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码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译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  FUNCTION MD5(
  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    RETURN retval;
  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        DEFAUTL 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 :=
              dbms_obfuscation_toolkit.des3decrypt(UTL_RAW.CAST_TO_VARCHAR2(iValue),
  6                                                   key_string => raw_key,
  7                                                   which => iMode);
  8        return vDecrypted;
  9     end;
 10
 11     FUNCTION ENC3KEY(iValue in varchar2, iMode in pls_integer) return raw as
 12        vEncrypted       varchar2(4000);
 13        vEncryptedRaw Raw(2048);
 14     begin
 15        vEncrypted := dbms_obfuscation_toolkit.des3encrypt(iValue,
 16                                                   key_string => raw_key,
 17                                                   which          => iMode);
 18
 19        vEncryptedRaw := UTL_RAW.CAST_TO_RAW(vEncrypted);
 20        return vEncryptedRaw;
 21     end;
 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('AAABIPAABAAAFRSAAA')
----------------------------------------------------------------
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
◦◦◦◦◦◦

/
```

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_CRYPTO AS

  ---------------------------------------------------------------------------
  --
  -- PACKAGE NOTES
  --
  -- DBMS_CRYPTO 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                                  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 :
434232324530433439413733453045304341463544333138324545334637383739354330323343424631393031313533
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。

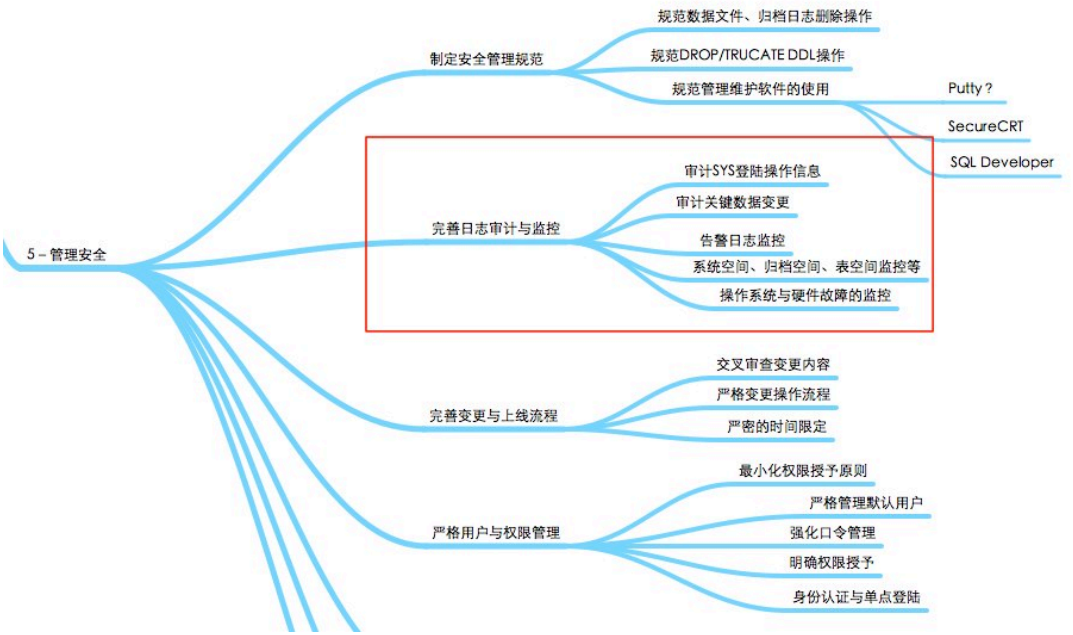
明察秋毫，见微知著

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、可量化的标准和规范，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/onlinelog/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/rdsk/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/rdsk/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/rdsk/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_dailytable_20.270.656595577
     error=15078 txt: ''
Automatic datafile offline due to write error on
file 50: +DG_DATA_03/billbj/datafile/tbs_band_dailytable_20.270.656595577
KCF: write/open error block=0x42ab9 online=1
     file=57 +DG_DATA_03/billbj/datafile/tbs_band_dailytable_20.273.656599591
```