	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，结果导到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 是否是用户进程
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，避免因为不规范带来的故障。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订时原本是要在 SQLPLUS 登陆后 SHUTOWN IMMEDIATE 当那时敲得太快没注意 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 1 0 几個小時夜班後正準備下班，點進 VM 執行 init 0，卻忘記有從這個 VM 窗口 telnet 到生產環境 cp 參數文件而且等數據庫狀態監控報警後才反應過來... 還好是 RAC，但也造成不小影響，從此下任何指

	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于避免空格錯誤
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，避免因为误操作而“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，辛好没重要数据，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 -inittab*.crsd 等几个

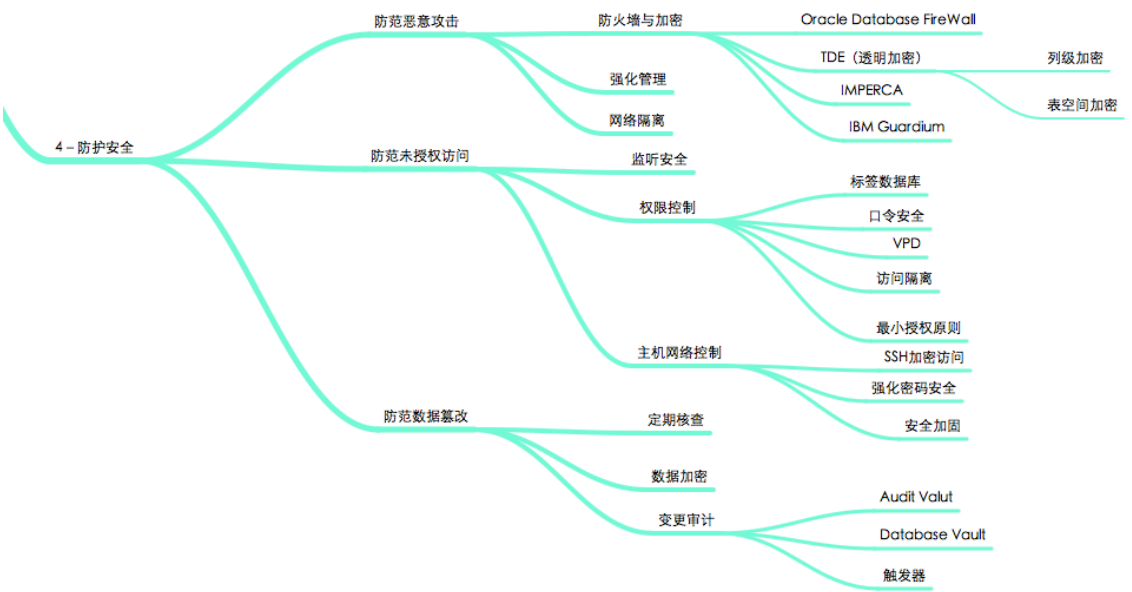
	rac 的启动文件，一不留神吧 AIX 的一个文件删了，结果系统起不来了。后来多亏 ibm 的工程师恢复了系统。结果晚上 3 点才收工

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

通过分析与解决，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: 0x8000000 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  itl: 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
```