导出时，Oracle 会给出提示。

```
D:\>expdp eygle/eygle directory=temp dumpfile=tde2.dmp tables=TDE
Export: Release 11.2.0.2.0 - Production on Thu Sep 8 15:35:19 2011
Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 11g Enterprise Edition Release 11.2.0.2.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
Starting "EYGLE"."SYS_EXPORT_TABLE_01": eygle/***** directory=temp
dumpfile=tde2.dmp tables=TDE
Estimate in progress using BLOCKS method...
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 64 KB
Processing object type TABLE_EXPORT/TABLE/TABLE
. . exported "EYGLE"."TDE"                    5.562 KB          9 rows
ORA-39173: Encrypted data has been stored unencrypted in dump file set.
Master table "EYGLE"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
*****
Dump file set for EYGLE.SYS_EXPORT_TABLE_01 is:
D:\TEMP\TDE2.DMP
Job "EYGLE"."SYS_EXPORT_TABLE_01" completed with 1 error(s) at 15:35:23
```

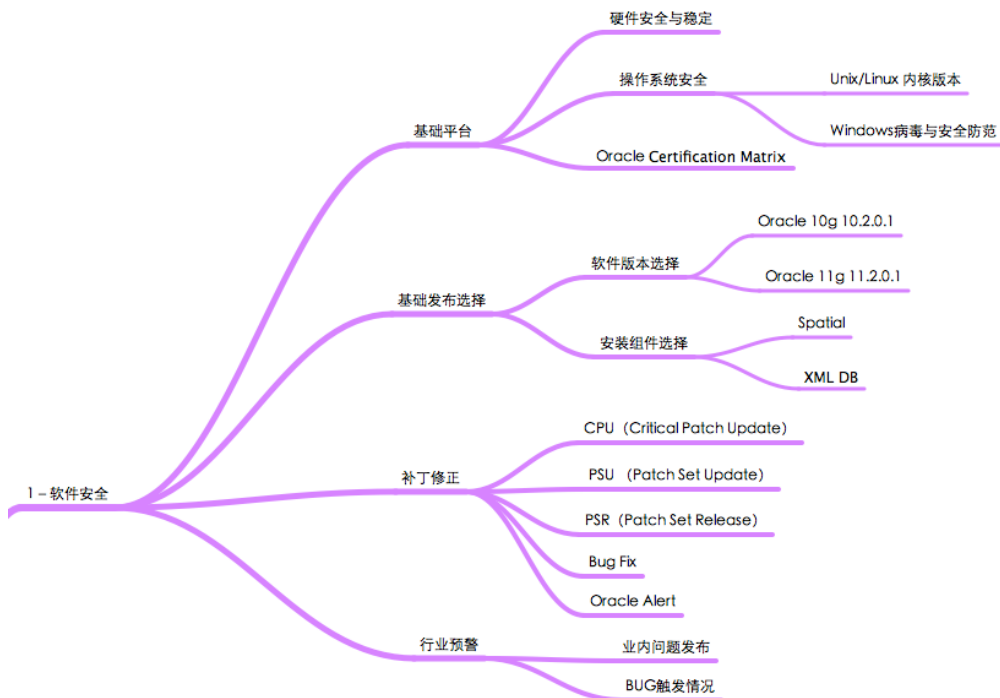

合抱之木，起于毫末

合抱之木，生于毫末；

九层之台，起于垒土；

千里之行，始于足下。

——《老子》第六十四章



“合抱之木，起于毫末”，这句话是说即便是参天大树，也是从幼苗开始生长，如果不能在事物的开端做好准备、规划，则后期的成长就可能会出现种种问题。

对于数据库来说，能否做好初始化部署与安装，及时修正已知的高风险漏洞，打好扎实的数据基础，消除数据库成长的隐患，是我们应该认真考量的重要方向之一——软件安全。

很多企业在部署软件之初，就选择了错误的版本。比如在已经发布了 Oracle Database 11.2.0.3 版本的今天，很多企业仍然在安装使用 11.2.0.1 版本，这就是非常不恰当的部署和应用。Oracle 在每一个主要补丁集中都修正了成百上千的 Bug，置如此多的问题于不顾，数据库中可能存在的种种风险由此可以想象。此外，Oracle 数据库中包含了很多产品组件，如果不需要，通常建议不要安装，因为这些组件可能会带来很多安全风险。

篇首图列举了软件安全层面可能涉及的诸多因素。

Oracle 数据库软件发布序列

对于 Oracle 数据库软件，其软件及补丁共分为以下几种类型。

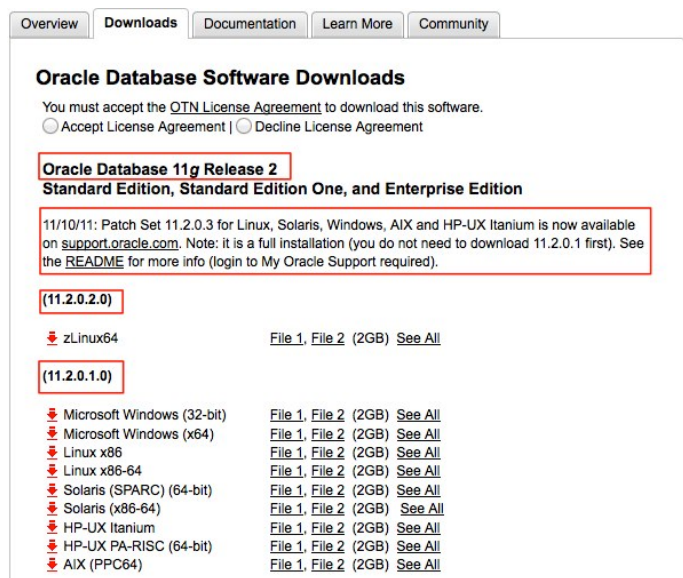
名 称	说 明
Release	标准产品发布。如 11g R2 的第一个发布版本为 11.2.0.1，在 OTN 可以下载
PSU（Patch Set Update）	补丁集更新。Oracle 选取在每一季用户下载数量多，并且得到验证具有较低风险的补丁放入到每个季度的 PSU 中
CPU（Critical Patch Update）	紧急补丁更新。Oracle 在 2005 年开始引入的产品安全更新策略，一般来说 CPU 包含了 Oracle 产品安全漏洞的修复补丁集
PSR（Patch Set Release）	包含 PSU 和 CPU，以及其他 Bug 修正的补丁集合
Patch	对于特定 Bug 的单独修复补丁

以上类型中,公开的 Release 版本可以从 OTN 上下载,但是其余修正内容,则只能从 MOS(My Oracle Support) 中下载。MOS 只对购买了 Oracle 产品服务的用户开放。

从 Oracle 11g 开始,一个新的补丁策略被引入,11.2.0.1 之后发布的 Patch Set 本身就是一个完整的安装包,不再需要基础的发布版本安装,在 OTN 上可以看到重要的提示信息。比如下页图中 2011 年 11 月 11 日给出的信息是: Patch Set 11.2.0.3 的 Linux/Solaris/Windows/AIX/HP-UX Itanium 版本在 MOS 上已经可用,其为完整的安装版本,不需要预先下载标准发布版本。

对于 Oracle 数据环境的部署,一定要选择当前最为成熟稳定的发布版本,并仔细阅读补丁修正中的修复描述。比如对于 Oracle Database 11g 的部署,目前最为合适的版本就是 11.2.0.3,而对于 Oracle Database 10g 的部署,就应当优先选择 10.2.0.5 版本,这是 Oracle 10g 最后的 Patch Set 发布版本。

在基础的软件发布和补丁集之外,Oracle 对于其产品每个季度发行一次的安全补丁包 CPU 与 PSU 补丁,通常是为了修复产品中的安全隐患,并可能包含对一些严重 Bug 及功能组件的修复。



Oracle 的 CPU/PSU 一般固定在下表所列 4 个月份发布，发布日期为最接近 17 号的星期二。2012 年的几个发布时间分别如下。

月 份	内 容	日 期
January	CPU/PSU	17 January 2012
April	CPU/PSU	17 April 2012
July	CPU/PSU	17 July 2012
October	CPU/PSU	16 October 2012

CPU 是 Oracle 在 2005 年开始引入的产品安全更新策略，是 Oracle 关于安全方面的唯一修正来源。Oracle 致力于通过 CPU 为客户周期性地提供累积性补丁以修复安全漏洞。CPU 中修正的多数是已批露和报告的高危安全漏洞，Oracle 强烈推荐用户应用 CPU 补丁集。

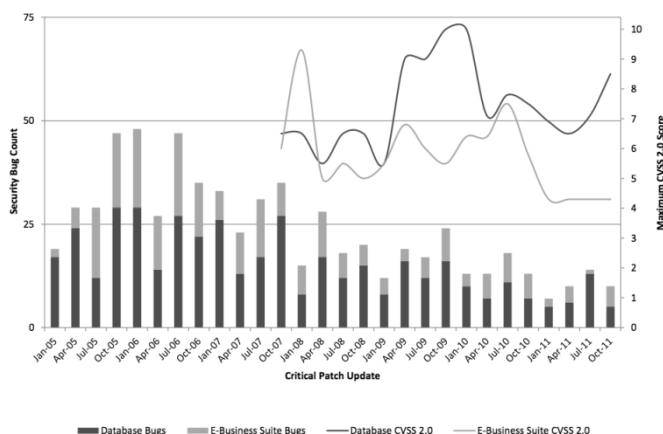
Oracle 在每个季度会将用户下载数量多，并且经过验证具有较低风险的补丁放入到 PSU 中。在 PSU 中，不但包含 Bug 修复，而且包含最新的 CPU。PSU 主要用于解决 PSR 周期长更新不及时的问题，同时用于处理补丁冲突等问题。

下面列举了一些近期 Oracle 发布的高危漏洞说明，其中的很多漏洞和 Oracle 的一些产品组件相关。

- CVE-2011-3525: 这个安全漏洞将导致任意 APEX 用户有机会全权控制主机服务器。这个安全漏洞的风险性极高, 建议任何安全部署 APEX 的用户立即修正这一安全漏洞。
- CVE-2011-3512: 对于所有的 Oracle RDBMS 用户来说, 这都是最严重的安全风险。除安装补丁外, 没有任何的临时方案。这一漏洞不需要任何特别的 PACKAGE 权限, 利用 Spatial 对象进行 SQL 注入, 任何用户都可以获得数据库的控制权限。
- CVE-2011-2301: 这个风险会导致任何有组件权限的用户获得权限提升, 默认的具有 EXECUTE ANY PROCEDURE 权限的用户或者 CTXSYS 用户获得对数据库服务器的完全控制, 任何用户都应当立即应用这个修正。
- CVE-2011-3511: 这个漏洞会允许任何拥有 DB_ACTMGR (Account Manager users) 权限的用户绕过 Database Vault 的保护去修改 VALUT 用户密码, 使用 Database Vault 的用户都应该立即修正该 Bug。Oracle Database Vault 在保护用户账户时采取了分离权限的措施, 带有 SYSDBA 权限(CVE-2011-2322) 或 DV_ACCTMGR 能力 (CVE-2011-3511) 的用户可绕过这些保护措施, 更改用户调用 OCIPasswordChange 客户端 API 的密码。
- CVE-2011-2322: 这是另外一个会导致权限提升的漏洞, 使用 Database Vault 会使用户获得 SYSDBA 超级权限。这个问题曾在 April 2011 CPU 中得到部分修正。

可以看到, 在以上的安装部署中, 受到影响的组件有 Spatial、TEXT、Valut 等, 这些组件使数据库遭受到了严重的安全风险。所以通常建议用户在确定不需要某些组件时, 不要安装相应的数据库模块, 以减少数据库的安全风险。

下图列举了 Oracle 安全策略制订以来, 定期发布的安全补丁数量。图中, 深色部分标示的是数据库的安全补丁数量。



正确评估这些风险, 选择性应用相应修正, 是数据库管理和服务的重大重要工作内容。

一个逻辑坏块引发的灾难

作为重要的核心数据库系统，应当密切关注 Oracle 的补丁修正和同行业的风险报告，以及社区事件报告。Oracle 官方也会根据需要提供补丁或安全预警，提醒用户及时修正已知的重要缺陷或 Bug。这些内容十分重要，应当预先关注而不是等数据库遇到这些问题时才去了解。

案例警示

这个案例给了我们以下的教训。

1. 对于软件及更新的持续关注不可懈怠。

在这个案例中，数据库因为已知的 Bug 而遭遇严重事故。这告诉我们，应当对基础软件保持持续的关注，同时参考行业的分析警示公告，定期评估基础软件中可能存在的 Bug 及安全风险，并认真决策软件的修正与维护操作。

很多企业对于稳定运行的系统存在心理麻痹，想当然地认为系统仍然可以继续稳定运行下去。而事实上，很多 Bug 或漏洞，仅在特定的条件下触发，安全的漏洞更需要一段时间去暴露。如果我们掉以轻心，风险也许就会在不久后不期而遇。

2. 对于数据库系统的使用方式应当明确。

客户对于自有的业务系统，应当列表总结明确数据库的访问和使用方式，针对特殊的访问和使用方式，如 DB Link、CTAS、NOLOGGING 等，需要定期跟踪 Oracle 发布的安全预警和 Bug 修正，确认自己的系统不会遭受同样问题的影响。

越是清晰地了解自己的系统，就越可能及早发现问题，规避风险。

3. 储备和维系快速反应的技术专家团队。

很多时候故障发生在一瞬间，及时地判断和响应对于解决故障至关重要。所以维系、储备专家资源和技术团队对于核心数据库应用必不可少。在关键时刻，通过既定的职责和团队分工、内容分解，可以大幅加快分析和处理速度。

同时保障一个可以用于测试实验的环境也同样重要，很多想法和判断只有在快速测试验证之后，才能够支

持我们去应用到生产环境中。

当一个陌生的故障或异常出现时，解决时间是 2 小时、4 小时还是更久，取决于我们事前做了什么样的准备工作，一个充满活力的学习型团队和组织是企业数据环境的强大支撑。

技术回放

如果一个已知问题没有被注意到，则有可能在核心系统中埋下巨大的安全隐患。作为数据库从业人员，应当知道数据库只有两类，一类是已经崩溃的，另一类是即将崩溃的，舍此无他。

某客户的核心数据库系统，曾经因为一个已知的 Bug 反复遭遇数据灾难，这个著名的 Bug 就是

Bug 5386204 Block corruption / OERI[kddummy_blkchk] after direct load of ASSM segment。

这个 Bug 出现在对于 ASSM 存储表的直接路径加载后，其影响范围极其广泛。在以下 Bug 描述列出的影响范围中，可以看到 10.2.0.3 和 10.2.0.4 都在其列。这两个版本应用极其广泛，而很多用户都在数据环境中使用 Direct Load 方式加载数据，可以想象有多少风险存在。很多用户在 10.2.0.5 版本发布之后，迟迟不愿应用这个重要补丁集，但是至少应该关注在这个补丁集中修正了哪些可能遇到的 Bug。

Product (Component)	Oracle Server (Rdbms)
Range of versions believed to be affected	Versions < 11
Versions confirmed as being affected	9.2.0.8 10.2.0.1 10.2.0.2 10.2.0.3 10.2.0.4
Platforms affected	Generic (all / most platforms affected)

更惨痛的是，这个 Bug 在 10.2.0.4 版本中曾经声明已修正，但事实上直到 10.2.0.5 版本才彻底排除。

Description

Block corruption / ORA-600 [kddummy_blkchk][file#] [block#] [18038]
can occur on a segment which has been direct loaded.

(The corruption shows as a PAGETABLE SEGMENT HEADER
having blocks in the "Auxillary Map" outside of the "Extent Map"
range)

Note:
This bug was previously incorrectly listed as fixed in 10.2.0.4

这个 Bug 在某些金融客户系统中曾多次遭遇，损失惨重。

一个硬盘坏块引发的灾难

以下这次数据库故障非常曲折，但究其根本原因，竟然仅仅是一个硬盘坏块。

灾难描述

以下是一次惨痛的多层次数据库灾难。

1. 客户为了增强系统安全性，对系统根卷进行镜像。
2. 镜像后出现错误，系统重启后需要恢复，于是执行了操作系统级别的恢复。
3. 恢复使得某磁盘组重新生成了 PVID。
4. 该卷原为 ASM 磁盘的裸分区。
5. ASM 磁盘头覆盖损坏，ASM 磁盘无法加载。
6. 灾难形成。

这次复杂和曲折的数据库灾难，客户原本为了安全而采取的措施，最终却引发了一次故障。即便在本书中，这也已经不是第一次出现。这个案例最终找出的原因是镜像盘存在一个坏块，这个坏块引发的修复及后续操作破坏了 ASM 磁盘。

案例警示

通过这个案例的详细发生过程，我们获得了以下启示。

1. **重要维护应当详细检测系统各方面的安全与稳定性。**

在这个案例中，最后发现问题的根源在于，加入根卷镜像的磁盘存在坏块错误，镜像后，原有的根卷盘也出现了同步坏块错误，使得系统提示需要进行恢复。

由于 ODM 存储于根盘，ODM 在恢复时进行了一系列的操作，这其中包括了对于磁盘卷组的 PVID 重算，并最终导致了故障。这场无妄之灾仅仅是因为一个坏块。

所以我们建议大家，在进行重要的维护操作时，应当详细检查相关资源的运行状况，确保系统当前状态的

稳定稳固，确保新增加的设备、配件的健康安全，确保不要将问题引入到当前的系统中来。

2. 复杂的维护需要选择停机窗口长的时段进行。

即便有非常乐观的估计，在进行复杂维护时，也要尽量争取充足的停机窗口。因为隐藏的故障可能大大增加维护的复杂度，一个意外就可能几倍的时间需求，所以对维护进行充足的时间考量，是我们重要的维护计划内容。

这个案例中客户选择了在节假日进行，使得为故障处理赢得了充足的时间窗口，未对业务产生不利影响。

3. 做好最坏的打算才能有最好的结局。

在进行数据运维或者系统维护、变更时，一定要做最坏的打算，这样才能有最好的结局。通常很多工程师都会下意识地假设，进行操作系统级别维护或者网络维护跟数据库无关，非常简单，等等。但是注意，这种因素都和数据库有关，任何一个环节出现问题都可能导致数据库故障。

所以，在进行数据库系统运维时，最好先假定各个环节都可能出现问题，然后进行逐层分析，做好可能性防范，这样才能最大程度地降低维护的风险。在当今高度耦合的计算环境中，系统的相关性使得任何一个环节都不是孤立存在的，必须充分认识这种复杂性从而有效地保障安全。

4. 项目计划要做好充分的资源准备。

在进行维护计划制订时，要做好充足的资源准备，以便在需要时可以快速获得技术支援，做出正确判断。充足的资源准备对于应对故障、快速解决疑难具有决定性作用。

良好的计划、充足的准备是信心的保证，必须具备十足的把握和信心，才能够在维护变更和应对异常时游刃有余，应付自如。

5. 数据库的重要补丁和版本升级非常重要

ASM 磁盘头损坏的情况，一直以来困扰了很多企业和 DBA。在缺少备份情况下，修复 ASM 磁盘头可能出现的种种损坏是相当复杂的。但是从 Oracle 10g 10.2.0.5.0 版本开始，一个自动备份块的引入，使得这种情况下的恢复变得极其简单。

所以，及时准确地获得正确的软件版本，及时应用重要的补丁修正，对于数据库稳定运行和安全具有重大影响。我们希望用户都能够随时关注软件版本情况，为数据库运行构建稳定的基础支持。万丈高楼平地起，关于基础维护，不能出现任何疏忽。

这就是这个案例带给我们的经验与教训。

技术回放

AIX 系统 ODM 简介

对于这个案例，我们需要首先了解一下 ODM 的概念。

在 AIX 操作系统上,内置了一个用于管理的小型关系型数据库,叫做 ODM 数据库(Object Data Manager)。AIX 的许多特性与功能的配置信息都存储在 ODM 库中，如 SMIT、系统配置信息、动态配置信息、磁盘配置信息等。

在 ODM 的设备配置（Device Configuration）区中包含了所有配置的物理卷、卷组和逻辑卷的相关信息。这些信息是对 VGDA（卷组描述区）中信息的镜像。常规的操作，当我们导入（import）一个 VGDA 的过程包括把导入卷组的 VGDA 数据拷贝到 ODM 中。当一个卷组被导出（export）时，保存在 ODM 中的有关该卷组的数据被从 ODM 数据库中删除。

ODM 数据库信息存放于以下三个目录中。

目录结构	记录内容
/etc/objrepos	主要存储不能网络共享的数据，如用户设备定义类等
/usr/lib/objrepos	主要存储只能由 AIX 系统只读共享的数据
/usr/share/lib/objrepos	主要存储可共享，不依赖于 AIX 的数据

在这个案例中，由于 AIX ODM 的损坏与恢复，导致磁盘组的 PVID 重新计算并写入了 ASM 磁盘组，最终引发了故障。

如果没有 ASM 磁盘头信息的备份，手工去修复 ASM 磁盘头是非常困难的。客户的系统存在备份，因而通过备份快速恢复了数据，但是我们仍然要探讨一下 ASM 磁盘头的问题。

由于 ASM 使用裸盘进行数据存储，而操作系统也经常会尝试占用第一个数据块，因而系统工程师的一个处置不当，就可能导致严重的数据库故障。

ASM 头块备份机制

在 Oracle Database Patch Set 10.2.0.5.0 发布之后，我们强烈建议用户升级到该版本上来。因为在这一版本上（头块备份功能最初在 Oracle 11g R1 中引入，随后被引入到 10.2.0.5 版本中），Oracle 会对 ASM 磁盘头自动完成一个镜像备份，这个镜像备份使得常规的磁盘头块损坏的恢复变得易如反掌。

以下是一个常规的 ASM 头块的主要信息，通过 kfed 可以读取，其 Oracle 数据库版本为 10.2.0.5。

```
p55a@/u01/admin/cnd> kfed read /dev/rhdisk3

kfbh.endian:                0 ; 0x000: 0x00
kfbh.hard:                   130 ; 0x001: 0x82
kfbh.type:                   1 ; 0x002: KFBTYP_DISKHEAD
kfbh.datfmt:                 1 ; 0x003: 0x01
kfbh.block.blk:              0 ; 0x004: T=0 NUMB=0x0
kfbh.block.obj:              2147483648 ; 0x008: TYPE=0x8 NUMB=0x0
kfbh.check:                   3015734793 ; 0x00c: 0xb3c07609
kfbh.fcn.base:                44 ; 0x010: 0x0000002c
kfbh.fcn.wrap:                0 ; 0x014: 0x00000000
kfbh.spare1:                  0 ; 0x018: 0x00000000
kfbh.spare2:                  0 ; 0x01c: 0x00000000
kfdhdb.driver.provstr:        ORCLDISK ; 0x000: length=8
kfdhdb.driver.reserved[0]:    0 ; 0x008: 0x00000000
kfdhdb.driver.reserved[1]:    0 ; 0x00c: 0x00000000
kfdhdb.driver.reserved[2]:    0 ; 0x010: 0x00000000
kfdhdb.driver.reserved[3]:    0 ; 0x014: 0x00000000
kfdhdb.driver.reserved[4]:    0 ; 0x018: 0x00000000
kfdhdb.driver.reserved[5]:    0 ; 0x01c: 0x00000000
kfdhdb.compat:                168820736 ; 0x020: 0x0a100000
kfdhdb.dsknum:                0 ; 0x024: 0x0000
kfdhdb.grptyp:                1 ; 0x026: KFDGTP_EXTERNAL
kfdhdb.hdrsts:                3 ; 0x027: KFDHDR_MEMBER
kfdhdb.dskname:               DATA_0000 ; 0x028: length=9
kfdhdb.grpname:               DATA ; 0x048: length=4
kfdhdb.fgname:               DATA_0000 ; 0x068: length=9
```

```

kfdhdb.capname:                ; 0x088: length=0
kfdhdb.crestmp.hi:              32890869 ; 0x0a8: HOUR=0x15 DAYS=0x1f MNTH=0x7
YEAR=0x7d7
kfdhdb.crestmp.lo:              256121856 ; 0x0ac: USEC=0x0 MSEC=0x107 SECS=0x34
MINS=0x3
kfdhdb.mntstmp.hi:              32956086 ; 0x0b0: HOUR=0x16 DAYS=0x15 MNTH=0x7
YEAR=0x7db
kfdhdb.mntstmp.lo:              2504303616 ; 0x0b4: USEC=0x0 MSEC=0x129 SECS=0x14
MINS=0x25
kfdhdb.secsz:                   512 ; 0x0b8: 0x0200
kfdhdb.blksz:                   4096 ; 0x0ba: 0x1000
kfdhdb.ausiz:                   1048576 ; 0x0bc: 0x00100000
kfdhdb.mfact:                   113792 ; 0x0c0: 0x0001bc80
kfdhdb.dksiz:                   51200 ; 0x0c4: 0x0000c800
kfdhdb.pmcnt:                   2 ; 0x0c8: 0x00000002
kfdhdb.fstlocl:                 1 ; 0x0cc: 0x00000001
kfdhdb.altlocl:                 2 ; 0x0d0: 0x00000002
kfdhdb.f1b1locl:                2 ; 0x0d4: 0x00000002
kfdhdb.redomirrors[0]:          0 ; 0x0d8: 0x0000
kfdhdb.redomirrors[1]:          65535 ; 0x0da: 0xffff
kfdhdb.redomirrors[2]:          65535 ; 0x0dc: 0xffff
kfdhdb.redomirrors[3]:          65535 ; 0x0de: 0xffff
kfdhdb.dbcompat:                168820736 ; 0x0e0: 0x0a100000
kfdhdb.grpstmp.hi:              32890869 ; 0x0e4: HOUR=0x15 DAYS=0x1f MNTH=0x7
YEAR=0x7d7
kfdhdb.grpstmp.lo:              255995904 ; 0x0e8: USEC=0x0 MSEC=0x8c SECS=0x34
MINS=0x3

```

在随后的第 510 块上（具体的块号根据版本不同会有所不同），存储着一个备份块，这个块的内容和头块完全相同。在遇到故障时，可以通过 kfed 读取和恢复这个数据块，因而使得传统的 ASM 头块恢复变得非常简单。这是 Oracle 在多个版本的总结之后做出的一个重要改进，也是 DBA 们期待已久的一个重要功能。

```

p55a@/u01/admin/cnd> kfed read /dev/rhdisk3 blkn=510
kfbh.endian:                    0 ; 0x000: 0x00
kfbh.hard:                      130 ; 0x001: 0x82

```

```

kfbh.type:                1 ; 0x002: KFBTYP_DISKHEAD
kfbh.datfmt:              1 ; 0x003: 0x01
kfbh.block.blk:           0 ; 0x004: T=0 NUMB=0x0
kfbh.block.obj:          2147483648 ; 0x008: TYPE=0x8 NUMB=0x0
kfbh.check:              3015734793 ; 0x00c: 0xb3c07609
kfbh.fcn.base:            44 ; 0x010: 0x0000002c
kfbh.fcn.wrap:            0 ; 0x014: 0x00000000
kfbh.spare1:              0 ; 0x018: 0x00000000
kfbh.spare2:              0 ; 0x01c: 0x00000000
kfdhdb.driver.provstr:    ORCLDISK ; 0x000: length=8
kfdhdb.driver.reserved[0]: 0 ; 0x008: 0x00000000
kfdhdb.driver.reserved[1]: 0 ; 0x00c: 0x00000000
kfdhdb.driver.reserved[2]: 0 ; 0x010: 0x00000000
kfdhdb.driver.reserved[3]: 0 ; 0x014: 0x00000000
kfdhdb.driver.reserved[4]: 0 ; 0x018: 0x00000000
kfdhdb.driver.reserved[5]: 0 ; 0x01c: 0x00000000
kfdhdb.compat:           168820736 ; 0x020: 0x0a100000
kfdhdb.dsknum:            0 ; 0x024: 0x0000
kfdhdb.grptyp:            1 ; 0x026: KFDGTP_EXTERNAL
kfdhdb.hdrsts:            3 ; 0x027: KFDHDR_MEMBER
kfdhdb.dskname:           DATA_0000 ; 0x028: length=9
kfdhdb.grpname:           DATA ; 0x048: length=4
kfdhdb.fgname:           DATA_0000 ; 0x068: length=9
kfdhdb.capname:           ; 0x088: length=0
kfdhdb.crestmp.hi:        32890869 ; 0x0a8: HOUR=0x15 DAYS=0x1f MNTH=0x7
YEAR=0x7d7
kfdhdb.crestmp.lo:        256121856 ; 0x0ac: USEC=0x0 MSEC=0x107 SECS=0x34
MINS=0x3
kfdhdb.mntstmp.hi:        32956086 ; 0x0b0: HOUR=0x16 DAYS=0x15 MNTH=0x7
YEAR=0x7db
kfdhdb.mntstmp.lo:        2504303616 ; 0x0b4: USEC=0x0 MSEC=0x129 SECS=0x14
MINS=0x25
kfdhdb.secsz:             512 ; 0x0b8: 0x0200

```

```

kfdhdb.blksize:          4096 ; 0x0ba: 0x1000
kfdhdb.auseize:          1048576 ; 0x0bc: 0x00100000
kfdhdb.mfact:            113792 ; 0x0c0: 0x0001bc80
kfdhdb.dsksize:          51200 ; 0x0c4: 0x0000c800
kfdhdb.pmcnt:            2 ; 0x0c8: 0x00000002
kfdhdb.fstlocn:          1 ; 0x0cc: 0x00000001
kfdhdb.altlocn:          2 ; 0x0d0: 0x00000002
kfdhdb.f1b1locn:        2 ; 0x0d4: 0x00000002
kfdhdb.redomirrors[0]:   0 ; 0x0d8: 0x0000
kfdhdb.redomirrors[1]:   65535 ; 0x0da: 0xffff
kfdhdb.redomirrors[2]:   65535 ; 0x0dc: 0xffff
kfdhdb.redomirrors[3]:   65535 ; 0x0de: 0xffff
kfdhdb.dbcompat:         168820736 ; 0x0e0: 0x0a100000
kfdhdb.grpstmp.hi:       32890869 ; 0x0e4: HOUR=0x15 DAYS=0x1f MNTH=0x7
YEAR=0x7d7
kfdhdb.grpstmp.lo:       255995904 ; 0x0e8: USEC=0x0 MSEC=0x8c SECS=0x34
MINS=0x3

```

kfed 工具编译与使用

修复 ASM 故障需要用到 Oracle 的 kfed 工具，该工具默认未编译。对于 Oracle 10g R2 版本，可以通过如下步骤获得 kfed 工具。

```

cd $ORACLE_HOME/rdbms/lib
make -f ins_rdbms.mk ikfed

```

以下是可能的编译输出（引自 Oracle 10g R2+Linux 环境）。

```

[root@danaly ~]# su - oracle
[oracle@danaly ~]$ cd $ORACLE_HOME/rdbms/lib
[oracle@danaly lib]$ make -f ins_rdbms.mk ikfed
Linking KFED utility (kfed)
rm -f /opt/oracle/product/10.2.0/rdbms/lib/kfed
gcc -o /opt/oracle/product/10.2.0/rdbms/lib/kfed ...
mv -f /opt/oracle/product/10.2.0/bin/kfed /opt/oracle/product/10.2.0/bin/kfed0

```

```
mv: cannot stat `/opt/oracle/product/10.2.0/bin/kfed': No such file or directory
make: [ikfed] Error 1 (ignored)
mv /opt/oracle/product/10.2.0/rdbms/lib/kfed /opt/oracle/product/10.2.0/bin/kfed
chmod 751 /opt/oracle/product/10.2.0/bin/kfed
[oracle@danaly lib]$ which kfed
~/product/10.2.0/bin/kfed
```

在 Oracle 10g 中，kfed 工具的常用参数如下。

```
[oracle@bosssdb-rac2 lib]$ kfed -help
as/mlib          ASM Library [asmlib='lib']
aun/um           AU number to examine or update [AUNUM=number]
aus/z            Allocation Unit size in bytes [AUSZ=number]
blkcn/um         Block number to examine or update [BLKNUM=number]
blks/z           Metadata block size in bytes [BLKSZ=number]
ch/ksum          Update checksum before each write [CHKSUM=YES/NO]
cn/t             Count of AUs to process [CNT=number]
d/ev            ASM device to examine or update [DEV=string]
o/p             KFED operation type [OP=READ/WRITE/MERGE/NEW/FORM/FIND/STRUCT]
p/rovnm         Name for provisioning purposes [PROVNM=string]
s/seek          AU number to seek to [SEEK=number]
te/xt           File name for translated block text [TEXT=string]
ty/pe           ASM metadata block type number [TYPE=number]
```

通过以上帮助说明可以看到，kfed 支持对于 ASM 信息的 READ/WRITE/MERGE/NEW/FORM/FIND/STRUCT 等操作，大体上非常完备了，只是还缺少一个重要选项。这个选项在 Oracle 11g 中加入了进来，它就是 REPAIR（修复），以下是引自 Oracle Database 11g 的 kfed 帮助信息输出。

```
[ora11g@hpserver2 ~]$ kfed
as/mlib          ASM Library [asmlib='lib']
aun/um           AU number to examine or update [AUNUM=number]
aus/z            Allocation Unit size in bytes [AUSZ=number]
blkcn/um         Block number to examine or update [BLKNUM=number]
blks/z           Metadata block size in bytes [BLKSZ=number]
ch/ksum          Update checksum before each write [CHKSUM=YES/NO]
cn/t             Count of AUs to process [CNT=number]
```

de/v	ASM device to examine or update [DEV=string]
dm/pall	Don't suppress repeated lines when dumping corrupt blocks [DMPALL=YES/NO]
o/p	KFED operation type [OP=READ/WRITE/MERGE/ REPAIR /NEW/FORM/FIND/STRUCT]
p/rovnm	Name for provisioning purposes [PROVNM=string]
s/seek	AU number to seek to [SEEK=number]
te/xt	File name for translated block text [TEXT=string]
ty/pe	ASM metadata block type number [TYPE=number]

这是一个期待已久的功能，有了备份块和 kfed 的修复功能，在磁盘头损坏之后，只需一个 repair 命令就可以完成恢复。

如下这条命令就能够恢复一个损坏的 ASM 磁盘头。

```
kfed repair /dev/rhdisk3
```


手工修复 ASM 案例一则

如果没有 ASM 自动的头块备份,也没有用户的主动备份,那么恢复 ASM 头块损坏将是非常复杂和困难的。以下就是这样一则恢复案例。

灾难描述

以下是这次数据库故障的大致情形。

1. 客户 RAC 系统的一个节点出现故障。
2. 重新安装后,加入 Cluster 成功。
3. 添加 ASM 实例出现错误,无法正确识别磁盘组。
4. 故障形成。

在这个案例中,用户的数据库版本为 10.2.0.4,无法利用到 Oracle 自动备份数据块的特性,也就无法直接恢复。由于没有头块备份,因此只能通过手工方式进行恢复。

技术回放

在类似的案例中,一般来说,重装过程中最复杂的是 Cluster 的清除和添加。如果 Cluster 已经添加成功,那么剩下的就是数据库级别的操作,相对来说会简单得多。根据上面的信息,初步判断问题可能发生在以下几个方面。

- 新节点的 ORACLE_HOME 的版本和旧节点不一致。
- 10204 补丁没有在 Cluster 上安装。
- 在新节点上没有给 Oracle 用户授权。
- 共享存储在新节点上挂载存在问题。

PROVISIONED 磁盘状态分析

到了客户现场后才发现，问题和我们的初始判断大相径庭，数据库的版本、权限等都不存在问题。事实上，新节点上的 ASM 实例已经启动，且两个磁盘组中的一个已经 MOUNT，但另外一个磁盘组无法 MOUNT。

```
SQL> alter diskgroup datag1 mount;
alter diskgroup datag1 mount
*
ERROR at line 1:
ORA-15032: not all alterations performed
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DATAG1"
```

ORA-15032 只是一个描述性错误，而关键的 ORA-15063 错误比较少见。

由于另外一个节点从 RAC 环境出现故障后一直没有停机，因此可以从这个节点的 ASM 实例上检查磁盘组和磁盘的状态。这又一次给了我们一个提醒，在故障时不要贸然关闭或者重启数据库，否则可能使问题变得更加复杂。

```
SQL> select group_number, name, state, total_mb, free_mb from v$asm_diskgroup;
GROUP_NUMBER NAME STATE TOTAL_MB FREE_MB
-----
1 DATAG1 MOUNTED 512000 217396
2 DATAG2 MOUNTED 1024000 347457

SQL> select group_number, disk_number, mount_status, header_status, name, path
2 from v$asm_disk where group_number = 1;
GROUP_NUMBER DISK_NUMBER MOUNT_STATUS HEADER_STATUS NAME PATH
-----
1 0 CACHED PROVISIONED DATAG1_0000 /dev/raw/raw4
1 1 CACHED MEMBER DATAG1_0001 /dev/raw/raw5

SQL> select instance_number, instance_name from v$instance;
INSTANCE_NUMBER INSTANCE_NAME
-----
2 +ASM2
```

可以看到，磁盘组 DATAG1 中的磁盘 DATAG1_0000 的文件头状态不正确，其状态为 PROVISIONED。

从新添加的实例 1 上检查 ASM 磁盘信息。

```
SQL> select group_number, name, state, total_mb, free_mb from v$asm_diskgroup;
```

GROUP_NUMBER	NAME	STATE	TOTAL_MB	FREE_MB
1	DATAG1	DISMOUNTED	0	0
2	DATAG2	MOUNTED	1024000	347457

```
SQL> select group_number, disk_number, mount_status, header_status, name, path
      2  from v$asm_disk;
```

GROUP_NUMBER	DISK_NUMBER	MOUNT_STATUS	HEADER_STATUS	NAME	PATH
0	0	CLOSED	FOREIGN		/dev/raw/raw1
0	1	CLOSED	FOREIGN		/dev/raw/raw2
0	2	CLOSED	FOREIGN		/dev/raw/raw8
0	3	CLOSED	FOREIGN		/dev/raw/raw10
0	4	CLOSED	FOREIGN		/dev/raw/raw9
0	8	CLOSED	PROVISIONED		/dev/raw/raw4
0	5	CLOSED	MEMBER		/dev/raw/raw5
2	1	CACHED	MEMBER	DATAG2_0001	/dev/raw/raw7
2	0	CACHED	MEMBER	DATAG2_0000	/dev/raw/raw6

```
SQL> select instance_number, instance_name from v$instance;
```

INSTANCE_NUMBER	INSTANCE_NAME
1	+ASM1

由于节点 1 上的 ASM 磁盘组 1 无法 MOUNT，因此可以看到/dev/raw/raw4 和/dev/raw/raw5 两个磁盘对应的 GROUP_NUMBER 是 0，且 MOUNT_STATUS 是 CLOSED，而 raw4 对应的 HEADER_STATUS 也是 PROVISIONED，显然就是这个错误的状态值导致当前节点无法 MOUNT 磁盘组。

由于另外一个实例一直没有关闭过，因此虽然磁盘头的状态不正常，但是并不影响这个磁盘组的正常使用，而如果这时对 ASM 实例进行重启，则这个磁盘组必然无法再次 MOUNT。

根据 Oracle 文档的描述，PROVISIONED 和 CANDIDATE 状态有所区别。二者虽然都表示磁盘为候选磁盘，不属于任何一个磁盘组，不过 CANDIDATE 是正常的候选状态，而 PROVISIONED 则表示磁盘经过特殊的处理，比如操作系统工具对磁盘进行过特殊的操作。

既然这个磁盘的状态是 PROVISIONED，那么能否通过再次添加这个磁盘到磁盘组，使得磁盘头的状态变成正常？于是我们尝试在问题节点再次添加这个磁盘，但是由于磁盘组处于 NOMOUNT 状态，因而添加操作

失败，而尝试在目前正常工作的节点添加磁盘，结果同样报错。

```
SQL> alter diskgroup datag1 add disk '/dev/raw/raw4';
alter diskgroup datag1 add disk '/dev/raw/raw4'
*
ERROR at line 1:
ORA-15032: not all alterations performed
ORA-15029: disk '/dev/raw/raw4' is already mounted by this instance
```

看来常规手段已经无法解决这个问题，于是只剩两个办法：一是重建 ASM 磁盘，二是直接修改 ASM 磁盘头信息。

重建意味着较长的停机时间，而且客户当前环境中配置了 DATA GUARD，重建 ASM 磁盘组也不是一件轻松的事情。当前的 PRIMARY 数据库和 STANDBY 数据库都是两节点的 RAC 环境，在数据库中还配置了 STREAM 的 CAPTURE，这使得数据库架构异常复杂，进行 SWITCHOVER 的难度也相对比较大。而且对于当前运行的节点而言，一旦关闭，很有可能导致 ASM 磁盘无法 MOUNT，从而导致原本计划中的 SWITCHOVER 变成 FAILOVER，甚至有可能损失 ONLINE REDO LOGFILE 的风险。因此，无论从哪个角度看，重建都不是一个最佳的解决方案，如果能直接将 ASM 磁盘的头信息改写正确，无疑是最直接、代价最小的解决方案。

使用 kfed 修改 ASM 磁盘头信息

修改 ASM 磁盘头信息，需要借助 Oracle 的工具 kfed。以下用 kfed 来检查 ASM 磁盘裸设备的文件头信息。

```
[oracle@node1 data]$ kfed read /dev/raw/raw4 > raw4.txt
[oracle@node1 data]$ kfed read /dev/raw/raw5 > raw5.txt
[oracle@node1 data]$ kfed read /dev/raw/raw6 > raw6.txt
```

将磁盘组 DATAG1 的两个磁盘头和另一个磁盘组 DATAG2 的第一个文件的头信息输出到文本文件，对这三个文件进行比对，找到 raw4 文件中异常的部分。

由于每个磁盘组中有两个文件，且有两个磁盘组，因而比对的工作非常顺利，定位到 raw4 中存在两个标志位异常。以下 kfed 输出标示出了两个特殊位置。

```
[oracle@nccpxdb1 lib]$ kfed read /dev/raw/raw4
kfbh.endian: 1 ; 0x000: 0x01
kfbh.hard: 130 ; 0x001: 0x82
kfbh.type: 1 ; 0x002: KFBTYP_DISKHEAD
kfbh.datfmt: 1 ; 0x003: 0x01
```

```

kfbh.block.blk: 0 ; 0x004: T=0 NUMB=0x0
kfbh.block.obj: 2147483648 ; 0x008: TYPE=0x8 NUMB=0x0
kfbh.check: 4024565597 ; 0x00c: 0xefef1ff5d
kfbh.fcn.base: 952047 ; 0x010: 0x000e86ef
kfbh.fcn.wrap: 0 ; 0x014: 0x00000000
kfbh.spare1: 0 ; 0x018: 0x00000000
kfbh.spare2: 0 ; 0x01c: 0x00000000
kfdhdb.driver.provstr: ORCLDISK ; 0x000: length=8
kfdhdb.driver.reserved[0]: 0 ; 0x008: 0x00000000
kfdhdb.driver.reserved[1]: 0 ; 0x00c: 0x00000000
kfdhdb.driver.reserved[2]: 0 ; 0x010: 0x00000000
kfdhdb.driver.reserved[3]: 0 ; 0x014: 0x00000000
kfdhdb.driver.reserved[4]: 0 ; 0x018: 0x00000000
kfdhdb.driver.reserved[5]: 0 ; 0x01c: 0x00000000
kfdhdb.compat: 168820736 ; 0x020: 0x0a100000
kfdhdb.dsknum: 0 ; 0x024: 0x0000
kfdhdb.grptyp: 1 ; 0x026: KFDGTP_EXTERNAL
kfdhdb.hdrsts: 3 ; 0x027: KFDHDR_MEMBER
kfdhdb.dskname: DATAG1_0000 ; 0x028: length=11
kfdhdb.grpname: DATAG1 ; 0x048: length=6
kfdhdb.fgname: DATAG1_0000 ; 0x068: length=11
kfdhdb.capname: ; 0x088: length=0
kfdhdb.crestmp.hi: 32928501 ; 0x0a8: HOUR=0x15 DAYS=0x17 MNTH=0xc YEAR=0x7d9
kfdhdb.crestmp.lo: 2195144704 ; 0x0ac: USEC=0x0 MSEC=0x1d0 SECS=0x2d MINS=0x20
kfdhdb.mntstmp.hi: 32940275 ; 0x0b0: HOUR=0x13 DAYS=0x7 MNTH=0x8 YEAR=0x7da
kfdhdb.mntstmp.lo: 3201116160 ; 0x0b4: USEC=0x0 MSEC=0x34a SECS=0x2c MINS=0x2f
kfdhdb.secsz: 512 ; 0x0b8: 0x0200
kfdhdb.blksz: 4096 ; 0x0ba: 0x1000
kfdhdb.ausize: 1048576 ; 0x0bc: 0x00100000
kfdhdb.mfact: 113792 ; 0x0c0: 0x0001bc80
kfdhdb.dksz: 512000 ; 0x0c4: 0x0007d000
kfdhdb.pmcnt: 6 ; 0x0c8: 0x00000006
kfdhdb.fstlocln: 1 ; 0x0cc: 0x00000001

```

```

kfdhdb.altlocn: 2 ; 0x0d0: 0x00000002
kfdhdb.f1b1locn: 2 ; 0x0d4: 0x00000002
kfdhdb.redomirrors[0]: 0 ; 0x0d8: 0x0000
kfdhdb.redomirrors[1]: 65535 ; 0x0da: 0xffff
kfdhdb.redomirrors[2]: 65535 ; 0x0dc: 0xffff
kfdhdb.redomirrors[3]: 65535 ; 0x0de: 0xffff
kfdhdb.dbcompat: 168820736 ; 0x0e0: 0x0a100000
kfdhdb.grpstmp.hi: 32928501 ; 0x0e4: HOUR=0x15 DAYS=0x17 MNTH=0xc YEAR=0x7d9
kfdhdb.grpstmp.lo: 2195053568 ; 0x0e8: USEC=0x0 MSEC=0x177 SECS=0x2d MINS=0x20
kfdhdb.ub4spare[0]: 0 ; 0x0ec: 0x00000000
kfdhdb.ub4spare[1]: 0 ; 0x0f0: 0x00000000
.....
kfdhdb.ub4spare[40]: 0 ; 0x18c: 0x00000000
kfdhdb.ub4spare[41]: 0 ; 0x190: 0x00000000
kfdhdb.ub4spare[42]: 0 ; 0x194: 0x00000000
kfdhdb.ub4spare[43]: 104436 ; 0x198: 0x000197f4
kfdhdb.ub4spare[44]: 0 ; 0x19c: 0x00000000
.....
kfdhdb.ub4spare[57]: 0 ; 0x1d0: 0x00000000
kfdhdb.acdb.aba.seq: 0 ; 0x1d4: 0x00000000
kfdhdb.acdb.aba.blk: 0 ; 0x1d8: 0x00000000
kfdhdb.acdb.ents: 0 ; 0x1dc: 0x0000
kfdhdb.acdb.ub2spare: 43605 ; 0x1de: 0xaa55

```

而正常磁盘 raw5 的 kfdhdb.ub4spare[43]和 kfdhdb.acdb.ub2spare 相关标志位全为 0，是一致的状态。

```

[oracle@nccpxdb1 lib]$ kfed read /dev/raw/raw5
kfbh.endian: 1 ; 0x000: 0x01
kfbh.hard: 130 ; 0x001: 0x82
kfbh.type: 1 ; 0x002: KFBTYP_DISKHEAD
kfbh.datfmt: 1 ; 0x003: 0x01
kfbh.block.blk: 0 ; 0x004: T=0 NUMB=0x0
kfbh.block.obj: 2147483649 ; 0x008: TYPE=0x8 NUMB=0x1
kfbh.check: 1802212223 ; 0x00c: 0x6b6b937f
kfbh.fcn.base: 0 ; 0x010: 0x00000000

```

```

kfbh.fcn.wrap: 0 ; 0x014: 0x00000000
kfbh.spare1: 0 ; 0x018: 0x00000000
kfbh.spare2: 0 ; 0x01c: 0x00000000
kfdhdb.driver.provstr: ORCLDISK ; 0x000: length=8
kfdhdb.driver.reserved[0]: 0 ; 0x008: 0x00000000
kfdhdb.driver.reserved[1]: 0 ; 0x00c: 0x00000000
kfdhdb.driver.reserved[2]: 0 ; 0x010: 0x00000000
kfdhdb.driver.reserved[3]: 0 ; 0x014: 0x00000000
kfdhdb.driver.reserved[4]: 0 ; 0x018: 0x00000000
kfdhdb.driver.reserved[5]: 0 ; 0x01c: 0x00000000
kfdhdb.compat: 168820736 ; 0x020: 0x0a100000
kfdhdb.dsknum: 1 ; 0x024: 0x0001
kfdhdb.grptyp: 1 ; 0x026: KFDGTP_EXTERNAL
kfdhdb.hdrsts: 3 ; 0x027: KFDHDR_MEMBER
kfdhdb.dskname: DATAG1_0001 ; 0x028: length=11
kfdhdb.grpname: DATAG1 ; 0x048: length=6
kfdhdb.fgname: DATAG1_0001 ; 0x068: length=11
kfdhdb.capname: ; 0x088: length=0
kfdhdb.crestmp.hi: 32928501 ; 0x0a8: HOUR=0x15 DAYS=0x17 MNTH=0xc YEAR=0x7d9
kfdhdb.crestmp.lo: 2195144704 ; 0x0ac: USEC=0x0 MSEC=0x1d0 SECS=0x2d MINS=0x20
kfdhdb.mntstmp.hi: 32940275 ; 0x0b0: HOUR=0x13 DAYS=0x7 MNTH=0x8 YEAR=0x7da
kfdhdb.mntstmp.lo: 3201116160 ; 0x0b4: USEC=0x0 MSEC=0x34a SECS=0x2c MINS=0x2f
kfdhdb.secsz: 512 ; 0x0b8: 0x0200
kfdhdb.blksz: 4096 ; 0x0ba: 0x1000
kfdhdb.ausize: 1048576 ; 0x0bc: 0x00100000
kfdhdb.mfact: 113792 ; 0x0c0: 0x0001bc80
kfdhdb.dsksize: 512000 ; 0x0c4: 0x0007d000
kfdhdb.pmcnt: 6 ; 0x0c8: 0x00000006
kfdhdb.fstlocl: 1 ; 0x0cc: 0x00000001
kfdhdb.altlocl: 2 ; 0x0d0: 0x00000002
kfdhdb.f1b1locl: 0 ; 0x0d4: 0x00000000
kfdhdb.redomirrors[0]: 0 ; 0x0d8: 0x0000
kfdhdb.redomirrors[1]: 0 ; 0x0da: 0x0000

```

```

kfdhdb.redomirrors[2]: 0 ; 0x0dc: 0x0000
kfdhdb.redomirrors[3]: 0 ; 0x0de: 0x0000
kfdhdb.dbcompat: 168820736 ; 0x0e0: 0x0a100000
kfdhdb.grpstmp.hi: 32928501 ; 0x0e4: HOUR=0x15 DAYS=0x17 MNTH=0xc YEAR=0x7d9
kfdhdb.grpstmp.lo: 2195053568 ; 0x0e8: USEC=0x0 MSEC=0x177 SECS=0x2d MINS=0x20
kfdhdb.ub4spare[0]: 0 ; 0x0ec: 0x00000000
kfdhdb.ub4spare[1]: 0 ; 0x0f0: 0x00000000
.....
kfdhdb.ub4spare[38]: 0 ; 0x184: 0x00000000
kfdhdb.ub4spare[39]: 0 ; 0x188: 0x00000000
kfdhdb.ub4spare[40]: 0 ; 0x18c: 0x00000000
kfdhdb.ub4spare[41]: 0 ; 0x190: 0x00000000
kfdhdb.ub4spare[42]: 0 ; 0x194: 0x00000000
kfdhdb.ub4spare[43]: 0 ; 0x198: 0x00000000
kfdhdb.ub4spare[44]: 0 ; 0x19c: 0x00000000
kfdhdb.ub4spare[45]: 0 ; 0x1a0: 0x00000000
.....
kfdhdb.acdb.ents: 0 ; 0x1dc: 0x0000
kfdhdb.acdb.ub2spare: 0 ; 0x1de: 0x0000

```

而磁盘组 DATAG2 的两个磁盘 raw6 和 raw7 的标志位也都是全 0。

确定了问题后，手工修改这个文本文件，将 kfdhdb.ub4spare[43]和 kfdhdb.acdb.ub2spare 均改为 0。

```

kfdhdb.ub4spare[43]: 0 ; 0x198: 0x00000000
kfdhdb.ub4spare[44]: 0 ; 0x19c: 0x00000000
.
.
.
kfdhdb.acdb.ents: 0 ; 0x1dc: 0x0000
kfdhdb.acdb.ub2spare: 0 ; 0x1de: 0x0000

```

然后利用 kfed 的 merge 命令将其写回磁盘头。

```

[oracle@node1 data]$ kfed merge /dev/raw/raw4 text=raw4_new.txt
[oracle@node1 data]$ export ORACLE_SID=+ASM1
[oracle@node1 data]$ sqlplus / as sysdba

```

SQL*Plus: Release 10.2.0.4.0 - Production on Mon Jun 13 18:45:54 2011

Copyright (c) 1982, 2007, Oracle. All Rights Reserved.

Connected to:

Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, Real Application Clusters, Oracle Label Security, OLAP,
Data Mining and Real Application Testing options

SQL> set pages 100 lines 120

SQL> select group_number, name, state, total_mb, free_mb from v\$asm_diskgroup;

GROUP_NUMBER	NAME	STATE	TOTAL_MB	FREE_MB
1	DATAG1	DISMOUNT	0	0
2	DATAG2	MOUNTED	1024000	347457

SQL> select group_number, disk_number, mount_status, header_status, name, path
2 from v\$asm_disk;

GROUP_NUMBER	DISK_NUMBER	MOUNT_STATUS	HEADER_STATUS	NAME	PATH
0	0	CLOSED	FOREIGN		/dev/raw/raw1
0	1	CLOSED	FOREIGN		/dev/raw/raw2
0	2	CLOSED	FOREIGN		/dev/raw/raw8
0	3	CLOSED	FOREIGN		/dev/raw/raw10
0	4	CLOSED	FOREIGN		/dev/raw/raw9
0	8	CLOSED	MEMBER		/dev/raw/raw4
0	5	CLOSED	MEMBER		/dev/raw/raw5
2	1	CACHED	MEMBER	DATAG2_0001	/dev/raw/raw7
2	0	CACHED	MEMBER	DATAG2_0000	/dev/raw/raw66

SQL> alter diskgroup datag1 mount;

Diskgroup altered.

SQL> shutdown immediate

ASM diskgroups dismounted

ASM instance shutdown

SQL> startup

ASM instance started

Total System Global Area 130023424 bytes

```
Fixed Size 2082208 bytes
Variable Size 102775392 bytes
ASM Cache 25165824 bytes
ASM diskgroups mounted
SQL> exit
```

利用 `merge` 修改磁盘头后，磁盘状态正常，磁盘组顺利挂载。此时检查节点 1 上的磁盘头信息，状态也恢复了正常。

在运行 `merge` 命令的时候，整个磁盘组并没有卸载，数据库也在正常运行。也就是说，在整个操作过程中，并没有一秒钟的停机时间，所有的修改都是 `ONLINE` 进行的。

ASM 数据抽取恢复

通过 AMDU 恢复数据案例一则

2012 年 2 月 21 日，一个香港用户的 ASM 破坏，请求数据恢复。

灾难描述

这则案例是由于存储误操作引起的。

1. 用户进行存储维护和磁盘添加操作。
2. 维护后发现 CRS 无法启动。
3. 检查发现 OCR 盘损坏，ASM 磁盘组受损。
4. 经用户反复确认，故障原因是误操作磁盘导致的 ASM 磁盘受损。
5. 为减少意外，客户请求在不更改配置等的情况下安全抽取数据。
6. 数据库为 3 节点 RAC 系统。

灾难再一次由于疏忽而降临。

案例警示

这类案例我们已经遭遇了很多，在这里再次郑重提示以下内容。

1. 存储的使用分配应当自始至终建立和维护详细的档案。

鉴于众多惨痛的数据灾难，结合标准化流程要求，我们建议用户对于数据库的存储规划、分配，建立详细的档案记录，在进行后续的维护操作时，严格通过文档进行对比确认，杜绝低级的误操作行为。

标准化和文档维护不仅仅是流程和管理的需要，也是对技术人员屏蔽错误、保障数据安全的基本要求。我们不能把文档当做过场或可有可无的摆设，必须将其上升到数据安全的保障层面。

2. 涉及存储的调整，必须多部门协同反复确认。

由于底层存储对于数据库的核心作用，因而必须在进行维护时反复确认维护计划，多部门统一协调、共同确认，避免流程割裂导致的误操作行为。

常规的存储维护在企业中被定义为系统或存储部门的责任，但是运行于其上的数据库是数据存储的核心使用者，在存储维护的过程中，一定要数据部门介入和确认。对于 Oracle 数据库来说，由于前期的存储划分可能非常零散，包括 OCR、VOTING、REDO、DATA 等都可能存在独立的存储分区，所以如果不进行严格管理，在后期维护中就可能对其中的部分存储卷产生误操作，导致破坏。

当然，关于备份的重要性，如何强调都不为过，始终保有有效的备份才能够在出现问题时有备无患。

技术回放

对于这个案例，我们有多种手段可以恢复，只要 ASM 磁盘组完好，就可以很容易地从中提取数据。本案例我们使用了 AMDU 工具进行恢复。

AMDU 工具

在 Oracle 10g 中，ASM 磁盘组的信息需要在 Mount 之后才能通过内部视图查询，如果磁盘组因为故障无法正常加载，那么信息将不可用，这为诊断带来了诸多不便。

从 Oracle 11g 开始，Oracle 提供了一个工具 AMDU 用于协助诊断。通过这个工具可以在磁盘组加载之前将 ASM 的元数据抽取出来，用于数据库诊断。这个工具可以向后兼容，引入到 Oracle 10g 中。

通过 `amdu -h` 可以查看详细的帮助说明。直接调用 `amdu`，将自动生成一个以时间命名的目录，其下的报告文件记录了磁盘组的相关信息。

```
[oracle@enmou1 ~]$ amdu
amdu_2011_03_29_10_28_41/
[oracle@enmou1 ~]$ cd amdu_2011_03_29_10_28_41/
[oracle@enmou1 amdu_2011_03_29_10_28_41]$ ls
report.txt
```

该报告的主要内容如下。

```
[oracle@enmou1 amdu_2011_03_29_10_28_41]$ more report.txt
- *- amdu - *-
```

***** AMDU Settings *****

ORACLE_HOME = /u01/app/db/11.2.0

System name: Linux

Node name: enmou1

Release: 2.6.18-128.el5

Version: #1 SMP Wed Dec 17 11:41:38 EST 2008

Machine: x86_64

amdu run: 29-MAR-11 10:28:41

Endianness: 1

----- Operations -----

***** DISCOVERY *****

----- DISK REPORT N0001 -----

Disk Path: ORCL:VOL1

Unique Disk ID:

Disk Label: VOL1

Physical Sector Size: 512 bytes

Disk Size: 1954 megabytes

Group Name: CRSDG

Disk Name: VOL1

Failure Group Name: VOL1

Disk Number: 0

Header Status: 3

Disk Creation Time: 2011/03/17 11:39:10.772000

Last Mount Time: 2011/03/29 09:21:38.608000

Compatibility Version: 0x0b200000(11020000)

Disk Sector Size: 512 bytes

Disk size in AUs: 1954 AUs

Group Redundancy: 1

Metadata Block Size: 4096 bytes →元数据块大小, 4KB

AU Size: 1048576 bytes →AU 大小: 1MB

Stride: 113792 AUs

```
Group Creation Time: 2011/03/17 11:39:10.671000
File 1 Block 1 location: AU 2          →文件使用, 从 AU 2 开始
OCR Present: NO
```

***** END OF REPORT *****

定义特定的参数可以获得 ASM 磁盘组内部的区间分配等详细信息。以下命令指定转储 CRS DG 的磁盘组信息, 除报告文件外, 还生成了 map 和 img 信息文件。

```
[oracle@enmou1 ~]$ amdu -diskstring '/dev/oracleasm/disks/VOL*' -dump 'CRSDG'
amdu_2011_03_29_10_36_03/
[oracle@enmou1 ~]$ cd amdu_2011_03_29_10_36_03/
[oracle@enmou1 amdu_2011_03_29_10_36_03]$ ls
CRSDG_0001.img  CRS DG.map  report.txt
```

这里 map 文件的信息如下, 其中描述了 ASM 元数据在磁盘组中的位置, 最后部分就是指针信息。

```
[oracle@enmou1 amdu_2011_03_29_10_36_03]$ more CRS DG.map
N0001 D0000 R00 A00000000 F00000000 I0 E00000000 U00 C00256 S0001 B00000000000
N0001 D0000 R00 A00000001 F00000000 I0 E00000000 U00 C00256 S0001 B0001048576
N0001 D0000 R00 A00000002 F00000001 I0 E00000000 U00 C00256 S0001 B0002097152
N0001 D0000 R00 A00000003 F00000002 I0 E00000000 U00 C00256 S0001 B0003145728
N0001 D0000 R00 A00000004 F00000003 I0 E00000000 U00 C00256 S0001 B0004194304
```

而 img 文件则是元数据块的镜像转储, 为二进制文件。这些文件在 ASM 出现故障时, 可用于收集信息, 分析故障。

AMDU 有一个重要参数 extract, 该参数可用于从 ASM 磁盘组中抽取数据文件。以下是 AMDU 的帮助信息摘录。

```
extr/act          Files to extract
-extract <diskgroup>.<file_number>: This extracts the numbered file
      from the named diskgroup, case insensitive. This option may be
      specified multiple times to extract multiple files. The extracted
      file is placed in the dump directory under the name
      <diskgroup>_<number>.f where <diskgroup> is the diskgroup name
      in uppercase, and <number> is the file number. The -output option
      may be used to write the file to any location. The extracted file
```

will appear to have the same contents it would have if accessed through the database. If some portion of the file is unavailable then that portion of the output file will be filled with 0xBADF00A, and a message will appear on stderr.

这个选项可用于直接从 ASM 磁盘组中抽取数据文件。

文件分析

由于磁盘组不能 Mount，控制文件也无法访问，因此需要首先分析数据库的文件分布情况，进而通过文件的 ASM 存储序号来进行文件抽取。

通过告警日志，可以找到数据库的控制文件信息。如下所示，控制文件的 ASM 文件号是 270。

System parameters with non-default values:

```
processes                = 150
spfile                   = "+DG_DATA/proda02/spfileproda02.ora"
memory_target            = 4G
control_files             = "+DG_REDO/proda02/controlfile/current.270.768047731"
db_block_size            = 8192
compatible               = "11.2.0.0.0"
cluster_database         = TRUE
db_create_file_dest       = "+DG_DATA"
db_create_online_log_dest_1= "+DG_REDO"
thread                   = 2
undo_tablespace          = "UNDOTBS2"
instance_number          = 2
remote_login_passwordfile= "EXCLUSIVE"
db_domain                = ""
remote_listener           = "rac03-cluster-scan:1521"
audit_file_dest           = "/u01/oracle/admin/proda02/adump"
audit_trail              = "DB"
db_name                  = "proda02"
open_cursors              = 300
diagnostic_dest           = "/u01/oracle"
```

随后即可通过 `amdu` 提取控制文件。

```
su - oracle
mkidr -p /u01/d02
cd /u01/d02
amdu -extract DG_REDO.270
```

取得控制文件之后，可以通过控制文件内容获得数据库的数据文件及日志文件分布情况。以下是从控制文件中获得的信息输出。

```
strings DG_REDO.270 | grep +DG_DATA>02_DATAFILE.TXT
strings DG_REDO.270 | grep +DG_REDO >> 02_DATAFILE.TXT
cat ./02_DATAFILE.TXT
+DG_DATA/proda02/datafile/system.282.769365511
+DG_DATA/proda02/datafile/sysaux.278.769365517
+DG_DATA/proda02/datafile/undotbs1.281.769365521
+DG_DATA/proda02/datafile/undotbs2.279.769365529
+DG_DATA/proda02/datafile/undotbs3.277.769365529
+DG_DATA/proda02/datafile/users.276.769365531

+DG_REDO/proda02/onlinelog/group_1.274.769365509
+DG_REDO/proda02/onlinelog/group_2.273.769365509
+DG_REDO/proda02/onlinelog/group_5.276.769366163
+DG_REDO/proda02/onlinelog/group_6.275.769366163
+DG_REDO/proda02/onlinelog/group_3.272.769366163
+DG_REDO/proda02/onlinelog/group_4.271.769366165
```

有了文件分布信息，接下来的恢复工作就大大简化了。

AMDU 文件恢复

获得文件的分布信息之后，即可使用 `amdu` 工具进行文件提取工作。根据如上的数据文件和日志文件信息，创建如下脚本以抽取对应的日志文件和数据文件。

```
amdu -extract DG_DATA.282
amdu -extract DG_DATA.278
amdu -extract DG_DATA.281
```

```

amdu -extract DG_DATA.280
amdu -extract DG_DATA.279
amdu -extract DG_DATA.277
amdu -extract DG_DATA.276
amdu -extract DG_DATA.284
amdu -extract DG_DATA.283
amdu -extract DG_REDO.274
amdu -extract DG_REDO.273
amdu -extract DG_REDO.276
amdu -extract DG_REDO.275
amdu -extract DG_REDO.272
amdu -extract DG_REDO.271

```

运行以上脚本，就可以将相应的数据文件和日志文件从磁盘组中提取出来，然后将这些文件整合到一个统一的目录中。

通过配置好的参数文件和抽取的控制文件，可以将数据库启动到 mount 状态。具体过程如下。

```

export ORACLE_SID=d02
startup nomount pfile='/u01/d02/ora.pfile'
alter database mount;

```

接下来编写如下文件重命名脚本，将文件指向统一的存储目录。

```

alter database rename file '+DG_DATA/proda02/datafile/system.265.768047733'
to '/u01/d02/DG_DATA_265.f';
alter database rename file '+DG_DATA/proda02/datafile/sysaux.266.768047739'
to '/u01/d02/DG_DATA_266.f';
alter database rename file '+DG_DATA/proda02/datafile/undotbs1.267.768047743'
to '/u01/d02/DG_DATA_267.f';

alter database rename file '+DG_DATA/proda02/datafile/undotbs2.269.768047751'
to '/u01/d02/DG_DATA_269.f';
alter database rename file '+DG_DATA/proda02/datafile/undotbs3.270.768047753'
to '/u01/d02/DG_DATA_270.f';
alter database rename file '+DG_DATA/proda02/datafile/users.271.768047753'
to '/u01/d02/DG_DATA_271.f';

```

在这个案例中，由于文件和日志完好，数据库随后就可以成功打开。

对于特定的文件，通过以下测试可以验证 amdu 的恢复过程和文件完好性。

```
+DG_DATA/proda02/datafile/users.271.768047753'
```

通过 amdu 提取文件。

```
[oracle@oracle]$ amdu -diskstring '/dev/oracleasm/disks/DISK*' -extract 'DG_DATA.271'  
amdu_2012_02_22_02_02_09/
```

通过更名来重定向数据文件。

```
SQL> ALTER DATABASE RENAME FILE '+DG_DATA/proda02/datafile/users.271.768047753'  
to '/u01/amdu_2012_02_22_02_02_09/DG_DATA_271.f';
```

```
Database altered.
```

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

```
Database altered.
```

```
SQL> select name from v$datafile where name like 'DG%';
```

```
NAME
```

```
-----  
/u01/amdu_2012_02_22_02_02_09/DG_DATA_271.f
```

这个案例的幸运之处在于磁盘组未发生更为严重的损坏，数据文件和日志文件都是完好的，而 Oracle 的 AMDU 工具在这种情况下为我们提供了便利的恢复手段。

未雨绸缪，防患未然

宜未雨而绸缪，毋临渴而掘井。

——朱用纯《治家格言》

君子以思患而豫防之。

——《周易·既济》



在企业数据环境中，已经有很多 DBA 因为疏忽而遭受了惨痛的教训。通过了解和学习这些案例，可以帮助我们熟悉常见的错误种类和错误条件，进而增强规范、建立制度约束和技术防范，规避和减少技术风险，实现数据环境的安全运行和保障。

在数据管理阶段出现的问题，我们定义为管理安全范畴。在这一阶段，如果企业缺乏足够的管理规范，就可能出现很多安全事故，如误删除、Truncate 数据表等。

篇首图列举了管理安全范畴中可能涉及的诸多方面。

在本篇中，我们将主要描述一下各行业 DBA 遭遇的管理安全问题。从这些现实案例中总结经验，吸取教训。从灾难中学习，进而未雨绸缪，防患于未然，这也是每个企业数据运维人员的职责所在。

DBA 四大守则

通过分析所经历和遭遇到的各种案例，我曾总结出 DBA 生存四大守则，用于提醒 DBA 们在工作中应当注意和遵循的内容，这些守则直至今天仍然具有其现实意义。

以下是我认为极其重要的、需要铭记于心的 DBA 生存守则。

1. 备份重于一切。

我们必须知道，系统总是要崩溃的，硬盘总是要损坏的；没有有效的备份只是在等哪一天死！我经常开玩笑地说，唯一会使 DBA 在梦中惊醒的就是：**没有有效的备份**。

如果你睡前想一想，“那个没有备份的数据库，如果今晚硬盘损毁，明天如何去恢复数据和业务？”，如果真的没有备份，恐怕没有 DBA 能够安然入睡。

2. 三思而后行。

Think thrice before you act.

任何时候都要清楚你所做的一切，否则宁可不做！

有时候一个回车、一条命令就会造成不可恢复的灾难，所以，你必需清楚确认自己所做的一切，并在必要时保护现场，保证所做操作至少不会使事情变得更糟。

我们已见过太多草率的操作导致的数据灾难。

3. rm 是危险的。

要知道在 UNIX/Linux 下，这个操作意味着你可能将永远失去这个命令后面的东西，所以，请确认你的操作！

已经有太多的人在“rm -rf”上悲痛欲绝。当年写下这条守则时是一个凌晨，一个朋友将我吵醒，说他误操作 rm -rf 删除掉了 200GB 的数据库，并且没有备份。

我当时能告诉他的只有一句话：要保持冷静，离开键盘，然后让你的领导知道这件事。

作为一个技术人员，当事情超出了你的掌握，最好的决定就是让更多的人知道这件事，通过共同的决策来制订恢复方案，防止因为个人的再次判断失误造成进一步的损失。

`rm` 是危险的，以此类推，在数据库内部执行 `DROP/TRUNCATE` 等破坏性操作时，同样应当谨慎，小心一万次也不多，疏忽一次就可能致命。

4. 你来制订规范。

良好的规范是减少故障的基础。所以，作为一个 DBA，你需要来制订规范，规范开发甚至系统人员，这样甚至可以规避有意或是无意的误操作，减少数据库的风险。

而作为企业数据环境的管理人员，更应该从管理、流程上制订规范，防止因为管理流程不当导致的数据安全、数据灾难出现。

我们知道，在管理良好的数据库服务器上，`rm -rf` 甚至都可能是不允许使用的。

我们需要遵守的可能更多，所以我一直强调 DBA 一定要严谨专注，在管理数据库的同时，要承担起数据责任，不能有丝毫的马虎和大意，草率的判断和轻忽的选择对数据来说很可能是致命的。当然我也非常喜欢另外一句话：坚韧卓绝之人，必能成就万事。如果有兴趣、有毅力，就一定能够做好数据的守护者。

以上四大守则，愿与诸位 DBA 朋友共勉。

DBA 守则外两则

在《Oracle DBA 手记 2》一书中，我的朋友、作者之一郭岳，曾经提出了几点 DBA 守则，我深以为然，并对其中几点略作引申，收录在这里供大家参考。

1. 高度的信息安全意识。

本书的宗旨是安全，希望能够对大家的数据库安全起到帮助。在数据安全的诸多方面中，信息安全是核心环节。DBA 或相关技术人员能够接触到核心数据（虽然各种技术手段不断发展衍生出来用于防范 DBA 的数据获取），但是接触也就意味着责任，权利与责任从来都是如影随形的。能够访问获取数据，并不意味着你就可以去利用和传播这些数据，DBA 的职业道德要求你仅仅完成责任范围内的数据库维护，对于业务数据应当敬而远之，视而不见。

在 2011 年，一则新闻触动了很多数据工作者的神经：陕西 1400 多万的手机用户信息被泄露，不法分子利用这些信息以短信推广等方式非法牟利。对于运营商来说，保有用户信息也意味着责任，如果不能保障这些数据的安全，就可能被不法分子利用，侵犯公民的个人隐私甚至经济利益；而对于数据工作人员，接触数据的同时也要遵守职业道德和法律法规，严格避免不必要的信息泄露和传播。

以下是这则案件的摘要信息。

根据 2011 年 09 月 23 日的新闻报道，西安警方破获了全国首例非法出售、获取公民个人信息的系列案件。

据了解，这一系列案件导致陕西省近 1400 万手机用户的个人信息被泄露。

根据最后的破获结果，犯罪嫌疑人是一家科技公司的技术人员，他代表公司负责研发和维护陕西省某通信公司的计费经营系统。

2011 年 3 月以来，他利用工作便利，多次侵入这家通信公司的用户数据库，盗取手机用户个人信息。

2011 年 4 月，他应另一嫌疑人的要求，再次侵入某通信公司客户数据库，窃取了西安、咸阳、铜川等 7 个地市 1394 万手机用户的个人信息。

据警方介绍，这 1394 万手机用户的个人信息，占全省手机用户的 60%~70%。这些信息被非法交易的背后，是利益的驱动，使得 4 名犯罪嫌疑人形成承上启下的非法利益链条。警方同时指出，

该案件的侦破，也暴露出通信运营商和为通信运营商提供技术支持的科技企业，都存在监督职责的缺失。

律师认为，盗取手机信息的技术人员，首先违反了民事法律规定，需承担相应民事赔偿责任。刑法修正案中有涉及泄漏、盗取私人信息的相关规定：非法泄露个人信息，最高可判三年有期徒刑。

国家机关或者金融、电信、交通等单位的工作人员，违反国家规定，将本单位在履行职责或者提供服务过程中获得的公民个人信息，出售或者非法提供给他人，情节严重的，处三年以下有期徒刑或者拘役，并处或者单处罚金。窃取或者以其他方法获取上述信息，情节严重的，依照前款的规定处罚。

以下文字节选自《Oracle DBA 手记 2》。

运营商最为看重的事情是什么？如果我问你，你对运营商最不满意的是什么？或许你会说，是网络质量，是资费，是服务，等等。其实这些都不是运营商最为看重的事情。假设你的通话记录会被泄密，我想，即便这家运营商的网络质量好到你在海底 10000 米都能流畅通话，资费低到打长途比市话还便宜，服务好到给你配个一对一服务的美女客户经理，你还是会选择离开这家运营商。

既然客户这么看重信息安全，那么，运营商最为看重的，必然也是信息安全。作为运营商的 DBA，由于工作的需要，能接触到很多核心的信息，例如，运营商最终用户的身份信息，什么姓名、家庭住址，甚至是身份证号码，还有通话的详单等等。人都是好奇心的，当你开始做运营商的 DBA 时，你会发现，以前怎么都不可能得到的信息，现在，只需举手之劳就能偷窥到。可能仅仅是为了在朋友面前证明下自己多么强大，多么厉害，赢取朋友羡慕的眼神，然后，你就偷窥了这些信息。记住，这样的事情，不仅仅会毁了你自己的 DBA 职业生涯，甚至会毁掉你的公司。

当经历了无数次的信息安全泄密事件的通报，经历了领导的一次一次的安全教育以后，你已经不会再主动地为了赢取朋友羡慕的眼神而去偷窥不该看到的信息，这个时候，最为严重的可能就是无意的泄密了。

关于信息安全泄密的事件，在我工作的运营商那里，有两个典型的案例，一次发生在我去那里干活之前，另外一次，很巧合地发生在我离开那里之后。

先说说第一次吧，在我去那里干活以前，一家厂商的工程师，一时迷糊窃取了用户的一些信息泄密给外面很有名的调查公司。在查出来以后，他受到的惩罚是开除、赔偿，同时，去其他任何一家 IT 公司，都会被原来公司发律师函，说明其为什么被开除。我相信，从此以后，他再也不用从事 IT 行业的工作了。

另外一次，是某技术人员在网上论坛发帖，为了演示某个 SQL 功能，查询显示了部分人员的姓名、手机号码和 E-mail 地址等信息。这引发了一次安全危机，以致动用了大量的人力物力去追查。

追查的结果是一个做系统监控的工作人员，无意中泄露了信息，我不知道最后会如何处理这个人，但是估计不会很轻。

其实这样的泄密，一看就是无意识的。很多时候，DBA 就是要对无意识的事情，保持高度的警惕。为了这样的事情，影响到自己的职业生涯，是绝对不划算的。建议 DBA 在发布任何会有第三方知道的文档之前，先问问自己，这份东西会导致泄密么？多问自己几次，多确认几次，如果不能确认，那就和你的主管确认吧。我还记得，以前我们有一条铁的规定，非项目经理及以上级别，不得公开对甲方发布任何技术文档及承诺。这条规定就很好，可以避免很多不必要的麻烦。

2. 忘记你的系统有备份。

虽然我曾经反复强调，备份重于一切，但是我不得不承认，有时候，忘记你的系统有备份的确是一个重要的提示。

在本书部分案例中，客户存在备份，但是由于数据量庞大或者空间有限，使用备份进行恢复的成本很高或者不现实，于是客户不得不选择采取一些较为极端的方式来进行异常修复。

而很多 DBA 之所以犯下错误，也正是因为觉得别人做过备份，已经有了备份。这些想当然的前提在故障出现之后可能不再成立，而这正是很多经典故障的根源。

所以，我非常欣赏郭岳在书中提出的观点：**在有些时候，忘记你的系统有备份。**

很多人，看到这个标题，可能觉得十分诧异，作为一个 DBA，需要遵守的守则中的第一条，就是要备份。

在 eygle 的 DBA 四大守则中的第一条，写的就是备份重于一切，并且很明白地说明，唯一能使 DBA 半夜惊醒的事情，就是系统没有备份。首先不得不说明，我完全同意这个观点，但是在维护电信运营商系统的时候，请你一定要忘记你的系统是有备份的。

我提出这个观点，基于以下两个原因。

首先，对于运营商级别的系统，数据量之大，如果没有到达非常让人崩溃的情况（例如，生产系统崩溃，容灾系统无法启动，并且应急系统也无法运行这样极端），是不会去采用恢复数据的方案的，因为数据量太大，恢复的时间太长，而运营商是要求业务能运行为第一要素的。

其次，人都是有依赖性的，当你知道你的系统有备份的时候，你的操作就开始不那么如履薄冰了，就开始毛躁了，因为你的潜意识会认为，反正我的系统有备份，大不了恢复回来。

忘记你的系统有备份吧，忘记它吧。

可能是我个人的习惯，如果一项特性对于我来说是绝对弊大于利，比如 RAC 的 load balance 这

样的特性，我一般会选择忘记它。记得自己的系统有备份，对平时的工作没有任何的帮助，只会让自己的潜意识操作不那么精细，不那么谨慎，所以，请忘记系统有备份。

但是，请不要忘记，定期检查你的备份是否有效！如果你连定期检查系统的备份是否有效也一起忘记了，那就有点过犹不及，得不偿失了。嗯，忘记你的系统有备份，但是别忘记定期检查系统备份的有效性。

各种惨痛的案例

我们曾经在网络上收集过大家曾经犯过的错误信息，这些信息千奇百怪、五花八门，对于管理者，了解一下这些错误有助于制定规范、防范错误；对于 DBA 则有助于吸取教训，增长经验，防患于未然。

对于这些案例，我多数保留了原始的描述和记录，做了一些简单的编辑整理和分类工作，这些案例收集自网络，与大家共为警醒。

系统级误删除案例

以下一组案例与删除有关，在操作系统上执行 `rm` 命令应当非常谨慎，重要的生产环境，应当将 `RM` 命令进行别名重定义。在我的 DBA 生存四大守则里，“`rm` 是危险的”就是其中非常重要的一条。本书中就有多个案例与 `RM` 删除数据文件有关。

在这些案例之前，我想引用一下在程序员的世界里，2011 年影响深远的的一个 BUG，这个 BUG 被戏称为“一个空格引发的惨剧”：

bumblebee 是一个开源项目，这个名字也就是变形金刚里的大黄蜂，这个项目是这样介绍自己的。

bumblebee is Optimus support for Linux, with real offloading, and not switchable graphics.. More important.. it works on Optimus Laptops without a graphical multiplexer..

Optimus 是 NVIDIA 的“优驰”技术，其可以将您的笔记本电脑 PC 提升到绝佳状态，提供出色的图形性能，并在需要时延长电池续航时间。这个项目是把这个技术移到 Linux 上来。

这个项目本来不出名，不过，程序在其安装脚本 `install.sh` 里的一个 bug 让这个项目一下子成了全世界最瞩目的项目，这个 bug 的 fix 如下：

```
@@ -348,7 +348,7 @@ case "$DISTRO" in
- rm -rf /usr/lib/nvidia-current/xorg/xorg
+ rm -rf /usr/lib/nvidia-current/xorg/xorg
```

（引自 <http://coolshell.cn/articles/4875.html>）

注意到么，在 `rm -rf /usr` 后面多了一个空格，这会导致用户的 `/usr` 下所有文件被删除，一个简单的空格造就了一个伟大的 BUG，并且引发了全球数以万计的程序员前往围观留言。

而关于这样一个空格引发的灾难在数据库 DBA 的世界里，也层出不穷。

根据这些灾难和经验总结，我们建议所有重要的数据环境中可以参考以下建议：

1. 通过别名或重定义方式提示或禁用 `rm` 操作

或者制定一个规范，通过 `mv` 的方式进行文件转移，通过一定时间段（如一周）观察无误后，再彻底清除数据文件。

`rm` 操作的危险性必须得到技术人员的充分重视。

2. 加强数据环境的空间监控

很多用户是在空间达到 100% 之后才去匆忙进行空间清理，匆忙常常会带来考虑不周，误操作等意外发生。

所以我们建议加强数据环境的存储空间监控，不要等到 100% 再去应急，应当总是使空间留有馥量，提前进行空间维护，避免手忙脚乱的应急处理。

3. 在紧急删除之前做好备份

如果不可避免的要进行紧急的文件删除工作，那么在条件允许的情况下，应当做好备份转移到其他主机或存储，避免无法回退恢复的灾难。

通常文件的转移并不会花费太多的时间，在可能情况下用转移替代删除，在必须删除时，也要考虑能否保留最后一个备份。

4. 避免在持续工作或者凌晨仓促的进行文件删除等工作

人在疲劳和不清醒的状态下极易犯下错误，所以应当尽量避免在连续工作的疲惫状态下，或者在梦中惊醒的凌晨迷糊状态下进行维护工作，比如文件删除，在这种状态下，极易出现误判，造成误操作。

另外，在操作之前确认你的当前路径，很多灾难是由于当前路径错误导致的，在 Unix/Linux 下，可以通过 `pwd` 命令来确认。

5. 重要的操作实现人员备份

前面提到过这点建议，再次重申，在执行重要操作时，最好有两个人同时在场，互相监督审核，避免一个人草率或者考虑不周导致的误操作。

这些系统级别的误删除操作，导致了一次又一次的数据灾难，这些灾难极其相似，也极其惨痛，我们每个人在学习之后都不应该再犯下这样的错误。

这些案例中，也有很多空格在一些 DBA 的职业生涯中刻骨铭心：

案例概述	案例详情
误删除 Oracle 软件	硬件维护人员删除归档日志的时候,把节点 2 的整个 ORACLE_HOME 都删除了。 ...在删除的时候没有注意到目录改变了,还键盘做了一个向上的动作,刚好就是刚刚使用的 <code>rm -rf *</code> , 然后一个下意识的动作回车就这么按下去了
误删除 Oracle 软件	曾经犯过 <code>update</code> 和 <code>rm</code> 两个错误、哎~~~, <code>update</code> 了一大片数据, <code>rm</code> 了整个 <code>\$ORACLE_HOME</code> 目录,后者那个错误是在测试库,前者那个错误是在生产库
误删除 Oracle 软件	<code>rm -rf /opt/ora92/*</code> 在测试库中本来想删除数据库,结果错误的把 ORACLE 软件删除了。郁闷啊,幸好不是生产库
误删除所有数据文件	在 linux 平台上,一次不小心操作,把 <code>oradata</code> 下所有的东西全删除了。至今铭刻于心
误删除软件及数据	不小心用 <code>rm -rf /home</code> 删除了目录下的所有文件 <code>/home</code> 目录下放的账务系统的 app。一看删除的路径的不对,已经来不及了
误删除所有文件	一次在一个目录下发现有个*开头的文件,就写了个 <code>rm -rf *</code> 后缀,结果可想而知吧
误删除软件及数据	在生产环境目录下,执行 <code>rm -R *.*</code> ,结果数据库、应用全部 over。停机 5 天,叫了 N 个人才搞好

空格导致的误删除

我最难忘的: root 用户 在根目录下 `rm -rf abc *`,
abc 和*之间有个空格, 结果把 os 删除了。

已经成为佳话。什么事情都可能发生的。

从此, 整个人好像变了一样, 做什么事情, 都三思而后行了

空格导致的误删除

在测试环境下遇到过在删除一些临时文件时, 在
* 和 .tmp 中间多加了空格. 后果很严重啊! 想想
都后怕,如果是在生产环境下, 一身冷汗!

```
rm *.tmp
```

空格导致的误删除

列举一次绝对毁灭性的操作:

重建表空间后文件系统里有些数据文件没有用了, 我打算清理空间本来语句类似是这样写:

```
rm -rf ts_tab_test*
```

由于网络有延迟, 导致了手欠, 多敲了一个空格,
写成了

```
rm -rf ts_tab_test *
```

结果这个目录下所有的数据文件都被我删除了, 绝对崩溃

空格导致的误删除

偶的教训不是很深刻, 不过意义很重大:

删除一些 trace 文件, 然后就直接删除 `rm orcl*`,
结果通过 vpn 到生产的, 网络太慢, 命令刚刚慢慢的
显示出来, 看都没看直接按回车, 结果执行的命令却是
`rm orcl *`, 因为 orcl 和星号中间有个空格, 所以把这个
目录下面所有的内容全部删除了。出了一身冷汗,
试想, 如过是删除数据文件目录下的内容, 那立马死翘翘了

到现在为止, 每次都要等命令完全显示出来, 从

头到尾看一遍再执行。不过，大多数错误都是在很繁忙或者很疲劳的情况下发生的，呵呵，看来 dba 应该多休息才是

空格导致的误删除

删除日志时，输入 `rm *.log`，因为星号*与 log 之间有空格，所以删除了该目录下所有的文件，吓得我出了一身汗

空格导致的误授权

安装数据库的时候 `su - chmod 777 -R /oracle`，多输入一个空格

变成 `chmod 777 -R / oracle`，许多系统文件属性变坏，Unix 瘫痪

这个错误犯了两次，用系统恢复磁带重做系统，幸好是测试机。

从此以后系统部门的同事□不肯给□roo□口令

误删除日志文件链接

我所經歷的.

RAW 磁盤的链接文件，整理的時候，下 RM 命令加*，把正在運行的一個實例的 REDO LOG FLIE 的連接文件刪除了。當時就感覺出錯誤了。那個實例當時還沒有 DOWN。還來切換 REDO LOG 時候找不到文件就 DOWN 了。幸好 RAC 其他實例正常運行，用戶和其他部門都沒有感覺到。

還來把那個連接文件重新建立，又可以啟動了。
自此下 RM 命令很小心

误删除数据文件

當時，那幾天都是很疲勞的。在開發環境作數據文件分佈調整時，先 cp 完某個表空間所有文件到其他地方，然後作*匹配 rm 了此表空間在此目錄的數據文件。但是 rename 時發現居然有一數據文件沒 cp 過來，忘了說了，此表空間是 system 表空間。沒辦法，開發人員明天還要使用這個環境。幸虧之前有一備份，不

過當時磁盤空間不是很充裕，足足折腾了一夜才搞定！想起來都後怕哦，幸虧不是正式環境了！再以後，就很少用 cp,rm 了，特別是 rm *..,一般是此類操作用 mv 來完成。需要 rm 的東西，一般 mv 到一臨時目錄了，再 rm 了！呵呵，可能都有點謹慎過頭了哦

脚本中误删除文件

自己写了个 rman 备份以及备份成功后 rm 旧 log 的 shell 脚本，log 的目录赋值給变量，结果执行時目录赋值沒成功，該变量指向了另一個目录，结果下面的东东全没了，系统立即报错（把用户的 home 目录删了）。幸好当时头脑还很清醒，也没误删什么重要的数据，很快就搞定了。以后脚本中要 rm 某个目录的东东再也不敢用变量表示了，直接 hardcode 进去算了，这样也放心。另外出问题后一定要冷静，定位出问题原因后再动

误删除目录中挂载

一次生产环境 linux 系统,做整个项目目录的移植,cp 一份确认正常执行后直接 rm 原来的目录,没想到,子目录中居然有 mount 到其他 server 的 XX 目录,结果可想而知...linux 啊.....

误删除自动匹配文件

由于 linux 下的 tab 可以自动出来文件名。

一次快速操作，直接 rm -rf 文件时，用 tab 键出来文件名，没有仔细看，直接把文件都删除了，吓了一跳。

后来一 pwd 看，幸好是/root/目录下。不然，要是把整个/给删除了，那就麻烦了

误删除数据文件

刚进现在的公司不久时,做一个数据仓库项目,同事周日加了一天班把数据抽到一个大表空间里,大概 100G,第二天因为临时表空间增长很快,决定重建,这个临时表空间的开头和那个大表空间名字是一样的,只是后面加了一个_temp,当时也是因为事情比较多,认为这是很简单的,结果输入名字就忘了输入_temp,把大表

空间删除了,同事白加了一个星期天,虽然没影响什么进度(数据可以重抽),但这次教训是深刻的.

个人教训:

1.rm 的时候一定不要用*之类的,要用的话要看好再用,否则会有意想不到的效果.

2.人在累的时候最容易出错误,所以每一次回车都要看好.

数据库误删除案例

有些威胁来自数据库外部,而有些威胁则来自数据库内部,对于数据库外部,破坏性的操作有 rm ,而在数据库内部,同样有破坏性操作,如 Truncate.

分析总结以下的种种灾难,我们做出以下建议:

1. 通过触发器约束或禁用特定的 DDL 操作,防范数据库风险

对于 TRUNCATE 等高风险的数据库 DDL 操作,可以考虑通过触发器进行禁用,防止未授权的操作损害数据。

很多轻忽的数据灾难都来自于 Truncate,这个类似于系统级别的 rm 命令极具破坏性,而且 DDL 不可以回退,即便发现已经为时过晚。所以我们建议用户可以考虑使用 DDL 触发器来禁用 Truncate 之类的危险操作,以达到安全防范的目的。

2. 严格进行权限管理,以最小权限原则进行授权

过度授权即是为数据库埋下安全隐患,在进行用户授权时一定要遵循最小权限授予原则,避免因过度授权而带来的安全风险。

3. 明确用户职责,加强用户管理

应当明确不同的数据库用户能够用于的工作范围,应当使用普通用户身份的,就绝对不应该使用 DBA 的用户身份,只有职权相称,才能够避免错误。

即便是拥有管理员职责的用户,也应当遵循以不同身份执行不同任务的习惯,比如 SYS 和 SYSTEM 用户的使用就应当进行区分和界定。

4. 在任何数据破坏之前进行备份

在进行数据表的截断、删除之前，进行备份，将备份养成一种习惯，这样才能够避免误操作之后的措手不及。

5. 以重命名代替删除操作

不论操作系统级别还是数据库级别的删除操作，尽量以重命名替代删除，如重命名数据表，重命名数据文件，然后通过一段时间的观察和确认后再彻底删除。

Oracle 10g 中引入的回收站功能，就是将我们执行的 DROP 操作变更为重命名进行保护，当我们发现了失误之后，可以通过回收站找回，但是注意回收站保存对象的时间和空间有关，如果存储空间不足，对象会被自动释放。

我们在管理中借鉴这个回收站思想是很有帮助的。

6. 尽量争取充足的时间

不要低估任何一次简单的维护操作，因为一个意外就可能大幅延长你的维护时间。所以，应当尽量争取充足的时间，包括做好充足的准备工作，加快无关紧要步骤的执行，减少不必要的时间消耗，时间越充裕，你用来应对可能出现的故障的时间就越多。

7. 审核你的剪贴板

很多错误是由于粘贴剪贴板的内容引起的，所以，当你准备向一个窗口或者命令行粘贴你看不到的内容时，提高你的警惕性。在 Windows 上，有很多剪贴板增强工具，可以帮助我们记录和展现剪贴板的内容，可以考虑选用。

审核你的剪贴板，确保其中的内容是你期望的。

8. 没有认真看过的脚本就绝不要执行

对于 DBA 来说，如果一个脚本你从来没有认真读取了解过，就不要去执行，脚本中的一个错误就可能导致严重的数据灾难。我们遇到过案例，由于脚本中的一个变量错误，导致所有数据文件被删除，教训惨痛。

如果实在无法审核脚本的内容，那么在进行重要操作之前，备份你的数据。

通过触发器实现 DDL 监控

关于建议 1 中提到的触发器监控，以下我们将做详细的介绍和说明。

如下触发器实现对于特定表的 DROP、TRUNCATE 防范：

```

CREATE OR REPLACE TRIGGER trg_dtdeny
  BEFORE DROP OR TRUNCATE ON DATABASE
BEGIN
  IF LOWER (ora_dict_obj_name ()) = 'test'
  THEN
    raise_application_error (num      => -20000,
                             msg       =>  'You Can not Drop/Truncate Table '
                                           || ora_dict_obj_name ()
                                           || ' Pls check you plan.'
                             );

  END IF;
END;
/

```

如果用户试图对 test 表进行 DROP 或 TRUNCATE 操作，则将遇到错误：

```

SQL> truncate table test;
truncate table test
      *
ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20000: You Can not Drop/Truncate Table TEST Pls check you plan.
ORA-06512: at line 4
SQL> drop table test;
drop table test
      *
ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20000: You Can not Drop/Truncate Table TEST Pls check you plan.
ORA-06512: at line 4

```

以下触发器可以实现全库级别的 DDL 防范：

```

create or replace trigger ddl_deny
before create or alter or drop or truncate on database
declare

```

```

l_errmsg varchar2(100):= 'You have no permission to this operation';
begin
  if ora_sysevent = 'CREATE' then
    raise_application_error(-20001, ora_dict_obj_owner || ' '
      || ora_dict_obj_name || ' ' || l_errmsg);
  elsif ora_sysevent = 'ALTER' then
    raise_application_error(-20001, ora_dict_obj_owner || ' '
      || ora_dict_obj_name || ' ' || l_errmsg);
  elsif ora_sysevent = 'DROP' then
    raise_application_error(-20001, ora_dict_obj_owner || ' '
      || ora_dict_obj_name || ' ' || l_errmsg);
  elsif ora_sysevent = 'TRUNCATE' then
    raise_application_error(-20001, ora_dict_obj_owner || ' '
      || ora_dict_obj_name || ' ' || l_errmsg);
  end if;

exception
  when no_data_found then
    null;
end;
/

```

在以下类似操作中，触发器的作用就体现出来：

```

SQL> create table eygle as select * from dual;
create table eygle as select * from dual
                                *

ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20001: EYGLE.EYGLE You have no permission to this operation
ORA-06512: at line 5
SQL> drop trigger trg_dropdeny;
drop trigger trg_dropdeny
*

ERROR at line 1:

```

```

ORA-00604: error occurred at recursive SQL level 1
ORA-20001: EYGLE.TRG_DROPDENY You have no permission to this operation
ORA-06512: at line 11

```

对于某些数据库环境，也可以限定 DDL 操作只能在数据库服务器本地执行，对于远程执行则予以禁止，类似的触发器可以参考如下代码，以下代码基于 Schema 模式建立，需要对于 V\$SESSION 的访问授权，自定义的记录信息被写入告警日志文件：

```

SQL> connect / as sysdba
SQL> grant select on v_$session to eygle;
SQL> grant execute on dbms_system to eygle;
CREATE or replace TRIGGER ddl_trigger
  before ddl on eygle.schema
declare
  n          number;
  str_stmt   varchar2(4000);
  sql_text   ora_name_list_t;
  l_trace    number;
  str_session v$session%rowtype;
BEGIN
  select count(*)
    into l_trace
  from dual
 where utl_inaddr.GET_HOST_ADDRESS is not null
    and sys_context('userenv', 'ip_address') is not null
    and sys_context('userenv', 'ip_address') <>
      utl_inaddr.GET_HOST_ADDRESS;

  if l_trace > 0 then

    n := ora_sql_txt(sql_text);

    for i in 1 .. n loop
      str_stmt := substr(str_stmt || sql_text(i), 1, 3000);
    end loop;

```

```

select *
  into str_session
  from v$session
 where audsid = userenv('sessionid');

sys.dbms_system.ksdwrt(2,
                      to_char(sysdate, 'yyyymmdd hh24:mi:ss') ||
                      ' ORA-20001 user: ' || user || ' program: ' ||
                      str_session.program || ' IP: ' ||
                      sys_context('userenv', 'ip_address') ||
                      ' object: ' || ora_dict_obj_name || ' DDL: ' ||
                      str_stmt);

raise_application_error(-20001,
                      'DDL is deny from remote connection.');
```

end if;

END;

/

此时如果通过远程进行 DDL 操作，就会收到错误：

```

C:\>sqlplus eygle/eygle@orcl
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> drop table eygle;
drop table eygle
*
ERROR at line 1:
ORA-00604: error occurred at recursive SQL level 1
ORA-20001: DDL is deny from remote connection.
ORA-06512: at line 37
```

在告警日志文件中会同时记录移行错误信息：

```
Wed Feb  1 14:02:43 2012
20120201 14:02:43 ORA-20001 user: EYGLE program: sqlplus.exe IP: 192.168.0.104 object:
EYGLE DDL: drop table eygle
```

在本地的 DROP 操作可以进行：

```
SQL> connect eygle/eygle
Connected.
SQL> create table eygle as select * from dual;
Table created.
SQL> drop table eygle;
Table dropped.
```

以上是一些示范，供参考，在系统中采用需要经过测试和改进。

因为 DDL 的重要性，在 Oracle 11g 中，DDL 日志机制被引入，可以通过 enable_ddl_logging 参数设置，如果启用日志，DDL 操作的信息都将被记录到告警日志中：

```
Fri Feb 17 17:33:14 2012
ALTER SYSTEM SET enable_ddl_logging=TRUE SCOPE=BOTH;
Fri Feb 17 17:33:25 2012
create table eygle as select * from user$
```

在 Oracle Database 12c 中，为了防止 DDL 日志对于告警日志的干扰，DDL 日志进一步的独立出来，记录了一个独立的 DDL 日志。

```
oel*orcl12c-/u01/app/oracle/diag/rdbms/eygle/eygle/log/ddl$ ls -l
总用量 8
-rw-r----- 1 oracle oinstall 4235  2月 17 17:32 log.xml
```

以下是日志中记录的 DROP 表操作：

```
Fri Feb 17 17:32:27 2012
diag_adl:drop table eygle
```

由此可见，DDL 审计和记录是众多 Oracle 数据库用户的需求，最终 Oracle 做出了改变。

以下是一些来自于数据库内部的误删除操作系列案例。

案例概述	案例详情
误删除字典表	昨天太大意了，一不小心把 file\$中的内容给删除了，造成 tablespace 里的数据文件列表看不到了
误删除数据表	一次误删了个表,最后恢复了,丢了一天数据.加了一晚上班,至今记得.人越累的时候就更容易犯错误,我就是在最后快下班的几分钟犯的错误
误截断数据表	我最惨，有一次把一个表一不小心给 truncate 了，上千万条记录一眨眼就没了；提心吊胆的陪了 3 天也没有把这个表搞定；最后不了了之了
误删除字典视图	我 drop 掉过 view sys.v\$sql,然后迅速重新 create 了。 还有 alter system set shared_pool 降低 shared_pool 的 size, 等几分钟没结束，只好 ctrl+C, 但是后台 session 还在，导致 CKPT 占用整个 CPU, 两个小时后发现，kill session 才恢复正常
误删除数据表	原来接手一个部门的所有数据库，结果漏了一个，也没人告诉我，所以我不知道这个数据库存在。一天一个程序人员误按了一个按钮，把大量的数据全部删除，找到我后，发现数据库没有归档，也没有任何备份。结果是程序人员补了几天的数据，我的奖金也直接泡汤
误删除表数据	有一次把删除操作和 commit 全部写在一个脚本里了，执行后才发现删错了，幸好 10g 有个闪回功能
误删除表数据	我的错误是在一个 2 亿条记录的表里删除一些错误数据，结果条件写错一个，误删无数的正确记录，幸亏我有早上 7:00 才备份好的备份，将数据倒回去，损失一丁点数据而已

误截断数据表

生产和测试的窗口一定不能同时打开！有一次把测试环境用的一溜 `truncate` 语句在生产环境执行了

误删除用户

刚从事 DBA 不久，可已经犯了个让我终生难忘的错误。原本是要将测试环境的一个 user 给删掉，由于桌面上开了多个窗口，结果 `drop user XXX cascade`，直接将正式环境的一个 user 给 drop 了，刚按下 enter，就感觉怪怪的，心想不会吧!! 已经 来不急了，还好这个 user 的信息是从另外一台服务器上同步过来的，要不然死定了。以后做什么动作我都习惯先看看是在哪个 DB 上那个服务器上，千万别搞错了

误删除用户

在一次测试过程中，把一个在本机执行的删除所有非系统用户的脚本，错误的粘到一个开发数据库的 sqlplus 窗口中。幸好在 30 秒内就意识到了错误，及时中止了脚本的运行，只删除了一个无关紧要的用户

误删除用户

把一套 rac 的其中一个节点的 oracle 用户删除,rac 那时候正在跑。。。刹那间心都碎了的

误删除表数据

以前公司，有一个程序员写好的脚本，一个实施人员去执行，脚本里面带了

```
delete * from xxx;
```

```
commit;
```

啥备份，归档都没有。

结果我们公司全部人员出动，抱着笔记本，台式机，去北京某区县所有的机关单位上门录了一星期人员信息。

至今记忆犹新

误截断数据表

开了多个窗口，把生产库里面最重要的表 `truncate` 了，本来想 `truncate` 测试库的。

这下子，惨了。。。还没备份。。。一个礼拜都不好意思

误截断数据表

第一件：想 truncate 测试库的数据，结果成正式库了。

第二件：晚上迷糊糊的把 DG 的主库写成 READONLY 了。最后第二天业务系统登录不进去了都（幸好是测试阶段）

第三件：想把某一普通用户下的表全部删除，结果登录的是 SYSTEM。。。后果可想而知

误删除表数据

一个 db2 数据库,前台程序挺烂,经常需要我 delete from table where 操作。

那天早上一到，还没进入工作状态，接了个电话需要删除些记录，晕了头了后面忘记加 where 条件了，

一个关键表数据没了，紧接着电话一个接一个的打来。还好生产库—查询库是做了数据复制的，几分钟后把数据同步回来了。

如果再晚上几分钟查询库同步一次数据也没了。后来想想还是很担心，再后来没有出过大错误了

误删除数据表

测试环境导出的脚本中包含 drop 语句，结果看都没看就直接在生产环境中做了，一下子物料表就没了，整个生产停线，后来做了恢复，丢了半天的数据。 教训：执行的脚本一定要认真检查

误删除数据表

很疲劳快下班的最后几分钟，连续 DROP 了 4 个表。我那时刚毕业，一下就傻了。好在我的领导在一旁一直鼓励我，安慰我。陪我加了一宿的班。至今想起这个事情都后怕，至今想起这个事情心里都是充满了对那位领导的感激

误删除数据表

一个不小心把表给删除了,包括结构,郁闷啊

误截断数据表

看错数据库 `truncate` 了几个关键业务表,当时电话不停的在耳边响,幸好没乱了方寸。如果做不完全恢复代价太大,最后从 `log` 表中恢复了数据,没影响到生产,不过也出了一身冷汗,从此做任何操作都很小心

误删除数据表

因为我学的不深入,所以一般情况下还是很小心

但一次本是 `truncate` 一个表,但结果 `drop` 了,还好是一个临时存放数据的表,重建就可以了;

再一次,本来是在测试数据库删除一个 `snapshots`,结果眼花了,把正式的给删除,幸运的是只是 `SNAPSHOTS`,重建就可以了,但当时我是吓得一身汗,本来一天很困,但当时就完全清醒了。

所以后来我再做事的时候,累了就不做,不会把测试和生产的数据库同时开着,做操作时再三看□是否进对了数据库

脚本错误误删除

used to have a script written by someone else to run in default directoy, it will delete all the dump file, logs, etc, one day by mistake run it under `$ORACLE_HOME`... end up the binary was gone

luckily it was after work and dev environment, Call NOC to restore everything asap (within 1hr)...

lesson: **never run script if you donot read it carefully and know exactly what it is**

主备环境错误案例

很多企业，或者很多 DBA 养成的一个不良习惯是，他们常常忽视生产环境和测试环境，他们甚至不做环境校验就草率的执行任务，结果造成了很多不应该发生的灾难和错误。

针对这些情况，我们建议：

1. 测试环境和生产环境应当处于不可互通的物理网络

互通就意味着同时可以访问，也就可能带来很多意想不到的安全风险，企业应当将测试环境和生产环境部署于不可互通，或者不可同时访问的网络环境中，避免因为错误连接而发生的数据库灾难。

分离部署一方面可以降低误操作的可能性，也可以屏蔽一些无关的访问可能，从而从网络链路上保证数据安全。

2. 在执行任务之前确认连接访问的数据环境

通过查询数据库的视图（V\$INSTANCE,V\$DATABASE）就可以获得数据库的主机、实例名称等信息，在任何重要任务执行之前，都应当明确确认连接到的环境是正确的。

```
SQL> select instance_name,host_name from v$instance;
INSTANCE_NAME      HOST_NAME
-----
ORCL                hpserver2.enmotech.com
```

这应当成为 DBA 的习惯。

3. 避免打开过多的窗口以致操作错误

在执行任务时，保持尽量少的打开窗口，我经常见到工程师桌面打开众多凌乱的窗口，混乱与错误同行，尤其是在通宵加班等环境下。

保持简介清晰的工作界面，是一个工程师应当具备的基本素质。

4. 在执行重要任务时应保持良好的状态

良好的状态是高效率 and 高质量工作的保障，如果是夜间工作，应该保障充足的睡眠，以清醒的头脑面对重要的工作；并且一定要避免在疲劳状态下连续工作，疲劳作战是对自己和数据的不负责任。

5. 避免匆忙之下进行重要的工作或决定

很多误操作都是因为急着下班，急着回家，临门一脚导致的失误，所以当我们去执行一项工作时，应当保

持平和的心态，避免仓促紧急的决定。

从来匆忙和仓促都不是一个正确的方法。

6. 测试环境和产品环境密码设置不能相同

有些测试环境或者非产品环境是利用产品环境恢复得到的，DBA 在建立了测试环境后，就没有修改数据库用户的登录密码；经常性的，DBA 也习惯在所有环境中设置通用的密码；这些习惯为系统带来了很多风险和不确定性。

我们建议用户在不同环境中采用不同的密码设置，这是因为一方面产品环境和测试环境面对的访问用户不同，密码设置相同则意味着产品环境的安全性完全得不到保障；另一方面，DBA 登录到不同的数据库需要使用不同的密码，这进一步减低了 DBA 在错误的环境下执行命令的可能性。

以下是一系列因为弄混乱了生产环境和测试环境，出现的错误：

案例概述

案例详情

生产与测试环境错误

开了两个 PL/SQL DEVELOPE 窗口，一个生产的，一个非生产的，同名用户，同表空间名，结果非生产的建用户脚本在生产中跑了一下，非生产是 grant limit tablespace to XXX 的，在生产中跑了以后，生产中的用户变成 LIMIT 了，结果程序出错，表空间不足。

导致应用出错半个小时后才处理好。

这个太惨痛了，建议所有的使用多个环境的人，并且操作多个 PL/SQL DEVELOPE 的人尽量只开一个窗口操作，或者是操作生产的时候，用只读的查询用户

生产与测试环境错误

最严重的一个数据库错误是，在正式环境上执行了一个应该是测试环境的脚本。结果导致用户无法连接数据库

生产与测试环境错误

自己电脑装了 ORACLE 数据库,平时操作都在自己创建的库上.....经常删除用户,重新导最新的数据进去

那天也是快下班了.....急,直接删除用户,删除的

	时候还在想就算是正式库权限不够没关系,看也没看就敲回车了..... 后来不说了,两个字,郁闷
生产与测试环境错误	我有一次本来要删除测试库的,结果差点删除生产库的一个表的所有数据,还好强行 <code>ctrl_alt_delete</code>,最后回滚了,哈哈,居然一条数据都没有删除。确实是快下班,比较累。以后不能在心急的时候维护数据库
生产与测试环境错误	测试库导入数据,不小心把正式库给 drop 了
生产与测试环境错误	主机和笔记本的 oracle 服务名一样,连接错误,通过业务程序把数据库给初始化了,那个惨
生产与测试环境错误	我犯过的错误也不少,大大小小,已经都快记不清了。 很多情况是测试库环境跟生产库的不太一样,结果是在测试库中没有出问题的脚本,在生产库里出问题了。 所以在执行脚本时,还是要仔细检查一下,在正式库中的环境是否一致
生产与测试环境错误	也是开了多个窗口,一个窗口建库,另一个窗口是生产的库。搞错了,在生产的服务器上直接 shutdown 了,立刻电话就上来了。好在没有造成太大影响,也是提心吊胆的。多窗口危险很大
生产与测试环境错误	我一同事自己做了个测试环境,想把生产库的数据导入到他自己的 DB 里,结果不小心,干掉了生产库里好多表,直接导致产线停了 4 个小时
生产与测试环境错误	有次删除测试库的 USER,结果把正式库的给删除了
生产与测试环境错误	尤记得那年我还很冲动,测试环境中发现表空间不够了,就加了一个文件。一会有人打电话说生产库

总报一个提示。

马上去看，发现我的数据文件竟然加在生产库上！而且路径类似 windows 的，非常奇怪，冷汗！靠，原来写错 tns 串了，见鬼的是测试环境和生产环境网络竟然是互通的！生产环境是 rac，裸设备，9i……后来只好把这个本地文件脱机，数据倒没有丢失，但总有个删不掉的脱机文件！后来找个理由升级成 10g 了，我心里的石头才算放下了。

从此以后我从来没有犯错

生产与测试环境错误

有一次把 dmp 文件导到正式环境上了，应该是测试环境下的

生产与测试环境错误

开了多个窗口，把生产库里面最重要的表 truncate 了，本来想 truncate 测试库的。

这下子，惨了。。。还没备份。。。一个礼拜都不好意思

生产与测试环境错误

一哥们用 tar 包迁徙了生产库的 oracle 环境和数据到测试机，测试机和生产库是 vpn 连起来的，user 和 sid,服务名也都一样。

我就在测试机敲了 sqlplus / as sysdba，开始 dorp user cascade 和 imp。跑了一段时间后，

然后又开了个 sqlplus user/user@db,准备 drop 别的东西的时候，真是老天保佑!!! 看着这个和连生产库一样的连接串，让我当时灵光一闪，会不会连到生产库上了呢，

然后，select utl_inaddr.get_host_name() from dual;

结果赫然是生产库的主机名!!!! 回头看看头一个窗口已经 drop 完了 user 正在跑 imp.... 当然脑袋嗡的一

下就空白了，手脚发麻的感觉直到现在都在回味...，幸好头一个窗口偷懒用了 / as sysdba 连接，不然我就卷铺盖走人了。

后来一查，那哥们迁移完了后，tns 里没改....赫然是生产库的 ip 和 sid...

问之为啥没改，答曰。一是平常都用 / as sysdba。二是忘记了。。。。

上帝保护!!

我和他约法三章，以后测试环境和生产环境的 user, sid, 服务名都不能一样。又安排了一个计划，准备 12 月调整机房网络，单独让生产环境只和一个网段连，然后再用 vpn 连这个网段

误删除生产环境数据

有一次在测试库 drop 掉一个表，drop 完发现把生产库中的表给 DROP 了,1000 多万笔纪录啊.当时产线就停了，最后一级生产事故。偶公开检讨

教训：不能用 TOAD 同时打开两个以上的库

生产与测试环境错误

为了给员工做培训，我把正式库的数据导入到测试库上，但是不知道谁修改我机器上的盘点系统的配置文件，结果显示的是测试库,实际上是正式库,IMP 过程中其实已经表现异常了,但没意识到,幸好,这个 DMP 的时间点只差 2 个小时,当时的中间的操作也没多少

生产与备份环境错误

最惨的一次是和公司的一个哥们一起出差,那个哥们不知道出于什么考虑,将主服务器和备份服务器的 IP 反了一下,但是 tnsnames 没做修改,我准备重做备服的时候,使用了 drop user cascade,把所有的用户都干掉了一遍,刚刚干完,所有科室上夜班的护士小妹妹都给我打电话,说科室里的电脑全部不能用了,当时急的不行了,还好习惯还不错,来的前一天做了一个全库冷备，立

刻进行了恢复,不过也丢失了一整天的数据

业务高峰误操作案例

在维护生产环境时，尤其是负载极高的核心生产环境，我们需要注意的是，你的每一个操作，都可能导致系统负载波动，甚至产生严重的性能问题。

种种案例表明，我们应当严格遵守一下生产环境的维护守则，以避免不必要的业务影响和数据灾难。

1. 在高峰期禁止在数据库中进行 DDL 操作

DDL 操作会导致一系列的 SQL 重解析，依赖对象失效等数据库连锁反应，一旦 SQL 重解析集中出现，系统必然经历负荷峰值，如果系统繁忙，可能就此挂起；DDL 导致的依赖对象失效，甚至无法编译通过，可能长时间影响业务系统正常运行。

所以，在生产环境中，应当严格禁止高峰期的 DDL 操作，避免因为操作不当或考虑不周带来的手忙脚乱或数据库灾难。

2. 慎重进行统计信息收集和索引创建等

统计信息收集和索引调整是优化数据库的常用手段，可是切记业务峰值期间的统计信息收集，或者收集之后导致不可预期的执行计划改变可能使数据库瞬间停滞；而贸然添加的索引，也有可能导致其他 SQL 执行计划的恶化。

所以，在生产环境中，统计信息的收集或索引增减，都应当是非常慎重的，避免因为考虑和测试不周带来额外的麻烦。

以下这些案例就是一些贸然操作导致的数据库问题：

案例概述

案例详情

业务期间统计信息收集

客户业务系统上线后由于存在部分性能问题，我对一个表作了 dbms_stats....造成一个 sql（涉及多个大表）执行计划改变(性能特差)主机基本瘫痪了两个小时。最后给 sql 加 hint 才解决问题！

一个 sql 搞死一个数据库

业务期间 DDL 操作

2004 年一次下午 17 点左右, 在 schema A 下一个表上增加一个字段(对于在 schema A 范围来说这个字段增加当时是不会有问题的), 一加上, 系统 load 立即狂飙……结果在 schema B 下有一个包, 里面有引用 schema A 的这个表, 没 check 倚赖关系以为 A 和 B 之间没有联系, 结果这个包编译不过去被大量进程尝试编译, 最后只有杀掉该相关应用所有进程重新连接才恢复。

这次故障导致我们一个无故障最长时间的团队免费去海南旅游三天的机会丧失。

当时的教训就是任何 ddl 的变化都需要 check 这个对象可能被引用的对象, 现在已经延伸到任何频繁被访问的 sql 了, 基本频繁访问的应用要做 ddl 都要深夜才能做了

业务期间索引维护操作

偶遇到的严重事故: 其实也不是人为造成的。

Oracle 9i 的库, 由于需要 move tbs 来降低 HWM, 然后再做 alter index rebuild online, 脚本连续跑了 1 个多月了, 都没事情。某天突然发生问题, alert log 中无报错, 应用访问数据库效率奇低, 查了 n 多原因, 未见异常, 但是已经造成业务中断 3 小时。得到客户同意后, 做完数据库全备, 中午 12 点重启数据库解决该问题。_-!!

事后发现其实在凌晨 2 点的时候有一个 trc 文件生成, 看里面一堆的天书代码, 发现类似一个 object id, 去查 object id, object 果然是被重建索引, 估计是 rebuild online 的时候失败, 到白天业务高峰期间 smon 还在清理临时段, 因此业务堵塞

另外一个省也是类似的事情，也是做 rebuild online，但是估计中途失败了，再次做 rebuild online 的时候报错 ora-8106 的错误，按照 oerr 的指示，进行 rename SYS_JOURNAL_nnnnn 表，数据库一下子猛报 ora-600 的错误，且切出来大量的 udump 文件，害怕了，重新 rename 回，600 错误不再报，但是估计 smon 又开始忙活……8 点开始业务高峰来了……再次堵塞……一个字：“等”！-!!到 11 点，smon 清理完毕，恢复正常。

教训：（1）做 rebuild online 的时候一定要谨慎！！特别是大表的索引！（2）不要全信 oerr 的提示-_-！

业务期间索引维护

有过一次，我们的应用管理员为提高自己统计佣金语句的查询速度，自己自做主张在一张表上又建了一个索引，没过几分钟，tuxedo 的队列就开始阻塞，前台营帐某一应用特别的慢，问一下应用管理员最近有什么变动，他回答说：没有。

问题报到我这里，简单查一下，相应的应用几乎都在一条语句停留时间较长，看一下该语句的执行计划，发现走的索引不对，同时查了一下 ddl trigger 的 log，发现应用管理员在几分钟前在这个表上建了一索引。drop 掉新加的索引问题解决。应用管理员无语，领导发表了一通评论

业务期间索引维护

（05 年的事）刚换新东家的时候，入职第一天，服务部那听说开发部来了一个搞 DB 的，之后就马上过来找帮忙，说客户那边的查询很慢，需要解决方法。偶就做了一个优化脚本 dbms_stats，加 index，很得意的做法。后来发现查询快了，但是整个业务流程慢了，又被投诉，原来还是业务+查询混合使用的系统。

后来把 index 删除了，然后想了其它方法。

总结:在动 DB 之前一定要知道这 DB 的具体用途，在给 DB 加东西的时候，一定要多了解!!! 很多人把 DBA 当作神，但自己不可忘记自己不是神，一定要切合实际，要深入到真实环境中!!!

而大伙说的查询系统是整个业务系统里的一个子系统，提供查询的，偶以为是 DSS 之类的查询系统。被经验误导，从此之后到任何 DB 的东西都问得清清楚楚。不动不熟悉的系统 ~

业务期间 DDL 维护

没有犯过错的 DBA 不是好的 DBA，但经常犯错的 DBA 也不是好的 DBA。

我是大的错误没有犯过，只有两个算是小一点的错误：

一次是在业务繁忙的时候给一个最基础的表加一个字段，导致全公司程序停止半个小时；另一次是准备将测试机重启，结果将生产机给重启了

业务期间 DDL 操作

刚工作一年的时候，开发给了一段脚本就是给账户表某个字段修改长度（`alter table account_t modify...`），我当时太累了，发了的脚本也没有说明何时操作，我就直接在生产库上执行了。可想而知，大部分存储过程都失效，全省业务暂停 2 小时， 嘿嘿。。。领导以后就给了我称号“破坏王”。

分区维护致索引故障

9i 的 rac ,7x24 业务,删除一个分区表的分区时，没有加 `update global indexes`,导致索引失效，甲方要求用 `delete`，组织按部分 `delete` 时，有个表千万的记录，

和另外一个小表名字很像，结果大表删除时没有加条件限制，很久没有结束，然后终止，这时一个 instance crash，不久另外一个 crash。当时查了一下好像 service guard 有问题，非常想自己启动一下，还是没有做，让客户找小机工程师来看。小机工程师也没有看出什么问题，重启后好了。

业务停止了 2 个多小时。是做 dba 以来最惊心动魄的一夜。

教训就是：充分准备，测试

另外，看前面兄弟说的事情，很多是手误或者大脑不清醒。dba 干活经常是深更半夜，而且有时是连续作战，到后面脑子肯定不好使了。所以我基本上有这么个习惯，在干活前先把步骤理一遍，具体到每一个命令。

如果有测试环境，先在测试环境做一下。

然后就照着这个 step 做，最好是复制粘贴。操作时，关闭其他所有 shell 窗口，将操作窗口的日志保存；

这样操作过程都可以记录，如果在操作过程中发生意外，意外都是不可知的，有大有小，有时比计划做的事情还麻烦。这样意外昨晚后，不至于漏掉要做的步骤，比如如果并行建索引，完了要改为非并行。总体操作是受控的。

事情完成了，写报告也比较方便。

非常赞同使用 ISO 9000 的管理方法：

记下要做的，

按所写的做，

写下所做的。

备份级误操作案例

我们曾经反复强调：备份重于一切。

在很多人执行备份时，也遭遇了因为备份而导致的误操作故障，总结这些故障，我们有以下几点警示：

1. 执行的操作系统命令需要经过测试

很多操作系统级别的命令都具有一定的危险性，如 `rm/mv/tar` 等，如果你不理解这些命令的具体含义和参数意义，那就可能犯下无法挽回的错误。

所以，你需要执行的每条命令，都需要经过测试，确保其有确定的输出结果，然后才去执行它。如果你对某个命令没有把握，那永远不要去执行它。

2. 执行备份并且进行备份检查

很多企业觉得有了备份就高枕无忧了，可是备份和“有效备份”还是两回事，我们一定要检查备份成功与否，备份是否有效，这样才能保证危机关头有“备”无患。

3. 通过文档准备完善操作流程

在执行任务之前，准备文档手册，通过测试验证可行性，并且在执行时按照文档操作，确保不要节外生枝。

俗话说：台上一分钟，台下十年工。如果在台下做好准备，则台上的一分钟表演才能流畅完美；而如果台下只花一分钟准备，那么台上来收尾恐怕就要十年工。这样的案例比比皆是。

所以，多做些准备工作，磨刀不误砍柴工。

以下是一些备份相关的误操作和灾难案例：

案例概述	案例详情
无备份导致数据损毁	刚才同事告诉我，以前我的顶头上司，IT 经理引咎辞职了，仔细一问，原来是我的继任没有做备份，

资料全部损毁，这也是今年我第 2 次见到这种情况

当时走的时候，招了一个月也没有招到合适的，后来听同事说，从香港找了一个 DBA，一个月几万块，没想到才没有多久就出了这么大的事情。

以前也有客户的 DBA 损毁资料库而且也没有做备份导致工厂停产半个月，IT 经理走人的教训，现在不敬业的人还是那么多

TAR 操作覆盖文件

```
tar -cvf *.log
```

直接把前面几个 online redo log tar 进了最后一个 online redo 里面。。。幸好不是 current 的

TAR 操作覆盖文件

tar cvf 后面两个参数写反了，结果前面的数据文件没了

TAR 操作覆盖文件

刚开始接触 Linux tar -vzvf , tar -xzvf 后面的搞错了，备份全没了

马上重新备份，汗啊

TAR 操作覆盖文件

tar cvf 后面两个参数写反了，结果前面的数据文件没了。

我就是犯了類似的錯誤，把 system01.dbf 給壞了一部分！盡管馬上按 ctrl+c,

還好是EBS剛準備上線的那一次,幸好有前一天的冷備！

那次知道了，為什麼 DBA 一定要有經驗的！做 DBA 不光是只要技術的,心態和性格很重要

误操作 TAR 覆盖数据

半夜加班，系统上线和数据迁移一起，在开始前进行了冷备文件，当上线和数据迁移要完的时候，当时不知道怎么想的可能是半夜脑壳发昏，就在解压 TAR 把

	当前数据文件覆盖了，幸好当时意识到了中止了解压，并且被覆盖的数据文件还没有数据。当时赶快把数据文件离线，删除，重建，不然要被旁边的同事海揍
导出备份覆盖数据文件	做 exp 导出时,导到了 user.dbf 文件,还是生产库,结果生产服务器宕了 3 天才恢复好
备份时文件缺失	数据库运行在非归档，冷备时少了一个文件（别的同事做的备份），过了几天恢复数据库，用当时的冷备恢复，结果数据库起不来，丢失的文件还包括很多重要应用字典数据，没办法，重新输入这些字典数据，花了三天三夜。 还有几个月前，做测试，连到了生产库，把几个表空间删除了，出了一身冷汗！幸好是晚上，没有什么应用，及时恢复了数据库
断电导致数据丢失	有一次大厦停电，通知半夜 12 点停电，我就懒得去动数据库了，没有备份，结果第二天早上，磁盘阵列启动不了了。丢了周五一天的数据。我才发现：不能想当然的认为什么都不用做，这个错误让我更加记住了大家常说的：备份重于一切。呵呵……
误操作覆盖导出文件	一次做数据库的重建工作，按用户导出了所有的数据。 数据量挺大的，忙了很久。突然，脑袋发昏了，本来一个导入操作，鼠标粘贴出来了一个导出。结果一个大的 dmp 文件变成了 0k。还好，有一个全库的导出在另外一个目录。 教训就是：以后所有重要的导出数据，全部必须是 400
误操作覆盖导出文件	imp 错用了 exp. 结果把原来的 dmp 文件覆盖了. 数据丢了. 幸运的是数据不太重要. 历史帐单数据,. 一年刚好到期. 可以封存了. 当时我很想告诉

领导是我误操作，不过最后还是没有勇气去承认，人就是人，不是神

误覆盖重要数据文件

在线移动数据文件,dd 时把其他的覆盖了另外一个数据文件啊.

结果可想而知.从带库中花了几个小时才把这一个 datafile 恢复回来.还好有备份啊

导入数据库误操作

将配置好用户配置要导入到生产库中，结果由于生产库中两个用户名太像，结果导致正在用的那个用户下，一堆存储过程失效，导致业务中断……

进程级别误操作案例

很多维护工作涉及到进程级别的一些操作，强制终止进程是很多 DBA 都做过的工作，不过一旦失误，终止的进程出现问题，则会为数据库带来麻烦。

最常见的是误终止了后台进程，还有跟踪导致的大量进程转储信息导致空间耗尽。

这些案例给我们的警示是：

1. 明确区分后台进程和用户进程

数据库的后台进程是维持数据库正常工作的必要进程，如果误操作杀掉重要的后台进程，则数据库可能立即崩溃，所以一定要注意区分后台进程和用户进程的区别。

通常数据库的核心后台进程 SMON/DBWR/LGWR/CKPT/PMON 等，一旦异常终止就会导致数据库崩溃。

2. 反复确认后再执行操作

对于终止进程的命令，要反复确认后再执行，并且注意，如果进程正在执行特定的操作，比如索引重建、Truncate 数据表等，意外中断可能导致严重的数据库内部错误。

所以，应当通过系统级的跟踪命令对进程堆栈进行跟踪确认后再进行操作，并且进行进程强制终止时，要做好应对异常的准备。

以下是常见的一些案例情况：

案例概述

案例详情

误终止后台进程

一次一个 session 占用内存很大,这个 session id 比较大,所以以为是用户进程,kill,则库立刻 down 了,查日志后,才知道是一个后台进程,但详细是哪个进程,现在忘记了.

好的是库起来了,这个故障,我一直牢记于心.

现在做任何操作是,都要检查正确后再敲回车

误终止后台进程

一次 TUXEDO 服务出现严重堵塞,前台叫得又急。一看是数据库的 TX 锁堵塞,我查找堵塞别人的 SESSION,然后找出它们的 PID,在操作系统上直接 KILL 了,结果有一个是数据库核心进程(这个进程产生了 TS 类型的 enqueue,而不是 TX 的 enqueue,当时没有仔细看)。数据库马上 DOWN 了,吓出一身冷汗,马上重启数据库,重启服务,业务中断 10 分钟。

以后再怎么急,也要确认一下要 KILL 的 SESSION 是否为用户进程

进程跟踪空间占用

我前 2 天想对一个操作,进行跟踪,但是找不到他的 sid,后来弄了个 system 级别的跟踪,因为应用是 tuxedo,长连接.

第二天客户 udump 下的日志 20G,搞的数据库 hang 住了。哎,这个真的要小心

进程跟踪空间占用

应用是 tuxedo,进程很多。我想跟踪应用操作的 sql,我就做了 alter system 级别的跟踪,后来忘了关闭 system 级别跟踪,第二天,发现磁盘满了,导致数据库 hang 了 5 分钟,对客户造成了很大损失

数据文件误操作案例

在数据库的文件维护中，如果处置不善，也可能遭遇很多灾难，本书中就包含多个和文件离线相关的复杂案例。

根据这些案例，总结一下我们的警示：

1. 文件命名需要遵循规范

由于 Windows 上的文件名不区分大小写，这一和 Unix 截然相反的做法，使得很多 Windows 用户在 Unix 上犯了不少错误。

所以，对于数据库文件来说，一定要遵循一个统一的命名规范，避免因不规范带来的故障。当添加文件时，需要根据规范来进行添加，维持命名的一致性和规范性。

2. 对于裸设备的处理要反复确认

由于裸设备上的文件在文件系统不可见，所以在处理裸设备文件时一定要反复确认，确保遵循成功的操作步骤，按文档、按规范进行操作。

此外系统管理员和数据库管理员要加强沟通，明确哪些裸设备在用及用途，曾经有案例，裸设备被误操作格式化后，发现有数据文件存储于裸设备上。

3. 如果有可能选择 ASM 或文件系统

由于裸设备已经退出 Oracle 的支持序列，所以应当适当的将裸设备数据库转移到文件系统或者 ASM，ASM 是好的选择，但是仍然需要专业的学习和维护才能够良好使用；而文件系统的友好性对于普通用户是方便维护的。

根据不同的用户选择合适的技术非常重要。

以下是一些 DBA 们犯下的和文件相关的错误：

案例概述

案例详情

误操作同名文件覆盖

一次在 AIX 下给客户 rename datafile，文件太多，弄到 windows 下用 excel 编辑，然后做个脚本去 rename，结果有两个数据文件名字相同，大小写不一样，在 excel 里为了文件名清晰，用了个 UPPER 函数，rename 以后，mv 脚本就把其中一个覆盖掉了，还好有备份，

恢复了5个小时。之后再也不敢用 excel 做这些脚本了

误操作同名文件覆盖

昨天犯了个错，两个磁盘都有相同的一个 datafile，要求腾出一块磁盘来，我直接 mv 过去将一个 datafile overwrite 掉了，这个表空间就这么挂了，惨啊

误操作更名文件

一次把生产环境下的/oradata 下面的 datafile 通过 MV 误更名了，幸好及时发现。mv 回来。

还有一次，晚间，进行数据的跨平台迁移。在导入用户下数据的时候。因为之前写好了导出脚本，一个是新写的，一个是旧的，结果复制的时候，搞混了。一个用户数据导出重复了，而少导了一个用户。导完后，也忘记再确认一次。就把这个磁盘分配给了另一台 server 使用。并进行了格式化操作。幸好，这个用户下面没有数据。只是一些权限。后来，从另一个相似的 DB 中，导出这个用户。并导入到新的 DB 中。才没有酿成大祸

误操作裸设备覆盖

表空间添加 datafile, 用的 raw device, 结果误用了一个已存在且在使用中的 raw device. 结果另一个 datafile 就被覆盖了，不幸中更不幸的是恰巧前几天的备份没成功，没法恢复。系统停了多少天不知道，反正据说是花了好几万 RMB 请高人做了磁盘恢复才搞定。现在一碰裸设备就想到这个 case, 每次都要用 fuser, lvscan 确认半天才敢动。教训深刻啊

误操作裸设备错误

我在一个表空间上添加一个数据文件,对于 DBA 来说是再平常再简单的不过一件事了，可是由于添加一个数据文件,差点当机

由于系统用得是 raw device,我在添加一个数据文件时,事先没有检查这个 LV 是否存在,简单地看了当前的数据库中的数据文件所用的 LV 序号,就以简单序号

+1 的方式添加了,结果也算是不走运,正好没有这个 LV,ORACLE 或者说 UNIX 操作系统当作了一个一般的文件来创建了.由于是创建在/dev/vgxx 中,所以这时搞得 UNIX 的根目录没有了空间,这个数据文件刚创建完成,其他用户就无法登录了,无法创建新的连接了.因为根目录没有空间了 .更不幸的是已经断开了这个操作系统连接.新的连接又无法创建.急呀.

不过不幸中万幸是一个同事正好有一个连接还在上面,所以马上过去直接 su - . 接下来的事大家都知道了,所以搞得现在一提起要加数据文件,就怕得要死

误操作数据库错误

累的时候多个窗口打开，加入 A 数据库的数据文件加入到了相识的 B 数据库

误操作覆盖文件

做 Standby 時把 Standby DB 的數據文件 Copy 到主庫，覆蓋掉了主庫的 SYSAUX01.DBF

误关闭生产库案例

很多 DBA 还经历过误操作关闭主机或生产数据库的情况，这种误操作绝对是刻骨铭心的，往往一个回车下去，就幡然醒悟，但是很多时候为时已晚。

总结这些常见的案例，我们的警示有：

1. 尽量避免层层跳转的服务器登陆方式

虽然很多企业数据环境通常都要经过层层跳转才能够访问，但是不可避免的，跳转的次数增多也就增加了出错的可能性，所以应当尽量减少跳转次数，禁止在一个主生产节点再跳转到另外的主生产节点。

在操作时，也应当通过 hostname 等方式确认连接到的服务器主机。

2. 完成操作尽快退出生产业务服务器

当在生产服务器上完成工作后，应当尽快退出，以防止其他工作干扰后，因为疏忽而出现误操作。尤其是当离开电脑前时，应当退出或锁定操作界面，防止他人误操作。

3. 经常性确认服务器、数据库和路径标示

应当经常性确认主机名称、当前路径、数据库名称等信息，防止无意识的误操作。

尤其是当重新或临时接触到操作终端时，如果不能明确看到服务器或数据库标示，则应当首先查看这些信息。

以下这些常见的命令行操作应当成为 DBA 的条件反射：

```
[oracle@hpserver2 ~]$ hostname
hpserver2.enmotech.com

[oracle@hpserver2 ~]$ pwd
/home/oracle

[oracle@hpserver2 ~]$ ps -ef | grep smon
ora11g      1484      1  0   2011 ?          00:02:05 ora_smon_eygle
oracle      5200 14397  0 15:14 pts/3    00:00:00 grep smon

[oracle@hpserver2 ~]$ id
uid=505(oracle) gid=501(oinstall) groups=501(oinstall),502(dba)
```

以下是一些典型的误操作关闭主机和服务器的案例：

案例概述	案例详情
误关闭生产数据库	有一次 ssh 了 n 次，连接到了一个省平台，当时还以为在测试机上，执行了 shutdown immediate,等了几秒钟没反应，马上意识到搞错了。 立即取消。还好发现的早，没把数据库搞 Down 以后制定了一系列规范防止这些低级错误
误关闭生产数据库	说一个刚做 DBA 的时候的事儿，大家别笑啊， 一边在本机上做实验的时候一边监控生产库，机器钟开了 N 个黑窗口,。。。,累了，本机上改完配置后需要重启库，shutdown immediate，2 分钟没有反应，脑袋“嗡”的一下，知道发生什么事情了，马上重新连接一个 session,shutdown abort； 然后通知应用人员，数据库发生误操作，需要马

上重启应用,,, OK,数据库起来,应用起来,新数据进来,,,

前后总共宕机时间 13 分钟,不过在线数据没有丢失,因为应用端有写 CACHE 机制。

结果还好,没有被追究责任,算作一次维护操作。

经验:以后每次敲完命令,按回车之前,停一秒钟

误关闭生产数据库

想关闭一个地市的数据库,结果把另外一个地市的数据库给关闭了,nb 死了

误操作关闭数据库主机

俺的一次数据库打补丁时原本是要在 SQLPLUS 登陆后 SHUTDOWN IMMEDIATE 当时敲得太快没注意 SQLPLUS SESSION 已经退出了结果是在 OS 级别 SHUTDOWN IMMEDIATE 把服务器给停了,我的乖乖啊,问题是该服务器远在国外啊,只好叫正享受周末假日的国外的同事赶到机房去启动,那个囧啊

误操作影响主机 HA

有一次在 HP 的 SERVERGUARD 的双机环境,备机是用做测试库的。我发现起了一个生产机的 INSTANCE 在上面。尝试了 alter database mount,发现并没有 MOUNT。而且正常情况下,SERVERGUARD 的备机应该不会有生产机的 INSTANCE。断定是起了无效的的 INSTANCE。只是 INSTANCE,并没有 MOUNT,然后将该 INSTANCE 执行了一个 SHUTDOWN IMMEDIATE。。没有想到那边的生产机也在进行 shutdown。而且,serverguard 的包有问题,数据库每次起来以后就自动就重起了。折腾了 2 个多小时

误操作关闭数据库主机

昨晚重启生产库,结果不知怎么的在 OS 下面敲了 shutdown,然后……

打电话到移动找人重启机子,搞到两点过,郁闷

误操作关闭数据库主机

有一次半夜被 call 到机房，头有些晕沉，想找一台 windows telnet 上 db 去检查检查，因为用了屏幕切换器，一个 ctrl+alt+del 组合键下去，一台 db 服务器被我 reboot 了(linux 下没有屏蔽掉 ctrl+alt+del 三键重启)，吓出一身冷汗来，幸亏是一个小型 dw 应用，晚上不会用得到。

此后，凡是在 linux 下跑的 oracle，装好 os 后我一律最先将/etc/inittab 里的 ca::ctrlaltdel:/sbin/shutdown -t3 -r now 这一行给屏蔽掉

误操作关闭主机

和数据库无关，去客户那里做升级，白天交易时间（证券），由于不熟悉客户机房里设备的摆放，以为屏幕下的键盘就是这台主机的，直接 ctrl+alt+del 启动（无盘站）结果这个键盘控制的是另外机架的主机，是乾隆转码机器，吓了一身冷汗，尽快重启。

教训：去客户那里一定先熟悉环境，最好和客户一起做

误操作关闭生产主机

我的一次，双机，os 升级，先在备机上 update，patch 什么的都打完后，在 terminal 里爽爽的敲 reboot 的时候发现把主机给起了。冒冷汗的同时想起那个 terminal 是 telnet 到主机上的...结果库还起不来，冷静地检查了一把，发现主机正跑着 alter trablespace begin backup....，赶紧把中断的备份都给 end backup 了，弄得现在一要重起都习惯性的先打 hostname

误操作关闭生产主机

俺最惨的一次是上了 10 几个小时夜班后正准备下班，点进 VM 执行 init 0，却忘记有从这个 VM 窗口 telnet 到生产环境 cp 参数文件而且等数据库状态监控报警后才反应过来...

还好是 RAC，但也造成不小影响，从此下任何指

令前先 CHECK 過。

另外個人總結在 unix 下盡量用 tab 得到文件名和路徑名有助於避免空格錯誤

误操作关闭生产主机

刚入行，就犯了一个严重误操作：想关闭测试环境服务器，结果把生产环境服务器关闭了

系统存储级误删除案例

除了数据库层面，在主机、操作系统、存储层面也有很多典型案例，如果不够谨慎，主机网络层面的误操作也可能对系统产生致命的影响。

以下是来自这些案例的警示：

1. 超级用户和数据库用户严格分离

在生产环境中，不应该给 DBA 以 root 权限，以防止不到操作给整个系统带来的影响，即便 DBA 可能也很了解系统，但是专业分工要求有系统管理员去执行系统层面的维护工作。

避免因为 DBA 的操作不当导致的系统故障。

2. 事关存储无小事

存储最终容纳着用户的所有数据，所以针对存储的任何操作都不能草率，当增减硬盘，格式化分区时，都要严格进行磁盘确认、分区比较，避免因误操作而“釜底抽薪”。

3. 电源即 Power

电源也就是 Power，是所有动力的来源，所以当中断电源时，系统的所有环境都可能遭受影响。在处理面对电源问题时，应当慎之又慎，因为断电而导致数据库无法启动的案例比比皆是。不要让数据库因为电源问题而崩溃。

以下是一些操作系统和存储级别的误操作案例：

案例概述

案例详情

误发出系统命令

hp unix oracle10.2,我用 root 登陆后,建立了一个新主机用户,不知不觉的敲了个 `hostname -a`,大家知道后边发生什么了吗?

…是和 `uname -a` 搞混了。。。 `hostname -a` 直接把主机名改成-a 了。。。

`listener` 是监听主机名的,现在找不到主机了,连续报错,还有后台 `trc` 文件也连续报错,这个主机上共有 4 个实例。。同时连接不上。。。壮观啊。。。很快日志占满文件系统。

查不出原因,但是发现文件系统使用的很快,就想先停库,再查原因了。。。结果,启动的时候都 ora600 了。。。好在是测试环境的数据库,,不是正式的。。。真是刻骨铭心啊

误格式在用存储

存储厂商过来划分存储,不小心将已经在用的盘重新划分,导致 2 个应用测试库和一个培训库瘫痪。万幸的是没把在一个阵列上的生产库也给搞掉

误切换生产存储

一次冰凉透顶的操作,去年某天下午,本来是对灾备端的盘柜做 HA 切换,头脑一昏,随手一按,把生产端的盘柜进行了手动 HA 切换,20 多套数据库系统在上面跑的....后果不堪设想,还好一个急智,赶快又切换回来,装作什么事没有,手一直颤抖.....过后偷偷问一些在线服务系统有没有什么异常,MM 只是说有几分钟很慢,数据库没反应,过后又正常了,汗....

从此对生产环境有一种“非诚勿扰”的感觉,敬而远之

误清空在用磁盘

RAC 环境,dd 裸设备的时候,本来只打算清空 `data diskgroup` 的,结果不小心清空了所有 `raw disk`, `ocr` 和 `vt` 当场挂掉了,不过是测试环境,哈哈,要是 `prod` 我

可以直接卷铺盖走人了

误关闭 UPS 电源

一不小心把 UPS 的关了，导致主机停电，幸好数据库可以正常启起来

误关闭存储电源

亲眼见过一次，当数据库的 R A C 配置成功后，用户不小心把存储电源给拔了，结果只能重做，幸好没重要数据，要不

误操作损坏文件系统

Linux 下，在文件系统没有卸载的情况下，使用 fsck 命令，导致文件系统损坏，所有数据全部丢失！后悔了好几天

误操作损坏文件系统

在 LINUX 下输入了 FSCK 直接回车敲了几下，导致 LINUX 系统挂掉，数据库也一样的挂了

存储维护危险误操作

在 cx700 的存储 navisphere 管理界面,配置一个存储;同事接过去打开了生产环境另外一个存储的 ie 窗口;我又接手过来,一恍惚看这个存储的配置与我打开的一样,就开始做删除 STORAGE GROUP 的操作;还好我旁边另外一个同事看主机名不对,制止了我继续删除(我当时对他讲解了一下配置存储的步骤然后开始操作);

删除了 lun 就丢生产环境的 crm 数据了;

这个事情很可怕,那天人状态不怎么好;以后做事情越是知道状态不好,越要加倍谨慎;还有以前删除文件用相对路径来删除, ../path 方式,误删除了测试环境的 oracle 程序(以后都用绝对路径了)

误操作执行系统命令

生产环境增加节点，熬了两天两夜，同事在生产机上执行了 pvid=yes 导致数据丢失，最后奋战两天重新安装 RAC

误删除操作系统文件

一次在 ibm p570 上安装 rac，由于客户网络有问题，结果失败，在删除 rac 时 rm -initab*.crsd 等几个

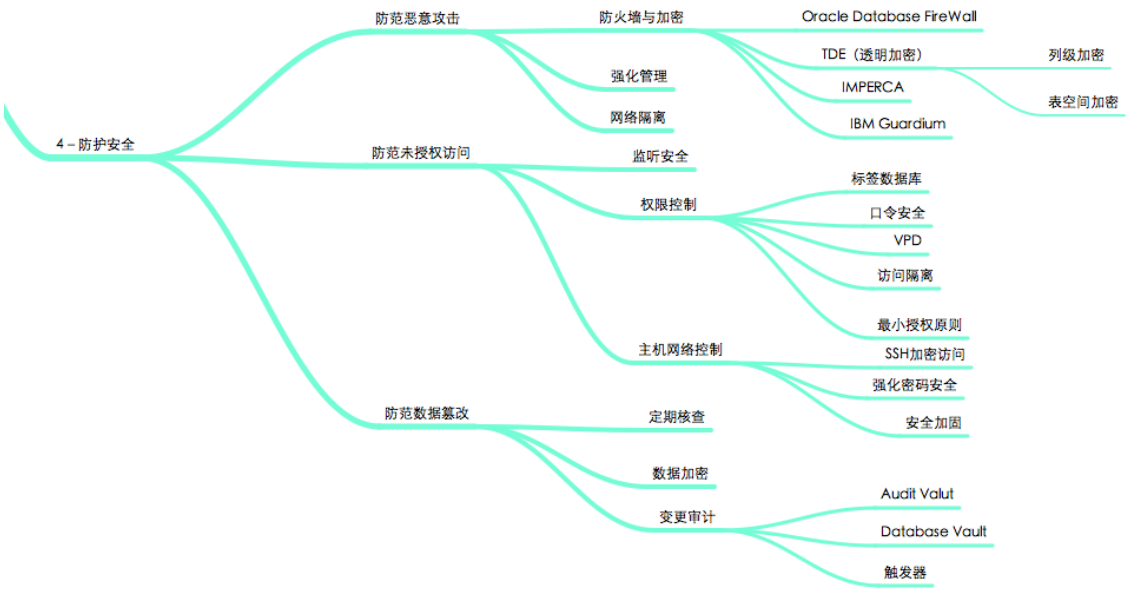
rac 的启动文件，一不留神吧 AIX 的一个文件删了，结果系统起不来了。后来多亏 ibm 的工程师恢复了系统。结果晚上 3 点才收工

亡羊补牢，未为迟也

见兔而顾犬，未为晚也；

亡羊而补牢，未为迟也。

——《战国策·楚策四》



近年来，数据泄露等数据安全问题越来越突出，随着安全问题逐步批露和报道出来，越来越多的企业开始重视自身的安全问题。

其实很多企业早已经面对着数据安全隐患，并且承受了很多安全问题的困扰，亡羊补牢，未为迟也，通过 Oracle 数据库的很多成熟技术和手段，我们可以逐步构建安全架构与解决方案，渐进式去修补一度疏为防范的数据库安全。

关于防护安全，篇首图列举了一些相关的技术内容，其中实现网络隔离，并且严格控制权限对安全防护非常重要，而审计则可以帮助用户清晰的统计数据变更来源，实现明明白白的数据管理：

万事只怕开头难，只要开启了数据安全的航程，我们的数据库就会一点一点、一步一步的走向稳固与安全。

数据篡改案例解析

以下是一则关于数据篡改的案例，客户的数据库因为安全管理和防范不善，导致数据库信息被经常性篡改，客户长时间来束手无策。这则案例的现象可能普遍存在，很多用户对于数据安全缺乏认知，对于安全隐患的发现和解决感到无从入手。

案例描述

以下是这则案例的详细信息描述：

1. 客户数据库中的某些账户余额信息被修改
2. 此类篡改经常发生
3. 客户通过程序定时记录那些欠费用户
4. 发现无交易记录但是余额变更就判定为恶意篡改，然后通过备份记录再更新恢复
5. 某日，某账户余额从 0 被修改为 40000
6. 客户请求协助分析

这是客户在隐忍多时之后向我们提出的服务请求，希望通过技术手段来找到篡改途径，解决一直以来的安全隐患。

案例警示

通过分析解决，这个案例给我们的警示是：

1. 数据安全防范应当从点滴做起

数据库的安全措施和手段，从无到有就是一个巨大的飞跃，对于一个从不考虑安全的企业，数据的安全可靠是没有保障的。数据安全防范，应当在任何一个数据库系统中纳入到日程上来，从点滴做起，系统化分步骤去实施。

充分的重视就是数据安全的基础。

2. 安全防范的首要任务是从内部做起

不可否认，绝大部分安全问题都来自于企业内部，来自最紧密、最轻易的接触和访问，企业的人员变动，岗位变更，都可能导致数据安全问题的出现，单存依靠对管理员的信任不足以保障数据安全，必须通过规章、制度与规范的约束才能够规避安全风险。

很多企业为了便利而舍弃规范、规章或者安全限制是得不偿失的做法。

安全防范应当从内部做起，从限制约束自我做起，当最紧密相关的访问都遵从守则，那么系统的安全性就能够获得大幅度的提升。

以下是一些可能需要考虑的限制措施，以防范内部风险：

1. 对数据管理和维护任务进行审核
2. 必要时进行跨部门或由外部专家进行复查
3. 进行必要的任务分解，不同工作由不同人来完成
4. 遵守最小授权原则
5. 严格控制管理员密码
6. 限制数据库管理维护来源

只有首先保障内部安全，外部安全才具有根本意义。

3. 利用数据库的自有功能实现安全防范

很多企业总觉得在安全上的投入见效缓慢，甚至根本看不到效果，因此就推迟或放弃安全防范，这是绝对错误的做法。

常规数据库本身就可以实现很多安全防范，从最基本的安全功能入手，也能利用数据库产品自身打造一套严密的数据安全解决方案。

Oracle 数据库本身就提供多种安全解决方案，从基本的内部包加密到数据库审计、数据库 Vault、防火墙等等，能够提供全系列的安全防范手段，了解并应用这些手段可以帮助我们大大强化数据库安全。

数据安全与防范，从点滴做起，从内部做起，逐步构建安全稳固的数据环境是我们共同的职责和使命。

技术回放

在这个客户案例中，我们分析篡改信息时，其中的一个步骤是通过 Oracle 的日志文件来进行解析，试图找

出修改的时间、终端等信息，通过这些信息可以辅助判断篡改情况。

故障分析的过程

通常归档日志不仅可以作为备份、容灾的必要手段，还可以作为数据安全核查的辅助。

但是在这则案例中，工程师在解析日志期间遇到一个问题，发现通过解析后的信息，过滤 SQL_REDO 根本无法找到相应表的 UPDATE 修改记录。

我开始的第一步分析也是解析日志文件，结果发现最终定位的记录，其 SQL_REDO 记录为 Unsupported，也就是不支持，无法解析出 SQL，如果通过 SQL_REDO 去分析篡改 SQL，则将一无所获：

```
SQL> select ABS_FILE#,REL_FILE#,DATA_BLK#,DATA_OBJ#,SEG_NAME ,rs_id
       2  from v$logmnr_contents where session#=90 and seg_name='BROAD_SUBSCRB';
```

ABS_FILE#	REL_FILE#	DATA_BLK#	DATA_OBJ#	SEG_NAME	RS_ID
2	47	7600	66237	BROAD_SUBSCRB	0x00309e.00028a0a.0010
47	47	7602	66237	BROAD_SUBSCRB	0x00309e.00028b4d.0010

```
SQL> select TIMESTAMP,ABS_FILE#,REL_FILE#,DATA_BLK#,DATA_OBJ#,sql_redo
       2  from v$logmnr_contents where session#=90 and seg_name='BROAD_SUBSCRB';
```

TIMESTAMP	ABS_FILE#	REL_FILE#	DATA_BLK#	DATA_OBJ#	SQL_REDO
2011-07-05 16:41:38	2	47	7600	66237	Unsupported
2011-07-05 16:41:54	47	47	7602	66237	Unsupported

尝试直接转储日志文件，通过日志转储找到了这则被篡改的记录，以下信息即为从日志转储中获得的信息，增加了部分说明：

```

KDO Op code: URP row dependencies Disabled
  xtype: XA bdba: 0x0bc01db2 hdba: 0x0900e509
  itli: 3 ispac: 0 maxfr: 4863
  tabn: 0 slot: 47(0x2f) flag: 0x0c lock: 0 ckix: 0
  ncol: 33 nnew: 2 size: -1
col 7: [ 1] 35
col 15: [ 1] 80
>>>>这里记录了修改前的值,分别修改了第7和第15列信息(由0编号,等于第8和16列)
>>>>80就是十进制的0
CHANGE #2 TYP:2 CLS: 1 AFN:47 DBA:0x0bc01db2 SCN:0x0001.4be1efdb SEQ: 1 OP:11.5
>>>>OP:11.5 指更新行记录信息
KTB Redo
op: 0x01 ver: 0x01
op: F xid: 0x0003.022.0019482c uba: 0x00801542.af17.05
KDO Op code: URP row dependencies Disabled
  xtype: XA bdba: 0x0bc01db2 hdba: 0x0900e509
  itli: 3 ispac: 0 maxfr: 4863
  tabn: 0 slot: 47(0x2f) flag: 0x0c lock: 3 ckix: 0
  ncol: 33 nnew: 2 size: 1
col 7: [ 1] 31
col 15: [ 2] c3 05
>>>>这里是更新后的值,c3 05就是40000

```

实际上这里就是篡改的内容,80 就是 0, 是前镜像信息, 修改之前的账户余额; 而 c305 即 40000, 是修改后的金额:

```

SQL> select dump(0,16) "0",dump(40000,16) "40000" from dual;

0                               40000
-----
Typ=2 Len=1: 80 Typ=2 Len=2: c3,5

```

通过这个信息定位, 获取 SESSION 会话数据, 我们准确的找到了篡改数据的嫌疑人, 实际上一贯以来的篡改都是内部人士所为了, 再结合一些其他信息, 可以完全确定:

```

SQL> select SESSION_INFO from logcont where rbablk=166733 and data_obj#=66237;
SESSION_INFO

```

```

-----
login_username=CINMS client_info= OS_username=Administrator
Machine_name=WORKGROUP\CGR

```

所以我们说: 数据安全防范, 应当从内部做起, 安全问题往往是从内部出现。

日志文件的转储

前面我们提到通过直接的日志文件转储用于分析，这里简要的介绍一下如何进行日志文件转储。

最简单的做法是通过如下命令转储：

```
ALTER SYSTEM DUMP LOGFILE 'filename';
```

示例如下：

```
SQL> select member from v$logfile;
MEMBER
-----
/mwredo/oracle/redo01.log
/mwredo/oracle/redo02.log
/mwredo/oracle/redo03.log
/mwredo/oracle/redo04.log
SQL> alter system dump logfile '/mwredo/oracle/redo01.log';
System altered.
```

但是这个命令会转储整个日志文件，转储日志会较大，以上日志文件为 20M 转储文件大小为 30M 左右：

```
[oracle@db82 udump]$ ls -l cmwapdb_ora_12732.trc
-rw-r----- 1 oracle dba 34553876 Feb  8 17:31 cmwapdb_ora_12732.trc
```

我们可以通过如下命令，进行指定 RBA 的日志转储：

```
ALTER SYSTEM DUMP LOGFILE 'filename'
RBA MIN seqno . blockno    RBA MAX seqno . blockno;
```

例如如下示范：

```
SQL> select * from v$logfile;
GROUP# STATUS TYPE MEMBER
-----
1 ONLINE /mwredo/oracle/redo01.log
2 ONLINE /mwredo/oracle/redo02.log
3 ONLINE /mwredo/oracle/redo03.log
4 ONLINE /mwredo/oracle/redo04.log
SQL> select sequence#,group# from v$log;
```

SEQUENCE#	GROUP#
172919	1
172920	2
172921	3
172922	4

```
SQL> alter system dump logfile '/mwredo/oracle/redo01.log'
```

```
2 rba min 172919 . 10 rba max 172919 . 20;
```

```
System altered.
```

检查跟踪日志文件，可以看到日志转储由第 10 块开始，日志文件的块大小是 512 Bytes：

```
DUMP OF REDO FROM FILE '/mwredo/oracle/redo01.log'
```

```
Opcodes *.*
```

```
DBA's: (file # 0, block # 0) thru (file # 65534, block # 4194303)
```

```
RBA's: 0x02a377.0000000a.0000 thru 0x02a377.00000014.0000
```

```
SCN's scn: 0x0000.00000000 thru scn: 0xffff.ffffffff
```

```
Times: creation thru eternity
```

```
FILE HEADER:
```

```
Software vsn=153092096=0x9200000, Compatibility Vsn=153092096=0x9200000
```

```
Db ID=4176528780=0xf8f0c58c, Db Name='CMWAPDB'
```

```
Activation ID=4202226610=0xfa78e3b2
```

```
Control Seq=967810=0xec482, File size=40960=0xa000
```

```
File Number=1, Blksiz=512, File Type=2 LOG
```

```
descrip:"Thread 0001, Seq# 0000172919, SCN 0x081f4051d185-0x081f40525721"
```

```
thread: 1 nab: 0x5d68 seq: 0x0002a377 hws: 0x2 eot: 0 dis: 0
```

```
reset logs count: 0x20b2f50b scn: 0x0000.00dfe6d0
```

```
Low scn: 0x081f.4051d185 02/06/2012 02:30:00
```

```
Next scn: 0x081f.40525721 02/07/2012 02:30:02
```

```
Enabled scn: 0x0000.00dfe6d0 01/26/2005 12:37:31
```

```
Thread closed scn: 0x081f.4051d185 02/06/2012 02:30:00
```

```
Log format vsn: 0x80000000 Disk cksum: 0xcf25 Calc cksum: 0xcf25
```

```
Terminal Recovery Stamp scn: 0x0000.00000000 01/01/1988 00:00:00
```

```
Most recent redo scn: 0x0000.00000000
```

```
Largest LWN: 0 blocks
```

```
End-of-redo stream : No
Unprotected mode
Miscellaneous flags: 0x0

REDO RECORD - Thread:1 RBA: 0x02a377.0000000a.0054 LEN: 0x00ec VLD: 0x01
SCN: 0x081f.4051d18b SUBSCN: 1 02/06/2012 02:31:20
CHANGE #1 TYP:0 CLS:34 AFN:2 DBA:0x00800270 SCN:0x081f.4051d18b SEQ: 1 OP:5.1
ktudb redo: siz: 96 spc: 1310 flg: 0x0022 seq: 0xb848 rec: 0x2f
          xid: 0x0009.00e.0012c2a1
ktubu redo: slt: 14 rci: 46 opc: 10.22 objn: 33248 objd: 33248 tsn: 60
Undo type: Regular undo      Undo type: Last buffer split: No
```

如下命令可以转储日志文件头:

```
alter session set events 'immediate trace name redohdr level 10';
```

例如:

```
SQL> alter session set events 'immediate trace name redohdr level 10';
Session altered.
```

其转储内容会包含所有日志文件头信息, 在恢复中很有参考价值。

以下命令可以用于转储对于指定数据块的修改 (对于 Oracle 10g 之后, fileno . blockno 之间无需再加句点分隔符):

```
ALTER SYSTEM DUMP LOGFILE 'filename' DBA MIN fileno . blockno DBA MAX fileno . blockno;
```

例如以下范例:

```
SQL> select file_id,block_id,blocks from dba_extents where segment_name='OBJ$';
```

FILE_ID	BLOCK_ID	BLOCKS
1	121	8
1	4433	8
1	4577	8
1	4849	8
1	5697	8
1	8249	8

1	12809	8
1	15185	8
1	20057	8
1	23617	8
1	23649	8
1	23969	8
1	24009	8
1	24065	8
1	24617	8
1	24657	8
1	25353	128
1	29577	128

18 rows selected.

```
SQL> alter system dump logfile '/mwredo/oracle/redo01.log'
```

```
2 dba min 1 . 121 dba max 1 . 130;
```

System altered.

对于 10g、11g 转储命令类似如下：

```
SQL> alter system dump logfile '/home/ora11g/oradata/orcl11g/redo01.log'
```

```
2 dba min 1 222 dba max 1 4433;
```

System altered.

以下是摘录的一个 Redo 记录：

```
REDO RECORD - Thread:1 RBA: 0x02a377.00000011.0010 LEN: 0x01fc VLD: 0x01
SCN: 0x081f.4051d192 SUBSCN: 1 02/06/2012 02:32:00
CHANGE #1 TYP:0 CLS:21 AFN:2 DBA:0x00800029 SCN:0x081f.4051d18e SEQ: 1 OP:5.2
ktudh redo: slt: 0x001f sqn: 0x0012b52b flg: 0x0012 siz: 124 fbi: 0
          uba: 0x008003e9.c008.16 pxid: 0x0000.000.00000000
CHANGE #2 TYP:0 CLS:22 AFN:2 DBA:0x008003e9 SCN:0x081f.4051d18d SEQ: 1 OP:5.1
ktudb redo: siz: 124 spc: 4684 flg: 0x0012 seq: 0xc008 rec: 0x16
          xid: 0x0003.01f.0012b52b
ktubl redo: slt: 31 rci: 0 opc: 11.1 objn: 222 objd: 222 tsn: 0
Undo type: Regular undo          Begin trans      Last buffer split: No
```

```

Temp Object: No
Tablespace Undo: No
                0x00000000 prev ctl uba: 0x008003e9.c008.15
prev ctl max cmt scn: 0x081f.4051cd56 prev tx cmt scn: 0x081f.4051cd5a
KDO undo record:
KTB Redo
op: 0x04 ver: 0x01
op: L itli: xid: 0x0003.007.0012b531 uba: 0x008003e9.c008.14
                flg: C--- lkc: 0 scn: 0x081f.4051d187
KDO Op code: LKR row dependencies Disabled
        xtype: XA bdba: 0x00400652 hdba: 0x00400651
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 0 to: 0
CHANGE #3 TYP:2 CLS: 1 AFN:1 DBA:0x00400652 SCN:0x081f.4051d18a SEQ: 1 OP:11.4
KTB Redo
op: 0x11 ver: 0x01
op: F xid: 0x0003.01f.0012b52b uba: 0x008003e9.c008.16
Block cleanout record, scn: 0x081f.4051d192 ver: 0x01 opt: 0x02, entries follow
...
        itli: 2 flg: 2 scn: 0x081f.4051d18a
KDO Op code: LKR row dependencies Disabled
        xtype: XA bdba: 0x00400652 hdba: 0x00400651
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 0 to: 1
CHANGE #4 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ: 0 OP:5.19

```

以下命令完成基于 SCN 的转储：

```
ALTER SYSTEM DUMP LOGFILE 'filename' SCN MIN minscn SCN MAX maxscn;
```

如下示例，通过 V\$LOG 视图获得 SCN 范围，通过指定范围进行 SCN 转储：

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

```

        GROUP#          FIRST_CHANGE#
-----
1              8930316112261

```

```

2      8930316146465
3      8930316147134
4      8930316174686

```

```

SQL> alter system dump logfile '/mwredo/oracle/redo01.log'
      scn min 8930316112261 scn max 8930316112300;
System altered.

```

熟悉日志转储方式，对于我们的 Oracle 学习研究具有很大的帮助。

LOGMNR 解析

那么在这个案例中，为什么 SQL_REDO 无法解析呢？

在正常情况下使用 LOGMNR 在线分析和挖掘日志，是非常方便的，以下是 Oracle 10g 中的一个简单测试，使用当前在线的数据字典。

首先执行一些 DDL 或 DML 操作：

```

SQL> connect eygle/eygle
Connected.
SQL> alter system switch logfile;
System altered.
SQL> create table eygle as select * from dba_users;
Table created.
SQL> select count(*) from eygle;
COUNT(*)
-----
19

```

然后可以执行 LOGMNR 解析工作：

```

SQL> connect / as sysdba
Connected.
SQL> select GROUP# from v$log where status='CURRENT';
GROUP#
-----
2

```

```

SQL> SELECT MEMBER from v$logfile where group#=2;
MEMBER
-----
/opt/oracle/oradata/mmstest/redo02.log
SQL> exec dbms_logmnr.add_logfile
      ('/opt/oracle/oradata/mmstest/redo02.log',dbms_logmnr.new);
PL/SQL procedure successfully completed.
SQL> exec dbms_logmnr.start_logmnr(options=>dbms_logmnr.dict_from_online_catalog);
PL/SQL procedure successfully completed.
SQL> select count(*) from v$logmnr_contents;
COUNT(*)
-----
      136
SQL> select sql_redo from v$logmnr_contents;
SQL_REDO
-----
-----
----
      set transaction read write;
      insert into
"SYS"."OBJ$ "("OBJ#", "DATAOBJ#", "OWNER#", "NAME", "NAMESPACE", "SUBNAME", "TYPE#", "CTIME",
"MTIME", "STIME", "STATUS", "REMOTEOWNER", "LINKNAME", "FLAGS", "OID$", "SPARE1", "SPARE2", "S
PARE3", "SPARE4", "SPARE5", "SPARE6") values ( '25847', '25847', '31', 'EYGLE', '1', NULL, '2',
TO_DATE('01-JUL-09', 'DD-MON-RR'), TO_DATE('01-JUL-09',
'DD-MON-RR'), TO_DATE('01-JUL-09',
'DD-MON-RR'), '1', NULL, NULL, '0', NULL, '6', '1', NULL, NULL, NULL, NULL);
.....
      create table eygle as select * from dba_users;
      set transaction read write;
      Unsupported
      update "SYS"."TSQ$" set "TS#" = '0', "GRANTOR#" = '43080', "BLOCKS" = '0',
"MAXBLOCKS" = '0', "PRIV1" = '0', "PRIV2" = '0' where "TS#" = '0' and "GRANTOR#" =
'43072' and "BLOCKS" = '0' and "MAXBLOCKS" = '0' and "PRIV1" = '0' and "PRIV2" = '0'

```

```
and ROWID = 'AAAAKAABAAABbAAF';
```

```
commit;  
set transaction read write;  
SQL> exec dbms_logmnr.end_logmnr
```

PL/SQL procedure successfully completed.

很多时候拿 LOGMNR 来追踪一些误操作是有效的方式，甚至在自己定制的数据同步中，LOGMNR 也大有可为。

案例之深入解析

对于这个案例，在技术上我们需要进一步的研究。为什么会出现不支持的情况呢？是否是 LOGMNR 的限制导致的呢？

根据 Oracle 的文档，LOGMNR 的确存在一定的限制，以下一些类型不被支持：

1. Simple and nested abstract datatypes (ADTs)
2. Collections (nested tables and VARRAYs)
3. Object Refs
4. Index organized tables (IOTs)
5. CREATE TABLE AS SELECT of a table with a clustered key

但是这个表不存在这些数据类型。

我们再来仔细审视一下这个 REDO RECORD，信息都能够被手工解析出来，为何 LOGMNR 无法解析呢？

首先这个记录中包含两个改变向量，第一个是对于 UNDO 的修改，记录前值 0，第二个是对于数据块的修改，修改值为 40000，注意这里记录的 RBA 信息，其中 28b4d 代表的是 REDO 块地址，转换成 10 进制就是 166733：

```
REDO RECORD - Thread:1 RBA: 0x00309e.00028b4d.0010 LEN: 0x0114 VLD: 0x01  
SCN: 0x0001.4be86fb9 SUBSCN: 1 07/05/2011 16:41:54  
CHANGE #1 TYP:0 CLS:22 AFN:2 DBA:0x00801542 SCN:0x0001.4be86efe SEQ: 1 OP:5.1  
ktudb redo: siz: 116 spc: 7530 flg: 0x0022 seq: 0xaf17 rec: 0x05  
xid: 0x0003.022.0019482c  
ktubu redo: slt: 34 rci: 4 opc: 11.1 objn: 66237 objd: 67018 tsn: 8
```

```

Undo type: Regular undo      Undo type: Last buffer split: No
Tablespace Undo: No
                0x00000000
KDO undo record:
KTB Redo
op: 0x04 ver: 0x01
op: L itl: xid: 0x000a.01e.001a0c96 uba: 0x00800c0a.b193.29
                flg: C--- lkc: 0 scn: 0x0001.4bb7744a
KDO Op code: URP row dependencies Disabled
        xtype: XA bdba: 0x0bc01db2 hdba: 0x0900e509
itli: 3 ispac: 0 maxfr: 4863
tabn: 0 slot: 47(0x2f) flag: 0x0c lock: 0 ckix: 0
ncol: 33 nnew: 2 size: -1
col 7: [ 1] 35
col 15: [ 1] 80
CHANGE #2 TYP:2 CLS: 1 AFN:47 DBA:0x0bc01db2 SCN:0x0001.4be1efdb SEQ: 1 OP:11.5
KTB Redo
op: 0x01 ver: 0x01
op: F xid: 0x0003.022.0019482c uba: 0x00801542.af17.05
KDO Op code: URP row dependencies Disabled
        xtype: XA bdba: 0x0bc01db2 hdba: 0x0900e509
itli: 3 ispac: 0 maxfr: 4863
tabn: 0 slot: 47(0x2f) flag: 0x0c lock: 3 ckix: 0
ncol: 33 nnew: 2 size: 1
col 7: [ 1] 31
col 15: [ 2] c3 05

```

这条记录一定存在与众不同之处，经过对比分析，我们发现这里的 flag 值 0x0c 与其他记录不同，在日志转储中，多数修改行的标志位都是 0x2c，0x0c 只占很少一部分：

```

eygle$ grep flag xab.log|grep -v leaf|awk '{print $5 $6}'|sort -u
flag:0x0c
flag:0x2c
eygle$ grep flag xab.log|grep -v leaf|awk '{print $5 $6}'|grep 0x0c|wc -l

```

```
eygle$ grep flag xab.log|grep -v leaf|awk '{print $5 $6}'|grep 0x2c|wc -l
1956
```

那么这个 FLAG 代表什么呢?

根据文档, FLAG 具有以下可选项, 在块头用 8 位来表示:

```
#define KDRHFK 0x80 Cluster Key
#define KDRHFC 0x40 Clustered table member
#define KDRHFH 0x20 Head piece of row
#define KDRHFD 0x10 Deleted row
#define KDRHFF 0x08 First data piece
#define KDRHFL 0x04 Last data piece
#define KDRHFP 0x02 First column continues from Previous piece
#define KDRHFN 0x01 Last column continues in Next piece
```

通常正常的 FLAG 为 2c, 即包含 First data piece 和 Last data piece 以及 Head piece of row, 在块头表示为 --H-FL--; 而 0c, 则不包含 Head piece of row, 这意味着数据行出现了行迁移, 在块头的表示则为----FL--。

我们可以用 BBED 来进行一些判断和验证, 以下是配置信息, 块大小设置为 512 Byte, 这是在测试库上进行的一个后续分析:

```
[u@h]$ cat par.txt
mode=edit
blocksize=512
listfile=data.txt
[u@h]$ more data.txt
1          /opt/oracle/data/1_12446.dbf
```

启动 BBED, 定位到 166733 块, 偏移量 160 即为块头标志:

```
[oracle@db880 lib]$ bbed parfile=par.txt
Password: blockedit
```

```
BBED: Release 2.0.0.0.0 - Limited Production on Sun Jan 15 20:35:08 2012
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.
***** !!! For Oracle Internal Use only !!! *****
```

```
BBED> set block 166733 offset 160
```

BLOCK# 166733

BBED> dump

File: /opt/oracle/data/1_12446.dbf (1)

Block: 166733 Offsets: 160 to 511 DbA:0x00428b4d

```
-----
0c000000 002f2102 ffff0000 00010000 0007000f 35040014 80010100 0b050001
0000002f 0bc01db2 4be1efdb 00010000 01020000 000c0018 001d0004 00010002
01010000 00000000 00030022 0019482c 00801542 af170500 0bc01db2 0900e509
12ff0501 03000000 0c030000 002f2102 00010000 000104b1 0007000f 31000000
c3052a22 00000190 01010001 4be86fba 0502001f 00000002 00800079 4be86fad
00010000 01000000 00040020 00170008 0019a567 008003fa b1913e00 00120088
00000000 00000000 00000000 05010020 00000002 008003fa 4be86fac 00010000
03000000 000e0014 00300020 001d0002 00070000 00880328 00120008 00080017
0019a567 b1913e00 00010141 00012b97 00000008 00000000 0b011700 00080001
008003fa b1913b00 4be86b4a 00010000 4be86b9d 00010000 04010000 00000000
00040017 0019b223 00800434 b1c73a00 80000001 4be84f0d 0b81445b 09007e49
```

<32 bytes per line>

现在的块头标记为 0c，将块头修改为正常值 2c:

BBED> modify /x 2c000000

File: /opt/oracle/data/1_12446.dbf (1)

Block: 166733 Offsets: 160 to 511 DbA:0x00428b4d

```
-----
2c000000 002f2102 ffff0000 00010000 0007000f 35040014 80010100 0b050001
0000002f 0bc01db2 4be1efdb 00010000 01020000 000c0018 001d0004 00010002
01010000 00000000 00030022 0019482c 00801542 af170500 0bc01db2 0900e509
12ff0501 03000000 0c030000 002f2102 00010000 000104b1 0007000f 31000000
c3052a22 00000190 01010001 4be86fba 0502001f 00000002 00800079 4be86fad
```

接下来解析日志，LOGMNR 会提示 166733 块损坏，这是因为 BBED 修改的问题导致的，我们需要将日志的标记位修改一下：

SQL> exec dbms_logmnr.add_logfile('/opt/oracle/data/1_12446.dbf');

PL/SQL procedure successfully completed.

SQL> exec dbms_logmnr.start_logmnr

PL/SQL procedure successfully completed.

SQL> select SESSION_INFO from v\$logmnr_contents where rbablk=166733 and data_obj#=66237;

*

ERROR at line 1:

ORA-00334: archived log: '/opt/oracle/data/1_12446.dbf'

修改 REDO 块标志，将 checksum 清空：

BBED> set block 166733

BLOCK# 166733

BBED> set offset 14

OFFSET 14

BBED> dump

File: /opt/oracle/data/1_12446.dbf (1)

Block: 166733 Offsets: 14 to 511 DbA:0x00428b4d

fb830000 01140101 00014be8 6fb90501 00160000 00020080 15424be8 6efe0001
00000100 00000010 00140018 0020001d 00040001 00010074 1d6a0022 00000003
00220019 482caf17 05000001 02bd0001 05ca0000 00080000 00000b01 22040000
00000401 00000000 0000000a 001e001a 0c960080 0c0ab193 29008000 00014bb7
744a0bc0 1db20900 e50912ff 05010300 00002c00 0000002f 2102ffff 00000001
00000007 000f3504 00148001 01000b05 00010000 002f0bc0 1db24be1 efdb0001
00000102 0000000c 0018001d 00040001 00020101 00000000 00000003 00220019

BBED> modify /x 0000

Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) Y

File: /opt/oracle/data/1_12446.dbf (1)

Block: 166733 Offsets: 14 to 511 DbA:0x00428b4d

00000000 01140101 00014be8 6fb90501 00160000 00020080 15424be8 6efe0001
00000100 00000010 00140018 0020001d 00040001 00010074 1d6a0022 00000003
00220019 482caf17 05000001 02bd0001 05ca0000 00080000 00000b01 22040000
00000401 00000000 0000000a 001e001a 0c960080 0c0ab193 29008000 00014bb7
744a0bc0 1db20900 e50912ff 05010300 00002c00 0000002f 2102ffff 00000001

```
00000007 000f3504 00148001 01000b05 00010000 002f0bc0 1db24be1 efdb0001
00000102 0000000c 0018001d 00040001 00020101 00000000 00000003 00220019
```

再次解析日志，注意此时 SQL_REDO 就能够被正确解析出来，这里的核心在于 ROWID 信息，更新通过 ROWID 强制，对于行迁移记录，缺省的并不记录头块的 ROWID，则意味着 LOGMNR 无法解析出真实的 ROWID，对于常规记录则没有问题。

我们现在修改了日志，标记记录为正常，则 ROWID 被当做块头 ROWID 被解析出来，但是按照这个 ROWID 去执行重做，则可能出现问题：

```
SQL> exec dbms_logmnr.add_logfile('/opt/oracle/data/1_12446.dbf');
PL/SQL procedure successfully completed.
SQL> exec dbms_logmnr.start_logmnr
PL/SQL procedure successfully completed.
SQL> select sql_redo from v$logmnr_contents where rbablk=166733 and data_obj#=66237;
SQL_REDO
```

```
-----
update "UNKNOWN"."OBJ# 66237" set "COL 8" = HEXTORAW('31'), "COL 16" =
HEXTORAW('c305') where "COL 8" = HEXTORAW('35') and "COL 16" = HEXTORAW('80') and
ROWID = 'AAAQXKAAvAAAB2yAAv';
```

接下来让我们通过测试来验证一下以上的分析和判断。

首先创建一个测试表，ncom 字段被设置为 4000 字符，这使得一个 8K 的数据块只能存储两条足位存储的记录：

```
SQL> create table eygle (name varchar2(10),ncom varchar2(4000));
Table created.
```

插入三条记录，初始的，这三条记录位于同一个数据块中：

```
SQL> insert into eygle values('E1','a');
1 row created.
SQL> insert into eygle values('E2','b');
1 row created.
SQL> insert into eygle values('E3','c');
1 row created.
SQL> commit;
Commit complete.
```

通过 ROWID 查询可以确认这几条记录位于同一个数据块:

SQL> select name,rowid,dbms_rowid.rowid_block_number(rowid) block_number from eygle;

NAME	ROWID	BLOCK_NUMBER
-----	-----	-----
E1	AAAEURAABAAKVxAAA	42353
E2	AAAEURAABAAKVxAAB	42353
E3	AAAEURAABAAKVxAAC	42353

执行一次全表扫描查询，注意查询产生了 4 个逻辑读:

SQL> set autotrace on

SQL> select name,rowid,dbms_rowid.rowid_block_number(rowid) block_number from eygle;

NAME	ROWID	BLOCK_NUMBER
-----	-----	-----
E1	AAAEURAABAAKVxAAA	42353
E2	AAAEURAABAAKVxAAB	42353
E3	AAAEURAABAAKVxAAC	42353

Execution Plan

Plan hash value: 4048929491

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	57	2 (0)	00:00:01
1	TABLE ACCESS FULL	EYGLE	3	57	2 (0)	00:00:01

Statistics

0 recursive calls
0 db block gets
4 consistent gets

0 physical reads

接下来更新这三条记录，当记录被充满之后，一个 8k 的数据块将不足以存储这三条记录：

```
SQL> update eygle set ncom=lpad('a',4000,'a') where name='E1';
1 row updated.
SQL> update eygle set ncom=lpad('b',4000,'b') where name='E2';
1 row updated.
SQL> update eygle set ncom=lpad('c',4000,'c') where name='E3';
1 row updated.
SQL> commit;
Commit complete.
```

现在的查询显示，这个 SQL 消耗了五个逻辑读，教之前多出一次，这就是因为有一行记录发生了行迁移，多了一个数据块的读取操作：

```
SQL> set autotrace on
SQL> select name,rowid,dbms_rowid.rowid_block_number(rowid) block_number from eygle;
```

NAME	ROWID	BLOCK_NUMBER
-----	-----	-----
E1	AAAEURAABAAKVxAAA	42353
E2	AAAEURAABAAKVxAAB	42353
E3	AAAEURAABAAKVxAAC	42353

Execution Plan

```
-----
Plan hash value: 4048929491
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	

0	SELECT STATEMENT		3	57	2 (0)	00:00:01	
1	TABLE ACCESS FULL	EYGLE	3	57	2 (0)	00:00:01	

Statistics

```

-----
0 recursive calls
0 db block gets
5 consistent gets
0 physical reads
0 redo size

```

接下来转储数据块观察一下：

```

SQL> alter system dump datafile 1 block min 42353 block max 42354;
System altered.

```

跟踪文件中记录的信息通过裁剪，简要记录如下：

```

Block header dump: 0x0040a571
Object id on Block? Y
seg/obj: 0x4510 csc: 0x00.3ea00 itc: 2 flg: - typ: 1 - DATA
fsl: 0 fnx: 0x0 ver: 0x01

```

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0006.00c.0000010c	0x00c00baa.0042.25	--U-	3	fsc 0x0000.0003ea89
0x02	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000

```

bdba: 0x0040a571
data_block_dump,data header at 0xeab405c
=====

```

```

tsiz: 0x1fa0
hsiz: 0x18
pbl: 0x0eab405c
76543210
flag=-----
ntab=1
nrow=3
frre=-1
fsbo=0x18
fseo=0x2d
avsp=0x2d

```

```

tosp=0x2d
0xe:pti[0]   nrow=3   offs=0
0x12:pri[0]  offs=0xfdf
0x14:pri[1]  offs=0x36
0x16:pri[2]  offs=0x2d
block_row_dump:
tab 0, row 0, @0xfdf

```

注意这里每一行的行头上都有一个标志为 FB - Flag Bit,正常行的标记位为 2c 即如下的--H-FL-- :

```

t1: 4009 fb: --H-FL-- lb: 0x1 cc: 2
col 0: [ 2] 45 31
col 1: [4000]
61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
tab 0, row 1, @0x36
t1: 4009 fb: --H-FL-- lb: 0x1 cc: 2
col 0: [ 2] 45 32
col 1: [4000]
62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62
tab 0, row 2, @0x2d

```

注意这里的迁移行，起头标记仅有一个 H，意味着仅包含头块的地址信息：

```

t1: 9 fb: --H----- lb: 0x1 cc: 0
nrid: 0x0040a572.0
end_of_block_dump

```

注意,这里的 nrid 指对于行链接或者行迁移来说的下一个 rowid 的值,这里的下一个 ROWID 指向 40a572, 以下的转储信息正是这个数据块的内容:

```

Block dump from disk:
buffer tsn: 0 rdba: 0x0040a572 (1/42354)
scn: 0x0000.0003ea89 seq: 0x01 flg: 0x06 tail: 0xea890601
frmt: 0x02 chkval: 0x9ffc type: 0x06=trans data
Hex dump of block: st=0, typ_found=1
Block header dump: 0x0040a572
Object id on Block? Y

```

```
seg/obj: 0x4510 csc: 0x00.3ea86 itc: 3 flg: 0 typ: 1 - DATA
fsl: 0 fnx: 0x0 ver: 0x01
```

Itl	Xid	Uba	Flag	Lck	Scn/Fsc
0x01	0x0006.00c.0000010c	0x00c00baa.0042.24	--U-	1	fsc 0x0000.0003ea89
0x02	0x0000.000.00000000	0x00000000.0000.00	----	0	fsc 0x0000.00000000
0x03	0x0000.000.00000000	0x00000000.0000.00	C---	0	scn 0x0000.00000000

```
bdba: 0x0040a572
```

```
data_block_dump,data header at 0xeab4074
```

```
=====
```

```
tsiz: 0x1f88
```

```
hsiz: 0x14
```

```
pbl: 0x0eab4074
```

```
76543210
```

```
flag=-----
```

```
ntab=1
```

```
nrow=1
```

```
frre=-1
```

```
fsbo=0x14
```

```
fseo=0xfd9
```

```
avsp=0xfc5
```

```
tosp=0xfc5
```

```
0xe:pti[0] nrow=1 offs=0
```

```
0x12:pri[0] offs=0xfd9
```

```
block_row_dump:
```

```
tab 0, row 0, @0xfd9
```

```
tl: 4015 fb: ----FL-- lb: 0x1 cc: 2
```

注意这里的 hrid 指 Head ROWID，指向头块的地址，即块 40a571,其 FL 标记表明这是一个迁移行，在数据块的内部，原数据块和迁移块上都有 ROWID 的记录：

```
hrid: 0x0040a571.2
```

```
col 0: [ 2] 45 33
```

```
col 1: [4000]
```

```
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63
```

```
end_of_block_dump
```

```
End dump data blocks tsn: 0 file#: 1 minblk 42353 maxblk 42354
```

当更新记录导致行迁移时，由于 ROWID 的问题，日志可能无法解析出正确的 SQL，所以 LOGMNR 直接将这些记录标记为不支持。

但是当启用附加日志之后，日志可以被正常解析，LOGMNR 会解析出相关的 SQL_REDO 信息。

以下是对附加日志的测试过程：

```
SQL> drop table eygle purge;
```

```
Table dropped.
```

```
SQL> alter system switch logfile;
```

```
System altered.
```

```
SQL> create table eygle (name varchar2(10),ncom varchar2(4000));
```

```
Table created.
```

```
SQL> insert into eygle values('E1','a');
```

```
1 row created.
```

```
SQL> insert into eygle values('E2','b');
```

```
1 row created.
```

```
SQL> insert into eygle values('E3','c');
```

```
1 row created.
```

对于数据库启用附加日志，然后进行更新，引发迁移：

```
SQL> alter database add supplemental log data;
```

```
Database altered.
```

```
SQL> select name,rowid,dbms_rowid.rowid_block_number(rowid) block_number from eygle;
```

NAME	ROWID	BLOCK_NUMBER
E1	AAAEUTAABAAAKVxAAA	42353
E2	AAAEUTAABAAAKVxAAB	42353
E3	AAAEUTAABAAAKVxAAC	42353

```
SQL> update eygle set ncom=lpad('a',4000,'a') where name='E1';
```

```
1 row updated.
```

```
SQL> update eygle set ncom=lpad('b',4000,'b') where name='E2';
```

```
1 row updated.
```


1.行迁移的日志说明

当行迁移首次发生时，ROWID 都能够被正确记录，只不过缺省的 LOGMNR 解析时会跳过这些迁移行。

如下日志信息中，可以看到迁移行的 hrid 信息：

```
REDO RECORD - Thread:1 RBA: 0x02a30f.00000003.0104 LEN: 0x108c VLD: 0x01  
SCN: 0x081f.404726af SUBSCN: 1 01/16/2012 13:09:03  
CHANGE #1 TYP:0 CLS:32 AFN:2 DBA:0x00800422 SCN:0x081f.404726af SEQ: 1 OP:5.1  
ktudb redo: siz: 60 spc: 4198 flg: 0x0022 seq: 0xbc11 rec: 0x21  
xid: 0x0008.000.0012b96e  
ktubu redo: slt: 0 rci: 32 opc: 11.1 objn: 38921 objd: 38921 tsn: 0  
Undo type: Regular undo Undo type: Last buffer split: No  
Tablespace Undo: No  
0x00000000  
KDO undo record:  
KTB Redo  
op: 0x03 ver: 0x01  
op: Z  
KDO Op code: DRP row dependencies Disabled  
xtype: XA bdba: 0x0040992b hdba: 0x00409929  
itli: 1 ispac: 0 maxfr: 4863  
tabn: 0 slot: 0(0x0)  
CHANGE #2 TYP:0 CLS: 1 AFN:1 DBA:0x0040992b SCN:0x081f.404726af SEQ: 4 OP:11.2  
KTB Redo  
op: 0x01 ver: 0x01  
op: F xid: 0x0008.000.0012b96e uba: 0x00800422.bc11.21  
KDO Op code: IRP row dependencies Disabled  
xtype: XA bdba: 0x0040992b hdba: 0x00409929  
itli: 1 ispac: 24 maxfr: 4863  
tabn: 0 slot: 0(0x0) size/delt: 4015  
fb: ----FL-- lb: 0x1 cc: 2  
hrid: 0x0040992a.2  
null: --  
col 0: [ 2] 45 33  
col 1: [4000]  
63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63 63
```

2.迁移行的再次更新

当 DML 操作再次更新迁移行时，Hrid 不会被记录，此时 Flag 标记显示为 0x0c，为迁移行：

[illegible]

3.附加日志信息与 ROWID

当启用附加日志后，附件 ROWID 信息可以从日志文件的二进制编码中解析出来，缺省的日志转储不显示该 ROWID 信息，以下是日志转储信息的示范，可以注意到其末尾记录有 ROWID 信息：

```
00003410h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003420h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003430h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003440h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003450h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003460h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003470h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003480h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003490h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034a0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034b0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034c0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034d0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034e0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000034f0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003500h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003510h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003520h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003530h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003540h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003550h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003560h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003570h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003580h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003590h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035a0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035b0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035c0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035d0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035e0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
000035f0h: 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 ; ddddddddddddddd
00003600h: 00 02 A3 11 00 00 00 1B 2E 0E CF A2 00 00 42 A4 ; ..?...息..B?
```

正常的日志转储不显示附加信息是 Oracle 的内部开关问题， 附加日志信息（SUPPLEMENTAL DATA）缺省的不会转储出来，从 Oracle 10g 开始，启用附加日志转储需要设置两个事件：

```
alter session set events = '1354 trace name context forever, level 32768';
alter session set events = '1348 trace name context forever, level 1032';
```

设置了这两个事件之后，日志转储会显示类似如下附加日志信息：

```
LOGMINER DATA 10i:
opcode: INSERT
Number of columns supplementally logged: 0  Flag: SE kdogspare1: 0x0 [ ] Objv#: 1
```

```

segcol# in Undo starting from 0
segcol# in Redo starting from 1

```

以下对之前的测试做了进一步的测试，可以获得直观的信息：

```

SQL> alter system switch logfile;
System altered.
SQL> connect eygle/eygle
Connected.
SQL> update eygle set ncom=lpad('c',4000,'c') where name='E3';
1 row updated.
SQL> commit;
Commit complete.
SQL> select group#,sequence#,status from v$log;
  GROUP#  SEQUENCE#  STATUS
-----
1         145  INACTIVE
2         146  CURRENT
3         144  INACTIVE
SQL> alter system dump logfile '/home/ora11g/oradata/orcl11g/redo02.log';
System altered.
SQL> alter session set events = '1354 trace name context forever, level 32768 ';
Session altered.
SQL> alter session set events = '1348 trace name context forever, level 1032 ';
Session altered.
SQL> alter system dump logfile '/home/ora11g/oradata/orcl11g/redo02.log';
System altered.

```

对比前后两次的跟踪文件，可以显著发现不同之处，附加日志的解析内容被清晰的记录下来，以下日志显示了 Head Rowid 的内容：

```
LOGMINER DATA 10i:
Head Rowid (dba: 101cd9f, tabn: 0, sno: 2) opcode: UPDATE
Number of columns supplementally logged: 0 Flag: E kdogspare1: 0x1000 [ NSRCI] Objv#: 1
seqcol# in Undo starting from 2
seqcol# in Redo starting from 2
CHANGE #2 TYP:2 CLS:1 AFN:4 DBA:0x0101cd9b OBJ:33681 SCN:0x0000.008f0162 SEQ:1 OP:11.5 ENC:0 RBL:0
KTB Redo
op: 0x11 ver: 0x01
compat bit: 4 (post-11) padding: 1
op: F xid: 0x0003.002.000024fd uba: 0x00c2f265.05d6.15
Block cleanup record, scn: 0x0000.008f0194 ver: 0x01 opt: 0x02, entries follow...
itli: 2 flg: 2 scn: 0x0000.008f0162
KDO Op code: URP row dependencies Disabled
xtype: XA flags: 0x00000000 bdba: 0x0101cd9b hdba: 0x0101cd9a
itli: 3 ispac: 0 maxfr: 4858
tabn: 0 slot: 0(0x0) flag: 0x0c lock: 3 ckix: 0
ncol: 2 nnew: 1 size: 0
```

在 Oracle 10g 中，以上操作是以 UPDATE 完成的，Oracle 将相关字段的内容更新为 NULL，然后写入 nrid 信息：

```
SQL> select RBASQN,RBABLK,SQL_REDO from v$logmnr_contents;
```

RBASQN	RBABLK	SQL_REDO
196	2	set transaction read write;
196	2	Unsupported
196	3	Unsupported
196	12	update "UNKNOWN"."OBJ# 61278" set "COL 1" = NULL, "COL 2" = NULL where "COL 1" = HEXTORAW('4533') and "COL 2" = HEXTORAW('63') and ROWID = 'AAA09eAABAAAPxaAAC';
196	13	commit;

5.行溢出时信息

REDO 记录了额外的信息如下，前者操作代码为 5.1，是 UNDO 信息；后者 11.6 是行溢出信息，将原行位置更新为 nrid 信息：

```
REDO RECORD - Thread:1 RBA: 0x0000be.0000000c.0090 LEN: 0x0118 VLD: 0x01
SCN: 0x0000.00412ded SUBSCN: 5 01/16/2012 11:49:47
CHANGE #1 TYP:0 CLS:32 AFN:2 DBA:0x00800058 OBJ:4294967295 SCN:0x0000.00412ded SEQ: 1 OP:5.1
ktudb redo: siz: 112 spc: 1646 flg: 0x0022 seq: 0x07ee rec: 0x33
          xid: 0x0008.018.00000c32
ktubu redo: slt: 24 rci: 50 opc: 11.1 objn: 61275 objd: 61275 tsn: 0
Undo type: Regular undo          Undo type: Last buffer split: No
Tablespace Undo: No
          0x00000000
KDO undo record:
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x00800058.07ee.31
KDO Op code: ORP row dependencies Disabled
  xtype: XA flags: 0x00000000 bdba: 0x0040fc5a hdba: 0x0040fc59
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 2(0x2) size/delt: 8
fb: --H-FL-- lb: 0x1 cc: 2
null: --
col 0: [ 2] 45 33
col 1: [ 1] 63
CHANGE #2 TYP:0 CLS: 1 AFN:1 DBA:0x0040fc5a OBJ:61275 SCN:0x0000.00412dec SEQ: 3 OP:11.6
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x00800058.07ee.33
KDO Op code: ORP row dependencies Disabled
  xtype: XA flags: 0x00000000 bdba: 0x0040fc5a hdba: 0x0040fc59
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 2(0x2) size/delt: 9
fb: --H----- lb: 0x1 cc: 0
nrid: 0x0040fc5b.0
null:
```

对于 UNDO 的操作代码，记录如下：

OP Code	Description
5.1	Undo block or undo segment header
5.2	Update rollback segment header
5.4	Commit transaction
5.11	Rollback DBA in transaction table entry
5.19	Transaction start audit log record
5.20	Transaction continue audit log record

对于 DML 操作代码，其含义如下：

OP Code	Description
11.2	Insert Row Piece
11.3	Drop Row Piece
11.4	Lock Row Piece
11.5	Update Row Piece
11.6	Overflow Row Piece
11.11	Insert Row Array
11.12	Delete Row Array

密码安全与加密

2011 年底，一系列的网站被爆出密码泄露事件，一时间沸沸扬扬，数以千万计的用户密码被泄露，明文密码成了罪恶之源。其实在数据库中通过适当加密对敏感数据进行安全防护是很方便的，拒绝明文密码就能够保护很多用户的隐私。

本节我们将简要介绍一下密码防护和数据安全。

Oracle 数据库的用户信息及密码存储于一个名为 `USER$` 的数据表中（所有者为 `SYS` 用户），我们可以通过基于 `USER$` 表建立的 `DBA_USERS` 视图来查询和获得这些信息，包括加密的口令串。

在 Oracle Database 11g 之前，用户口令通过 DES 加密算法进行加密，使用用户名作为“Salt”，密码最长为 30 个字符，所有字母被强制转换为大写。从 Oracle 7 至 Oracle 10g，加密一直使用 username 和 password 串连之后进行 HASH 运算。

从 Oracle Database 11g 开始，Oracle 允许最多使用 30 个字符、大小写混合方式作为密码，同时支持 DES 和 SHA-1 算法进行加密（SHA-1 算法支持大小写混合，通过初始化参数 `SEC_CASE_SENSITIVE_LOGON` 开关），使用 `password||salt` 的方式进行 HASH 加密。

我们稍微解释一下 SALT 的含义：

SALT 作为“盐”的词义，在密码中经常作为一种添加“佐料”出现，作为干扰字符，以提升密码加密安全。通常如果我们直接对密码进行散列加密，那么黑客也就可以通过对一个已知密码进行散列，然后通过对比散列值得到某用户的密码。

加 SALT 可以在一定程度上解决这一问题，当用户首次提供密码时，由系统自动向这个密码里撒一些“佐料”，然后再进行散列。而当用户登录时，系统为用户提供的代码附加同样的“佐料”，然后散列，再比较散列值，以确定密码是否正确，这样就提高了加密的安全性。通常这个 SALT 既可以由用户提供，也可以随机生成，或者为数据库指定规定的 SALT 值。

以下是 Oracle 9i 数据库中口令的加密形式，`DBA_USERS` 视图的 `PASSWORD` 字段显示了加密后的密钥：

```
SQL> select username,password from dba_users
2 where username in ('SYS','SYSTEM','EYGLE');
USERNAME      PASSWORD
-----
EYGLE         B726E09FE21F8E83
SYSTEM        A204A4CEB2C2F2FB
SYS           7ABF43E21B3DD9A3
```

在 Oracle 11g 中，密码从 DBA_USERS 视图中隐藏起来，这进一步的增强了安全性，即便具有访问视图权限的用户，也无法获得口令的加密串，由此我们也可以看出 Oracle 数据库软件的安全增强历程：

```
SQL> select username,password from dba_users
2 where username in ('SYS','SYSTEM','EYGLE');
USERNAME      PASSWORD
-----
EYGLE
SYS
SYSTEM
```

口令的加密内容存储在底层的核心表（USER\$是 Oracle 数据库的元数据表之一）中，以下 PASSWORD 字段存储的是 DES 加密值，SPARE4 存储的是 SHA-1 加密信息：

```
SQL> select * from v$version where rownum <2;
BANNER
-----
Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production

SQL> select name,password,spare4 from user$
2 where name in ('SYS','SYSTEM','EYGLE');
NAME      PASSWORD      SPARE4
-----
SYS       8A8F025737A9097A S:BBEFCBB86319E6A40372B9584D8CCA6B0158FE0C7DDF5B9593FB618E0D80
SYSTEM    2D594E86F93B17A1 S:C576FB5A54D009440AC047827392215C673528067BC06659EC56E3178BAB
EYGLE     B726E09FE21F8E83 S:65857F36842AEE4470828E9BE630FEED90A67CEF0D2B40C9FE9B558F6B49
```

关于口令的维护，Oracle 支持各种约束性限制（通过 utlpwdmg.sql 脚本启用），诸如复杂程度、长度、有效期、失败登陆次数等等，通过这些增强，Oracle 的口令限制可以定制出非常稳固的安全解决方案，如果你从未接触和研究过这些手段，那么可能就说明你的数据库还缺乏足够的第一层的安全防守。

如果用户设计的程序也能够对密码进行加密，哪怕使用最简单的加密算法，那么互联网上铺天盖地的密码泄露事件就不会那么突出和损失惨重了。

对于 Oracle 数据库的用户口令，从 Oracle 7 到 Oracle 10gR2，是使用 DES 算法对口令进行加密的，以下是 DES 算法的简要介绍：

DES 算法为密码体制中的对称密码体制，又被称为美国数据加密标准，是 1972 年美国 IBM 公司研制的对称密码体制加密算法。其密钥长度为 56 位，明文按 64 位进行分组，将分组后的明文组和 56 位的密钥按位替代或交换形成密文组。DES 加密算法特点：分组比较短、密钥太短、密码生命周期短、运算速度较慢。

DES 工作的基本原理是，其入口参数有三个:key、data、mode。key 为加密解密使用的密钥，data 为加密解密的数据，mode 为其工作模式。当模式为加密模式时，明文按照 64 位进行分组，形成明文组，key 用于对数据加密，当模式为解密模式时，key 用于对数据解密。

实际运用中，密钥只用到了 64 位中的 56 位，这样才具有高的安全性。DES(Data Encryption Standard)算法，于 1977 年得到美国政府的正式许可，是一种用 56 位密钥来加密 64 位数据的方法。

Oracle 的口令加密是将用户名和口令联合起来，按 64 位进行分组，再进行加密，而且以用户名作为“Salt”，这种方式存在的一个缺陷是，只要用户名和口令相同，在任何数据库中计算出来的口令都是相同的，而且由于用户名和口令是连排的，所以理论上对于类似“EYGLEEYGLE”这样的 username + password 字符串，如何分组得到的密钥都是相同的：

```
SQL> create user eygle identified by eygle;
User created.

SQL> create user eyg identified by leeygle;
User created.

SQL> select username,password from dba_users where username like 'EYG%';
```

USERNAME	PASSWORD
EYGLE	B726E09FE21F8E83
EYG	B726E09FE21F8E83

Oracle 数据库内部，通过内置的加密包可以实现密码或关键数据加密，这是一种简单有效的加密方式，可以实现密码防护和数据保护。

从 Oracle 8i 开始，dbms_obfuscation_toolkit 工具包被引入到 Oracle 数据库中，利用这个内置在数据库中的工具包，我们可以实现对于数据的加密和解密过程，实现数据保护。

dbms_obfuscation_toolkit 工具包中主要有以下几个存储过程：

过程名称	功 能
PROCEDURE DES3DECRYPT	用于 Triple DES 算法解密数据
PROCEDURE DES3ENCRYPT	用于 Triple DES 算法加密数据
PROCEDURE DES3GETKEY	基于 Triple DES 算法产生密钥
PROCEDURE DESDECRYPT	用于 DES 算法解密数据
PROCEDURE DESENCRYPT	用于 DES 算法加密数据
PROCEDURE DESGETKEY	基于 DES 算法产生密钥
PROCEDURE MD5	用于 MD5 算法加密数据

对应以上 7 个过程,该工具包中还包含了 14 个函数,每个过程提供两个函数分别返回 RAW 和 VARCHAR2 类型输出:

函数名称	返回值类型	功 能
FUNCTION DES3DECRYPT	RAW VARCHAR2	返回 Triple DES 解密串
FUNCTION DES3ENCRYPT	RAW VARCHAR2	返回 Triple DES 加密串
FUNCTION DES3GETKEY	RAW VARCHAR2	返回 Triple DES 密钥
FUNCTION DESDECRYPT	RAW VARCHAR2	返回 DES 解密串
FUNCTION DESENCRYPT	RAW VARCHAR2	返回 DES 加密串
FUNCTION DESGETKEY	RAW VARCHAR2	返回 DES 密钥
FUNCTION MD5	RAW VARCHAR2	返回 MD5 加密串

dbms_obfuscation_toolkit 工具包允许加密 VARCHAR2 和 RAW 类型的数据,但是由于 VARCHAR2 类型的不同字符集的数据库之间转换可能出现错误,所以我们通常推荐对 VARCHAR2 类型处理前先通过函数 utl_raw.cast_to_raw 进行 RAW 型转换,当解密后通过 utl_raw.cast_to_varchar2 转换为 VARCHAR2 类型。

通过 `dbms_obfuscation_toolkit` 包可以对数据进行 DES、Triple DES 或者 MD5 加密，虽然各种技术手段不断衍生出来用于加密破解，但是对于数据的基本加密是每个数据维护者都应当考虑的事情。以下通过实例来了解一下这个加密包的用法。

1. MD5 加密

以下是关于 MD5 算法的简要介绍：

MD5 广泛用于操作系统的登陆认证上，如 Unix、各类 BSD 系统登录密码、数字签名等诸多方面。如在 UNIX 系统中用户的密码是以 MD5（或其它类似的算法）经 Hash 运算后存储在文件系统中。当用户登录的时候，系统把用户输入的密码进行 MD5 Hash 运算，然后再去和保存在文件系统中的 MD5 值进行比较，进而确定输入的密码是否正确。通过这样的步骤，系统在并不知道用户密码的明文的情况下就可以确定用户登录系统的合法性。这可以避免用户的密码被具有系统管理员权限的用户知道。

MD5 将任意长度的“字节串”映射为一个 128bit 的大整数，并且是通过该 128bit 反推原始字符串是困难的，换句话说就是，即使你看到源程序和算法描述，也无法将一个 MD5 的值变换回原始的字符串，从数学原理上说，是因为原始的字符串有无穷多个。正是因为这个原因，现在被黑客使用最多的一种破解密码的方法就是一种被称为“跑字典”的方法。有两种方法得到字典，一种是日常搜集的用做密码的字符串表，另一种是用排列组合方法生成的，先用 MD5 程序计算出这些字典项的 MD5 值，然后再用目标的 MD5 值在这个字典中检索。

MD5 加密是最为简单的过程，以下是 MD5 相关的函数和过程：

```
DBMS_OBFUSCATION_TOOLKIT.MD5(
    input          IN RAW,
    checksum       OUT raw_checksum);
DBMS_OBFUSCATION_TOOLKIT.MD5(
    input_string   IN VARCHAR2,
    checksum_string OUT varchar2_checksum);
DBMS_OBFUSCATION_TOOLKIT.MD5(
    input          IN RAW)
RETURN raw_checksum;
DBMS_OBFUSCATION_TOOLKIT.MD5(
    input_string   IN VARCHAR2)
RETURN varchar2_checksum;
```

以下过程完成了对于一个非加密口令表的加密过程，当然这个过程的更改要同程序的更改相结合：

```
SQL> drop table endeup purge;
```

```
Table dropped.
```

```
SQL> create table endeup(
```

```
2  userid varchar2(10),
```

```
3  passwd varchar(20),
```

```
4  enpswd char(32)
```

```
5  );
```

```
Table created.
```

```
SQL> insert into endeup values ('eygle1','oracle',null);
```

```
1 row created.
```

```
SQL> insert into endeup values ('eygle2','oracle123',null);
```

```
1 row created.
```

```
SQL> insert into endeup values ('eygle3','Oracle@)!'),null);
```

```
1 row created.
```

```
SQL> commit;
```

```
Commit complete.
```

```
SQL> CREATE OR REPLACE FUNCTIONMD5(
```

```
2  passwd IN VARCHAR2)
```

```
3  RETURN VARCHAR2
```

```
4  IS
```

```
5  retval varchar2(32);
```

```
6  BEGIN
```

```
7  retval := utl_raw.cast_to_raw(
```

```
DBMS_OBFUSCATION_TOOLKIT.MD5(INPUT_STRING => passwd)) ;
```

```
8  RETURNretval;
```

```
9  END;
```

```
10 /
```

```
Function created.
```

```
SQL> update endeup set enpswd=MD5(passwd);
```

```
3 rows updated.
```

```
SQL> commit;
```

```
Commit complete.
```

```
SQL> ALTER TABLE endeup DROP COLUMN passwd;
Table altered.
SQL> ALTER TABLE endeup RENAME COLUMN enpswd to passwd;
Table altered.
SQL> select * from endeup;
USERID      PASSWD
-----
eygle1      A189C633D9995E11BF8607170EC9A4B8
eygle2      754AC3325A7F835BD7B4FD99F85C25FF
eygle3      373AAEF24027643131DE91CE3B6F2761
```

将口令通过 MD5 加密之后,用户登陆输入口令,通过 MD5 函数进行运算后同数据库中的加密串进行比较,得出登陆判定。

由于 MD5 的不可逆性, MD5 加密能够对密码安全起到基本的保护和保密作用。

2.3DES 加密

DES 即 Data Encryption Standard - 数据加密标准, 诞生于 1970 年左右, 于 1976 年被美国政府选为国家标准, 随后被广泛采用。DES 基于对称密钥算法, 使用 56 位密钥, 对于很多应用, 目前 56 位密钥显得过短, 所以 DES 被认为不够安全, 随之 3DES 标准被引入, 用于强化 DES 标准。

以下是 3DES 算法的简要介绍:

3DES (或称为 Triple DES) 是三重数据加密算法 (TDEA, Triple Data Encryption Algorithm) 块密码的通称。它相当于是对每个数据块应用三次 DES 加密算法。由于计算机运算能力的增强, DES 密码的密钥长度变得容易被暴力破解; 3DES 即是设计用来提供一种相对简单的方法, 即通过增加 DES 的密钥长度来避免类似的攻击, 而不是设计一种全新的块密码算法。

DES 使用 56 位密钥和密码块的方法, 而在密码块的方法中, 文本被分成 64 位大小的文本块然后再进行加密。3DES 是 DES 加密算法的一种模式, 使用 3 条 56 位的密钥对数据进行三次加密。

以下是 3DES 的加密过程, 其中 which 参数指加密模式, 缺省值为 0 是 TwoKeyMode, 值为 1 代表的是 ThreeKeyMode:

```
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
    input          IN      RAW,
```

```

key          IN    RAW,
encrypted_data OUT  RAW,
which        IN    PLS_INTEGER DEFAULT TwoKeyMode
iv           IN    RAW          DEFAULT NULL);
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
input_string IN    VARCHAR2,
key_string   IN    VARCHAR2,
encrypted_string OUT VARCHAR2,
which        IN    PLS_INTEGER DEFAULT TwoKeyMode
iv_string    IN    VARCHAR2    DEFAULT NULL);
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
input      IN RAW,
key        IN RAW,
which      IN PLS_INTEGER DEFAULT TwoKeyMode
iv         IN RAW          DEFAULT NULL)
RETURN RAW;
DBMS_OBFUSCATION_TOOLKIT.DES3Encrypt(
input_string IN VARCHAR2,
key_string   IN VARCHAR2,
which        IN PLS_INTEGER DEFAULT TwoKeyMode
iv_string    IN VARCHAR2    DEFAULT NULL)
RETURN VARCHAR2;

```

由于 DES 算法以 64 位进行加密运算，所以如果输入加密串小于 8 Bytes 时，DES3DECRYPT 过程会抛出异常：ORA-28234 "Key length too short"。加密串以 8 字节或其倍数被接受，超过的位数会被抛弃，如 9 Bytes 加密串会被抛弃一个 Byte。

解密的过程和函数与加密类似：

```

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
input      IN    RAW,
key        IN    RAW,
decrypted_data OUT  RAW,
which      IN    PLS_INTEGER DEFAULT TwoKeyMode
iv         IN    RAW          DEFAULT NULL);

```

```

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
  input_string      IN   VARCHAR2,
  key_string        IN   VARCHAR2,
  decrypted_string  OUT  VARCHAR2,
  which             IN   PLS_INTEGER DEFAULT TwoKeyMode
  iv_string         IN   VARCHAR2   DEFAULT NULL);
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
  input      IN  RAW,
  key        IN  RAW,
  which      IN  PLS_INTEGER DEFAULT TwoKeyMode
  iv         IN  RAW          DEFAULT NULL)
RETURN RAW;
DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(
  input_string IN  VARCHAR2,
  key_string   IN  VARCHAR2,
  which        IN  PLS_INTEGER DEFAULT TwoKeyMode
  iv_string    IN  VARCHAR2   DEFAULT NULL)
RETURN VARCHAR2;

```

以下是 3DES 加密算法的简要测试用例，我们将密码设置为 8 位以简化测试，实际中可以考虑为不足位数的密码通过特殊字符进行补齐：

```

SQL> drop table endeup purge;
Table dropped.
SQL> create table endeup(
  2  userid varchar2(10),
  3  passwd char(8),
  4  enpswd char(32)
  5 );
Table created.
SQL> insert into endeup values ('eygle1','oracle12',null);
1 row created.
SQL> insert into endeup values ('eygle2','oracle!@',null);
1 row created.
SQL> commit;

```

Commit complete.

SQL> create or replace package MY_ENCDEC_PKG is

```
2   Key_string varchar2(20) := 'keepthesecretkey';
3   raw_key   RAW(128) := UTL_RAW.CAST_TO_RAW(Key_string);
4   function DEC3KEY(iValue in raw, iMode in pls_integer) return varchar2;
5   function ENC3KEY(iValue in varchar2, iMode in pls_integer) return raw;
6 end;
7 /
```

Package created.

SQL> create or replace package body MY_ENCDEC_PKG is

```
2   FUNCTION DEC3KEY(iValue in raw, iMode in pls_integer) return varchar2 as
3       vDecrypted varchar2(4000);
4   begin
5       vDecrypted :=
6           dbms_obfuscation_toolkit.des3decrypt(UTL_RAW.CAST_TO_VARCHAR2(iValue),
7           key_string => raw_key,
8           which => iMode);
9       return vDecrypted;
10  end;
11
12  FUNCTION ENC3KEY(iValue in varchar2, iMode in pls_integer) return raw as
13      vEncrypted   varchar2(4000);
14      vEncryptedRaw Raw(2048);
15  begin
16      vEncrypted := dbms_obfuscation_toolkit.des3encrypt(iValue,
17      key_string => raw_key,
18      which      => iMode);
19
20      vEncryptedRaw := UTL_RAW.CAST_TO_RAW(vEncrypted);
21      return vEncryptedRaw;
22  end;
23  /
```

```
Package body created.
```

```
SQL> col enc for a30
```

```
SQL> col dec for a30
```

```
SQL> SELECT userid,MY_ENCDEC_PKG.ENC3KEY(PASSWD,1) ENC,  
2 MY_ENCDEC_PKG.DEC3KEY(MY_ENCDEC_PKG.ENC3KEY(PASSWD,1),1) DEC FROM ENDEUP;
```

```
USERID      ENC                                DEC
```

```
-----
```

```
eygle1      5E7BA4E986653ECC                        oracle12
```

```
eygle2      DFC0CA37D34A4EEC                        oracle!@
```

以上通过加密解密，可以看到整个过程的可逆性，在加密解密的过程中，密钥是核心的一环，所以需要保护密钥的安全，通过 WRAP 工具对过程进行加密是一个最基本的程序安全防范(虽然 WRAP 已经可以被解密，但是我们预计在 Oracle Database 12c 的版本中，Oracle 可能修改)。以下是使用 WRAP 工具加密代码的一个简单示例。

如下源代码用于对 ROWID 进行基本转换：

```
create or replace function get_rowid  
(l_rowid in varchar2)  
return varchar2  
is  
ls_my_rowid      varchar2(200);  
rowid_type       number;  
object_number    number;  
relative_fno     number;  
block_number     number;  
row_number       number;  
begin  
  dbms_rowid.rowid_info(l_rowid,  
rowid_type,object_number,  
relative_fno, block_number, row_number);  
  ls_my_rowid := 'Object# is      :'||to_char(object_number)||chr(10)||  
                'Relative_fno is :'||to_char(relative_fno)||chr(10)||  
                'Block number is :'||to_char(block_number)||chr(10)||  
                'Row number is   :'||to_char(row_number);
```

```

    return ls_my_rowid ;
end;
/

```

该代码实现如下功能:

```

SQL> @f_get_rowid
Function created.
SQL> select rowid from dept where deptno=10;
ROWID
-----
AAABiPAABAAAFRSAAA
SQL> select get_rowid('AAABiPAABAAAFRSAAA') from dual;
GET_ROWID( 'AAABIPAAABAAAFRSAAA' )
-----
Object# is          :6287
Relative_fno is    :1
Block number is   :21586
Row number is     :0

```

使用 wrap 加密及加密后的代码如下摘要所示:

```

[oracle@jumper tools]$ wrap iname=f_get_rowid.sql oname=f_get_rowid.plb
PL/SQL Wrapper: Release 9.2.0.4.0- Production on Mon Nov 15 21:59:39 2004
Copyright (c) Oracle Corporation 1993, 2001. All Rights Reserved.
Processing f_get_rowid.sql to f_get_rowid.plb

```

```

[oracle@jumper tools]$ cat f_get_rowid.plb
create or replace function get_rowid wrapped
0
abcd
abcd
abcd
abcd
abcd
abcd

```

```

1GET_ROWID:
1L_ROWID:
1VARCHAR2:
1RETURN:
1LS_MY_ROWID:
1200:
1ROWID_TYPE:
1NUMBER:
1OBJECT_NUMBER:
1RELATIVE_FNO:
1BLOCK_NUMBER:
1ROW_NUMBER:
1DBMS_ROWID:
1ROWID_INFO:
1object# is      :::
1||:
1TO_CHAR:
1CHR:
110:
1Relative_fno is :::
1Block number is :::
1Row number is   :::
0

0
0
83
2
0 a0 8d 8f a0 b0 3d b4
:2 a0 2c 6a a3 a0 51 a5 1c
81 b0 a3 a0 1c 81 b0 a3
a0 1c 81 b0 a3 a0 1c 81
b0 a3 a0 1c 81 b0 a3 a0

```

```

1c 81 b0 :2 a0 6b :6 a0 a5 57
a0 6e 7e :2 a0 a5 b b4 2e
7e a0 51 a5 b b4 2e 7e
6e b4 2e 7e :2 a0 a5 b b4
2e 7e a0 51 a5 b b4 2e
7e 6e b4 2e 7e :2 a0 a5 b
b4 2e 7e a0 51 a5 b b4
2e 7e 6e b4 2e 7e :2 a0 a5
b b4 2e d :2 a0 65 b7 a4
b1 11 68 4f 1d 17 b5

```

oooooo

/

随后可以使用测试加密后的代码

```
SQL> @f_get_rowid.plb
```

```
Function created.
```

```
SQL> select get_rowid('AAABiPAABAAAFRSAAA') from dual;
```

```
GET_ROWID('AAABiPAABAAAFRSAAA')
```

```
-----
Object# is      :6287
```

```
Relative_fno is :1
```

```
Block number is :21586
```

```
Row number is   :0
```

注意，如果在加密过程中出现类似如下错误：

```
kgepop: no error frame. to pop to for error 1801
```

则需要设置正确的环境变量 NLS_LANG，例如在 WIndows 下可以如下设置：

```
C:\oracle\ora92\bin>set NLS_LANG=CHINESE_CHINA.ZHS16GBK
```

通过对代码的加密，可以在一定程度上提高程序的安全性。

3.加密信用卡号

除了加密常规的密码之外，dbms_obfuscation_toolkit 包还可以用于加密其他敏感数据，Oracle 提供了一个范例用于演示加密诸如信用卡号之类的数据内容，以下范例来自 MOS Note 197400.1,供读者参考：

```
SQL> drop table cards;
```

```
Table dropped.
```

```
SQL> create table cards (cust_id number primary key, encrypted_card_id raw(64));
```

```
Table created.
```

以下过程用于插入数据时进行加密：

```
SQL> create or replace procedure insert_card( cust_id IN number,
2                                     plain_card_id IN varchar2,
3                                     password in raw) as
4  random_seed raw(80);
5  random_IV  raw(8);
6  pseudo_string varchar2(100);
7  plain_card_raw raw(256);
8  encrypted_card_raw raw(256);
9  begin
10  -- generate a random IV, it does not need to be secret, only random
11  pseudo_string := to_char(sysdate,'yyyymmddssmi');
12  pseudo_string := rpad(pseudo_string,80,pseudo_string);
13  random_seed := utl_raw.cast_to_raw(pseudo_string);
14  dbms_obfuscation_toolkit.desgetkey(seed => random_seed,
15                                     key => random_IV);
16  -- prefix the plain_card_id with the random IV
17  plain_card_raw := random_IV||utl_raw.cast_to_raw(plain_card_id);
18  -- encrypt the plain card id
19  dbms_obfuscation_toolkit.DES3Encrypt(
20                                     input => plain_card_raw,
21                                     key => password,
22                                     encrypted_data => encrypted_card_raw,
23                                     which => 1);
```

```

24  insert into cards values (cust_id,encrypted_card_raw);
25  commit;
26  end;
27  /

```

Procedure created.

以下过程用于访问时的数据解密:

```

SQL> create or replace procedure get_card(cust_id IN number,
2                                     password IN raw,
3                                     plain_card_id OUT varchar2) as
4  plain_card_raw raw(256);
5  encrypted_card_raw raw(256);
6  decrypted_card_id varchar2(24);
7  begin
8  select encrypted_card_id into encrypted_card_raw
9  from cards where cust_id = cust_id;
10  dbms_obfuscation_toolkit.DES3Decrypt(
11                                     input => encrypted_card_raw,
12                                     key => password,
13                                     decrypted_data => plain_card_raw,
14                                     which => 1);
15  decrypted_card_id := utl_raw.cast_to_varchar2(plain_card_raw);
16  -- discard the random IV
17  plain_card_id := substr(decrypted_card_id,9);
18  end;
19  /

```

Procedure created.

以下过程插入一条数据, 通过加密进行存储:

```

SQL> set serveroutput on
SQL> declare
2  password raw(256);
3  begin
4  -- this is a sample password, do not use this password in real applications

```

```

5    password := hextoraw('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');
6    -- you must supply a string that is a multiple of 8 bytes
7    insert_card(1,'1234567890123456',password);
8  end;
9  /

```

PL/SQL procedure successfully completed.

查询数据表，可以看到卡号信息以加密形式被存储下来：

```
SQL> col encrypted_card_id for a60
```

```
SQL> select * from cards;
```

```
      CUST_ID  ENCRYPTED_CARD_ID
```

```
-----
      1  9A0C234E7207F069CE33A34EF2A333755C88A0370713F00C
```

以下是读取时的逆向过程：

```
SQL> declare
```

```

2    password raw(256);
3    plain_card_id varchar2(16);
4  begin
5    password := hextoraw('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');
6    get_card(1,password,plain_card_id);
7    dbms_output.put_line(plain_card_id);
8  end;
9  /

```

```
1234567890123456
```

PL/SQL procedure successfully completed.

数据安全性与防范没有止境，一步一步从无到有，是安全防范的重要准则。

4.DBMS_CRYPTO 加密

从 Oracle 10g 开始，DBMS_CRYPTO 程序包被引入到数据库中，用于增强和替代 DBMS_OBFUSCATION_TOOLKIT，DBMS_CRYPTO 支持多种加密算法，包括 Advanced Encryption Standard(AES) 对称密码新标准，

该标准由 NIST（the National Institute of Standards and Technology）美国国家标准技术研究所提供，替代 DES 数据加密标准。

DBMS_CRYPTO 中的过程可以生成私有密钥，也可以自己指定并存储密钥。可以对 oracle 的通用数据类型进行加密和解密，包括 RAW、LOB 类型，常见的图像、声音、文档文件等都可以被加密，BLOB、CLOB 类型也同样被支持。此外，DBMS_CRYPTO 还提供了全球化支持，可以在不同字符集的数据库之间进行数据加密与解密工作。

DBMS_CRYPTO 支持以下加密算法：

- 1. Data Encryption Standard (DES), Triple DES (3DES, 2-key and 3-key)
- 2. Advanced Encryption Standard (AES)
- 3. MD5, MD4, and SHA-1 cryptographic hashes
- 4. MD5 and SHA-1 Message Authentication Code (MAC)

DBMS_CRYPTO 也提供分组密码编辑器功能，并且有几种补位选项可供选择，包括 PKCS (Public Key Cryptographic Standard) #5，以及多种块密码链接模式，包括 CBC (Cipher Block Chaining). 该程序包由 \$ORACLE_HOME/rdbms/admin/dbmsobtk.sql 脚本创建,其中包含了 DBMS_CRYPTO 和 dbms_obfuscation_toolkit 的脚本，下表列出了这两个程序包的异同之处，我们可以看到 DBMS_CRYPTO 更加复杂和强大，从 10g 开始，应该使用这个 PACKAGE 用于加密：

Package Feature	DBMS_CRYPTO	DBMS_OBFUSCATION_TOOLKIT
Cryptographic algorithms	DES, 3DES, AES, RC4, 3DES_2KEY	DES, 3DES
Padding forms	PKCS5, zeroes	none supported
Block cipher chaining modes	CBC, CFB, ECB, OFB	CBC
Cryptographic hash algorithms	MD5, SHA-1, MD4	MD5
Keyed hash (MAC) algorithms	HMAC_MD5, HMAC_SH1	none supported
Cryptographic pseudo-random number generator	RAW, NUMBER, BINARY_INTEGER	RAW, VARCHAR2
Database types	RAW, CLOB, BLOB	RAW, VARCHAR2

DBMS_CRYPTO 不直接支持 VARCHAR2 数据类型，必须将其转换成 RAW 的数据类型进行处理,这也是对 dbms_obfuscation_toolkit 程序包的改进之处。

以下是 DBMS_CRYPTO 包的注释说明，其中明确提到，在加密之前，数据应当被转换为统一的 AL32UTF8 字符集,以避免可能因为字符集问题带来的迁移损失。

```
CREATE OR REPLACE PACKAGE DBMS_CRYPT0 AS
```

```
-----
--
-- PACKAGE NOTES
--
-- DBMS_CRYPT0 contains basic cryptographic functions and
-- procedures. To use correctly and securely, a general level of
-- security expertise is assumed.
--
-- VARCHAR2 datatype is not supported. Cryptographic operations
-- on this type should be prefaced with conversions to a uniform
-- character set (AL32UTF8) and conversion to RAW type.
--
-- Prior to encryption, hashing or keyed hashing, CLOB datatype is
-- converted to AL32UTF8. This allows cryptographic data to be
-- transferred and understood between databases with different
-- character sets, across character set changes and between
-- separate processes (for example, Java programs).
--
-----
```

以下是使用该过程实现类似信用卡卡号加密和解密的过程：

```
SQL> DECLARE
```

```

 2  l_credit_card_no VARCHAR2(19) := '1234567890123456';
 3  l_card_no_raw RAW(128) := utl_raw.cast_to_raw(l_credit_card_no);
 4
 5                                     l_raw_key                                RAW(128)      :=
utl_raw.cast_to_raw('0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF');
 5
 6  v_encrypted_raw RAW(2048);
 7  v_decrypted_raw RAW(2048);
 8  BEGIN
 9      dbms_output.put_line('Credit Card Number : ' || l_credit_card_no);
10
11      v_encrypted_raw := dbms_crypto.encrypt(l_card_no_raw,
                                           dbms_crypto.des_cbc_pkcs5, l_raw_key);
12
13      dbms_output.put_line('Encrypted : '
                             || RAWTOHEX(utl_raw.cast_to_raw(v_encrypted_raw)));
14
15      v_decrypted_raw := dbms_crypto.decrypt(src => v_encrypted_raw,
```

```

                                typ => dbms_crypto.des_cbc_pkcs5, key => l_raw_key);
16
17   dbms_output.put_line('Decrypted : '
                                || utl_raw.cast_to_varchar2(v_decrypted_raw));
18 END;
19 /
Credit Card Number : 1234567890123456
Encrypted :
43423232453043343941373345304530434146354433313832454533463738373935433032334342
46313930313135333
Decrypted : 1234567890123456

```

对于字符型数据的加密和解密可以通过 UTL_I18N.STRING_TO_RAW 和 UTL_I18N.RAW_TO_CHAR 来完成。以下是包含 UTL_I18N 转换的范例：

```

SQL> DECLARE
2   input_string      VARCHAR2 (200) := 'Oracle Database 12c';
3   output_string VARCHAR2 (200);
4   encrypted_raw RAW (2000);          -- stores encrypted binary text
5   decrypted_raw RAW (2000);          -- stores decrypted binary text
6   num_key_bytes NUMBER := 256/8; -- key length 256 bits (32 bytes)
7   key_bytes_raw RAW (32);           -- stores 256-bit encryption key
8   encryption_type PLS_INTEGER := -- total encryption type
9   DBMS_CRYPTO.ENCRYPT_AES256 + DBMS_CRYPTO.CHAIN_CBC + DBMS_CRYPTO.PAD_PKCS5;
10 BEGIN
11   DBMS_OUTPUT.PUT_LINE ( 'Original string: ' || input_string);
12   key_bytes_raw := DBMS_CRYPTO.RANDOMBYTES (num_key_bytes);
13   encrypted_raw := DBMS_CRYPTO.ENCRYPT (
14       src => UTL_I18N.STRING_TO_RAW (input_string, 'AL32UTF8'),
15       typ => encryption_type, KEY => key_bytes_raw );
16   DBMS_OUTPUT.PUT_LINE ('Encrypted string: ' || encrypted_raw);
17   decrypted_raw := DBMS_CRYPTO.DECRYPT ( src => encrypted_raw,
18       typ => encryption_type, KEY => key_bytes_raw );
19   output_string := UTL_I18N.RAW_TO_CHAR (decrypted_raw, 'AL32UTF8');
20   DBMS_OUTPUT.PUT_LINE ('Decrypted string: ' || output_string);

```

```
21  END;
```

```
22  /
```

Original string: Oracle Database 12c

Encrypted string: 6071580A787F1BA155CA93993175EB81C1A7D4344577DC145449F09959B5EB85

Decrypted string: Oracle Database 12c

PL/SQL procedure successfully completed.

DBMS_CRYPTO 的功能非常强大，对于安全增强具有很大帮助，值得深入研究并应用到安全实践之中。

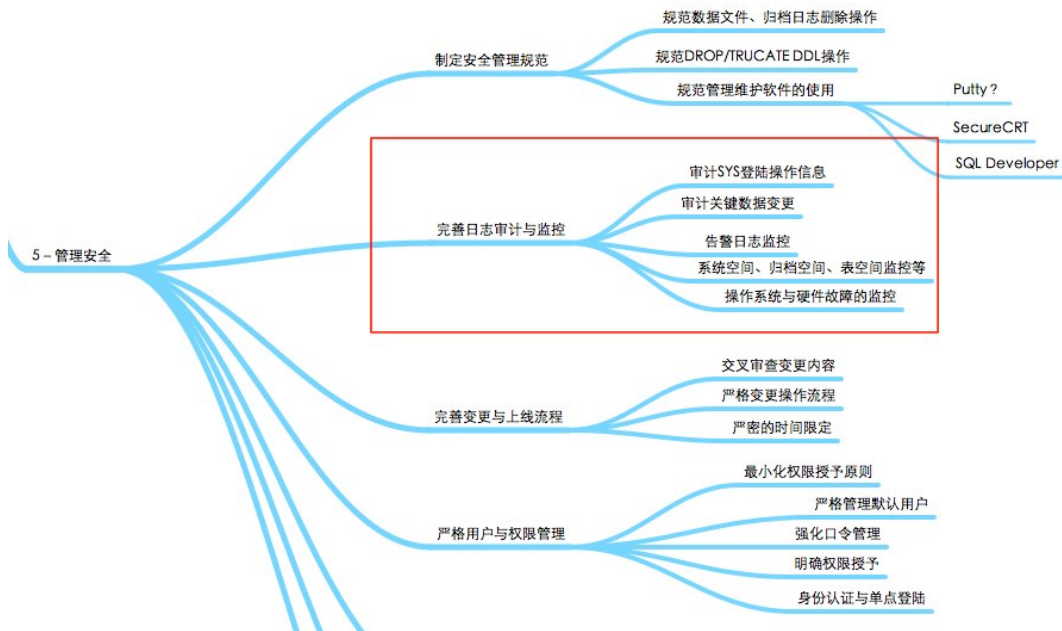
明察秋毫，见微知著

明足以察秋毫之末，而不见舆薪，则王许之乎？

——《孟子·梁惠王上》

圣人见微以知萌，见端以知末，故见象箸而怖，知天下不足也。

——《韩非子·说林上》



在数据库运维和技术工作中，越来越要求我们具有细致、耐心、洞察力，细致能够规避错误，耐心能够按部就班、善始善终，而洞察力则能够帮助我们穿过迷雾，察知本源，规避问题。这部分属于管理安全的范畴，通过严密的监控与分析，我们才能够察知变化，防微杜渐，预防性的消灭安全隐患，保障数据安全。对于性能的跟踪和评估，与此道理相同。

篇首图是管理安全部分的局部导图，本章的部分案例，完全能够通过日志监控防范和规避：

我以前曾经提到，对于一个技术候选人,我希望他,勤奋,严谨,具有钻研精神及独立思考能力,技术其实往往并不是我最关心的内容,因为具备了前面的素质之后,经过 1~2 年的锻炼,一个人绝对不会知道的太少。而这些基本素质，并非能够轻易获得。

一次碰撞引发的灾难

ASM 保护式文件离线引发故障

作为数据库维护人员，少不了和机房、服务器打交道，也许很多 DBA 都遭遇过和技术无关的机房意外，我记得刚入行时，有一次在联通机房调试设备，有个同事进来一脚将服务器的电源踢掉，结果文件系统的检查和恢复就花费了数个小时。

与电源有关的灾难在后来遇到了很多很多。

灾难描述

某客户在夜间维护之后，发生了点小意外：

1. 机房维护工程师不小心将光纤交换机碰掉电
2. 数据库发现网络中断出现故障
3. 交换机加电恢复，但是发现数据库 ASM 磁盘组无法在线
4. 数据库服务遭受影响
5. 灾难形成

一次小小的、不经意的碰撞，随之却导致了复杂的数据库故障，可见严谨、谨慎时刻不可或缺。

案例警示

这个案例给我们的警示有：

1. 数据中心维护应当遵循安全守则

如果维护涉及数据环境的硬件设备，在可能的情况下，最好先关闭数据库；如果数据库不能关闭，则应当详细论证，维护是否涉及到数据环境的核心设备。存储、网络设备、主机等异常断电都可能引发数据库灾难。

在维护是尽量避免扎堆作业，在狭窄的环境中穿梭，想不出问题是不容易的。

2. 重要设备维护应当组织团队论证

数据中心的重要设备、设施维护，应当召集各方面相关专家团队进行论证，有时候硬件的维护不仅仅是系统管理员或者网管的事情，上层软件方面更值得关注，硬件的故障和异常可以通过更换来解决，而软件层面的异常则往往会更加复杂，尤其是数据库对于一致性要求极高。

在维护工作时，绝对不能“头痛医头，脚痛医脚”，仅仅想着自扫门前雪是要不得的，系统运维是一项系统工程。

3. 维护操作中要需要实现全局监控

这是我们反复强调的常识之一，在系统运维的过程中，要随时保持对于各级日志的监控，如果日志中出现任何错误，都要获得及时的响应和处理，不要将错误隐藏拖延到维护之后。不要以为某个环节出现的问题会仅是局部的，系统的关联性使得一个问题会在全局放大。

4. 遵循或参照 ITIL 标准与流程实现安全运维

ITIL 即 IT 基础架构库(Information Technology Infrastructure Library)，最初由英国政府部门制订，现由英国商务部负责管理，主要适用于 IT 服务管理（ITSM）。

随着后期的发展和演进，ITIL 得到了广泛的应用，其主要模块包括业务管理、服务管理、ICT 基础架构管理、IT 服务管理规划与实施、应用管理和安全管理，其中服务管理是其最核心的模块。

ITIL 为企业的 IT 服务管理实践提供了一个客观、严谨、量化的标准和规范，其核心思想值得我们在运维中予以遵循和借鉴，只有用流程规范和约束的变更才能减少异常，保障安全。

总之，遵循一定的规则，通过规则约束来降低疏忽和失误，是数据库运维过程中必须遵守的原则。

技术回放

在告警日志中，我们注意到在凌晨 1: 29 数据库报出心跳网络中断的错误，数据库崩溃，然后在凌晨 4: 32 启动数据库：

```
Thu Jun 25 01:20:39 2009
Thread 1 advanced to log sequence 134760
Current log# 5 seq# 134760 mem# 0: +DG_DATA_01/billbj/onlineolog/group_5
```

```

Thu Jun 25 01:29:36 2009
ospid 25202: network interface with IP address 10.1.1.1 is DOWN
Thu Jun 25 04:32:51 2009
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Interface type 1 lan1 10.1.1.0 configured from OCR for use as a cluster interconnect
Interface type 1 lan0 192.168.0.0 configured from OCR for use as a public interface

```

由于网络中一台交换机断电，部分链路受到影响，启动数据库时出现了如下错误：

```

Thu Jun 25 05:00:11 2009
Errors in file /u01/app/oracle/admin/billbj/udump/billbj1_ora_8184.trc:
ORA-15062: ASM disk is globally closed
ORA-15025: could not open disk '/dev/rdisk/c12t0d2'
ORA-27041: unable to open file
HPUX-ia64 Error: 6: No such device or address
Additional information: 3
Thu Jun 25 05:00:29 2009
Errors in file /u01/app/oracle/admin/billbj/udump/billbj1_ora_8761.trc:
ORA-15062: ASM 磁盘已全局关闭
ORA-15025: 无法打开磁盘 '/dev/rdisk/c12t0d2'
ORA-27041: 无法打开文件
HPUX-ia64 Error: 6: No such device or address
Additional information: 3
Thu Jun 25 05:00:29 2009
Errors in file /u01/app/oracle/admin/billbj/udump/billbj1_ora_8759.trc:
ORA-15062: ASM disk is globally closed
ORA-15025: could not open disk '/dev/rdisk/c12t0d2'
ORA-27041: unable to open file
HPUX-ia64 Error: 6: No such device or address
Additional information: 3

```

注意在这个提示中，有一个重要提示：ORA-15062: ASM 磁盘已全局关闭。也就是说，由于磁盘无法访问，ASM 将磁盘在全局关闭，ASM 磁盘组也不可用。

用户将数据库打开，当数据库试图对无法访问的文件进行读写时，就遇到如下错误，磁盘组被强制卸载：

```
Thu Jun 25 06:08:18 2009
Errors in file /u01/app/oracle/admin/billbj/bdump/billbj1_j000_28682.trc:
ORA-12012: 自动执行作业 373272 出错
ORA-12008: 实体化视图的刷新路径中存在错误
ORA-01115: 从文件 125 读取块时出现 IO 错误 (块 # 261773)
ORA-01110: 数据文件 125: '+DG_DATA_03/billbj/datafile/phoenix_default.304.679315039'
ORA-15078: 已强制卸载 ASM 磁盘组
ORA-06512: 在 "SYS.DBMS_SNAPSHOT", line 2254
ORA-06512: 在 "SYS.DBMS_SNAPSHOT", line 2460
ORA-06512: 在 "SYS.DBMS_IREFRESH", line 683
ORA-06512: 在 "SYS.DBMS_REFRESH", line 195
ORA-06512: 在 line 1
```

然后 Oracle 将该磁盘组（DG_DATA_03）中的所有文件 Offline 离线处理，这实际上是实现了数据保护：

```
Thu Jun 25 06:10:41 2009
KCF: write/open error block=0xce0db online=1
      file=148 +DG_DATA_03/billbj/datafile/tbs_band_table_20.256.654268217
      error=15081 txt: ''
Automatic datafile offline due to write error on
file 148: +DG_DATA_03/billbj/datafile/tbs_band_table_20.256.654268217
KCF: write/open error block=0x72b online=1
      file=21 +DG_DATA_03/billbj/datafile/tbs_default_idx_20.265.654273237
      error=15078 txt: ''
Automatic datafile offline due to write error on
file 21: +DG_DATA_03/billbj/datafile/tbs_default_idx_20.265.654273237
KCF: write/open error block=0x4c6e9 online=1
      file=50 +DG_DATA_03/billbj/datafile/tbs_band_daillytable_20.270.656595577
      error=15078 txt: ''
Automatic datafile offline due to write error on
file 50: +DG_DATA_03/billbj/datafile/tbs_band_daillytable_20.270.656595577
KCF: write/open error block=0x42ab9 online=1
      file=57 +DG_DATA_03/billbj/datafile/tbs_band_daillytable_20.273.656599591
```

```

error=15078 txt: ''
Automatic datafile offline due to write error on
file 57: +DG_DATA_03/billbj/datafile/tbs_band_daillytable_20.273.656599591
KCF: write/open error block=0x8e20b online=1
file=68 +DG_DATA_03/billbj/datafile/tbs_band_table_idx_20.281.659614825
error=15078 txt: ''

```

此时检查 ASM 的告警日志，也可以发现 ASM 实例无法加载磁盘组（DG_DATA_03）的错误：

```

Thu Jun 25 05:10:49 2009
NOTE: recovering COD for group 1/0x6808040 (DG_DATA_01)
SUCCESS: completed COD recovery for group 1/0x6808040 (DG_DATA_01)
NOTE: recovering COD for group 2/0x6908041 (DG_DATA_02)
SUCCESS: completed COD recovery for group 2/0x6908041 (DG_DATA_02)
Thu Jun 25 05:12:59 2009
Errors in file /u01/app/oracle/admin/+ASM/bdump/+asm1_gmon_21761.trc:
ORA-27091: unable to queue I/O
ORA-27072: File I/O error
HPUX-ia64 Error: 6: No such device or address
Additional information: 4
Additional information: 2044
Additional information: -1
Thu Jun 25 05:13:29 2009
WARNING: cache failed to read fn=3 blk=0 from disk(s): 1
ORA-15062: ASM disk is globally closed
NOTE: cache initiating offline of disk 1 group 3
Thu Jun 25 05:22:57 2009
ERROR: no PST quorum in group 3: required 1, found 0
Thu Jun 25 05:22:57 2009
Errors in file /u01/app/oracle/admin/+ASM/bdump/+asm1_n000_28730.trc:
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DG_DATA_03"
Thu Jun 25 05:22:57 2009
ERROR: async update- could not update PST (grp 3)
Thu Jun 25 05:22:57 2009
Errors in file /u01/app/oracle/admin/+ASM/bdump/+asm1_n000_28730.trc:

```

```
ORA-15063: ASM discovered an insufficient number of disks for diskgroup "DG_DATA_03"
```

当用户解决了交换机问题之后，试图恢复数据库运行时，遇到了另外一个问题，从数据库角度看来，始终有一个磁盘组处于 Mounted 的状态，数据库无法连接，也就无法访问其中的数据：

```
SQL> select name,state from v$asm_diskgroup_stat;
```

NAME	STATE
-----	-----
DG_DATA_01	CONNECTED
DG_DATA_02	CONNECTED
DG_DATA_03	MOUNTED

并且从告警日志来看，ASM 磁盘组在挂载后会很快被自动卸载：

```
Thu Jun 25 17:45:11 2009
SUCCESS: diskgroup DG_DATA_03 was mounted
SUCCESS: diskgroup DG_DATA_03 was dismounted
SUCCESS: diskgroup DG_DATA_03 was mounted
SUCCESS: diskgroup DG_DATA_03 was dismounted
SUCCESS: diskgroup DG_DATA_03 was mounted
SUCCESS: diskgroup DG_DATA_03 was dismounted
```

如何修正磁盘组的状态成为了一个难题，如果该磁盘组无法连接，则数据库就无法访问其中的数据。

恢复过程

我们尝试直接拷贝复制磁盘组中的文件，发现可以成功，而在文件备份过程中，磁盘组的状态在数据库中转为正常的 CONNECTED 模式：

```
oracle@ccnbjdc1[billbj1]:/u01/app/oracle/admin/billbj/bdump$ rman target /
Recovery Manager: Release 10.2.0.3.0 - Production on Thu Jun 25 17:45:00 2009
Copyright (c) 1982, 2005, Oracle. All rights reserved.
connected to target database: BILLBJ (DBID=2424179062)
RMAN> copy datafile '+DG_DATA_03/billbj/datafile/tbs_default_20.264.654269073' to '/backup/a.dbf';
Starting backup at 25-JUN-09
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
```

```

channel ORA_DISK_1: sid=1152 instance=billbj1 devtype=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: sid=1155 instance=billbj1 devtype=DISK
allocated channel: ORA_DISK_3
channel ORA_DISK_3: sid=1153 instance=billbj1 devtype=DISK
allocated channel: ORA_DISK_4
channel ORA_DISK_4: sid=1150 instance=billbj1 devtype=DISK
channel ORA_DISK_1: starting datafile copy
input datafile fno=00161
name=+DG_DATA_03/billbj/datafile/tbs_default_20.264.654269073
output filename=/backup/a.dbf tag=TAG20090625T174511 recid=2 stamp=690486558
channel ORA_DISK_1: datafile copy complete, elapsed time: 00:04:09
Finished backup at 25-JUN-09

```

```

Starting Control File and SPFILE Autobackup at 25-JUN-09
piece handle=/backup/ctlbak/c-2424179062-20090625-01 comment=NONE
Finished Control File and SPFILE Autobackup at 25-JUN-09

```

至此，结合前面分析的文件 Offline 状态，我们得出以下结论：

当磁盘组中的所有文件 Offline，则 Oracle 不访问该 DG 中的磁盘，该磁盘就保持了 MOUNTED 状态，数据库无需连接磁盘组；我们只要尝试访问该磁盘中的文件，该磁盘组就会显示为数据库连接的 CONNECTED 状态。

接下来通过 Recover 那些被 Offline 的文件，再执行 Online 操作，就将数据库恢复到了正常状态。

```

Thu Jun 25 17:54:22 2009
ALTER DATABASE RECOVER datafile 151
Thu Jun 25 17:54:22 2009
Media Recovery Start
parallel recovery started with 3 processes
ORA-279 signalled during: ALTER DATABASE RECOVER datafile 151 ...
Thu Jun 25 17:54:24 2009
ALTER DATABASE RECOVER CONTINUE DEFAULT
Thu Jun 25 17:54:24 2009
Media Recovery Log +DG_DATA_02/billbj/1_134766_634298105.dbf

```

```

ORA-279 signalled during: ALTER DATABASE RECOVER    CONTINUE DEFAULT    ...
Thu Jun 25 17:54:24 2009
ALTER DATABASE RECOVER    CONTINUE DEFAULT
Thu Jun 25 17:54:24 2009
Media Recovery Log +DG_DATA_02/billbj/2_90114_634298105.dbf
ORA-279 signalled during: ALTER DATABASE RECOVER    CONTINUE DEFAULT    ...
Thu Jun 25 17:54:25 2009
ALTER DATABASE RECOVER    CONTINUE DEFAULT
Thu Jun 25 17:54:25 2009
Media Recovery Log +DG_DATA_02/billbj/2_90115_634298105.dbf
ORA-279 signalled during: ALTER DATABASE RECOVER    CONTINUE DEFAULT    ...
Thu Jun 25 17:54:27 2009
ALTER DATABASE RECOVER    CONTINUE DEFAULT
Thu Jun 25 17:54:27 2009
Media Recovery Log +DG_DATA_02/billbj/1_134767_634298105.dbf
Thu Jun 25 17:54:28 2009
Media Recovery Complete (billbj1)

```

这个数据库是一个 TB 级别的核心计费数据库，非常重要：

```

SQL> select sum(bytes)/1024/1024/1024 from v$datafile;
SUM(BYTES)/1024/1024/1024
-----
1046.01709

```

因为简单的一次维护碰撞，业务遭受了一天的影响。

又一次碰撞引发的灾难

文件离线与归档缺失案例

前面提到的用户属于幸运者，如果用户不能及时发现和解决这个问题，如果不是一个 ASM 磁盘组的整体问题，也许故障就会被掩盖，灾难就会更惨重。

从一个侧面来说，Oracle 的自动数据库文件离线保护机制是有问题的，或者说提示不够明确，数据库应当在启动过程中，予以明确提示用户，部分文件因为保护离线，请用户处理，如果能够在这个环境做出提示，本章的几个案例就都会大大简化。

灾难描述

以下是与上一节完全类似的一个案例，但是错误走得更远，灾难也就更加严重：

1. 集成商为用户扩展存储，增加硬盘
2. 无意中将光线交换机碰断电
3. 部分数据库文件出现读写错误离线
4. 重启数据库后未察觉
5. 多日后发现，执行在线恢复
6. 发现丢失了归档日志，数据文件无法加载
7. 灾难形成

丢失了归档日志，文件离线，使得这个案例变得异常复杂。

案例警示

这个案例给我们的警示有：

1. 墨菲定律总是无处不在

墨菲定律告诉我们，你越是害怕出现问题的地方，就越是会出现问题；哪怕你觉得需要 N 多个条件才能出

现的问题，这 N 多个条件终将满足。

所以不能在任何一个环节掉以轻心，我们举一个管理学中常见的比喻：假定一个故障在可靠性低于 60% 才能发生，我们认为这个概率已经很低；另一个条件，通常我们觉得将一个工作的准确性达到 90% 已经足够优秀；现在的问题时，如果多个环节都做到 90% 会怎样？

假定数据环境涉及到了网络、主机、存储、操作系统和数据库五大环节，如果每个环境都做到了 90 分，那么最后整个系统的可靠性有多少呢？

$$90\% \times 90\% \times 90\% \times 90\% \times 90\% = 59.049\%$$

最终答案是 59.049%，这个指标已经低于了 60%，这个系统的稳定性将会出现问题。所以在我们经手的每一个环境，都应当力争做到 100 分，这样才能为其他环境留下机会，为系统的稳健提供保护。

2. 在维护工作之后进行日志检查

通常维护工作都在深夜来完成，而人在疲劳加班之后，潜意识会想要尽快完成工作，离开现场，这就为工作留下了隐患。

根据我们的经验，在维护时出现的问题，通过日志监控和检查都可以发现。

我们建议，对于数据库环境，在维护期间应当提炼摘取维护期生成的所有日志，确保日志中没有出现错误，或者出现的错误都得到了处理。

这样至少可以避免多数基本故障。

3. 不要过分信赖数据库的自我修复能力

对于类似这样的故障，很多用户认为，Oracle 应当能够通过自我调整来完成故障恢复，但是显然，我们不能对数据库要求太高。在这个案例中，保护性离线之后，Oracle 不会自动进行文件恢复和在线尝试（实际上这种情况是应当可以自动修复的），如果用户疏忽，则故障就可能出现。

对于自动保护的文件离线问题，实际上我认为是 Oracle 数据库的 BUG，Oracle 应当能够分辨哪些情况文件是由于保护方式离线的，并且应当在数据库启动之后，给予用户强制性提示，要求用户进行判断和处理，如果这样，本章所描述的案例就不会触发如此复杂严重的事故。

技术和管理相结合，才能确保数据库的安全。

技术回放

在检查用户数据库告警日志时，首先发现客户数据库早就存在 ORA-600 错误：

```
Fri Mar 11 01:58:11 2011
Errors in file /opt/oracle/admin/orcl/udump/orcl_ora_233946.trc:
ORA-00600: internal error code, arguments: [25012], [1], [2], [], [], [], [], []
ARC0: Completed archiving log 5 thread 1 sequence 81253
ARC0: Evaluating archive log 7 thread 2 sequence 27079
ARC0: Beginning to archive log 7 thread 2 sequence 27079
Creating archive destination LOG_ARCHIVE_DEST_1: '/arch/2_27079.dbf'
Fri Mar 11 01:58:11 2011
Errors in file /opt/oracle/admin/orcl/udump/orcl_ora_233946.trc:
ORA-00600: internal error code, arguments: [25012], [1], [2], [], [], [], [], []
```

数据库的参数文件中，设置了大量隐含参数进行错误屏蔽：

```
db_file_multiblock_read_count= 16
fast_start_mttr_target      = 300
rollback_segments          = system
_corrupted_rollback_segments= _SYSSMU1$, _SYSSMU2$, _SYSSMU3$, _SYSSMU4$, _SYSSMU5$,
_SYSSMU6$, _SYSSMU7$, _SYSSMU8$, _SYSSMU9$, _SYSSMU10$, _SYSSMU11$, _SYSSMU12$,
_SYSSMU13$, _SYSSMU14$, _SYSSMU15$, _SYSSMU16$, _SYSSMU17$, _SYSSMU18$, _SYSSMU19$,
_SYSSMU20$
undo_management             = MANUAL
undo_tablespace             = UNDOTBS4
undo_retention              = 10800
remote_login_passwordfile= EXCLUSIVE
```

这其中的部分参数设置存在问题，另外，一个生产数据库，是不应该在这些参数的保护下运行的，如果错误解决不能排除所有异常，那么数据库就应当重建。

在客户交换机问题出现时，数据库出现如下错误，文件无法读写：

```
Sun Jul 24 10:09:24 2011
Errors in file /opt/oracle/admin/orcl/udump/orcl_ora_618684.trc:
ORA-01115: IO error reading block from file 72 (block # 500191)
```

```

ORA-27063: skgfospo: number of bytes read/written is incorrect
IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 32768
ORA-01115: IO error reading block from file 72 (block # 500191)
ORA-27063: skgfospo: number of bytes read/written is incorrect
IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 32768
ORA-01115: IO error reading block from file 72 (block # 500191)
ORA-27063: skgfospo: number of bytes read/written is incorrect
IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 32768

```

接下来数据库保护性的将发生读写错误的数据文件离线：

```

Sun Jul 24 10:31:56 2011
KCF: write/open error block=0x1f1d71 online=1
      file=36 /dev/ro_nlv_img_08
      error=27063 txt: 'IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 8192'
Sun Jul 24 10:31:56 2011
KCF: write/open error block=0xa2b online=1
      file=82 /dev/ro_dt_vio_index_
      error=27063 txt: 'IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 8192'
Sun Jul 24 10:31:56 2011
Errors in file /opt/oracle/admin/orcl/udump/orcl_ora_278924.trc:
ORA-01115: IO error reading block from file 58 (block # 1020235)
ORA-27063: skgfospo: number of bytes read/written is incorrect
IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1

```

```
Additional information: 8192
Automatic datafile offline due to write error on
file 36: /dev/ro_nlv_img_08
Sun Jul 24 10:31:57 2011
Automatic datafile offline due to write error on
file 82: /dev/ro_dt_vio_index_
```

在修复了交换机问题之后，用户将数据库启动：

```
Sun Jul 24 11:39:22 2011
ALTER DATABASE OPEN
Sun Jul 24 11:39:23 2011
Beginning crash recovery of 1 threads
Sun Jul 24 11:39:23 2011
Started first pass scan
Sun Jul 24 11:39:24 2011
Completed first pass scan
    160936 redo blocks read, 2669 data blocks need recovery
>>数据库执行恢复
Sun Jul 24 11:39:24 2011
Started recovery at
    Thread 1: logseq 120428, block 177143, scn 0.0
Recovery of Online Redo Log: Thread 1 Group 1 Seq 120428 Reading mem 0
    Mem# 0 errs 0: /dev/ro_log1_01
Recovery of Online Redo Log: Thread 1 Group 5 Seq 120429 Reading mem 0
    Mem# 0 errs 0: /dev/ro_log1_03
Sun Jul 24 11:39:28 2011
Completed redo application
>>数据库完成恢复
Sun Jul 24 11:39:30 2011
Ended recovery at
    Thread 1: logseq 120429, block 133288, scn 2868.1571390680
    2669 data blocks read, 2669 data blocks written, 160936 redo blocks read
Crash recovery completed successfully
Sun Jul 24 11:39:31 2011
```

```

LGWR: Primary database is in CLUSTER CONSISTENT mode
Thread 1 advanced to log sequence 120430
Thread 1 opened at log sequence 120430
  Current log# 2 seq# 120430 mem# 0: /dev/ro_log1_02
Successful open of redo thread 1.
Sun Jul 24 11:39:31 2011
SMON: enabling cache recovery
>>数据库开始工作归档
Sun Jul 24 11:39:31 2011
ARC0: Evaluating archive   log 5 thread 1 sequence 120429
ARC0: Beginning to archive log 5 thread 1 sequence 120429
Creating archive destination LOG_ARCHIVE_DEST_1: '/arch/1_120429.dbf'
ARC0: Completed archiving   log 5 thread 1 sequence 120429

```

注意，此时用户认为数据库已经恢复了正常，能够提供服务，开始切换归档日志。但是没有人注意到，有几个数据库文件已经 Offline 离线。

等到用户注意到这个问题，尝试去加载这些文件时，发现归档日志丢失了，没有了归档日志，这些数据文件无法被 Online：

```

Mon Jul 25 14:25:12 2011
ALTER DATABASE RECOVER   datafile '/dev/ro_dt_vio_dat_02'
Mon Jul 25 14:25:12 2011
Media Recovery Datafile: '/dev/ro_dt_vio_dat_02'
Media Recovery Start
Starting datafile 94 recovery in thread 1 sequence 120428
Datafile 94: '/dev/ro_dt_vio_dat_02'
Media Recovery Log
ORA-279 signalled during: ALTER DATABASE RECOVER   datafile '/dev/ro_dt_vio_d...
Mon Jul 25 14:25:15 2011
ALTER DATABASE RECOVER   CONTINUE DEFAULT
Media Recovery Log /arch/1_120428.dbf
Errors with log /arch/1_120428.dbf.
ORA-308 signalled during: ALTER DATABASE RECOVER   CONTINUE DEFAULT   ...
Mon Jul 25 14:25:15 2011

```

```
ALTER DATABASE RECOVER    CONTINUE DEFAULT
Media Recovery Log /arch/1_120428.dbf
Errors with log /arch/1_120428.dbf.
ORA-308 signalled during: ALTER DATABASE RECOVER    CONTINUE DEFAULT ...
Mon Jul 25 14:25:15 2011
ALTER DATABASE RECOVER CANCEL
Media Recovery Cancelled
Completed: ALTER DATABASE RECOVER CANCEL
```

这里的错误 ORA-279 在前台出现的错误提示就是归档日志不可用：

```
[oracle@hpserver2 ~]$ oerr ora 279
00279, 00000, "change %s generated at %s needed for thread %s"
// *Cause: The requested log is required to proceed with recovery.
// *Action: Please supply the requested log with "ALTER DATABASE RECOVER
//          LOGFILE <file_name>" or cancel recovery with "ALTER DATABASE
//          RECOVER CANCEL".
```

用户数据库大小近 3TB，包含了长期以来的数据积累：

```
select sum(bytes)/1024/1024/1024 GB from v$datafile;
          GB
```

```
-----
2816.17435
```

离线的文件主要有以下系列：

```
select file#,name,bytes/1024/1024 MB,status from v$datafile where status like 'REC%';
```

FILE#	NAME	MB	STATUS
36	/dev/ro_nlv_img_08	32767.9922	RECOVER
82	/dev/ro_dt_vio_index_	4000	RECOVER
94	/dev/ro_dt_vio_dat_02	10000	RECOVER
95	/dev/ro_dt_vio_dat_03	10000	RECOVER
96	/dev/ro_dt_vio_dat_04	10000	RECOVER

对于丢失了归档日志文件的情况，正常情况下，Oracle 不允许跳过归档加载文件，因为丢失归档日志意味着数据库的一致性被破坏，应当通过备份来恢复数据。

但是如果没有备份，我们就只能通过特殊的手段来进行恢复尝试，这种尝试仅在迫不得已的情况下使用，并且应当在之后重建数据库。

恢复过程

以下是本书提供的一个恢复测试说明。

1.BBED 修改文件头跳过归档日志

首先创建一个包含两个数据文件的表空间 USERS：

```
SQL> alter database datafile 'C:\ORACLE\ORADATA\ORA9\USERS01.DBF' resize 2M;
Database altered.

SQL> select name,bytes/1024/1024 from v$datafile;

NAME                                                    BYTES/1024/1024
-----
C:\ORACLE\ORADATA\ORA9I\SYSTEM01.DBF                    250
C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF                    200
C:\ORACLE\ORADATA\ORA9I\USERS01.DBF                      2

SQL> alter tablespace users add datafile 'C:\ORACLE\ORADATA\ORA9\USERS02.DBF' size 2M;
Tablespace altered.

SQL> alter database datafile 'C:\ORACLE\ORADATA\ORA9\USERS01.DBF' autoextend off;
Database altered.
```

在表空间创建一个数据表，使用完所有的空间：

```
SQL> create table eygle tablespace users as select * from dba_objects;
Table created.

SQL> insert into eygle select * from eygle;
6323 rows created.

SQL> insert into eygle select * from eygle;
insert into eygle select * from eygle
*
ERROR at line 1:
```

ORA-01653: unable to extend table SYS.EYGLE by 128 in tablespace USERS

SQL> select count(*) from eygle;

```
COUNT(*)
-----
12646
```

确保数据库运行在归档模式下，将数据文件离线：

SQL> archive log list;

Database log mode	Archive Mode
Automatic archival	Disabled
Archive destination	c:\oracle\9.2.0\RDBMS
Oldest online log sequence	72
Next log sequence to archive	74
Current log sequence	74

SQL> archive log start;

Statement processed.

SQL> alter database datafile 'C:\ORACLE\ORADATA\ORA9\USERS02.DBF' offline;

Database altered.

切换一些归档，然后删除这些归档日志：

SQL> alter system switch logfile;

System altered.

SQL> alter system switch logfile;

System altered.

SQL> alter system switch logfile;

System altered.

SQL> select file#,status from v\$datafile;

```
FILE# STATUS
-----
1 SYSTEM
2 ONLINE
3 ONLINE
4 RECOVER
```

如果此时尝试 Online 数据文件，会要求进行恢复，如果无法找到需要的归档日志，则恢复无法进行，这就是用户面对的情况：

```
SQL> alter database datafile 4 online;
alter database datafile 4 online
*
ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'
SQL> recover datafile 4;
ORA-00279: change 223897 generated at 01/11/2012 16:19:23 needed for thread 1
ORA-00289: suggestion : C:\ORACLE\9.2.0\RDBMS\ARC00074.001
ORA-00280: change 223897 for thread 1 is in sequence #74
```

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

```
ORA-00308: cannot open archived log 'C:\ORACLE\9.2.0\RDBMS\ARC00074.001'
ORA-27041: unable to open file
OSD-04002: 无法打开文件
O/S-Error: (OS 2) The system cannot find the file specified.
SQL> select file#,name from v$datafile;
```

FILE#	NAME
1	C:\ORACLE\ORADATA\ORA9I\SYSTEM01.DBF
2	C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF
3	C:\ORACLE\ORADATA\ORA9I\USERS01.DBF
4	C:\ORACLE\ORADATA\ORA9I\USERS02.DBF

解决这个问题的一个办法是通过 BBED 来进行数据文件头，跳过缺失的归档日志文件，然后强制挂接数据文件。以下是详细的过程介绍。

BBED 的基本配置

首先编辑好 BBED 需要的参数文件，一个用于提供文件列表，一个用于设定基本参数：

```
E:\>type data.txt
      1 C:\ORACLE\ORADATA\ORA9I\SYSTEM01.DBF
      2 C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF
      3 C:\ORACLE\ORADATA\ORA9I\USERS01.DBF
      4 C:\ORACLE\ORADATA\ORA9I\USERS02.DBF

E:\>type par.txt
listfile=data.txt
mode=edit
blocksize=8192
```

使用如下命令启动 BBED 程序：

```
E:\>bbed parfile=par.txt
```

最简单的，由于文件 3 和 4 属于同一个表空间，3 号文件一切完好，我们可以将 3 号文件的头块覆盖到 4 号文件的文件头上。

BBED COPY 进行块复制恢复

BBED 的 COPY 命令语法如下：

```
COPY [ DBA | FILE | FILENAME | BLOCK ] TO [ DBA | FILE | FILENAME | BLOCK ]
```

以下命令将文件 3 的头块拷贝复制到文件 4 的头块上：

```
BBED> copy file 3 block 1 to file 4 block 1;
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1          Offsets: 0 to 7          Dba:0x01000001
-----
0b020000 0100c000

<32 bytes per line>
BBED> set file 4 block 1;
```

```

FILE#          4
BLOCK#         1

BBED> dump
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1          Offsets:  0 to    7          DbA:0x01000001
-----
0b020000 0100c000

```

<32 bytes per line>

数据文件头的信息可以通过 `map` 命令展示出来，以下输出显示数据文件头主要的数据结构为 `kcvfh`，共占用 360 字节存储，右侧显示的是具体内容的偏移量：

```

BBED> map /v
File: (0)
Block: 1                      DbA:0x00000000
-----

Data File Header

struct kcvfh, 360 bytes          @0
  struct kcvfhbfh, 20 bytes      @0
  struct kcvfhhdr, 76 bytes      @20
  ub4 kcvfhbdb                   @96
  struct kcvfhcrs, 8 bytes       @100
  ub4 kcvfhcrt                   @108
  ub4 kcvfhrlc                   @112
  struct kcvfhrls, 8 bytes       @116
  ub4 kcvfhbti                   @124
  struct kcvfhbsc, 8 bytes       @128
  ub2 kcvfhbth                   @136
  ub2 kcvfhsta                   @138
  struct kcvfhckp, 36 bytes      @140
  ub4 kcvfhcpc                   @176
  ub4 kcvfhrts                   @180
  ub4 kcvfhccc                   @184

```

```

struct kcvfhbcp, 36 bytes      @188
ub4 kcvfhhbz                  @224
struct kcvfhxcd, 16 bytes     @228
word kcvfhstn                 @244
ub2 kcvfh1n                   @248
text kcvfh1nm[30]             @250
ub4 kcvfh1rn                   @280
struct kcvfh1rfs, 8 bytes     @284
ub4 kcvfh1rft                  @292
struct kcvfh1afs, 8 bytes     @296
ub4 kcvfh1bbc                  @304
ub4 kcvfh1ncb                  @308
ub4 kcvfh1mcb                  @312
ub4 kcvfh1lcb                  @316
ub4 kcvfh1bcs                  @320
ub2 kcvfh1ofb                  @324
ub2 kcvfh1nfb                  @326
ub4 kcvfh1prc                  @328
struct kcvfh1prs, 8 bytes     @332
struct kcvfh1prfs, 8 bytes    @340
ub4 kcvfh1trt                  @356

```

```

ub4 tailchk                    @8188

```

使用 print 命令可以具体打印出相关变量的信息:

BBED> p kcvfh

```

struct kcvfh, 360 bytes      @0
  struct kcvfhbfh, 20 bytes   @0
    ub1 type_kcbh             @0      0x0b
    ub1 frmt_kcbh             @1      0x02
    ub1 spare1_kcbh           @2      0x00
    ub1 spare2_kcbh           @3      0x00
    ub4 rdba_kcbh              @4      0x01000001
    ub4 bas_kcbh               @8      0x00000000

```

ub2 wrp_kcbh	@12	0x0000
ub1 seq_kcbh	@14	0x01
ub1 flg_kcbh	@15	0x04 (KCBHFCKV)
ub2 chkval_kcbh	@16	0x31d1
ub2 spare3_kcbh	@18	0x0000

拷贝之后还有几个内容需要修改，主要是文件号相关的信息。偏移量 4 记录的是 RDBA 信息，其中包含文件号信息，此处需要根据情况由 3 改为 4：

```
BBED> set offset 4
      OFFSET          4

BBED> dump
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1              Offsets:  4 to  11          DbA:0x01000001
-----
0100c000 00000000

<32 bytes per line>
BBED> modify /x 01000001
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1              Offsets:  4 to  11          DbA:0x01000001
-----
01000001 00000000

<32 bytes per line>
```

此外偏移量 52 处存储的是文件号信息：

ub2 kccfhfno	@52	0x0004
ub2 kccfhtyp	@54	0x0003

也需要同样修改：

```
BBED> set offset 52
      OFFSET          52

BBED> dump
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
```

```

Block: 1                      Offsets: 52 to 59          Dba:0x01000001
-----
03000300 00000000

<32 bytes per line>
BBED> modify /x 04
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1                      Offsets: 52 to 59          Dba:0x01000001
-----
04000300 00000000

<32 bytes per line>

```

偏移量 280 处存储的是相对文件号:

ub4 kcvfhrfn	@280	0x00000004
struct kcvfhrfs, 8 bytes	@284	
ub4 kscnbas	@284	0x00000000
ub2 kscnwrp	@288	0x0000
ub4 kcvfhrft	@292	0x2e086f78

以下需要相应的修改:

```

BBED> set offset 280
      OFFSET          280

BBED> dump
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1                      Offsets: 280 to 287          Dba:0x01000001
-----
03000000 00000000

<32 bytes per line>
BBED> modify /x 04
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1                      Offsets: 280 to 287          Dba:0x01000001
-----

```

```
04000000 00000000
```

```
<32 bytes per line>
```

```
BBED> sum apply
```

数据文件创建时间与 SCN 校验

如果此时尝试恢复数据文件会遇到如下错误，提示文件 4 的创建 SCN 错误：

```
SQL> recover datafile 4;
ORA-00283: recovery session canceled due to errors
ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'
ORA-01122: database file 4 failed verification check
ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'
ORA-01203: wrong incarnation of this file - wrong creation SCN
```

数据文件的 SCN 来自数据字典 file\$视图：

```
SQL> select file#,crscnwrp,crscnbas,to_char(crscnbas,'xxxxxx') scn from file$;
```

FILE#	CRSCNWRP	CRSCNBAS	SCN
1	0	9	9
2	0	4480	1180
3	0	5812	16b4
4	0	222765	3662d

数据文件的创建 SCN 存储与偏移量 100 处：

struct kcvfhcrs, 8 bytes	@100	
ub4 kscnbas	@100	0x0003662d
ub2 kscnwrp	@104	0x0000

根据这个文件的具体信息，修改这个 SCN：

```
BBED> set offset 100
```

```
OFFSET          100
```

```
BBED> dump
```

```
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
```

```

Block: 1                      Offsets: 100 to 107          Dba:0x01000001
-----
b4160000 00000000

<32 bytes per line>
BBED> modify /x 2d660300
File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)
Block: 1                      Offsets: 100 to 107          Dba:0x01000001
-----
2d660300 00000000

<32 bytes per line>
BBED> verify
DBVERIFY - 验证正在启动
FILE =C:\ORACLE\ORADATA\ORA9I\USERS02.DBF
BLOCK = 1

块 1 已毁坏
***
Corrupt block relative dba: 0x01000001 (file 0, block 1)
Bad check value found during verification
Data in bad block -
type: 11 format: 2 rdba: 0x01000001
last change scn: 0x0000.00000000 seq: 0x1 flg: 0x04
consistency value in tail: 0x00000b01
check value in block header: 0xc78e, computed block checksum: 0x715a
spare1: 0x0, spare2: 0x0, spare3: 0x0
***
DBVERIFY - 验证完成

检查的总块数: 1
已处理的总块数 (数据): 0
无法处理的总块数 (数据): 0

```

已处理的总块数 (索引): 0

无法处理的总块数 (索引): 0

空的总块数: 0

标记为损坏的总数块: 1

汇入的块总数: 0

BBED> sum apply

Check value for File 4, Block 1:

current = 0xb6d4, required = 0xb6d4

此时如果尝试恢复数据文件, 则会遇到如下错误, 这是提示创建时间错误, 通过 file\$ 仍然可以获得这个信息, 需要同样修改。

SQL> recover datafile 4;

ORA-00283: recovery session canceled due to errors

ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'

ORA-01122: database file 4 failed verification check

ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'

ORA-01202: wrong incarnation of this file - wrong creation time

创建时间存储于文件头偏移量 108 位置:

ub4 kcvfhcrt	@108	0x2e086327
ub4 kcvfhr1c	@112	0x2cdbbe56

修改数据文件的创建时间信息:

BBED> set offset 108

OFFSET 108

BBED> dump

File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)

Block: 1 Offsets: 108 to 115 DbA:0x01000001

72bedb2c 56bedb2c

<32 bytes per line>

BBED> modify /x 2763082e

File: C:\ORACLE\ORADATA\ORA9I\USERS02.DBF (4)

```

Block: 1                      Offsets: 108 to 115          DbA:0x01000001
-----
2763082e 56bedb2c

<32 bytes per line>
BBED> sum apply
Check value for File 4, Block 1:
current = 0x6952, required = 0x6952

```

旧的控制文件与新的数据文件

此时执行恢复，数据库提示控制文件比数据文件旧，我们需要重建控制文件：

```

SQL> recover datafile 4;
ORA-00283: recovery session canceled due to errors
ORA-01122: database file 4 failed verification check
ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'
ORA-01207: file is more recent than controlfile - old controlfile

```

通过如下步骤重建控制文件（正确的重建控制文件是每个 DBA 应当具备的基本功）：

```

SQL> alter database backup controlfile to trace;
Database altered.
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
ORACLE instance started.

```

```

Total System Global Area 126950956 bytes
Fixed Size                 454188 bytes
Variable Size             92274688 bytes
Database Buffers          33554432 bytes
Redo Buffers               667648 bytes
SQL> CREATE CONTROLFILE REUSE DATABASE "ORA9I" NORESETLOGS  ARCHIVELOG

```

```

2  -- SET STANDBY TO MAXIMIZE PERFORMANCE
3      MAXLOGFILES 5
4      MAXLOGMEMBERS 3
5      MAXDATAFILES 100
6      MAXINSTANCES 1
7      MAXLOGHISTORY 226
8  LOGFILE
9      GROUP 1 'C:\ORACLE\ORADATA\ORA9\REDO01.LOG' SIZE 10M,
10     GROUP 2 'C:\ORACLE\ORADATA\ORA9\REDO02.LOG' SIZE 10M,
11     GROUP 3 'C:\ORACLE\ORADATA\ORA9\REDO03.LOG' SIZE 10M
12  -- STANDBY LOGFILE
13  DATAFILE
14     'C:\ORACLE\ORADATA\ORA9\SYSTEM01.DBF',
15     'C:\ORACLE\ORADATA\ORA9\UNDOTBS01.DBF',
16     'C:\ORACLE\ORADATA\ORA9\USERS01.DBF',
17     'C:\ORACLE\ORADATA\ORA9\USERS02.DBF'
18  CHARACTER SET ZHS16GBK
19  ;

```

Control file created.

此时尝试打开数据库，提示文件 4 需要恢复：

```

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 4 needs media recovery
ORA-01110: data file 4: 'C:\ORACLE\ORADATA\ORA9I\USERS02.DBF'

```

执行恢复，将从新的日志读取信息，文件 4 得以成功恢复：

```

SQL> recover datafile 4;
Media recovery complete.
SQL> alter database open;
Database altered.

```

```
SQL> select name,status from v$datafile
2 /
NAME                                STATUS
-----
C:\ORACLE\ORADATA\ORA9I\SYSTEM01.DBF    SYSTEM
C:\ORACLE\ORADATA\ORA9I\UNDOTBS01.DBF    ONLINE
C:\ORACLE\ORADATA\ORA9I\USERS01.DBF      ONLINE
C:\ORACLE\ORADATA\ORA9I\USERS02.DBF      ONLINE
SQL> select count(*) from eygle;
COUNT(*)
-----
12646
```

以下是告警日志文件中的信息输出：

```
Wed Jan 11 17:06:05 2012
alter database open
ORA-1113 signalled during: alter database open...
Wed Jan 11 17:06:10 2012
ALTER DATABASE RECOVER datafile 4
Media Recovery Start
Wed Jan 11 17:06:10 2012
Recovery of Online Redo Log: Thread 1 Group 3 Seq 78 Reading mem 0
  Mem# 0 errs 0: C:\ORACLE\ORADATA\ORA9I\REDO03.LOG
Media Recovery Complete
Completed: ALTER DATABASE RECOVER datafile 4
Wed Jan 11 17:06:13 2012
alter database open
Wed Jan 11 17:06:13 2012
LGWR: Primary database is in CLUSTER CONSISTENT mode
Wed Jan 11 17:06:13 2012
ARCH: Evaluating archive log 1 thread 1 sequence 76
ARCH: Beginning to archive log 1 thread 1 sequence 76
Creating archive destination LOG_ARCHIVE_DEST_1: 'C:\ORACLE\9.2.0\RDBMS\ARC00076.001'
ARCH: Completed archiving log 1 thread 1 sequence 76
```

```
Wed Jan 11 17:06:13 2012
LGWR: Primary database is in CLUSTER CONSISTENT mode
Thread 1 advanced to log sequence 79
Thread 1 opened at log sequence 79
  Current log# 1 seq# 79 mem# 0: C:\ORACLE\ORADATA\ORA9I\RED001.LOG
Successful open of redo thread 1
Wed Jan 11 17:06:13 2012
SMON: enabling cache recovery
Wed Jan 11 17:06:14 2012
Successfully onlined Undo Tablespace 1.
Dictionary check beginning
Tablespace 'TEMP' #2 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Dictionary check complete
Wed Jan 11 17:06:14 2012
SMON: enabling tx recovery
Wed Jan 11 17:06:15 2012
Database Characterset is ZHS16GBK
replication_dependency_tracking turned off (no async multimaster replication found)
Completed: alter database open
```

注意这种方法是不得已情况下采取的非常手段，通常不被官方支持，如果跳过的日志中涵盖的大量事务，数据将不可避免的出现不一致的问题，一般建议用户通过导出重建数据库。

空间与文件离线

离线表空间加载修复

与前面两个案例极其相似，我在 2012 年又遇到了一则类似的案例，这一次是某银行用户，数据同样极其重要。

灾难描述

这是又一次由于文件离线保护而导致的数据灾难：

1. 用户的归档空间满，数据库崩溃
2. 用户删除了归档日志，释放空间
3. 用户恢复数据库运行
4. 发现有两个数据文件离线，数据量大约 40G
5. 由于归档日志删除，数据文件无法加载
6. 无备份，数据无法追补，灾难形成

这则故障与前面一则故障类似，恢复方法也是类似的。

案例警示

这个案例给我们的警示有：

1. **备份重于一切，备份是数据库应用的第一要义**

再次重申：唯一能让公司主管从夜里警醒的就是，数据备份。

如果每天晚上睡觉前，你想一想，哪个重要的业务数据库没有备份，假定当晚磁盘损坏，系统崩溃，文件丢失，你将如何应对？

我想每一个 IT 主管都无法在没有备份保障机制下安然入睡。

备份对于数据库来说重于一切。

2. 不要轻易的删除任何一个归档日志

哪怕是在空间紧张的情况下，也不要轻易删除任何一个归档日志文件，压缩或者转移会是临时释放空间的办法。

由于归档日志对于数据库的恢复不可缺少，所以，在你不能确定数据库完全无故障隐患之前，最好不要草率删除日志。我们已经遇到了众多与日志缺失有关的案例，所以我们严正的提醒大家：**不要轻易的删除任何一个归档日志。**

3. 日志监控与检查是防范问题的根本

有了备份的保障，再加上日常监控与检查才能够让数据库高枕无忧。

这里所说的监控和检查其实可以简化到 Oracle 数据库的告警日志（alert 日志）检查。Oracle 会将数据库的异常全部记录在告警日志文件当中，实时或准时时监控或检查告警日志，发现其中的问题，并做出处理和响应，是防范问题的基础与根本。

这个案例，如果用户能够检查日志，做好基本工作，那就可以防范灾难的出现，而如果有良好的备份，则灾难出现后也就不会手足无措。

技术回放

用户数据量大约 3T，是一个非常重要的核心数据库：

```
SYS@bank>select sum(bytes)/1024/1024 from v$datafile;  
SUM(BYTES)/1024/1024
```

```
-----
```

```
3415000
```

首先由于磁盘空间满，导致 Oracle 数据库对于文件的读写失败，有两个文件被离线，这是 Oracle 数据库的一种保护机制：

```
Fri Jan 13 19:22:21 2012
```

```
KCF: write/open error block=0xf1fa6 online=1
```

```
file=73 /dev/rbank_gdm05
```

```
error=27063 txt: 'IBM AIX RISC System/6000 Error: 22: Invalid argument
```

```

Additional information: -1
Additional information: 557056'
Automatic datafile offline due to write error on
file 73: /dev/rbank_gdm05
Fri Jan 13 19:22:53 2012
KCF: write/open error block=0xf91cd online=1
      file=74 /dev/rbank_gdm06
      error=27063 txt: 'IBM AIX RISC System/6000 Error: 5: I/O error
Additional information: -1
Additional information: 32768'
Automatic datafile offline due to write error on
file 74: /dev/rbank_gdm06
Fri Jan 13 19:22:53 2012
KCF: write/open error block=0xf9145 online=1
      file=74 /dev/rbank_gdm06
      error=27063 txt: 'IBM AIX RISC System/6000 Error: 22: Invalid argument
Additional information: -1
Additional information: 32768'
Automatic datafile offline due to write error on
file 74: /dev/rbank_gdm06

```

接下来控制文件的写操作失败,数据库崩溃,控制文件的故障表现在于队列无法获取,队列 CF 代表 Control File:

```

Fri Jan 13 19:38:12 2012
Errors in file /oracle/admin/bank/bdump/bank_arc1_438808.trc:
ORA-00494: enqueue [CF] held for too long (more than 900 seconds) by 'inst 1, osid
475952'
Fri Jan 13 19:38:14 2012
System State dumped to trace file /oracle/admin/bank/bdump/bank_arc1_438808.trc
Killing enqueue blocker (pid=475952) on resource CF-00000000-00000000
  by killing session 3302.1
Fri Jan 13 19:38:19 2012
Errors in file /oracle/admin/bank/bdump/bank_arc0_229686.trc:
ORA-00494: enqueue [CF] held for too long (more than 900 seconds) by 'inst 1, osid

```

```

475952'
Fri Jan 13 19:38:20 2012
System State dumped to trace file /oracle/admin/bank/bdump/bank_arc0_229686.trc
Killing enqueue blocker (pid=475952) on resource CF-00000000-00000000
  by killing session 3302.1
Fri Jan 13 19:43:17 2012
Errors in file /oracle/admin/bank/bdump/bank_arc1_438808.trc:
ORA-00494: enqueue [CF] held for too long (more than 900 seconds) by 'inst 1, osid
475952'
Fri Jan 13 19:43:19 2012
System State dumped to trace file /oracle/admin/bank/bdump/bank_arc1_438808.trc
Killing enqueue blocker (pid=475952) on resource CF-00000000-00000000
  by terminating the process
ARC1: terminating instance due to error 2103
Instance terminated by ARC1, pid = 438808

```

客户清理日志，释放空间后启动数据库：

```

Fri Jan 13 20:45:08 2012
ALTER DATABASE OPEN
Fri Jan 13 20:45:08 2012
Beginning crash recovery of 1 threads
  parallel recovery started with 15 processes
Fri Jan 13 20:45:09 2012
Started redo scan

```

注意此处，有 40479 个数据块需要恢复，这是相当多的内容：

```

Fri Jan 13 20:45:26 2012
Completed redo scan
  2561915 redo blocks read, 40479 data blocks need recovery

```

以下所有的日志组都需要用于恢复，起始的日志序号为 89206，起始的日志块号为 300488，以下显示了这些日志的恢复过程：

```

Fri Jan 13 20:45:54 2012
Started redo application at

```

```

Thread 1: logseq 89206, block 300488
Fri Jan 13 20:45:54 2012
Recovery of Online Redo Log: Thread 1 Group 1 Seq 89206 Reading mem 0
  Mem# 0: /dev/rbank_redo1_01
  Mem# 1: /dev/rbank_redo1_11
Fri Jan 13 20:46:03 2012
Recovery of Online Redo Log: Thread 1 Group 2 Seq 89207 Reading mem 0
  Mem# 0: /dev/rbank_redo1_02
  Mem# 1: /dev/rbank_redo1_12
Fri Jan 13 20:46:23 2012
Recovery of Online Redo Log: Thread 1 Group 3 Seq 89208 Reading mem 0
  Mem# 0: /dev/rbank_redo1_03
  Mem# 1: /dev/rbank_redo1_13
Fri Jan 13 20:46:41 2012
Completed redo application

```

完成恢复，读取了 2561915 个日志块，40447 个数据块写出。

```

Fri Jan 13 20:46:41 2012
Completed crash recovery at
Thread 1: logseq 89208, block 928298, scn 3386095894
40479 data blocks read, 40447 data blocks written, 2561915 redo blocks read

```

数据库启动之后，日志继续切换应用，注意这里的日志序列从 89209 开始：

```

Fri Jan 13 20:47:04 2012
Thread 1 advanced to log sequence 89209 (thread open)
Thread 1 opened at log sequence 89209
  Current log# 1 seq# 89209 mem# 0: /dev/rbank_redo1_01
  Current log# 1 seq# 89209 mem# 1: /dev/rbank_redo1_11
Successful open of redo thread 1
Fri Jan 13 20:47:05 2012
MTTR advisory is disabled because FAST_START_MTTR_TARGET is not set
Fri Jan 13 20:47:05 2012
SMON: enabling cache recovery
Fri Jan 13 20:47:07 2012

```

```

Successfully online Undo Tablespace 1.
Fri Jan 13 20:47:07 2012
SMON: enabling tx recovery
Fri Jan 13 20:47:07 2012
Database Characterset is ZHS16GBK
Opening with internal Resource Manager plan
where NUMA PG = 1, CPUs = 16
replication_dependency_tracking turned off (no async multimaster replication found)
Starting background process QMNC
QMNC started with pid=35, OS id=540710
Fri Jan 13 20:47:19 2012
db_recovery_file_dest_size of 2048 MB is 10.33% used. This is a
user-specified limit on the amount of space that will be used by this
database for recovery-related files, and does not reflect the amount of
space available in the underlying filesystem or ASM diskgroup.
Fri Jan 13 20:47:19 2012
Completed: ALTER DATABASE OPEN
Fri Jan 13 21:07:03 2012
ALTER SYSTEM ARCHIVE LOG
Fri Jan 13 21:07:03 2012
Thread 1 cannot allocate new log, sequence 89210
Private strand flush not complete
    Current log# 1 seq# 89209 mem# 0: /dev/rbank_redo1_01
    Current log# 1 seq# 89209 mem# 1: /dev/rbank_redo1_11
Fri Jan 13 21:07:05 2012
Thread 1 advanced to log sequence 89210 (LGWR switch)
    Current log# 2 seq# 89210 mem# 0: /dev/rbank_redo1_02
    Current log# 2 seq# 89210 mem# 1: /dev/rbank_redo1_12

```

注意这里提示闪回归档区 100%使用率，空间用完了：

```

Fri Jan 13 21:07:39 2012
Errors in file /oracle/admin/bank/udump/bank_ora_332386.trc:
ORA-19815: WARNING: db_recovery_file_dest_size of 2147483648 bytes is 100.00% used,
and has 0 remaining bytes available.

```

Fri Jan 13 21:07:39 2012

You have following choices to free up space from flash recovery area:

1. Consider changing RMAN RETENTION POLICY. If you are using Data Guard, then consider changing RMAN ARCHIVELOG DELETION POLICY.
2. Back up files to tertiary device such as tape using RMAN BACKUP RECOVERY AREA command.
3. Add disk space and increase db_recovery_file_dest_size parameter to reflect the new space.
4. Delete unnecessary files using RMAN DELETE command. If an operating system command was used to delete files, then use RMAN CROSSCHECK and DELETE EXPIRED commands.

Fri Jan 13 21:08:03 2012

ALTER SYSTEM ARCHIVE LOG

Fri Jan 13 21:08:03 2012

Thread 1 cannot allocate new log, sequence 89211

Private strand flush not complete

Current log# 2 seq# 89210 mem# 0: /dev/rbank_redo1_02

Current log# 2 seq# 89210 mem# 1: /dev/rbank_redo1_12

Fri Jan 13 21:08:05 2012

Thread 1 advanced to log sequence 89211 (LGWR switch)

Current log# 3 seq# 89211 mem# 0: /dev/rbank_redo1_03

Current log# 3 seq# 89211 mem# 1: /dev/rbank_redo1_13

Fri Jan 13 21:08:13 2012

Errors in file /oracle/admin/bank/udump/bank_ora_536702.trc:

ORA-19815: WARNING: db_recovery_file_dest_size of 2147483648 bytes is 100.00% used, and has 0 remaining bytes available.

Fri Jan 13 21:08:13 2012

You have following choices to free up space from flash recovery area:

1. Consider changing RMAN RETENTION POLICY. If you are using Data Guard, then consider changing RMAN ARCHIVELOG DELETION POLICY.

2. Back up files to tertiary device such as tape using RMAN
BACKUP RECOVERY AREA command.
3. Add disk space and increase db_recovery_file_dest_size parameter to
reflect the new space.
4. Delete unnecessary files using RMAN DELETE command. If an operating
system command was used to delete files, then use RMAN CROSSCHECK and
DELETE EXPIRED commands.

Fri Jan 13 21:12:02 2012

ALTER SYSTEM ARCHIVE LOG

Fri Jan 13 21:12:02 2012

Thread 1 cannot allocate new log, sequence 89212

Checkpoint not complete

Current log# 3 seq# 89211 mem# 0: /dev/rbank_redo1_03

Current log# 3 seq# 89211 mem# 1: /dev/rbank_redo1_13

Thread 1 cannot allocate new log, sequence 89212

Private strand flush not complete

Current log# 3 seq# 89211 mem# 0: /dev/rbank_redo1_03

Current log# 3 seq# 89211 mem# 1: /dev/rbank_redo1_13

Fri Jan 13 21:12:05 2012

Thread 1 advanced to log sequence 89212 (LGWR switch)

Current log# 1 seq# 89212 mem# 0: /dev/rbank_redo1_01

Current log# 1 seq# 89212 mem# 1: /dev/rbank_redo1_11

注意此时空间被释放出来

Fri Jan 13 21:12:07 2012

db_recovery_file_dest_size of 2048 MB is 0.06% used. This is a
user-specified limit on the amount of space that will be used by this
database for recovery-related files, and does not reflect the amount of
space available in the underlying filesystem or ASM diskgroup.

然后用户才发现文件离线，之后这两个文件无法 Online，故障出现了：

Fri Jan 13 21:23:01 2012

alter database recover datafile 73

```
Fri Jan 13 21:23:01 2012
Media Recovery Start
  parallel recovery started with 15 processes
ORA-279 signalled during:
alter database recover datafile 73
...
Fri Jan 13 21:28:13 2012
alter database datafile 73 online

Fri Jan 13 21:28:13 2012
ORA-1113 signalled during:
alter database datafile 73 online
...
Fri Jan 13 21:28:29 2012
alter database datafile 74 online
Fri Jan 13 21:28:30 2012
ORA-1113 signalled during:
alter database datafile 74 online
...
Fri Jan 13 21:28:41 2012
alter database recover datafile 74
Fri Jan 13 21:28:42 2012
Media Recovery Start
ORA-275 signalled during:
alter database recover datafile 74
...
Fri Jan 13 21:29:09 2012
alter database datafile 74 online
Fri Jan 13 21:29:09 2012
ORA-1113 signalled during:
alter database datafile 74 online
...
```

查询 v\$datafile 视图，可以看到这两个文件处于 RECOVER 状态：

```
SYS@bank>select file#,status from v$datafile where status='RECOVER';
```

```
FILE# STATUS
-----
73 RECOVER
74 RECOVER
```

恢复过程

通过控制文件转储，可以开始分析一下文件的检查点等相关信息：

```
SYS@bank>alter database backup controlfile to trace;
```

```
Database altered.
```

```
SYS@bank>alter session set events 'immediate trace name file_hdrs level 12';
```

```
Session altered.
```

以下摘录三个文件的内容，这三个文件属于同一个表空间：

```
72 /dev/rbank_gdm04
73 /dev/rbank_gdm05
74 /dev/rbank_gdm06
```

以下是来自文件头和控制文件的信息：

```
DATA FILE #72:
```

```
(name #80) /dev/rbank_gdm04
```

```
creation size=640000 block size=32768 status=0xe head=80 tail=80 dup=1
```

```
tablespace 27, index=27 krfil=72 prev_file=71
```

```
unrecoverable scn: 0x0000.c4664adc 12/27/2011 11:48:51
```

控制文件中记录的文件检查点信息：

```
Checkpoint cnt:68648 scn: 0x0000.c9d832b3 01/13/2012 23:19:03
```

```
Stop scn: 0x0000.c9d832b3 01/13/2012 23:19:03
```

```
Creation Checkpointed at scn: 0x0000.50bf07ae 04/21/2010 09:45:20
```

```
thread:1 rba:(0x404b.e8de9.10)
```

```
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
```

```

Offline scn: 0x0000.00000000 prev_range: 0
Online Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
V10 STYLE FILE HEADER:
    Compatibility Vsn = 169870080=0xa200300
    Db ID=2256689008=0x86825770, Db Name='BANK'
    Activation ID=0=0x0
    Control Seq=5695898=0x56e99a, File size=1600000=0x186a00
    File Number=72, Blksiz=32768, File Type=3 DATA
Tablespace #27 - BANK2 rel_fn:72
Creation at scn: 0x0000.50bf07ae 04/21/2010 09:45:20
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2997f230 scn: 0x0000.00000001 reset logs terminal rcv data:0x0
scn: 0x0000.00000000
prev reset logs count:0x0 scn: 0x0000.00000000 prev reset logs terminal rcv data:0x0
scn: 0x0000.00000000
recovered at 01/13/2012 20:45:09
status:0x0 root dba:0x00000000 chkpt cnt: 68648 ctl cnt:68647
begin-hot-backup file size: 0

```

数据文件头上记录的检查点 SCN，可以看到文件头与控制文件一致，这说明该文件一致。时间点是 23: 19: 03

```

Checkpointed at scn: 0x0000.c9d832b3 01/13/2012 23:19:03
thread:1 rba:(0x15c7c.1a4d8.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
Backup Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000

```

```

DATA FILE #73:
(name #81) /dev/rbank_gdm05

```

```
creation size=640000 block size=32768 status=0x1c head=81 tail=81 dup=1
tablespace 27, index=27 krfil=73 prev_file=72
unrecoverable scn: 0x0000.c4664adf 12/27/2011 11:48:52
```

对于 73 号文件，其控制文件记录该文件的检查点停留在 19: 20: 21

```
Checkpoint cnt:68641 scn: 0x0000.c9d2c793 01/13/2012 19:20:21
Stop scn: 0x0000.c9d337f8 01/13/2012 19:22:52
Creation Checkpointed at scn: 0x0000.50bf08bb 04/21/2010 09:49:06
thread:1 rba:(0x404b.e910d.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
Offline scn: 0x0000.00000000 prev_range: 0
Online Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
V10 STYLE FILE HEADER:
    Compatibility Vsn = 169870080=0xa200300
    Db ID=2256689008=0x86825770, Db Name='BANK'
    Activation ID=0=0x0
    Control Seq=5695894=0x56e996, File size=1600000=0x186a00
    File Number=73, Blksiz=32768, File Type=3 DATA
Tablespace #27 - BANK2 rel_fn:73
Creation at scn: 0x0000.50bf08bb 04/21/2010 09:49:06
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2997f230 scn: 0x0000.00000001 reset logs terminal rcv data:0x0
scn: 0x0000.00000000
prev reset logs count:0x0 scn: 0x0000.00000000 prev reset logs terminal rcv data:0x0
scn: 0x0000.00000000
recovered at 01/13/2012 21:23:02
status:0x4 root dba:0x00000000 chkpt cnt: 68641 ctl cnt:68640
begin-hot-backup file size: 0
```

数据文件头记录的信息和控制文件一致，停留在 19:20:21。

```
Checkpointed at scn: 0x0000.c9d2c793 01/13/2012 19:20:21
```

接下来这里的 RBA 信息中止处，恢复该文件的日志文件序号是 89205 (15c75)，这正是数据库崩溃前最后一个日志序列号。

数据库启动之后恢复的序列从 89206 开始，这之后的恢复与 73 号文件无关，因为该文件已经离线。

```
thread:1 rba:(0x15c75.2.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
Backup Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
```

```
DATA FILE #74:
```

```
(name #82) /dev/rbank_gdm06
creation size=640000 block size=32768 status=0x1c head=82 tail=82 dup=1
tablespace 27, index=27 krfil=74 prev_file=73
unrecoverable scn: 0x0000.c4664ae2 12/27/2011 11:48:52
```

74 号文件和 73 号文件相同，检查点停留在数据库崩溃之前

```
Checkpoint cnt:68641 scn: 0x0000.c9d2c793 01/13/2012 19:20:21
Stop scn: 0x0000.c9d3b918 01/13/2012 20:47:04
Creation Checkpointed at scn: 0x0000.50bf0a62 04/21/2010 09:52:55
thread:1 rba:(0x404b.e9549.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
Offline scn: 0x0000.00000000 prev_range: 0
Online Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
V10 STYLE FILE HEADER:
Compatibility Vsn = 169870080=0xa200300
Db ID=2256689008=0x86825770, Db Name='BANK'
Activation ID=0=0x0
Control Seq=5695583=0x56e85f, File size=1600000=0x186a00
```

```

File Number=74, Blksiz=32768, File Type=3 DATA
Tablespace #27 - BANK2 rel_fn:74
Creation at scn: 0x0000.50bf0a62 04/21/2010 09:52:55
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2997f230 scn: 0x0000.00000001 reset logs terminal rcv data:0x0
scn: 0x0000.00000000
prev reset logs count:0x0 scn: 0x0000.00000000 prev reset logs terminal rcv data:0x0
scn: 0x0000.00000000
recovered at 01/01/1988 00:00:00
status:0x4 root dba:0x00000000 chkpt cnt: 68641 ctl cnt:68640
begin-hot-backup file size: 0

```

文件头记录的信息和之前一致:

```

Checkpointed at scn: 0x0000.c9d2c793 01/13/2012 19:20:21
thread:1 rba:(0x15c75.2.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
Backup Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000

```

这种情况我们只能尽量挽救数据，由于文件离线，归档日志丢失，无法进行恢复，只能通过修改数据文件头信息的方式，使数据库跳过缺失的文件，强制加载数据文件。

通常如果恢复之后出现错误，则需要导出数据，重建数据库；如果幸运，则不会出现任何错误，数据库可以继续运行。

最简单的操作，我们可以将 72 号文件的头块复制到 73 号和 74 号文件，修改相应的文件号之后，可以打开数据库，其修改过程与上一则案例相同：

```

copy file 72 block 1 to file 73 block 1;
copy file 72 block 1 to file 74 block 1;

```

○○○○

之后重建控制文件后，即可打开数据库：

```

SYS@bank>startup nomount;
ORACLE instance started.

```

```

Total System Global Area 1.8874E+10 bytes
Fixed Size                2117456 bytes
Variable Size             704643248 bytes
Database Buffers          1.8153E+10 bytes
Redo Buffers              14659584 bytes
SYS@bank>@cr
Control file created.
SYS@bank>recover datafile 73;
ORA-00283: recovery session canceled due to errors
ORA-00264: no recovery required
SYS@bank>recover datafile 74;
ORA-00283: recovery session canceled due to errors
ORA-00264: no recovery required
SYS@bank>alter database open;
Database altered.
SYS@bank>select file#,status from v$datafile where status='RECOVER';
0 rows selected.

```

查询数据文件上的数据，能够成功返回结果：

```

SYS@bank>col segment_name for a40
SYS@bank>select owner,segment_name,segment_type
           2 from dba_extents where file_id=73 and rownum < 2;

```

OWNER	SEGMENT_NAME	SEGMENT_TYPE
BODANK	F_CB_KDA_AUTD_H	TABLE

```

SYS@bank>select count(*) from BODANK.F_CB_KDA_AUTD_H;
COUNT(*)
-----
1141005

```

增加临时表空间：

```

SYS@bank>ALTER TABLESPACE TEMP ADD TEMPFILE '/dev/rbank_temp02' REUSE;
Tablespace altered.
SYS@bank>ALTER TABLESPACE TEMP ADD TEMPFILE '/dev/rbank_temp01' REUSE;

```

Tablespace altered.

这样就完美的完成了这则数据灾难的恢复工作，这次故障恢复之所以顺利，是因为用户在之后快速的发现并处理，及时发现问题是控制灾难影响的关键。

技术提示

关于归档空间的设置

我们注意到很多用户在设置了归档模式之后，却没有针对性的归档备份清理机制，也没有良好的空间规划，结果归档模式为用户带来了许多麻烦和灾难。

虽然归档模式是为了数据安全而设定的，但是如果用户无法掌握关于归档的维护，则这一设定是得不偿失的。

尤其在 Oracle 10g 引入了闪回恢复区之后，db_recovery_file_dest_size 的设置让很多客户遭受了惨痛的教训。

在本案例中，客户的存储空间并非不足，但是缺省的 db_recovery_file_dest_size 设置仅有 2G，客户未能主动去修改和调整这个参数：

```
db_recovery_file_dest    = /oracle/flash_recovery_area
db_recovery_file_dest_size= 2147483648
```

以下是告警日志中的信息摘录，这 2G 的归档空间在高峰期可能会很快耗尽：

```
db_recovery_file_dest_size= 2147483648
db_recovery_file_dest_size of 2048 MB is 10.33% used. This is a
ORA-19815: WARNING: db_recovery_file_dest_size of 2147483648 bytes is 100.00% used,
and has 0 remaining bytes available.
```

而且可能更为折磨的是，如果仅只通过手工删除归档日志，Oracle 并不认可这种空间释放，必须通过 Oracle 的手段清理，才能让数据库认识到这种变化。

以下是一个案例，当 db_recovery_file_dest_size 100% 用尽时，告警日志中会出现以下处理提示：

```
ORA-19815: WARNING: db_recovery_file_dest_size of 2147483648 bytes is 100.00% used,
and has 0 remaining bytes available.
```

```
*****
```

You have the following choices to free up space from
flash recovery area:

1. Consider changing your RMAN retention policy.
If you are using dataguard, then consider changing your
RMAN archivelog deletion policy.
2. Backup files to tertiary device such as tape using the
RMAN command BACKUP RECOVERY AREA.
3. Add disk space and increase the db_recovery_file_dest_size
parameter to reflect the new space.
4. Delete unnecessary files using the RMAN DELETE command.
If an OS command was used to delete files, then use
RMAN CROSSCHECK and DELETE EXPIRED commands.

```
*****
```

此时，手工释放空间，甚至切换闪回区到一个全新的磁盘，仍然无法解决问题，在数据库内部记录的信息不会变更：

```
SYS AS SYSDBA>set linesize 120
```

```
SYS AS SYSDBA>SELECT substr(name, 1, 30) name, space_limit AS quota,
```

```
2      space_used      AS used,
```

```
3      space_reclaimable AS reclaimable,
```

```
4      number_of_files  AS files
```

```
5  FROM  v$recovery_file_dest;
```

NAME	QUOTA	USED	RECLAIMABLE	FILES
-----	-----	-----	-----	-----
/data5/flash_recovery_area	2147483648	2144863232	0	227

可以看到，闪回区仍然记录了 227 个文件，USED 空间并未释放，释放的方式，可以通过 RMAN 进行 crosscheck，删除过期或废弃的文件：

```
RMAN> crosscheck archivelog all;
```

```
using target database controlfile instead of recovery catalog
```

```
allocated channel: ORA_DISK_1
```

```
channel ORA_DISK_1: sid=144 devtype=DISK
```

```
validation failed for archived log
```

```

archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_05_17/o1_mf_1_790_0bjq36ps_.arc recid=1 stamp=526401126
validation failed for archived log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_05_17/o1_mf_1_791_0bkbcy7x_.arc recid=2 stamp=526420862
validation failed for archived log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_05_17/o1_mf_1_792_0bklds4d_.arc recid=3 stamp=526428057
.....
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1014_0hh3zsrp_.arc recid=225 stamp=531678074
validation failed for archived log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1015_0hh40qyp_.arc recid=226 stamp=531678104
validation failed for archived log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1016_0hh41jqj_.arc recid=227 stamp=531678129
Crosschecked 227 objects
RMAN> delete noprompt expired archivelog all;
deleted archive log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_05_17/o1_mf_1_790_0bjq36ps_.arc recid=1 stamp=526401126
deleted archive log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_05_17/o1_mf_1_791_0bkbcy7x_.arc recid=2 stamp=526420862
deleted archive log
.....
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1014_0hh3zsrp_.arc recid=225 stamp=531678074
deleted archive log
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1015_0hh40qyp_.arc recid=226 stamp=531678104
deleted archive log

```

```
archive log filename=/opt/oracle/flash_recovery_area/EYGLE/
archivelog/2004_07_16/o1_mf_1_1016_0hh41jqj.arc recid=227 stamp=531678129
Deleted 227 EXPIRED objects
```

此时空间才会得以释放:

```
SYS AS SYSDBA on 28-MAR-05 >SELECT substr(name, 1, 30) name, space_limit AS quota,
2      space_used      AS used,
3      space_reclaimable AS reclaimable,
4      number_of_files  AS files
5  FROM  v$recovery_file_dest;

NAME                                QUOTA      USED RECLAIMABLE  FILES
-----
/data5/flash_recovery_area          2147483648  9959424         0        1
```

如果磁盘空间充足，最快速的解决方案是通过命令扩展闪回区的空间设定，以下是一个案例处理过程的参数修改:

```
SQL> show parameter recov

NAME                                TYPE        VALUE
-----
db_recovery_file_dest               string       /msflsh
db_recovery_file_dest_size          big integer  50G
recovery_parallelism                integer      0

SQL> alter system set db_recovery_file_dest_size=65G scope=both;
System altered

SQL> show parameter recov

NAME                                TYPE        VALUE
-----
db_recovery_file_dest               string       /msflsh
db_recovery_file_dest_size          big integer  65G
recovery_parallelism                integer      0
```

此外，如果归档日志可以全部清除，不需备份，则可以通过类似如下方式的指定日期直接删除:

```
RMAN> DELETE NOPROMPT ARCHIVELOG UNTIL TIME 'SYSDATE-2';
using target database control file instead of recovery catalog
```

allocated channel: ORA_DISK_1

channel ORA_DISK_1: SID=27 device type=DISK

List of Archived Log Copies for database with db_unique_name ORCL11G

=====

Key	Thrd	Seq	S	Low Time
7	1	101	A	19-JAN-12
				Name: /archivelog/2012_01_19/o1_mf_1_101_7khsv8o5_.arc
8	1	102	A	19-JAN-12
				Name: /archivelog/2012_01_19/o1_mf_1_102_7khsv9rr_.arc
9	1	103	A	19-JAN-12
				Name: /archivelog/2012_01_19/o1_mf_1_103_7khs82o_.arc
10	1	104	A	19-JAN-12
				Name: /archivelog/2012_01_19/o1_mf_1_104_7kht29ns_.arc
11	1	105	A	19-JAN-12
				Name: /archivelog/2012_01_21/o1_mf_1_105_7kmvyxlx_.arc
12	1	106	A	21-JAN-12
				Name: /archivelog/2012_01_23/o1_mf_1_106_7krdp0qd_.arc

deleted archived log

archived log file name=/2012_01_19/o1_mf_1_101_7khsv8o5_.arc RECID=7 STAMP=772998632

deleted archived log

archived log file name=/2012_01_19/o1_mf_1_102_7khsv9rr_.arc RECID=8 STAMP=772998633

deleted archived log

archived log file name=/2012_01_19/o1_mf_1_103_7khs82o_.arc RECID=9 STAMP=772998664

deleted archived log

archived log file name=/2012_01_19/o1_mf_1_104_7kht29ns_.arc RECID=10 STAMP=772998857

deleted archived log

archived log file name=/2012_01_21/o1_mf_1_105_7kmvyxlx_.arc RECID=11 STAMP=773131877

deleted archived log

archived log file name=/2012_01_23/o1_mf_1_106_7krdp0qd_.arc RECID=12 STAMP=773280072

Deleted 11 objects

关于检查点的一致性调整

文件离线，如果恢复在线状态，主要是 SCN 与检查点以及 RBA 在发挥作用，如果归档日志不可用，则通过 BBED 方式跳过日志是可以尝试的。

直接修改文件检查点和 RBA 信息，也可以达到同样的效果，以下是一个测试说明，供参考。

首先将文件离线：

```
SQL> col name for a50
```

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

```
FILE# NAME
```

```
-----
1 /u01/app/oracle/oradata/ORCL/system01.dbf
2 /u01/app/oracle/oradata/ORCL/undotbs01.dbf
3 /u01/app/oracle/oradata/ORCL/sysaux01.dbf
4 /u01/app/oracle/oradata/ORCL/users01.dbf
5 /u01/app/oracle/oradata/ORCL/marven01.dbf
6 /u01/app/oracle/oradata/ORCL/xxx.dbf
7 /u01/app/oracle/oradata/ORCL/eygle01.dbf
```

```
7 rows selected.
```

```
SQL> alter database datafile 7 offline drop;
```

```
Database altered.
```

该文件状态变更为 RECOVER：

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

```
FILE# STATUS
```

```
-----
1 SYSTEM
2 ONLINE
3 ONLINE
4 ONLINE
5 ONLINE
6 ONLINE
```

7 RECOVER

7 rows selected.

切换日志，覆盖就的 REDO 日志，当再次执行恢复时，需要的日志将不可用：

```
SQL> alter system switch logfile;
```

```
System altered.
```

```
SQL> /
```

```
System altered.
```

```
SQL> /
```

```
System altered.
```

```
SQL> /
```

```
System altered.
```

```
SQL> recover datafile 7;
```

```
ORA-00279: change 4282927 generated at 01/16/2012 16:43:36 needed for thread 1
```

```
ORA-00289: suggestion :
```

```
/u01/app/oracle/flash_recovery_area/ORCL/archivelog/2012_01_16/o1_mf_1_204_%u_.a  
rc
```

```
ORA-00280: change 4282927 for thread 1 is in sequence #204
```

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

```
auto
```

```
ORA-00308: cannot open archived log
```

```
'/u01/app/oracle/flash_recovery_area/ORCL/archivelog/2012_01_16/o1_mf_1_204_%u_.  
arc'
```

```
ORA-27037: unable to obtain file status
```

```
Linux-x86_64 Error: 2: No such file or directory
```

```
Additional information: 3
```

```
ORA-00308: cannot open archived log
```

```
'/u01/app/oracle/flash_recovery_area/ORCL/archivelog/2012_01_16/o1_mf_1_204_%u_.  
arc'
```

```
ORA-27037: unable to obtain file status
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3
```

使用 BBED，通过 SYSTEM 表空间作为参照物，修改离线数据文件的检查点信息等：

```
[oracle@hpserver2 lib]$ bbed parfile=par.txt
Password:
```

```
BBED: Release 2.0.0.0.0 - Limited Production on Mon Jan 16 17:29:06 2012
```

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

```
***** !!! For Oracle Internal Use only !!! *****
```

```
BBED> p kcvfhckp file 1
```

```
struct kcvfhckp, 36 bytes                                @484
  struct kcvcp scn, 8 bytes                               @484
    ub4 kscnbas                                           @484      0x00416bb3 < 当前检查点信息
    ub2 kscnwrp                                           @488      0x0000
  ub4 kcvcptim                                           @492      0x2e0f0c15 < 当前检查点时间
  ub2 kcvcpthr                                           @496      0x0001
  union u, 12 bytes                                     @500
    struct kcvcp rba, 12 bytes                            @500
      ub4 kcrbaseq                                         @500      0x000000d0
      ub4 kcrbabno                                         @504      0x00000002
      ub2 kcrbabof                                         @508      0x0010
  ub1 kcvcpetb[0]                                         @512      0x02
  ub1 kcvcpetb[1]                                         @513      0x00
  ub1 kcvcpetb[2]                                         @514      0x00
  ub1 kcvcpetb[3]                                         @515      0x00
  ub1 kcvcpetb[4]                                         @516      0x00
  ub1 kcvcpetb[5]                                         @517      0x00
  ub1 kcvcpetb[6]                                         @518      0x00
  ub1 kcvcpetb[7]                                         @519      0x00
```

```
BBED> p kcvfhckp file 7
```

```

struct kcvfhckp, 36 bytes          @484
    struct kcvcpscn, 8 bytes      @484
        ub4 kscnbas               @484      0x00415a2f < 文件目前的检查点
        ub2 kscnwrp               @488      0x00000
    ub4 kvcptim                   @492      0x2e0f01b8
    ub2 kvcpthr                   @496      0x00001
    union u, 12 bytes             @500
        struct kvcprba, 12 bytes  @500
            ub4 kcrbaseq           @500      0x000000cc
            ub4 kcrbabno           @504      0x000000374
            ub2 kcrbabof           @508      0x00010
    ub1 kvcpetb[0]                @512      0x02
    ub1 kvcpetb[1]                @513      0x00
    ub1 kvcpetb[2]                @514      0x00
    ub1 kvcpetb[3]                @515      0x00
    ub1 kvcpetb[4]                @516      0x00
    ub1 kvcpetb[5]                @517      0x00
    ub1 kvcpetb[6]                @518      0x00
    ub1 kvcpetb[7]                @519      0x00

```

BBED> set count 8

```
COUNT          8
```

BBED> dump

File: /u01/app/oracle/oradata/ORCL/eygle01.dbf (7)

```
Block: 1          Offsets: 484 to 491          DbA:0x01c00001
```

```
-----
2f5a4100 00000000
```

<32 bytes per line>

以下将文件的检查点修改为 SYSTEM 表空间的检查点:

BBED> modify /x b36b file 7 block 1 offset 484 < 修改为 SYSTEM 表空间的检查点

File: /u01/app/oracle/oradata/ORCL/eygle01.dbf (7)

```
Block: 1          Offsets: 484 to 491          DbA:0x01c00001
```

```
b36b4100 00000000
```

```
<32 bytes per line>
```

以下命令修改检查点时间:

```
BBED> modify /x 150c0f2e file 7 block 1 offset 492 < 修改检查点时间
```

```
File: /u01/app/oracle/oradata/ORCL/eygle01.dbf (7)
```

```
Block: 1           Offsets: 492 to 499           DbA:0x01c00001
```

```
-----  
150c0f2e 01000000
```

```
<32 bytes per line>
```

以下偏移量 500 为文件的 RBA 信息, 修改该信息与 SYSTEM 表空间一致:

```
BBED> modify /x d0 file 7 block 1 offset 500
```

```
File: /u01/app/oracle/oradata/ORCL/eygle01.dbf (7)
```

```
Block: 1           Offsets: 500 to 507           DbA:0x01c00001
```

```
-----  
d0000000 74030000
```

```
<32 bytes per line>
```

```
BBED> modify /x 7403 file 7 block 1 offset 504
```

```
File: /u01/app/oracle/oradata/ORCL/eygle01.dbf (7)
```

```
Block: 1           Offsets: 504 to 511           DbA:0x01c00001
```

```
-----  
74030000 1000fab8
```

```
<32 bytes per line>
```

修改完成之后, 可以通过 SUM 重新计算并应用新的校验值, 否则数据块会被标记为损坏:

```
BBED> sum apply file 7
```

```
Check value for File 7, Block 1:
```

```
current = 0x6df9, required = 0x6df9
```

```
BBED> verify file 7
```

```
DBVERIFY - Verification starting
FILE = /u01/app/oracle/oradata/ORCL/eygle01.dbf
```

```
DBVERIFY - Verification complete
```

```
Total Blocks Examined          : 1280
Total Blocks Processed (Data)  : 697
Total Blocks Failing   (Data) : 0
Total Blocks Processed (Index): 0
Total Blocks Failing   (Index): 0
Total Blocks Empty        : 575
Total Blocks Marked Corrupt : 0
Total Blocks Influx       : 0
```

最后通过简单的 Recover 即可将文件 Online 加载:

```
[oracle@hpserver2 lib]$ sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.4.0 - Production on Mon Jan 16 17:33:28 2012
Copyright (c) 1982, 2007, Oracle. All Rights Reserved.
```

Connected to:

Oracle Database 10g Enterprise Edition Release 10.2.0.4.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

```
SQL> recover datafile 7;
```

```
Media recovery complete.
```

```
SQL> alter database datafile 7 online;
```

```
Database altered.
```

这也是这类故障恢复的重要方法之一。

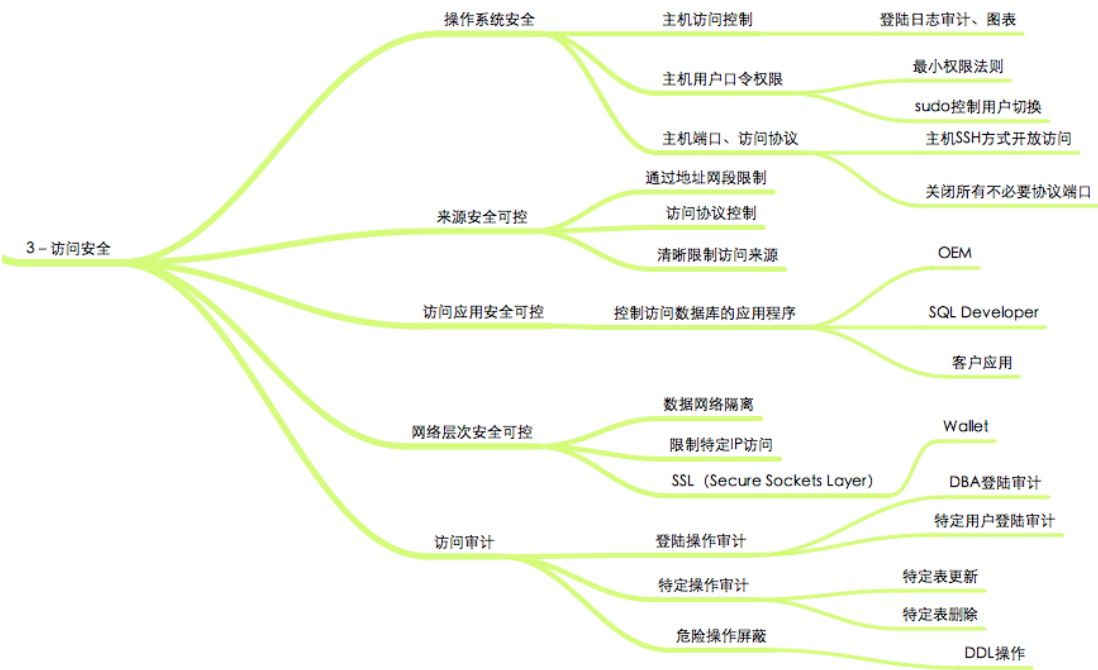
心存目想，三思后行

杨君緘书贲图请予为记。予按图握笔，心存目想，覩缕梗概，十不得其二三。

——白居易《白苹洲五亭记》

季文子三思而后行。子闻之曰：“再，斯可矣。”

——《论语·公冶长》



我们遇到的很多数据灾难，其产生过程往往就在一念之间，有时候眼睛看到的，未作认真思考，下意识的一个动作就使数据库陷入苦难。

所以当我们进行数据库的维护工作时，应当将目之所见，经过心之所想，再转化为手之所动，三思而后行，这样才能够减少灾难的发生。

本章涉及的案例与访问安全有关，随意的、无规范的管理常常导致数据灾难，很多用户在图形工具远程管理之下，严重破坏了访问安全守则，PL/SQL Developer 或 OEM 在很多情况下导致了严重的数据库故障，简便易用必然导致安全的牺牲。

篇首图对访问安全的诸多方面进行了分析，从操作系统安全，到应用访问来源等等，通常我们应当能够连接数据库的程序，很多工具是危险的：

控制访问安全，是实现数据安全的核心环节。

Truncate 导致的灾难

核心字典表误操作 TRUNCATE

灾难描述

以下这则案例就发生在一念之间，手之所动未经心之所想。

1. 客户 DBA 使用 PL/SQL Developer 之类工具管理数据库
2. 发现几个表占用的空间很大
3. 这几个表不是业务数据表
4. DBA 随手 Truncate 截断了这几个数据表
5. 数据库告警报错，运行不再正常
6. 检查发现这几张表是数据库的字典表
7. 灾难形成

回顾这个灾难的形成过程，我们可以想象，如果 DBA 经过多几分钟的思考，这个故障就不会发生，一念之间，天差地别。

案例警示

这个案例给我们的警示是：

1. 任何业务数据库的操作都不能草率

这个数据库是非常重要的一个业务数据库，DBA 这样的操作是非常草率的，我们警示任何数据库的维护人员，在接触数据时都不能掉以轻心，草率行事。

在权限管理上，应当做到权限分离，作为业务维护人员不应当授予 DBA 权限，而 DBA 在进行非维护操作时，也无需以 SYSDBA 角色进行登陆。

DBA 常规的维护工作可以用 SYSTEM 账户登陆数据库，这个账户会屏蔽掉一些危险的操作，防止破坏性

灾难的发生。

2. 眼之所见必须要经过心之所想才能转换为手之所做

这次故障实际上是一时轻率酿成的，眼之所见未经过用心思考，就转换为手之动作，动作一发出可能 DBA 就已经后悔了，但是 DDL 语句无法回退，没有退路可选。

所以我们警示 DBA，在执行任何维护操作时都应当三思后行，慎之再慎。

3. 数据库维护操作尽量以命令行方式完成

对于数据库的维护操作，应当在数据库本地，以命令行方式完成，远程的图形终端操作是不可靠、不安全的，应当严格杜绝远程的维护操作。

图形界面中的一个功能描述，可能很多 DBA 无法直观的将其对应到后台命令上去，这就导致了麻痹疏忽，如果主动发出 Truncate 的命令，我相信绝大多数 DBA 都会多想一想。

通过触发器可以实现 DDL 本地化限制，这可以参考第二章中的内容。

4. 养成文档和操作记录的习惯

在进行数据库维护操作工作时，应当进行事前的文档记录和准备工作，然后再具体执行，即便是简单的工作，用铅笔在纸上做一些条目梳理也有助于帮助我们理清思路、延缓操作、减少失误。

5. 任何关键性的修改应两个人以上确认

任何人都有思维不清晰的时刻，而两个人同时犯一个低级错误的可能性就要小得多。因此建议执行关键性操作或执行有潜在风险的操作之前，一定要找同事一起确认脚本或命令，这样可以最大程度降低由于头脑一时不清所引发的灾难。如果身边没有熟悉数据库的人员，可以常识将自己的步骤和操作讲给其他人听，如果你的步骤中存在明显的错误，很可能在你讲解的过程中自己就发现了。

规范、计划、记录、思考，这些习惯在 DBA 的工作中应当时刻记在心头。

技术回放

这个案例可以概括为由于 DBA 的轻率误操作，导致部分数据字典信息丢失，影响了数据库的稳定性和一致性。

IDL_CHAR\$等字典被截断

在现场经过检查确定，这些被 Truncate 的表主要有：

```
SQL> select object_name,object_type from dba_objects where object_name like 'IDL%';
```

OBJECT_NAME	OBJECT_TYPE
-----	-----
IDL_CHAR\$	TABLE
IDL_SB4\$	TABLE
IDL_UB1\$	TABLE
IDL_UB2\$	TABLE

这些字典表用于存放 Oracle 数据库的存储过程、触发器、Package 编译后的执行信息，缺少这些信息，数据库的任何 DML 操作都无法进行。对于一个存在大量过程、触发器等数据库来说，这些字典表的存储可能越来越大，在有些用户的数据库中，这些字典表可能占用几个 GB 到 10 几个 GB 的容量。

这几个字典表的定义可以从 sql.bsq 及相关文件中找到：

```
create table idl_ub1$                                /* idl table for ub1 pieces */
( obj#          number not null,                      /* object number */
  part          number not null,
                /* part: 0 = diana, 1 = portable pcode, 2 = machine-dependent pcode */
  version       number,                               /* version number */
  piece#        number not null,                      /* piece number */
  length        number not null,                      /* piece length */
  piece         long raw not null)                   /* ub1 piece */
  storage (initial 10k next 100k maxextents unlimited pctincrease 0)
/

create table idl_char$                              /* idl table for char pieces */
( obj#          number not null,                      /* object number */
  part          number not null,
                /* part: 0 = diana, 1 = portable pcode, 2 = machine-dependent pcode */
  version       number,                               /* version number */
  piece#        number not null,                      /* piece number */
  length        number not null,                      /* piece length */
  piece         long not null)                       /* char piece */
```

```

storage (initial 10k next 100k maxextents unlimited pctincrease 0)
/
create table idl_ub2$                                /* idl table for ub2 pieces */
( obj#          number not null,                      /* object number */
  part          number not null,
                /* part: 0 = diana, 1 = portable pcode, 2 = machine-dependent pcode */
  version       number,                               /* version number */
  piece#        number not null,                     /* piece number */
  length        number not null,                     /* piece length */
  piece         long ub2 not null)                   /* ub2 piece */
storage (initial 10k next 100k maxextents unlimited pctincrease 0)
/
create table idl_sb4$                                /* idl table for sb4 pieces */
( obj#          number not null,                      /* object number */
  part          number not null,
                /* part: 0 = diana, 1 = portable pcode, 2 = machine-dependent pcode */
  version       number,                               /* version number */
  piece#        number not null,                     /* piece number */
  length        number not null,                     /* piece length */
  piece         long sb4 not null)                   /* sb4 piece */
storage (initial 10k next 100k maxextents unlimited pctincrease 0)
/

```

这几个表被 Truncate 之后又生成了少量记录，这些信息来自于用户的一些编译尝试：

```

SQL> select count(*) from idl_ub1$;
COUNT ( * )
-----
33

SQL> select count(*) from idl_ub2$;
COUNT ( * )
-----
66

SQL> select count(*) from idl_sb4$;
COUNT ( * )

```

66

```
SQL> select count(*) from idl_char$;
```

```
COUNT(*)
```

33

ORA-600 17069 导致故障

此时数据库出现的典型错误是 ORA-600 错误:

```
Mon Jan 21 16:10:07 2008
```

```
Errors in file /ora_arch/admin/bdump/bgcw2_j000_20123.trc:
```

```
ORA-00600: internal error code, arguments: [17069], [0xC0000000B2E4A750], [], [], [],
[], [], []
```

```
Mon Jan 21 16:10:08 2008
```

```
Errors in file /ora_arch/admin/bdump/bgcw2_j000_20123.trc:
```

```
ORA-00600: internal error code, arguments: [17069], [0xC0000000B2E4A750], [], [], [],
[], [], []
```

```
Mon Jan 21 16:10:08 2008
```

```
Trace dumping is performing id=[cdmp_20080121161008]
```

```
Mon Jan 21 16:11:05 2008
```

```
Errors in file /ora_arch/admin/bdump/bgcw2_j001_20125.trc:
```

```
ORA-00600: internal error code, arguments: [17069], [0xC0000000B2E4A750], [], [], [],
[], [], []
```

由于这一故障的出现, 数据库无法正常使用, 所有的过程调用都会出现 ORA-600 错误。

ORA-600 17069 错误, 其含义是进程无法 Pin 锁定某个 library cache object, 这个问题通常与过程的失效或删除有关, 通常通过重新编译过程等对象就可以解决。

只不过对于这个案例, 是用户手工截断了相关的字典表, 一些尝试编译工作也失败了。进程跟踪文件如下:

```
*** SESSION ID:(36.7) 2008-01-20 12:32:27.357
```

```
*** 2008-01-20 12:32:27.357
```

```
ksedmp: internal or fatal error
```

ORA-00600: internal error code, arguments: [17069], [0xC0000000DDDFa690], [], [], [],
[], [], []

No current SQL statement being executed.

----- Call Stack Trace -----

calling location	call type	entry point	argument values in hex (? means dubious value)
ksedmp()+528	call	_etext_f()+23058430 09102745972	000000000 ? C000000000000996 ? 4000000002801BE0 ?
ksfdmp()+64	call	_etext_f()+23058430 09102745972	000000003 ?
kgeriv()+432	call	_etext_f()+23058430 09102745972	600000000004A0F0 ? 000000003 ? C000000000000716 ? 40000000052BAFB0 ? FF9C00000001802F ? 60000000004DCC48 ? 000000000 ? 000000000 ?

进程状态信息显示该进程正在等待 Library cache pin,请求对象的地址为 c0000000dcd182e0:

S0: c0000000d62e1730, type: 4, owner: c0000000d6298f90, flag: INIT/-/-/0x00
(session) trans: 0000000000000000, creator: c0000000d6298f90, flag: (8000041)
USR/- BSY/-/-/-/-/-
DID: 0001-0010-00000030, short-term DID: 0000-0000-00000000
txn branch: 0000000000000000
oct: 0, prv: 0, sql: c0000000ddcb7398, psql: 0000000000000000, user:
39/FMIS_SHARE
O/S info: user: , term: , ospid: , machine:
program:
last wait for 'library cache pin' blocking sess=0x0 seq=53 wait_time=44
handle address=c0000000dddfa690, pin address=c0000000dcd182e0,
100*mode+namespace=12d

```
temporary object counter: 0
```

```
-----
```

在进程的授权队列上，我们可以找到详细的依赖信息，以下是进程的主要信息：

```
-----process-----
```

```
proc version      : 3
Local node        : 0
gid               : 0
pid               : 5675
lkp_node          : 0
proc state        : KJP_NORMAL
Total accesses    : 684
Imm.  accesses    : 683
Locks on ASTQ     : 0
Locks Pending AST : 0
Granted locks     : 1
AST_Q             : NULL
PENDING_Q         : NULL
GRANTED_Q         : 0xc0000000d68e75b8
```

```
AST_Q:
```

```
PENDING_Q:
```

```
GRANTED_Q:
```

```
lp 0xc0000000d68e75b8 gl KJUSERPR rp 0xc0000000d680b3c0 master 0 pid 5675
```

```
bast 0 rseq 0 history 0x14351435 open opt KJUSERDEADLOCK KJUSERPROCESS_OWNED
```

接下来的依赖关系显示 c0000000dcd182e0 请求 Pin 的对象是 SYS.STANDARD 标准包，这是数据库最核心的一个系统包，该 Package 失效，大量依赖它的对象全部失效，系统的症结首先体现在这里：

```

-----
SO: c0000000d633e198, type: 3, owner: c0000000d6298f90, flag: INIT/-/-/0x00
(call) sess: cur c0000000d62e1730, rec c0000000d62e1730, usr c0000000d62e1730; depth: 0
-----
SO: c0000000d633c8a8, type: 3, owner: c0000000d633e198, flag: INIT/-/-/0x00
(call) sess: cur c0000000d62e1730, rec 0, usr c0000000d62e1730; depth: 1
-----
SO: c0000000dcd182e0 type: 52, owner: c0000000d633c8a8, flag: INIT/-/-/0x00
LIBRARY OBJECT PIN: pin=c0000000dcd182e0 handle=c0000000dddfa690 mode=S lock=c0000000dcd17550
user=c0000000d62e1730 session=c0000000d62e1730 count=1 mask=001d savepoint=230 flags=[00]
-----
SO: c0000000dcd17550, type: 51, owner: c0000000d633c8a8, flag: INIT/-/-/0x00
LIBRARY OBJECT LOCK: lock=c0000000dcd17550 handle=c0000000dddfa690 mode=S
call pin=c0000000dcd182e0 session pin=0000000000000000
htl=c0000000dcd175e0[c0000000dcd79a88, c0000000dcd79a88] htb=c0000000dcd79a88
user=c0000000d62e1730 session=c0000000d62e1730 count=1 flags=PNC/[04] savepoint=230
LIBRARY OBJECT HANDLE: handle=c0000000dddfa690
name=SYS_STANDARD
hash=bf7b5f21 timestamp=11-19-2001 00:00:00
namespace=TABLE/PRCD/TYP flags=KGHP/TIM/SML/[02000000]
kkkk- dddd-1111=0000-001d-001d lock=S pin=S latch#=1
lwt=c0000000dddfa6c0[c0000000dddfa6c0, c0000000dddfa6c0] ltm=c0000000dddfa6d0[c0000000dddfa6d0, c0000000dddfa6d0]
pwt=c0000000dddfa6f0[c0000000dddfa6f0, c0000000dddfa6f0] ptm=c0000000dddfa780[c0000000dddfa780, c0000000dddfa780]
ref=c0000000dddfa6a0[c0000000dddfa6a0, c0000000dddfa6a0] lnd=c0000000dddfa798[c0000000dddfa798, c0000000dddfa798]
LOCK INSTANCE LOCK: id=LB238ccb5b1667ba4a
PIN INSTANCE LOCK: id=NB238ccb5b1667ba4a mode=S release=F flags=[00]
INVALIDATION INSTANCE LOCK: id=IV0000028713010101 mode=S
LIBRARY OBJECT: object=c0000000ddca9ef0
type=PCPG flags=EXS/LDC[0005] pflags=NST/IVR [201] status=VALD load=0
DATA BLOCKS:
data# heap pointer status pins change
-----
0 c0000000ddcaa208 c0000000ddca9c00 I/P/A 0 NONE
2 c0000000ddcaa010 0 -/P/- 1 NONE
3 c0000000ddcaa0c0 0 -/P/- 1 NONE
4 c0000000ddca9b78 0 -/P/- 1 NONE
-----

```

恢复过程

解决这个问题，我们首先想到的思路是：

主要损坏的字典表存储的信息来自对象编译过程，那么通过重新编译所有对象可以重新生成这些数据，从而使数据库恢复正常运行，基于这一考虑，我们可以将主要对象的创建脚本进行再次运行，重新编译数据库相关的 Procedure/Trigger/Package/Package Body 等，从而解决这一故障。

这是一个 RAC 集群数据库，首先应当设置 cluster_database 参数为 false：

```
SQL> alter system set cluster_database=false scope=spfile sid='bgcw2';
```

```
System altered.
```

```
SQL> shutdown immediate;
```

```
Database closed.
```

```
Database dismounted.
```

```
ORACLE instance shut down.
```

然后将数据库启动在 Migrate 模式执行编译：

```
SQL> startup migrate
ORACLE instance started.

Total System Global Area 2422165840 bytes
Fixed Size                  729424 bytes
Variable Size               738197504 bytes
Database Buffers            1677721600 bytes
Redo Buffers                 5517312 bytes
Database mounted.
Database opened.
SQL> @?/rdbms/admin/catalog
SQL> @?/rdbms/admin/catproc
SQL> @?/rdbms/admin/utlrp
```

具体执行过程如下，首先执行 catalog.sql 脚本：

```
SQL> @?/rdbms/admin/catalog
DOC>#####
DOC>#####
DOC>    The following statement will cause an "ORA-01722: invalid number"
DOC>    error and terminate the SQLPLUS session if the user is not SYS.
DOC>    Disconnect and reconnect with AS SYSDBA.
DOC>#####
DOC>#####
DOC>#

no rows selected

Package created.

Package body created.

Grant succeeded.
```

Package created.

Synonym created.

Grant succeeded.

Table created.

.....,

然后可以执行 catproc.sql 脚本:

```
SQL> @?/rdbms/admin/catproc
```

```
DOC>#####
```

```
DOC>#####
```

```
DOC>    The following PL/SQL block will cause an ORA-20000 error and
```

```
DOC>    terminate the current SQLPLUS session if the user is not SYS.
```

```
DOC>    Disconnect and reconnect with AS SYSDBA.
```

```
DOC>#####
```

```
DOC>#####
```

```
DOC>#
```

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

View created.

View created.

Comment created.

Comment created.

Comment created.

Comment created.

Comment created.

Comment created.

Comment created.

Synonym created.

Grant succeeded.

View created.

Comment created.

.....

完成这两个步骤之后，可以通过运行 utlrp.sql 脚本重新编译失效对象完成恢复。

恢复完成之后，所有相关数据被成功恢复过来：

```
SQL> select count(*) from idl_ub1$;
```

```
COUNT(*)
```

```
-----
```

```
50947
```

```
SQL> select count(*) from idl_ub2$;
```

```
COUNT(*)
```

```
-----
```

```
50719
```

```
SQL> select count(*) from idl_sb4$;
```

```
COUNT(*)
```

```
-----
```

```
61987
```

```
SQL> select count(*) from idl_char$;
```

```
COUNT(*)
-----
24804
```

检查数据库的运行日志报告，其中信息输出正常，RAC Cluster 能够正常构建，数据库恢复了正常运行：

```
Mon Jan 21 21:17:58 2008
Successful mount of redo thread 2, with mount id 2791344706
Mon Jan 21 21:17:58 2008
Database mounted in Shared Mode (CLUSTER_DATABASE=TRUE).
Completed: ALTER DATABASE MOUNT
Mon Jan 21 21:17:58 2008
ALTER DATABASE OPEN
This instance was first to open
Picked Lamport scheme to generate SCNs
Mon Jan 21 21:17:58 2008
Thread 2 opened at log sequence 29826
Current log# 6 seq# 29826 mem# 0: /dev/vgoracle/rora_redo23a
Successful open of redo thread 2
Mon Jan 21 21:17:58 2008
SMON: enabling cache recovery
Instance recovery: looking for dead threads
Instance recovery: lock domain invalid but no dead threads
Mon Jan 21 21:17:59 2008
Successfully onlined Undo Tablespace 6.
Mon Jan 21 21:17:59 2008
SMON: enabling tx recovery
Mon Jan 21 21:17:59 2008
Database Characterset is ZHS16GBK
Mon Jan 21 21:18:28 2008
replication_dependency_tracking turned off (no async multimaster replication found)
Completed: ALTER DATABASE OPEN
```

新的节点加入，RAC 环境重新配置：

```
Mon Jan 21 21:19:29 2008
```

```

Reconfiguration started (old inc 1, new inc 2)
List of nodes:
0 1
Global Resource Directory frozen
Communication channels reestablished
Master broadcasted resource hash value bitmaps
Non-local Process blocks cleaned out
Resources and enqueues cleaned out
Resources remastered 1517
191750 GCS shadows traversed, 0 cancelled, 658 closed
191092 GCS resources traversed, 0 cancelled
126678 GCS resources on freelist, 210430 on array, 210430 allocated
set master node info
Submitted all remote-enqueue requests
Update rdomain variables
Dwn-cvts replayed, VALBLKS dubious
All grantable enqueues granted
191750 GCS shadows traversed, 107340 replayed, 658 unopened
Submitted all GCS remote-cache requests
0 write requests issued in 83752 GCS resources
0 PIs marked suspect, 0 flush PI msgs
Mon Jan 21 21:19:31 2008
Reconfiguration complete
Mon Jan 21 21:19:31 2008
Instance recovery: looking for dead threads
Instance recovery: lock domain invalid but no dead threads
Mon Jan 21 21:26:30 2008
Thread 2 advanced to log sequence 29827
    Current log# 4 seq# 29827 mem# 0: /dev/vgoracle/rora_redo21a

```

经业务系统正常运行测试通过，数据库恢复正常，故障处理完成。

脚本错误导致的灾难

数据库整体被删除故障

以下这则灾难初始是由于脚本问题，后续的发展则是由于工程师的判断失误，导致事件逐步恶化，最后不可收拾。

灾难描述

这次灾难的出现过程如下：

1. 客户的备份和数据库位于同一主机的不同卷
2. 备份策略是每日执行备份，压缩传递至远程后，删除本地备份
3. 某日脚本出错，带入的环境变量不是备份目录，而是主机数据库目录
4. 再次误执行脚本，本地备份也被删除
5. 数据库所有文件及本地备份被删除
6. DBA 尝试传输远程到本地恢复
7. 当解压后恢复时，发现备份的文件不足
8. 灾难形成

在这些处理过程中，DBA 为了遮掩自己的失误，一直试图弱化故障的严重程度，直至最后发现备份不足以用来恢复，才将事情如实向上汇报，但是这已经造成了不可挽回的损失。

特别是当本地目录已经被解压覆盖，使得基于文件系统的恢复也不再可能。

案例警示

这个案例与之前很多案例具有类似之处，部分警示前面已经描述过，此处不再赘述。有一些新的警示是：

1. 解决问题最好的办法是诚实，不要隐瞒问题

当系统因为种种原因遭遇故障后，最好的解决途径是通过集体的力量，个人在极端情况下，思维容易走入

死胡同，不断做出错误的判断。

所以，当出现故障或误操作时，最重要的是不要隐瞒问题，以解决问题为核心，发动团队的智慧。另外，向上级诚实反应情况，也有助于更准确的判断问题的严重程度，以及对外的应对方式。隐瞒和草率的掩盖往往使事情变得更加糟糕。

在这个案例中,DBA 在原目录下解压缩备份文件,导致磁盘存储被覆盖,否则原有文件是可以恢复出来的。

2. 在出现问题时，不要急于做出第一个判断

最为初始的判断会决定后续恢复的走向，如果处置不当，则可能使问题恶化。如果是 DBA 个人的判断，很有可能走向草率的极端，错误叠加错误，所以当问题出现时，不要急于做第一个判断和举措，除非具有十足的把握。

依赖团队和专业人士，有时候可能一个电话就可以解决问题。对于这个案例，也许从文件描述符就可以恢复问题，至不济还可以从磁盘恢复，但是最后覆盖的结果是，数据彻底损失。

这个案例的数据库是一个公司几百员工，几年的工作积累，如果丢失，则损失极为惨痛。

技术回放

在数据库文件被删除之后，告警日志中首先出现错误：

```
Wed Apr 7 11:39:46 2010
Thread 1 advanced to log sequence 20265
Current log# 7 seq# 20265 mem# 0: /opt/oracle/oradata/MEDIA/redo07.log
Wed Apr 7 11:39:46 2010
ARC0: Beginning to archive log# 6 seq# 20264
ARC0: Completed archiving log# 6 seq# 20264
Wed Apr 7 12:24:24 2010
Errors in file /opt/oracle/admin/MEDIA/udump/media_ora_12238.trc:
ORA-01115: IO error reading block from file 12 (block # 901174)
ORA-01110: data file 12: '/opt/oracle/oradata/MEDIA/media_04.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
```

```

ORA-01115: IO error reading block from file 12 (block # 856947)
ORA-01110: data file 12: '/opt/oracle/oradata/MEDIA/media_04.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3
Wed Apr  7 12:25:19 2010
Errors in file /opt/oracle/admin/MEDIA/bdump/media_pmon_5364.trc:
ORA-00604: error occurred at recursive SQL level 1
ORA-01115: IO error reading block from file 1 (block # 89)
ORA-01110: data file 1: '/opt/oracle/oradata/MEDIA/system01.dbf'
ORA-27041: unable to open file
SVR4 Error: 2: No such file or directory
Additional information: 3

```

注意以上信息，SYSTEM 表空间文件也不可访问，这是一次非常彻底的批量误删除操作。

恢复过程

经过分析备份失败的原因，我们发现实质上是由于本地空间不足，在压缩时 SYSTEM 表空间等文件压缩失败，也就没有传送到远程备份主机，而另外还有 3 个数据文件没有备份，遗漏掉了。

另外一点幸运的是，备份卷只是删除了备份文件，并没有被重写覆盖，所以我们首先从备份卷上恢复除了 SYSTEM 表空间，还后利用不完全的备份，结合起来，就只缺少 4 个数据文件，而其中的三个数据文件，是一个表空间的 1, 2, 3 号文件，该表空间共有 19 个数据文件，这几个缺失的文件之前的已经基本写满，19 号文件也从删除中恢复了出来；现在我们就利用这几个历史文件备份、恢复出来的新的文件、以及最近的成功备份，整合起来，通过文件修复、同步，将数据库成功打开。

以下是部分数据文件，来自 2008 年的备份，以及从磁盘恢复出来最新的 SYSTEM 表空间和第 19 号媒体文件：

```

oracle@v880 ~/product/8.1.7.4/dbs$ls -al /opt/oracle/oradata/MEDIA/
total 76024148
drwxr-xr-x  2 oracle  dba          512 Apr 10 18:17 .
drwxr-xr-x  7 oracle  dba          512 Apr 10 19:33 ..
-rw-r----- 1 oracle  dba    16031744 Apr 10 18:21 control01.ctl

```

```

-rw-r----- 1 oracle dba      16031744 Apr 10 18:21 control02.ctl
-rw-r----- 1 oracle dba      8588976128 Sep 24 2008 media_01.dbf
-rw-r----- 1 oracle dba      8587608064 Sep 24 2008 media_02.dbf
-rw-r----- 1 oracle dba      8510480384 Sep 24 2008 media_03.dbf
-rw-r----- 1 oracle dba      5028233216 Sep 24 2008 media_07.dbf
-rw-r----- 1 oracle dba      7759470592 Apr  6 05:45 media_19.dbf
-rw-r----- 1 oracle dba      398467072 Apr  6 06:33 system01.dbf

```

在以下目录中，找到了最近的其他备份文件：

```

oracle@v880 ~/oradata/data_2010_04_02$ls -l
total 190233104
-rw-r----- 1 oracle dba      1048584192 Apr  2 06:14 PEPSI_01.dbf
-rw-r----- 1 oracle dba      1048584192 Apr  2 06:15 PEPSI_02.dbf
-rw-r----- 1 oracle dba      16031744 Apr 10 13:26 control01.ctl
-rw-r----- 1 oracle dba      8482045952 Apr  2 03:02 media_04.dbf
-rw-r----- 1 oracle dba      8587509760 Apr  2 03:15 media_05.dbf
-rw-r----- 1 oracle dba      7961485312 Apr  2 03:28 media_06.dbf
-rw-r----- 1 oracle dba      5956255744 Apr  2 03:41 media_08.dbf
-rw-r----- 1 oracle dba      5515517952 Apr  2 03:51 media_09.dbf
-rw-r----- 1 oracle dba      5614084096 Apr  2 03:59 media_10.dbf
-rw-r----- 1 oracle dba      6912811008 Apr  2 04:08 media_11.dbf
-rw-r----- 1 oracle dba      6736060416 Apr  2 04:19 media_12.dbf
-rw-r----- 1 oracle dba      7914659840 Apr  2 04:30 media_13.dbf
-rw-r----- 1 oracle dba      6316630016 Apr  2 04:44 media_14.dbf
-rw-r----- 1 oracle dba      7050633216 Apr  2 04:55 media_15.dbf
-rw-r----- 1 oracle dba      6970941440 Apr  2 05:07 media_16.dbf
-rw-r----- 1 oracle dba      4823457792 Apr  2 05:19 media_17.dbf
-rw-r----- 1 oracle dba      6396321792 Apr  2 05:27 media_18.dbf

```

所以现在我们拥有的是来自三个时期的文件。

经过检查，恢复出来的 SYSTEM 文件是完好的，这就为恢复提供了第一个可能性：

```

oracle@v880 ~/oradata/MEDIA$dbv file=system01.dbf blocksize=8192
DBVERIFY: Release 8.1.7.4.0 - Production on Sat Apr 10 20:04:03 2010
(c) Copyright 2000 Oracle Corporation. All rights reserved.

```

```
DBVERIFY - Verification starting : FILE = system01.dbf
```

```
DBVERIFY - Verification complete
```

```
Total Pages Examined          : 48640
```

```
Total Pages Processed (Data) : 26289
```

```
Total Pages Failing   (Data) : 0
```

```
Total Pages Processed (Index): 4211
```

```
Total Pages Failing   (Index): 0
```

```
Total Pages Processed (Other): 15284
```

```
Total Pages Empty          : 2856
```

```
Total Pages Marked Corrupt  : 0
```

```
Total Pages Influx          : 0
```

为了简化恢复，抛弃了一些不必要的数据库文件：

```
SQL> archive log list;
```

```
Database log mode          Archive Mode
```

```
Automatic archival        Enabled
```

```
Archive destination       /opt/oracle/admin/MEDIA/arch_opt
```

```
Oldest online log sequence 19922
```

```
Next log sequence to archive 19925
```

```
Current log sequence       19925
```

```
SQL> alter database datafile '/opt/oracle/oradata/MEDIA/indx01.dbf' offline;
```

```
Database altered.
```

```
SQL> alter database datafile '/opt/oracle/oradata/MEDIA/drsys01.dbf' offline;
```

```
Database altered.
```

在不完全恢复尝试打开时，数据库会提示，这些文件在 Resetlogs 之后会被丢弃，我们先手工将文件 DROP：

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

```
alter database open resetlogs
```

```
*
```

```
ERROR at line 1:
```

```
ORA-01245: offline file 6 will be lost if RESETLOGS is done
```

```
ORA-01110: data file 6: '/opt/oracle/oradata/MEDIA/indx01.dbf'
```

```
SQL> alter database datafile '/opt/oracle/oradata/MEDIA/indx01.dbf' offline drop;
Database altered.
```

对于从历史备份中恢复的文件，可以通过 BBED 从其他文件的 SCN 检查点进行同步，强制修正一致性，这可以参考上一章的做法。

由于控制文件和部分数据文件不符，所以尝试用较旧的控制文件是无法打开数据库的，数据库提示文件较控制文件新，这需要重建控制文件来解决：

```
SQL> alter database open resetlogs;
alter database open resetlogs
*
ERROR at line 1:
ORA-01248: file 33 was created in the future of incomplete recovery
ORA-01110: data file 33: '/opt/oracle/oradata/MEDIA/media_16.dbf'
```

重建控制文件之后，强制打开数据库，出现实例崩溃：

```
SQL> alter database open resetlogs;
alter database open resetlogs
*
ERROR at line 1:
ORA-03113: end-of-file on communication channel
```

但是此时告警日志已经显示了熟悉的信息。除了几个抛弃的文件之外，就是 4194 的回滚错误信息：

```
Sat Apr 10 23:33:49 2010
SMON: enabling cache recovery
Sat Apr 10 23:33:49 2010
Dictionary check beginning
Tablespace 'INDX' #5 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'DRSYS' #6 found in data dictionary,
but not in the controlfile. Adding to controlfile.
File #6 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00006' in the controlfile.
This file can no longer be recovered so it must be dropped.
File #7 found in data dictionary but not in controlfile.
```

```

Creating OFFLINE file 'MISSING000007' in the controlfile.
This file can no longer be recovered so it must be dropped.
File #22 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING000022' in the controlfile.
This file can no longer be recovered so it must be dropped.
Dictionary check complete
Sat Apr 10 23:33:49 2010
SMON: enabling tx recovery
Sat Apr 10 23:33:50 2010
Errors in file /opt/oracle/admin/MEDIA/bdump/media_smon_3695.trc:
ORA-00600: internal error code, arguments: [4194], [53], [14], [], [], [], [], []
Recovery of Online Redo Log: Thread 1 Group 7 Seq 1 Reading mem 0
  Mem# 0 errs 0: /opt/oracle/oradata/MEDIA/redo07.log
Sat Apr 10 23:33:51 2010
Errors in file /opt/oracle/admin/MEDIA/bdump/media_smon_3695.trc:
ORA-00600: internal error code, arguments: [4194], [53], [14], [], [], [], [], []
SMON: terminating instance due to error 600
Instance terminated by SMON, pid = 3695

```

这个错误通过重建 Undo 表空间就可以消除，这是一个 Oracle 8i 的数据库，我们在参数文件中设置了如下参数：

```

_allow_resetlogs_corruption=true
_corrupted_rollback_segments='RBS0','RBS1','RBS2','RBS3','RBS4','RBS5','RBS6','RBS7','RBS8','RBS9','RBS10','RBS11','RBS12','RBS13','RBS14','RBS15','RBS16','RBS17','RBS18','RBS19','RBS20','RBS21','RBS22','RBS23','RBS24','RBS25','RBS26','RBS27','RBS28','RBS29'
_offline_rollback_segments='RBS0','RBS1','RBS2','RBS3','RBS4','RBS5','RBS6','RBS7','RBS8','RBS9','RBS10','RBS11','RBS12','RBS13','RBS14','RBS15','RBS16','RBS17','RBS18','RBS19','RBS20','RBS21','RBS22','RBS23','RBS24','RBS25','RBS26','RBS27','RBS28','RBS29'

```

然后数据库久违的可以重新启动：

```

oracle@v880 ~/admin/MEDIA/bdump$ sqlplus "/ as sysdba"
SQL*Plus: Release 8.1.7.0.0 - Production on Sat Apr 10 23:38:16 2010
(c) Copyright 2000 Oracle Corporation. All rights reserved.

```

Connected to an idle instance.

SQL> startup

ORACLE instance started.

Total System Global Area 9873338044 bytes

Fixed Size 102076 bytes

Variable Size 915955712 bytes

Database Buffers 8955625472 bytes

Redo Buffers 1654784 bytes

Database mounted.

Database opened.

SQL> select name from v\$datafile;

NAME

/opt/oracle/oradata/MEDIA/system01.dbf
/opt/oracle/oradata/MEDIA/tools01.dbf
/opt/oracle/oradata/MEDIA/rbs01.dbf
/opt/oracle/oradata/MEDIA/temp01.dbf
/opt/oracle/oradata/MEDIA/users01.dbf
/opt/oracle/product/8.1.7.4/dbs/MISSING000006
/opt/oracle/product/8.1.7.4/dbs/MISSING000007
/opt/oracle/oradata/MEDIA/media_01.dbf
/opt/oracle/oradata/MEDIA/media_idx_01.dbf
/opt/oracle/oradata/MEDIA/media_02.dbf
/opt/oracle/oradata/MEDIA/media_03.dbf
.....

然后删除重建回滚表空间:

SQL> drop tablespace RBS including contents;

Tablespace dropped.

SQL> CREATE TABLESPACE rbs

2 DATAFILE '/opt/oracle/oradata/MEDIA/rbs201.dbf' SIZE 500M REUSE AUTOEXTEND ON

3 NEXT 100M MAXSIZE 4096M;

Tablespace created.

再然后数据库可以被成功打开：

```
SQL> startup
ORACLE instance started.

Total System Global Area 9873338044 bytes
Fixed Size                  102076 bytes
Variable Size              915955712 bytes
Database Buffers          8955625472 bytes
Redo Buffers               1654784 bytes
Database mounted.
Database opened.
SQL> select owner,count(*) from dba_tables group by owner;
OWNER                                COUNT(*)
-----
AURORA$$JIS$UTILITY$                14
CTXSYS                              31
MDSYS                               15
MED                                  4
MEDIA                              391
MID                                  1
ORDSYS                               9
OSE$HTTP$ADMIN                       3
OUTLN                                2
PEPSI                               19
PERFSTAT                             28
SYS                                 203
SYSTEM                               52

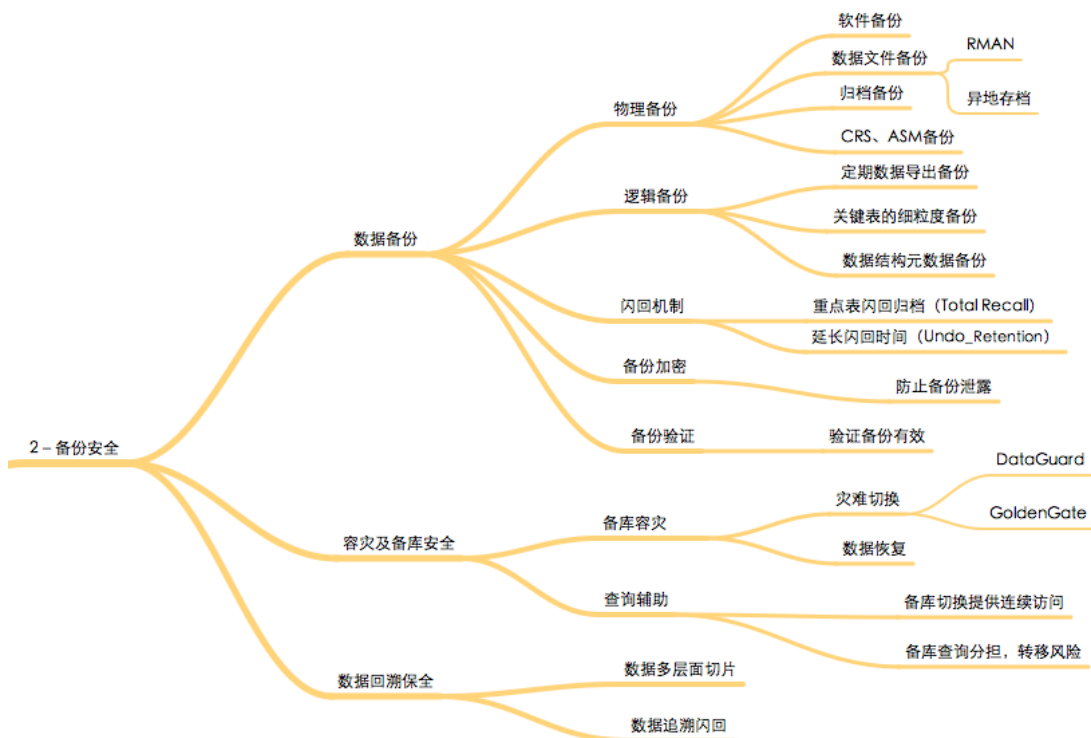
13 rows selected.
```

在这种情况下，我们建议用户通过导出数据，重建数据库。

千里之堤，溃于蚁穴

千丈之堤，溃于蚁穴，以蝼蚁之穴溃；百尺之室，以突隙之烟焚。

——先秦·韩非《韩非子·喻老》



有时候，防守坚固的数据库，可能因为一些微小的误操作、误处理，就走上了毁灭之路，这期间，技术往往是一个次要的环节，缺少规范、管理不善、盲目试错，最终导致了灾难的不可挽回。

本篇内容涉及管理安全方面的变更与上线管理，通过明确的变更规范和流程规范，本篇的几个数据灾难就可以避免：

以下的两个案例都是因为微小失误导致的多米诺效应，这些微小的失误最终使数据库陷入灾难。

一个字符引发的灾难

大小写字符疏忽导致的维护故障

以下是一个字符引发的灾难，不要小看任何一个字符：

灾难描述

用户是这样描述这次灾难的：

1. 进行删除表空间重建的维护，删除某表空间
2. 重建表空间后，发现创建有问题，再次尝试删除
3. 发现无法删除，提示表空间不存在，再创建提示裸设备已经使用
4. 离线、在线操作表空间数据库没问题
5. 多次操作后，出现 ORA-600 错误
6. 数据库无法正常运行，灾难形成

这个案例在《Oracle DBA 手记 2》中曾经有更为详细的描述，在本书中，我们仅仅作为一个警示收录在这里。

案例警示

经过分析处理后我们了解了案情的来龙去脉，问题的根源在于一个本应该大写的 K，被写成了小写 k，一个字符最后导致了这场数据库的大灾难。

这个案例给我们的警示有：

1. 不以规矩不成方圆，规范是技术的保障与补充。

在进行数据库维护，编写脚本时，要注重规范，统一编码，避免不必要的麻烦和风险，有时候规范比技术能力本身更重要。这些脚本必须经过测试才能够被应用到生产环境，而且，如果是非常重要的维护操作，至少申请一位审核人员对代码进行进一步的检查确认。

有的规范要靠企业和客户，有的规范来自于 DBA 自身，一个 DBA 应该养成自己的良好习惯和规范，避免让自己陷入不可预期的困境。

2. 不要在常规任务中处理超出把握的异常

如果你在工作中遇到了类似的情况，当数据库的表征超出了你的预期，那么最好停下来，仔细检查或咨询他人，避免在生产数据库上进行无把握的试错和尝试。**机会喜欢光顾有准备的头脑，灾难喜欢降临在无把握的尝试。**

不要在常规的工作任务中，处理毫无把握、超出预期的异常，你的尝试很有可能使得情况越来越糟糕。

3. 事先定制明确的回退方案

我们主张在重要的变更之前，准备好回退方案，如果没有，那么至少在你进行无把握的操作前，想一想如果出现不可预期的情况，你应当如何回退。

如果你的脑海中有回退的概念，我想很多灾难就不会发生，或者你根本就不会去走向不可预期。

在这个案例中，不规范的操作是导致灾难的根源。

案情解析

最初收到这个案例之后，我们首先分析告警日志文件，理清事件的来龙去脉。根据用户提供的信息，故障是因为表空间维护导致的，最初出现问题的是一个名称叫做"EMIS116K"表空间。

表空间的创建与删除

在告警日志中，这个表空间最初于 Apr 27 创建：

```
Tue Apr 27 09:05:27 2010
CREATE TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 23M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Tue Apr 27 09:11:05 2010
Completed: CREATE TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
```

这个表空间在 Jun 24 被删除，注意这里用户发出的名称“EMIS116k”，最后是一个小写的字母，这个小写的 k 会被数据库强制转换成大写，所以这个删除命令能够成功执行。但是这一个字母的微小差异，最终造成了数据库无法运行的灾难：

```
Thu Jun 24 16:37:53 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 16:47:48 2010
Deleted file /dev/vg02/rlvdata224_8G
```

再然后，一个新的表空间在该文件上建立，表空间名称与原来“相同”。注意这里的相同是加了引号的，用户在这里指定的表空间名称是"EMIS116k"，这与原来的"EMIS116K"是不同的。大小写在引号之中，会存在不同的含义，引号强制了这个小写字母现在创建出来的表空间名称包含了最后一个小写字母 k：

```
Thu Jun 24 16:49:01 2010
CREATE TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Thu Jun 24 16:53:39 2010
Completed: CREATE TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

随后用户向这个表空间创建对象，无法成功，就试图去删除这个表空间，这里仍然给出的是 EMIS116k，这个拼写经过 Oracle 的强制转换，就变成了 EMIS116K,这个名字的表空间是不存在的，于是出现了 ORA-959 错误：

```
Thu Jun 24 16:57:19 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 16:57:19 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

ORA-959 错误的含义是指定的表空间不存在：

```
ORA-00959 tablespace 'string' does not exist
Cause: A statement specified the name of a tablespace that does not exist.
Action: Enter the name of an existing tablespace. For a list of tablespace names,
query the data dictionary.
```

由于前面的操作报出的提示是表空间不存在，客户认为该表空间没有创建成功，或是已经被删除，于是接下来继续在同样的裸设备上创建新的表空间，这一次的名称是全部大写的"EMIS116K"：

```
Thu Jun 24 19:04:12 2010
CREATE TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

由于上一次创建的表空间仍然存在，在同一个裸设备上创建表空间必然不会成功，这一次出现的是ORA-1537号错误：

```
Thu Jun 24 19:04:12 2010
ORA-1537 signalled during:
CREATE TABLESPACE "EMIS116K" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
...
```

ORA-01537 的错误是指，指定的文件已经是数据库的一部分了，不能再次被使用：

```
ORA-01537 cannot add data file 'string' - file already part of database
Cause: During CREATE or ALTER TABLESPACE, a file being added is already part of the
database.
Action: Use a different file name.
```

客户并没有认识到这样的错误，紧跟着一个创建语句再次被发出，此次的表空间名称又变成了大小写混合的"EMIS116k"：

```
Thu Jun 24 19:04:28 2010
CREATE TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
```

这显然也不会成功，这次的错误提示是ORA-1543：

```
Thu Jun 24 19:04:28 2010
ORA-1543 signalled during:
CREATE TABLESPACE "EMIS116k" DATAFILE '/dev/vg02/rlvdata224_8G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
```

```
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
...
```

ORA-01543 错误是指，要创建的表空间已经存在：

```
ORA-01543 tablespace 'string' already exists
```

Cause: An attempt was made to create a tablespace which already exists.

Action: Use a different name for the new tablespace.

好了，到这里打住。

我在此再次提醒大家的一点是：如果你在工作中遇到了类似的情况，当数据库的表征超出了你的预期，那么最好停下来，仔细检查或咨询他人，避免在生产数据库上进行无把握的试错和尝试。

这次的事情还要继续，接下来两个 DROP 命令被发出，但是 EMIS116K 和 EMIS116k 都不被数据库认可，两次尝试又失败了：

```
Thu Jun 24 19:05:09 2010
DROP TABLESPACE EMIS116K INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:05:09 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116K INCLUDING CONTENTS AND DATAFILES
...
Thu Jun 24 19:05:31 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
```

数据文件的离线与在线

用户开始尝试将该文件离线再 Online，试图通过这样的操作消除这个不明原因的无法 DROP 问题。这个尝试过程又引入了恢复的环节，还好一切顺利，表空间的数据文件能够被重新 Online，这样看起来数据文件是没有问题的：

```
Thu Jun 24 19:20:32 2010
alter database datafile '/dev/vg02/rlvdata224_8G' offline
Thu Jun 24 19:20:33 2010
Completed: alter database datafile '/dev/vg02/rlvdata224_8G' offline
```

```

Thu Jun 24 19:21:02 2010
alter database datafile '/dev/vg02/rlvdata224_8G' online
Thu Jun 24 19:21:02 2010
ORA-1113 signalled during: alter database datafile '/dev/vg02/rlvdata224_8G'
online...
Thu Jun 24 19:21:31 2010
ALTER DATABASE RECOVER datafile '/dev/vg02/rlvdata224_8G'
Thu Jun 24 19:21:31 2010
Media Recovery Start
parallel recovery started with 15 processes
Thu Jun 24 19:21:35 2010
Media Recovery Complete (cwgk1)
Completed: ALTER DATABASE RECOVER datafile '/dev/vg02/rlvdata224_8G'
Thu Jun 24 19:21:45 2010
alter database datafile '/dev/vg02/rlvdata224_8G' online
Thu Jun 24 19:21:45 2010
Completed: alter database datafile '/dev/vg02/rlvdata224_8G' online

```

再回过头来继续尝试 DROP 表空间的操作，在没找到根本原因之前，这样的操作当然又失败了：

```

Thu Jun 24 19:48:18 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:48:18 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...
Thu Jun 24 19:48:54 2010
DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
Thu Jun 24 19:48:54 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k INCLUDING CONTENTS AND DATAFILES
...

```

终于客户发现了大小写的问题，尝试将这个小写的 EMIS116k 更改为大写的 EMIS116K 名称。但是注意，不加入双引号，Oracle 认为这两者是没有区别的：

```

Thu Jun 24 19:52:52 2010
ALTER TABLESPACE EMIS116k RENAME TO EMIS116K

```

```
Thu Jun 24 19:52:52 2010
ORA-710 signalled during: ALTER TABLESPACE EMIS116k RENAME TO EMIS116K
...
```

ORA-710 是说这两个名字是一样的，无须更改。

```
ORA-00710 new tablespace name is the same as the old tablespace name
Cause : An attempt to rename a tablespace failed because the new name is the same as
the old name.
Action: No action required.
```

用户继续 DROP，仍然是 ORA-959。

```
Thu Jun 24 20:00:04 2010
DROP TABLESPACE EMIS116k
Thu Jun 24 20:00:04 2010
ORA-959 signalled during: DROP TABLESPACE EMIS116k
...
```

看着这样的过程，我们是否会有很着急的感觉？我相信在现场时，你可能会是焦急！

ORA-00600 4348 错误引发故障

终于用户发现了本质问题，发出了带着双引号的删除命令，这本身是没有问题的，可是恰恰此时数据库出了问题，一个异常的 ORA-00600 错误出现了，4348 出现：

```
Thu Jun 24 20:03:59 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:03:59 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_25919.trc:
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

再然后，案情基本成立，任何 DROP 命令都提示表空间不存在了：

```
Thu Jun 24 20:04:12 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:04:13 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116k"
```

```

...
Thu Jun 24 20:04:30 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:04:30 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116k"
...
Thu Jun 24 20:04:40 2010
DROP TABLESPACE "EMIS116n"
Thu Jun 24 20:04:40 2010
ORA-959 signalled during: DROP TABLESPACE "EMIS116n"
...

```

接下来用户继续尝试创建其他的表空间，此时 ORA-00600 25013 错误出现：

```

Fri Jun 25 09:14:00 2010
CREATE TABLESPACE "EMIS116N2" DATAFILE '/dev/vg02/rlvdata088_16G'
SIZE 7900M AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED LOGGING
EXTENT MANAGEMENT LOCAL SEGMENT SPACE MANAGEMENT AUTO
Fri Jun 25 09:18:40 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_20031.trc:
ORA-00600: internal error code, arguments: [25013], [0], [229], [EMIS116N2],
[EMIS116k], [184], [179], []
Fri Jun 25 09:18:45 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [],
[]
Fri Jun 25 09:18:47 2010
Trace dumping is performing id=[cdmp_20100625091847]
Fri Jun 25 09:18:47 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [229], [179], [184], [], [], [],
[]

```

伴随这些错误，有一个新的提示出现，Oracle 实例进行事务恢复时遇到 600 错误，这已经给我们一个暗示，

数据库内部事务出现了一致性的问题：

```
Fri Jun 25 09:19:01 2010
ORACLE Instance cwgk1 (pid = 30) - Error 600 encountered while recovering transaction
(203, 10).
```

此后，后台进程异常终止，数据库挂起，最后被迫 Abort 关闭了数据库：

```
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_pmon_4050.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
PMON: terminating instance due to error 474
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms3_4071.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms1_4067.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms0_4065.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:06 2010
Errors in file /oracle/admin/cwgk/bdump/cwgk1_lms2_4069.trc:
ORA-00474: SMON process terminated with error
Fri Jun 25 09:24:08 2010
Shutting down instance (abort)
License high water mark = 265
Fri Jun 25 09:24:12 2010
Instance terminated by PMON, pid = 4050
Fri Jun 25 09:24:13 2010
Instance terminated by USER, pid = 24022
```

接下来用户执行了一系列的恢复尝试，包括重建控制文件，强制 Resetlogs 打开数据库等，最终导致情况失控。

我们回过头来分析一下最初的错误出现时机，这里用户执行了正确的语句，但是数据库却不再配合，报出了 ORA-00600 4348 错误，这个错误号在 Metalink 上也没有收录，错误原因未知：

```
Thu Jun 24 20:03:59 2010
DROP TABLESPACE "EMIS116k"
Thu Jun 24 20:03:59 2010
Errors in file /oracle/admin/cwgk/udump/cwgk1_ora_25919.trc:
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

但是结合后面的出错信息：

```
Fri Jun 25 09:19:01 2010
ORACLE Instance cwgk1 (pid = 30) - Error 600 encountered while recovering transaction
(203, 10).
```

我们猜测，DROP TABLESPACE 实质上执行了后台递归操作，但是这个 DDL 事务因为异常，在失败后没有完成回滚，造成了数据字典不一致。

技术回放

要解决这个数据库故障，首先要分析 ORA-00600 4348 错误是如何出现的。

明白了所有的内部操作过程，我们又可以继续分析 ORA-00600 4348 号错误的成因了：

```
ORA-00600: internal error code, arguments: [4348], [U], [0], [229], [], [], [], []
```

我说过，猜测是一种重要的学习能力，现在我们就要运用这种能力了。

通过猜测推演问题

这个错误的 4384 后的第一个代码是 U，应该是 Update 的缩写，更新的应该就是 file\$，ts\$ 字典表。这里的 229 应该就是更新的表空间号，Update 出错的可能就是被更新的记录不存在了。

那么更新的记录为什么会不存在？

这个我猜测可能是某个 DBA 考虑到是这个表空间的问题，手工执行了对于字典表的 Delete 操作，这就导致了后面的异常；当然也有可能是 Oracle 数据库自身丢失了这条记录（这种可能性不大）。

通过实验验证问题

可以验证一下这个猜测，首先通过手工删除一个表空间信息。当然我们需要严正提示：这些操作仅可在测试环境，或者说属于你自己的测试环境上进行，测试之前请先备份你的数据库。

删除可以通过 dba_tablespaces 和 ts\$表进行，我们选择了对 ts\$表直接操作：

```
SQL> delete from dba_tablespaces where tablespace_name='ENMO';
```

已删除 1 行。

```
SQL> select ts#,name,online$ from ts$ ;
```

TS#	NAME	ONLINE\$

0	SYSTEM	1
1	UNDOTBS1	1
2	SYSAUX	1
3	TEMP	1
4	USERS	1
5	UNDOTBS2	3
6	EYGLE	1
7	EYGL	3

已选择 8 行。

```
SQL> rollback;
```

回退已完成。

```
SQL> select ts#,name,online$ from ts$ ;
```

TS#	NAME	ONLINE\$

0	SYSTEM	1
1	UNDOTBS1	1
2	SYSAUX	1
3	TEMP	1
4	USERS	1
5	UNDOTBS2	3
6	EYGLE	1

7 EYGL	3
8 ENMO	1

已选择 9 行。

```
SQL> delete from ts$ where ts#=8;
```

已删除 1 行。

```
SQL> commit;
```

提交完成。

删除了字典表信息之后，如果此时再来执行 drop tablespace 的操作，即出现了 4348 错误，后面的参数 8 是指表空间号是 8，也就是说当试图去 Update 修改 8 号表空间的状态时出现了异常，也就是发现这个表空间的信息不存在了，在正常情况下，这种情况绝不应该发生，因为数据库根本不对 ts\$ 执行 DELETE 操作：

```
SQL> drop tablespace enmo;
```

```
drop tablespace enmo
```

```
*
```

第 1 行出现错误：

```
ORA-00600: 内部错误代码, 参数: [4348], [U], [0], [8], [], [], [], []
```

告警日志中也记录了错误信息，并且产生了一个跟踪文件，这里的 8 号表空间也就是 ENMO 表空间：

```
Mon Aug 09 15:58:10 2010
```

```
drop tablespace enmo
```

```
Mon Aug 09 15:58:10 2010
```

```
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:
```

```
ORA-00600: 内部错误代码, 参数: [4348], [U], [0], [8], [], [], [], []
```

```
Mon Aug 09 15:58:11 2010
```

```
ORA-600 signalled during: drop tablespace enmo...
```

跟踪文件中也显示全部的相关信息，绑定变量等：

```
Cursor#3(04D60CA4) state=BOUND curiob=04D7B9B4
```

```
curflg=5 fl2=0 par=04D60C24 ses=34343FE4
```

```
sqltxt(3028CDA4)=
```

```
update ts$ set name=:2,online$=:3,content$=:4,undofile#=:5,undoblock#=:6,
```

```
blocksize=:7,dflmaxext=:8,dflinit=:9,dflincr=:10,dflxtpct=:11,dflminext=:12,dflminle
```

```

n=:13,owner#=:14,scnwrp=:15,scnbas=:16,pitrscnwrp=:17,pitrscnbas=:18,dflogging=:19,bi
tmapped=:20,inc#=:21,flags=:22,plugged=:23,spare1=:24,spare2=:25 where ts#=:1
    hash=29d4143c7a377b40366bdc581b9a48d1
    parent=2FED6560 maxchild=02 plk=31B316EC ppn=n
cursor instantiation=04D7B9B4 used=1281340689
    child#0(3028CC60) pcs=2FED6764
        clk=31B31BBC ci=2FED5D00 pn=00000000 ctx=2F966490
    kgscclfg=0 llk[04D7B9B8,04D7B9B8] idx=0
    xscflg=a0100426 fl2=5000400 fl3=2218c fl4=0
    sharing failure(s)=10
    Bind bytecodes
        Opcode = 5    Bind Rpi Scalar Sql In (not out) Nocopy
.....,
kkscoacd
    Bind#0
        oacdty=01 mxl=32(04) mxlc=00 mal=00 scl=00 pre=00
        oacflg=18 fl2=0001 frm=01 csi=852 siz=32 off=0
        kxsbbbf=30284552 bln=32 avl=04 flg=09
        value="ENM0"
    Bind#1
        oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
        oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
        kxsbbbf=04d734f4 bln=24 avl=02 flg=05
        value=2
    Bind#2~ Bind#4 value=0
    Bind#5
        oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
        oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
        kxsbbbf=04d7345c bln=24 avl=03 flg=05
        value=8192
    Bind#6
        oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
        oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0

```

kxsbbbf=04d73438 bln=24 avl=06 flg=05
value=2147483645

Bind#7

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d73414 bln=24 avl=02 flg=05
value=8

Bind#8

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d733f0 bln=24 avl=03 flg=05
value=128

Bind#9

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d733cc bln=24 avl=01 flg=05
value=0

Bind#10

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d733a8 bln=24 avl=02 flg=05
value=1

Bind#11

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d7337c bln=24 avl=02 flg=05
value=8

Bind#12~ Bind#15 value=0

Bind#16

oacdy=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbf=04d7284c bln=24 avl=04 flg=05
value=907939

Bind#17

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=04d72828 bln=24 avl=02 flg=05
value=1
```

Bind#18

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=04d72804 bln=24 avl=02 flg=05
value=8
```

Bind#19

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=04d727e0 bln=24 avl=02 flg=05
value=1
```

Bind#20

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=04d727bc bln=24 avl=02 flg=05
value=33
```

Bind#21~ Bind#23 value=0

Bind#24

```
oacdty=02 mxl=22(22) mxlc=00 mal=00 scl=00 pre=00
oacflg=08 fl2=0001 frm=00 csi=00 siz=24 off=0
kxsbbbfp=04d73518 bln=22 avl=02 flg=05
value=8
```

Frames pfr 04D713C0 siz=7188 efr 04D71338 siz=7172

Cursor frame dump

```
enxt: 4.0x00000404  enx: 3.0x000009b8  enx: 2.0x000001f4  enx: 1.0x00000c54
pnxt: 2.0x00000004  pnxt: 1.0x0000000c
kxscphp 04D70064 siz=2016 inu=0 nps=624
kxscbhp 04D66510 siz=2016 inu=0 nps=1432
```

```

Cursor#1(04D60C24) state=BOUND curiob=04D74B38
  curflg=4c fl2=400 par=00000000 ses=34352EC4
  child cursor: 3
  sqltxt(04AAFC04)=drop tablespace enmo
  hash=00000000000000000000000000000000
  parent=04AA2870 maxchild=01 plk=04AAFD24 ppn=y
cursor instantiation=04D74B38 used=1281340689
  child#0(04AA2264) pcs=04AA2684
    clk=04AA2334 ci=04AA2010 pn=04AA23A4 ctx=04AA1814
  kgscclg=0 llk[04D74B3C,04D74B3C] idx=0
  xscflg=c0802276 fl2=c000400 fl3=2202008 fl4=100
  Frames pfr 04D74AFC siz=240 efr 04D74AAC siz=232
  Cursor frame dump
    enxt: 1.0x000000e8
    pnxt: 1.0x00000008
  kxscphp 04D66020 siz=1000 inu=0 nps=140
  kxscefh 04D704C0 siz=4072 inu=0 nps=244
-----

```

而如果此时我们尝试去重新创建 ENMO 表空间,则会出现 25013 / 25015 错误,客户的情况完全得以重演。这里的 25015 错误跟随着几个参数,其中 8 指表空间号,新创建的 ENMO 表空间被重新赋予了 8 号表空间,第二个参数 6 是指表空间记录号,5 是指数据文件号:

```

SQL> create tablespace enmo datafile size 10M;
create tablespace enmo datafile size 10M
*
```

第 1 行出现错误:

```

ORA-00603: ORACLE server session terminated by fatal error
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []
ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []
ORA-00600: internal error code, arguments: [25013], [0], [8], [ENMO], [ENMO], [5],
[6], []

```

进程 ID: 0

会话 ID: 159 序列号: 14

检查跟踪文件中的错误信息，其错误情况与客户的情况完全相符，最后数据库实例崩溃：

Mon Aug 09 16:12:59 2010

ORA-600 signalled during: create tablespace enmo datafile size 10M...

Mon Aug 09 16:13:00 2010

Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:

ORA-00600: 内部错误代码, 参数: [25015], [8], [6], [5], [], [], [], []

ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [ENMO], [ENMO], [5], [6], []

Mon Aug 09 16:13:00 2010

Errors in file d:\oracle\admin\enmo\udump\enmo_ora_4736.trc:

ORA-00600: 内部错误代码, 参数: [kccocx_01], [], [], [], [], [], [], []

ORA-00600: 内部错误代码, 参数: [25015], [8], [6], [5], [], [], [], []

ORA-00600: 内部错误代码, 参数: [25013], [0], [8], [ENMO], [ENMO], [5], [6], []

Mon Aug 09 16:17:34 2010

ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction (6, 42).

Mon Aug 09 16:17:34 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_4824.trc:

ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

Mon Aug 09 16:27:39 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_pmon_4252.trc:

ORA-00474: SMON process terminated with error

Instance terminated by PMON, pid = 4252

在下次启动数据库时，告警日志中出现如下错误序列，随后数据库实例 Crash，也完全符合用户的情况：

Mon Aug 09 17:04:28 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:

ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

Mon Aug 09 17:04:30 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:

ORA-00600: internal error code, arguments: [kccocx_01], [], [], [], [], [], [], []

ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

ORACLE Instance enmo (pid = 8) - Error 600 encountered while recovering transaction (6, 42).

Mon Aug 09 17:04:30 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_3808.trc:

ORA-00600: internal error code, arguments: [25015], [8], [6], [5], [], [], [], []

至此，我们已经完全能够解析出用户故障的来龙去脉。数据库中回滚段 6 上 Slot 42 存在一个死事务，也就是表空间创建失败后的事务无法回滚：

```
SQL> select ADDR,KTUXEUSN,KTUXESLT,KTUXESQN,KTUXESIZ,KTUXECFL
       2  from x$ktuxe where ktuxeusn=6 and ktuxeslt=42;
```

ADDR	KTUXEUSN	KTUXESLT	KTUXESQN	KTUXESIZ	KTUXECFL
------	----------	----------	----------	----------	----------

094878F0	6	42	332	1	DEAD
----------	---	----	-----	---	------

```
SQL> select distinct KTUXECFL,count(*) from x$ktuxe group by KTUXECFL;
```

KTUXECFL	COUNT (*)
----------	-----------

DEAD	1
------	---

NONE	577
------	-----

将 UNDO Header 转储出来，可以验证这个判断：

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

USN	NAME
-----	------

0	SYSTEM
---	--------

1	_SYSSMU1\$
---	------------

2	_SYSSMU2\$
---	------------

3	_SYSSMU3\$
---	------------

4	_SYSSMU4\$
---	------------

5	_SYSSMU5\$
---	------------

6	_SYSSMU6\$
---	------------

```
7 _SYSSMU7$
8 _SYSSMU8$
9 _SYSSMU9$
10 _SYSSMU10$
```

已选择 11 行。

```
SQL> alter system dump undo header "_SYSSMU6$";
```

系统已更改。

从跟踪文件中可以找到 6 号回滚段的内容，其中 2a 正是 10 进制的 42，此处的活动事务不能回退导致数据库实例的挂起和故障，由于我们指定了损坏参数来打开数据库，这个回滚段头被标记为损坏：

```
Block Checking: DBA = 8388697, Block Type = System Managed Segment Header Block
ERROR: SMU Segment Header Corrupted. Error Code = 38508
ktu4smck: starting extent(0x11) of txn slot #0x2a is invalid.
```

```
valid value (0 - 0x10)
*****
Undo Segment: _SYSSMU6$ (6)
*****
```

Extent Control Header								

oooooooo								
TRN TBL::								
index	state	cflags	wrap#	uel	scn	dba	nub	stmt_num
cmt								

oooooooo								
0x26	9	0x00	0x014c	0x002b	0x0000.000ddc57	0x008003ee	0x00000007	
1281340831								
0x27	9	0x00	0x014c	0x0029	0x0000.000ddc4a	0x00000000	0x00000000	
1281340831								
0x28	9	0x00	0x014c	0x002d	0x0000.000e7d51	0x00000000	0x00000000	
1281346061								
0x29	9	0x00	0x014c	0x002e	0x0000.000ddc4c	0x008003ea	0x00000001	

```

1281340831
    0x2a    10    0x10  0x014c  0x0011  0x0000.000ddd7e  0x008003ed  0x00000001  0
    0x2b     9    0x00  0x014c  0x0028  0x0000.000e2dac  0x00000000  0x00000000
1281344675
    0x2c     9    0x00  0x014c  0x0026  0x0000.000ddc55  0x008003ed  0x00000003
1281340831
    0x2d     9    0x00  0x014c  0xffff  0x0000.000e7ebc  0x00000000  0x00000000
1281346136
    0x2e     9    0x00  0x014c  0x0025  0x0000.000ddc4f  0x00000000  0x00000000
1281340831
    0x2f     9    0x00  0x014b  0x0003  0x0000.000dcea2  0x008003de  0x00000001
1281332703

```

如果此时我们将 6 号回滚段标记为损坏，则可以避免回滚时出现的问题，正常无误的启动数据库：

```
SQL> alter system set "_corrupted_rollback_segments"='_SYSSMU6$' scope=spfile;
```

系统已更改。

```
SQL> alter system set "_offline_rollback_segments"='_SYSSMU6$' scope=spfile;
```

系统已更改。

```
SQL> shutdown immediate;
```

数据库已经关闭。

已经卸载数据库。

ORACLE 例程已经关闭。

```
SQL> startup
```

ORACLE 例程已经启动。

```
Total System Global Area  612368384 bytes
```

```
Fixed Size                  1298160 bytes
```

```
Variable Size               167772432 bytes
```

```
Database Buffers           436207616 bytes
```

```
Redo Buffers                7090176 bytes
```

数据库装载完毕。

数据库已经打开。

```
SQL> show parameter rollback
```

NAME	TYPE	VALUE
------	------	-------

```

-----
_corrupted_rollback_segments      string      _SYSSMU6$
_offline_rollback_segments        string      _SYSSMU6$
fast_start_parallel_rollback      string      LOW
rollback_segments                 string
transactions_per_rollback_segment integer      5

```

如果此时尝试 DROP 回滚段，则数据库还会出现 600 错误：

```

SQL> drop rollback segment "_SYSSMU6$";
drop rollback segment "_SYSSMU6$"
*
```

第 1 行出现错误：

ORA-00607：当更改数据块时出现内部错误

ORA-00600：内部错误代码，参数：[kddummy_blkchk], [2], [89], [38508], [], [], [], []

```
SQL> drop tablespace enmo;
```

表空间已删除。

告警日志信息如下，这里 kddummy_blkchk 不必检索文档，大致可以猜测是数据块检查出现问题。此处检查的文件 2，数据块 89 正是我们的 6 号回滚段：

```

Mon Aug 09 17:46:41 2010
drop rollback segment "_SYSSMU6$"
Mon Aug 09 17:46:42 2010
Errors in file d:\oracle\admin\enmo\udump\enmo_ora_2432.trc:
ORA-00600: 内部错误代码, 参数: [kddummy_blkchk], [2], [89], [38508], [], [], [], []

Mon Aug 09 17:46:43 2010
Doing block recovery for file 2 block 89
Block recovery from logseq 4, block 405 to scn 970707
Mon Aug 09 17:46:43 2010
Recovery of Online Redo Log: Thread 1 Group 1 Seq 4 Reading mem 0
  Mem# 0: D:\ORACLE\ORADATA\ENMO\REDO01.LOG
Block recovery completed at rba 4.407.16, scn 0.970708
ORA-607 signalled during: drop rollback segment "_SYSSMU6$"...
Mon Aug 09 17:46:43 2010

```

Corrupt Block Found

TSN = 1, TSNAME = UNDOTBS1

RFN = 2, BLK = 89, RDBA = 8388697

OBJN = 0, OBJD = -1, OBJECT = C_TS#, SUBOBJECT =

SEGMENT OWNER = SYS, SEGMENT TYPE = Cluster Segment

Mon Aug 09 17:46:43 2010

Errors in file d:\oracle\admin\enmo\bdump\enmo_smon_6092.trc:

ORA-00600: internal error code, arguments: [kddummy_blkchk], [2], [89], [38508], [],
[], [], []

此后通过如下一系列的处理，数据库可以成功被打开，但是根据之前的分析，我们应当知道，数据库因此被强制放弃了一些事务的一致性，最好通过导出/导入进行数据库重构：

SQL> alter system set "_allow_resetlogs_corruption"=true scope=spfile;

系统已更改。

SQL> shutdown immediate;

数据库已经关闭。

已经卸载数据库。

ORACLE 例程已经关闭。

SQL> startup mount;

ORACLE 例程已经启动。

Total System Global Area 612368384 bytes

Fixed Size 1298160 bytes

Variable Size 167772432 bytes

Database Buffers 436207616 bytes

Redo Buffers 7090176 bytes

数据库装载完毕。

SQL> recover database using backup controlfile until cancel;

ORA-00279: 更改 1011115 (在 08/09/2010 17:52:04 生成) 对于线程 1 是必需的

ORA-00289: 建议: D:\ORACLE\10.2.0\RDBMS\ARC00006_0726573063.001

ORA-00280: 更改 1011115 (用于线程 1) 在序列 #6 中

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

cancel

介质恢复已取消。

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

数据库已更改。

```
SQL> create undo tablespace undotbs2 datafile size 10M;
```

表空间已创建。

```
SQL> alter system set undo_tablespace=undotbs2;
```

系统已更改。

```
SQL> alter tablespace undotbs1 offline;
```

表空间已更改。

```
SQL> drop tablespace undotbs1;
```

表空间已删除。

这就是一个字符引发的灾难。

一个盘符引发的灾难

判断失误导致的误格式化故障

我们众多的案例都表明，在数据环境中，任何一个微弱的疏忽，都可能导致数据环境的万劫不复。

这个故障是关于盘符的。

灾难描述

用户是这样描述这次灾难的：

1. 一个 1TB 的核心数据库无备份
2. 客户扩展了一块 2TB 的硬盘用于备份
3. 工作在凌晨进行，安装新硬盘后重新启动
4. 分区，格式化后启动数据库
5. ASM 报错无法启动
6. 工程师介入检查，发现盘符发生变化，原有数据盘被格式化
7. 灾难形成

在这个案例发生的过程中，客户操作人员曾经询问厂商人员，是这块盘没错吧？结果厂商人员给了一个模棱两可的答复，应该没错。结果恰恰是错了。

案例警示

这个案例给我们的警示是：

1. 专业的工作应当由专业的人来完成

与核心数据相关的工作，哪怕是再简单的工作，也没有小事情，也不容疏忽，专业技术人员的经验就是最好的回报。

所以对于数据系统的各层面维护工作，都应当尽量由专业的人员来完成专业的工作，如果不能获得专业支持，那么做好备份。

2. 在疲惫的时刻不要轻易做重要的判断

在这个案例中，当格式化硬盘前，客户曾经提问，似乎有问题，但是工程师在凌晨多个小时加班之后，下意识的期望早点完成工作，草率的未加详细验证就判断没问题，可以进行格式化，轻率判断的结果是 1T 左右的数据荡然无存。

所以我们建议，在疲惫时刻要尽量稍作判断，工作的步骤应该来自预先规划，一旦遇到存在疑问的地方，不要担心浪费时间，一定得到精确的确认后才能执行操作。

3. 重要维护应当做好测试准备工作

在核心系统中的重要操作，应当提前做好测试工作，并且明确可能存在的故障点，并且准备回退方案。在重要的任务点上，设置两个角色，互为备份、补充和审核，避免出现误操作的概率。

这一节我们主要说的是人的问题，机器过劳要出故障，人过劳同样会犯低级错误，所以要防范人的错误，要从工作时间、工作监督保障入手。

技术回放

在 Linux、UNIX 系统中，当系统的磁盘发生变化时，如果未作绑定，预先分配的磁盘盘符可能发生变化，引发不必要的麻烦。

以下是当时客户的磁盘情况：

```
[root@smsdb ~]# fdisk -l
Disk /dev/sda: 145.9 GB, 145999527936 bytes
255 heads, 63 sectors/track, 17750 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

    Device Boot      Start         End      Blocks   Id  System
/dev/sda1    *           1          129      1036161   83   Linux
/dev/sda2             130         7013      55295730   83   Linux
/dev/sda3          7014         9563      20482875   83   Linux
```

/dev/sda4	9564	17750	65762077+	5	Extended
/dev/sda5	9564	10825	10136983+	83	Linux
/dev/sda6	10826	12087	10136983+	83	Linux
/dev/sda7	12088	13094	8088696	83	Linux
/dev/sda8	13095	13223	1036161	83	Linux
/dev/sda9	13224	15312	16779861	82	Linux swap

Disk /dev/sdb: 968 MB, 968884224 bytes
 30 heads, 62 sectors/track, 1017 cylinders
 Units = cylinders of 1860 * 512 = 952320 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sdb1		1	1017	945779	83	Linux

Disk /dev/sdc: 897 MB, 897581056 bytes
 28 heads, 62 sectors/track, 1009 cylinders
 Units = cylinders of 1736 * 512 = 888832 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sdc1		1	1009	875781	83	Linux

Disk /dev/sdd: 779 MB, 779091968 bytes
 24 heads, 62 sectors/track, 1022 cylinders
 Units = cylinders of 1488 * 512 = 761856 bytes

Device	Boot	Start	End	Blocks	Id	System
/dev/sdd1		1	1022	760337	83	Linux

Disk /dev/sde: 664 MB, 664797184 bytes
 21 heads, 61 sectors/track, 1013 cylinders
 Units = cylinders of 1281 * 512 = 655872 bytes

Device	Boot	Start	End	Blocks	Id	System
--------	------	-------	-----	--------	----	--------

```
/dev/sde1          1          1013          648796    83  Linux
```

注意，这里的 /dev/sdf 就是后增加的 1 T 左右的硬盘，由于盘符变化，这个硬盘插入到原来的盘序中间：

```
Disk /dev/sdf: 1209.4 GB, 1209462790144 bytes
255 heads, 63 sectors/track, 147042 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdf1		1	147042	1181114833+	83	Linux

```
Disk /dev/sdg: 284.5 GB, 284538961920 bytes
255 heads, 63 sectors/track, 34593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdg1		1	34593	277868241	83	Linux

注意，这里的 /dev/sdh 是原有的数据盘之一，客户缺省的认为新增加的硬盘应该位于最后一个盘符，将这个 898 G 的硬盘格式化了：

```
Disk /dev/sdh: 898.3 GB, 898388459520 bytes
255 heads, 63 sectors/track, 109222 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks	Id	System
/dev/sdh1		1	109222	877325683+	83	Linux

这个数据库灾难的后续恢复非常复杂，由于 Oracle ASM 直接使用裸设备，客户使用了两块硬盘构建磁盘组，现在损失了一块硬盘，但是另外一块完好，ASM 的 AU 是在两块硬盘上均衡分布的，通过残存硬盘和格式化硬盘进行数据拼接和重组，恢复出了大部分数据，但是被文件系统 inode 块覆盖的数据就彻底丢失了，这是一次不完全恢复，恢复率大约为 99% 左右。只能以此作为最大限度进行数据挽救。

物尽其用，人尽其才

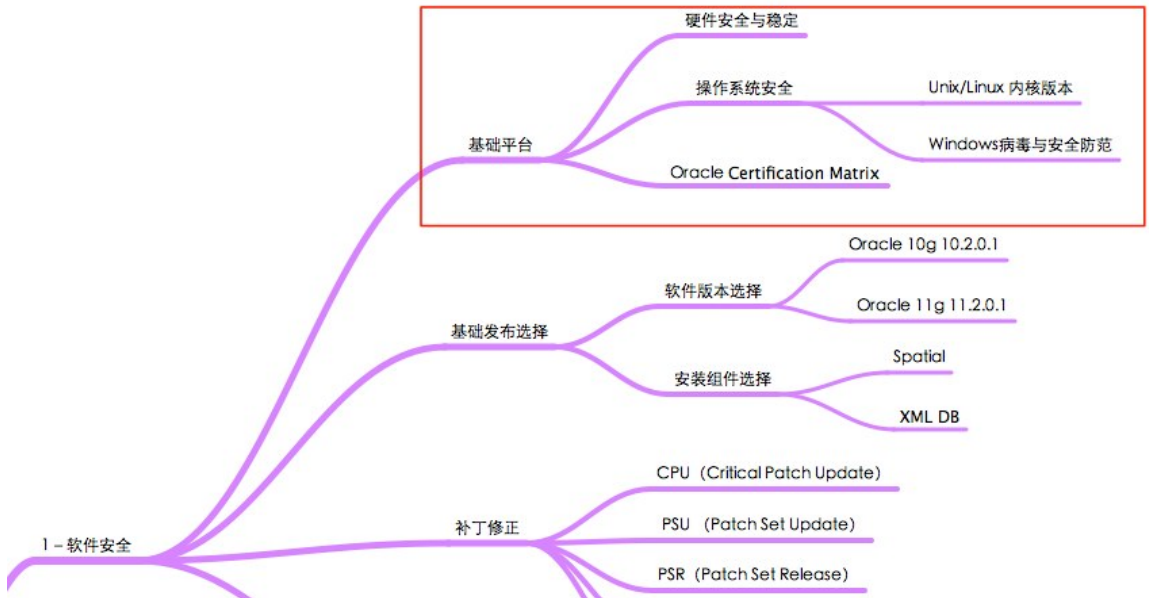
若乃人尽其才；悉用其力。

——《诗经·大雅·荡》

夫尺有所短，寸有所长，物有所不足。

智有所不明，数有所不逮，神有所不通。

——战国·楚·屈原《卜居》



在现实中，通常我们希望能够达到人尽其才，物尽其用的境界。

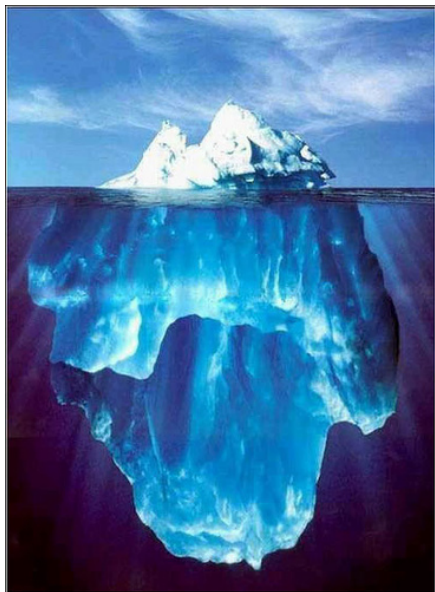
当我们使用了 Oracle 数据库时，你应当了解最为基本的数据库知识，如果你认同将重要的数据存储于数据库当中，那么最基本的了解是对自我数据的负责。虽然 Oracle 数据库经过了 30 年的发展完善，在自动化管理和优化方面已经有了大幅提升，但是最为基本的维护和“保养”仍然需要我们主动去完成。

本篇内容涉及基本的管理安全与基础平台安全，硬件的安全稳定对于数据库系统具有支撑和决定作用。

在整个数据库系统中，我们应当对使用的各类技术不足有所认知，尺有所短，物有不足，只有扬长避短才能充分发挥系统的优势，为业务发展提供有利支撑。

很多企业在选择数据库时，仅仅看到了软件的购置成本，完全不考虑软件的维护、优化等隐性成本，这是完全错误的做法，就如同购买一辆汽车一样，如果不进行定期保养维护，我们是无法对车辆的长期稳定行驶做出确保的。

对于数据库系统，我们不应当仅仅看到冰山一角，我们要时刻认识到，在水面之下，还有很多需要我们思考的问题存在，忽略这些问题就如同一叶障目，可能使我们的数据随时遭遇风险，下图正是冰山之一角，与大家共为警醒的是水面之下，我们更应该关注的部分：



本篇我们将与大家分享关于冰山的几个角落。

关库与关机

强制关机导致的写丢失故障

以下是一则与责任有关的数据库灾难案例，用户甚至对于数据库最为基本的启动和关闭常识都了解不足，几乎一无所知的管理方式最终使得数据库陷入了困境。

灾难描述

在这个案例中，用户从来都使用关机替代关库的方式进行管理，以下是案例的主要描述：

1. 用户数据库在主机重启后无法启动
2. 经检查发现用户半年多来，从来没有正常关闭过数据库
3. 用户的关库就意味着关机
4. 在最近的关机重启后数据库无法启动
5. 数据库提示控制文件不一致，数据库需要恢复
6. 灾难形成

这则案例看起来用户太缺少数据库维护的基本常识了，通过简单的培训和学习，获得基本的数据库运维知识，这是对于数据库最基本的“尊重”和重视。

案例警示

这一则案例给我们的警示是：

1. 尊重数据库即是保护数据资产

企业以 Oracle 承载数据，即是以 Oracle 作为重要的数据资产存储地。在 IT 技术快速发展的今天，企业对于数据的依赖越来越强，尤其是电子商务企业，**数据即是企业的命脉**。

根据 2007 年 Gartner 的一组调查数据显示：在经历了数据完全丢失而导致系统停运的企业中，有 2/5 再也没能恢复运营，余下的企业也有 1/3 在两年内宣告破产。由此可见数据灾难对于企业的影响是多么巨大而深远。

另据报告显示，公开披露数据丢失的企业预计将导致其客户量及相关收入降低 8%；对于上市企业而言，每股股价会下降 8%；平均每丢失一个客户记录便会造成 100 美元的额外损失。

由此可见，数据即资产，数据即财富，如果用户认同自我数据的价值，那么必须对数据库持有一定的敬畏和尊重，至少需要了解数据库最为基本的启动关闭步骤和最简单常规的备份方式。

如果完全忽视数据库的工作机制与原理，则数据安全是完全没有保障的。

2. 必要的培训和制度规范需要严格遵守

企业在部署数据库软件时，必须同时储备数据运维人才或者通过培训获得基本的运维技能，技术与维护配合才能够充分发挥 IT 系统的价值。

目前很多企业将培训视为过场，不能够从培训中获得必要的技能，很多集成商也将培训视为可有可无。

我们认为，负责任的企业，应当将最为核心和基本的数据维护技能，通过简单的方式方法传达给用户，并且将生产库的重要操作步骤规章制度化，帮助用户减少问题出现的概率可能，唯有如此，才能够保证最低层次的数据安全。

尊重与重视，使得数据库能够物尽其用，这是数据库应用的最基本原则。

恢复过程

以下是从告警日志中分析得来的一些数据，用户的数据库自 2011 年 6 月创建，以模板方式建立：

```
bogon: eygle$ strings alert_sxxhdts.log |head -5
Sun Jun 12 18:30:33 2011
alter database rename global_name to sxxhdts
Completed: alter database rename global_name to sxxhdts
```

半年多间，数据库有 326 次启动信息：

```
bogon: eygle$ strings alert_sxxhdts.log |grep "Starting ORACLE instance (normal)"|wc -l
326
```

但是数据库从未正常关闭过，以下正常关闭的信息，来自创建数据库时：

```
bogon: eygle$ strings alert_sxxhdts.log |grep "Shutting down instance"
Shutting down instance: further logons disabled
```

```
Shutting down instance (normal)
```

控制文件一致性维护

我们尝试启动这个数据库,首先报出的是控制文件不一致,其中第三个控制文件版本比较新,版本号为 2623:

```
SQL> startup pfile=initora9i.ora
ORACLE instance started.
```

```
Total System Global Area  126950956 bytes
Fixed Size                  454188 bytes
Variable Size               92274688 bytes
Database Buffers           33554432 bytes
Redo Buffers                667648 bytes
ORA-00214: controlfile 'D:\ORACLE\ORADATA\SXXHDS\CONTROL03.CTL' version 2623
inconsistent with file 'D:\ORACLE\ORADATA\SXXHDS\CONTROL02.CTL' version 2619
```

通常 Oracle 的三个控制文件是互为镜像的,其内容完全相同,Oracle 通过并行写入来更新控制文件,如果出现不一致现象,那么意味着存储的写入出现丢失,损失了物理 IO。

我们稍微来解析一下控制文件的更新维护过程,以下通过 dbms_monitor 来跟踪一下 CKPT 后台进程对于控制文件的维护操作:

```
SQL> select * from v$version where rownum <2;
BANNER
-----
Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - 64bi
SQL> select sid,serial#,program from v$session where program like '%CKPT%';
      SID      SERIAL#  PROGRAM
-----
      55         1 oracle@hpserver2.enmotech.com (CKPT)
SQL> select spid,program from v$process where program like '%CKPT%';
SPID      PROGRAM
-----
6168      oracle@hpserver2.enmotech.com (CKPT)
SQL> exec dbms_monitor.session_trace_enable(55,1,TRUE,TRUE);
```

PL/SQL procedure successfully completed.

SQL> alter system checkpoint;

System altered.

SQL> exec dbms_monitor.session_trace_disable;

PL/SQL procedure successfully completed.

在 CKPT 进程的跟踪文件中，可以找到执行 Checkpoint 检查点后，对于控制文件的写操作，以下是摘录信息：

```
WAIT #0: nam='db file sequential read' ela= 18 file#=1 block#=1 blocks=1 obj#=-1
tim=1297696398228494
WAIT #0: nam='db file single write' ela= 4952 file#=1 block#=1 blocks=1 obj#=-1
tim=1297696398233526
WAIT #0: nam='db file sequential read' ela= 24 file#=2 block#=1 blocks=1 obj#=-1
tim=1297696398233664
WAIT #0: nam='db file single write' ela= 4206 file#=2 block#=1 blocks=1 obj#=-1
tim=1297696398237931
WAIT #0: nam='db file sequential read' ela= 24 file#=3 block#=1 blocks=1 obj#=-1
tim=1297696398238070
WAIT #0: nam='db file single write' ela= 9713 file#=3 block#=1 blocks=1 obj#=-1
tim=1297696398247844
WAIT #0: nam='db file sequential read' ela= 19 file#=4 block#=1 blocks=1 obj#=-1
tim=1297696398247979
WAIT #0: nam='db file single write' ela= 4774 file#=4 block#=1 blocks=1 obj#=-1
tim=1297696398252817
WAIT #0: nam='db file sequential read' ela= 19 file#=5 block#=1 blocks=1 obj#=-1
tim=1297696398252949
WAIT #0: nam='db file single write' ela= 9004 file#=5 block#=1 blocks=1 obj#=-1
tim=1297696398262018
WAIT #0: nam='db file sequential read' ela= 19 file#=6 block#=1 blocks=1 obj#=-1
tim=1297696398262214
WAIT #0: nam='db file single write' ela= 11076 file#=6 block#=1 blocks=1 obj#=-1
tim=1297696398273357
WAIT #0: nam='control file parallel write' ela= 20004 files=3 block#=12 requests=3
obj#=-1 tim=1297696398293494
```

```
WAIT #0: nam='control file parallel write' ela= 24585 files=3 block#=15 requests=3
obj#=-1 tim=1297696398318200
WAIT #0: nam='control file parallel write' ela= 23752 files=3 block#=10 requests=3
obj#=-1 tim=1297696398342077
WAIT #0: nam='control file parallel write' ela= 24042 files=3 block#=8 requests=3
obj#=-1 tim=1297696398366275
WAIT #0: nam='control file parallel write' ela= 23531 files=3 block#=1 requests=3
obj#=-1 tim=1297696398389955
```

注意后台的控制文件并行写（control file parallel write）操作就是检查点执行时，CKPT 进程更新控制文件的过程，这一等待事件的三个参数内容如下：

```
SQL> select name,parameter1,parameter2,parameter3
       2  from v$event_name where name ='control file parallel write';
NAME                                PARAMETER1          PARAMETER2  PARAMETER3
-----
control file parallel write         files                block#      requests
```

在跟踪文件显示的参数内容分别如下，其中 files 和 Requests 的参数值相同，请求 3 次 IO 操作，更新了 3 个控制文件，块号显示更新的控制文件块：

等待事件	Files	Block#	Requests
control file parallel write	3	12	3
control file parallel write	3	15	3
control file parallel write	3	10	3
control file parallel write	3	8	3
control file parallel write	3	1	3

如果这发出的三次 IO 丢失了一个或者两个，则控制文件不一致的现象就可能出现了。

继续前面的案例分析，我们可以跟踪一下数据库启动这个过程：

```
SQL> startup nomount pfile=initora9i.ora
ORACLE instance started.

Total System Global Area  126950956 bytes
Fixed Size                  454188 bytes
Variable Size              92274688 bytes
```

Database Buffers 33554432 bytes

Redo Buffers 667648 bytes

```
SQL> alter session set events '10046 trace name context forever,level 12';
```

Session altered.

```
SQL> alter database mount;
```

```
alter database mount
```

*

ERROR at line 1:

ORA-00214: controlfile 'D:\ORACLE\ORADATA\SXXHDTs\CONTROL03.CTL' version 2623

inconsistent with file 'D:\ORACLE\ORADATA\SXXHDTs\CONTROL02.CTL' version 2619

查看后台生成的 10046 跟踪文件，摘录部分信息：

*** SESSION ID:(9.1) 2012-01-10 13:16:41.183

APPNAME mod='sqlplus.exe' mh=0 act='' ah=0

=====

PARSING IN CURSOR #1 len=68 dep=0 uid=0 oct=42 lid=0 tim=12373534530 hv=1346161232

ad='6a3c9d04'

alter session set events '10046 trace name context forever,level 12'

END OF STMT

EXEC #1:c=0,e=32,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=12373520716

WAIT #1: nam='SQL*Net message to client' ela= 6 p1=1111838976 p2=1 p3=0

WAIT #1: nam='SQL*Net message from client' ela= 4508814 p1=1111838976 p2=1 p3=0

XCTEND rlbk=0, rd_only=1

=====

PARSING IN CURSOR #1 len=20 dep=0 uid=0 oct=35 lid=0 tim=12378053544 hv=1379354989

ad='6a3c8d64'

alter database mount

END OF STMT

PARSE #1:c=0,e=3026,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=12378053538

BINDS #1:

WAIT #1: nam='reliable message' ela= 27 p1=1760935332 p2=1760905580 p3=1761534860

WAIT #1: nam='rdbms ipc reply' ela= 2724 p1=5 p2=900 p3=0

WAIT #1: nam='control file sequential read' ela= 254 p1=0 p2=1 p3=1

WAIT #1: nam='control file sequential read' ela= 219 p1=1 p2=1 p3=1

```

WAIT #1: nam='control file sequential read' ela= 208 p1=2 p2=1 p3=1
WAIT #1: nam='reliable message' ela= 996893 p1=1760935332 p2=1760905876 p3=1761534860
WAIT #1: nam='reliable message' ela= 999155 p1=1760935332 p2=1760905876 p3=1761534860
WAIT #1: nam='reliable message' ela= 990131 p1=1760935332 p2=1760905876 p3=1761534860
EXEC #1:c=0,e=3010793,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=12381070548
ERROR #1:err=214 tim=1237671
WAIT #1: nam='SQL*Net break/reset to client' ela= 5 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net break/reset to client' ela= 83 p1=1111838976 p2=0 p3=0
WAIT #1: nam='SQL*Net message to client' ela= 6 p1=1111838976 p2=1 p3=0

```

我们可以看到数据库在 Mount 过程中，顺序读取了 3 个控制文件的第一个数据块，进行比较，如果控制文件的版本不一致，就会抛出前面的异常。以下是一个控制文件的转储，头块上记录的 Seq 就是控制文件的版本号，控制文件的校验仅读取这个信息就可以完成：

```

*** 2012-01-10 11:13:27.919
*** SESSION ID:(9.3) 2012-01-10 11:13:27.868
DUMP OF CONTROL FILES, Seq # 2626 = 0xa42
FILE HEADER:
    Software vsn=153092096=0x92000000, Compatibility Vsn=134217728=0x80000000
    Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
    Activation ID=0=0x0
    Control Seq=2626=0xa42, File size=246=0xf6
    File Number=0, Blksiz=8192, File Type=1 CONTROL

```

Oracle 的文件恢复判断

正常情况下，Oracle 的控制文件是完全相同的三个拷贝，由于出现不一致现象，三号控制文件较新，所以我们使用该控制文件来尝试启动数据库。

这时数据库提示 UNDO 文件需要恢复：

```

SQL> startup pfile=initora9i.ora
ORACLE instance started.

```

```

Total System Global Area  126950956 bytes
Fixed Size                  454188 bytes

```

```

Variable Size          92274688 bytes
Database Buffers      33554432 bytes
Redo Buffers          667648 bytes
Database mounted.
ORA-01113: file 2 needs media recovery
ORA-01110: data file 2: 'D:\ORACLE\ORADATA\SXXHDTS\UNDOTBS01.DBF'

```

首先我们来分析一下 Oracle 如何判断 UNDOTBS01.DBF 文件需要恢复，通过 10046 事件跟踪一下数据库 Open 过程：

```

SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.

SQL> alter database open;
alter database open
*

ERROR at line 1:
ORA-01113: file 2 needs media recovery
ORA-01110: data file 2: 'D:\ORACLE\ORADATA\SXXHDTS\UNDOTBS01.DBF'

```

查看跟踪文件获得如下信息：

```

*** 2012-01-10 11:49:02.328
APPNAME mod='sqlplus.exe' mh=0 act='' ah=0
=====
PARSING IN CURSOR #1 len=68 dep=0 uid=0 oct=42 lid=0 tim=7114430174 hv=1346161232
ad='6a3cbec0'
alter session set events '10046 trace name context forever,level 12'
END OF STMT
EXEC #1:c=0,e=29,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=7114401350
WAIT #1: nam='SQL*Net message to client' ela= 7 p1=1111838976 p2=1 p3=0
*** 2012-01-10 11:49:17.259
WAIT #1: nam='SQL*Net message from client' ela= 14920900 p1=1111838976 p2=1 p3=0
XCTEND rlbk=0, rd_only=1
=====
PARSING IN CURSOR #1 len=19 dep=0 uid=0 oct=35 lid=0 tim=7129363012 hv=2631704207
ad='6a3c6d48'

```

```

alter database open
END OF STMT
PARSE #1:c=0,e=205,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=7129363006
BINDS #1:
WAIT #1: nam='control file sequential read' ela= 745 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 148 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 142 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 155 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 90731 p1=3 p2=910 p3=0
WAIT #1: nam='control file sequential read' ela= 188 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 141 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 158 p1=0 p2=12 p3=1
WAIT #1: nam='direct path read' ela= 23 p1=1 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=2 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=3 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=4 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=5 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=6 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=7 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=8 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 7 p1=9 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=10 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=201 p2=1 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 171 p1=3 p2=2147483647 p3=0
EXEC #1:c=10014,e=101420,p=11,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=7129464473
ERROR #1:err=1113 tim=712510
WAIT #1: nam='SQL*Net break/reset to client' ela= 5 p1=1111838976 p2=1 p3=0

```

```
WAIT #1: nam='SQL*Net break/reset to client' ela= 63 p1=1111838976 p2=0 p3=0
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1111838976 p2=1 p3=0
```

对以上跟踪信息进行一点深入分析，以下一段通过'control file sequential read'读取控制文件中的数据块：

```
WAIT #1: nam='control file sequential read' ela= 745 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 148 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 142 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 155 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=0 p2=239 p3=1
WAIT #1: nam='rdbms ipc reply' ela= 90731 p1=3 p2=910 p3=0
WAIT #1: nam='control file sequential read' ela= 188 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 147 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 144 p1=2 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 141 p1=0 p2=239 p3=1
WAIT #1: nam='control file sequential read' ela= 158 p1=0 p2=12 p3=1
```

等待事件 control file sequential read 包含三个参数输出，这三个参数分别代表文件号，块号和读取的块的数量：

```
SQL> select name,parameter1,parameter2,parameter3
       2  from v$event_name where name ='control file sequential read';
```

NAME	PARAMETER1	PARAMETER2	PARAMETER3
control file sequential read	file#	block#	blocks

以上输出说明启动过程中，数据库读取了控制文件的第 1、12、239 块。再然后，读取了每个数据文件的第一个数据块：

```
WAIT #1: nam='direct path read' ela= 23 p1=1 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=2 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=3 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=4 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=5 p2=1 p3=1
```

```

WAIT #1: nam='direct path read' ela= 4 p1=6 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 5 p1=7 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=8 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 7 p1=9 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=10 p2=1 p3=1
WAIT #1: nam='direct path read' ela= 4 p1=201 p2=1 p3=1

```

然后抛出了异常，提示 UNDO 文件需要恢复。

我们可以转储一下控制文件，看看控制文件中记录了什么信息，以及为何 UNDO 文件需要恢复：

```

SQL> alter session set events 'immediate trace name file_hdrs level 12';
Session altered.

```

控制文件中记录的信息包含了对于恢复至关重要的 RBA 信息，RBA 由三部分组成，分别为日志序号，日志块号，偏移量信息，数据文件的恢复需要从这个 RBA 开始。

以下日志中显示，对于 SYSTEM 表空间其 RBA 为 0x13e.2.10：

```

FILE HEADER:
    Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
    Db ID=615401347=0x24ae4783, Db Name='SXHDTS'
    Activation ID=0=0x0
    Control Seq=2624=0xa40, File size=52480=0xcd00
    File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM rel_fn:1
Creation at scn: 0x0000.00000000b 05/12/2002 16:17:58
Backup taken at scn: 0x0000.000000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 12/31/2011 17:47:30
status:0x4 root dba:0x004001a1 chkpt cnt: 942 ctl cnt:941
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.0156eaa8 12/31/2011 17:47:31
thread:1 rba:(0x13e.2.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000

```

而对于 UNDO 表空间，其 RBA 信息为 0x13d.2.10。这个 RBA 及其对应的检查点小于 SYSTEM 表空间的相应信息：

FILE HEADER:

```
Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
Activation ID=0=0x0
Control Seq=2618=0xa3a, File size=25600=0x6400
File Number=2, Blksiz=8192, File Type=3 DATA
Tablespace #1 - UNDOTBS1 rel_fn:2
Creation at scn: 0x0000.0002dd31 05/12/2002 20:22:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 12/31/2011 08:28:19
status:0x4 root dba:0x00000000 chkpt cnt: 927 ctl cnt:926
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.0155b0f1 12/31/2011 08:28:19
thread:1 rba:(0x13d.2.10)
```

而数据库的检查点信息如下，显然检查点信息和 UNDO 最后的检查点相符合，SYSTEM 等文件的检查点已经超前。

以下是 REDO THREAD 记录，其中明确说明当前数据库有 3 组日志文件，第一个是 1，第三个是 3，当前使用日志组 1，最后的序列号是 13d:

```
*****
REDO THREAD RECORDS
*****
(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0xf thread links forward:0 back:0
#logs:3 first:1 last:3 current:1 last used seq#:0x13d
enabled at scn: 0x0000.0002e872 06/12/2011 18:30:27
disabled at scn: 0x0000.00000000 01/01/1988 00:00:00
opened at 12/31/2011 08:28:19 by instance sxxhdts
Checkpointed at scn: 0x0000.0155b0f1 12/31/2011 08:28:19
thread:1 rba:(0x13d.2.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
log history: 316
```

日志记录的最后检查点 SCN 是 0155b0f1,其 RBA 为 13d.2.10 。注意，这个数据库最后完成的检查点位于 2011 年 12 月 31 日，这也是一个没有撑过新年的数据库之一。

前面提到 SYSTEM 表空间的检查点 SCN 为 0156eaa8,RBA 为 13e.2.10,对比列表如下：

	Checkpoint SCN	RBA
SYSTEM	0156eaa8	0x13e.2.10
UNDO	0155b0f1	0x13d.2.10
Diff	80311	

由于 UNDO 文件的 RBA 及检查点信息滞后，所以优先提示从 UNDO 表空间进行恢复，恢复的起点就是 0x13d.2.10:

```
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.
SQL> recover datafile 2;
Media recovery complete.
```

以下是告警日志中恢复 UNDO 文件的提示：

```
Tue Jan 10 13:35:35 2012
ALTER DATABASE RECOVER datafile 2
Media Recovery Start
WARNING! Recovering data file 2 from a fuzzy backup. It might be an online
backup taken without entering the begin backup command.
Tue Jan 10 13:35:35 2012
Recovery of Online Redo Log: Thread 1 Group 1 Seq 317 Reading mem 0
  Mem# 0 errs 0: D:\ORACLE\ORADATA\SXXHDTs\RED001.LOG
Media Recovery Complete
Completed: ALTER DATABASE RECOVER datafile 2
```

恢复完成之后，检查控制文件及数据文件信息，可以确认恢复是被正确执行了的。控制文件中的检查点信息如下：

```
*****
CHECKPOINT PROGRESS RECORDS
*****
(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
```

```

THREAD #1 - status:0x1 flags:0x0 dirty:0
low cache rba:(0xffffffff.ffffffff.ffff) on disk rba:(0x13d.ac56.0)
on disk scn: 0x0000.01569c86 12/31/2011 17:45:38
resetlogs scn: 0x0000.0002e872 06/12/2011 18:30:27
heartbeat: 772188978 mount id: 633905756
MTTR statistics status: 3
Init time: Avg: 9938392, Times measured: 3
File open time: Avg: 12349, Times measured: 39
Log block read time: Avg: 24, Times measured: 32770
Data block handling time: Avg: 2067, Times measured: 301

```

其中 On Disk RBA 是恢复的终点, 也即 0x13d.ac56.0, UNDO 文件恢复之后, 其文件信息如下, 注意其 RBA 信息同样推进到 0x13d.ac56.0, 也就是说所有的 REDO 都被应用, 这是基于所有 REDO 的完全恢复:

```

WAIT #1: nam='db file sequential read' ela= 166 p1=2 p2=1 p3=1
FILE HEADER:
    Software vsn=153092096=0x92000000, Compatibility Vsn=134217728=0x80000000
    Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
    Activation ID=0=0x0
    Control Seq=2625=0xa41, File size=25600=0x6400
    File Number=2, Blksiz=8192, File Type=3 DATA
Tablespace #1 - UNDOTBS1 rel_fn:2
Creation at scn: 0x0000.0002dd31 05/12/2002 20:22:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 01/10/2012 13:35:35
status:0x0 root dba:0x00000000 chkpt cnt: 930 ctl cnt:929
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.01569c86 12/31/2011 17:45:38
thread:1 rba:(0x13d.ac56.0)

```

ORA-600 2758 错误解析

分析以上的恢复过程, 根据 RBA 的推进, 说明恢复过程是成功的, 实现了所有 REDO 的应用, 属于完全恢复。

但是当我们依次恢复了所有表空间之后，尝试 Open 数据库时，出现了如下错误：

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-00600: internal error code, arguments: [2758], [1], [4294967295], [204800],
[10], [], [], []
```

这个错误在 MOS 上没有相关说明，需要我们进一步的深入去分析其根本原因，通过猜测、进而深入，是学习和研究 Oracle 数据库技术的重要手段。

此时转储控制文件，查看检查点信息和数据文件信息，可以发现检查点和 On Disk RBA 信息还是正确的：

```
*****
CHECKPOINT PROGRESS RECORDS
*****

(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0x2 flags:0x0 dirty:72
low cache rba:(0x13d.9d72.0) on disk rba:(0x13d.ac56.0)
on disk scn: 0x0000.01569c86 12/31/2011 17:45:38
resetlogs scn: 0x0000.0002e872 06/12/2011 18:30:27
heartbeat: 772199262 mount id: 633916567
MTTR statistics status: 3
Init time: Avg: 9938392, Times measured: 3
File open time: Avg: 9695, Times measured: 39
Log block read time: Avg: 24, Times measured: 32770
Data block handling time: Avg: 2067, Times measured: 301
```

但是 REDO 记录的检查点信息出现了问题，注意检查点时间推进到 2012-01-10 13:41:37,但是 RBA 信息的块号显示为 ffffffff，这是最大值填充，显然是一个错误的信息：

```
*****
REDO THREAD RECORDS
*****

(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0xf thread links forward:0 back:0
```

```
#logs:3 first:1 last:3 current:1 last used seq#:0x13d
enabled at scn: 0x0000.0002e872 06/12/2011 18:30:27
disabled at scn: 0x0000.00000000 01/01/1988 00:00:00
opened at 01/10/2012 13:41:37 by instance ora9i
Checkpointed at scn: 0x0000.0156eaa9 01/10/2012 13:41:37
thread:1 rba:(0x13d.ffffffff.10)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
log history: 316
```

继续检查数据文件信息，以下为 SYSTEM 表空间内容，发现数据文件头信息的 RBA 也已经变更，同样块号被标记为 ffffffff，注意这里的检查点信息 0156eaa9，其十进制表示为 22473385：

```
DATA FILE #1:
  (name #8) D:\ORACLE\ORADATA\SXXHDTs\SYSTEM01.DBF
creation size=0 block size=8192 status=0xe head=8 tail=8 dup=1
tablespace 0, index=6 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
Checkpoint cnt:943 scn: 0x0000.0156eaa9 01/10/2012 13:41:37
Stop scn: 0xffff.ffffffff 12/31/2011 17:45:38
Creation Checkpointed at scn: 0x0000.0000000b 05/12/2002 16:17:58
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Offline scn: 0x0000.0002e871 prev_range: 0
Online Checkpointed at scn: 0x0000.0002e872 06/12/2011 18:30:27
thread:1 rba:(0x1.2.0)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
FILE HEADER:
  Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
  Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
  Activation ID=0=0x0
```

```
Control Seq=2638=0xa4e, File size=52480=0xcd00
File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM rel_fn:1
Creation at scn: 0x0000.00000000b 05/12/2002 16:17:58
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 12/31/2011 17:47:30
status:0x4 root dba:0x004001a1 chkpt cnt: 943 ctl cnt:941
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.0156eaa9 01/10/2012 13:41:37
thread:1 rba:(0x13d.ffffffff.10)
```

经过检查发现所有文件的 RBA 信息都指向了 0x13d.ffffffff.10,这里的 ffffffff 代表的是 4294967295,即 4G , 13d 代表的是 317 日志序列。

对于 ORA-600 错误, 其中的参数 [1] 代表的是日志组 1, 第二个参数 4294967295 代表的就是 RBA 的块号, 参数 4 代表的是偏移量:

```
SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-00600: internal error code, arguments: [2758], [1], [4294967295], [204800],
[10], [], [], []
```

那么 Redo 日志组 1 最终写到了什么位置呢?

通过如下命令转储日志文件, 将 AC55 之后的 REDO 信息转储出来:

```
SQL> alter system dump logfile 'D:\ORACLE\ORADATA\SXXHDS\REDO01.LOG' rba min 317 . 44117 ;
System altered.
```

可以看到 Redo 最后的记录写到日志序列 317 的 ac55 位置, 也就是说下一个 RBA 就正是 On-Disk-Rba 记录的 ac56 :

```
DUMP OF REDO FROM FILE 'D:\ORACLE\ORADATA\SXXHDS\REDO01.LOG'
Opcodes *.*
DBA's: (file # 0, block # 0) thru (file # 65534, block # 4194303)
RBA's: 0x00013d.0000ac55.0000 thru 0xffffffff.ffffffff.ffff
SCN's scn: 0x0000.00000000 thru scn: 0xffff.ffffffff
```

Times: creation thru eternity

FILE HEADER:

Software vsn=153092096=0x9200000, Compatibility Vsn=153092096=0x9200000

Db ID=615401347=0x24ae4783, Db Name='SXXHDS'

Activation ID=615377539=0x24adea83

Control Seq=2618=0xa3a, File size=204800=0x32000

File Number=1, Blksiz=512, File Type=2 LOG

descrip:"Thread 0001, Seq# 0000000317, SCN 0x00000155b0f0-0xffffffffffff"

thread: 1 nab: 0xffffffff seq: 0x0000013d hws: 0x2 eot: 1 dis: 0

reset logs count: 0x2cebbf43 scn: 0x0000.0002e872

Low scn: 0x0000.0155b0f0 12/31/2011 08:28:19

Next scn: 0xffff.ffffffff 01/01/1988 00:00:00

Enabled scn: 0x0000.0002e872 06/12/2011 18:30:27

Thread closed scn: 0x0000.0155b0f0 12/31/2011 08:28:19

Log format vsn: 0x8000000 Disk cksum: 0x13d8 Calc cksum: 0x13d8

Terminal Recovery Stamp scn: 0x0000.00000000 01/01/1988 00:00:00

Most recent redo scn: 0x0000.00000000

Largest LWN: 0 blocks

End-of-redo stream : No

Unprotected mode

Miscellaneous flags: 0x0

CHANGE #1 MEDIA RECOVERY MARKER SCN:0x0000.00000000 SEQ: 0 OP:17.3

Crash Recovery at scn: 0x0000.0155b0ef

REDO RECORD - Thread:1 RBA: 0x00013d.0000ac55.0090 LEN: 0x0054 VLD: 0x01

SCN: 0x0000.01569c85 SUBSCN: 1 12/31/2011 17:45:38

CHANGE #1 TYP:0 CLS:17 AFN:2 DBA:0x00800009 SCN:0x0000.01569c84 SEQ: 1 OP:5.4

ktucm redo: slt: 0x0006 sqn: 0x00004336 srt: 0 sta: 9 flg: 0x2

ktucf redo: uba: 0x00800370.01ef.34 ext: 2 spc: 4390 fbi: 0

END OF REDO DUMP

----- Redo read statistics for thread 1 -----

Read rate (ASYNC): 22058Kb in 0.92s => 22.83 Mb/sec

Longest record: 1Kb, moves: 4/91535 (0%)

```
Change moves: 31544/212533 (14%), moved: 2Mb
```

以上这些输出和控制文件中记录的信息想符合，问题在于最后打开数据库时，检查点 RBA 被异常增进到 ffffffff,这就导致了异常，如果我们将检查点的信息回退到 ac56 位置，则错误应当可以被消除。

Oracle 控制文件中记录的 On Disk RBA 指向 REDO 中最后一个 RBA 的下一个地址，此时重做日志中最后一个 RBA 是 0x00013d.0000ac55.0090 ,因此下一个地址可以将 RBA 指向 ac56.读者可以通过正常数据库的记录分析一下 RBA 和文件头的信息相关性。

通过 BBED 解决 ORA-600 错误

使用 BBED 可以修改控制文件中的检查点信息：

```
E:\>bbed parfile=par.txt
```

```
口令:blockedit
```

```
BBED: Release 2.0.0.0.0 - Limited Production on 星期二 1月 10 16:00:34 2012
Copyright (c) 1982, 2002, Oracle Corporation. All rights reserved.
```

```
***** !!! For Oracle Internal Use only !!! *****
```

找到第七个数据块：

```
BBED> set file 11 block 7
```

```
FILE#          11
BLOCK#         7
```

```
BBED> dump
```

```
File: D:\Oracle\ORADATA\SXHXDTS\CONTROL03.CTL (11)
```

```
Block: 7          Offsets:  0 to  511          Dbf:0x02c00007
```

```
15020000 07000000 4e0a0000 ffff0104 e6320000 0f000000 a9ea5601 00000000
11ee062e 01000000 3d010000 ffffffff 10001804 02000000 00000000 00000000
03000100 03000100 3d010000 72e80200 00000000 43bfeb2c 00000000 00000000
00000000 3c010000 6f726139 69007300 00000000 00000000 11ee062e 00000000
```

找到全 f 的 RBA 标志位:

```
BBED> find /x ffffffff
```

```
File: D:\Oracle\ORADATA\SXHXDTS\CONTROL03.CTL (11)
```

```
Block: 7                      Offsets: 44 to 555                      DbA:0x02c00007
```

```
-----  
ffffff 10001804 02000000 00000000 00000000 03000100 03000100 3d010000  
72e80200 00000000 43bfef2c 00000000 00000000 00000000 3c010000 6f726139  
69007300 00000000 00000000 11ee062e 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

将该参数值修改为 ac56，由于换位存储，实际修改为 56ac：

```
BBED> modify /x 56ac0000
```

```
Warning: contents of previous BIFILE will be lost. Proceed? (Y/N) Y
```

```
File: D:\Oracle\ORADATA\SXHXDTS\CONTROL03.CTL (11)
```

```
Block: 7                      Offsets: 44 to 555                      DbA:0x02c00007
```

```
-----  
56ac0000 10001804 02000000 00000000 00000000 03000100 03000100 3d010000  
72e80200 00000000 43bfef2c 00000000 00000000 00000000 3c010000 6f726139  
69007300 00000000 00000000 11ee062e 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

修改后的数据块会标记为损坏，需要重新计算校验位并应用：

```
BBED> verify
```

```
DBVERIFY - 验证正在启动
```

```
FILE =D:\Oracle\ORADATA\SXHXDTS\CONTROL03.CTL
```

```
BLOCK = 7
```

```
块 7 已毁坏
```

```
***
```

```
Corrupt block relative dba: 0x00000007 (file 0, block 7)
```

```
Bad check value found during verification
```

```
Data in bad block -
```

```

type: 21 format: 2 rdba: 0x00000007
last change scn: 0xffff.00000a4e seq: 0x1 flg: 0x04
consistency value in tail: 0x0a4e1501
check value in block header: 0x32e6, computed block checksum: 0xac56
spare1: 0x0, spare2: 0x0, spare3: 0x0
***

```

DBVERIFY - 验证完成

检查的总块数: 1

已处理的总块数 (数据): 0

无法处理的总块数 (数据): 0

已处理的总块数 (索引): 0

无法处理的总块数 (索引): 0

空的总块数: 0

标记为损坏的总数块: 1

汇入的块总数: 0

BBED> sum apply

Check value for File 11, Block 7:

current = 0x9eb0, required = 0x9eb0

然后来尝试启动数据库, 注意此时数据库可以顺利被打开, 不再需要执行任何恢复或者日志重置:

```
SQL> startup nomount pfile=initora9i.ora
```

ORACLE instance started.

```

Total System Global Area 126950956 bytes
Fixed Size                 454188 bytes
Variable Size              92274688 bytes
Database Buffers          33554432 bytes
Redo Buffers               667648 bytes

```

```
SQL> alter database mount;
```

Database altered.

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

Database altered.

最后再来查询一下文件头信息内容:

```
SQL> alter session set events 'immediate trace name file_hdrs level 12';
```

Session altered.

可以看到现在文件头上的检查点信息全部恢复了正常:

FILE HEADER:

Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000

Db ID=615401347=0x24ae4783, Db Name='SXXHDS'

Activation ID=0=0x0

Control Seq=2647=0xa57, File size=52480=0xcd00

File Number=1, Blksiz=8192, File Type=3 DATA

Tablespace #0 - SYSTEM rel_fn:1

Creation at scn: 0x0000.00000000b 05/12/2002 16:17:58

Backup taken at scn: 0x0000.000000000 01/01/1988 00:00:00 thread:0

reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 01/10/2012 16:01:22

status:0x4 root dba:0x004001a1 chkpt cnt: 947 ctl cnt:946

begin-hot-backup file size: 0

Checkpointed at scn: 0x0000.015738cc 01/10/2012 16:01:22

thread:1 rba:(0x13e.2.10)

告警日志信息解析

在告警日志文件中, 记录了数据库打开的过程:

Tue Jan 10 16:01:22 2012

alter database open

Tue Jan 10 16:01:22 2012

Beginning crash recovery of 1 threads

Tue Jan 10 16:01:22 2012

Started redo scan

Tue Jan 10 16:01:22 2012

Completed redo scan

```
0 redo blocks read, 0 data blocks need recovery
```

注意此处明确提示,数据库恢复从日志序列 317 开始,日志块 44118(16 进制就是 ac56),SCN 为 22473385,也正是控制文件中记录的 0x156EAA9。

所以我们说,告警日志中记录的每一条记录都有其重要含义,如果能够深入分析,研究清楚其中的内容,那么我们的 Oracle 学习和深入就可以变得相当容易。

```
Tue Jan 10 16:01:22 2012
Started recovery at
  Thread 1: logseq 317, block 44118, scn 0.22473385
Tue Jan 10 16:01:22 2012
Recovery of Online Redo Log: Thread 1 Group 1 Seq 317 Reading mem 0
  Mem# 0 errs 0: D:\ORACLE\ORADATA\SXXHDTs\RED001.LOG
Tue Jan 10 16:01:22 2012
Completed redo application
```

接下来日志记录了恢复终点,注意由于 44118 是 ac56,最后一个 RBA 块,所以恢复就此停止,而 SCN 序列号被推进了 1 位,不知道诸位读者是否会拍案叫绝,Oracle 数据库就是如此精密,如此句句言之有物:

```
Tue Jan 10 16:01:22 2012
Ended recovery at
  Thread 1: logseq 317, block 44118, scn 0.22493386
  0 data blocks read, 0 data blocks written, 0 redo blocks read
Crash recovery completed successfully
Tue Jan 10 16:01:22 2012
Thread 1 advanced to log sequence 318
Thread 1 opened at log sequence 318
  Current log# 2 seq# 318 mem# 0: D:\ORACLE\ORADATA\SXXHDTs\RED002.LOG
Successful open of redo thread 1
Tue Jan 10 16:01:22 2012
SMON: enabling cache recovery
Tue Jan 10 16:01:23 2012
Successfully onlined Undo Tablespace 1.
Tue Jan 10 16:01:23 2012
SMON: enabling tx recovery
Tue Jan 10 16:01:23 2012
```

```

Database Characterset is ZHS16GBK
Updating 9.2.0.1.0 NLS parameters in sys.props$
-- adding 9.2.0.8.0 NLS parameters.
replication_dependency_tracking turned off (no async multimaster replication found)
Completed: alter database open
Tue Jan 10 16:02:27 2012
Restarting dead background process QMN0
QMN0 started with pid=9, OS id=4072

```

对于这种情况，主要的问题在于控制文件的信息存在缺失，使得恢复的终点判断有误，写入了错误的 RBA 信息。

重建控制文件恢复

如果通过重建控制文件，清除控制文件中记录的检查点信息，其余内容从数据文件头上提取，则可以将恢复推进到日志文件尾部，完成完全恢复。

以下是重建控制文件的过程：

```

Tue Jan 10 16:36:36 2012
Starting ORACLE instance (normal)
Tue Jan 10 16:36:37 2012
CREATE CONTROLFILE REUSE DATABASE "SXXHDTs" NORESETLOGS  NOARCHIVELOG
--  SET STANDBY TO MAXIMIZE PERFORMANCE
    MAXLOGFILES 50
    MAXLOGMEMBERS 5
    MAXDATAFILES 100
    MAXINSTANCES 1
    MAXLOGHISTORY 226
LOGFILE
  GROUP 1 'D:\ORACLE\ORADATA\SXXHDTs\RED001.LOG'  SIZE 100M,
  GROUP 2 'D:\ORACLE\ORADATA\SXXHDTs\RED002.LOG'  SIZE 100M,
  GROUP 3 'D:\ORACLE\ORADATA\SXXHDTs\RED003.LOG'  SIZE 100M
-- STANDBY LOGFILE
DATAFILE

```

```
'D:\ORACLE\ORADATA\SXXHDS\SYSTEM01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\UNDOTBS01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\CWMLITE01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\DRSYS01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\EXAMPLE01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\INDX01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\ODM01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\TOOLS01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\USERS01.DBF',
'D:\ORACLE\ORADATA\SXXHDS\XDB01.DBF'
```

CHARACTER SET ZHS16GBK

Tue Jan 10 16:36:37 2012

Successful mount of redo thread 1, with mount id 634003285

Tue Jan 10 16:36:37 2012

Completed: CREATE CONTROLFILE REUSE DATABASE "SXXHDS" NORESE

重建后控制文件主要内容如下:

*** 2012-01-10 16:36:51.549

*** SESSION ID:(9.1) 2012-01-10 16:36:51.539

DUMP OF CONTROL FILES, Seq # 2627 = 0xa43

FILE HEADER:

Software vsn=153092096=0x92000000, Compatibility Vsn=134217728=0x80000000

Db ID=615401347=0x24ae4783, Db Name='SXXHDS'

Activation ID=0=0x0

Control Seq=2627=0xa43, File size=278=0x116

File Number=0, Blksiz=8192, File Type=1 CONTROL

以下数据库条目中的检查点信息来自于日志文件, 其检查点 SCN 为 0155b0f0, 以下内容都是重新构建出来的:

DATABASE ENTRY

(blkno = 0x1, size = 192, max = 1, in-use = 1, last-recid= 0)

```

DF Version: creation=0x9200000 compatible=0x8000000, Date 01/10/2012 16:36:37
DB Name "SXXHDTs"
Database flags = 0x00400102
Controlfile Creation Timestamp 01/10/2012 16:36:37
Incplmt recovery scn: 0x0000.00000000
Resetlogs scn: 0x0000.0002e872 Resetlogs Timestamp 06/12/2011 18:30:27
Prior resetlogs scn: 0x0000.00000001 Prior resetlogs Timestamp 05/12/2002 16:16:56
Redo Version: creation=0x9200000 compatable=0x9200000
#Data files = 10, #Online files = 10
Database checkpoint: Thread=1 scn: 0x0000.0155b0f0
Threads: #Enabled=1, #Open=1, Head=1, Tail=1
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Max log members = 5, Max data members = 1
Arch list: Head=0, Tail=0, Force scn: 0x0000.00000000scn: 0x0000.00000000
Controlfile Checkpointed at scn: 0x0000.00000000
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000

```

重建控制文件之后，检查点记录部分被清空，这部分信息无从获得，恢复将依据控制文件进行推进，以下是检查点进程的信息记录：

```

*****
CHECKPOINT PROGRESS RECORDS
*****
(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0x0 flags:0x0 dirty:0
low cache rba:(0x0.0.0) on disk rba:(0x0.0.0)
on disk scn: 0x0000.00000000 01/01/1988 00:00:00
resetlogs scn: 0x0000.00000000 01/01/1988 00:00:00
heartbeat: 772257562 mount id: 634003285
MTTR statistics status: 1
Init time: Avg: 0, Times measured: 0
File open time: Avg: 100000, Times measured: 1

```

Log block read time: Avg: 20, Times measured: 1

Data block handling time: Avg: 558, Times measured: 1

以下是日志检查点信息,注意这里的检查点信息能够正确获得,其信息来自 REDO 日志文件,检查点为 0155b0f0,这个内容对应到了 REDO 日志头块的 SCN,RBA 的块号显示为 2,也就是头块之后的第一个日志块:

```
*****
REDO THREAD RECORDS
*****

(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0x7 thread links forward:0 back:0
#logs:3 first:1 last:3 current:1 last used seq#:0x13d
enabled at scn: 0x0000.0002e872 06/12/2011 18:30:27
disabled at scn: 0x0000.00000000 01/01/1988 00:00:00
opened at 01/01/1988 00:00:00 by instance
Checkpointed at scn: 0x0000.0155b0f0 12/31/2011 08:28:19
thread:1 rba:(0x13d.2.0)
enabled threads: 01000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
log history: 0
```

接下来检查数据文件信息,来自文件头的信息尚未有变化,其信息由文件头提出,所以固有不变:

DATA FILE #1:

```
(name #13) D:\ORACLE\ORADATA\SXXHDT\SYSTEM01.DBF
creation size=0 block size=8192 status=0x12 head=13 tail=13 dup=1
tablespace 0, index=1 krfil=1 prev_file=0
unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
Checkpoint cnt:942 scn: 0x0000.0156eaa8 12/31/2011 17:47:31
Stop scn: 0xffff.ffffffff 01/10/2012 16:36:37
Creation Checkpointed at scn: 0x0000.0000000b 05/12/2002 16:17:58
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Offline scn: 0x0000.00000000 prev_range: 0
Online Checkpointed at scn: 0x0000.00000000
```

```

thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
FILE HEADER:
    Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
    Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
    Activation ID=0=0x0
    Control Seq=2624=0xa40, File size=52480=0xcd00
    File Number=1, Blksiz=8192, File Type=3 DATA
Tablespace #0 - SYSTEM rel_fn:1
Creation at scn: 0x0000.0000000b 05/12/2002 16:17:58
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 12/31/2011 17:47:30
status:0x4 root dba:0x004001a1 chkpt cnt: 942 ctl cnt:941
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.0156eaa8 12/31/2011 17:47:31
thread:1 rba:(0x13e.2.10)

```

以下为 UNDO 表空间的文件头信息，其内容与之前一致：

```

DATA FILE #2:
    (name #12) D:\ORACLE\ORADATA\SXXHDTs\UNDOTBS01.DBF
creation size=0 block size=8192 status=0x12 head=12 tail=12 dup=1
tablespace 1, index=2 krfil=2 prev_file=0
unrecoverable scn: 0x0000.00000000 01/01/1988 00:00:00
Checkpoint cnt:927 scn: 0x0000.0155b0f1 12/31/2011 08:28:19
Stop scn: 0xffff.ffffffff 01/10/2012 16:36:37
Creation Checkpointed at scn: 0x0000.0002dd31 05/12/2002 20:22:54
thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Offline scn: 0x0000.00000000 prev_range: 0
Online Checkpointed at scn: 0x0000.00000000

```

```

thread:0 rba:(0x0.0.0)
enabled threads: 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000
Hot Backup end marker scn: 0x0000.00000000
aux_file is NOT DEFINED
FILE HEADER:
    Software vsn=153092096=0x9200000, Compatibility Vsn=134217728=0x8000000
    Db ID=615401347=0x24ae4783, Db Name='SXXHDTs'
    Activation ID=0x0
    Control Seq=2618=0xa3a, File size=25600=0x6400
    File Number=2, Blksiz=8192, File Type=3 DATA
Tablespace #1 - UNDOTBS1 rel_fn:2
Creation at scn: 0x0000.0002dd31 05/12/2002 20:22:54
Backup taken at scn: 0x0000.00000000 01/01/1988 00:00:00 thread:0
reset logs count:0x2cebbf43 scn: 0x0000.0002e872 recovered at 12/31/2011 08:28:19
status:0x4 root dba:0x00000000 chkpt cnt: 927 ctl cnt:926
begin-hot-backup file size: 0
Checkpointed at scn: 0x0000.0155b0f1 12/31/2011 08:28:19
thread:1 rba:(0x13d.2.10)

```

此时尝试直接 Open，遇到 ORA-01110 错误，提示 SYSTEM 表空间需要恢复，实际上整个数据库都需要向前推演恢复：

```

SQL> alter database open;
alter database open
*
ERROR at line 1:
ORA-01113: file 1 needs media recovery
ORA-01110: data file 1: 'D:\ORACLE\ORADATA\SXXHDTs\SYSTEM01.DBF'

```

执行恢复后数据库可以直接 Open 打开：

```

SQL> recover database;
Media recovery complete.
SQL> alter session set events 'immediate trace name CONTROLF level 12';
Session altered.

```

```
SQL> alter session set events 'immediate trace name file_hdrs level 12';
Session altered.

SQL> alter database open;
Database altered.
```

这样就完成了恢复，告警日志中记录了如下过程：

```
Tue Jan 10 16:41:59 2012
Started redo scan
Tue Jan 10 16:41:59 2012
Completed redo scan
  44116 redo blocks read, 0 data blocks need recovery
```

这里的恢复由 317 日志开始，SCN 为 22393072

```
Tue Jan 10 16:41:59 2012
Started recovery at
  Thread 1: logseq 317, block 2, scn 0.22393072
Tue Jan 10 16:41:59 2012
Recovery of Online Redo Log: Thread 1 Group 1 Seq 317 Reading mem 0
  Mem# 0 errs 0: D:\ORACLE\ORADATA\SXXHDTs\RED001.LOG
Tue Jan 10 16:41:59 2012
Completed redo application
```

恢复的终点是块 44118，SCN 22473382

```
Tue Jan 10 16:41:59 2012
Ended recovery at
  Thread 1: logseq 317, block 44118, scn 0.22473382
  0 data blocks read, 0 data blocks written, 44116 redo blocks read
Crash recovery completed successfully
Tue Jan 10 16:41:59 2012
Thread 1 advanced to log sequence 318
Thread 1 opened at log sequence 318
  Current log# 2 seq# 318 mem# 0: D:\ORACLE\ORADATA\SXXHDTs\RED002.LOG
Successful open of redo thread 1
Tue Jan 10 16:41:59 2012
```

```

SMON: enabling cache recovery
Tue Jan 10 16:41:59 2012
Successfully onlined Undo Tablespace 1.
Dictionary check beginning
Tablespace 'TEMP' #2 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Dictionary check complete
Tue Jan 10 16:41:59 2012
SMON: enabling tx recovery
Tue Jan 10 16:41:59 2012
Database Characterset is ZHS16GBK
Updating 9.2.0.1.0 NLS parameters in sys.props$
-- adding 9.2.0.8.0 NLS parameters.
replication_dependency_tracking turned off (no async multimaster replication found)
Completed: alter database open

```

技术提示

关于 Low Cache RBA 和 On Disk RBA，以下是一点补充说明，从动态视图 X\$KCCP 可以获得检查点信息及 RBA 进程进度信息：

```

SQL> |
1  SELECT cpdrt,
2  cplrba_seq || '.' || cplrba_bno || '.' || cplrba_bof "low_cache_rba",
3  cpodr_seq || '.' || cpodr_bno || '.' || cpodr_bof "on_disk_rba"
4* FROM x$kcccp where cplrba_seq >0
SQL> /

```

CPDRT	low_cache_	on_disk_rb
205	208.2625.0	208.4721.0

转储控制文件，其中的检查点记录如下：

```

*****
CHECKPOINT PROGRESS RECORDS

```

```

*****
(size = 8180, compat size = 8180, section max = 11, section in-use = 0,
  last-recid= 0, old-recno = 0, last-recno = 0)
(extent = 1, blkno = 2, numrecs = 11)
THREAD #1 - status:0x2 flags:0x0 dirty:205
low cache rba:(0xd0.a41.0) on disk rba:(0xd0.1271.0)
on disk scn: 0x0000.00417af3 01/16/2012 17:47:58
resetlogs scn: 0x0000.00080634 12/06/2011 14:58:40
heartbeat: 772248185 mount id: 1299842038

```

以上两者信息一致:

```

SQL> select to_number('d0','xxx') seq,
          2 to_number('a41','xxx') lowblock,to_number('1271','xxx') highblock
          3 from dual;

```

SEQ	LOWBLOCK	HIGHBLOCK
208	2625	4721

从小恙到灾难

重建控制文件失误导致的故障

在 2012 新年后的第一个工作日，我们又接收到一则客户服务请求，又一次数据灾难发生。这是又一次我们称为从感冒治疗成癌症的过程。原本并不复杂的一次数据库故障，在经过了错误的处理之后，数据库的小病变成了重恙。

灾难描述

这次灾难是这样发生的：

1. 客户数据库因为断电而崩溃
2. 服务器重启后控制文件出现不一致错误
3. 服务商通过重建控制文件，试图恢复控制文件的一致性
4. 通过重置日志进行数据库强制打开
5. 打开数据库后实例崩溃
6. 数据库无法正常打开，并且发现重建控制文件时遗失了大量文件
7. 灾难形成

重建控制文件失误，这是最基本技能的缺失。

案例警示

这个案例对我们的警示是：

1. **技术人员的基本功必须扎实可靠**

如果技术人员没有扎实的基本功，则可能在简单的问题上犯下错误，这些简单的错误有可能积累最终致命。

在这个案例中，如果技术人员能够正确的创建控制文件，不遗漏数据文件，则灾难就不至于形成。

2. 技术专家的推理判断必须能够还原事实真相

对于技术专家，必须能够准确判断故障的成因，逐步推理，解释清除，并且要能够从现场搜集证据，支持自己的判断。

专家的作用在于清晰的解构案情的来龙去脉，从而做出正确判断，避免走不必要的弯路。唯有正确的判断，才能有正确的处理。

3. 在完成技术分析之前不要草率进行恢复尝试

在面对灾难时，不要急于进行恢复尝试，首先应该在不改变、不破坏环境的情况下，进行充分的技术分析，根据分析结果确定恢复方案，避免草率的尝试。

当然，在进行实际操作之前，必须做好备份工作。

再一次提示大家，无知者不能够无畏。

技术回放

这个数据库崩溃于 2011 年 12 月 30 日，同样是一个倒在 2012 前的数据库。让我们看一看灾难的形成过程。

以下是用户重建控制文件的过程，注意这个创建脚本是用户手工编写的，其中仅仅指定了一个数据文件，即 SYSTEM 表空间：

```
Sat Jan 28 19:38:04 2012
CREATE CONTROLFILE REUSE DATABASE "ORCL" RESETLOGS NOARCHIVELOG
LOGFILE
  GROUP 1 'c:\z1\RED001.LOG' SIZE 100M,
  GROUP 2 'c:\z1\RED002.LOG' SIZE 100M,
  GROUP 3 'c:\z1\RED003.LOG' SIZE 100M
DATAFILE
  'H:\2011\oradata\orcl\SYSTEM01.DBF'
CHARACTER SET ZHS16GBK
```

```
Sat Jan 28 19:38:04 2012
Successful mount of redo thread 1, with mount id 1301551900.
```

Sat Jan 28 19:38:04 2012

Completed: CREATE CONTROLFILE REUSE DATABASE "ORCL" RESETLOGS

这个创建是成功的，我们推测，用户或许认为，SYSTEM 表空间是数据库的核心，重建控制文件包含一个文件即可，但是显然这个想法是错误的。

此后，用户在参数文件中设置了一系列的隐含参数，用于强制 Open 打开数据库：

SYS auditing is disabled

Starting up ORACLE RDBMS Version: 9.2.0.1.0.

System parameters with non-default values:

```

processes                = 500
timed_statistics         = TRUE
event                   = 10513 trace name context forever,level 10,
                        10512 trace name context forever,level 10,
                        10511 trace name context forever,level 10,
                        10510 trace name context forever,level 10,
                        10015 trace name context forever, level 10,
                        10269 trace name context forever, level 10,
                        10061 trace name context forever, level 10

shared_pool_size        = 134217728
large_pool_size         = 268435456
java_pool_size          = 33554432
control_files           = c:\zl\control01.ctl
db_block_size           = 8192
db_cache_size           = 629145600
compatible              = 9.2.0.0.0
db_file_multiblock_read_count= 16
fast_start_mttr_target  = 300
_allow_resetlogs_corruption= TRUE
_allow_read_only_corruption= TRUE
_offline_rollback_segments= _SYSSMU1$, _SYSSMU2$, _SYSSMU3$, _SYSSMU4$, _SYSSMU5$,
_SYSSMU6$, _SYSSMU7$, _SYSSMU8$, _SYSSMU9$, _SYSSMU10$
_corrupted_rollback_segments= _SYSSMU1$, _SYSSMU2$, _SYSSMU3$, _SYSSMU4$, _SYSSMU5$,
_SYSSMU6$, _SYSSMU7$, _SYSSMU8$, _SYSSMU9$, _SYSSMU10$

```

```

undo_management          = MANUAL
undo_tablespace          = system
undo_retention           = 0
remote_login_passwordfile= NONE
db_domain                =
global_names             = FALSE
instance_name            = orcl
dispatchers              = (PROTOCOL=TCP) (SERVICE=orclXDB)
job_queue_processes      = 10
_system_trig_enabled     = FALSE

```

在之后强制打开数据库时，Oracle 进行自动的数据字典检查，发现大量控制文件中没有的数据文件，并强制将这些文件添加到控制文件中：

```

Sat Jan 28 19:44:35 2012
SMON: enabling cache recovery
Sat Jan 28 19:44:35 2012
Dictionary check beginning
<注意以下文件被添加入控制文件，删减部分信息>
Tablespace 'UNDOTBS1' #1 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'TEMP' #2 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'CWMLITE' #3 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'DIGITALSCANDATA' #12 found in data dictionary,
but not in the controlfile. Adding to controlfile.
Tablespace 'DIGITALSCANDB' #13 found in data dictionary,
but not in the controlfile. Adding to controlfile.
<注意以下系列文件，被以 Offline 离线方式加入到数据库中>
File #2 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00002' in the controlfile.
This file can no longer be recovered so it must be dropped.
File #3 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00003' in the controlfile.

```

```

This file can no longer be recovered so it must be dropped.
File #4 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00004' in the controlfile.
<注意，数据库提示这些文件不能被恢复，只能被删除>
File #32 found in data dictionary but not in controlfile.
Creating OFFLINE file 'MISSING00032' in the controlfile.
This file can no longer be recovered so it must be dropped.
<受限于创建控制文件的 MAXDATAFILES 参数设置，不能添加更多的文件到数据库中，添加数据文件工作中断>
Sat Jan 28 19:44:36 2012
Errors in file f:\oracle\admin\orcl\udump\orcl_ora_2056.trc:
ORA-01176: 控制文件允许数据字典具有多于 32 个文件

```

```

Error 1176 happened during db open, shutting down database
USER: terminating instance due to error 1176
Instance terminated by USER, pid = 2056
ORA-1092 signalled during: alter database open resetlogs...

```

遗憾的是，这些信息并未引起用户的重视，进一步的强制打开被尝试：

```

Sat Jan 28 19:46:31 2012
Database mounted in Exclusive Mode.
Completed: ALTER DATABASE MOUNT
Sat Jan 28 19:49:43 2012
alter database open resetlogs
Sat Jan 28 19:49:43 2012
ORA-1139 signalled during: alter database open resetlogs...
Sat Jan 28 19:50:53 2012
ALTER DATABASE RECOVER database until cancel
<在此执行 SYSTEM 表空间的恢复>
Sat Jan 28 19:50:53 2012
Media Recovery Start
Starting datafile 1 recovery in thread 1 sequence 1
Datafile 1: 'H:\2011\ORADATA\ORCL\SYSTEM01.DBF'
Media Recovery Log
ORA-279 signalled during: ALTER DATABASE RECOVER database until cancel ...

```

```

Sat Jan 28 19:51:03 2012
ALTER DATABASE RECOVER      CANCEL
Sat Jan 28 19:51:03 2012
ORA-1547 signalled during: ALTER DATABASE RECOVER      CANCEL  ...
Sat Jan 28 19:51:03 2012
ALTER DATABASE RECOVER CANCEL
ORA-1112 signalled during: ALTER DATABASE RECOVER CANCEL ...
<用户发出 resetlog 命令强制打开数据库>
Sat Jan 28 19:51:27 2012
alter database open resetlogs
Sat Jan 28 19:51:27 2012
RESETLOGS is being done without consistency checks. This may result
in a corrupted database. The database should be recreated.
<强制打开的 SCN 时间点>
RESETLOGS after incomplete recovery UNTIL CHANGE 337553167
Resetting resetlogs activation ID 1301553802 (0x4d94228a)
Sat Jan 28 19:51:46 2012
Assigning activation ID 1301534482 (0x4d93d712)
<日志被重置，日志序列从 1 开始重新编号>
Thread 1 opened at log sequence 1
    Current log# 3 seq# 1 mem# 0: C:\ZL\RED003.LOG
Successful open of redo thread 1.
Sat Jan 28 19:51:46 2012
SMON: enabling cache recovery
Sat Jan 28 19:51:46 2012
<数据字典检查，发现表空间离线>
Dictionary check beginning
File #2 is offline, but is part of an online tablespace.
data file 2: 'E:\ORACLE\DATABASE\MISSING000002'
File #3 is offline, but is part of an online tablespace.
data file 3: 'E:\ORACLE\DATABASE\MISSING000003'
File #4 is offline, but is part of an online tablespace.
.....

```

```
File #32 is offline, but is part of an online tablespace.
data file 32: 'E:\ORACLE\DATABASE\MISSING00032'
Sat Jan 28 19:51:46 2012
Errors in file f:\oracle\admin\orcl\udump\orcl_ora_3892.trc:
ORA-01176: 控制文件允许数据字典具有多于 32 个文件
<数据库 Crash, 无法正常开启>
Error 1176 happened during db open, shutting down database
USER: terminating instance due to error 1176
Instance terminated by USER, pid = 3892
ORA-1092 signalled during: alter database open resetlogs...
```

在之前的尝试中, ORA-00600 的 4194 错误也是常见的:

```
Fri Jan 20 02:02:04 2012
Errors in file e:\oracle\admin\orcl\udump\orcl_ora_4796.trc:
ORA-00600: 内部错误代码, 参数: [4194], [34], [26], [], [], [], [], []

Fri Jan 20 02:02:04 2012
Recovery of Online Redo Log: Thread 1 Group 3 Seq 3 Reading mem 0
  Mem# 0 errs 0: F:\ORACLE\ORADATA\ORCL\RED003.LOG
Recovery of Online Redo Log: Thread 1 Group 3 Seq 3 Reading mem 0
  Mem# 0 errs 0: F:\ORACLE\ORADATA\ORCL\RED003.LOG
Fri Jan 20 02:02:05 2012
Errors in file e:\oracle\admin\orcl\udump\orcl_ora_4796.trc:
ORA-00604: 递归 SQL 层 1 出现错误
ORA-00607: 当更改数据块时出现内部错误
ORA-00600: 内部错误代码, 参数: [4194], [34], [26], [], [], [], [], []
```

```
Error 604 happened during db open, shutting down database
USER: terminating instance due to error 604
Fri Jan 20 02:02:05 2012
Errors in file e:\oracle\admin\orcl\bdump\orcl_pmon_5704.trc:
ORA-00604: error occurred at recursive SQL level
```

在这个案例的处理过程中, 主要的错误来自于控制文件的创建, 如果我们判断是控制文件的损坏, 则完全

可以用正确的脚本重建，并且使用 Noresetlogs 方式来尝试恢复数据，打开数据库，如果日志文件完好，这个恢复会非常的顺利。

以下让我们来分析一下这个过程。

尝试启动挂载数据库，可以发现数据库提供控制文件不一致，其中 1 号控制文件比较新，2 号控制文件陈旧，数据库因为控制文件不一致无法 Mount：

```
SQL> startup mount;
ORACLE instance started.

Total System Global Area  126950956 bytes
Fixed Size                  454188 bytes
Variable Size              92274688 bytes
Database Buffers          33554432 bytes
Redo Buffers                667648 bytes
ORA-00214: controlfile 'C:\ORACLE\ORADATA\ORA9I\CONTROL01.CTL' version 121362
inconsistent with file 'C:\ORACLE\ORADATA\ORA9I\CONTROL02.CTL' version 121347
```

使用 10046 事件跟踪一下数据库 Mount 的后台过程：

```
SQL> alter session set events '10046 trace name context forever,level 12';
Session altered.

SQL> alter database mount;
alter database mount
*
ERROR at line 1:
ORA-00214: controlfile 'C:\ORACLE\ORADATA\ORA9I\CONTROL01.CTL' version 121362
inconsistent with file 'C:\ORACLE\ORADATA\ORA9I\CONTROL02.CTL' version 121347
```

摘录后台的跟踪文件，与前面的案例类似，可以看到在控制文件一致性判断的过程中，仅读取控制文件的第一个数据块即可完成：

```
PARSING IN CURSOR #1 len=20 dep=0 uid=0 oct=35 lid=0 tim=42638703883 hv=1379354989
ad='6a3c8b8c'
alter database mount
END OF STMT
PARSE #1:c=0,e=1324,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=42638703877
```

```

BINDS #1:
WAIT #1: nam='rdbms ipc reply' ela= 3 p1=5 p2=2147483647 p3=0
WAIT #1: nam='rdbms ipc reply' ela= 331 p1=5 p2=900 p3=0
WAIT #1: nam='rdbms ipc reply' ela= 2102 p1=5 p2=900 p3=0
WAIT #1: nam='control file sequential read' ela= 276 p1=0 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 191 p1=1 p2=1 p3=1
WAIT #1: nam='control file sequential read' ela= 210 p1=2 p2=1 p3=1

```

由于数据库无法 Mount，此时无法生成创建控制文件的脚本，我们尝试使用单一的控制文件进行加载数据库：

```

SQL> startup mount;
ORACLE instance started.

```

```

Total System Global Area 126950956 bytes
Fixed Size                  454188 bytes
Variable Size               92274688 bytes
Database Buffers           33554432 bytes
Redo Buffers                667648 bytes
Database mounted.

```

```
SQL> show parameter control
```

NAME	TYPE	VALUE
control_file_record_keep_time	integer	7
control_files	string	c:\oracle\oradata\ora9i\contro l01.ctl

此时查询 v\$datafile 视图，会发现数据库提示 ORA-00235 错误，控制文件由于并发更新而不允许查询，这说明控制文件上存在不一致的事务，更新状态未完成：

```

SQL> select count(*) from v$datafile;
select count(*) from v$datafile
          *

```

```
ERROR at line 1:
```

```
ORA-00235: controlfile fixed table inconsistent due to concurrent update
```

但是这不妨碍生成创建控制文件的脚本，以下一条命令就可以获得准确无误的控制文件创建脚本：

```
SQL> alter database backup controlfile to trace;  
Database altered.
```

对于这个数据库，摘录一下重要的信息：

```
CREATE CONTROLFILE REUSE DATABASE "ORCL" NORESETLOGS NOARCHIVELOG  
-- SET STANDBY TO MAXIMIZE PERFORMANCE  
    MAXLOGFILES 50  
    MAXLOGMEMBERS 5  
    MAXDATAFILES 100  
    MAXINSTANCES 1  
    MAXLOGHISTORY 14520  
LOGFILE  
    GROUP 1 'F:\ORACLE\ORADATA\ORCL\RED001.LOG' SIZE 100M,  
    GROUP 2 'F:\ORACLE\ORADATA\ORCL\RED002.LOG' SIZE 100M,  
    GROUP 3 'F:\ORACLE\ORADATA\ORCL\RED003.LOG' SIZE 100M  
-- STANDBY LOGFILE  
DATAFILE  
    'F:\ORACLE\ORADATA\ORCL\SYSTEM01.DBF',  
    'F:\ORACLE\ORADATA\ORCL\UNDOTBS01.DBF',  
    'F:\ORACLE\ORADATA\ORCL\CWMLITE01.DBF',  
    'G:\ORADATA\DIGITALSCANDATA.DAT',  
    'G:\ORADATA\DIGITALSCANDB.DAT',  
    'F:\ORACLE\ORADATA\ORCL\REPAIRDB.DBF',  
    'F:\ORACLE\ORADATA\ORCL\STANDARDDB.DBF',  
    'G:\ORADATA\DIGITALSCANDB.DAT',  
    'G:\ORADATA\DIGITALSCANDB01.DAT',  
    'F:\ORACLE\ORA92\DATABASE\NETAPPLYTS.DAT',  
    'F:\ORADATA\CSETS.DBF',  
    'H:\ORADATA\DIGITALSCANDB60.DAT'  
CHARACTER SET ZHS16GBK  
;
```

通过这个脚本我们可以看到，用户的数据文件分布在多个磁盘目录上，甚至还有文件创建于

\$ORACLE_HOME/database 目录下，这样混乱的管理，在备份时想不漏掉文件也是困难的，手工编辑控制文件创建脚本也会变得相当复杂。

但是如果我们能够生成这样一个脚本，通过脚本来创建控制文件，则一切问题都可以避免。

所以我们说，对于一个技术人员，扎实的基本功是非常重要的。

通过转储控制文件的 2 进制内容可以看到数据库更为详细的信息：

```
SQL> alter session set events 'immediate trace name controlf level 12';
Session altered.
```

以下是数据库的一些关键信息，通过检查点信息可以看到数据库最后的 On Disk SCN 时间是 2011 年 12 月 30 日，从那之后，数据库就没有被正常打开过：

```
*****
CHECKPOINT PROGRESS RECORDS
*****

(blkno = 0x4, size = 104, max = 1, in-use = 1, last-recid= 0)
THREAD #1 - status:0x1 flags:0x0 dirty:0
low cache rba:(0xffffffff.ffffffff.ffff) on disk rba:(0x5501.580c.0)
on disk scn: 0x0000.2e999200 12/30/2011 16:05:47
resetlogs scn: 0x0000.0002e872 07/07/2009 15:53:52
heartbeat: 773955197 mount id: 1301712020
MTTR statistics status: 3
Init time: Avg: 8603925, Times measured: 3
File open time: Avg: 10283, Times measured: 174
Log block read time: Avg: 2, Times measured: 80611
Data block handling time: Avg: 247, Times measured: 10661
```

这是一次处置不当带来的复杂恢复，属于不应当出现的故障，由于部分文件在备份时以后，后期的恢复时又改写了部分文件，使得完全恢复不再可能，我们通过本书前面描述的方法修改文件头部信息可以完成恢复。

尺有所短，物有不足

硬件故障导致的灾难一则

我们在和系统打交道时，要清楚的知道各种设备的不足之处，扬长避短，这样才能够规避一些可能存在的问题，维持系统的稳定和安全。

2011 年遇到的一个金融客户案例，客户的存储系统，持续运做了近 7 年，最终彻底崩溃，硬盘灯常绿，但是热备盘已经损坏，Raid 5 里又同时损坏 2 块硬盘。

实际上大家都应该具备这样的经验：通常硬件的稳定期在 3 年左右，通常这段时间系统硬件会相当稳定，故障率低，而一旦超过 3 年，就进入了故障多发期，可能会遇到较多的故障。在规划 IT 系统时，应当在 3~5 年左右时考虑更替硬件，规避可能出现的重大故障。

硬件和人一样，过劳都会出问题。如果我们忽视硬件的生命周期，则很有可能在关键时刻遭遇灾难。

灾难描述

这次故障发生的过程很简单：

1. 客户数据库崩溃
2. 检查故障时发现系统读写存储时报警，存储离线
3. 数据库未进行及时有效备份
4. 客户寻找第三方支持寻求故障处理

硬件故障不是人力所能控制的，套用一句俗话：所有的系统要么已经出现故障，要么正在走向故障。我们能够做的工作就是加强维护监控，并且在必要时及时更新硬件，将故障点不断推迟下去。

案例警示

这一则案例给我们的借鉴有：

1. 关键时刻要保护现场寻求支持

这个客户具备非常专业的素质和判断，在数据库出现问题时，首先保护现场，进行认真的分析，当客户觉得事故超出现有资源的控制能力后，接下来就通过各种渠道寻找技术专家来进行判断决策，在此过程中客户未进行任何盲目的恢复尝试。

正是由于客户的冷静判断，认真分析，正确决策，才有了故障的快速恢复与解决，这是任何客户都应该学习借鉴的。

2. 要尊重硬件的使用年限和生命周期

在系统规划时一定不能忽视硬件的生命周期，有时候可能超限使用的硬件一切看起来正常无误，但是一旦听之任之，等着硬件去出现故障，就有可能遭遇严重灾难。

所以，即便在系统完全稳定运行时，也应当做出足够的提前预估，进行硬件设备的维护升级、更新换代。

在这个案例中，硬件是故障根源，但是用户的准确判断最终成为了数据安全的保障。

技术回放

在操作系统的日志中，我们可以找到硬件的出错信息，出现了 IO 读写错误：

```
Nov 19 01:08:03 DBSVRA scsi:
[ID 107833 kern.warning] WARNING: /pci@8,6000000/pci@1/scsi@4/sd@0,0 (sd31):
Nov 19 01:08:03 DBSVRA      SCSI transport failed: reason 'reset': retrying command
Nov 19 01:08:03 DBSVRA scsi:
[ID 107833 kern.warning] WARNING: /pci@8,6000000/pci@1/scsi@4/sd@0,1 (sd63):
Nov 19 01:08:03 DBSVRA      SCSI transport failed: reason 'reset': retrying command
Nov 19 01:08:09 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: read error on /dev/did/dsk/d4s0
Nov 19 01:08:09 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d102: read error on /dev/did/dsk/d5s0
Nov 19 01:08:15 DBSVRA last message repeated 1 time
Nov 19 01:08:15 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: read error on /dev/did/dsk/d4s0
Nov 19 01:08:21 DBSVRA md_stripe:
```

```

[ID 641072 kern.warning] WARNING: md: dbsvr/d101: write error on /dev/did/dsk/d4s0
Nov 19 01:08:21 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d102: read error on /dev/did/dsk/d5s0
Nov 19 01:08:27 DBSVRA last message repeated 1 time
Nov 19 01:08:27 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: read error on /dev/did/dsk/d4s0
Nov 19 01:08:36 DBSVRA last message repeated 1 time
Nov 19 01:08:39 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: write error on /dev/did/dsk/d4s0
Nov 19 01:08:45 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: read error on /dev/did/dsk/d4s0
Nov 19 01:09:00 DBSVRA last message repeated 6 times
Nov 19 01:09:00 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: write error on /dev/did/dsk/d4s0
Nov 19 01:09:03 DBSVRA md_stripe:
[ID 641072 kern.warning] WARNING: md: dbsvr/d101: read error on /dev/did/dsk/d4s0
Nov 19 01:09:03 DBSVRA scsi:
[ID 107833 kern.warning] WARNING: /pci@8,600000/pci@1/scsi@4/sd@0,0 (sd31):

```

在这样的情况下，虽然 IO 读写出现问题，但是通常物理硬盘没有损坏，通过磁盘级别的恢复和数据提取可以恢复出数据文件，再对文件进行简单修复即可挽救数据。

这个用户的操作系统是 Solaris，使用 UFS 文件系统，我们首先通过独立的盘阵挂接故障盘，进行磁盘镜像，保护原有数据盘，然后通过镜像盘进行恢复，非常安全和可靠。

恢复完成的数据在启动时遇到 4193 错误，通过重建 UNDO 表空间得以解决，最后数据库顺利的恢复了运行：

```

Mon Nov 21 18:24:19 2011
Errors in file /oracle/admin/db/bdump/db_smon_13638.trc:
ORA-01595: error freeing extent (3) of rollback segment (25))
ORA-00607: Internal error occurred while making a change to a data block
ORA-00600: internal error code, arguments: [4193], [31866], [5416], [], [], [], [],
[]
SMON: about to recover undo segment 27
SMON: mark undo segment 27 as needs recovery

```

这个案例的顺利恢复得益于用户的正确判断，未进行任何混乱的尝试，值得其他用户借鉴。

附录一 BBED 的说明

在 Linux 和 Unix 上，BBED 缺省未安装，需要通过编译生成可执行文件，以下 make 用于生成 BBED，范例为 Oracle 10g:

```
[oracle@danaly lib]$cd $ORACLE_HOME/rdbms/lib
[oracle@danaly lib]$make -f ins_rdbms.mk $ORACLE_HOME/rdbms/lib/bbed
Linking BBED utility (bbed)
rm -f /opt/oracle/product/10.2.0/rdbms/lib/bbed
gcc -o /opt/oracle/product/10.2.0/rdbms/lib/bbed ...
-L/opt/oracle/product/10.2.0/lib
```

以下为 BBED 的默认帮助:

```
[oracle@danaly lib]$ bbed
Password:
```

```
BBED: Release 2.0.0.0.0 - Limited Production on Sun Sep 3 12:42:59 2006
```

```
Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

```
***** !!! For Oracle Internal Use only !!! *****
```

```
BBED> help ALL
```

```
SET DBA [ dba | file#, block# ]
```

```
SET FILENAME 'filename'
```

```
SET FILE file#
```

```
SET BLOCK [+/-]block#
```

```
SET OFFSET [ [+/-]byte offset | symbol | *symbol ]
```

```
SET BLOCKSIZE bytes
```

```
SET LIST[FILE] 'filename'
```

```
SET WIDTH character_count
```

```
SET COUNT bytes_to_display
```

```

SET IBASE [ HEX | OCT | DEC ]
SET OBASE [ HEX | OCT | DEC ]
SET MODE [ BROWSE | EDIT ]
SET SPOOL [ Y | N ]
SHOW [ | ALL ]
INFO
MAP[/v] [ DBA | FILENAME | FILE | BLOCK ]
DUMP[/v] [ DBA | FILENAME | FILE | BLOCK | OFFSET | COUNT ]
PRINT[/x|d|u|o|c] [ DBA | FILE | FILENAME | BLOCK | OFFSET | symbol | *symbol ]
EXAMINE[/Nuf] [ DBA | FILE | FILENAME | BLOCK | OFFSET | symbol | *symbol ]
:
N - a number which specifies a repeat count.
u - a letter which specifies a unit size:
    b - b1, ub1 (byte)
    h - b2, ub2 (half-word)
    w - b4, ub4(word)
    r - Oracle table/index row
f - a letter which specifies a display format:
    x - hexadecimal
    d - decimal
    u - unsigned decimal
    o - octal
    c - character (native)
    n - Oracle number
    t - Oracle date
    i - Oracle rowid
FIND[/x|d|u|o|c] numeric/character string [ TOP | CURR ]
COPY [ DBA | FILE | FILENAME | BLOCK ] TO [ DBA | FILE | FILENAME | BLOCK ]
MODIFY[/x|d|u|o|c] numeric/character string
    [ DBA | FILE | FILENAME | BLOCK | OFFSET | symbol | *symbol ]
ASSIGN[/x|d|u|o] =
: [ DBA | FILE | FILENAME | BLOCK | OFFSET | symbol | *symbol ]
: [ value | ]

```

```

SUM [ DBA | FILE | FILENAME | BLOCK ] [ APPLY ]
PUSH [ DBA | FILE | FILENAME | BLOCK | OFFSET ]
POP [ALL]
REVERT [ DBA | FILE | FILENAME | BLOCK ]
UNDO
HELP [ | ALL ]
VERIFY [ DBA | FILE | FILENAME | BLOCK ]
CORRUPT [ DBA | FILE | FILENAME | BLOCK ]
BBED> exit

```

Oracle Database 11g 中缺省的未提供 BBED 库文件，但是可以用 10g 的文件编译出来，参考如下步骤：

1.复制 Oracle 10g 文件

```
Copy $ORA10g_HOME/rdbms/lib/ssbbded.o to $ORA11g_HOME/rdbms/lib
```

```
Copy $ORA10g_HOME/rdbms/lib/sbbdpt.o to $ORA11g_HOME/rdbms/lib
```

```
Copy $ORA10g_HOME/rdbms/msg/bbedus.msb to $ORA11g_HOME/rdbms/msg
```

```
Copy $ORA10g_HOME/rdbms/msg/bbedus.msg to $ORA11g_HOME/rdbms/msg
```

```
Copy $ORA10g_HOME/rdbms/msg/bbedar.msb to $ORA11g_HOME/rdbms/msg
```

2.编译

```
make -f $ORA11g_HOME/rdbms/lib/ins_rdbms.mk BBED=$ORACLE_HOME/bin/bbed $ORACLE_HOME/bin/bbed
```

对于 Windows 平台，BBED 在 9i 的某些版本缺省（如 9.2.0.6）提供，可以提取供以后版本使用。从 Oracle 10g 开始，Windows 平台不再提供 BBED 工具。

附录二 函数 f_get_from_dump

```
CREATE OR REPLACE FUNCTION F_GET_FROM_DUMP
(
  P_DUMP IN VARCHAR2,
  P_TYPE IN VARCHAR2
)
RETURN VARCHAR2 AS
  V_LENGTH_STR VARCHAR2(10);
  V_LENGTH NUMBER DEFAULT 7;
  V_DUMP_ROWID VARCHAR2(30000);

  V_DATE_STR VARCHAR2(100);
  TYPE T_DATE IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
  V_DATE T_DATE;

  FUNCTION F_ADD_PREFIX_ZERO (P_STR IN VARCHAR2, P_POSITION IN NUMBER) RETURN VARCHAR2
  AS
    V_STR VARCHAR2(30000) := P_STR;
    V_POSITION NUMBER := P_POSITION;
    V_STR_PART VARCHAR2(2);
    V_RETURN VARCHAR2(30000);
  BEGIN
    WHILE (V_POSITION != 0) LOOP
      V_STR_PART := SUBSTR(V_STR, 1, V_POSITION - 1);
      V_STR := SUBSTR(V_STR, V_POSITION + 1);

      IF V_POSITION = 2 THEN
        V_RETURN := V_RETURN || '0' || V_STR_PART;
      ELSIF V_POSITION = 3 THEN
```

```

        V_RETURN := V_RETURN || V_STR_PART;
    ELSE
        RAISE_APPLICATION_ERROR(-20002, 'DUMP ERROR CHECK THE INPUT ROWID');
    END IF;

    V_POSITION := INSTR(V_STR, ',');
    END LOOP;
    RETURN REPLACE(V_RETURN, ',');
END F_ADD_PREFIX_ZERO;

BEGIN
    IF SUBSTR(P_DUMP, 1, 3) = 'Typ' THEN
        V_DUMP_ROWID := SUBSTR(P_DUMP, INSTR(P_DUMP, ':') + 2);
    ELSE
        V_DUMP_ROWID := P_DUMP;
    END IF;

    IF P_TYPE = 'VARCHAR2' OR P_TYPE = 'CHAR' THEN

        V_DUMP_ROWID := F_ADD_PREFIX_ZERO(V_DUMP_ROWID || ',', INSTR(V_DUMP_ROWID, ','));

        RETURN(UTL_RAW.CAST_TO_VARCHAR2(V_DUMP_ROWID));

    ELSIF P_TYPE = 'NVARCHAR2' OR P_TYPE = 'NCHAR' THEN

        V_DUMP_ROWID := F_ADD_PREFIX_ZERO(V_DUMP_ROWID || ',', INSTR(V_DUMP_ROWID, ','));

        RETURN(UTL_RAW.CAST_TO_NVARCHAR2(V_DUMP_ROWID));

    ELSIF P_TYPE = 'NUMBER' THEN

        V_DUMP_ROWID := F_ADD_PREFIX_ZERO(V_DUMP_ROWID || ',', INSTR(V_DUMP_ROWID, ','));

```

```

RETURN(UTL_RAW.CAST_TO_NUMBER(V_DUMP_ROWID));

ELSIF P_TYPE = 'DATE' THEN

V_DUMP_ROWID := ',' || V_DUMP_ROWID || ',';

FOR I IN 1..7 LOOP
    V_DATE(I) := TO_NUMBER(SUBSTR(V_DUMP_ROWID, INSTR(V_DUMP_ROWID, ',', 1, I) + 1,
        INSTR(V_DUMP_ROWID, ',', 1, I + 1) - INSTR(V_DUMP_ROWID, ',', 1, I) - 1), 'XXX');
END LOOP;

V_DATE(1) := V_DATE(1) - 100;
V_DATE(2) := V_DATE(2) - 100;

IF ((V_DATE(1) < 0) OR (V_DATE(2) < 0)) THEN
    V_DATE_STR := '-' || LTRIM(TO_CHAR(ABS(V_DATE(1)), '00')) || LTRIM(TO_CHAR(ABS(V_DATE(2)),
'00'));
ELSE
    V_DATE_STR := LTRIM(TO_CHAR(ABS(V_DATE(1)), '00')) || LTRIM(TO_CHAR(ABS(V_DATE(2)), '00'));
END IF;

V_DATE_STR := V_DATE_STR || '-' || TO_CHAR(V_DATE(3)) || '-' || TO_CHAR(V_DATE(4)) || '-' ||
    TO_CHAR(V_DATE(5) - 1) || ':' || TO_CHAR(V_DATE(6) - 1) || ':' || TO_CHAR(V_DATE(7) - 1);
RETURN (V_DATE_STR);

ELSIF ((P_TYPE LIKE 'TIMESTAMP(_)' ) OR (P_TYPE = 'TIMESTAMP')) THEN

V_DUMP_ROWID := ',' || V_DUMP_ROWID || ',';

FOR I IN 1..11 LOOP
    V_DATE(I) := TO_NUMBER(SUBSTR(V_DUMP_ROWID, INSTR(V_DUMP_ROWID, ',', 1, I) + 1,
        INSTR(V_DUMP_ROWID, ',', 1, I + 1) - INSTR(V_DUMP_ROWID, ',', 1, I) - 1), 'XXX');
END LOOP;

```

```

V_DATE(1) := V_DATE(1) - 100;
V_DATE(2) := V_DATE(2) - 100;

IF ((V_DATE(1) < 0) OR (V_DATE(2) < 0)) THEN
    V_DATE_STR := '-' || LTRIM(TO_CHAR(ABS(V_DATE(1)), '00')) || LTRIM(TO_CHAR(ABS(V_DATE(2)),
'00'));
ELSE
    V_DATE_STR := LTRIM(TO_CHAR(ABS(V_DATE(1)), '00')) || LTRIM(TO_CHAR(ABS(V_DATE(2)), '00'));
END IF;

V_DATE_STR := V_DATE_STR || '-' || TO_CHAR(V_DATE(3)) || '-' || TO_CHAR(V_DATE(4)) || '-' ||
TO_CHAR(V_DATE(5) - 1) || '-' || TO_CHAR(V_DATE(6) - 1) || '-' || TO_CHAR(V_DATE(7) - 1) || '-' ||
SUBSTR(TO_CHAR(V_DATE(8) * POWER(256, 3) + V_DATE(9) * POWER(256, 2) + V_DATE(10) * 256 +
V_DATE(11)),
1, NVL(TO_NUMBER(SUBSTR(P_TYPE, 11, 1)), 6));
RETURN (V_DATE_STR);

ELSIF P_TYPE = 'RAW' THEN

V_DUMP_ROWID := F_ADD_PREFIX_ZERO(V_DUMP_ROWID || ',', INSTR(V_DUMP_ROWID, ','));

RETURN(V_DUMP_ROWID);

ELSIF P_TYPE = 'ROWID' THEN

V_DUMP_ROWID := F_ADD_PREFIX_ZERO(V_DUMP_ROWID || ',', INSTR(V_DUMP_ROWID, ','));

RETURN (DBMS_ROWID.ROWID_CREATE(
1,
TO_NUMBER(SUBSTR(V_DUMP_ROWID, 1, 8), 'XXXXXXXXXX'),
TRUNC(TO_NUMBER(SUBSTR(V_DUMP_ROWID, 9, 4), 'XXXXXX')/64),
TO_NUMBER(MOD(TO_NUMBER(SUBSTR(V_DUMP_ROWID, 9, 4), 'XXXXXX'), 64) ||

```

```
        TO_NUMBER(SUBSTR(V_DUMP_ROWID, 13, 4), 'XXXXXXXXXX'),
        TO_NUMBER(SUBSTR(V_DUMP_ROWID, 17, 4), 'XXXXXX')));

ELSE
    RAISE_APPLICATION_ERROR(-20001, 'TYPE NOT VALID OR CAN"T TRANSALTE ' || P_TYPE || ' TYPE');
END IF;

END;

/
```

参考资料

[1]Bumblebee 项目, <https://github.com/MrMEEE/bumblebee/commit/a047be85247755cdbe0acce6#diff-1>.

[2]一个空格引发的惨剧, <http://coolshell.cn/articles/4875.html>.

[3]ITPUB 论坛技术讨论: 职业生涯误操作篇, <http://www.itpub.net/thread-911086-1-1.html>.

[4]猜测的力量, 刘磊 ACOUG 主题演讲.

博文视点诚邀精锐作者加盟

九载耕耘奠定专业地位

以书为证彰显卓越品质

《代码大全》、《Windows内核情景分析》、《加密与解密》、《编程之美》、《VC++深入详解》、《SEO实战密码》、《PPT演义》……

“圣经”级图书光耀夺目,被无数读者朋友奉为案头手册传世经典。

潘爱民、毛德操、张亚勤、张宏江、咎辉Zac、李刚、曹江华……

“明星”级作者济济一堂,他们的名字熠熠生辉,与IT业的蓬勃发展紧密相连。

九年的开拓、探索和励精图治,成就博古通今、文圆质方、视角独特、点石成金之计算机图书的风向标杆:博文视点。

“凤翱翔于千仞兮,非梧不栖”,博文视点欢迎更多才华横溢、锐意创新的作者朋友加盟,与大师并列于IT专业出版之巅。

英雄帖

江湖风云起,代有才人出。

IT界群雄并起,逐鹿中原。

博文视点诚邀天下技术英豪加入,

指点江山,激扬文字

传播信息技术,分享IT心得

• 专业的作者服务 •

博文视点自成立以来一直专注于IT专业技术图书的出版,拥有丰富的与技术图书作者合作的经验,并参照IT技术图书的特点,打造了一支高效运转、富有服务意识的编辑出版团队。我们始终坚持:

善待作者——我们会把出版流程整理得清晰简明,为作者提供优厚的稿酬服务,解除作者的顾虑,安心写作,展现出最好的作品。

尊重作者——我们尊重每一位作者的技术能力和生活习惯,并会参照作者实际的工作、生活节奏,量身制定写作计划,确保合作顺利进行。

提升作者——我们打造精品图书,更要打造知名作者。博文视点致力于通过图书提升作者的个人品牌和技术影响力,为作者的事业开拓带来更多的机会。



联系我们

博文视点官网: <http://www.broadview.com.cn>

新浪官方微博: <http://weibo.com/broadviewbj>

投稿电话: 010-51260888 88254368

CSDN官方博客: <http://blog.csdn.net/broadview2006/>

腾讯官方微博: <http://t.qq.com/bowenshidian>

投稿邮箱: jsj@phei.com.cn