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，避免因为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$中的内容给删除了，造成 tablespace 里的数据文件列表看不到了
误删除数据表	一次误删了个表,最后恢复了,丢了一天数据.加了一晚上班,至今记得.**人越累的时候就越容易犯错误**,我就是在最后快下班的几分钟犯的错误
误截断数据表	我最惨，有一次把一个表一不小心给 truncate 了，上千万条记录一眨眼就没了；提心吊胆的陪了 3 天也没有把这个表搞定；最后不了了之了
误删除字典视图	我 drop 掉过 view sys.v$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 語句，結果看都沒看就直接在生產環境中做了，一下子物料表就沒了，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和高质量工作的保障，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 ctrl_alt_delete，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	我一同事自己做了個測試環境，想把生產庫的數據導入到他自己的 DB 里，結果不小心，干掉了生產庫里好多表，直接導致產線停了 4 個小時
生产与测试环境错误	有次删除測試庫的 USER，結果把正式庫的給删除了
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	有一次在測試庫 drop 掉一個表，drop 完發現把生產庫中的表給 DROP 了,1000 多万筆紀錄啊.當時產線就停了，最後一級生產事故．偶公開檢討 教訓：**不能用 TOAD 同時打開兩個以上的庫**
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)，一加上去，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 个过月了，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 剛准備上線的那一次,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 不光是只要技朮的,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，辛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