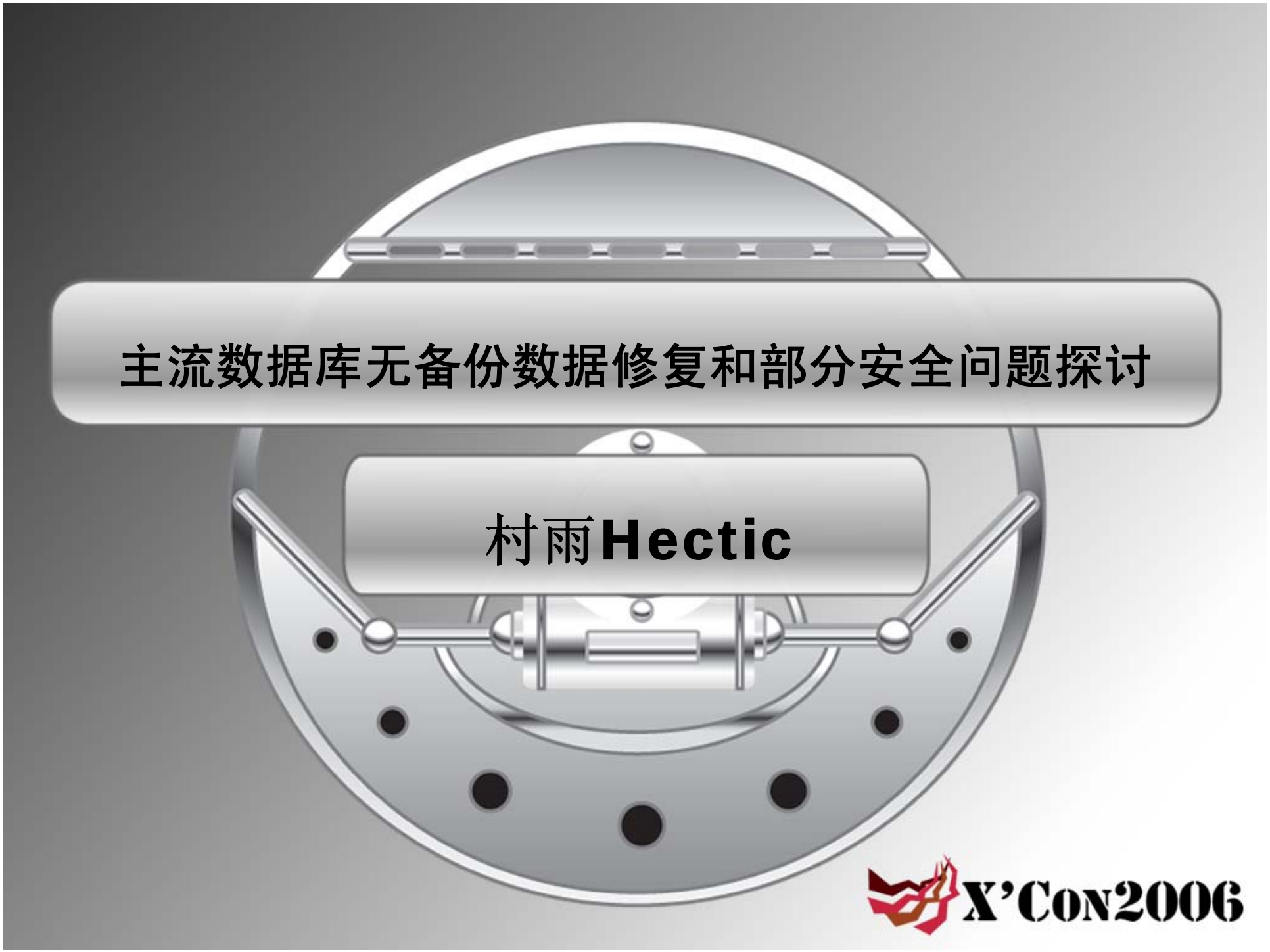
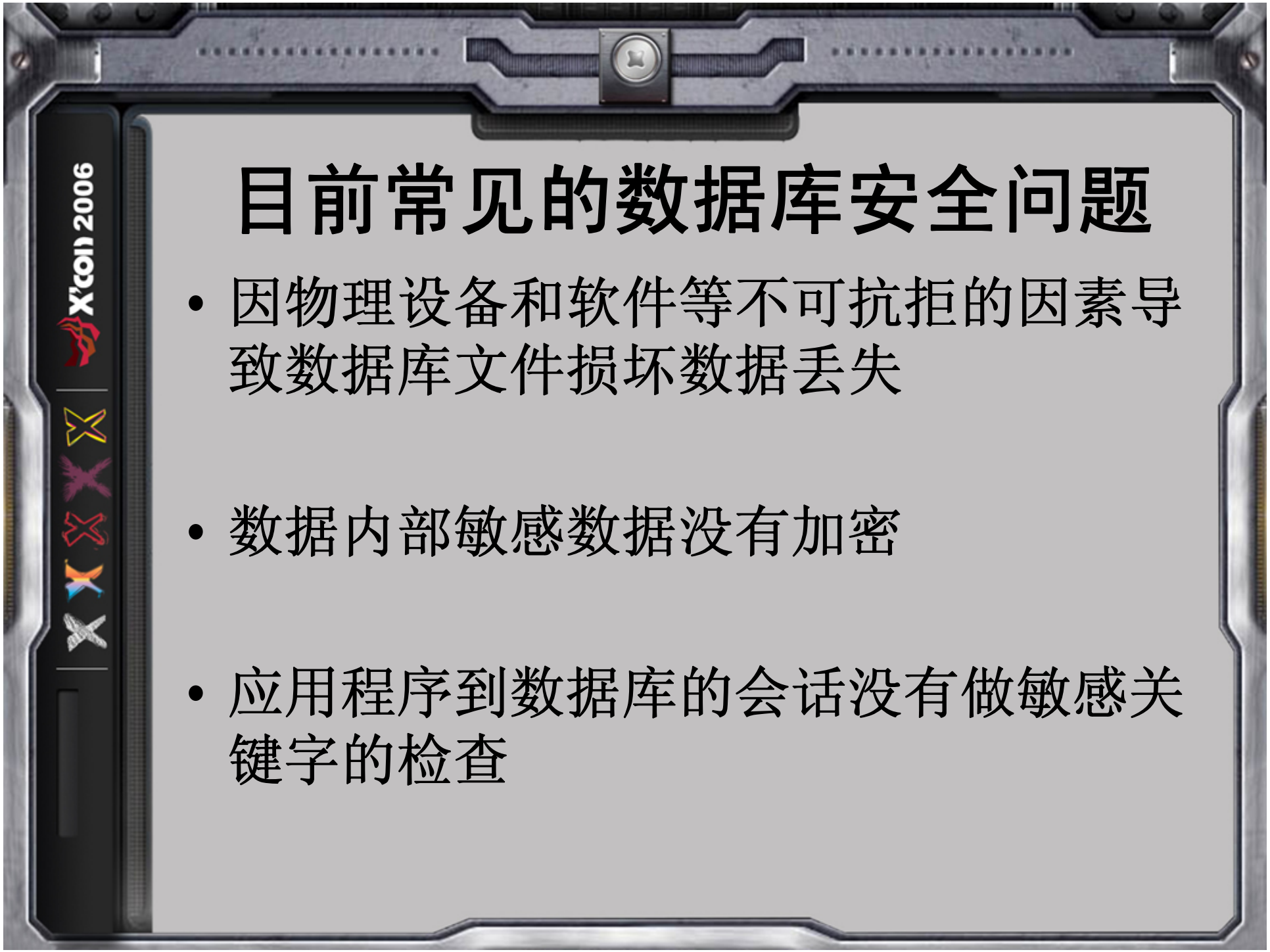

主流数据库无备份数据修复和部分安全问题探讨
村雨Hectic
X'Con2006

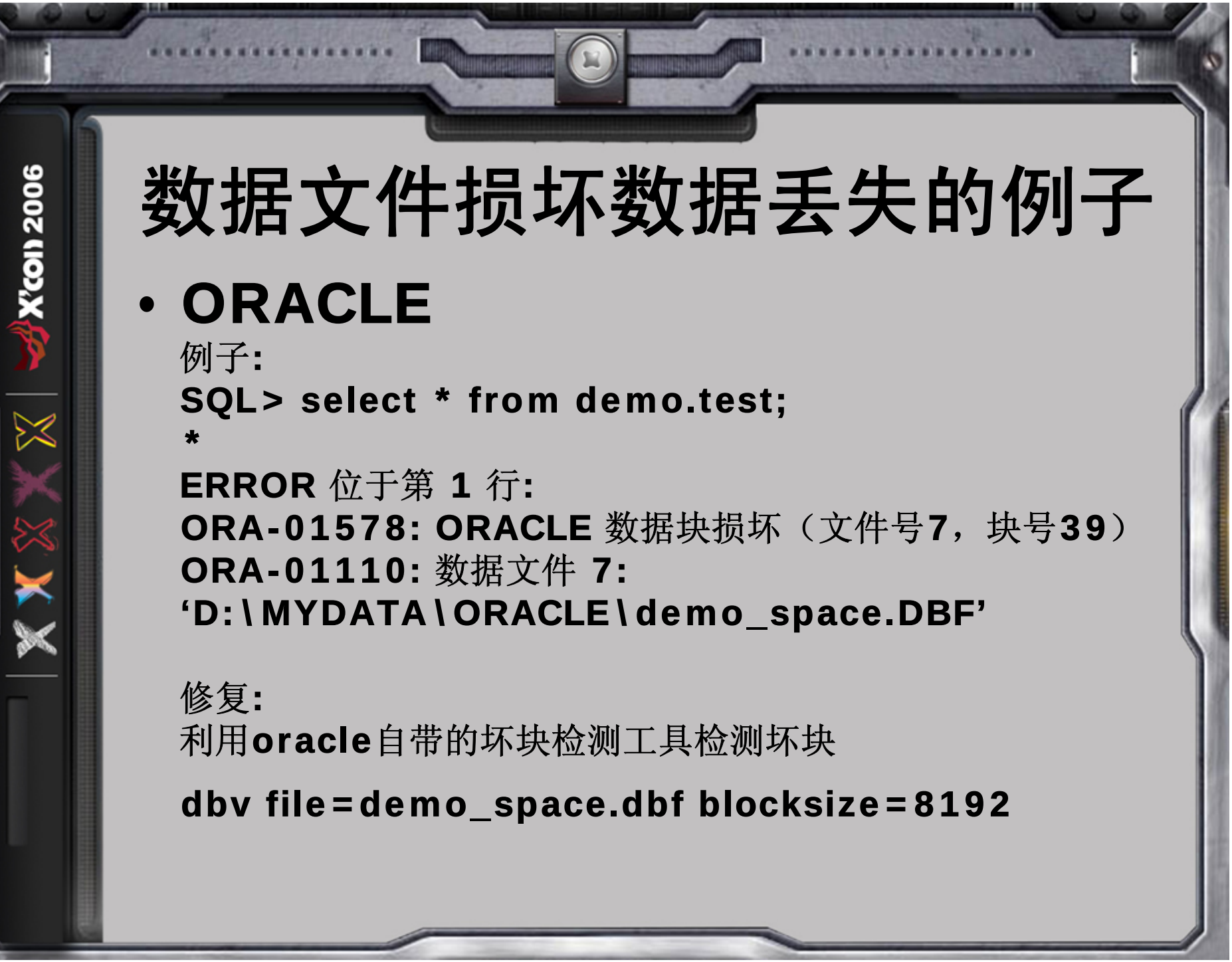
目前常见的应用数据库

- ORACLE
- Microsoft SQL SERVER
- MYSQL



目前常见的数据库安全问题

- 因物理设备和软件等不可抗拒的因素导致数据库文件损坏数据丢失
- 数据内部敏感数据没有加密
- 应用程序到数据库的会话没有做敏感关键字的检查



数据文件损坏数据丢失的例子

- **ORACLE**

  例子:

  SQL> select * from demo.test;

  *****

  ERROR 位于第 **1** 行:

  ORA-01578: ORACLE 数据块损坏（文件号**7**，块号**39**）

  ORA-01110: 数据文件 **7:**

  ‘D:\MYDATA\ORACLE\demo_space.DBF’

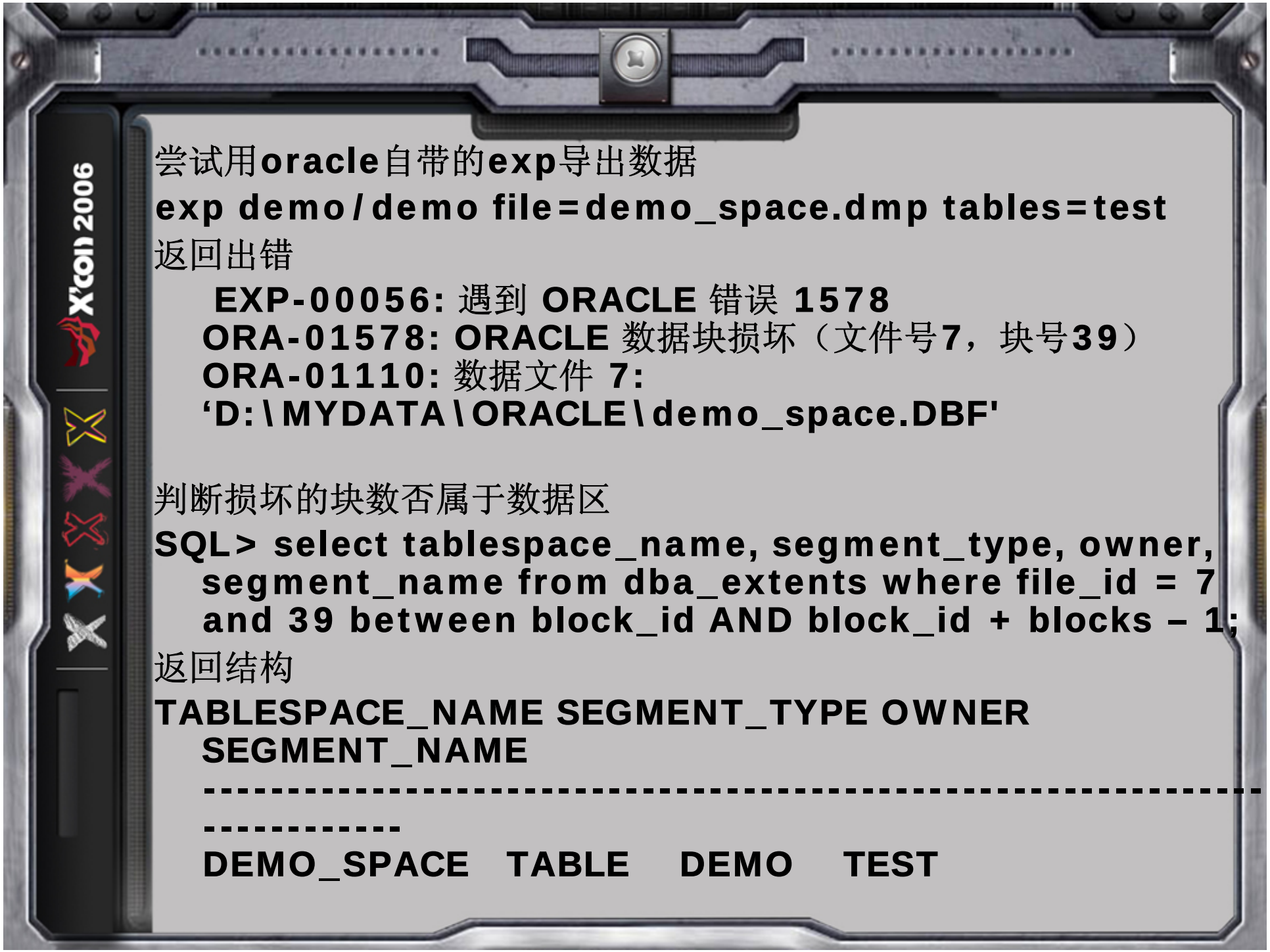
  修复:

  利用**oracle**自带的坏块检测工具检测坏块

  dbv file=demo_space.dbf blocksize=8192

DBV的返回结果

DBVERIFY - 验证正在开始：FILE = demo_space.dbf
标记为损坏的页39
***
Corrupt block relative dba: 0x01000056 (file 7, block 39)
Bad check value found during dbv:
Data in bad block -
type: 6 format: 2 rdba: 0x01000056
last change scn: 0x0000.00044057 seq: 0x1 flg: 0x06
consistency value in tail: 0x90430608
check value in block header: 0xb6ef, computed block checksum: 0x5e0f
spare1: 0x0, spare2: 0x0, spare3: 0x0
***



尝试用oracle自带的exp导出数据

```
exp demo/demo file=demo_space.dmp tables=test
```

返回出错

```
 EXP-00056: 遇到 ORACLE 错误 1578
ORA-01578: ORACLE 数据块损坏（文件号7，块号39）
ORA-01110: 数据文件 7:
'D:\MYDATA\ORACLE\demo_space.DBF'
```

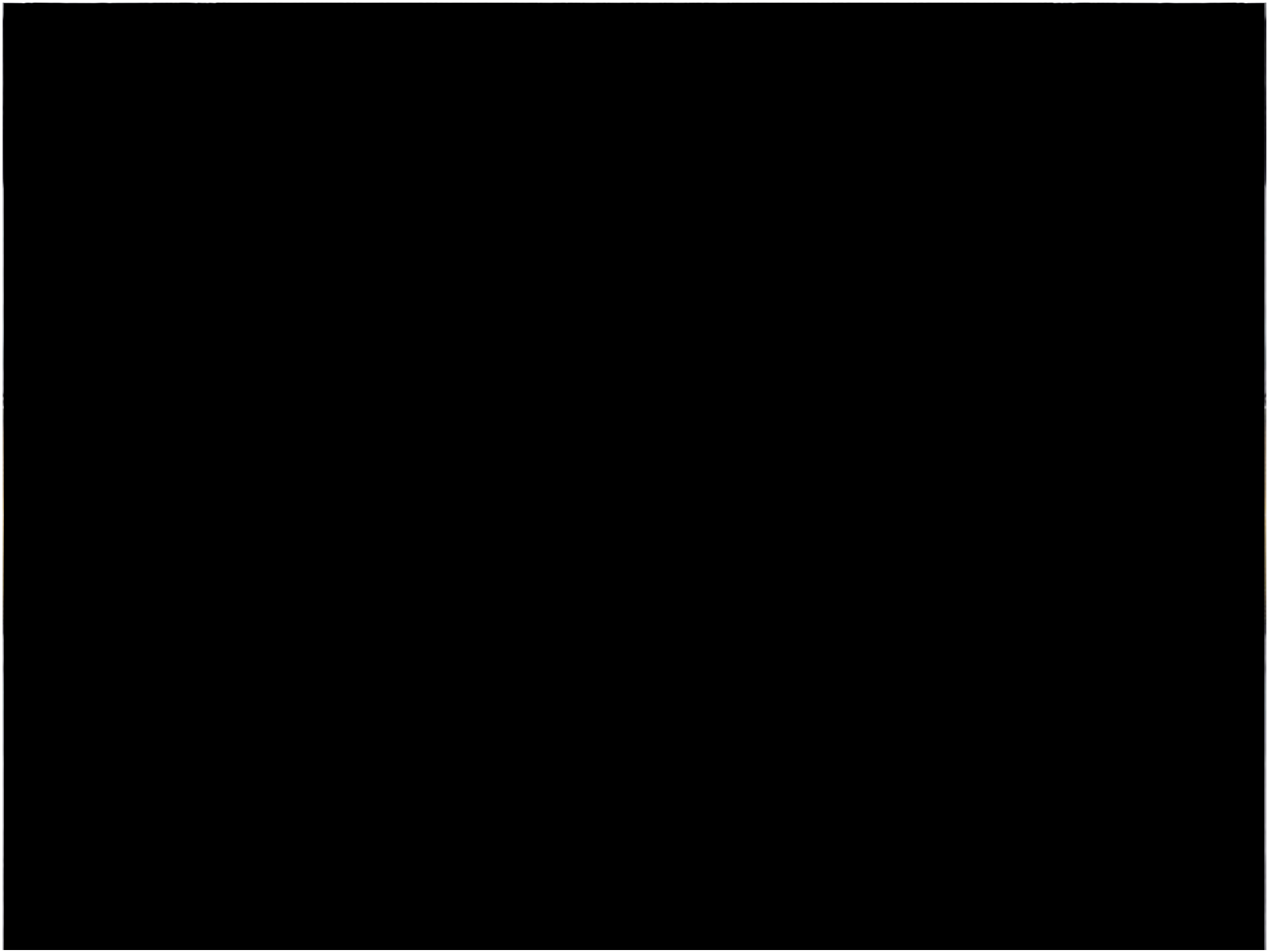
判断损坏的块数否属于数据区

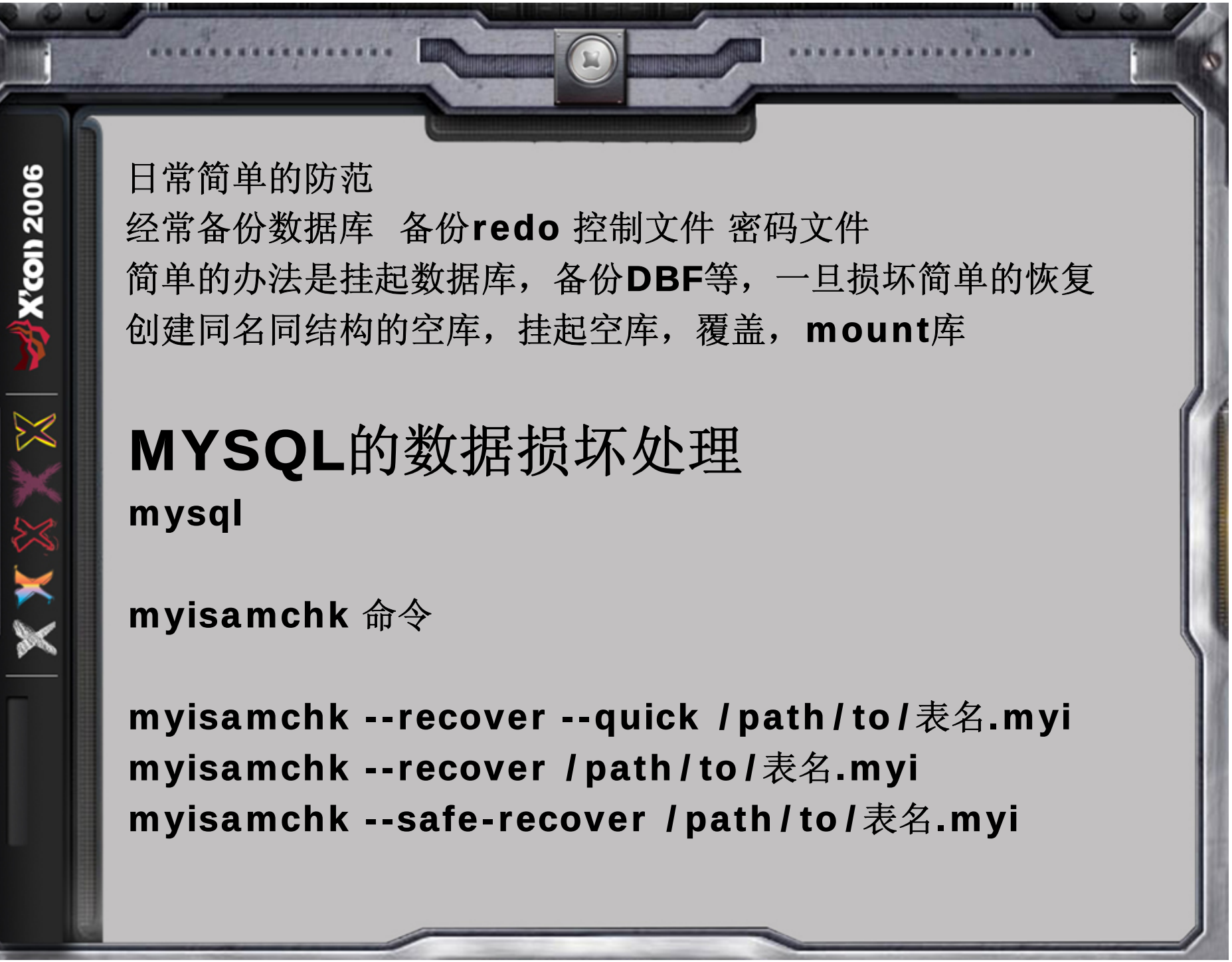
```
SQL> select tablespace_name, segment_type, owner,
   segment_name from dba_extents where file_id = 7
   and 39 between block_id AND block_id + blocks – 1;
```

返回结构

```
TABLESPACE_NAME SEGMENT_TYPE OWNER
   SEGMENT_NAME
   ------------------------------------------------------------
   -----------
   DEMO_SPACE   TABLE    DEMO    TEST
```


如果损坏的仅仅只是数据区那么自定义内部异常事件来跳过
导出时候遇到的坏块
alter system set events='10231 trace name context
forever,level 10' ;
这个自定义内部事件可以让oracle在全表扫描的时候跳过坏块
然后再次执行exp导出没有损坏的数据
exp demo/demo file=demo_space.dmp tables=test
返回
. . 正在导出表 TEST 7942 行被导出
在没有警告的情况下成功终止导出。
原先表的行数 – 7942 = 我们丢失的数据行数
然后利用drop table和recreate重新创建表空间
最后用imp导回数据
imp demo/demo file=test.dmp tables=test





日常简单的防范

经常备份数据库 备份**redo** 控制文件 密码文件

简单的办法是挂起数据库，备份**DBF**等，一旦损坏简单的恢复

创建同名同结构的空库，挂起空库，覆盖，**mount**库

MYSQL的数据损坏处理

mysql

myisamchk 命令

myisamchk --recover --quick /path/to/表名**.myi**

myisamchk --recover /path/to/表名**.myi**

myisamchk --safe-recover /path/to/表名**.myi**

日常简单的防范

经常备份数据库 备份**redo** 控制文件 密码文件

简单的办法是挂起数据库，备份**DBF**等，一旦损坏简单的恢复

创建同名同结构的空库，挂起空库，覆盖，**mount**库

MYSQL的数据损坏处理

Mysql对应表的三个文件

Tablename.frm 表结构

Tablename.myd 数据文件

Tablename.myi 索引文件

myisamchk 命令

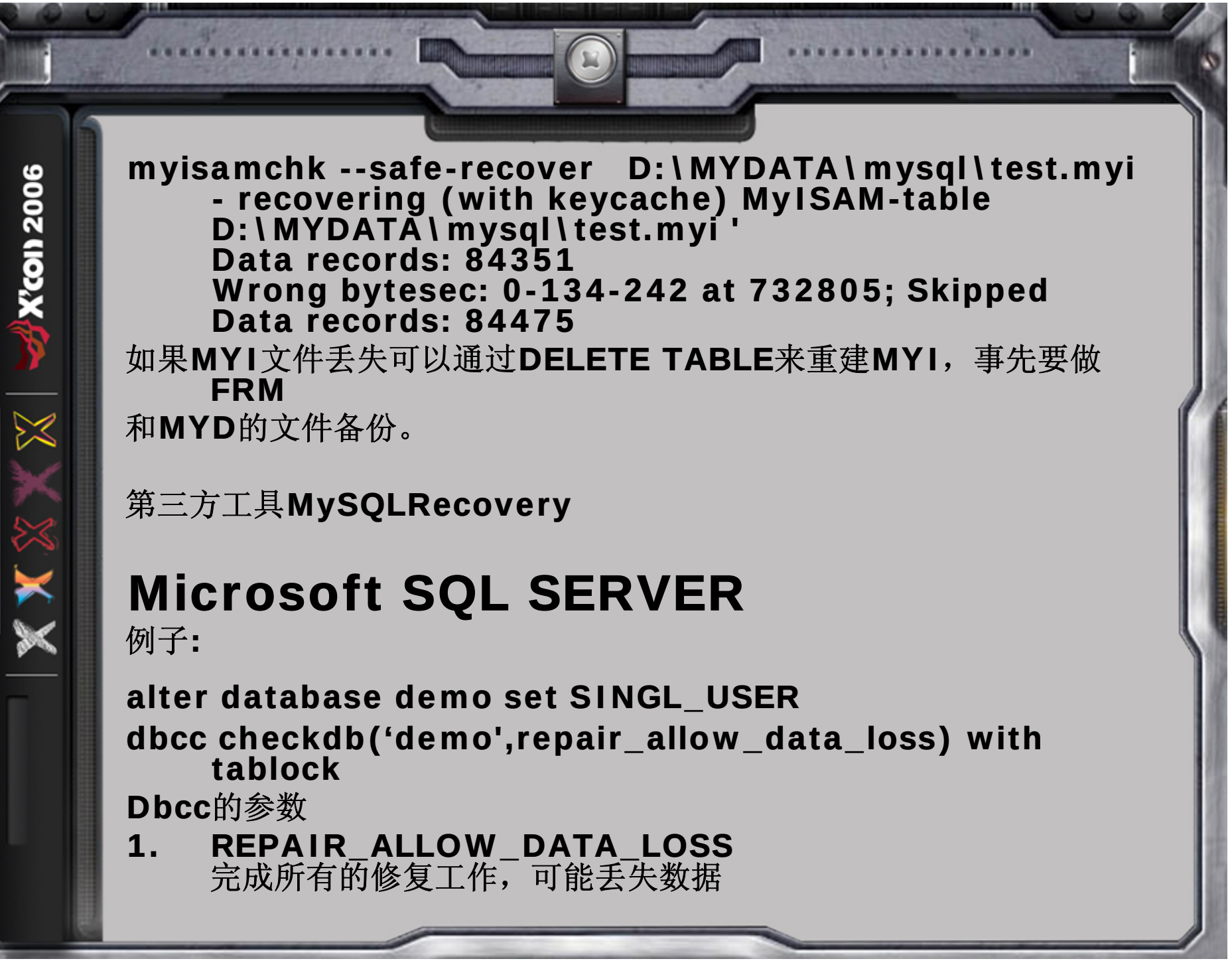
检查表的错误

Myisamchk 表名**.myi**

myisamchk --recover --quick /path/to/表名**.myi**

myisamchk --recover /path/to/表名**.myi**

myisamchk --safe-recover /path/to/表名**.myi**



myisamchk --safe-recover D:\MYDATA\mysql\test.myi
- recovering (with keycache) MyISAM-table
D:\MYDATA\mysql\test.myi '
Data records: 84351
Wrong bytesec: 0-134-242 at 732805; Skipped
Data records: 84475

如果**MYI**文件丢失可以通过**DELETE TABLE**来重建**MYI**，事先要做**FRM**
和**MYD**的文件备份。

第三方工具**MySQLRecovery**

Microsoft SQL SERVER

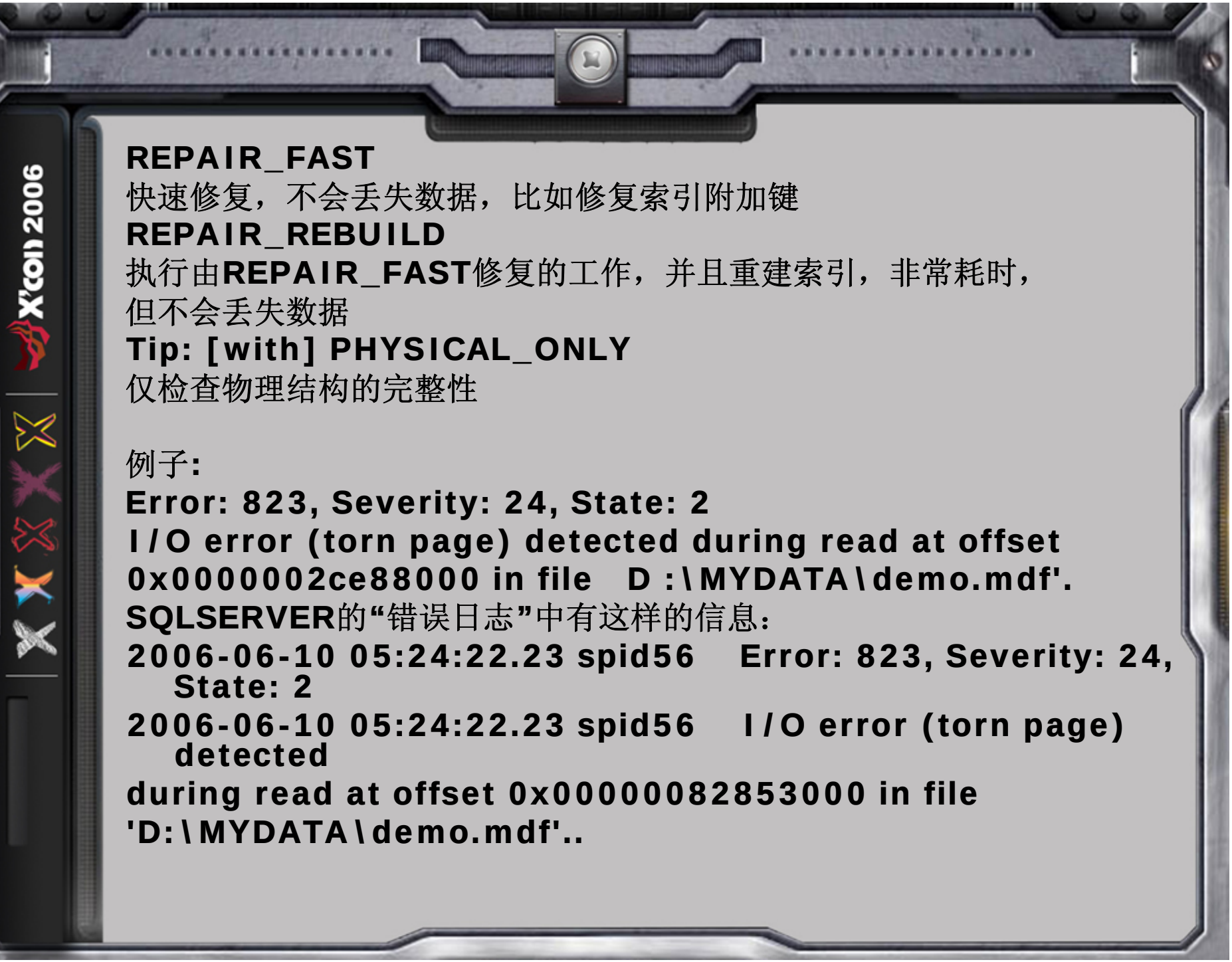
例子:

alter database demo set SINGL_USER

dbcc checkdb('demo',repair_allow_data_loss) with tablock

Dbcc的参数

1. **REPAIR_ALLOW_DATA_LOSS**
   完成所有的修复工作，可能丢失数据



REPAIR_FAST
快速修复，不会丢失数据，比如修复索引附加键
REPAIR_REBUILD
执行由**REPAIR_FAST**修复的工作，并且重建索引，非常耗时，
但不会丢失数据
Tip: [with] PHYSICAL_ONLY
仅检查物理结构的完整性

例子:
Error: 823, Severity: 24, State: 2
I/O error (torn page) detected during read at offset 0x0000002ce88000 in file D :\MYDATA\demo.mdf'.
SQLSERVER的“错误日志”中有这样的信息：
2006-06-10 05:24:22.23 spid56 Error: 823, Severity: 24, State: 2
2006-06-10 05:24:22.23 spid56 I/O error (torn page) detected during read at offset 0x00000082853000 in file 'D:\MYDATA\demo.mdf'..

Dbcc checkdb(‘demo') with NO_INFOMSGS,PHYSICAL_ONLY

- **NO_INFOMSGS屏蔽所有的信息何报告数据**

Dbcc checkdb(‘demo',repair_allow_data_loss) with tablock

其他情况:

当数据库文件崩溃的时候

如果数据文件和日志文件都完整

sp_attach_db

如果缺少日志文件

sp_attach_single_file_db

如果无法**attach**

sp_configure 'allow updates', 1

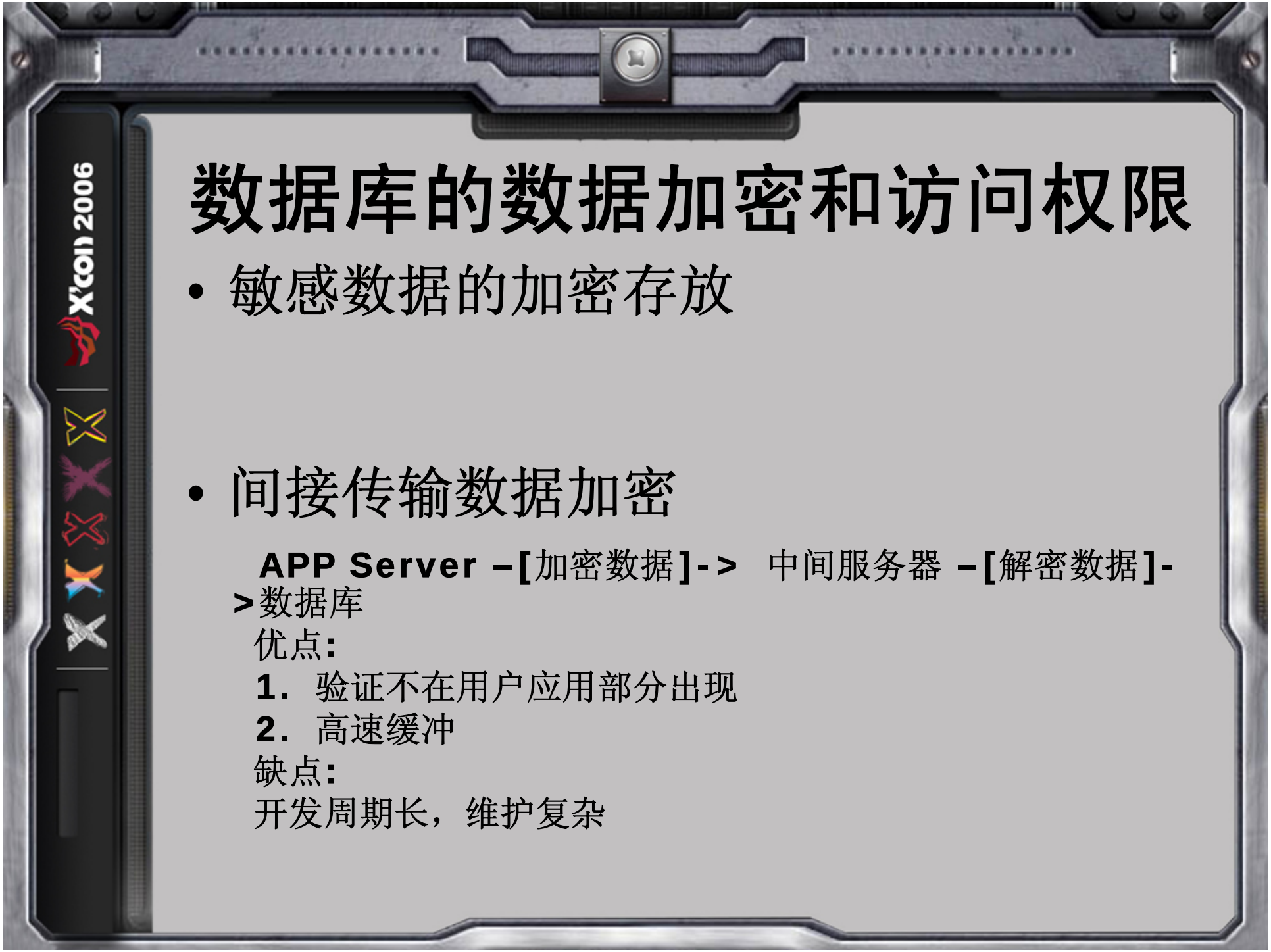
reconfigure with override

允许用户修改系统表

先把数据库设置为紧急事务状态

即**sysdatabases**的**status**为**-32768**

最后重建日志 **dbcc rebuild_log(‘demo’, ‘d:\demo.ldf’)**



数据库的数据加密和访问权限

- 敏感数据的加密存放

- 间接传输数据加密

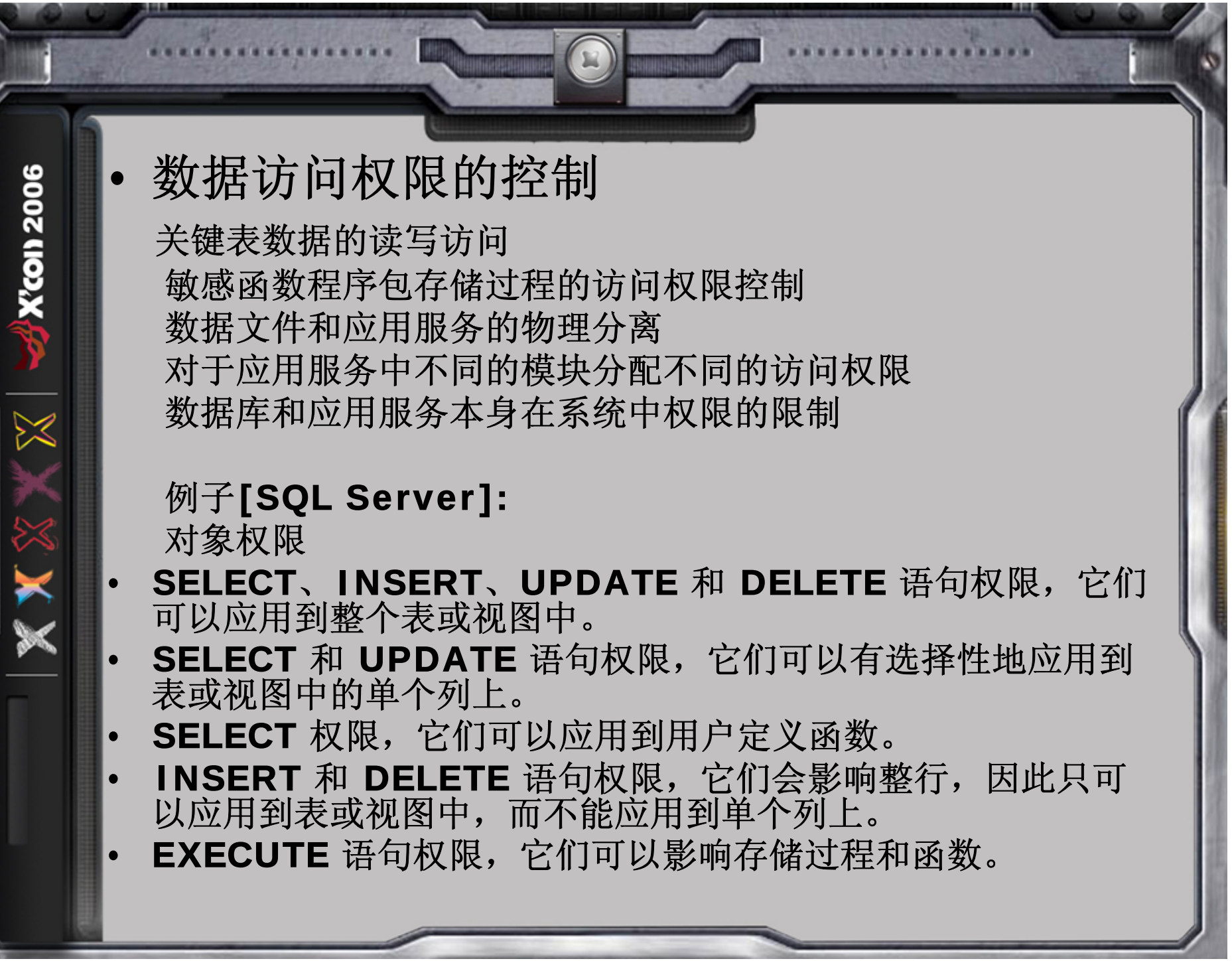
  APP Server –[加密数据**]->** 中间服务器 **–[**解密数据**]->**数据库

  优点:

  1. 验证不在用户应用部分出现
  2. 高速缓冲

  缺点:

  开发周期长，维护复杂



- 数据访问权限的控制

  关键表数据的读写访问

  敏感函数程序包存储过程的访问权限控制

  数据文件和应用服务的物理分离

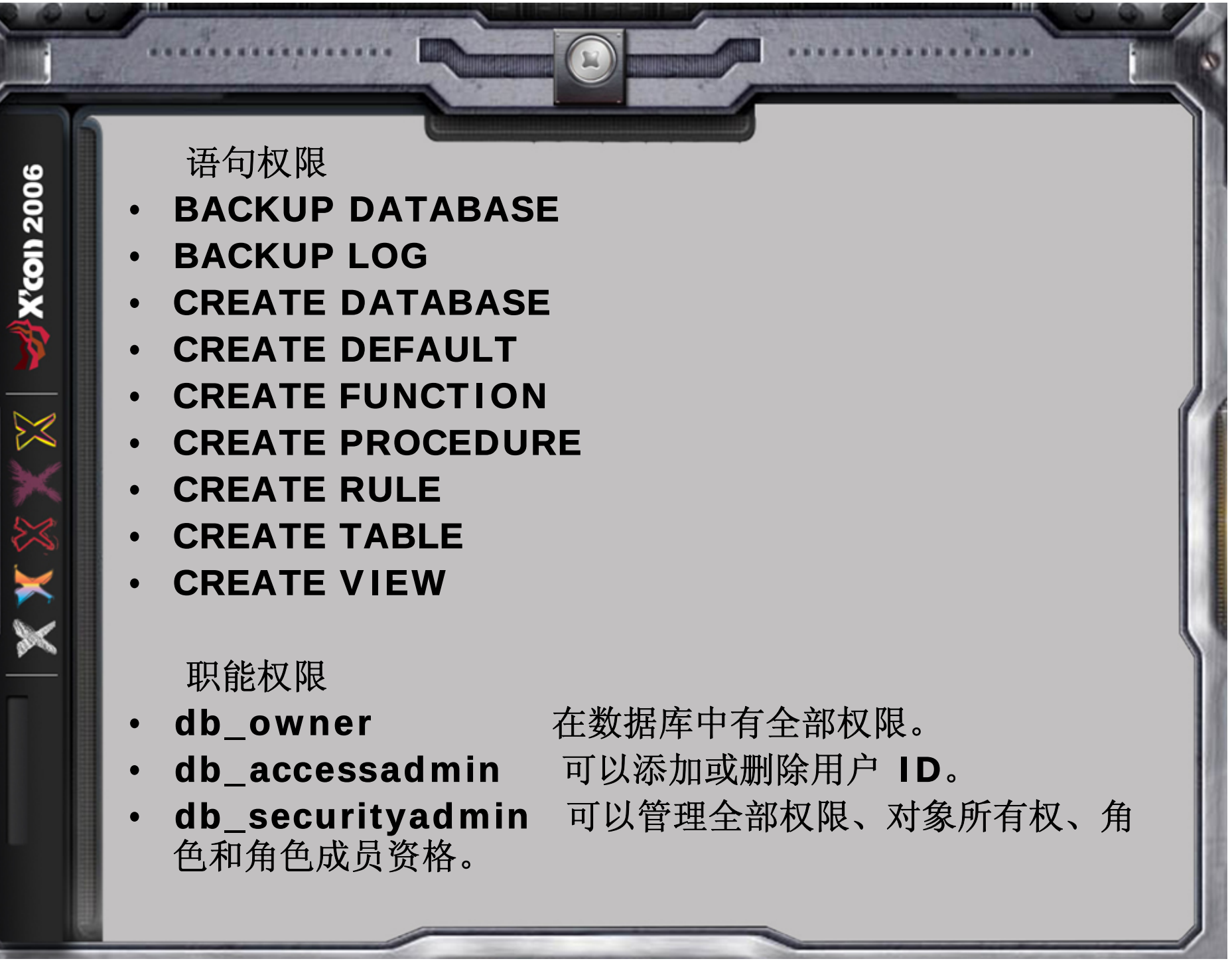
  对于应用服务中不同的模块分配不同的访问权限

  数据库和应用服务本身在系统中权限的限制

  例子**[SQL Server]**:

  对象权限

- **SELECT**、**INSERT**、**UPDATE** 和 **DELETE** 语句权限，它们可以应用到整个表或视图中。
- **SELECT** 和 **UPDATE** 语句权限，它们可以有选择性地应用到表或视图中的单个列上。
- **SELECT** 权限，它们可以应用到用户定义函数。
- **INSERT** 和 **DELETE** 语句权限，它们会影响整行，因此只可以应用到表或视图中，而不能应用到单个列上。
- **EXECUTE** 语句权限，它们可以影响存储过程和函数。

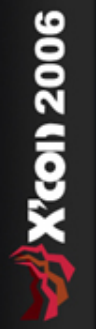


语句权限

- **BACKUP DATABASE**
- **BACKUP LOG**
- **CREATE DATABASE**
- **CREATE DEFAULT**
- **CREATE FUNCTION**
- **CREATE PROCEDURE**
- **CREATE RULE**
- **CREATE TABLE**
- **CREATE VIEW**

职能权限

- **db_owner** 在数据库中有全部权限。
- **db_accessadmin** 可以添加或删除用户 **ID**。
- **db_securityadmin** 可以管理全部权限、对象所有权、角色和角色成员资格。



- **db_ddladmin** 可以发出 **ALL DDL**，但不能发出 **GRANT**、**REVOKE** 或 **DENY** 语句。
- **db_backupoperator** 可以发出 **DBCC**、**CHECKPOINT** 和 **BACKUP** 语句。
- **db_datareader** 可以选择数据库内任何用户表中的所有数据。
- **db_datawriter** 可以更改数据库内任何用户表中的所有数据。
- **db_denydatareader** 不能选择数据库内任何用户表中的任何数据。
- **db_denydatawriter** 不能更改数据库内任何用户表中的任何数据。

存储过程

- **xp_cmdshell**
- **xp_regaddmultistring**
- **xp_regdeletekey**
- **xp_regdeletevalue**
- **xp_regenumkeys**
- **xp_regenumvalues**
- **xp_regread**
- **xp_regremovemultistring**
- **xp_regwrite**

- sp_OACreate
- sp_OADestroy
- sp_OAMethod
- sp_OAGetProperty
- sp_OASetProperty
- sp_OAGetErrorInfo
- sp_OAStop

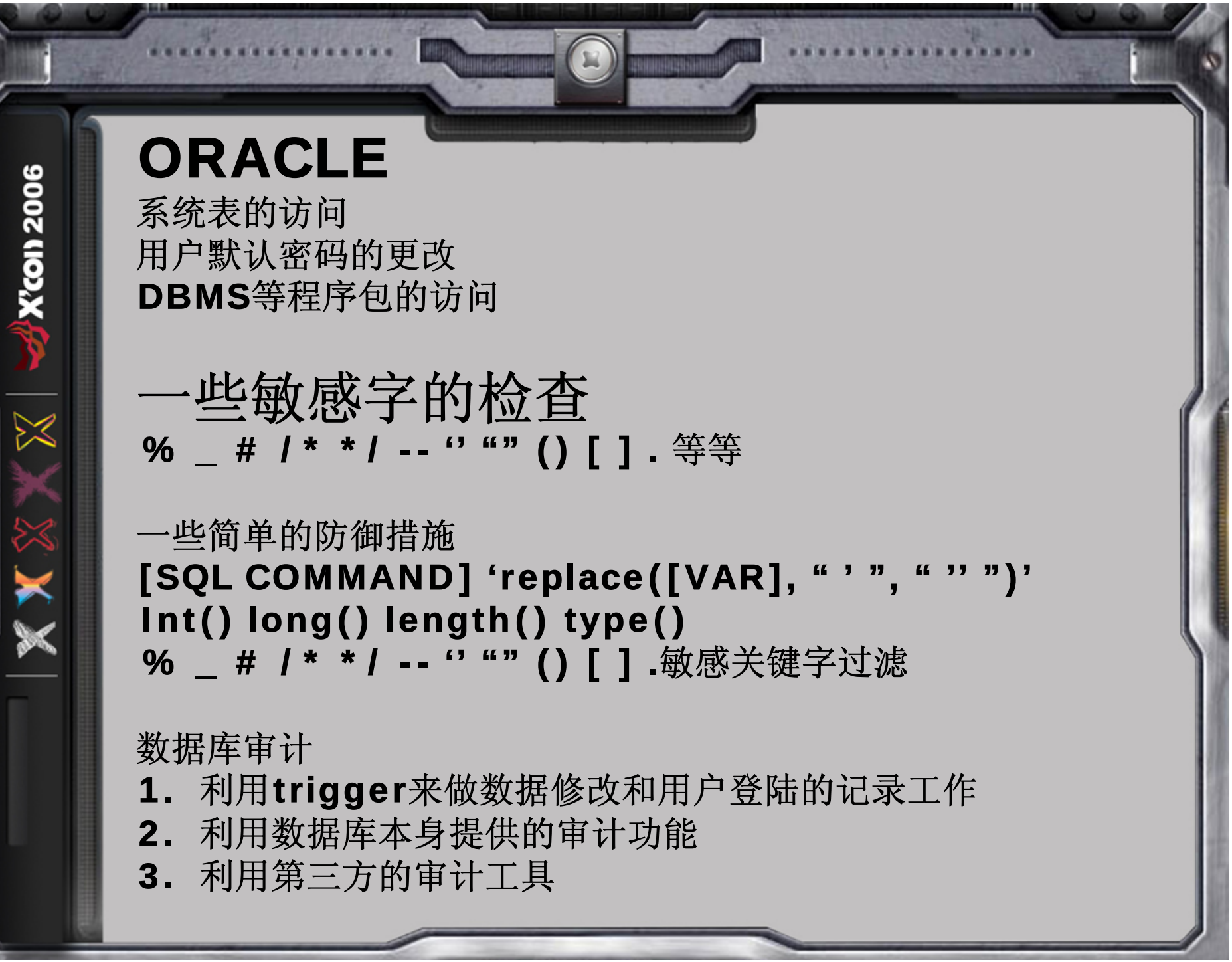
MYSQL

系统表的访问

Into outfile 权限的限制

Into dumpfile 权限的限制

Root的远程访问



ORACLE

系统表的访问
用户默认密码的更改
DBMS等程序包的访问

一些敏感字的检查

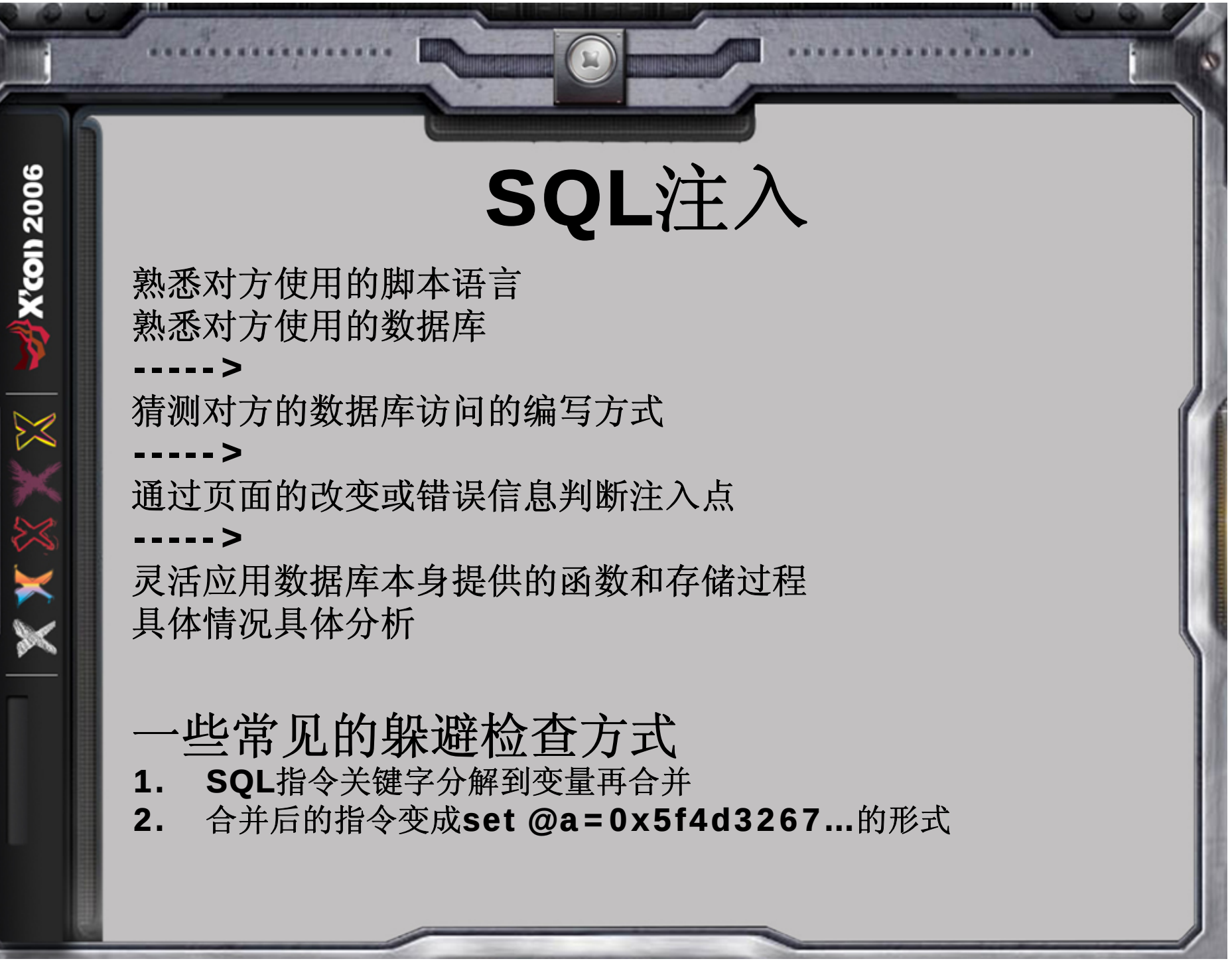
% _ # /* */ -- ‘’ “” () [ ] . 等等

一些简单的防御措施

[SQL COMMAND] ‘replace([VAR], “ ’ ”, “ ’’ ”)’
Int() long() length() type()
% _ # /* */ -- ‘’ “” () [ ] .敏感关键字过滤

数据库审计

1. 利用**trigger**来做数据修改和用户登陆的记录工作
2. 利用数据库本身提供的审计功能
3. 利用第三方的审计工具



SQL注入

熟悉对方使用的脚本语言
熟悉对方使用的数据库
----->
猜测对方的数据库访问的编写方式
----->
通过页面的改变或错误信息判断注入点
----->
灵活应用数据库本身提供的函数和存储过程
具体情况具体分析

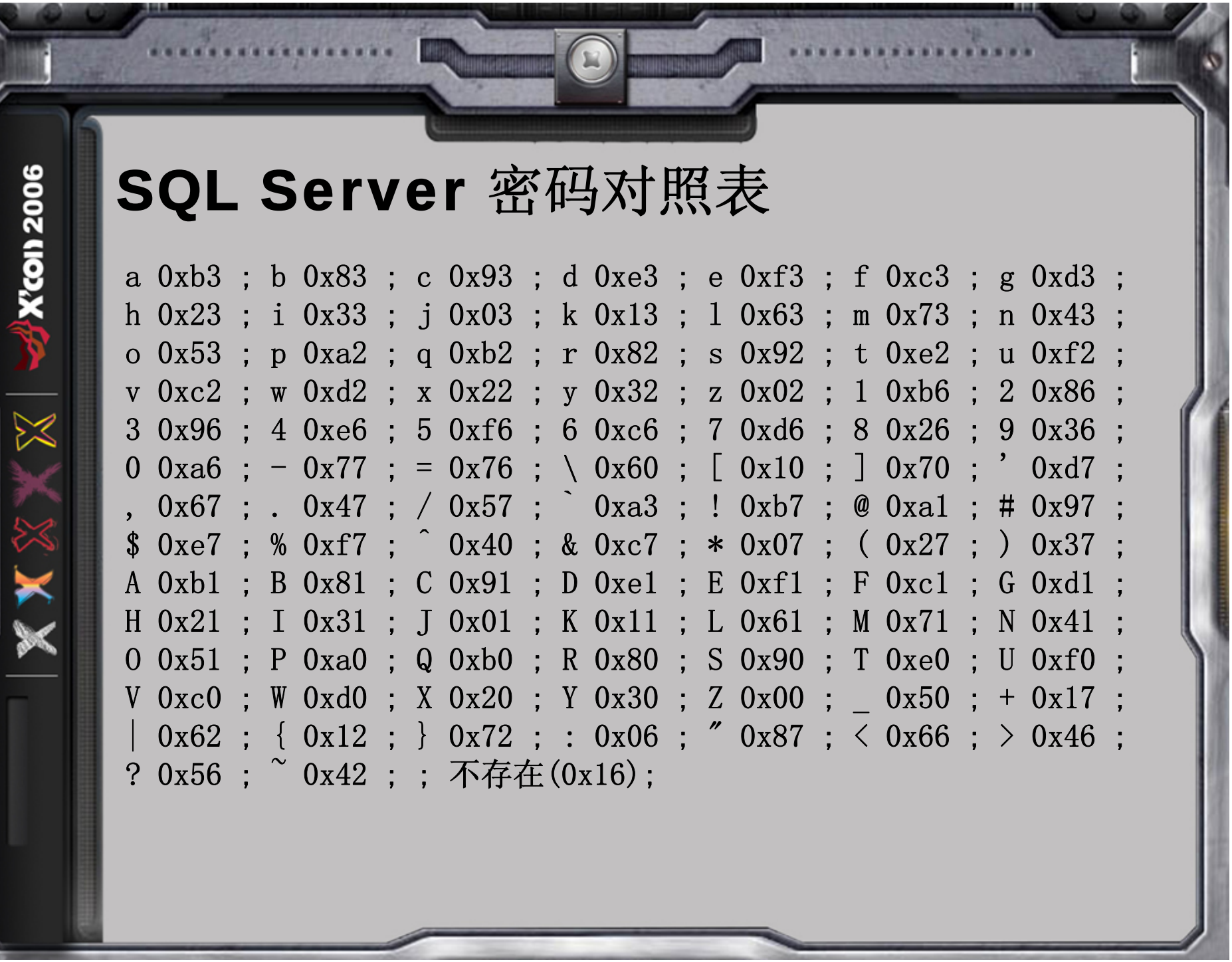
一些常见的躲避检查方式

1. SQL指令关键字分解到变量再合并
2. 合并后的指令变成set @a=0x5f4d3267...的形式

3. set @a=0x5f4d3267...变成%67%7e%5f%ec...的形式
4. Unicode再次分解%2567%257e%255f...的形式
5. /**/ /*xxxxx*/代替空格
6. -- /* # 注释符号

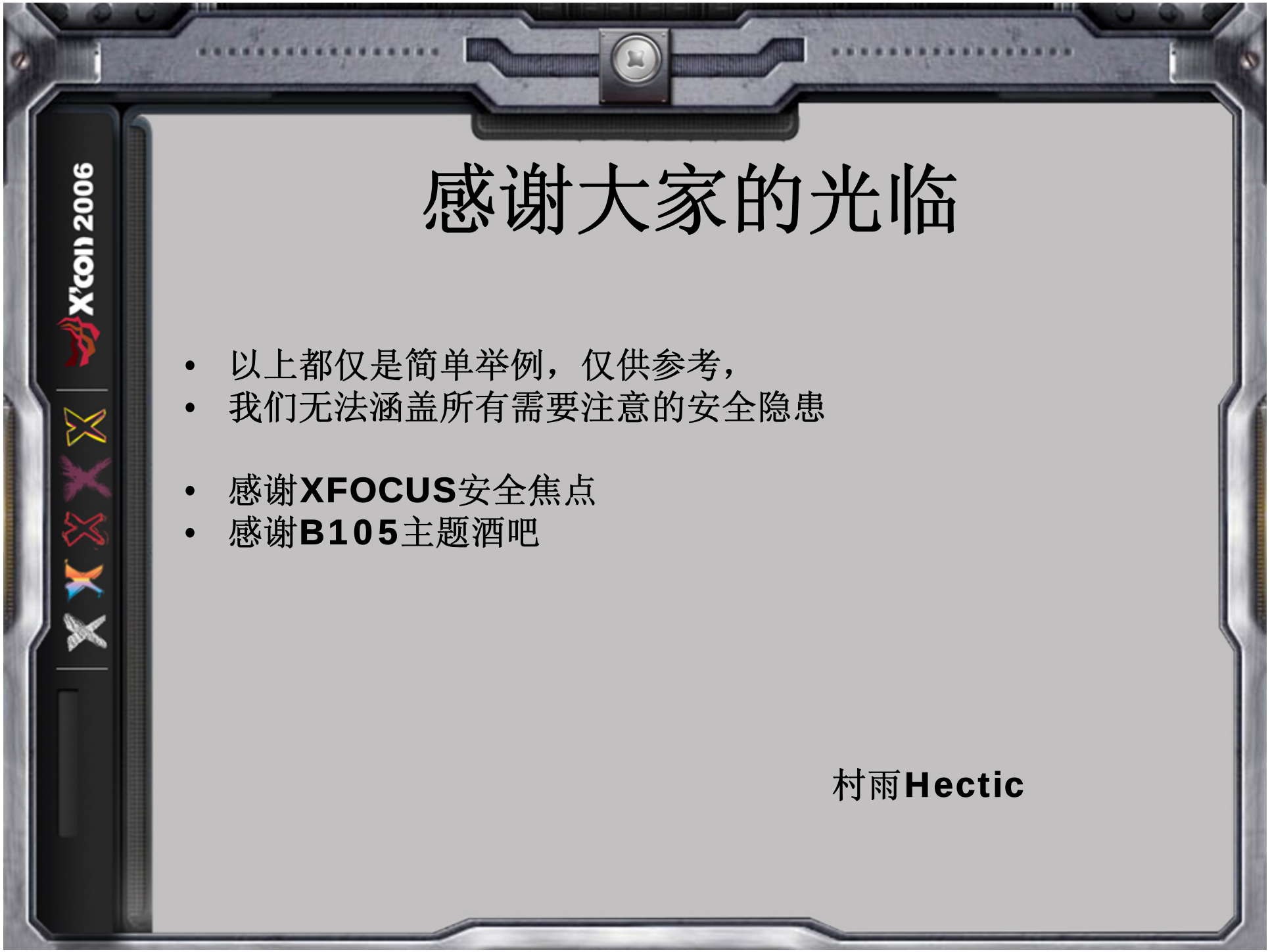
一些常见的入侵形式

- 调用存储过程，函数，程序包
- Union联合查询
- Left length right mid asc ord chr等函数猜测字段数据
- 利用String=integer类型错误返回信息泄漏敏感数据
- Backup database backup log backup database with different 构造webshell
- 将shellcode写入0x5f4d3267...类型的变量，然后调用存在溢出漏洞的存储过程制造溢出从而获得系统权限
- Output file 构造webshell
- Opendatasource openrowset [...]等访问外部资源



SQL Server 密码对照表

a 0xb3 ; b 0x83 ; c 0x93 ; d 0xe3 ; e 0xf3 ; f 0xc3 ; g 0xd3 ;
h 0x23 ; i 0x33 ; j 0x03 ; k 0x13 ; l 0x63 ; m 0x73 ; n 0x43 ;
o 0x53 ; p 0xa2 ; q 0xb2 ; r 0x82 ; s 0x92 ; t 0xe2 ; u 0xf2 ;
v 0xc2 ; w 0xd2 ; x 0x22 ; y 0x32 ; z 0x02 ; 1 0xb6 ; 2 0x86 ;
3 0x96 ; 4 0xe6 ; 5 0xf6 ; 6 0xc6 ; 7 0xd6 ; 8 0x26 ; 9 0x36 ;
0 0xa6 ; - 0x77 ; = 0x76 ; \ 0x60 ; [0x10 ;] 0x70 ; ’ 0xd7 ;
, 0x67 ; . 0x47 ; / 0x57 ; ` 0xa3 ; ! 0xb7 ; @ 0xa1 ; # 0x97 ;
$ 0xe7 ; % 0xf7 ; ^ 0x40 ; & 0xc7 ; * 0x07 ; (0x27 ;) 0x37 ;
A 0xb1 ; B 0x81 ; C 0x91 ; D 0xe1 ; E 0xf1 ; F 0xc1 ; G 0xd1 ;
H 0x21 ; I 0x31 ; J 0x01 ; K 0x11 ; L 0x61 ; M 0x71 ; N 0x41 ;
O 0x51 ; P 0xa0 ; Q 0xb0 ; R 0x80 ; S 0x90 ; T 0xe0 ; U 0xf0 ;
V 0xc0 ; W 0xd0 ; X 0x20 ; Y 0x30 ; Z 0x00 ; _ 0x50 ; + 0x17 ;
| 0x62 ; { 0x12 ; } 0x72 ; : 0x06 ; ″ 0x87 ; < 0x66 ; > 0x46 ;
? 0x56 ; ~ 0x42 ; ; 不存在(0x16);



感谢大家的光临

- 以上都仅是简单举例，仅供参考，
- 我们无法涵盖所有需要注意的安全隐患

- 感谢**XFOCUS**安全焦点
- 感谢**B105**主题酒吧

村雨**Hectic**