达梦数据库

达梦技术丛书

DM8 安全管理

前言

概述

本文档主要介绍 DM 的安全体系结构，各安全管理模块的功能、原理及相关的数据库管理员和用户如何利用这些安全管理功能对数据库或自己的数据进行安全保护。

读者对象

本文档主要适用于 DM 数据库的：

- 数据库管理员
- 开发工程师
- 测试工程师
- 技术支持工程师

通用约定

在本文档中可能出现下列标志，它们所代表的含义如下：

表 0.1 标志含义

标志	说明
 警告：	表示可能导致系统损坏、数据丢失或不可预知的结果。
 注意：	表示可能导致性能降低、服务不可用。
 小窍门：	可以帮助您解决某个问题或节省您的时间。
 说明：	表示正文的附加信息，是对正文的强调和补充。

在本文档中可能出现下列格式，它们所代表的含义如下：

表 0.2 格式含义

格式	说明
宋体	表示正文。
黑体	标题、警告、注意、小窍门、说明等内容均采用黑体。
Courier new	表示代码或者屏幕显示内容。
粗体	表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。
<>	语法符号中，表示一个语法对象。
::=	语法符号中，表示定义符，用来定义一个语法对象。定义符左边为语法对象，右边为相应的语法描述。
|	语法符号中，表示或者符，限定的语法选项在实际语句中只能出现一个。
{ }	语法符号中，大括号内的语法选项在实际的语句中可以出现 0…N 次(N 为大于 0 的自然数)，但是大括号本身不能出现在语句中。
[ ]	语法符号中，中括号内的语法选项在实际的语句中可以出现 0…1 次，但是中括号本身不能出现在语句中。
关键字	关键字在 DM_SQL 语言中具有特殊意义，在 SQL 语法描述中，关键字以大写形式出现。但在实际书写 SQL 语句时，关键字既可以大写也可以小写。

访问相关文档

如果您安装了 DM 数据库，可在安装目录的“\doc”子目录中找到 DM 数据库的各种手册与技术丛书。

您也可以通过访问我们的网站 www.dameng.com 阅读或下载 DM 的各种相关文档。

联系我们

如果您有任何疑问或是想了解达梦数据库的最新动态消息，请联系我们：

网址：www.dameng.com

技术服务电话：400-991-6599

技术服务邮箱：dmtech@dameng.com

目录

1 概述

1.1 数据库安全管理

数据库安全管理是指采取各种安全措施对数据库及其相关文件和数据进行保护。数据库系统的重要指标之一是确保系统安全，以各种防范措施防止非授权使用数据库，主要通过数据库管理系统进行实现。数据库系统中一般采用用户标识与鉴别、存取控制以及密码存储等技术进行安全控制。

数据库安全的核心和关键是其数据安全。数据安全指以保护措施确保数据的完整性、保密性、可用性、可控性和可审查性。由于数据库存储着大量的重要信息和机密数据，而且在数据库系统中大量数据集中存放，供多用户共享，因此，必须加强对数据库访问的控制和数据安全防护。

1.2 DM 安全管理

1.2.1 DM 安全管理功能系统结构

DM 的安全管理就是为保护存储在 DM 数据库中的各类敏感数据的机密性、完整性和可用性提供必要的技术手段，防止对这些数据的非授权泄露、修改和破坏，并保证被授权用户能按其授权范围访问所需要的数据。

DM 作为安全数据库，提供了包括用户标识与鉴别、自主与强制访问控制、通信与存储加密、审计等丰富的安全功能，且各安全功能都可进行配置，满足各类型用户在安全管理方面不同层次的需求。

DM 的安全管理功能体系结构如图 1.1 所示。

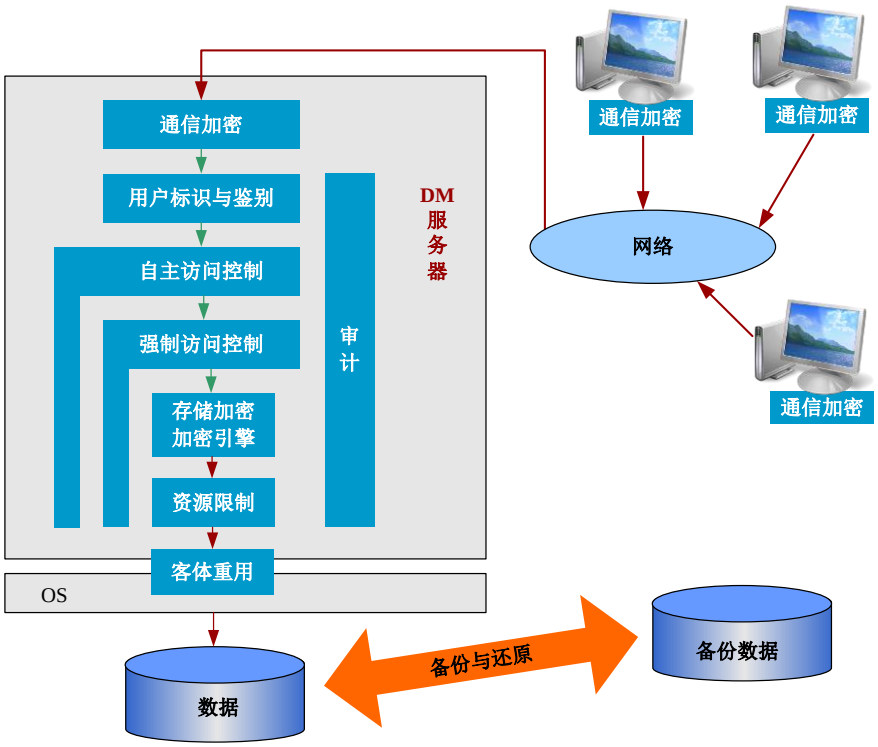

图 1.1 DM 安全功能体系结构

1.2.2 安全管理功能简介

从图 1.1 可以看出 DM 的安全管理功能相当丰富与完善，各安全功能的简介见表 1.1。

表 1.1 DM 安全管理功能简介

安全功能	简介
用户标识与鉴别	可以通过登录帐户区别各用户，并通过口令方式防止用户被冒充
自主访问控制	通过权限管理，使用户只能访问自己权限内的数据对象
强制访问控制	通过安全标记，使用户只能访问与自己安全级别相符的数据对象
审计	审计人员可以查看所有用户的操作记录，为明确事故责任提供证据支持
通信、存储加密	用户可以自主的将数据以密文的形式存储在数据库中。也可以对在网络上传输的数据进行加密
加密引擎	用户可以用自定义的加密算法来加密自己的核心数据
资源限制	可以对网络资源和磁盘资源进行配额设置，防止恶意资源抢占
客体重用	实现了内存与磁盘空间的释放清理，防止信息数据的泄露

1.3 DM 安全版本

DM 数据库版本有标准版、企业版和安全版的区别，不同版本提供的功能有所不同，且需要与版本匹配的 License 才能正常提供服务。标准版和企业版在安全功能上没有区别，因此在本文中，将标准版和企业版统称为普通版本，以与安全版本进行区分。

普通版本提供的安全管理功能有：用户标识与鉴别、自主访问控制、审计、通信加密、存储加密、加密引擎、资源限制与客体重用；安全版本在普通版本的基础上，还提供“四权分立”安全机制和强制访问控制安全管理功能，能满足对安全管理有更高需求的数据库用户的要求。

另外，在普通版本提供的安全功能中，有一些具体的更强的安全功能细节也仅在安全版本提供，将在后面具体介绍的地方一一指出。

1.4 一点说明

本书中的示例，除了具体给出建表语句的除外，其余使用的表均为 DM 示例库中的表。DM 示例库可在安装 DM 数据库时选择安装，其具体说明可参考《DM8_SQL 语言使用手册》。

2 用户标识与鉴别

用户标识与鉴别对试图登录数据库进行数据访问的用户进行身份验证，以确认此用户是否能与某一数据库用户进行关联，并根据关联的数据库用户的权限对此用户在数据库中的数据访问活动进行安全控制。

2.1 DM 的管理用户

在现实生活中，任何一个系统如果将所有的权利都赋予给某一个人，而不加以监督和控制，势必会产生权利滥用的风险。从数据库安全角度出发，一个大型的数据库系统有必要将数据库系统的权限分配给不同的角色来管理，并且各自偏重于不同的工作职责，使之能够互相限制和监督，从而有效保证系统的整体安全。

DM 数据库采用"三权分立"或"四权分立"的安全机制，将系统中所有的权限按照类型进行划分，为每个管理员分配相应的权限，管理员之间的权限相互制约又相互协助，从而使整个系统具有较高的安全性和较强的灵活性。

可在创建 DM 数据库时通过建库参数 PRIV_FLAG 设置使用"三权分立"或"四权分立"安全机制，0 表示"三权分立"，1 表示"四权分立"。此参数仅在 DM 安全版本下提供，即仅 DM 安全版本提供"四权分立"安全机制，缺省采用"三权分立"安全机制。

使用"三权分立"安全机制时，将系统管理员分为数据库管理员、数据库安全员和数据库审计员三种类型。在安装过程中，DM 数据库会预设数据库管理员账号 SYSDBA、数据库安全员账号 SYSSSO 和数据库审计员账号 SYSAUDITOR，其缺省口令都与用户名一致。

使用"四权分立"的安全机制时，将系统管理员分数据库管理员、数据库对象操作员、数据库安全员和数据库审计员四种类型，在"三权分立"的基础上，新增数据库对象操作员账户 SYSDBO，其缺省口令为 SYSDBO。

各管理员应在安装过程中或安装完毕后立即修改缺省口令，避免因口令泄漏造成的安全问题。

- **数据库管理员（DBA）**

每个数据库至少需要一个 DBA 来管理，DBA 可能是一个团队，也可能是一个人。在不同的数据库系统中，数据库管理员的职责可能也会有比较大的区别，总体而言，数据库管理员的职责主要包括以下任务：

- 评估数据库服务器所需的软、硬件运行环境
- 安装和升级 DM 服务器
- 数据库结构设计
- 监控和优化数据库的性能
- 计划和实施备份与故障恢复

■ **数据库安全员（SSO）**

有些应用对于安全性有着很高的要求，传统的由 DBA 一人拥有所有权限并且承担所有职责的安全机制可能无法满足企业实际需要，此时数据库安全员和数据库审计员两类管理用户就显得异常重要，它们对于限制和监控数据库管理员的所有行为都起着至关重要的作用。

数据库安全员的主要职责是制定并应用安全策略，强化系统安全机制。数据库安全员 SYSSSO 是 DM 数据库初始化的时候就已经创建好的，可以以该用户登录到 DM 数据库来创建新的数据库安全员。

SYSSSO 或者新的数据库安全员都可以制定自己的安全策略，在安全策略中定义安全级别、范围和组，然后基于定义的安全级别、范围和组来创建安全标记，并将安全标记分别应用到主体（用户）和客体（各种数据库对象，如表、索引等），以便启用强制访问控制功能。

数据库安全员不能对用户数据进行增、删、改、查，也不能执行普通的 DDL 操作如创建表、视图等。他们只负责制定安全机制，将合适的安全标记应用到主体和客体，通过这种方式可以有效的对 DBA 的权限进行限制，DBA 此后就不能直接访问添加有安全标记的数据，除非安全员给 DBA 也设定了与之匹配的安全标记，DBA 的权限受到了有效的约束。数据库安全员也可以创建和删除新的安全用户，向这些用户授予和回收安全相关的权限，具体参见3.3 角色权限的分配与回收。

■ **数据库审计员（AUDITOR）**

我们可以想象一下，某个企业内部 DBA 非常熟悉公司内部 ERP 系统的数据库设计，该系统包括了员工工资表，里面记录了所有员工的工资，公司的出纳通过查询系统内部员工工资表来发放工资。传统的 DBA 集所有权利于一身，可以很容易修改工资表，从而导致公司工资账务错乱。为了预防该问题，可以采用前面数据库安全员制定安全策略的方法，避免 DBA 或者其他数据库用户具有访问该表的权限。为了能够及时找到 DBA 或者其他用户的非

法操作，在 DM 数据库中还可以在系统建设初期，由数据库审计员（SYSAUDITOR 或者其他由 SYSAUDITOR 创建的审计员）来设置审计策略（包括审计对象和操作），在需要时，数据库审计员可以查看审计记录，及时分析并查找出幕后真凶。

从上面的介绍中我们也可以看出，在 DM 数据库中，审计员的主要职责就是创建和删除数据库审计员，设置/取消对数据库对象和操作的审计设置，查看和分析审计记录等。

- **数据库对象操作员（DBO）**

数据库对象操作员是"四权分立"新增加的一类用户，可以创建数据库对象，并对自己拥有的数据库对象（表、视图、存储过程、序列、包、外部链接等）具有所有的对象权限并可以授出与回收，但其无法管理与维护数据库对象。

2.2 如何创建用户

数据库系统在运行的过程中，往往需要根据实际需求创建用户，然后为用户指定适当的权限。创建用户的操作一般只能由系统预设用户 SYSDBA、SYSSSO 和 SYSAUDITOR 完成，如果普通用户需要创建用户，必须具有 CREATE USER 的数据库权限。

使用 MANAGER 执行 CREATE USER 语句创建的用户会同时具备 SVI（此时 SVI 属于 PUBLIC）和 VTI 角色权限。使用 DIsql 执行 CREATE USER 语句创建的用户只具备 SVI 角色权限。CREATE USER 语句创建的用户无 SOI 权限，如果想让这些用户能够查看系统表，还需要额外授予 SOI 权限。CREATE USER 创建的新用户不具备 SOI 权限。

2.2.1 创建用户

创建用户的命令是 CREATE USER，创建用户所涉及的内容包括为用户指定用户名、认证模式、口令、口令策略、空间限制、只读属性以及资源限制。其中用户名是代表用户账号的标识符，长度为 1~128 个字符。用户名可以用双引号括起来，也可以不用，但如果用户名以数字开头，必须用双引号括起来。

在 DM 中使用 CREATE USER 语句创建用户，具体的语法格式如下：

```
CREATE USER <用户名>IDENTIFIED<身份验证模式>  [PASSWORD_POLICY <口令策略>][<锁定子句>][<存储加密密钥>][<空间限制子句>][<只读标志>][<资源限制子句>][<允许 IP 子句>][<禁止 IP 子句>][<允许时间子句>][<禁止时间子句>][< TABLESPACE 子句>];
<身份验证模式> ::= <数据库身份验证模式>|<外部身份验证模式>
```

<数据库身份验证模式> ::= BY <口令>[<散列选项>]

<散列选项> ::= HASH WITH [<密码引擎名>.]<散列算法> [<加盐选项>]

<散列算法> ::= MD5 | SHA1 | SHA224 | SHA256 | SHA384 | SHA512

<加盐选项> ::= [NO] SALT

<外部身份验证模式> ::= EXTERNALLY | EXTERNALLY AS <用户 DN>

<口令策略> ::= 口令策略项的任意组合

<锁定子句> ::= ACCOUNT LOCK | ACCOUNT UNLOCK

<存储加密密钥> ::= ENCRYPT BY <口令>

<空间限制子句> ::= DISKSPACE LIMIT <空间大小>| DISKSPACE UNLIMITED

<只读标志> ::= READ ONLY | NOT READ ONLY

<资源限制子句> ::= DROP PROFILE |

PROFILE <profile 名> |

[LIMIT <资源设置>]

<资源设置> ::= <资源设置项>{,<资源设置项>} |

<资源设置项>{ <资源设置项>}

<资源设置项> ::= SESSION_PER_USER <参数设置>|

CONNECT_IDLE_TIME <参数设置>|

CONNECT_TIME <参数设置>|

CPU_PER_CALL <参数设置>|

CPU_PER_SESSION <参数设置>|

MEM_SPACE <参数设置>|

READ_PER_CALL <参数设置>|

READ_PER_SESSION <参数设置>|

FAILED_LOGIN_ATTEMPS <参数设置>|

PASSWORD_LIFE_TIME <参数设置>|

PASSWORD_REUSE_TIME <参数设置>|

PASSWORD_REUSE_MAX <参数设置>|

PASSWORD_LOCK_TIME <参数设置>|

PASSWORD_GRACE_TIME <参数设置>

<参数设置> ::=<参数值>| UNLIMITED| DEFAULT

```
<允许 IP 子句> ::= ALLOW_IP <IP 项>{,<IP 项>}
<禁止 IP 子句> ::= NOT_ALLOW_IP <IP 项>{,<IP 项>}
<IP 项> ::= <具体 IP>|<网段>
<允许时间子句> ::= ALLOW_DATETIME <时间项>{,<时间项>}
<禁止时间子句> ::= NOT_ALLOW_DATETIME <时间项>{,<时间项>}
<时间项> ::= <具体时间段> | <规则时间段>
<具体时间段> ::= <具体日期><具体时间> TO <具体日期><具体时间>
<规则时间段> ::= <规则时间标志><具体时间> TO <规则时间标志><具体时间>
<规则时间标志> ::= MON | TUE | WED | THURS | FRI | SAT | SUN
<TABLESPACE 子句> ::=DEFAULT TABLESPACE <表空间名>
```


注意：

外部身份验证仅在 DM 安全版本中才提供支持。

资源限制子句将在第 9 章进行详细介绍。

例如，创建用户名为 BOOKSHOP_USER、口令为 BOOKSHOP_PASSWORD、会话超时为 3 分钟的用户。

```
CREATE USER BOOKSHOP_USER IDENTIFIED BY BOOKSHOP_PASSWORD LIMIT CONNECT_TIME
3;
```

2.2.2 口令策略

用户口令最长为 48 字节，创建用户语句中的 PASSWORD POLICY 子句用来指定该用户的口令策略，系统支持的口令策略有：

- 0　无限制。但总长度不得超过 48 个字节
- 1　禁止与用户名相同
- 2　口令长度需大于等于 INI 参数 PWD_MIN_LEN 设置的值
- 4　至少包含一个大写字母（A-Z）
- 8　至少包含一个数字（0-9）
- 16　至少包含一个标点符号（英文输入法状态下，除"和空格外的所有符号）

口令策略可单独应用，也可组合应用。组合应用时，如需要应用策略 2 和 4，则设置口令策略为 2+4=6 即可。

除了在创建用户语句中指定该用户的口令策略，DM 的 INI 参数 PWD_POLICY 可以指定系统的默认口令策略，其参数值的设置规则与 PASSWORD_POLICY <口令策略>子句一致，缺省值为 2。若在创建用户时没有使用 PASSWORD_POLICY <口令策略>子句指定用户的口令策略，则使用系统的默认口令策略。

系统管理员可通过查询 V$PARAMETER 动态视图查询 PWD_POLICY 的当前值。

```
SELECT * FROM V$PARAMETER WHERE NAME= 'PWD_POLICY';
```

PWD_POLICY 参数是比较特殊，属于不包含在 DM.INI 中的 INI 配置项，因此不能使用 Console 工具进行修改。DBA 用户可以使用系统过程 SP_SET_PARA_VALUE 来配置 PWD_POLICY 参数值，关于 SP_SET_PARA_VALUE 的详细介绍请参考《DM8_SQL 语言使用手册》。

例如，将 PWD_POLICY 置为 8，同时修改文件和内存参数，由于 PWD_POLICY 为动态 INI 参数，这样设置后新的参数值可以立即生效。

```
SP_SET_PARA_VALUE(1, 'PWD_POLICY',8);
```


说明：

DM.INI 文件中包含的参数都可以使用 Console 工具或调用系统过程 SP_SET_PARA_VALUE/SP_SET_PARA_DOUBLE_VALUE/ SP_SET_PARA_STRING_VALUE 方法来进行设置。

因为 Console 为脱机工具，对参数值的修改通过修改 dm.ini 文件中的参数值来进行，无论参数类型是静态还是动态，都需要重启 DM 服务器才能使新设置的参数值生效。

2.3 用户身份验证模式

DM 提供数据库身份验证模式和外部身份验证模式来保护对数据库访问的安全。数据库身份验证模式需要利用数据库口令，即在创建或修改用户时指定用户口令，用户在登录时输入对应口令进行身份验证；外部身份验证模式支持基于操作系统(OS)的身份验证、LDAP 身份验证和 KERBEROS 身份验证。

2.3.1 基于操作系统的身份验证

基于 OS 的身份验证分为本机验证和远程验证，本机验证需要将 DM 配置文件 dm.ini 的 ENABLE_LOCAL_OSAUTH 参数设置为 1（缺省为 0）；而远程验证需要将 DM 配置文件 dm.ini 的 ENABLE_REMOTE_OSAUTH 参数设置为 1（缺省为 0），表示支持远程验证，同时还要将 dm.ini 的 ENABLE_ENCRYPT 参数设置为 1（缺省为 0），表示采用 SSL 安全连接。这三个参数均为静态参数，数据库管理员可以使用系统过程 SP_SET_PARA_VALUE 进行修改，但修改后需要重新启动 DM 服务器才能生效。

基于 OS 的身份验证需要首先将操作系统用户加入到操作系统的 dmdba|dmsso|dmauditor 用户组，分别对应数据库的 SYSDBA|SYSSSO|SYSAUDITOR 用户。也可以将操作系统用户加入到操作系统的 dmusers 用户组来进行基于 OS 的身份验证，对应数据库的同名用户，即此时数据库中需要存在一个与操作系统用户同名的用户。

例如，若当前操作系统用户名为 dameng，则 dameng 基于 OS 的身份验证登录数据库的操作步骤如下：

1.在数据库中创建一个操作系统的同名用户，并修改 INI 参数。

```
SQL> CREATE USER dameng IDENTIFIED BY "dameng12345";

SQL> SP_SET_PARA_VALUE(2,'ENABLE_LOCAL_OSAUTH',1);

SQL> COMMIT;
```

重新启动数据库服务器，使 INI 参数生效。

2.在操作系统中创建 dmusers 用户组，并将操作系统用户 dameng 添加到 dmusers 用户组。

Windows 操作系统下，以管理员身份运行命令提示符工具并执行以下命令：

```
C:\Windows\system32>net localgroup dmusers /add

C:\Windows\system32>net localgroup dmusers dameng /add
```

Linux 操作系统下：

```
[dameng@test163 ~]$ sudo groupadd dmusers

[dameng@test163 ~]$ sudo usermod -G dmusers dameng
```

3.操作系统用户 dameng 基于 OS 的身份验证登录数据库。

以 Windows 操作系统为例：

```
D:\dmdbms\bin>disql.exe /@localhost:5236 as users
```

注意：

基于操作系统的身份验证仅在 DM 安全版本中才提供支持。

DM 的 disql、dmfldr、dmdbg、dexp 和 dimp 等工具都支持基于操作系统的身份验证登录，具体登录方式可查看各工具操作手册。

2.3.2 LDAP 身份验证

DM 提供对 LDAP 的支持，主要利用 LDAP 服务器存储的账户数据信息，验证账户是否是数据库的合法用户。我们假定企业已经部署了 LDAP 服务器和 CA 服务器（或者花钱购买了证书认证）。

要使用 LDAP 身份验证，首先要在 DM 配置文件 dm.ini 中添加参数 LDAP_HOST。此参数为 LDAP 数据源所在机器的主库名，假如主库名为 ldapserver.dameng.org，则 LDAP_HOST = ldapserver.dameng.org。

LDAP 身份验证创建用户的语法如下：

```
CREATE USER <用户名>IDENTIFIED EXTERNALLY AS <用户 DN> ......;
```

说明：

“......”表示 2.2 节中创建用户语法从“<锁定子句>”开始往后的语法部分。

语法中的“用户 DN”指 LDAP 数据源中用户 DN，可以唯一标识每一个用户。

例如，如果域名为 dameng.org，若以 Administrator 用户登录，按 DN 规范书写则 certificate_dn 为 cn=Administrator,cn=users,dc=dameng,dc=org。则创建用户语法可以写为：

Windows 操作系统下：

```
CREATE USER USER01 IDENTIFIED EXTERNALLY AS 'cn=Administrator,cn=users,
dc=dameng,dc=org';
```

Linux 操作系统下：

```
CREATE USER USER01 IDENTIFIED EXTERNALLY AS 'cn=root, dc=dameng,dc=org';
```

当用户使用 DM 客户端或接口登录 DM 数据库时，将用户名写为“/username”的形式，则表示以 LDAP 验证方式进行登录，登录密码为 LDAP 数据源中创建用户时使用的密码。

2.3.3 KERBEROS 身份验证

KERBEROS 是为 TCP/IP 网络系统设计的可信的第三方认证协议，DM 支持 KERBEROS 身份验证。我们假设用户已经正确配置了 KERBEROS 使用环境。

KERBEROS 身份验证创建用户的语法如下：

```
CREATE USER <用户名>IDENTIFIED EXTERNALLY ......;
```

说明： **“......”表示 2.2 节中创建用户语法从“<锁定子句>”开始往后的语法部分。**

当用户使用 DM 客户端或接口登录 DM 数据库时，将用户名写为“///username”的形式，则表示以 KERBEROS 验证方式进行登录，不需要输入密码。

2.3.4 UKEY 身份验证

DM 支持 UKEY 身份验证，通过 INI 参数 CLIENT_UKEY 进行 UKEY 验证设置，缺省值为 0：

0：支持所有登录方式，是否采用 UKEY 验证由客户端驱动。当客户端输入 UKEY 名称和 UKEY_PIN 则采用 UKEY 验证，否则不使用 UKEY 验证；

1：客户端强制使用 UKEY 验证。

使用 UKEY 身份验证需要按照 8.1 节的介绍编写动态库，具体请参考 8.3 节的介绍。

2.4 如何修改用户信息

为了防止不法之徒盗取用户的口令，用户应该经常改变自己的口令。用户的口令不应该是类似 12345，abcdef 这样简单的字符串，更不要指定为自己的生日或姓名，也不要指定为一个英文单词，因为这样的口令很容易被破解。一个好的口令应该包含大小写字母、数字、特殊符号在内的混合字符串。统计表明，一个口令中包含的成分越复杂，就越难破译。

修改用户口令的操作一般由用户自己完成，SYSDBA、SYSSSO、SYSAUDITOR 可以无条件修改同类型的用户的口令。普通用户只能修改自己的口令，如果需要修改其他用户的口令，必须具有 ALTER USER 数据库权限。修改用户口令时，口令策略应符合创建该用户时指定的口令策略。

具有 ALTER USER 权限的用户可以修改包括管理员在内的所有用户的口令和用户资源限制，因此对 ALTER USER 权限的授予应慎之又慎。

使用 ALTER USER 语句可修改用户口令。除口令外，这个语句还可以修改用户的口令策略、空间限制、只读属性以及资源限制等。

ALTER USER 的语法与创建用户的语法极为相似，具体语法格式如下：

```
ALTER USER <用户名> [IDENTIFIED <身份验证模式>][PASSWORD_POLICY <口令策略>][<锁定子句>][<存储加密密钥>][<空间限制子句>][<只读标志>][<资源限制子句>][<允许 IP 子句>][<禁止 IP 子句>][<允许时间子句>][<禁止时间子句>][<TABLESPACE 子句>][<SCHEMA 子句>];

每个子句的具体语法参见创建用户语法说明
```

例如，下面的语句修改用户 BOOKSHOP_USER 的空间限制为 20M。

```
ALTER USER BOOKSHOP_USER DISKSPACE LIMIT 20;
```

不论 DM 的 INI 参数 DDL_AUTO_COMMIT 设置为自动提交还是非自动提交，ALTER USER 操作都会被自动提交。

2.5 如何删除用户

当一个用户不再需要访问数据库系统时，应将这个用户及时地从数据库中删除，否则可能会有安全隐患。

删除用户的操作一般由 SYSDBA、SYSSSO、SYSAUDITOR 完成，他们可以删除同类型的其他用户。普通用户要删除其他用户，需要具有 DROP USER 权限。

使用 DROP USER 语句删除语句，语法格式为：

```
DROP USER [IF EXISTS] <用户名> [RESTRICT | CASCADE];
```

一个用户被删除后，这个用户本身的信息，以及它所拥有的数据库对象的信息都将从数据字典中被删除。

指定 IF EXISTS 关键字后，删除不存在的用户时不会报错，否则会报错。

如果在删除用户时未使用 CASCADE 选项（缺省使用 RESTRICT 选项），若该用户建立了数据库对象，DM 将返回错误信息，而不删除此用户。

如果在删除用户时使用了 CASCADE 选项，除数据库中该用户及其创建的所有对象被删除外，若其他用户创建的对象引用了该用户的对象，DM 还将自动删除相应的引用完整性约束及依赖关系。

例如，假设用户 BOOKSHOP_USER 建立了自己的表或其他数据库对象，执行下面的语句：

```
DROP USER BOOKSHOP_USER;
```

将提示错误信息“试图删除被依赖对象[BOOKSHOP_USER]”。

下面的语句则能成功执行，会将 BOOKSHOP_USER 所建立的数据库对象一并删除。

```
DROP USER BOOKSHOP_USER CASCADE;
```

正在使用中的用户可以被其他具有 DROP USER 权限的用户删除，被删除的用户继续做操作或尝试重新连接数据库会报错。

3 自主访问控制

自主访问控制（Discretionary Access Control，DAC）是这样一种访问控制方式：由数据库对象的拥有者自主决定是否将自己拥有的对象的部分或全部访问权限授予其他用户。也就是说，在自主访问控制下，用户可以按照自己的意愿，有选择地与其他用户共享他拥有的数据库对象。

3.1 权限管理

DM 数据库对用户的权限管理有着严密的规定，如果没有权限，用户将无法完成任何操作。

用户权限有两类：数据库权限和对象权限。数据库权限主要是指针对数据库对象的创建、删除、修改的权限，对数据库备份等权限。而对象权限主要是指对数据库对象中的数据的访问权限。数据库权限一般由 SYSDBA、SYSAUDITOR 和 SYSSSO 指定，也可以由具有特权的其他用户授予。对象权限一般由数据库对象的所有者授予用户，也可由 SYSDBA 用户指定，或者由具有该对象权限的其他用户授权。

3.1.1 数据库权限

数据库权限是与数据库安全相关的非常重要的权限，其权限范围比对象权限更加广泛，因而一般被授予数据库管理员或者一些具有管理功能的角色。数据库权限与 DM 预定义角色有着重要的联系，一些数据库权限由于权力较大，只集中在几个 DM 系统预定义角色中，且不能转授，我们将在 3.2.1.1 中进行具体介绍。

DM 提供了 100 余种数据库权限，表 3.1 列出了最常用的几种数据库权限。完整的数据库权限列表将在 3.2.1.1 中与 DM 预定义角色一起进行介绍。

表 3.1 常用的几种数据库权限

数据库权限	说明
CREATE TABLE	在自己的模式中创建表的权限
CREATE VIEW	在自己的模式中创建视图的权限
CREATE USER	创建用户的权限
CREATE TRIGGER	在自己的模式中创建触发器的权限

ALTER USER	修改用户的权限
ALTER DATABASE	修改数据库的权限
CREATE PROCEDURE	在自己模式中创建存储程序的权限

不同类型的数据库对象，其相关的数据库权限也不相同。

例如，对于表对象，相关的数据库权限包括：

- CREATE TABLE：创建表
- CREATE ANY TABLE：在任意模式下创建表
- ALTER ANY TABLE：修改任意表
- DROP ANY TABLE：删除任意表
- INSERT TABLE：插入表记录
- INSERT ANY TABLE：向任意表插入记录
- UPDATE TABLE：更新表记录
- UPDATE ANY TABLE：更新任意表的记录
- DELETE TABLE：删除表记录
- DELETE ANY TABLE：删除任意表的记录
- SELECT TABLE：查询表记录
- SELECT ANY TABLE：查询任意表的记录
- REFERENCES TABLE：引用表
- REFERENCES ANY TABLE：引用任意表
- DUMP TABLE：导出表
- DUMP ANY TABLE：导出任意表
- GRANT TABLE：向其他用户进行表上权限的授权
- GRANT ANY TABLE：向其他用户进行任意表上权限的授权

而对于存储程序对象，其相关的数据库权限则包括：

- CREATE PROCEDURE：创建存储程序
- CREATE ANY PROCEDURE：在任意模式下创建存储程序
- DROP PROCEDURE：删除存储程序
- DROP ANY PROCEDURE：删除任意存储程序
- EXECUTE PROCEDURE：执行存储程序
- EXECUTE ANY PROCEDURE：执行任意存储程序

- GRANT PROCEDURE：向其他用户进行存储程序上权限的授权
- GRANT ANY PROCEDURE：向其他用户进行任意存储程序上权限的授权

需要说明的是，表、视图、触发器、存储程序等对象为模式对象，在默认情况下对这些对象的操作都是在当前用户自己的模式下进行的。如果要在其他用户的模式下操作这些类型的对象，需要具有相应的 ANY 权限。例如，要能够在其他用户的模式下创建表，当前用户必须具有 CREATE ANY TABLE 数据库权限，如果希望能够在其他用户的模式下删除表，必须具有 DROP ANY TABLE 数据库权限。

说明： **DM 中有一个较特殊的数据库权限“CREATE SESSSION”，表示创建会话连接数据库的权限。系统预设的管理员用户都具备此权限，新建用户缺省也具备此权限，管理员可根据实际需要回收指定用户的 CREATE SESSION 权限以限制该用户连接数据库。**

3.1.2 对象权限

对象权限主要是对数据库对象中的数据的访问权限，主要用来授予需要对某个数据库对象的数据进行操纵的数据库普通用户。表 3.2 列出了主要的对象权限。

表 3.2 常用的对象权限

数据库对象类型 对象权限	表	视图	存储程序	包	类	类型	序列	目录	域
SELECT	√	√					√		
INSERT	√	√							
DELETE	√	√							
UPDATE	√	√							
REFERENCES	√								
DUMP	√								
EXECUTE			√	√	√	√		√	
READ								√	
WRITE								√	
USAGE									√

SELECT、INSERT、DELETE 和 UPDATE 权限分别是针对数据库对象中的数据的查询、插入、删除和修改的权限。对于表和视图来说，删除操作是整行进行的，而查询、插入和修

改却可以在一行的某个列上进行，所以在指定权限时，DELETE 权限只要指定所要访问的表就可以了，而 SELECT、INSERT 和 UPDATE 权限还可以进一步指定是对哪个列的权限。

表对象的 REFERENCES 权限是指可以与一个表建立关联关系的权限，如果具有了这个权限，当前用户就可以通过自己的一个表中的外键，与对方的表建立关联。关联关系是通过主键和外键进行的，所以在授予这个权限时，可以指定表中的列，也可以不指定。

存储程序等对象的 EXECUTE 权限是指可以执行这些对象的权限。有了这个权限，一个用户就可以执行另一个用户的存储程序、包、类等。

目录对象的 READ 和 WRITE 权限指可以读或写访问某个目录对象的权限。

域对象的 USAGE 权限指可以使用某个域对象的权限。拥有某个域的 USAGE 权限的用户可以在定义或修改表时为表列声明使用这个域。

当一个用户获得另一个用户的某个对象的访问权限后，可以以“模式名.对象名”的形式访问这个数据库对象。一个用户所拥有的对象和可以访问的对象是不同的，这一点在数据字典视图中有所反映。在默认情况下用户可以直接访问自己模式中的数据库对象，但是要访问其他用户所拥有的对象，就必须具有相应的对象权限。

对象权限的授予一般由对象的所有者完成，也可由 SYSDBA 或具有某对象权限且具有转授权限的用户授予，但最好由对象的所有者完成。

3.2 角色管理

角色是一组权限的组合，使用角色的目的是使权限管理更加方便。假设有 10 个用户，这些用户为了访问数据库，至少拥有 CREATE TABLE、CREATE VIEW 等权限。如果将这些权限分别授予这些用户，那么需要进行的授权次数是比较多的。但是如果把这些权限事先放在一起，然后作为一个整体授予这些用户，那么每个用户只需一次授权，授权的次数将大大减少，而且用户数越多，需要指定的权限越多，这种授权方式的优越性就越明显。这些事先组合在一起的一组权限就是角色，角色中的权限既可以是数据库权限，也可以是对象权限，还可以是别的角色。

为了使用角色，首先在数据库中创建一个角色，这时角色中没有任何权限。然后向角色中添加权限。最后将这个角色授予用户，这个用户就具有了角色中的所有权限。在使用角色

的过程中，可以随时向角色中添加权限，也可以随时从角色中删除权限，用户的权限也随之改变。如果要回收所有权限，只需将角色从用户回收即可。

3.2.1 角色的创建与删除

3.2.1.1DM 预定义角色

在 DM 数据库中有两类角色，一类是 DM 预设定的角色，一类是用户自定义的角色。DM 提供了一系列的预定义角色以帮助用户进行数据库权限的管理。预定义角色在数据库被创建之后即存在，并且已经包含了一些权限，数据库管理员可以将这些角色直接授予用户。

在"三权分立"和"四权分立"机制下，DM 的预定义角色及其所具有的权限是不相同的。表 3.3 和表 3.4 分别列出了"三权分立"和"四权分立"机制下常见的系统角色及其简单介绍，完整的预定义角色权限列表请参见附录 1 和附录 2。

表 3.3 "三权分立"常见的数据库预设定的角色

角色名称	角色简单说明
DBA	DM 数据库系统中对象与数据操作的最高权限集合，拥有构建数据库的全部特权，只有 DBA 才可以创建数据库结构
RESOURCE	可以创建数据库对象，对有权限的数据库对象进行数据操纵，不可以创建数据库结构
PUBLIC	不可以创建数据库对象，只能对有权限的数据库对象进行数据操纵
VTI	具有系统动态视图的查询权限，VTI 默认授权给 DBA 且可转授
SOI	具有系统表的查询权限
SVI	具有基础 V 视图的查询权限
DB_AUDIT_ADMIN	数据库审计的最高权限集合，可以对数据库进行各种审计操作，并创建新的审计用户
DB_AUDIT_OPER	可以对数据库进行各种审计操作，但不能创建新的审计用户
DB_AUDIT_PUBLIC	不能进行审计设置，但可以查询审计相关字典表
DB_AUDIT_VTI	具有系统动态视图的查询权限，DB_AUDIT_VTI 默认授权给 DB_AUDIT_ADMIN 且可转授
DB_AUDIT_SOI	具有系统表的查询权限
DB_AUDIT_SVI	具有基础 V 视图和审计 V 视图的查询权限

DB_POLICY_ADMIN	数据库强制访问控制的最高权限集合，可以对数据库进行强制访问控制管理，并创建新的安全管理用户
DB_POLICY_OPER	可以对数据库进行强制访问控制管理，但不能创建新的安全管理用户
DB_POLICY_PUBLIC	不能进行强制访问控制管理，但可以查询强制访问控制相关字典表
DB_POLICY_VTI	具有系统动态视图的查询权限，DB_POLICY_VTI 默认授权给 DB_POLICY_ADMIN 且可转授
DB_POLICY_SOI	具有系统表的查询权限
DB_POLICY_SVI	具有基础 V 视图和安全 V 视图的查询权限

表 3.4 “四权分立”常见的数据库预设定的角色

角色名称	角色简单说明
DBA	拥有构建数据库的全部特权，只有 DBA 才可以创建数据库结构
RESOURCE	可以创建和删除角色
PUBLIC	只能查询相关的数据字典表
VTI	具有系统动态视图的查询权限，VTI 默认授权给 DBA 且可转授
SOI	具有系统表的查询权限
SVI	具有基础 V 视图的查询权限
DB_AUDIT_ADMIN	数据库审计的最高权限集合，可以对数据库进行各种审计操作，并创建新的审计用户
DB_AUDIT_OPER	可以对数据库进行各种审计操作，但不能创建新的审计用户
DB_AUDIT_PUBLIC	不能进行审计设置，但可以查询审计相关字典表
DB_AUDIT_VTI	具有系统动态视图的查询权限，DB_AUDIT_VTI 默认授权给 DB_AUDIT_ADMIN 且可转授
DB_AUDIT_SOI	具有系统表的查询权限
DB_AUDIT_SVI	具有基础 V 视图和审计 V 视图的查询权限
DB_POLICY_ADMIN	数据库强制访问控制的最高权限集合，可以对数据库进行强制访问控制管理，并创建新的安全管理用户
DB_POLICY_OPER	可以对数据库进行强制访问控制管理，但不能创建新的安全管理用户
DB_POLICY_PUBLIC	不能进行强制访问控制管理，但可以查询强制访问控制相关字典表
DB_POLICY_VTI	具有系统动态视图的查询权限，DB_POLICY_VTI 默认授权给 DB_POLICY_ADMIN 且可转授
DB_POLICY_SOI	具有系统表的查询权限

DB_POLICY_SVI	具有基础 v 视图和安全 v 视图的查询权限
DB_OBJECT_ADMIN	可以在自己的模式下创建各种数据库对象并进行数据操纵，也可以创建和删除非模式对象
DB_OBJECT_OPER	可以在自己的模式下创建数据库对象并进行数据操纵
DB_OBJECT_PUBLIC	不可以创建数据库对象，只能对有权限的数据库对象进行数据操纵
DB_OBJECT_VTI	具有系统动态视图的查询权限，DB_OBJECT_VTI 默认授权给 DB_OBJECT_ADMIN 且可转授
DB_OBJECT_SOI	具有系统表的查询权限
DB_OBJECT_SVI	和 SVI 权限一样

我们可以认为“三权分立”安全机制将用户分为了三种类型，“四权分立”安全机制将用户分为了四种类型，每种类型又各对应六种预定义角色。如：

- DBA 类型对应 DBA、RESOURCE、PUBLIC、VTI、SOI、SVI 预定义角色；
- AUDITOR 对应 DB_ADUTI_ADMIN、DB_AUDIT_OPER、DB_AUDIT_PUBLIC、DB_AUDIT_VTI、DB_AUDIT_SOI、DB_AUDIT_SVI 预定义角色；
- SSO 对应 DB_POLICY_ADMIN、DB_POLICY_OPER、DB_POLICY_PUBLIC、DB_POLICY_VTI、DB_POLICY_SOI、DB_POLICY_SVI 预定义角色；
- DBO 对应 DB_OBJECT_ADMIN、DB_OBJECT_OPER、DB_OBJECT_PUBLIC、DB_OBJECT_VTI、DB_OBJECT_SOI、DB_OBJECT_SVI 预定义角色。

初始时仅有管理员具有创建用户的权限，每种类型的管理员创建的用户缺省就拥有这种类型的 PUBLIC 预定义角色，如 SYSAUDITOR 新创建的用户缺省就具有 DB_AUDIT_PUBLIC 角色。之后管理员可根据需要进一步授予新建用户其他预定义角色。

管理员也可以将“CREATE USER”权限转授给其他用户，这些用户之后就可以创建新的用户了，他们创建的新用户缺省也具有与其创建者相同类型的 PUBLIC 和 SOI 预定义角色。

■ 限制 DBA 的“ANY”权限

从附录 1 和附录 2 的表中我们可以看到，缺省情况下，DBA 角色拥有许多的“ANY”权限，如 DROP ANY TABLE、INSERT ANY TABLE 等等。DM 提供了一种限制 DBA 的“ANY”权限的方法：在非 DM 安全版本中，只有 SYSDBA 用户才能限制 DBA 的“ANY”权限；在 DM 安全版本中，只有具有 DB_POLICY_ADMIN 角色的用户才能限制 DBA 的“ANY”权限。通过执行下面的系统过程来实现，系统过程执行完成后需要重启服务器该限制才能生效。

```
VOID
SP_RESTRICT_DBA(
       FLAG          INT
);
```

参数 FLAG 置为 1 表示限制 DBA 的“ANY”权限，此过程执行后所有的 DBA 用户将不再具有“ANY”权限；FLAG 置为 0 表示不限制 DBA 的“ANY”权限。

3.2.1.2 创建角色

具有“CREATE ROLE”数据库权限的用户也可以创建新的角色，其语法如下：

```
CREATE ROLE  <角色名>;
```

使用说明：

1. 创建者必须具有 CREATE ROLE 数据库权限；
2. 角色名的长度不能超过 128 个字符；
3. 角色名不允许和系统已存在的用户名重名；
4. 角色名不允许是 DM 保留字。

例如，创建角色 BOOKSHOP_ROLE1，赋予其 PERSON.ADDRESS 表的 SELECT 权限。

```
CREATE ROLE BOOKSHOP_ROLE1;
GRANT SELECT ON PERSON.ADDRESS TO BOOKSHOP_ROLE1;
```

3.2.1.3 删除角色

具有“DROP ROLE”权限的用户可以删除角色，其语法如下：

```
DROP ROLE [IF EXISTS] <角色名>;
```

即使已将角色授予了其他用户，删除这个角色的操作也将成功。此时，那些之前被授予该角色的用户将不再具有这个角色所拥有的权限，除非用户通过其他途径也获得了这个角色所具有的权限。

指定 IF EXISTS 关键字后，删除不存在的角色时不会报错，否则会报错。

例如，接上一小节的例子。将角色 BOOKSHOP_ROLE1 授予用户 BOOKSHOP_USER，则用户 BOOKSHOP_USER 具有了 SELECT 表 PERSON.ADDRESS 的权限。此时删除角色 BOOKSHOP_ROLE1，那么用户 BOOKSHOP_USER 将不再能查询表 PERSON.ADDRESS。但是如果用户 BOOKSHOP_USER 从别的途径（直接授权或通过别的角色授权）重复获得了 SELECT 表 PERSON.ADDRESS 的权限，那么即使删除了角色 BOOKSHOP_ROLE1，用户 BOOKSHOP_USER 仍然具有 SELECT 表 PERSON.ADDRESS 的权限。

3.2.2 角色的启用与禁用

某些时候，用户不愿意删除一个角色，但是却希望这个角色失效，此时可以使用 DM 系统过程 SP_SET_ROLE 来设置这个角色为不可用，将第二参数置为 0 表示禁用角色。

例如，下面的语句将角色 BOOKSHOP_ROLE1 禁用。

```
SP_SET_ROLE('BOOKSHOP_ROLE1', 0);
```

使用说明：

1. 只有拥有 ADMIN_ANY_ROLE 权限的用户才能启用和禁用角色，并且设置后立即生效；
2. 凡是包含禁用角色 A 的角色 M，M 中禁用的角色 A 将无效，但是 M 仍有效；
3. 系统预设的角色是不能设置的，如：DBA、PUBLIC、RESOURCE。

当用户希望启用某个角色时，同样可以通过 SP_SET_ROLE 来启用角色，只要将第二个参数置为 1 即可。

例如，下面的语句将启用角色 BOOKSHOP_ROLE1。

```
SP_SET_ROLE('BOOKSHOP_ROLE1', 1);
```

3.2.3 SVI 角色用法

拥有 SVI 角色权限的用户可以查询 V 视图。V 视图请参考《DM8 系统管理员手册》。下面是使用 SVI 角色的相关系统过程。

1. SP_INIT_SVI_SYS

定义：

SP_INIT_SVI_SYS(

```
CREATE_FLAG            int
)
```

功能说明：

可以手动重建所有 V 视图，并授权给 SVI、DB_POLICY_SVI、DB_OBJECT_SVI 和 DB_AUDIT_SVI。只有 DBA 和 DB_OBJECT_ADMIN 可以执行。

参数详解

CREATE_FLAG：1 重建所有 V 视图；0 删除所有 V 视图

返回值

无

举例说明

```
SP_INIT_SVI_SYS(1);
SP_INIT_SVI_SYS(0);
```

2. SP_SWITCH_SVI

定义：

```
SP_SWITCH_SVI
(
flag            int
)
```

功能说明：

SVI 角色启用或禁用。只有 DBA/DB_OBJECT_ADMIN 可以执行。SVI 启用时字典信息来源于 V 视图，SVI 禁用时字典信息来源于 SYS.数据字典表。

SP_INIT_SVI_SYS(1) 需和 SP_SWITCH_SVI(1) 一起使用，SP_INIT_SVI_SYS(0)需和 SP_SWITCH_SVI(0)一起使用，才有效果。否则报错。

SP_SWITCH_SVI 会对 MANAGER 工具造成访问影响，所以建议调用后立即重启数据库。

参数说明：

flag：1 启用 SVI；0 禁用 SVI

返回值：

启用返回 1；禁用返回 0

举例说明：

```
SP_SWITCH_SVI(0);
SP_SWITCH_SVI(1);
```

3. SF_GET_SVI

定义：

int

SF_GET_SVI ()

功能说明：

获取当前系统是否启用 SVI 角色 。

参数说明：

无

返回值：

启用返回 1；禁用返回 0

举例说明：

```
SP_SWITCH_SVI(0);
SELECT SF_GET_SVI;    //结果为 0
SP_SWITCH_SVI(1);
SELECT SF_GET_SVI;    //结果为 1
```

3.3 权限的分配与回收

可以通过 GRANT 语句将权限（包括数据库权限、对象权限以及角色权限）分配给用户和角色，之后也可以使用 REVOKE 语句将授出的权限再进行回收。

3.3.1 数据库权限的分配与回收

3.3.1.1 数据库权限的分配

可以使用 GRANT 语句授予用户和角色数据库权限。

数据库权限的授权语句语法为：

```
GRANT <特权> TO <用户或角色>{,<用户或角色>} [WITH  ADMIN OPTION];
<特权> ::= <数据库权限>{,<数据库权限>};
<用户或角色>::= <用户名> | <角色名>
```

使用说明：

1. 授权者必须具有对应的数据库权限以及其转授权；
2. 接受者必须与授权者用户类型一致；
3. 如果有 WITH ADMIN OPTION 选项，接受者可以再把这些权限转授给其他用户/角色。

例如，系统管理员 SYSDBA 把建表和建视图的权限授给用户 BOOKSHOP_USER1，并允许其转授。

```
GRANT CREATE TABLE, CREATE VIEW TO BOOKSHOP_USER1 WITH ADMIN OPTION;
```

3.3.1.2 数据库权限的回收

可以使用 REVOKE 语句回收授出的指定数据库权限。

回收数据库权限的语句语法为：

```
REVOKE [ADMIN OPTION FOR]<特权> FROM <用户或角色>{,<用户或角色>} ;
<特权> ::= <数据库权限>{,<数据库权限>}
<用户或角色>::= <用户名> | <角色名>
```

使用说明：

1. 权限回收者必须是具有回收相应数据库权限以及转授权的用户；
2. ADMIN OPTION FOR 选项的意义是取消用户或角色的转授权限，但是权限不回收。

例 1 接 3.3.1.1 中的例子，现在 SYSDBA 把用户 BOOKSHOP_USER1 的建表权限收回。

```
REVOKE CREATE TABLE FROM BOOKSHOP_USER1;
```

例 2 接 3.3.1.1 中的例子，之前 SYSDBA 把建视图的权限授给了用户 BOOKSHOP_USER1，并且允许转授。现在 SYSDBA 不让用户 BOOKSHOP_USER1 转授 CREATE VIEW 权限。

```
REVOKE ADMIN OPTION FOR CREATE VIEW FROM BOOKSHOP_USER1;
```

BOOKSHOP_USER1 仍有 CREATE VIEW 权限，但是不能将 CREATE VIEW 权限转授给其他用户。

3.3.1.3 限制相关数据库权限的授予与回收

通过 INI 参数 ENABLE_DDL_ANY_PRIV 限制 DDL 相关的 ANY 数据库权限的授予与回收，有 2 个取值：

- 1：可以授予和回收 DDL 相关的 ANY 系统权限；
- 0：不可以授予和回收 DDL 相关的 ANY 系统权限。缺省为 0。

例 1 当 ENABLE_DDL_ANY_PRIV=0 时，禁止授予或回收 create any trigger 的权限。

```
CONN SYSDBA/SYSDBA

create user DBSEC  identified by 123456789;

GRANT create any trigger TO DBSEC;     //报错

revoke create any trigger from DBSEC;//报错
```

3.3.2 对象权限的分配与回收

3.3.2.1 对象权限的分配

可以使用 GRANT 语句将对象权限授予用户和角色。

对象权限的授权语句语法为：

```
GRANT <特权> ON [<对象类型>] <对象> TO <用户或角色>{,<用户或角色>} [WITH GRANT
OPTION];

<特权>::= ALL [PRIVILEGES] | <动作> {, <动作>}

<动作>::= SELECT[(<列清单>)] |

          INSERT[(<列清单>)] |

          UPDATE[(<列清单>)] |

          DELETE |
```

```
        REFERENCES[(<列清单>)] |
        EXECUTE|
        READ|
        WRITE|
        USAGE|
        INDEX|
        ALTER
<列清单>::= <列名> {,<列名>}
<对象类型>::= TABLE | VIEW | PROCEDURE | PACKAGE | CLASS | TYPE | SEQUENCE |
DIRECTORY | DOMAIN
<对象> ::= [<模式名>.]<对象名>
<对象名> ::= <表名> | <视图名> | <存储过程/函数名> |<包名> |<类名> |<类型名> |<序列名>
| <目录名> | <域名>
<用户或角色>::= <用户名> | <角色名>
```

使用说明：

1. 授权者必须是具有对应对象权限以及其转授权的用户；
2. 如未指定对象的<模式名>，模式为授权者所在的模式。DIRECTORY 为非模式对象，没有模式；
3. 如设定了对象类型，则该类型必须与对象的实际类型一致，否则会报错；
4. 带 WITH GRANT OPTION 授予权限给用户时，则接受权限的用户可转授此权限；
5. 不带列清单授权时，如果对象上存在同类型的列权限，会全部自动合并；
6. 对于用户所在的模式的表，用户具有所有权限而不需特别指定；
7. INDEX 动作向其他用户授予指定表的创建和删除索引（包含全文索引）的权限；
8. ALTER 动作仅支持向其他用户授予指定表的修改权限。

当授权语句中使用了 ALL PRIVILEGES 时，会将指定的数据库对象上所有的对象权限都授予被授权者。不同类型的数据库对象相关的对象权限见 3.1.2 节的介绍。

例 1 SYSDBA 把 PERSON.ADDRESS 表的全部权限授给用户 BOOKSHOP_USER1。

```
GRANT SELECT, INSERT, DELETE, UPDATE, REFERENCES ON PERSON.ADDRESS TO
BOOKSHOP_USER1;
```

该语句也可写为以下形式：

```
GRANT ALL PRIVILEGES ON PERSON.ADDRESS TO BOOKSHOP_USER1;
```

例 2 假设用户 BOOKSHOP_USER1 创建了存储过程 BOOKSHOP_USER1_PROC1，数据库管理员 SYSDBA 把该存储过程的执行权 EXECUTE 授给已存在用户 BOOKSHOP_USER2，并使其具有该权限的转授权。

```
GRANT EXECUTE ON PROCEDURE BOOKSHOP_USER1.BOOKSHOP_USER1_PROC1
TO BOOKSHOP_USER2 WITH GRANT OPTION;
```

例 3 假设 SYSDBA 是表 BOOKSHOP_T1 的创建者，用户 BOOKSHOP_USER1、BOOKSHOP_USER2、BOOKSHOP_USER3 存在，且都不是 DBA 权限用户。

```
(1)以 SYSDBA 身份登录，并执行语句：
GRANT SELECT ON BOOKSHOP_T1 TO BOOKSHOP_USER1 WITH GRANT OPTION;
//正确
(2)以 BOOKSHOP_USER1 身份登录，并执行语句：
GRANT SELECT ON SYSDBA.BOOKSHOP_T1 TO BOOKSHOP_USER2;
//正确
(3)以 BOOKSHOP_USER2 身份登录，并执行语句：
GRANT SELECT ON SYSDBA.BOOKSHOP_T1 TO BOOKSHOP_USER3;
//错误，用户 BOOKSHOP_USER2 没有 SELECT 权限的转授权
```

例 4 用户 SYSDBA 创建一个名为 VIEW_PRODUCT 的视图，要求对于该视图，允许 BOOKSHOP_USER1 能够进行查询、插入、删除和更新操作，用户 BOOKSHOP_USER2 和 BOOKSHOP_USER3 也可进行同样的工作，但要求其操作权限由用户 BOOKSHOP_USER1 控制。

```
(1)以 SYSDBA 的身份登录：
CREATE VIEW VIEW_PRODUCT AS
SELECT * FROM PRODUCTION.PRODUCT
WHERE NOWPRICE>=20 AND ORIGINALPRICE<40 WITH CHECK OPTION;


GRANT SELECT, INSERT, DELETE, UPDATE ON VIEW_PRODUCT
```

```
TO BOOKSHOP_USER1 WITH GRANT OPTION;


(2)以用户 BOOKSHOP_USER1 的身份登录：
GRANT SELECT, INSERT, DELETE, UPDATE ON SYSDBA.VIEW_PRODUCT TO BOOKSHOP_USER2,
BOOKSHOP_USER3;
```

例 5 假定表的创建者 SYSDBA 把所创建的 PRODUCTION.PRODUCT 表的部分列的更新权限授予用户 BOOKSHOP_USER1，BOOKSHOP_USER1 再将此更新权限转授给用户 BOOKSHOP_USER2。

```
(1)以 SYSDBA 的身份登录，并执行语句：
GRANT UPDATE (ORIGINALPRICE, NOWPRICE) ON PRODUCTION.PRODUCT TO
BOOKSHOP_USER1 WITH GRANT OPTION;
(2)以 BOOKSHOP_USER1 的身份登录，并执行语句，把列级更新权限授给用户 BOOKSHOP_USER2：
GRANT UPDATE(ORIGINALPRICE, NOWPRICE)ON PRODUCTION.PRODUCT TO BOOKSHOP_USER2;
```

3.3.2.2 对象权限的回收

可以使用 REVOKE 语句回收授出的指定数据库对象的指定权限。

对象权限的回收语句语法为：

```
REVOKE [GRANT OPTION FOR] <特权> ON [<对象类型>]<对象> FROM <用户或角色> {,<用户或
角色>} [<回收选项>];
<特权>::= ALL [PRIVILEGES] | <动作> {, <动作>}
<动作>::= SELECT |
          INSERT |
          UPDATE |
          DELETE |
          REFERENCES |
          EXECUTE|
          READ|
          WRITE|
          USAGE|
```

```
        INDEX|
        ALTER
<对象类型>::= TABLE | VIEW | PROCEDURE | PACKAGE | CLASS | TYPE | SEQUENCE |
DIRECTORY | DOMAIN
<对象> ::= [<模式名>.]<对象名>
<对象名> ::= <表名> | <视图名> | <存储过程/函数名> |<包名> |<类名> |<类型名> | <序列名>
| <目录名> | <域名>
<用户或角色>::= <用户名> | <角色名>
<回收选项> ::= RESTRICT | CASCADE
```

使用说明：

1. 权限回收者必须是具有回收相应对象权限以及转授权的用户；
2. 回收时不能带列清单，若对象上存在同类型的列权限，则一并被回收；
3. 使用 GRANT OPTION FOR 选项的目的是收回用户或角色权限转授的权利，而不回收用户或角色的权限；并且 GRANT OPTION FOR 选项不能和 RESTRICT 一起使用，否则会报错；
4. 在回收权限时，设定不同的回收选项，其意义不同。具体如下：
   - 若不设定回收选项，无法回收授予时带 WITH GRANT OPTION 的权限，但也不会检查要回收的权限是否存在限制；
   - 若设定为 RESTRICT，无法回收授予时带 WITH GRANT OPTION 的权限，也无法回收存在限制的权限，如角色上的某权限被别的用户用于创建视图等；
   - 若设定为 CASCADE，可回收授予时带或不带 WITH GRANT OPTION 的权限，若带 WITH GRANT OPTION 还会引起级联回收。利用此选项时也不会检查权限是否存在限制。另外，利用此选项进行级联回收时，若被回收对象上存在另一条路径授予同样权限给该对象时，则仅需回收当前权限。

说明： **用户 A 给用户 B 授权且允许其转授，B 将权限转授给 C。当 A 回收 B 的权限的时候必须加 CASCADE 回收选项。**

例 1 接 3.3.2.1 节例 1，SYSDBA 从用户 BOOKSHOP_USER1 处回收其授出的 PERSON.ADDRESS 表的全部权限。

```
REVOKE ALL PRIVILEGES ON BOOKSHOP_T1 FROM BOOKSHOP_USER1 CASCADE;
```

例 2 接 3.3.2.1 节例 2，SYSDBA 从用户 BOOKSHOP_USER2 处回收其授出的存储过程 BOOKSHOP_USER1_PROC1 的 EXECUTE 权限。

```
REVOKE EXECUTE ON PROCEDURE BOOKSHOP_USER1.BOOKSHOP_USER1_PROC1
FROM BOOKSHOP_USER2 CASCADE;
```

例 3 假定系统中存在非 SYSDBA 用户 BOOKSHOP_USER1、BOOKSHOP_USER2、BOOKSHOP_USER3 和 BOOKSHOP_USER4，其中 BOOKSHOP_USER1 具有 CREATE TABLE 数据库权限，BOOKSHOP_USER2 具有 CREATE VIEW 数据库权限，且已成功执行了如下的语句：

```
BOOKSHOP_USER1 登录执行:
CREATE TABLE T1(ID INT, NAME VARCHAR(100));
GRANT SELECT ON T1 TO PUBLIC;
GRANT INSERT(ID) ON T1 TO BOOKSHOP_USER2 WITH GRANT OPTION;
BOOKSHOP_USER2 登录执行:
CREATE VIEW V1 AS SELECT NAME FROM BOOKSHOP_USER1.T1 WHERE ID < 10;
GRANT INSERT(ID) ON BOOKSHOP_USER1.T1 TO BOOKSHOP_USER3 WITH GRANT OPTION;
BOOKSHOP_USER3 登录执行:
GRANT INSERT(ID) ON BOOKSHOP_USER1.T1 TO BOOKSHOP_USER4 WITH GRANT OPTION;
BOOKSHOP_USER4 登录执行:
GRANT INSERT(ID) ON BOOKSHOP_USER1.T1 TO BOOKSHOP_USER2 WITH GRANT OPTION;
```

此时，若 BOOKSHOP_USER1 执行以下语句会引起之前授予的 INSERT(ID)权限被合并。

```
GRANT INSERT ON T1 TO BOOKSHOP_USER2 WITH GRANT OPTION;
```

若 BOOKSHOP_USER1 执行以下语句，会成功，因为这种方式不检查权限是否存在限制。

```
REVOKE SELECT ON T1 FROM PUBLIC;
```

若 BOOKSHOP_USER1 执行以下语句，则会失败，因为这种方式检查权限是否存在限制，即 BOOKSHOP_USER2 利用此权限创建了视图 V1。

```
REVOKE SELECT ON T1 FROM PUBLIC RESTRICT;
```

若 BOOKSHOP_USER1 执行以下语句，会进行级联回收 BOOKSHOP_USER2、BOOKSHOP_USER3、BOOKSHOP_USER4 授出的 INSERT(ID)权限。

```
REVOKE INSERT ON T1 FROM BOOKSHOP_USER2 CASCADE;
```

但若 BOOKSHOP_USER1 连续执行了以下两条语句，则后一条语句仅回收其授予 BOOKSHOP_USER2 的权限，而不会产生级联回收，因为有另一条路经授予了 BOOKSHOP_USER3 同样的权限 INSERT(ID)。

```
GRANT INSERT ON T1 TO BOOKSHOP_USER3 WITH GRANT OPTION;
REVOKE INSERT ON T1 FROM BOOKSHOP_USER2 CASCADE;
```

3.3.3 角色权限的分配与回收

3.3.3.1 角色权限的分配

通常角色包含权限或其他角色，通过使用 GRANT 语句将一个角色授予用户或另一角色可以使得用户和角色继承该角色所具有的权限。

授予角色权限的语句语法如下：

```
GRANT <角色名>{, <角色名>} TO <用户或角色>{,<用户或角色>} [WITH  ADMIN OPTION];
<用户名或角色名> ::= <用户名> | <角色名>
```

使用说明：

1. 角色的授予者必须为拥有相应的角色以及其转授权的用户；
2. 接受者必须与授权者类型一致(譬如不能把审计角色授予标记角色)；
3. 支持角色的转授；
4. 不支持角色的循环转授，如将 BOOKSHOP_ROLE1 授予 BOOKSHOP_ROLE2，BOOKSHOP_ROLE2 不能再授予 BOOKSHOP_ROLE1。

例如，假设存在角色 BOOKSHOP_ROLE1，角色 BOOKSHOP_ROLE2，用户 BOOKSHOP_USER1。

```
//让用户 BOOKSHOP_USER1 继承角色 BOOKSHOP_ROLE1 的权限
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_USER1;
//让角色 BOOKSHOP_ROLE2 继承角色 BOOKSHOP_ROLE1 的权限
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_ROLE2;
```

3.3.3.2 角色权限的回收

可以使用 REVOKE 语句回收用户或其它角色从指定角色继承过来的权限。

回收角色权限的语句语法为：

```
REVOKE  [ADMIN OPTION FOR] <角色名>{,<角色名>} FROM  <角色名或用户名>;
<角色名或用户名> ::= <用户名> | <角色名>
```

使用说明：

1. 权限回收者必须是具有回收相应角色以及转授权的用户；
2. 使用 ADMIN OPTION FOR 选项的目的是收回用户或角色权限转授的权利，而不回收用户或角色的权限。

例如，接 3.3.3.1 节的例子，回收用户 BOOKSHOP_USER1 和角色 BOOKSHOP_ROLE2 的 BOOKSHOP_ROLE1 角色。

```
REVOKE BOOKSHOP_ROLE1 FROM BOOKSHOP_USER1;
REVOKE BOOKSHOP_ROLE1 FROM BOOKSHOP_ROLE2;
```

4 强制访问控制

强制访问控制(Mandatory Access Control, MAC)是根据客体的敏感标记和主体的访问标记对客体访问实行限制的一种方法。在强制访问控制中，系统给主体和客体都分配一个特殊的安全标记，主体的安全标记反映了该主体可信的程度，客体的安全标记则与其包含信息的敏感度一致，且主体不能改变他自己及任何其它客体的安全标记，主体是否可以对客体执行特定的操作取决于主体和客体的安全标记之间的支配关系。因此，强制访问控制可以控制系统中信息流动的轨迹，能有效地抵抗特洛伊木马的攻击，这在一些对安全要求很高的数据库应用中是非常必要的。

DM 利用策略和标记来实现 DM 数据库的强制访问控制。执行强制访问控制的用户必须具有 LABEL_DATABASE 数据库权限，在新初始化的数据库中，只有 SYSSSO 具有这个权限。

强制访问控制功能仅在 DM 安全版中提供。DM 共享存储集群 DMDSC、DM 数据守护和读写分离集群都支持强制访问控制，但 DM 大规模并行处理集群 MPP 暂不支持强制访问控制。

4.1 如何创建策略

4.1.1 策略的组成

策略是一组预定义的标记组件，包括等级、范围和组这三种组件，三种组件分别从不同的维度对数据进行了描述。一个策略最多只能包含这三种组件，其中必须包含等级，范围和组可以缺省。

- **等级（Level）**

等级是线性有序的名称序列，用 L=（l1，l2，…，lp）表示。其中 li（1≤i≤p）表示第 i 个名称，任意两个名称 li、lj 之间，若 i≤j，则 li≤lj，于是有 l1≤l2≤…≤lp，其中 l1，l2，…，lp 称为等级分类（以下简称等级）。

在 DM 中，一个策略最大可定义 10000 个等级。用户在定义策略中的等级时，需要为其指定编号，其编号在 0－9999 之间（编号小的意味着级别较低）。

- **范围（Compartment）**

范围是集合类型，设集合 C={c1，c2，…，cm}中每一元素都是一名称，c1，c2，…，cm 间彼此独立，无序，则集合 C 及其任意子集称为非等级类别集合，其中 c1，c2，…，cm 称为非等级类别（以下简称为范围）。

在 DM 中，最大可定义 10000 个范围，需要用户设置编号，且编号在一个策略里面须是唯一的，编号之间没有级别高低之分。

- **组（Group）**

组为树形结构，有父子之分，可以用来描述组织结构。

设树 G={g1，g2，…，gm}，其中每一元素都是一名称，g1，g2，…，gm 间有父子之分，则 g1，g2，…，gm 称为组。

在 DM 中，最多能定义 10000 个组，也就是说组织结构的层次最多为 10000。组中只能有一个根组，除根组外，每个组有且仅有一个父组。

4.1.2 创建、修改与删除策略

4.1.2.1 创建策略

使用如下系统过程创建一个策略。

```
VOID
MAC_CREATE_POLICY(
       POLICY_NAME          VARCHAR(128)
);
```

参数说明：

POLICY_NAME　新创建的策略名称

使用说明：

只有具有 LABEL_DATABASE 数据库权限的用户才能执行此操作。

例如，创建策略 P_01

```
MAC_CREATE_POLICY('P_01');
```

4.1.2.2 修改策略

可以使用下面的系统过程修改一个策略的策略名。

```
VOID
MAC_ALTER_POLICY(
       POLICY_NAME          VARCHAR(128),
       NEW_NAME                    VARCHAR(128)
);
```

参数说明：

POLICY_NAME　待修改的策略名称

NEW_NAME　将修改成的策略名称

使用说明：

只有具有 LABEL_DATABASE 数据库权限的用户才能执行此操作。

例如，将策略 P_01 改名为 P_02。

```
MAC_ALTER_POLICY('P_01', 'P_02');
```

4.1.2.3 删除策略

可以使用下面的系统过程删除一个已存在的策略。

```
VOID
MAC_DROP_POLICY(
       POLICY_NAME          VARCHAR(128),
       DROP_COLUMN          INT
);
```

参数说明：

POLICY_NAME　待删除的策略名称

DROP_COLUMN　对应用此策略的表的标记列的处理方式，取值为 0 或 1

0：不删除应用此策略的所有表对应的标记列

1：删除应用此策略的所有表对应的标记列

若 DROP_COLUMN 为 NULL，则按默认为 0 进行处理

使用说明：

1. 只有具有 LABEL_DATABASE 数据库权限的用户才能执行此操作；

2. 指定待删除策略必须存在。

例如，删除策略 P_02。

```
MAC_DROP_POLICY('P_02');
```

4.1.3 为策略添加组件

创建一个策略后，需要对策略添加组件，这样策略才可以被应用到表和用户上。

4.1.3.1 为策略添加等级

使用下面的系统过程为指定的策略添加等级。

```
VOID
MAC_CREATE_LEVEL(
       POLICY_NAME          VARCHAR(128),
       LEVEL_NUM            INT,
       LEVEL_NAME           VARCHAR(128)
);
```

参数说明：

POLICY_NAME　　要添加等级的策略名

LEVEL_NUM　　创建的等级编号，在 0－9999 之间的整数

LEVEL_NAME　　创建的等级名称

使用说明：

1. LEVEL_NAME 不能包含“:”和“,”；

2. 该过程只能由具有 LABEL_DATABASE 的用户调用；

3. 指定策略必须存在；

4. 同一个策略中，等级 ID 和等级名称唯一；

5． 一个策略最多可以定义 10000 个等级；

6． 每个等级都要有一个等级 ID，ID 越小表示安全等级越低。

例如，创建策略 P_03，并给策略 P_03 添加等级 L_01，等级编号为 10。

```
MAC_CREATE_POLICY('P_03');

MAC_CREATE_LEVEL('P_03', 10, 'L_01');
```

在为策略添加等级后，还可以通过系统过程修改等级名称和删除等级。

- **修改等级**

修改等级名称的系统过程如下：

```
VOID

MAC_ALTER_LEVEL(

       POLICY_NAME          VARCHAR(128),

       LEVEL_NAME           VARCHAR(128),

       NEW_NAME             VARCHAR(128)

);
```

参数说明：

POLICY_NAME　要修改等级名的策略名

LEVEL_NAME　待修改的等级名称

NEW_NAME　要修改成的等级名称

使用说明：

1． NEW_LEVEL 不能包含“:”和“,”；

2． 该过程只能由具有 LABEL_DATABASE 的用户调用；

3． 待修改等级名必须存在。

例如，将策略 P_03 中的等级 L_01，更名为 L_02。

```
MAC_ALTER_LEVEL('P_03', 'L_01', 'L_02');
```

- **删除等级**

删除等级的系统过程如下：

```
VOID
MAC_DROP_LEVEL(
       POLICY_NAME          VARCHAR(128),
       LEVEL_NAME                  VARCHAR(128)
);
```

参数说明：

POLICY_NAME　待删除等级所在策略名

LEVEL_NAME　待删除的等级名称

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定等级必须存在；
3. 如果待删除等级被某个标记使用，则拒绝删除。

例如，删除策略 P_03 中的等级 L_02。

```
MAC_DROP_LEVEL('P_03', 'L_02');
```

4.1.3.2 为策略添加范围

使用下面的系统过程为指定的策略添加范围。

```
VOID
MAC_CREATE_COMPARTMENT(
       POLICY_NAME          VARCHAR(128),
       COMPART_NUM          INT,
       COMPART_NAME         VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要添加范围的策略名

COMPART_NUM　创建的范围编号，在 0－9999 之间的整数

COMPART_NAME　创建的范围名称

使用说明：

1. COMPART_NAME 不能包含“:”和“,”；
2. 该过程只能由具有 LABEL_DATABASE 的用户调用；
3. 指定策略必须存在；
4. 同一个策略中，范围 ID 和范围名称唯一；
5. 一个策略中最多可以定义 10000 个范围；
6. 范围独立无序，范围之间是平等关系，没有等级高低之分，范围之间的比较运算采用集合间的包含关系。

例如，给策略 P_03 添加范围 C_01。

```
MAC_CREATE_COMPARTMENT('P_03',10, 'C_01');
```

在为策略添加范围后，还可以通过系统过程修改范围名称和删除范围。

- **修改范围**

修改范围名称的系统过程如下：

```
VOID
MAC_ALTER_COMPARTMENT(
       POLICY_NAME          VARCHAR(128),
       COMPART_NAME         VARCHAR(128),
       NEW_NAME             VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要修改范围名的策略名

COMPART_NAME　待修改的范围名称

NEW_NAME　要修改成的范围名称

使用说明：

1. NEW_LEVEL 不能包含“:”和“,”；
2. 该过程只能由具有 LABEL_DATABASE 的用户调用；
3. 待修改范围名必须存在。

例如，将策略 P_03 中的范围 C_01，更名为 C_02。

```
MAC_ALTER_COMPARTMENT('P_03', 'C_01', 'C_02');
```

- **删除范围**

删除范围的系统过程如下：

```
VOID
MAC_DROP_COMPARTMENT(
       POLICY_NAME          VARCHAR(128),
       COMPART_NAME         VARCHAR(128)
);
```

参数说明：

POLICY_NAME　待删除范围所在策略名

COMPART_NAME　待删除的范围名称

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定范围必须存在；
3. 如果待删除范围被某个标记使用，则拒绝删除。

例如，删除策略 P_03 中的范围 C_02。

```
MAC_DROP_COMPARTMENT('P_03', 'C_02');
```

4.1.3.3 为策略添加组

使用下面的系统过程为指定的策略添加组。

```
VOID
MAC_CREATE_GROUP(
       POLICY_NAME          VARCHAR(128),
       GROUP_NUM            INT,
       GROUP_NAME           VARCHAR(128),
```

```
       PARENT_NAME          VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要添加组的策略名

GROUP_NUM　创建的组编号，在 0－9999 之间的整数

GROUP_NAME　创建的组名称

PARENT_NAME　新创建组的父组的名称

使用说明：

1． GROUP_NAME 不能包含“:”和“,”；

2． 该过程只能由具有 LABEL_DATABASE 的用户调用；

3． 指定策略必须存在；

4． 同一个策略中，组 ID 和组名称唯一；

5． 一个策略最多可以定义 10000 个组；

6． 同一个策略中，只能有一个根组，如果 PARENT_NAME 为 NULL，则创建根组；

7． 组之间的比较运算采用树形结构间的从属关系。

例如，给策略 P_03 创建根组 G_01；再以 G_01 为父组，创建组 G_02；以 G_02 为父组，创建组 G_03。

```
MAC_CREATE_GROUP ('P_03',10, 'G_01',NULL);
MAC_CREATE_GROUP ('P_03',20, 'G_02', 'G_01');
MAC_CREATE_GROUP ('P_03',30, 'G_03', 'G_02');
```

在为策略添加组后，还可以通过系统过程修改组名称、更新父组和删除组。

■ **修改组**

修改组名称的系统过程如下：

```
VOID
MAC_ALTER_GROUP(
       POLICY_NAME          VARCHAR(128),
       GROUP_NAME           VARCHAR(128),
```

```
        NEW_NAME            VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要修改组名的策略名

GROUP_NAME　待修改的组名称

NEW_NAME　要修改成的组名称

使用说明：

1. NEW_NAME 不能包含“:”和“,”；

2. 该过程只能由具有 LABEL_DATABASE 的用户调用；

3. 待修改组名必须存在。

例如，将策略 P_03 中的组 G_03，更名为 G_04。

```
MAC_ALTER_GROUP('P_03', 'G_03', 'G_04');
```

■ **更新父组**

更新父组的系统过程如下：

```
VOID
MAC_ALTER_GROUP_PARENT(
        POLICY_NAME         VARCHAR(128),
        GROUP_NAME          VARCHAR(128),
        PARENT_NAME         VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要更新父组的组所在的策略名

GROUP_NAME　待更新父组的组名称

PARENT_NAME　待修改成的父组名称

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；

2. 待修改组和待修改成的父组必须存在；

3． 父组不能是自身，同时不能是自己的子节点。

例如，将策略 P_03 中的组 G_04，更换父组为 G_01。

```
MAC_ALTER_GROUP_PARENT ('P_03', 'G_04', 'G_01');
```

- **删除组**

删除组的系统过程如下：

```
VOID
MAC_DROP_GROUP (
       POLICY_NAME          VARCHAR(128),
       GROUP_NAME           VARCHAR(128),
);
```

参数说明：

POLICY_NAME　待删除组所在策略名

GROUP_NAME　待删除的组名称

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定组必须存在；

3． 待删除的组不能有子节点存在，否则删除失败。

例如，删除策略 P_03 中的组 G_04。

```
MAC_DROP_GROUP('P_03', 'G_04');
```

4.2 如何创建标记

当把策略应用于用户或表时，那么该用户或表就获得了一个安全标记。一个安全标记由多个组件组成，其组件包括等级、范围和组。每个标记必须包含一个等级，范围和组则是可选的。

在 DM 中，标记用字符串表示，其最大长度为 4000。格式如下：

```
<等级>:[<范围>{,<范围>}]:[<组>{,<组>}];
```

假定系统中存在策略 P1，其包含等级 L1,L2,L3,范围 C1,C2,C3,C4,组 G1,G2,G3。则下列标记为合法标记：

```
L1::

L1:C1:

L1:C1,C2:

L1::G1

L1:C1:G1,G2
```

而下列标记为不合法标记：

```
:

:C1,C2

C1,C2

:C1:G1

::G1,G2

G1,G2
```

标记以二进制位的形式存储于数据库中。当一个策略被应用到表上时，需要指定标记的存储列名，此列的类型为 INT，此时表上每条元组的标记均存在于该列中。

在应用策略时，还可设置标记列是否被隐藏。当标记列被隐藏时，若用户在插入数据时不指定列清单，则在值列表中可以不设置该列的值，另外，在查询时也不显示该列数据。

标记列上可以建索引，设置列约束、改列名等，但不能修改列类型、设置缺省值等。标记列不能被用户显式删除，除非在表上取消该策略。

4.2.1 创建标记

可使用下面的系统过程创建一个标记。

```
VOID
MAC_CREATE_LABEL(
       POLICY_NAME          VARCHAR(128),
       LABEL_TAG            INT,
       LABEL_VALUE          VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　　要创建的标记所在的策略名

LABEL_TAG　　标记值，有效范围为 0~999999999

LABEL_VALUE　　标记串

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定策略必须存在；
3. 根据标记值和标记串，生成一个标记，若此标记已经存在，则返回错误。标记值由用户自己指定，且标记值在系统中应是唯一的，不同的策略中也不能使用相同的标记值。

例如，创建标记 L_01:C_01:G_02。

```
MAC_CREATE_POLICY('P_04');
MAC_CREATE_LEVEL ('P_04',100, 'L_01');
MAC_CREATE_COMPARTMENT ('P_04',100, 'C_01');
MAC_CREATE_GROUP ('P_04',100, 'G_01',NULL);
MAC_CREATE_GROUP ('P_04',200, 'G_02', 'G_01');
MAC_CREATE_LABEL('P_04',11, 'L_01:C_01:G_02');
```

上述例子执行后，将在 SYSMACLABELS 系统表中记录 TAG 值以及标记字符串'L1:C1:G1'的内部格式，这个格式是对字符串'L1:C1:G1'进行处理后的格式。

如果要查看系统中某个标记对应的具体的值，例如，要查看刚刚创建的 ID 为 11 的标记的具体值，可以使用如下系统函数：

```
SF_MAC_LABEL_TO_CHAR(11);
```

此系统函数的参数为标记的标记值，返回该标记对应的标记串。

4.2.2 修改标记

可根据标记的标记值或标记串对标记修改其标记串，使用如下的系统过程：

```
VOID
MAC_ALTER_LABEL(
       POLICY_NAME          VARCHAR(128),
```

```
       LABEL_TAG           INT,
       NEW_LABEL_VALUE     VARCHAR(4000)
);
VOID
MAC_ALTER_LABEL(
       POLICY_NAME         VARCHAR(128),
       LABEL_VALUE         VARCHAR(4000),
       NEW_LABEL_VALUE     VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　待修改标记所在策略名

LABEL_TAG　待修改标记的标记值

LABEL_VALUE　待修改标记的标记串

NEW_LABEL_VALUE　要修改为的标记串

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定策略必须存在；
3. 新的标记串必须是不存在的标记串，如果存在则报错；
4. 修改标记的用处主要在于可以不用更新表中的标记列，而直接修改标记串，来改变原始数据的安全级别；
5. 修改标记的标记串，如果新的标记串为 NULL，则保持原始值；
6. 如果修改的标记同时应用在用户上，那么用户的标记合法性可能会遭到破坏。

例如，更改标记 L_01:C_01:G_02。

```
MAC_ALTER_LABEL('P_04',11, 'L_01:C_01:G_01,G_02');
MAC_ALTER_LABEL('P_04', 'L_01:C_01:G_01,G_02', 'L_01:C_01:');
```

4.2.3 删除标记

可根据标记的标记值或标记串删除标记，使用如下的系统过程：

```
VOID
MAC_DROP_LABEL(
       POLICY_NAME          VARCHAR(128),
       LABEL_TAG            INT
);
VOID
MAC_DROP_LABEL(
       POLICY_NAME          VARCHAR(128),
       LABEL_VALUE          VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　待删除标记所在策略名

LABEL_TAG　待删除标记的标记值

LABEL_VALUE　待删除标记的标记串

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定策略必须存在；
3. 待删除的标记必须是已存在的标记，否则报错；
4. 标记可以进行删除，即使标记被应用在表或用户上，这样会导致表或用户的标记失效，使用时需注意。

例如，创建标记之后删除。

```
MAC_CREATE_LABEL('P_04',12, 'L_01::');
MAC_DROP_LABEL('P_04',12);
MAC_CREATE_LABEL('P_04',13, 'L_01::G_01');
MAC_DROP_LABEL('P_04', 'L_01::G_01');
```

4.2.4 隐式创建标记

可以使用下面的系统过程隐式创建标记。

```
VOID
SP_MAC_LABEL_FROM_CHAR(
       POLICY_NAME          VARCHAR(128),
       LABEL_VALUE          VARCHAR(4000),
       TAG                  INT
);
```

参数说明：

POLICY_NAME　标记所在的策略名

LABEL_VALUE　标记串

标记值　　输出参数，标记串对应的标记值

使用说明：

1． 指定策略必须存在；

这个系统过程首先会去系统表 SYSMACLABELS 查找是否具有指定标记串的标记，如果存在，则返回相应的标记值，如果不存在，系统会自动创建一个对应的标记，并返回给用户相应的标记值，标记值由系统自动生成。

DM 的 INI 参数 MAC_LABEL_OPTION 用于控制 SP_MAC_LABEL_FROM_CHAR 过程的使用范围，有三种取值：

- 0：只有 SSO 可以调用
- 1：所有用户都可以调用
- 2：所有用户可以调用，但是非 SSO 用户不会主动创建新的 LABEL

例 1 隐式创建标记，标记串不存在的情况。

```
DECLARE
TAG INT;
BEGIN
SP_MAC_LABEL_FROM_CHAR ('P_04', 'L_01::G_02', TAG);
PRINT TAG ;
END;
/
```

由于之前系统中没有指定标记串的标记，此时会为这个标记串创建一个标记，返回系统自动生成的 TAG 值。

例 2 隐式创建标记，标记值已存在的情况。

```
MAC_CREATE_LABEL('P_04',21, 'L_01:C_01:G_02');


DECLARE
TAG INT;
BEGIN
SP_MAC_LABEL_FROM_CHAR ('P_04', 'L_01:C_01:G_02', TAG);
PRINT TAG ;
END;
/
```

由于先创建了标记串为 L_01:C_01:G_02 的标记，指定标记值为 21，因此后面的 SP_MAC_LABEL_FROM_CHAR 过程调用不会创建新的标记，直接返回标记值 21。

4.3 如何对表应用策略

4.3.1 对表应用策略

将策略应用在一个表上时，就使该表处于一定的等级、范围和组内。一个表上可以应用多个策略，但一个策略对表只能应用一次。

对表应用策略时，会在表上新建一个标记列，用于记录标记，新建的标记列名必须和表中已有列名不同，同时还可以指定初始的标记值，以及是否将标记列设置为隐藏列。

使用下述系统过程对表应用策略：

```
VOID
MAC_APPLY_TABLE_POLICY（
       POLICY_NAME          VARCHAR(128),
       SCHEMANAME           VARCHAR(128),
       TABLENAME            VARCHAR(128),
       COLNAME              VARCHAR(128),
```

```
    LABELVALUE          VARCHAR(4000),
    OPTION              INT
);
```

参数说明：

POLICY_NAME	应用于指定表的策略名
SCHEMANAME	表所属模式名称
TABLENAME	策略所应用的表名称
COLNAME	用于记录标记的列名称
LABELVALUE	用于说明被应用了策略的表中，已有元组的等级、范围和组
OPTION	1 代表隐藏标记列，0 代表不隐藏标记列，缺省为 0

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 指定策略必须存在；
3. 策略不能应用在系统表、临时表、HUGE 表、水平分区表、物化视图、含有聚集主键的表、含有位图连接索引的表、含有位图索引的表上；

例如，将策略 P_04 应用于 PRODUCTION 模式中 PRODUCT 表，指定标记列为 LABEL_COL，不隐藏标记列。

```
MAC_APPLY_TABLE_POLICY ('P_04', 'PRODUCTION', 'PRODUCT', 'LABEL_COL',
'L_01::',0);
```

过程中的 OPTION 参数用来指定该标记列是否隐藏。若对表应用策略时指定新增的这一列隐藏，那么对表进行 INSERT 数据时，如果未指定标记列的列名，就不能对这一列插入数据，如果对其插入数据，则会出错；如果指明具体的列来插入数据，这一列是允许插入数据的。当执行 SELECT * 来查询该表数据时，标记列被隐藏不予显示；但也允许通过指明列名来查询该列。若对表应用策略时指定新增的这一列为 0，那么该列就可以被视为一个普通列。

当一个新的策略被应用于表上时，过程的 LABELVALUE 参数用于初始化表中已有的数据关于该策略的标记。当某个策略已被应用于表时，则其已有的等级、范围和组均不能被删除，除非从所有的用户和表上取消对该策略的应用。

4.3.2 取消表策略

使用下述系统过程取消表上的指定策略：

```
VOID
MAC_REMOVE_TABLE_POLICY（
       POLICY_NAME   VARCHAR(128),
       SCHEMANAME    VARCHAR(128),
       TABLENAME     VARCHAR(128),
       DROP_COLUMN   INT
）;
```

参数说明：

POLICY_NAME　表上待取消应用的策略

SCHEMANAME　表所属模式名称

TABLENAME　表名称

DROP_COLUMN　1 代表删除标记列，0 代表不删除标记列，缺省为 0

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在；

例如，取消应用在 PRODUCTION 模式中 PRODUCT 表上的策略，删除标记列。

```
MAC_REMOVE_TABLE_POLICY ('P_04', 'PRODUCTION', 'PRODUCT' ,1);
```

4.4 如何对用户应用策略

用户的标记来源于策略。对用户应用一个策略，要指定一个最高等级、最低等级、默认等级和行等级，其中默认等级介于最高等级和最低等级之间，最高等级限制了用户最高的读写权限，最低等级限制了用户的写权限，默认等级表示用户登录时会话标记的等级，行等级表示插入一行数据的标记使用的等级；指定范围和组，对范围需要指定读的范围、写的范围、默认范围和行级范围，其中写范围必须是读范围的子集，默认范围是读范围的子集，行级范围是写范围和默认范围的交集的子集。组的规则与范围相同。用户标记属性如下表所示：

表 4.1 用户标记属性表

组成	备注
max_level	最大读写级别
min_level	最小写级别
def_level	默认级别，登录时级别
row_level	列级别，用于默认插入
categories	包含的范围，可设置属性
groups	包含的组，可设置属性

4.4.1 设置用户等级

对用户应用策略，应先设置用户的等级，使用如下系统过程：

```
VOID
MAC_USER_SET_LEVELS(
       POLICY_NAME          VARCHAR(128),
       USER_NAME            VARCHAR(128),
       MAX_LEVEL            VARCHAR(128),
       MIN_LEVEL            VARCHAR(128),
       DEF_LEVEL            VARCHAR(128),
       ROW_LEVEL            VARCHAR(128)
);
```

参数说明：

POLICY_NAME　应用于用户的策略名称

USER_NAME　策略所应用的用户名称

MAX_LEVEL　应用于用户的最大等级

MIN_LEVEL　应用于用户的最小等级

DEF_LEVEL　应用于用户的默认等级

ROW_LEVEL　应用于用户的行等级

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在；

3. MAX_LEVEL 不能为空；

4. MIN_LEVEL 不指定时，设置为策略的最小级别；

5. DEF_LEVEL 不指定时，设置为 MAX_LEVEL；

6. ROW_LEVEL 不指定时，设置为 DEF_LEVEL；

7. 合法的 LEVEL 规则如下：

   MAX_LEVEL >= MIN_LEVEL

   MAX_LEVEL >= DEF_LEVEL >= MIN_LEVEL

   DEF_LEVEL >= ROW_LEVEL >= MIN_LEVEL

例如，设置 BOOKSHOP_USER 用户的等级。等级 L_01，L_02，L_03，L_04 已存在，且等级级别相等，或依次增加。

```
MAC_USER_SET_LEVELS('P_04', 'BOOKSHOP_USER', 'L_04', 'L_01', 'L_03', 'L_02');
```

4.4.2 设置用户范围

对用户设置了等级后，可以对用户设置相应的范围和组。

使用下面的系统过程为用户设置范围：

```
VOID
MAC_USER_SET_COMPARTMENTS(
       POLICY_NAME          VARCHAR(128),
       USER_NAME            VARCHAR(128),
       READ                 VARCHAR(128),
       WRITE                VARCHAR(128),
       DEFAULT              VARCHAR(128),
       ROW                  VARCHAR(128)
);
```

参数说明：

POLICY_NAME	应用于用户的策略名称
USER_NAME	策略所应用的用户名称
READ	应用于用户的可读范围

WRITE 应用于用户的可写范围

DEFAULE 应用于用户的默认范围

ROW 应用于用户的行范围

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；

2. 用户范围具有 READ、WRITE、DEFAULT 和 ROW 四个属性，说明如下：

1) READ：用户的可读范围，READ 不能为空；

2) WRITE：用户的可写范围，进行 UPDATE，DELETE 时需要使用，进行判断权限。WRITE 属于 READ，为空时使用 READ；

3) DEFAULT：会话默认使用的范围。DEFAULT 属于 READ，为空时使用 READ；

4) ROW：插入时不指定标记时使用。ROW 属于 DEFAULT 和 WRITE 的交集，为空时使用 DEFAULT 和 WRITE 的交集。

例如，假设范围 C_01,C_02,C_03 已存在，设置 BOOKSHOP_USER 用户的范围。

```
MAC_USER_SET_COMPARTMENTS('P_04','BOOKSHOP_USER','C_01,C_02,C_03','C_01,C_02'
,'C_01,C_03','C_01');
```

4.4.3 设置用户组

使用下面的系统过程为用户设置组：

```
VOID
MAC_USER_SET_GROUPS (
       POLICY_NAME          VARCHAR(128),
       USER_NAME            VARCHAR(128),
       READ_GROUP           VARCHAR(128),
       WRITE_GROUP          VARCHAR(128),
       DEF_GROUP            VARCHAR(128),
       ROW_GROUP            VARCHAR(128)
);
```

参数说明：

POLICY_NAME 应用于用户的策略名称

USER_NAME　　策略所应用的用户名称

READ_GROUP　　应用于用户的可读组

WRITE_GROUP　　应用于用户的可写组

DEF_GROUP　　应用于用户的默认组

ROW_GROUP　　应用于用户的行组

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 用户的组具有 READ、WRITE、DEFAULT 和 ROW 四个属性，说明如下：
   1) READ：用户的可读组；
   2) DEFAULT：会话默认使用的组；
   3) WRITE：进行 UPDATE，DELETE 时需要使用，进行判断权限；
   4) ROW：插入时不指定标记时使用；
   5) 合法的用户组规则如下：
      - WRITE 属于 READ
      - DEFAULT 属于 READ
      - ROW 属于 DEFAULT 和 WRITE 的交集

例如，假设组 G_01,G_02,G_03 已存在，设置 BOOKSHOP_USER 用户的组。

```
MAC_USER_SET_GROUPS ('P_04', 'BOOKSHOP_USER', 'G_01,G_02,G_03', 'G_02,G_03',
'G_01,G_03', 'G_03');
```

4.4.4 清除用户策略

使用下面的系统过程清除应用于某用户的指定策略。

```
VOID
MAC_USER_REMOVE_POLICY(
       POLICY_NAME          VARCHAR(128),
       USER_NAME            VARCHAR(128)
);
```

参数说明：

POLICY_NAME　　要清除的策略名

USER_NAME　　用户名

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在。

例如，清除用户 BOOKSHOP_USER 的策略 P_04。

```
MAC_USER_REMOVE_POLICY ('P_04', 'BOOKSHOP_USER');
```

4.5 如何对会话应用策略

为了方便用户在执行时进行 MAC 权限的设置，系统提供了针对会话的标记设置，这些设置会挂载在会话上，一旦会话结束，这些信息就完全被抛弃，而不会影响系统表。

用户可以设置自身当前会话的默认标记，但其必须在合法的范围内，等级必须在用户的 MIN_LEVEL 和 MAX_LEVEL 之间，范围必须是用户 READ_COMP 的子集，组必须是用户 READ_GROUP 的子集。用户还可以设置会话的行级标记，同样必须在合法的范围内。

4.5.1 设置会话默认标记

使用如下的系统过程设置会话的默认标记：

```
VOID
MAC_SET_SESSION_LABEL(
       POLICY_NAME          VARCHAR(128),
       LABELVALUE           VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　　策略名

LABELVALUE　　为会话设置的默认标记

使用说明：

1. 设置的等级必须在用户的 MIN_LEVEL 和 MAX_LEVEL 之间，范围必须是用户 READ_COMP 的子集，组必须是用户 READ_GROUP 的子集；
2. 由于行标记的范围来源于会话标记，故重置会话标记后，需对行标记进行调整。

例如，接 4.4.1、4.4.2、4.4.3 节的例子，BOOKSHOP_USER 用户设置会话默认标记。

```
MAC_SET_SESSION_LABEL('P_04', 'L_03:C_01,C_02:G_01,G_03');
```

4.5.2 设置会话行标记

使用如下的系统过程设置会话的行标记：

```
VOID
MAC_SET_SESSION_ROW_LABEL(
       POLICY_NAME          VARCHAR(128),
       LABELVALUE           VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　策略名

LABELVALUE　为会话设置的行标记

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. 设置的行标记必须是合法值。

例如，设置当前会话行标记。

```
MAC_SET_SESSION_ROW_LABEL('P_04', 'L_01:C_02:G_01');
```

4.5.3 清除会话标记

使用如下的系统过程清除会话上对应某策略的标记：

```
VOID
```

```
MAC_RESTORE_DEFAULT_LABELS(
       POLICY_NAME          VARCHAR(128)
);
```

参数说明：

POLICY_NAME　指定策略名

使用说明：

清除当前会话对应指定策略的所有的标记，仅用户的标记可用。

例如，清除当前会话对应策略 P_04 的所有标记。

```
MAC_RESTORE_DEFAULT_LABELS('P_04');
```

4.5.4 保存会话标记

会话上的标记只在会话生存期间存在，并没有保存到数据库的数据字典中，一旦会话结束，会话标记就被丢弃。DM 提供以下的系统过程允许用户将会话上指定策略刷入相应的数据字典，这时，用户标记会被会话标记覆盖，会话标记与用户标记一致。

```
VOID
MAC_SAVE_DEFAULT_LABELS (
       POLICY_NAME    VARCHAR(128)
);
```

参数说明：

POLICY_NAME　要保存的会话标记所在的策略名

例如，将当前会话上策略 P_04 的标记刷入相应的数据字典。

```
MAC_SAVE_DEFAULT_LABELS('P_04');
```

4.6 读写控制规则

DM 支持同时使用自主访问控制与强制访问控制策略。自主访问控制策略优先于强制访问控制策略。为了能访问一条元组，用户不仅首先要满足自主访问控制条件，还必须满足强制访问控制的条件。

用户访问表时，必须保证应用于表上的所有策略均适用于该用户，否则访问被拒绝。若表上未应用任何策略，则用户只需满足自主访问控制条件即可。

4.6.1 读访问规则

强制访问控制的读访问规则为：

1． 用户的等级必须大于等于数据的等级；

2． 用户的标记必须包含至少一个数据的组（或者是其某一个的父亲组）；

3． 用户的标记必须包含数据的所有的范围。

如果用户满足以上三个规则，即可以对数据进行读取。比较的先后顺序是等级，组，范围。DM 中实现读访问规则的流程如图 4.1 所示。

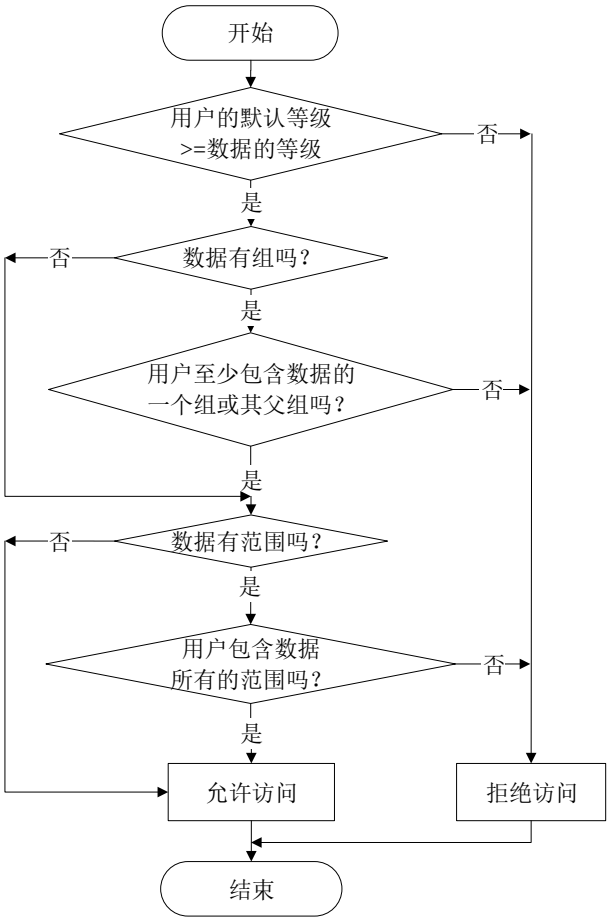

图 4.1 强制访问控制读访问规则

4.6.2 写访问规则

强制访问控制的写访问规则为：

1． 数据标记的等级必须大于等于用户标记的最小等级，小于或等于用户的会话标记的等级；

2． 用户的标记必须包含至少一个数据的组（或者是其某一个的父亲组）的写权限；

3． 用户的标记必须包含数据的所有的范围上的写权限。

如果用户满足以上三个规则，即可以对数据进行写操作。比较的先后顺序是等级，组，范围。DM 中实现写访问规则的流程如图 4.2 所示。

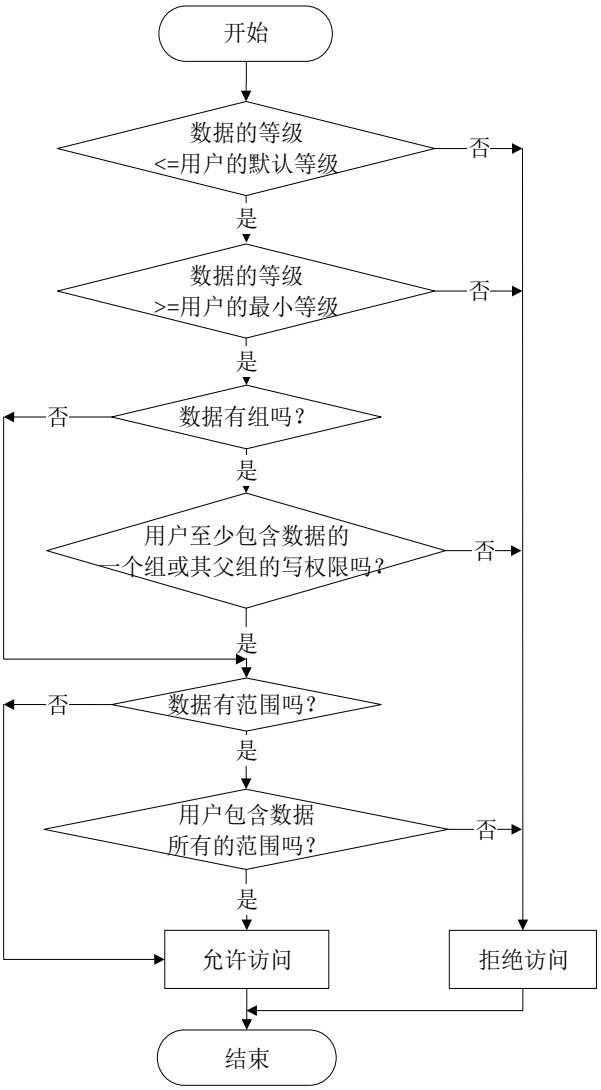

图 4.2 强制访问控制写访问规则

4.6.3 特权

在给用户应用策略时，可以同时授予策略特权。策略特权分为访问特权和行标记特权。

访问特权分为两种：

- ✓ READ：读数据时不受策略影响，但写数据访问控制仍然受到强制访问控制
- ✓ FULL：可以读写任何数据，不受策略影响

一旦一个行的标记设定后，就需要行标记特权才能 UPDATE 其标记列。行标记特权有如下三种：

- ✓ WRITE UP：用户可以利用该特权提升一个行的等级，同时不改变范围和组。这个等级可以提高到用户的最高等级，而该行的原始等级可能比用户的最低等级还低

- ✓ WRITE DOWN：用户可以利用该特权降低一个行的等级，同时不改变范围和组。这个等级可以降低到用户的最低等级，而该行的原始等级可能比用户的最低等级还低
- ✓ WRITE ACROSS：用户可以利用该特权修改一个行的范围和组，同时不改变等级。新的范围和组只要满足在策略中是合理的就可以了，不必限于用户拥有访问权的范围和组

使用下面的系统过程为用户设置特权：

```
VOID
MAC_USER_SET_USER_PRIVS(
       POLICY_NAME                VARCHAR(128),
       USER_NAME                        VARCHAR(128),
       PRIVS                            VARCHAR(128)
);
```

参数说明：

POLICY_NAME　特权对应的策略名

USER_NAME　为其设置特权的用户名

PRIVS　特权类型

使用说明：

1. 该过程只能由具有 LABEL_DATABASE 的用户调用；
2. PRIVS 取值可以是 READ、FULL、WRITEUP、WRITEDOWN、WRITEACCROSS 中的一种或几种的组合。

例如，设置 BOOKSHOP_USER 用户的特权。

```
MAC_USER_SET_USER_PRIVS('P1', 'BOOKSHOP_USER', 'READ,WRITEUP');
```

4.7 扩展客体标记

除了行标记，DM 的扩展客体标记支持对数据库除约束和目录外的所有的客体进行标记，如模式、表、索引等对象。一旦一个对象被应用了扩展客体标记，则用户只有在支配相应标记的情况下，才能访问客体。

创建的新对象不会含有默认的标记，需要安全员进行手工设置。

4.7.1 对客体应用标记

使用下面的系统过程可以对客体应用标记：

```
VOID
MAC_APPLY_OBJ_POLICY(
       POLICY_NAME          VARCHAR(128),
       OBJ_TYPE             VARCHAR(128),
       SCH_NAME             VARCHAR(128),
       OBJ_NAME             VARCHAR(128),
       COL_NAME             VARCHAR(128),
       LABEL                VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　策略名称

OBJ_TYPE　客体对象类型，必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”、“DOMAIN”、“CONTEXT INDEX”、“CONTEXT”

SCH_NAME　客体所在的模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词

OBJ_NAME　应用标记的客体名称，如果对模式应用策略，则此值和 SCH_NAME 值一致

COL_NAME　应用标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效

LABEL　应用的标记值

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在。

例如，对模式 PRODUCTION 应用策略 P1 的标记（L1:C1:G1）。

```
MAC_APPLY_OBJ_POLICY('P1', 'SCHEMA', 'PRODUCTION', 'PRODUCTION', NULL,
'L1:C1:G1');
```

4.7.2 修改客体标记

使用下面的系统过程修改客体的标记：

```
VOID
MAC_ALTER_OBJ_POLICY(
       POLICY_NAME          VARCHAR(128),
       OBJ_TYPE             VARCHAR(128),
       SCH_NAME             VARCHAR(128),
       OBJ_NAME             VARCHAR(128),
       COL_NAME             VARCHAR(128),
       LABEL                VARCHAR(4000)
);
```

参数说明：

POLICY_NAME　策略名称

OBJ_TYPE　客体对象类型，必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”、“DOMAIN”、“CONTEXT INDEX”、“CONTEXT”

SCH_NAME　客体所在的模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词

OBJ_NAME　修改标记的客体名称，如果对模式应用策略，则此值和 SCH_NAME 值一致

COL_NAME　修改标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效

LABEL　修改的标记值

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在。

例如，将模式 PRODUCTION 的标记修改为策略 P1 的标记（L1:C1,C2:G1）。

```
MAC_ALTER_OBJ_POLICY ('P1', 'SCHEMA', 'PRODUCTION', 'PRODUCTION', NULL,
'L1:C1,C2:G1');
```

4.7.3 删除客体标记

使用下面的系统过程删除指定客体上对应指定策略的标记：

```
VOID
MAC_DROP_OBJ_POLICY(
       POLICY_NAME          VARCHAR(128),
       OBJ_TYPE             VARCHAR(128),
       SCH_NAME             VARCHAR(128),
       OBJ_NAME             VARCHAR(128),
       COL_NAME             VARCHAR(128)
);
```

参数说明：

POLICY_NAME　策略名称

OBJ_TYPE　客体对象类型，必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”、“DOMAIN”、“CONTEXT INDEX”、“CONTEXT”

SCH_NAME　客体所在的模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词

OBJ_NAME　删除标记的客体名称，如果对模式应用策略，则此值和 SCH_NAME 值一致

COL_NAME　删除标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效

使用说明：

1． 该过程只能由具有 LABEL_DATABASE 的用户调用；

2． 指定策略必须存在。

例如，删除模式 PRODUCTION 与策略 P1 相关的标记。

```
MAC_DROP_OBJ_POLICY('P1', 'SCHEMA', 'PRODUCTION', 'PRODUCTION',NULL);
```

4.8 一个强制访问控制的例子

下面给出一个强制访问控制的例子，综合运用了本章介绍的一些主要概念和方法。强制访问控制在真实应用中的使用会复杂得多，需要安全管理员事先进行全面的策略设计。

1. SYSSSO 登录，创建一个策略 P_TEST

```
//SYSSSO/SYSSSO，创建策略
MAC_CREATE_POLICY('P_TEST');
MAC_CREATE_LEVEL('P_TEST', 11, 'L_01');
MAC_CREATE_LEVEL('P_TEST', 12, 'L_02');
MAC_CREATE_LEVEL('P_TEST', 13, 'L_03');
MAC_CREATE_LEVEL('P_TEST', 14, 'L_04');
MAC_CREATE_COMPARTMENT('P_TEST',11, 'C_01');
MAC_CREATE_COMPARTMENT('P_TEST',12, 'C_02');
MAC_CREATE_COMPARTMENT('P_TEST',13, 'C_03');
MAC_CREATE_COMPARTMENT('P_TEST',14, 'C_04');
MAC_CREATE_GROUP ('P_TEST',11, 'G_01',NULL);
MAC_CREATE_GROUP ('P_TEST',12, 'G_02','G_01');
MAC_CREATE_GROUP ('P_TEST',13, 'G_03','G_02');
```

2. SYSDBA 登录，创建一个用户 USER_TEST，并授予其 RESOURCE 角色

```
//SYSDBA/SYSDBA，创建用户 USER_TEST，并授予其 RESOURCE 角色
CREATE USER USER_TEST IDENTIFIED BY TEST12345;
GRANT RESOURCE TO USER_TEST;
```

3. USER_TEST 登录，创建一个表 TEST，插入两行数据，并查询，此时由于没有对表应用策略，表中没有标记列

```
// USER_TEST/TEST12345，创建表 TEST，插入数据
CREATE TABLE TEST(C1 INT, C2 INT);
INSERT INTO TEST VALUES(1,1);
INSERT INTO TEST VALUES(2,2);
```

```
COMMIT;
SELECT * FROM TEST;
行号        C1          C2
---------- ----------- -----------
1           1           1
2           2           2
```

4. SYSSSO 登录，为用户 USER_TEST 和表 USER_TEST.TEST 应用策略

```
// SYSSSO/SYSSSO，为用户 USER_TEST 应用策略，为表 USER_TEST.TEST 应用策略
MAC_USER_SET_LEVELS('P_TEST', 'USER_TEST', 'L_04', 'L_01', 'L_03', 'L_02');
MAC_USER_SET_COMPARTMENTS('P_TEST', 'USER_TEST', 'C_01,C_02,C_03',
'C_01,C_02', 'C_01,C_03', 'C_01');
MAC_USER_SET_GROUPS ('P_TEST', 'USER_TEST', 'G_01,G_02,G_03', 'G_02,G_03',
'G_01,G_03', 'G_03');
MAC_APPLY_TABLE_POLICY ('P_TEST', 'USER_TEST', 'TEST', 'LABEL_COL',
'L_01::',0);
```

5. USER_TEST 登录，此时查询表 TEST，可看到表中已有行的 LABEL_COL 值为应用表策略时指定的值

```
// USER_TEST/TEST12345
SELECT * FROM TEST;
行号        C1          C2          LABEL_COL
---------- ----------- ----------- -----------
1           1           1           11
2           2           2           11
```

6. USER_TEST 继续操作，向表 TEST 插入一行新的记录，可以看到这行记录的标记为 USER_TEST 的行标记

```
INSERT INTO TEST(C1,C2) VALUES(3,3);
COMMIT;
SELECT * FROM TEST;
行号        C1          C2          LABEL_COL
---------- ----------- ----------- -----------
```

```
1          1          1          11

2          2          2          11

3          3          3          21

SELECT SF_MAC_LABEL_TO_CHAR(21);

行号       SF_MAC_LABEL_TO_CHAR(21)

---------- ------------------------

1          L_02:C_01:G_03
```

7. USER_TEST 继续操作，设置会话默认标记和会话行标记，再向表 TEST 中插入一行数据，可看到这一行的标记为会话的行标记

```
MAC_SET_SESSION_LABEL('P_TEST', 'L_03:C_01,C_02:G_01,G_03');

MAC_SET_SESSION_ROW_LABEL('P_TEST', 'L_01:C_01:G_03');

INSERT INTO TEST(C1, C2) VALUES(4,4);

COMMIT;

SELECT * FROM TEST;

行号       C1          C2          LABEL_COL

---------- ----------- ----------- -----------

1          1           1           11

2          2           2           11

3          3           3           21

4          4           4           24

SELECT SF_MAC_LABEL_TO_CHAR(24);

行号       SF_MAC_LABEL_TO_CHAR(24)

---------- ------------------------

1          L_01:C_01:G_03
```

8. SYSDBA 登录，查询表 USER_TEST.TEST，由于此时 SYSDBA 没有被应用策略 P_TEST，因此一行数据也不能访问

```
// SYSDBA/SYSDBA

SELECT * FROM USER_TEST.TEST;

未选定行
```

9. SYSSSO 登录，为 SYSDBA 应用策略，设置 SYSDBA 的标记

```
// SYSSSO/SYSSSO，为 SYSDBA 应用策略
MAC_USER_SET_LEVELS('P_TEST', 'SYSDBA', 'L_01', 'L_01', 'L_01', 'L_01');
MAC_USER_SET_COMPARTMENTS('P_TEST', 'SYSDBA', 'C_01,C_02,C_03', 'C_01,C_02',
'C_01,C_03', 'C_01');
MAC_USER_SET_GROUPS ('P_TEST', 'SYSDBA', 'G_01,G_02,G_03', 'G_02,G_03',
'G_01,G_03', 'G_03');
```

10. SYSDBA 登录，再次查询表 USER_TEST.TEST，根据读访问规则，可以查询到三条记录

```
// SYSDBA/SYSDBA
SELECT * FROM USER_TEST.TEST;
行号        C1          C2          LABEL_COL
---------- ----------- ----------- -----------
1           1           1           11
2           2           2           11
3           4           4           24
```

11. SYSSSO 登录，为 SYSDBA 设置 READ 特权

```
// SYSSSO/SYSSSO，为 SYSDBA 设置特权
MAC_USER_SET_USER_PRIVS('P_TEST', 'SYSDBA', 'READ');
```

12. SYSDBA 登录，再次查询表 USER_TEST.TEST，由于拥有 READ 特权，可以查询到全部记录

```
// SYSDBA/SYSDBA
SELECT * FROM USER_TEST.TEST;
行号        C1          C2          LABEL_COL
---------- ----------- ----------- -----------
1           1           1           11
2           2           2           11
3           3           3           21
4           4           4           24
```

4.9 相关数据字典表

DM 中与强制访问控制相关的数据字典表有以下这些，这些字典表属于模式“SYSSSO”，只有具有 SSO 类型的用户才能查询。

1. SYSMACPLYS

记录策略定义。

序号	列	数据类型	说明
1	ID	INTEGER	策略 ID
2	NAME	VARCHAR(128)	策略名

2. SYSMACLVLS

记录策略的等级。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	等级 ID
3	NAME	VARCHAR(128)	等级名

3. SYSMACCOMPS

记录策略的范围。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	范围 ID
3	NAME	VARCHAR(128)	范围名

4. SYSMACGRPS

记录策略所在组的信息。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	SMALLINT	组 ID
3	PARENTID	SMALLINT	父节点 ID
4	NAME	VARCHAR(128)	组名

5. SYSMACLABELS

记录策略的标记信息。

序号	列	数据类型	说明
1	PID	INTEGER	策略 ID
2	ID	INTEGER	标记 ID

3	LABEL	VARCHAR(4000)	标记信息

6. SYSMACTABPLY

记录表策略信息。

序号	列	数据类型	说明
1	TID	INTEGER	表 ID
2	PID	INTEGER	策略 ID
3	COLID	SMALLINT	列 ID
4	OPTIONS	BYTE	可见性

7. SYSMACUSRPLY

记录用户的策略信息。

序号	列	数据类型	说明
1	UID	INTEGER	用户 ID
2	PID	INTEGER	策略 ID
3	MAXREAD	INTEGER	最大读标记 ID
4	MINWRITE	INTEGER	最小写标记 ID
5	DEFTAG	INTEGER	默认标记 ID
6	ROWTAG	INTEGER	行级标记 ID
7	PRIVS	BYTE	特权

8. SYSMACOBJ

记录扩展客体标记信息。

序号	列	数据类型	说明
1	OBJID	INTEGER	对象 ID
2	COLID	SMALLINT	列 ID
3	PID	INTEGER	策略 ID
4	TAG	INTEGER	标记 ID

5 审计

审计机制是 DM 数据库管理系统安全管理的重要组成部分之一。DM 数据库除了提供数据安全保护措施外，还提供对日常事件的事后审计监督。DM 具有一个灵活的审计子系统，可以通过它来记录系统级事件、个别用户的行为以及对数据库对象的访问。通过考察、跟踪审计信息，数据库审计员可以查看用户访问的形式以及曾试图对该系统进行的操作，从而采取积极、有效的应对措施。

5.1 审计开关

在 DM 系统中，专门为审计设置了开关，要使用审计功能首先要打开审计开关。审计开关由过程 VOID SP_SET_ENABLE_AUDIT（param int）;控制，过程执行完后会立即生效，param 有三种取值：

- ✓ 0：关闭审计
- ✓ 1：打开普通审计
- ✓ 2：打开普通审计和实时审计

缺省值为 0。

例，打开普通审计开关。

```
SP_SET_ENABLE_AUDIT (1);
```

审计开关必须由具有数据库审计员权限的管理员进行设置。

数据库审计员可通过查询 V$DM_INI 动态视图查询 ENABLE_AUDIT 的当前值。

```
SELECT * FROM V$DM_INI WHERE PARA_NAME='ENABLE_AUDIT';
```

5.2 审计的设置与取消

数据库审计员指定被审计对象的活动称为审计设置，只有具有 AUDIT DATABASE 权限的审计员才能进行审计设置。DM 提供审计设置系统过程来实现这种设置，被审计的对象可以是某类操作，也可以是某些用户在数据库中的全部行踪。只有预先设置的操作和用户才能被 DM 系统自动进行审计。

DM 允许在三个级别上进行审计设置，如表 5.1 所示。

表 5.1 审计级别

审计级别	说明
系统级	系统的启动与关闭，此级别的审计无法也无需由用户进行设置，只要审计开关打开就会自动生成对应审计记录
语句级	导致影响特定类型数据库对象的特殊 SQL 或语句组的审计。如 AUDIT TABLE 将审计 CREATE TABLE、ALTER TABLE 和 DROP TABLE 等语句
对象级	审计作用在特殊对象上的语句。如 test 表上的 INSERT 语句

审计设置存放于 DM 字典表 SYSAUDIT 中，进行一次审计设置就在 SYSAUDIT 中增加一条对应的记录，取消审计则删除 SYSAUDIT 中相应的记录。

5.2.1 语句级审计

语句级审计的动作是全局的，不对应具体的数据库对象。其审计选项如表 5.2 所示。

表 5.2 语句级审计选项

审计选项	审计的数据库操作	说明
ALL	所有的语句级审计选项	所有可审计操作
USER	CREATE USER ALTER USER DROP USER	创建 / 修改 / 删除用户操作
ROLE	CREATE ROLE DROP ROLE	创建 / 删除角色操作
TABLESPACE	CREATE TABLESPACE ALTER TABLESPACE DROP TABLESPACE	创建 / 修改 / 删除表空间操作
SCHEMA	CREATE SCHEMA DROP SCHEMA SET SCHEMA	创建 / 删除 / 设置当前模式操作
TABLE	CREATE TABLE ALTER TABLE DROP TABLE TRUNCATE TABLE	创建 / 修改 / 删除 / 清空基表操作
VIEW	CREATE VIEW ALTER VIEW DROP VIEW	创建 / 修改 / 删除视图操作
INDEX	CREATE INDEX DROP INDEX	创建 / 删除索引操作
PROCEDURE	CREATE PROCEDURE ALTER PROCEDURE	创建 / 修改 / 删除存储模块操作

	DROP PROCEDURE	
TRIGGER	CREATE TRIGGER ALTER TRIGGER DROP TRIGGER	创建 / 修改 / 删除触发器操作
SEQUENCE	CREATE SEQUENCE ALTER SEQUENCE DROP SEQUENCE	创建 / 修改/删除序列操作
CONTEXT	CREATE CONTEXT INDEX ALTER CONTEXT INDEX DROP CONTEXT INDEX	创建 / 修改 / 删除全文索引操作
SYNONYM	CREATE SYNONYM DROP SYNONYM	创建 / 删除同义词
GRANT	GRANT	授予权限操作
REVOKE	REVOKE	回收权限操作
AUDIT	AUDIT	设置审计操作
NOAUDIT	NOAUDIT	取消审计操作
INSERT TABLE	INSERT INTO TABLE	表上的插入操作
UPDATE TABLE	UPDATE TABLE	表上的修改操作
DELETE TABLE	DELETE FROM TABLE	表上的删除操作
SELECT TABLE	SELECT FROM TABLE	表上的查询操作
EXECUTE PROCEDURE	CALL PROCEDURE	调用存储过程或函数操作
PACKAGE	CREATE PACKAGE DROP PACKAGE	创建 / 删除包规范
PACKAGE BODY	CREATE PACKAGE BODY DROP PACKAGE BODY	创建 / 删除包体
MAC POLICY	CREATE POLICY ALTER POLICY DROP POLICY	创建 / 修改 / 删除策略
MAC LEVEL	CREATE LEVEL ALTER LEVEL DROP LEVEL	创建 / 修改 / 删除等级
MAC COMPARTMENT	CREATE COMPARTMENT ALTER COMPARTMENT DROP COMPARTMENT	创建 / 修改 / 删除范围
MAC GROUP	CREATE GROUP ALTER GROUP DROP GROUP ALTER GROUP PARENT	创建 / 修改 / 删除组，更新父组
MAC LABEL	CREATE LABEL ALTER LABEL DROP LABEL	创建 / 修改 / 删除标记
MAC USER	USER SET LEVELS	设置用户等级 / 范围 / 组 /

	USER SET COMPARTMENTS USER SET GROUPS USER SET PRIVS	特权
MAC TABLE	INSER TABLE POLICY REMOVE TABLE POLICY APPLY TABLE POLICY	插入 / 取消 / 应用表标记
MAC SESSION	SESSION LABEL SESSION ROW LABEL RESTORE DEFAULT LABELS SAVE DEFAULT LABELS	保存 / 取消会话标记 设置会话默认标记 设置会话行标记
CHECKPOINT	CHECKPOINT	检查点（checkpoint）
SAVEPOINT	SAVEPOINT	保存点
EXPLAIN	EXPLAIN	显示执行计划
NOT EXIST		分析对象不存在导致的错误
DATABASE	ALTER DATABASE	修改当前数据库操作
CONNECT	LOGIN LOGOUT	登录 / 退出操作
COMMIT	COMMIT	提交操作
ROLLBACK	ROLLBACK	回滚操作
SET TRANSACTION	SET TRX ISOLATION SET TRX READ WRITE	设置事务的读写属性和隔离级别
BACKUP	BACKUP DATABASE BACKUP TABLESPACE BACKUP TABLE BACKUP ARCHIVE	库/表空间/表/归档备份操作
RESTORE	RESTORE TABLESPACE RESTORE TABLE	表空间/表还原操作
DIMP	DIMP FULL DIMP OWNER DIMP SCHEMA DIMP TABLE	逻辑导入：库级/用户级/模式级/表级
DEXP	DEXP FULL DEXP OWNER DEXP SCHEMA DEXP TABLE	逻辑导出：库级/用户级/模式级/表级
FLDR	FLDR IN FLDR OUT	FLDR 工具导入/导出
WARNING	AUD SPACE WARNING	审计剩余可用空间不足
KEY	CREATE KEY DESTROY KEY	生成/销毁密钥
CRYPT	ENCRYPT DECRYPT	数据加密/解密
DTS	DTS IN	DTS 工具迁入/迁出

	DTS OUT	

设置语句级审计的系统过程如下：

```
VOID
SP_AUDIT_STMT(
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       WHENEVER      VARCHAR (20)
)
```

参数说明：

TYPE　　语句级审计选项，即上表中的第一列

USERNAME　　用户名，NULL 表示不限制

WHENEVER　　审计时机，可选的取值为：

- ALL：所有的
- SUCCESSFUL：操作成功时
- FAIL：操作失败时

例 1 审计表的创建、修改和删除。

```
SP_AUDIT_STMT('TABLE', 'NULL', 'ALL');
```

例 2 对 SYSDBA 创建用户成功进行审计。

```
SP_AUDIT_STMT('USER', 'SYSDBA', 'SUCCESSFUL');
```

例 3 对用户 USER2 进行的表的修改和删除进行审计，不管失败和成功。

```
SP_AUDIT_STMT('UPDATE TABLE', 'USER2', 'ALL');
SP_AUDIT_STMT('DELETE TABLE', 'USER2', 'ALL');
```

取消语句级审计的系统过程如下：

```
VOID
SP_NOAUDIT_STMT(
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       WHENEVER      VARCHAR (20)
```

)

参数说明：

TYPE　　语句级审计选项，即上表中的第一列

USERNAME　　用户名，NULL 表示不限制

WHENEVER　　审计时机，可选的取值为：

- ALL：所有的
- SUCCESSFUL：操作成功时
- FAIL：操作失败时

使用说明：

取消审计语句和设置审计语句进行匹配，只有完全匹配的才可以取消审计，否则无法取消审计。

例 1 取消对表的创建、修改和删除的审计。

```
SP_NOAUDIT_STMT('TABLE', 'NULL', 'ALL');
```

例 2 取消对 SYSDBA 创建用户成功进行审计。

```
SP_NOAUDIT_STMT('USER', 'SYSDBA', 'SUCCESSFUL');
```

例 3 取消对用户 USER2 进行的表的修改和删除的审计。

```
SP_NOAUDIT_STMT('UPDATE TABLE', 'USER2', 'ALL');
SP_NOAUDIT_STMT('DELETE TABLE', 'USER2', 'ALL');
```

5.2.2 对象级审计

对象级审计发生在具体的对象上，需要指定模式名以及对象名。其审计选项如表 5.3 所示。

表 5.3　对象级审计选项

审计选项(SYSAUDITRECORDS 表中 operation 字段对应的内容)	TABLE	VIEW	COL	PROCEDURE FUNCTION	TRIGGER
INSERT	√	√	√		
UPDATE	√	√	√		
DELETE	√	√	√		
SELECT	√	√	√		
EXECUTE				√	

MERGE INTO	√	√			
EXECUTE TRIGGER					√
LOCK TABLE	√				
BACKUP TABLE	√				
RESTORE TABLE	√				
ALL(所有对象级审计选项)	√	√	√	√	√

其中，对于 UPDATE 和 DELETE 操作，因为也需要做 SELECT 操作，所以只要设置审计 SELECT 操作时，UPDATE 和 DELETE 也会作为 SELECT 操作被审计。

设置对象级审计的系统过程如下：

```
VOID
SP_AUDIT_OBJECT (
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       SCHNAME       VARCHAR (128),
       TVNAME        VARCHAR (128),
       WHENEVER      VARCHAR (20)
)

VOID
SP_AUDIT_OBJECT (
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       SCHNAME       VARCHAR (128),
       TVNAME        VARCHAR (128),
       COLNAME        VARCHAR (128),
       WHENEVER      VARCHAR (20)
)
```

参数说明：

TYPE　　对象级审计选项，即上表中的第一列

USERNAME　　用户名

SCHNAME　　模式名，为空时置‘null’

TVNAME 表、视图、存储过程名不能为空

COLNAME 列名

WHENEVER 审计时机，可选的取值为：

- ALL：所有的
- SUCCESSFUL：操作成功时
- FAIL：操作失败时

例 1 对 SYSDBA 对表 PERSON.ADDRESS 进行的添加和修改的成功操作进行审计。

```
SP_AUDIT_OBJECT('INSERT', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');
SP_AUDIT_OBJECT('UPDATE', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');
```

例 2 对 SYSDBA 对表 PERSON.ADDRESS 的 ADDRESS1 列进行的修改成功的操作进行审计。

```
SP_AUDIT_OBJECT('UPDATE','SYSDBA','PERSON','ADDRESS','ADDRESS1','SUCCESSFUL');
```

取消对象级审计的系统过程如下：

```
VOID
SP_NOAUDIT_OBJECT (
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       SCHNAME       VARCHAR (128),
       TVNAME        VARCHAR (128),
       WHENEVER      VARCHAR (20)
)

VOID
SP_NOAUDIT_OBJECT (
       TYPE          VARCHAR(30),
       USERNAME      VARCHAR (128),
       SCHNAME       VARCHAR (128),
       TVNAME        VARCHAR (128),
       COLNAME        VARCHAR (128),
```

```
    WHENEVER    VARCHAR (20)
)
```

参数说明：

TYPE　　对象级审计选项，即上表中的第一列

USERNAME　　用户名

SCHNAME　　模式名，为空时置‘null’

TVNAME　　表、视图、存储过程名不能为空

COLNAME　　列名

WHENEVER　　审计时机，可选的取值为：

- ALL：所有的
- SUCCESSFUL：操作成功时
- FAIL：操作失败时

使用说明：

取消审计语句和设置审计语句进行匹配，只有完全匹配的才可以取消审计，否则无法取消审计。

例 1 取消对 SYSDBA 对表 PERSON.ADDRESS 进行的添加和修改的成功操作的审计。

```
SP_NOAUDIT_OBJECT('INSERT', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');
SP_NOAUDIT_OBJECT('UPDATE', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');
```

例 2 取消对 SYSDBA 对表 PERSON.ADDRESS 的 ADDRESS1 列进行的修改成功操作的审计。

```
SP_NOAUDIT_OBJECT('UPDATE','SYSDBA','PERSON','ADDRESS','ADDRESS1','SUCCESSFUL
');
```

5.2.3 语句序列审计

DM 还提供了语句序列审计功能，作为语句级审计和对象级审计的补充。语句序列审计需要审计员预先建立一个审计规则，包含 N 条 SQL 语句（SQL1,SQL2……），如果某个会话依次执行了这些 SQL 语句，就会触发审计。

建立语句序列审计规则的过程包括下面三个系统过程。

```
VOID
```

```
SP_AUDIT_SQLSEQ_START(
       NAME   VARCHAR (128)
)

VOID
SP_AUDIT_SQLSEQ_ADD(
       NAME   VARCHAR (128),
       SQL    VARCHAR (8188)
)

VOID
SP_AUDIT_SQLSEQ_END(
       NAME   VARCHAR (128)
)
```

参数说明：

NAME　　语句序列审计规则名

SQL　　需要审计的语句序列中的 SQL 语句

使用说明：

建立语句序列审计规则需要先调用 SP_AUDIT_SQLSEQ_START，之后调用若干次 SP_AUDIT_SQLSEQ_ADD，每次加入一条 SQL 语句，审计规则中的 SQL 语句顺序根据加入 SQL 语句的顺序确定，最后调用 SP_AUDIT_SQLSEQ_END 完成规则的建立。

例如，建立一个语句序列审计规则 AUDIT_SQL1。

```
SP_AUDIT_SQLSEQ_START('AUDIT_SQL1');
SP_AUDIT_SQLSEQ_ADD('AUDIT_SQL1', 'SELECT NAME FROM TEST1;');
SP_AUDIT_SQLSEQ_ADD('AUDIT_SQL1', 'SELECT ID FROM TEST2;');
SP_AUDIT_SQLSEQ_ADD('AUDIT_SQL1', 'SELECT * FROM TEST3;');
SP_AUDIT_SQLSEQ_END('AUDIT_SQL1');
```

可使用下面的系统过程删除指定的语句序列审计规则。

```
VOID
SP_AUDIT_SQLSEQ_DEL(
       NAME   VARCHAR (128)
)
```

参数说明：

NAME　　语句序列审计规则名

例如，删除语句序列审计规则 AUDIT_SQL1。

```
SP_AUDIT_SQLSEQ_DEL('AUDIT_SQL1');
```

5.2.4 关于审计设置的一些说明

- 只要审计功能被启用，系统级的审计记录就会产生；
- 在进行数据库审计时，审计员之间没有区别，可以审计所有数据库对象，也可取消其他审计员的审计设置；
- 语句级审计不针对特定的对象，只针对用户；
- 对象级审计针对指定的用户与指定的对象进行审计；
- 在设置审计时，审计选项不区分包含关系，都可以设置；
- 在设置审计时，审计时机不区分包含关系，都可以进行设置；
- 如果用户执行的一条语句与设置的若干审计项都匹配，只会在审计文件中生成一条审计记录。

5.3 审计文件管理

DM 审计信息存储在审计文件中。审计文件默认存放在数据库的 SYSTEM_PATH 指定的路径，即数据库所在路径。用户也可在 dm.ini 文件中添加参数 AUD_PATH 来指定审计文件的存放路径，例如：

```
#file location of dm.ctl
CTL_PATH      = D:\dmdbms\data\DAMENG\dm.ctl    #ctl file path
CTL_BAK_PATH  =D:\dmdbms\data\DAMENG\ctl_bak    #dm.ctl backup path
CTL_BAK_NUM   = 10    #backup number of dm.ctl, allowed to keep one more backup
file besides specified number.
```

```
SYSTEM_PATH   = D:\dmdbms\data\DAMENG        #system path
CONFIG_PATH   = D:\dmdbms\data\DAMENG        #config path
TEMP_PATH     = D:\dmdbms\data\DAMENG       #temporary file path
BAK_PATH      = D:\dmdbms\data\DAMENG\bak   #backup file path
DFS_PATH      =                      #path of db_file in dfs
AUD_PATH      =D:\dmdbms\data\DAMENG\aud
```

审计文件命名格式为"AUDIT_GUID_创建时间.log"，其中"GUID"为 DM 给定的一个唯一值。

审计文件的大小可以通过 DM 的 INI 参数 AUDIT_MAX_FILE_SIZE 指定。当单个审计文件超过指定大小时，系统会自动切换审计文件，自动创建新的审计文件，审计记录将写入新的审计文件中。AUDIT_MAX_FILE_SIZE 为动态系统级参数，缺省值为 100M，DBA 用户可通过系统过程 SP_SET_PARA_VALUE 对其进行动态修改，有效值范围为 1~4096M。

随着系统的运行，审计记录将会不断增加，审计文件需要更多的磁盘空间。在极限情况下，审计记录可能会因为磁盘空间不足而无法写入审计文件，最终导致系统无法正常运行。对这种情况的处理有两种策略，通过设置 DM 的 INI 参数 AUDIT_FILE_FULL_MODE 进行配置。当将 AUDIT_FILE_FULL_MODE 置为 1 时，将删除最老的审计文件，直至有足够的空间创建新审计文件。若将所有可以删除的审计文件都删除后空间仍旧不够，则数据库会挂起不再处理任何请求，直至磁盘空间被清理出足够创建新审计文件的空间；当将 AUDIT_FILE_FULL_MODE 置为 2 时，将不再写审计记录，缺省值为 1。AUDIT_FILE_FULL_MODE 为静态参数，可通过系统过程 SP_SET_PARA_VALUE 进行修改，但是修改需要重新启动 DM 数据库服务器才能生效。


注意：

两种策略都会导致审计记录的缺失，因此，管理员应该及时对审计文件进行备份。

若系统审计人员已对历史审计信息进行了充分分析，不再需要某个时间点之前的历史审计信息，可使用下面的系统过程删除指定时间点之前的审计文件，但是不会删除 DM 当前正在使用的审计文件。

```
VOID
SP_DROP_AUDIT_FILE(
      TIME_STR      VARCHAR(128),
```

```
    TYPE          INT
);
```

参数说明：

TIME_STR：指定的时间字符串

TYPE：审计文件类型，0 表示删除普通审计文件，1 表示删除实时审计文件

例如，指定删除 2015-12-6 16:30:00 以前的普通审计文件。

```
SP_DROP_AUDIT_FILE('2015-12-6 16:30:00',0);
```

DM 审计文件支持文件加密，审计管理员可使用下面的系统过程设置审计文件加密：

```
VOID
SP_AUDIT_SET_ENC(
    NAME          VARCHAR(128),
    KEY           VARCHAR(128)
)
```

参数说明：

NAME　　加密算法名，可使用 DM 支持的加密算法，见表 7.1；也支持用户自定义加密算法，见第 8 章加密引擎介绍

KEY　　加密密钥

5.4 审计信息查阅

当使用 DM 提供的审计机制进行了审计设置后，这些审计设置信息都记录在数据字典表 SYSAUDITOR.SYSAUDIT 中，结构如下表所示。审计类型用户可以查看此数据字典表查询审计设置信息。

表 5.4 SYSAUDITOR.SYSAUDIT 表结构

序号	列	数据类型	说明
1	LEVEL	SMALLINT	审计级别
2	UID	INTEGER	用户 ID
3	TVPID	INTEGER	表/视图/触发器/存储过程函数 ID
4	COLID	SMALLINT	列 ID
5	TYPE	SMALLINT	审计类型

6	WHENEVER	SMALLINT	审计情况

只要 DM 系统处于审计活动状态，系统按审计设置进行审计活动，并将审计信息写入审计文件。审计记录内容包括操作者的用户名、所在站点、所进行的操作、操作的对象、操作时间、当前审计条件等。审计用户可以通过动态视图 SYSAUDITOR.V$AUDITRECORDS 查询系统默认路径下的审计文件的审计记录，动态视图的结构如下表所示。

表 5.5 SYSAUDITOR.V$AUDITRECORDS 结构

序号	列	数据类型	说明
1	USERID	INTEGER	用户 ID
2	USERNAME	VARCHAR(128)	用户名
3	ROLEID	INTEGER	角色 ID。 没有具体角色的用户和 SQL 序列审计，没用角色信息。
4	ROLENAME	VARCHAR(128)	角色名。 没有具体角色的用户和 SQL 序列审计，没用角色信息。
5	IP	VARCHAR(25)	IP 地址
6	SCHID	INTEGER	模式 ID
7	SCHNAME	VARCHAR(128)	模式名
8	OBJID	INTEGER	对象 ID
9	OBJNAME	VARCHAR(128)	对象名
10	OPERATION	VARCHAR(128)	操作类型名
11	SUCC_FLAG	CHAR(1)	成功标记
12	SQL_TEXT	VARCHAR(8188)	SQL 文本
13	DESCRIPTION	VARCHAR(8188)	描述信息
14	OPTIME	DATETIME	操作时间
15	MAC	VARCHAR(25)	操作对应的 MAC 地址
16	SEQNO	TINYINT	DMDSC 环境下表示生成审计记录的节点号，非 DMDSC 环境下始终 0
17	BIND_INFO	VARCHAR(8188)	绑定参数具体值，如果绑定参数是复杂类型，仅记录类型名，不记录具体值

例如，SYSAUDITOR 进行如下的审计设置：

```
//SYSAUDITOR/SYSAUDITOR

SP_AUDIT_OBJECT('INSERT', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');

SP_AUDIT_OBJECT('UPDATE', 'SYSDBA', 'PERSON', 'ADDRESS', 'SUCCESSFUL');
```

之后查询 SYSAUDITOR.SYSAUDIT 数据字典表，可以看到对应的审计设置记录：

```
SELECT * FROM SYSAUDITOR.SYSAUDIT;
行号        LEVEL       UID          TVPID       COLID       TYPE  WHENEVER
---------- ----------- ----------- ----------- ----------- -----------


1           2           50331649    1196        -1          50          1


2           2           50331649    1196        -1          53          1
```

以 SYSDBA 登录，对表 PERSON.ADDRESS 进行插入和删除操作：

```
// SYSDBA/SYSDBA
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE)
VALUES('AAA','','BBB','111111');
DELETE FROM PERSON.ADDRESS WHERE ADDRESS1='AAA';
COMMIT;
```

再以 SYSAUDITOR 登录，查询 SYSAUDITOR.V$AUDITRECORDS，可以看到 SYSDBA 刚才的插入操作生成的审计记录：

```
// SYSAUDITOR/SYSAUDITOR
SELECT * FROM SYSAUDITOR.V$AUDITRECORDS WHERE USERNAME='SYSDBA';
行号        USERID      USERNAME ROLEID      ROLENAME IP  SCHID       SCHNAME
---------- ----------- -------- ----------- -------- --- ----------- -------
            OBJID       OBJNAME OPERATION SUCC_FLAG
            ----------- ------- --------- ---------
            SQL_TEXT
            ------------------------------------------------------------------
            DESCRIBTION OPTIME                      MAC               SEQNO
            ----------- --------------------------- ----------------- ----
            BIND_INFO
            ---------
1           50331649    SYSDBA   67108864    DBA      ::1 150995945   PERSON
            1196        ADDRESS INSERT    Y
```

```
          INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE)
VALUES('AAA','','BBB','111111');
                       2016-03-15 15:07:49.000000E0-3F-49-AE-86-5F 0
          NULL
```

由于并没有相关的审计设置，SYSDBA 之前执行的 DELETE 操作没有生成审计记录。

DM 还提供了以下系统函数供查看审计设置中涉及的审计类型 TYPE、审计级别 LEVEL、以及生成时机 WHENEVER 常量对应的具体信息，作为上述动态视图的补充。

- **SF_GET_AUDIT_TYPENAME**

```
VARCHAR
SF_GET_AUDIT_TYPENAME(
      TYPE  INT
);
```

参数说明：

TYPE　　审计类型，对应 SYSAUDIT 中 TYPE 字段值

返回值：

参数 TYPE 值的意义描述

例如，查看 SYSAUDIT 中 TYPE 值 50 的意义描述。

```
SELECT SF_GET_AUDIT_TYPENAME(50);
```

返回值为：INSERT

- **SF_GET_AUDIT_LEVELNAME**

```
VARCHAR
SF_GET_AUDIT_LEVELNAME(
      LEVEL  INT
);
```

参数说明：

LEVEL　　审计级别，对应 SYSAUDIT 中 LEVEL 字段值

返回值：

参数 LEVEL 值的意义描述

例如，查看 SYSAUDIT 中 LEVEL 值为 2 的意义描述。

```
SELECT SF_GET_AUDIT_LEVELNAME(2);
```

返回值为：OBJECT

- **SF_GET_AUDIT_WHENEVERNAME**

```
VARCHAR
SF_GET_AUDIT_WHENEVERNAME(
      WHENEVER INT
);
```

参数说明：

WHENEVER　　审计记录生成时机，对应 SYSAUDIT 中 WHENEVER 字段值

返回值：

参数 WHENEVER 值的意义描述

例如，查看 SYSAUDIT 中 WHENEVER 值为 1 的意义描述。

```
SELECT SF_GET_AUDIT_WHENEVERNAME(1);
```

返回值为：SUCCESSFUL

5.5 审计实时侵害检测

当执行 SP_SET_ENABLE_AUDIT (2);时，开启审计实时侵害检测功能。

实时侵害检测系统用于实时分析当前用户的操作，并查找与该操作相匹配的实时审计分析规则，如果规则存在，则判断该用户的行为是否是侵害行为，确定侵害等级，并根据侵害等级采取相应的响应措施。

5.5.1 创建与删除实时侵害检测规则

当执行 SP_SET_ENABLE_AUDIT (2);时，具有 AUDIT DATABASE 权限的用户可以使用下面的系统过程创建实时侵害检测规则。

```
VOID
SP_CREATE_AUDIT_RULE(
```

```
    RULENAME      VARCHAR(128),
    OPERATION     VARCHAR(30),
    USERNAME      VARCHAR(128),
    SCHNAME       VARCHAR(128),
    OBJNAME       VARCHAR(128),
    WHENEVER      VARCHAR(20),
    ALLOW_IP      VARCHAR(1024),
    ALLOW_DT      VARCHAR(1024),
    INTERVAL      INTEGER,
    TIMES         INTERGER
);
```

参数说明：

RULENAME　　创建的审计实时侵害检测规则名

OPERATION　　审计操作名，取值可选项下表

OPERATION 可选项
CREATE USER
DROP USER
ALTER USER
CREATE ROLE
DROP ROLE
CREATE TABLESPACE
DROP TABLESPACE
ALTER_TABLESPACE
CREATE SCHEMA
DROP SCHEMA
SET SCHEMA
CREATE TABLE
DROP TABLE
TRUNCATE TABLE
CREATE VIEW
ALTER VIEW
DROP VIEW
ALTER VIEW
CREATE INDEX
DROP INDEX

CREATE PROCEDURE
ALTER PROCEDURE
DROP PROCEDURE
CREATE TRIGGER
DROP TRIGGER
ALTER TRIGGER
CREATE SEQUENCE
DROP SEQUENCE
CREATE CONTEXT INDEX
DROP CONTEXT INDEX
ALTER CONTEXT INDEX
GRANT
REVOKE
AUDIT
NOAUDIT
CHECKPOINT
CREATE POLICY
DROP POLICY
ALTER POLICY
CREATE LEVEL
ALTER LEVEL
DROP LEVEL
CREATE COMPARTMENT
DROP COMPARTMENT
ALTER COMPARTMENT
CREATE GROUP
DROP GROUP
ALTER GROUP
ALTER GROUP PARENT
CREATE LABEL
DROP LABEL
ALTER LABEL
REMOVE TABLE POLICY
APPLY TABLE POLICY
USER SET LEVELS
USER SET COMPARTMENTS
USER SET GROUPS
USER SET PRIVS
USER REMOVE POLICY
SESSION LABEL

SESSION ROW LABEL
RESTORE DEFAULT LABELS
SAVE DEFAULT LABELS
SET TRX ISOLATION
SET TRX READ WRITE
CREATE PACKAGE
DROP PACKAGE
CREATE PACKAGE BODY
DROP PACKAGE BODY
CREATE SYNONYM
DROP SYNONYM
INSERT
SELECT
DELETE
UPDATE
EXECUTE
EXECUTE TRIGGER
MERGE INTO
LOCK TABLE
UNLOCK USER
ALTER DATABASE
SAVEPOINT
COMMIT
ROLLBACK
CONNECT
DISCONNECT
EXPLAIN
STARTUP
SHUTDOWN

USERNAME 用户名，没有指定或'NULL'表示所有用户

SCHNAME 模式名，没有时指定为'NULL'

OBJNAME 对象名，没有时指定为'NULL'

WHENEVER 审计时机，取值 ALL/SUCCESSFUL/FAIL

ALLOW_IP IP 列表，以’,’隔开。例如'"192.168.0.1","127.0.0.1"'

ALLOW_DT 时间串，格式如下：

```
ALLOW_DT::=  <时间段项>{,<时间段项>}
<时间段项> ::= <具体时间段> | <规则时间段>
```

```
<具体时间段> ::= <具体日期><具体时间> TO <具体日期><具体时间>
<规则时间段> ::= <规则时间标志><具体时间> TO <规则时间标志><具体时间>
<规则时间标志> ::= MON | TUE | WED | THURS | FRI | SAT | SUN
```

INTERVAL　　时间间隔，单位为分钟

TIMES　　次数

例 1 创建一个审计实时侵害检测规则 DANGEROUS_SESSION，该规则检测每个星期一 8：00 至 9：00 的所有非本地 SYSDBA 的登录动作。

```
SP_CREATE_AUDIT_RULE ('DANGEROUS_SESSION','CONNECT', 'SYSDBA', 'NULL', 'NULL',
'ALL', '"127.0.0.1"','MON "8:00:00" TO MON "9:00:00"',0, 0);
```

例 2 创建一个审计实时侵害检测规则 PWD_CRACK，该规则检测可能的口令暴力破解行为。

```
SP_CREATE_AUDIT_RULE ('PWD_CRACK','CONNECT', 'NULL', 'NULL', 'NULL', 'FAIL',
'NULL','NULL',1, 50);
```

当不再需要某个实时侵害检测规则时，可使用下面的系统过程进行删除。

```
VOID
SP_DROP_AUDIT_RULE(
       RULENAME      VARCHAR(128)
);
```

参数说明：

RULENAME　　待删除的审计实时侵害检测规则名

例如，删除已创建的实时侵害检测规则 DANGEROUS_SESSION。

```
SP_DROP_AUDIT_RULE ('DANGEROUS_SESSION');
```

5.5.2 实时侵害检测

当执行 SP_SET_ENABLE_AUDIT (2);时，实时侵害检测系统会实时分析用户的操作，将其与系统中创建的实时侵害检测规则进行匹配，判断该用户操作是否是侵害行为。

为了界定不同侵害行为对系统的危害情况，实时侵害检测系统定义了四种级别的侵害等级，从四级到一级侵害行为严重程度递增。这样根据用户操作匹配的实时侵害检测规则来可以判断其属于哪一级的侵害行为。

四种侵害等级如下：

- 四级：操作只违反了 IP 或者时间段设置之一，且对应累计次数在规定时间间隔内没有达到门限值，或者没有门限值设置；
- 三级：操作同时违反了 IP 和时间段设置，且累计次数在规定时间间隔内没有达到门限值，或者没有门限值设置；
- 二级：操作的 IP 和时间段都正常，且累计次数在规定时间间隔内达到了门限值；
- 一级：操作只违反了 IP 或时间段设置之一，且对应累计次数在规定时间间隔内达到了门限值，或者操作同时违反了 IP 和时间段设置，且累计次数在规定时间间隔内达到了门限值。

实时侵害检测规则的设置，使得某些操作可能会触发多条审计分析规则，也就是说可能存在多条审计分析规则同时匹配的情况。在这种情况下，需要把操作记录保存到所有匹配的、同时规定了操作频率门限值的审计分析规则下，并且使用这些审计分析规则对该操作进行分析，从所有满足的侵害等级中选择一个最高的侵害等级进行响应。

DM 实时侵害检测系统根据侵害检测结果做出相应的安全审计响应，由低到高四个等级分别为：

- 四级响应：实时报警生成，对四级侵害行为进行响应。当系统检测到四级侵害行为时，生成报警信息；
- 三级响应：违例进程终止，对三级侵害行为进行响应。当系统检测到三级侵害行为时，终止当前操作（用户当前连接仍然保持）。同时生成报警信息；
- 二级响应：服务取消，对二级侵害行为进行响应。当系统检测到二级侵害行为时，强制断开用户当前连接，退出登录。同时生成报警信息；
- 一级响应：账号锁定或失效，对一级侵害行为进行响应。当系统检测到一级侵害行为时，强制断开用户当前连接，退出登录，并且锁定账号或使账号失效。同时生成报警信息。

这些安全审计相应动作中除了生成报警信息，其余的都由 DM 数据库服务器即可完成。生成报警信息的实现包括 DM 数据库服务器将报警信息写入一个专门的日志文件，由 DM 的

审计告警工具 dmamon 实时分析这个日志文件，发现报警信息并将报警信息以邮件的形式发送给指定的邮箱。

5.5.3 审计告警工具 dmamon

审计告警工具 dmamon 用来在 DM 实时侵害检测系统检测到侵害事件且需要进行报警时可将报警信息以邮件的形式发送给指定邮箱。

dmamon 的运行需要指定对应的配置文件 dmamon.ini，此配置文件只能通过工具 dmamon_ctl 进行创建或修改。

5.5.3.1 创建或修改 dmamon.ini

使用 dmamon_ctl 工具创建或修改 dmamon.ini。

启动 dmamon_ctl 工具的命令格式为：

```
dmamon_ctl  [FILE =value]
或
dmamon_ctl  [FILE =value]  DCR_INI= value
```

参数说明：

FILE：用来指定需要编辑的 dmamon.ini 的路径。创建 dmamon.ini 时，可以不指定 FILE 参数。

DCR_INI：在 ASM 环境中，用来指定 dmdcr.ini 路径。非 ASM 环境不用指定。

例如：

```
dmamon_ctl  FILE=c:\dmamon.ini
```

当需要创建 dmamon.ini 时，可以不指定 FILE 参数，则 dmamon_ctl 工具会一步步引导用户配置 dm.ini 所在目录、发送和接收邮件的电子邮箱等信息，创建一个新的 dmamon.ini 文件。配置项如下表所示。

表 5.6 dmamon.ini 配置项

项目	项目意义	字段	字段意义
[PATH]	实时审计警报记录文件所在的文件目录	PATH	实时审计警报记录文件所在的文件目录
[SENDER]	发件人	EMAIL	发件人邮箱

		SMTP	发件人邮箱 SMTP 服务器
		USERNAME	发件人用户名
		PASSWORD	发件人密码
[RECEIVER]	收件人	EMAIL	收件人邮箱
[INTERVAL]	时间间隔	INTERVAL	检查审计警报记录文件的时间间隔，单位秒

dmamon_ctl 工具支持的命令如下表。

表 5.7 dmamon_ctl 命令

命令	说明
EDIT	修改 INI 文件信息
ADD	增加接收者的邮件地址信息
REMOVE	删除接收者的邮件地址信息
SHOW	显示当前控制文件的内容
SAVE	保存，保存路径必须为 dmamon.ini 的绝对路径。退出之前须保存，否则修改无效
EXIT	退出当前程序
HELP	显示帮助信息

5.5.3.2 使用 dmamon

配置好 dmamon.ini 后，就可以使用 dmamon 工具进行实时审计告警了。启动 dmamon 工具的命令格式为：

```
dmamon  PATH=<dmamon.ini 路径>  USERID=<userid>
或
dmamon  PATH=<dmamon.ini 路径>  USERID=<userid>  dcr_ini=<dmdcr.ini 路径>
<userid>::=<username>/<password> [@<connect_identifier>][<option>]
<connect_identifier> ::=<svc_name> | {<host>[:<port>]} | <unixsocket_file>
<option>::= #{ <exetend_option>=<value>{,<extend_option>=<value>} }  //此行外层{}是为了封装参数之用，书写时需要保留
```

参数说明：

PATH：指定 dmamon.ini 文件的路径，字符串类型；

USERID：指定登录数据库的信息；

<username>/<password>：指定用户名和密码，仅支持具有审计权限的用户使用 dmamon 工具。

<svc_name>：服务名。

<host>[:<port>]：服务器 IP 地址和端口号。缺省情况下默认为本地服务器和端口号 LOCALHOST:5236。当服务器为本机时，SERVER:PORT 可直接写 LOCALHOST。当连接其他服务器时，SERVER:PORT 需写上 IP 地址和 PORTNUM，例如：192.168.0.248:8888。

<unixsocket_file>：专门用于在 LINUX 系统中，当服务器与客户端之间使用 UNIXSOCETUNIX-IPC 方式通信时，指定客户端连接的 UNIXSOCKET 路径文件名。必须和 inet_type=UNIXSOCKET 同时使用。例如：

```
./dmamon PATH=/home/dmamon.ini
USERID=SYSAUDITOR/SYSAUDITOR@/home/test/foo.sock#{inet_type=UNIXSOCKET}
```

<option>为扩展选项，用法为<exetend_option>=<value>。所有 value 值不能包含空格，不能包含特殊的符号，如引号等。书写扩展选项时需要用引号#"{ }"进行封装，例如：#"{INET_TYPE=tcp,mpp_type=local}"。

现支持的扩展选项如下：

extend_option	value
mpp_type	MPP 登录属性，此属性的设置对非 MPP 系统没有影响。取值 GLOBAL 和 LOCAL，默认为 GLOBAL。GLOBAL 表示 MPP 环境下建立的会话为全局会话，对数据库的导入导出操作在所有节点进行；LOCAL 表示 MPP 环境下建立的会话为本地会话，对数据库的导入导出操作只在本地节点进行
inet_type	网络通信协议类型。取值 UDP/TCP/IPC/RDMA/UNIXSOCKET，分别对应 UDP 协议、TCP 协议、IPC（共享内存）、RDMA（远程直接内存访问）、UNIXSOCKET(unix domain socket - IPC)协议。缺省为 TCP
ssl_path	通信加密的 SSL 数字证书路径，缺省为不使用加密。数字证书路径由用户自己创建，将相应的证书需放入该文件夹中。其中服务器证书必须与 dmserver 目录同级，客户端目录可以任意设置。和 ssl_pwd 一起使用。 各用户只能使用自己的 SSL 数字证书，例如 SYSAUDITOR 账户只能使用 /home/dmdbms/bin/client_ssl/SYSAUDITOR 下的证书和密码，如果证书没有密码可以用缺省或任意数字代替。 例如：./dmamon PATH=/home/dmamon.ini USERID=SYSAUDITOR/SYSAUDITOR@192.168.1.64:5236#"{ssl_path= /home/dmdbms/bin/client_ssl/SYSAUDITOR,ssl_pwd=12345}"
ssl_pwd	通信加密的 SSL 数字证书密码。和 ssl_path 一起使用。缺省为不加密

DCR_INI：在 ASM 环境，用来指定 dmdcr.ini 路径。非 ASM 环境不用指定。

5.6 审计分析

DM 提供了图形界面的审计分析工具 Analyzer 以及通过命令行启动的审计分析工具 dmaudtool。

Analyzer 实现对审计记录的分析功能，能够根据所制定的分析规则，对审计记录进行分析，判断系统中是否存在对系统安全构成危险的活动。只有审计用户才能使用审计分析工具 Analyzer。

dmaudtool 实现对审计记录的解析、筛选以及导出功能。审计文件的内容是不可视的，dmaudtool 可解析审计文件中的审计记录并导出至指定输出文件当中以供用户查看。同时 dmaudtool 也提供筛选功能，可以针对审计记录中的用户名、模式名、对象名或者操作时间有选择性地筛选导出。审计分析工具 dmaudtool 的使用对用户无限制。

下面将详细介绍审计分析工具 Analyzer 和 dmaudtool 的使用方法。

5.6.1 审计分析工具 Analyzer

审计用户登录审计分析工具后，通过 Analyzer 可以创建和删除审计规则，并可以指定对某些审计文件应用某些规则。并将审计结果以表格的方式展现出来。

审计用户登录 Analyzer 后可看到工具主界面如图 5.1 所示。

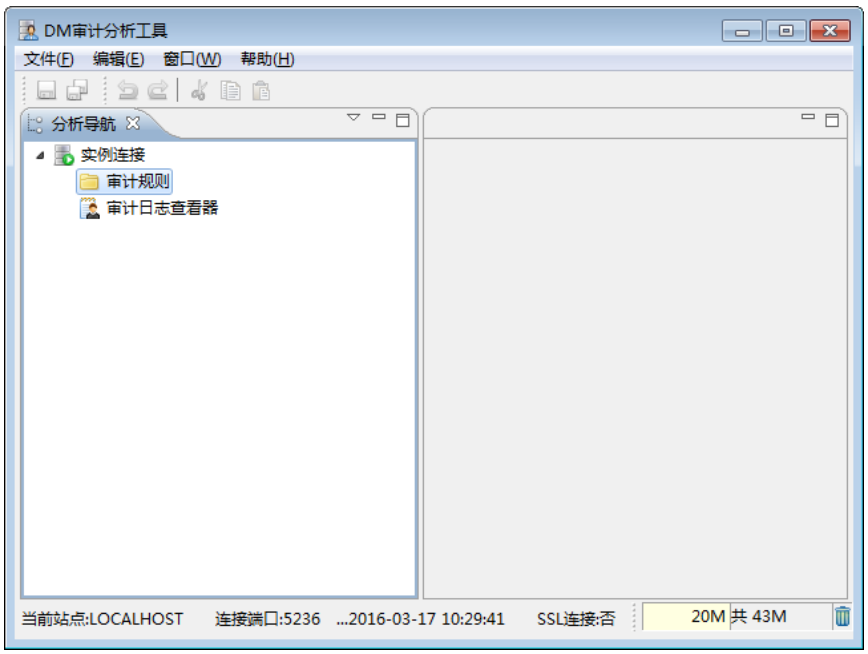

图 5.1 Analyzer 工具主界面

5.6.1.1 审计规则

右键单击导航树中的“审计规则”节点，在弹出的菜单中选择“新建审计分析规则”，可打开如下图所示的新建审计分析规则窗口。

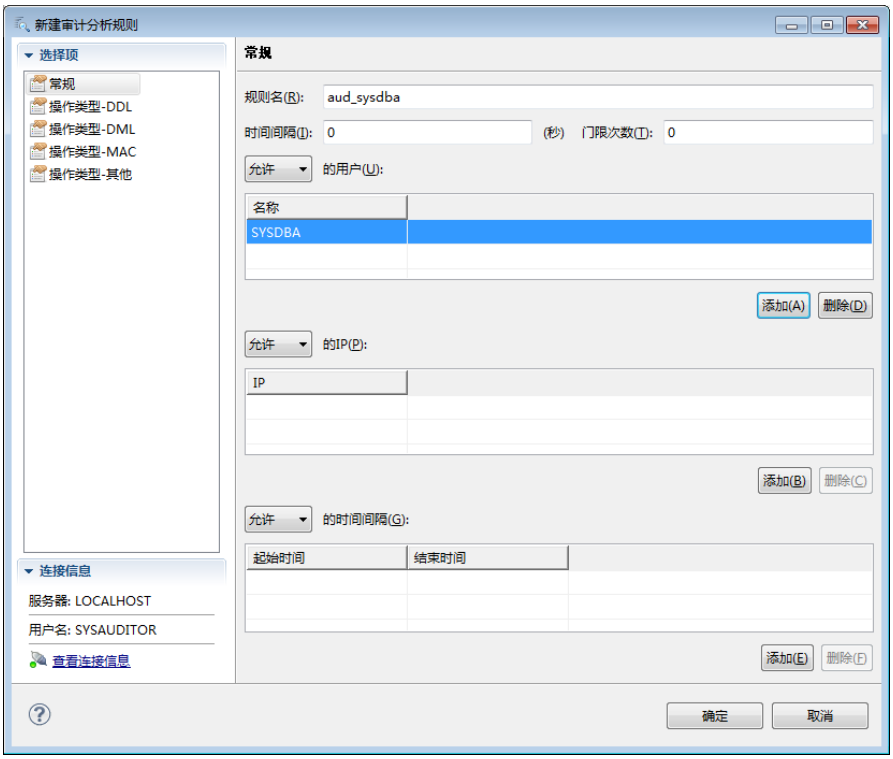

图 5.2 新建审计分析规则窗口

如上图中创建了一个名为 aud_sysdba 的审计分析规则，对 SYSDBA 的所有审计记录进行分析。之后我们就可以将这个审计分析规则应用于对审计记录文件的分析。右键点击“审计规则”或一个具体的审计分析规则节点，在弹出的菜单中选择“审计规则分析”，弹出如下图所示审计规则分析窗口。

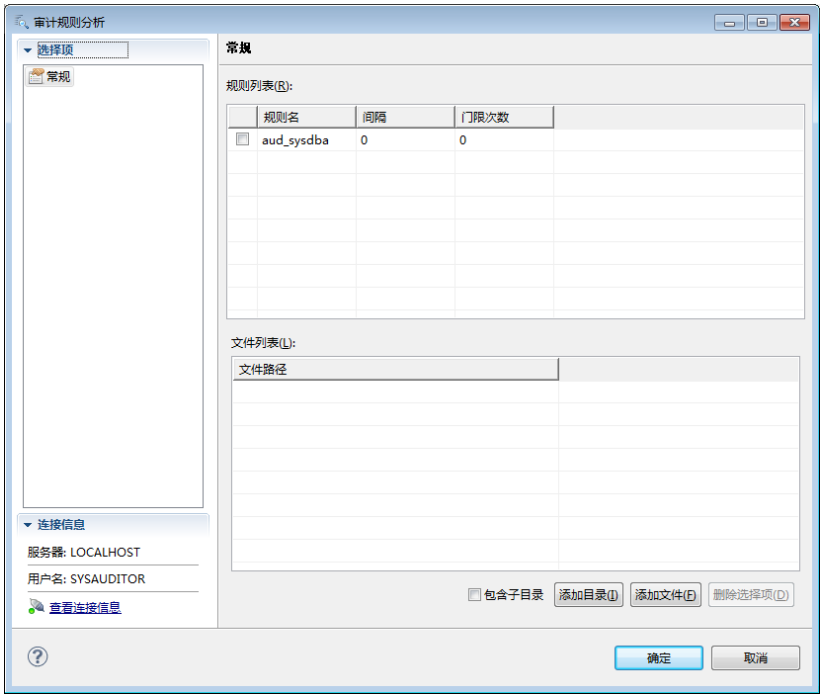

图 5.3 审计规则分析窗口

选择需要应用的审计分析规则，可以同时选择多条，在下面的文件列表中添加需要进行分析的审计记录文件，Analyzer 工具就会根据审计分析规则对文件中的审计记录进行分析，将满足规则的审计记录以表格的形式显示出来，如下图所示。

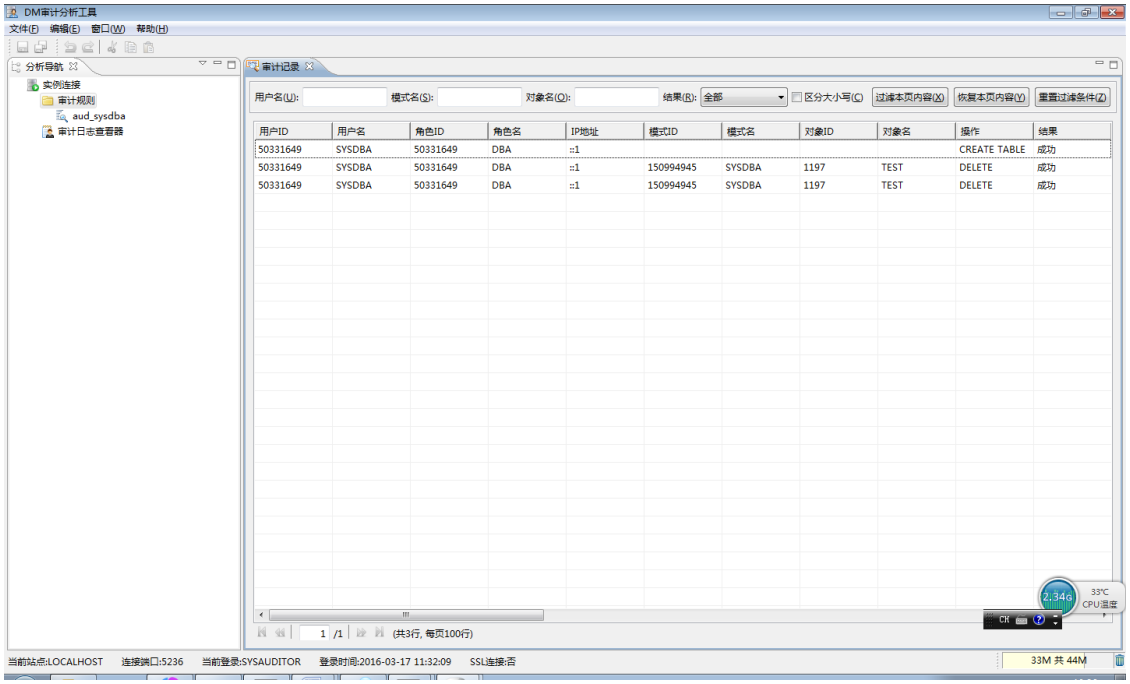

图 5.4 审计规则分析结果

在审计规则分析结果显示窗口中，还可以对显示的审计记录根据需要进行进一步过滤，进一步分析出需要的审计记录。

5.6.1.2 审计日志查看器

双击“审计日志查看器”节点，可打开如下图所示的条件设置窗口。

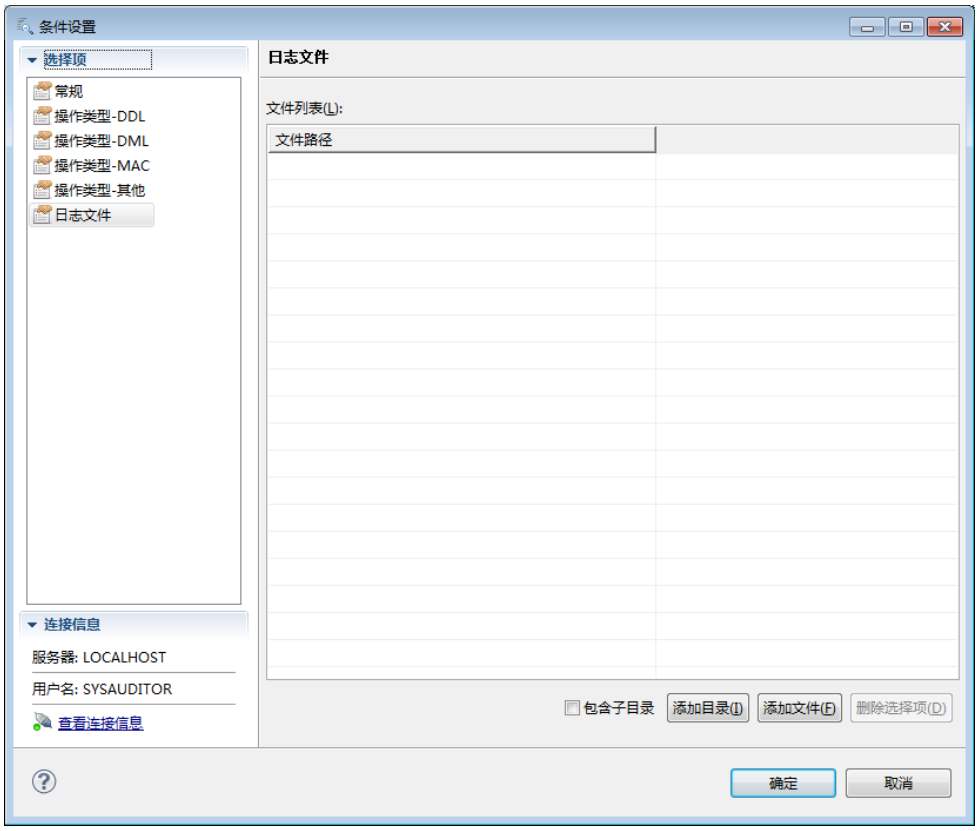

图 5.5 审计日志查看条件设置窗口

添加需要查看的审计记录文件，还可在窗口左侧的“选择项”中设置各种过滤条件，之后点击“确定”，将满足规则的审计记录以表格的形式显示出来，与图 5.5 的显示相同。

5.6.2 审计分析工具 dmaudtool

审计信息以审计记录的形式存放在 DM 数据库审计文件中，关于审计文件的存放路径等信息请参见 5.3 审计文件管理。审计文件的内容是不可视的，审计分析工具 dmaudtool 可以筛选解析审计文件中的审计记录并且导出至指定输出文件当中以供用户查看。审计分析工具 dmaudtool 的使用对用户无限制。

5.6.2.1 使用 dmaudtool 工具

dmaudtool 工具需要从命令行启动。用户需要在 cmd 命令行工具中找到 DM 安装目录的 bin 文件夹/dmdbms/bin，输入 dmaudtool 和参数后回车。参数在下一节详细介绍。

语法如下：

```
dmaudtool PARAMETER=<value> { PARAMETER=<value> }
```

PARAMETER：dmaudtool 参数。多个参数之间排列顺序无影响，参数之间使用空格间隔。

<value>：参数取值。

例如：分析用户名为 SYSDBA，密码为 123456789，IP 地址为 192.168.1.60，端口号为 5236 的数据库的审计文件，并将分析结果输出保存。审计文件所在路径为 E:\dmdbms\data\DAMENG 。分析结果输出路径为 E:\dmdbms\data\DAMENG\out\aud_out.txt，若已经存在，则报错。行间隔符设为 AAAA，输出文件大小不限。

分析本地磁盘上的审计文件时，要求 dmaudtool 与审计文件在同一台主机上。命令行操作如下：

```
dmaudtool userid=SYSDBA/123456789@192.168.1.60:5236

afil_path=E:\dmdbms\data\DAMENG out_path=E:\dmdbms\data\DAMENG\out\aud_out

R_SEP=AAAA
```

显示如下：

```
E:\dmdbms\bin>dmaudtool userid=SYSDBA/123456789@192.168.1.60:5236 afil_path=E:\dmdbms\data\DAMENG
out_path=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA
dmaudtool V8
Start to audit files in [E:\dmdbms\data\DAMENG]....
aud sys init success.
start file [E:\dmdbms\data\DAMENG\AUDIT_DMSERVER_BBCC122DDC054F47B4906F970245F606_2020-9-7-17-22-5
2.log]....
audit files time used: 10.379(ms)
Analyse the audit files successfully.
```

图 5.6 使用 dmaudtool 工具示例

存在同名文件时会报错：

```
E:\dmdbms\bin>dmaudtool userid=SYSDBA/123456789@192.168.1.60:5236 afil_path=E:\dmdbms\data\DAMENG
out_path=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA
dmaudtool V8
Start to audit files in [E:\dmdbms\data\DAMENG]....
aud sys init success.
输出文件[E:\dmdbms\data\DAMENG\out\aud_out.txt]已经存在
Analyse the audit files failed.
```

图 5.7 分析审计记录时存在同名文件

5.6.2.2 dmaudtool 参数一览表

本节提供 dmaudtool 的参数一览表，供用户快速参考。

表 5.8 dmaudtool 参数一览表

参数	含义	备注
USERID	用户名和口令。 USERID=<userid> <userid>::={{<username>[/<password>]} | /}[@<connect_identifier>][<option>] [<os_auth>] <connect_identifier> ::=<svc_name> | {<host>[:<port>]} | <unixsocket_file> <option>::= #{ <exetend_option>=<value>{,<extend_option>=<value>} } //此行外层{}是为了封装参数之用，书写时需要保留 <os_auth>::= AS {SYSDBA|SYSSSO|SYSAUDITOR|USERS|AUTO} 更详细的 USERID 参数介绍请参考 dmamon 工具	必选。仅支持本地连接
AFIL_PATH	指定审计文件所在的路径	必选
OUT_PATH	指定审计分析结果的输出文件完整路径	必选。与 OUT_SIZE 配合使用，若文件大小超过 OUT_SIZE 指定值，则会在目录下生成同名+序号的新输出文件
OUT_SIZE	指定审计分析结果输出文件的大小，单位为兆（M）。指定范围为 500~65534（M）	可选。缺省为 0，表示不限制文件大小

DCR_INI	DCR_INI 路径，分析存储在 ASM 上的审计文件时指定	分析存储在 ASM 上的审计文件时必选，否则可选，此时系统将不考虑该参数值
USERNAME	对应审计记录中的用户名，表示收集该用户名的审计记录	可选。若未指定则不考虑该参数值，若均未指定则默认分析指定目录下所有审计文件
SCHNAME	对应审计记录中模式名，表示收集该模式名的审计记录	
OBJNAME	对应审计记录中对象名，表示收集该对象名的审计记录	
TIME_FROM	对应审计记录中操作时间，用于指定收集审计记录的起始时间	
TIME_TO	对应审计记录中操作时间，用于指定收集审计记录的结束时间	
R_SEP	指定输出文件中的行分隔符，用于间隔输出文件中行与行之间的数据	可选。缺省为回车符。考虑到记录中可能存在的字符，间隔符需要设置相对复杂，避免解析出错
C_SEP	指定输出文件中的列分割符，用于间隔输出文件中列与列之间的数据	可选。缺省为'|'
HELP	打印帮助信息	可选

5.6.2.3 dmaudtool 工具使用示例

5.6.2.3.1 帮助 HELP

输入 dmaudtool help 即可查看帮助信息。帮助信息中会显示 dmaudtool 版本信息以及所有参数的大致信息，供用户快速参考。

命令行操作如下：

```
dmaudtool HELP
```

显示如下：

```
[test@test161 release]$ ./dmaudtool help
dmaudtool V8
version: 1-2-101-21.12.16-153499-10000-ENT
格式: ./dmaudtool  KEYWORD=value

例程: ./dmaudtool  USERID=sysauditor/sysauditor@192.168.0.33:4356 AFIL_PATH=/opt/dm7data/dameng/aud OUT_PATH=/opt/dm7da
ta/dameng/out/aud_out.txt

必选参数: USERID、AFIL_PATH、OUT_PATH

关键字              说明
--------------------------------------------------------------------------------
USERID              用户名/口令 格式:<username>/<password>[@<connect_identifier>][<option>], 必选
                    <connect_identifier> : [<svc_name> | host[:port] | <unixsocket_file>]
                    <option> : #{<exetend_option>=<value>[,<extend_option>=<value>]...}
                                --此行外层{}是为了封装参数之用, 书写时需要保留
AFIL_PATH           审计文件所在的路径, 必选
OUT_PATH            输出文件的生成路径,与OUT_SIZE配合使用, 若超过指定文件大小, 则会生成同名加序号后缀的文件, 必选。
OUT_SIZE            输出文件的指定大小, 单位为兆（M）, 默认为0, 表示不限制文件大小, 最小500, 最大65534。
DCR_INI             DCR_INI路径, 用于连接ASM使用。当审计文件存在与ASM上使用, 若没有, 则忽略。
USERNAME            对应审计记录中记录的用户名, 若指定, 则说明收集满足该用户名的审计记录。若未指定, 则不考虑
SCHNAME             对应审计记录中记录的模式名, 若指定, 则说明收集满足该模式名的审计记录。若未指定, 则不考虑
OBJNAME             对应审计记录中记录的对象名, 若指定, 则说明收集满足该对象名的审计记录。若未指定, 则不考虑
TIME_FROM           对应审计记录中操作时间, 若指定, 则收集指定时间之后的审计记录。若未指定, 则不考虑
TIME_TO             对应审计记录中操作时间, 若指定, 则收集指定时间之前的审计记录。若未指定, 则不考虑
R_SEP               输出文件中行分隔符, 默认回车符
C_SEP               输出文件中列分隔符, 默认|号符
HELP                打印帮助信息
```

图 5.8 使用 dmaudtool 的 help 命令

5.6.2.3.2 分析指定目录下所有审计文件

分析用户名为 SYSDBA，密码为 123456789，IP 地址为 192.168.1.60，端口号为 5236 的数据库的审计文件，并将分析结果输出保存。审计文件所在路径为 E:\dmdbms\data\DAMENG ，分析结果输出路径为 E:\dmdbms\data\DAMENG\out\aud_out.txt，行间隔符设为 AAAA，输出文件大小不限。

命令行操作如下：

```
dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236

AFIL_PATH=E:\dmdbms\data\DAMENG

OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA
```

显示如下：

```
E:\dmdbms\bin>dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236 AFIL_PATH=E:\dmdbms\data\DAMENG
 OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA
dmaudtool V8
Start to audit files in [E:\dmdbms\data\DAMENG]....
aud sys init success.
start file [E:\dmdbms\data\DAMENG\AUDIT_DMSERVER_BBCC122DDC054F47B4906F970245F606_2020-9-7-17-22-
52.log]....
audit files time used: 8.541(ms)
Analyse the audit files successfully.
```

图 5.9 分析指定目录下所有审计文件

5.6.2.3.3 分析用户名为 SYSDBA 的审计记录

数据库参照章节 5.6.2.3.2 分析指定目录下的所有审计文件。

命令行操作如下：

```
dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236

AFIL_PATH=E:\dmdbms\data\DAMENG

OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA USERNAME=SYSDBA
```

显示如下：

```
E:\dmdbms\bin>dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236 AFIL_PATH=E:\dmdbms\data\DAMENG
 OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA USERNAME=SYSDBA
dmaudtool V8
Start to audit files in [E:\dmdbms\data\DAMENG]....
aud sys init success.
start file [E:\dmdbms\data\DAMENG\AUDIT_DMSERVER_BBCC122DDC054F47B4906F970245F606_2020-9-7-17-22-
52.log]....
audit files time used: 15.679(ms)
Analyse the audit files successfully.
```

图 5.10 分析用户名为 SYSDBA 的审计记录

5.6.2.3.4 分析指定时间之后的审计文件

数据库参照章节 5.6.2.3.2 分析指定目录下的所有审计文件。

命令行操作如下：

```
dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236

AFIL_PATH=E:\dmdbms\data\DAMENG

OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA TIME_FROM=”2020/9/1

21:00:00”
```

显示如下：

```
E:\dmdbms\bin>dmaudtool USERID=SYSDBA/123456789@192.168.1.60:5236 AFIL_PATH=E:\dmdbms\data\DAMENG
 OUT_PATH=E:\dmdbms\data\DAMENG\out\aud_out.txt R_SEP=AAAA TIME_FROM="2020/9/1 21:00:00"
dmaudtool V8
Start to audit files in [E:\dmdbms\data\DAMENG]....
aud sys init success.
start file [E:\dmdbms\data\DAMENG\AUDIT_DMSERVER_BBCC122DDC054F47B4906F970245F606_2020-9-7-17-22-
52.log]....
audit files time used: 3.406(ms)
Analyse the audit files successfully.
```

图 5.11 分析指定时间之后的审计文件

5.6.2.3.5 分析存储在 ASM 上的审计文件

分析用户名为 SYSDBA，密码为 SYSDBA，IP 地址为 192.168.100.121，端口号为 5425 的数据库节点的审计文件，并将分析结果输出保存。审计文件所在路径为 +DMDATA/home/data/yy/GRP7_DB ， 分 析 结 果 输 出 路 径 为 /home/test/yy/out/aud_out.txt ， dmdcr.ini 配 置 文 件 的 路 径 为 /home/data/yy/DSC01/dmdcr.ini（分析存储在 ASM 上的审计文件时需指定 dmdcr.ini 配置文件的路径，即配置参数 DCR_INI），行间隔符设为 AAAA，输出文件大小不限。

分析存储在 ASM 上的审计文件时无需考虑审计分析工具是否与 ASM 服务器在同一台主机上。

命令行操作如下：

```
./dmaudtool USERID=SYSDBA/SYSDBA@192.168.100.121:5425

AFIL_PATH=+DMDATA/home/data/yy OUT_PATH=/home/test/yy/out/aud_out.txt

DCR_INI=/home/data/yy/DSC01/dmdcr.ini R_SEP=AAAA
```

显示如下：

```
[test@test121 bin]$ ./dmaudtool USERID=SYSDBA/SYSDBA@192.168.100.121:5425 AFIL_PATH=+DMDATA/home/data/yy/GRP7_DB
 OUT_PATH=/home/test/yy/out/aud_out.txt DCR_INI=/home/data/yy/DSC01/dmdcr.ini R_SEP=AAAA
dmaudtool V8
Start to audit files in [+DMDATA/home/data/yy/GRP7_DB]....
aud sys init success.
start file [+DMDATA/home/data/yy/GRP7_DB/AUDIT_GRP7_RT_DSC01_9B524F049B524F04209908EB7841ABD7_2020-9-9-14-0-5.lo
g]....
audit files time used: 2.038(ms)
Analyse the audit files successfully.
[test@test121 bin]$
```

图 5.12 分析存储在 ASM 上的审计文件

5.6.2.4 审计分析结果导入到目标库中

将审计分析结果导入到目标库中需要利用 DM 自带的导入导出工具 dmfldr。将 dmaudtool 的分析结果输出文件作为 dmfldr 的输入文件，并配置与 dmaudtool 分隔符相匹配的控制文件即可完成导入工作。dmaudtool 按照动态视图 V$AUDITRECORDS 的行列顺序输出审计结果，因此需要在目标库中创建一个与动态视图 V$AUDITRECORDS 结构一样的目标表，用以存放审计记录。

以章节 5.6.2.3.4 分析指定时间之后的审计文件中的分析结果输出文件为例，目标库也使用章节 5.6.2.3.4 分析指定时间之后的审计文件中的数据库，操作步骤如下：

1. 在目标库中创建一个与动态视图 V$AUDITRECORDS 结构相同的目标表；

SQL 语句如下：

```
CREATE TABLE aud_test
(
"USERID" INTEGER,
"USERNAME" VARCHAR(128),
"ROLEID" INTEGER,
"ROLENAME" VARCHAR(128),
"IP" VARCHAR(64),
"SCHID" INTEGER,
"SCHNAME" VARCHAR(128),
"OBJID" INTEGER,
"OBJNAME" VARCHAR(128),
"OPERATION" VARCHAR(128) NOT NULL,
"SUCC_FLAG" CHAR(1) NOT NULL,
"SQL_TEXT" VARCHAR(8188),
"DESCRIBTION" VARCHAR(8188),
"OPTIME" DATETIME(6) NOT NULL,
"MAC" VARCHAR(25),
"SEQNO" BYTE,
"BIND_INFO" VARCHAR(8188)
```

```
);
```

2. 创建控制文件 out.ctrl，并存放到 e:\dmdbms\data\DAMENG 目录下；

out.ctrl 中内容如下：

```
LOAD DATA
INFILE 'E:\dmdbms\data\DAMENG\out\aud_out.txt' STR 'AAAA'
INTO TABLE aud_test
FIELDS '|'
(
"USERID",
"USERNAME",
"ROLEID",
"ROLENAME",
"IP",
"SCHID",
"SCHNAME",
"OBJID",
"OBJNAME",
"OPERATION",
"SUCC_FLAG",
"SQL_TEXT",
"DESCRIBTION",
"OPTIME",
"MAC",
"SEQNO",
"BIND_INFO"
)
```

3.打开命令行窗口，进入 DM 安装目录的 bin 目录，运行 dmfldr，将分析结果导入到目标表中。

命令行操作如下：

```
dmfldr userid=SYSDBA/123456789@192.168.1.60:5236
```

```
control=’e:\dmdbms\data\DAMENG\out.ctrl’
```


注意：

审计记录中可能存在过长的执行语句，所以在导入目标表时可能出现“记录超长”的错误，可以使用新库作为目标库，初始化数据库时调大建库参数 page_size。例如，存在一个执行语句长度为 3900 字节，可以设置 page_size=32。

6 通信加密

DM 提供两种通信加密方式：一基于传输层的 SSL 协议加密；二基于应用层的消息包加密。两个层次的加密互不影响，可以同时使用。因为任何一种加密方式都足够安全，而且双重加密会占用更多的数据库资源，所以通常情况下，没必要同时开启这两种加密。

当未采用通信加密时，为了提高安全性，数据库系统自动对登录消息中的用户名密码使用内部算法进行了简单的加解密操作，如果用户需要对用户名密码进行增强加密，请参考11 登录用户名密码增强加密。

下面对基于传输层的 SSL 协议加密和基于应用层的消息包加密进行详细介绍。

6.1 基于传输层的 SSL 协议加密

选择是否使用 SSL 协议加密以 DM 数据库服务器端的设置为准，即通过设置服务器配置文件 DM.INI 中的 ENABLE_ENCRYPT 参数来指定，客户端以服务器采用的通信方式与其进行通信。

ENABLE_ENCRYPT 取值 0、1 和 2。0：不开启 SSL 加密和 SSL 认证；1：开启 SSL 加密和 SSL 认证；2：开启 SSL 认证但不开启 SSL 加密。缺省值为 0。不论设置为 1 还是 2，它内部所使用的密码套件是 SSL 协议自动协商的，达梦未进行任何干涉。

系统管理员可通过查询 V$PARAMETER 动态视图查询 ENABLE_ENCRYPT 的当前值。

```
SELECT * FROM V$PARAMETER WHERE NAME='ENABLE_ENCRYPT';
```

也可以通过使用客户端工具 Console 或调用系统过程 SP_SET_PARA_VALUE 重新设置 ENABLE_ENCRYPT 的值。不过由于这个参数为静态 INI 参数，修改后需要重启 DM 数据库服务器才能生效。

```
SP_SET_PARA_VALUE (2,'ENABLE_ENCRYPT',1);
```

另外，DM.INI 中的 MIN_SSL_VERSION 参数指定了允许连接的 SSL 最低版本，此参数设置仅当 ENABLE_ENCRYPT 不为 0 时有效。MIN_SSL_VERSION 的可取值包括：

- ✓ 0：所有版本
- ✓ 0x0301：TLSv1
- ✓ 0x0302：TLSv1.1

- ✓ 0x0303：TLSv1.2
- ✓ 0x0304：TLSv1.3

MIN_SSL_VERSION 缺省为 0，为静态 INI 参数，修改后需要重启 DM 数据库服务器才能生效。

当选择 SSL 加密时，需要在 DM 数据库服务器所在目录下的 server_ssl 子目录中存放 CA 的证书、服务器的证书和服务器的密钥，同时在客户端所在目录下的 client_ssl 子目录中存放 CA 的证书、客户端的证书和客户端的密钥，这样服务器和客户端的通信即是建立在加密的 SSL 连接之上的。此时如果没有配置好 SSL 环境，则通讯仍旧不加密。

当选择 SSL 认证时，不进行通信加密，只是检查客户端和服务器的证书是否匹配。此时如果服务器 SSL 环境没有配置则服务器无法正常启动，如果客户端 SSL 环境没有配置则无法连接服务器。

6.2 基于应用层的消息包加密

选择是否使用基于应用层的消息包加密以 DM 数据库服务器端的设置为准，即通过设置服务器配置文件 DM.INI 中的 COMM_ENCRYPT_NAME 参数来指定，客户端以服务器采用的通信方式与其进行通信。

COMM_ENCRYPT_NAME 取值空串、算法名分别代表不加密、加密。缺省为空串。当算法名写错，则使用加密算法 DES_CFB。DM 支持的加密算法名可以通过查询动态视图 V$CIPHERS 获取。应用层加密的密钥是通过 DH 密钥交换算法协商生成。

系统管理员可通过查询 V$PARAMETER 动态视图查询 COMM_ENCRYPT_NAME 的当前值。

```
SELECT * FROM V$PARAMETER WHERE NAME='COMM_ENCRYPT_NAME';
```

也可以通过使用客户端工具 Console 或调用系统过程 SP_SET_PARA_STRING_VALUE 重新设置 COMM_ENCRYPT_NAME 的值。不过由于这个参数都为静态 INI 参数，修改后需要重启 DM 数据库服务器才能生效。

```
SP_SET_PARA_STRING_VALUE(2,'COMM_ENCRYPT_NAME','DES_OFB');
```

7 存储加密

为了防止用户直接通过数据文件获取用户信息，DM 提供了全面的数据加密的功能，包括：

- ✓ 透明加密
- ✓ 半透明加密
- ✓ 非透明加密

存储加密在保证数据文件安全性的同时，也会带来一定的性能影响，不同的加密算法对性能的影响各有不同，用户需要根据自己的需求来决定是否进行加密以及加密算法的选择。

系统内置了常用的 DES，AES，RC4，SHA 等类型的加密和散列算法，以此来保护数据的安全性。DM 支持的加密和散列算法可通过查询动态视图 V$CIPHERS 得到，表 7.1 列出了 DM 内置的加密和散列算法。

表 7.1 DM 内置的加密和散列算法

算法名称	算法类型	分组长度	密钥长度
DES_ECB	分组加密算法	8	8
DES_CBC	分组加密算法	8	8
DES_CFB	分组加密算法	8	8
DES_OFB	分组加密算法	8	8
DESEDE_ECB	分组加密算法	8	16
DESEDE_CBC	分组加密算法	8	16
DESEDE_CFB	分组加密算法	8	16
DESEDE_OFB	分组加密算法	8	16
AES128_ECB	分组加密算法	16	16
AES128_CBC	分组加密算法	16	16
AES128_CFB	分组加密算法	16	16
AES128_OFB	分组加密算法	16	16
AES192_ECB	分组加密算法	16	24
AES192_CBC	分组加密算法	16	24
AES192_CFB	分组加密算法	16	24
AES192_OFB	分组加密算法	16	24
AES256_ECB	分组加密算法	16	32
AES256_CBC	分组加密算法	16	32

AES256_CFB	分组加密算法	16	32
AES256_OFB	分组加密算法	16	32
DES_ECB_NOPAD	分组加密算法	8	8
DES_CBC_NOPAD	分组加密算法	8	8
DESEDE_ECB_NOPAD	分组加密算法	8	16
DESEDE_CBC_NOPAD	分组加密算法	8	16
AES128_ECB_NOPAD	分组加密算法	16	16
AES128_CBC_NOPAD	分组加密算法	16	16
AES192_ECB_NOPAD	分组加密算法	16	24
AES192_CBC_NOPAD	分组加密算法	16	24
AES256_ECB_NOPAD	分组加密算法	16	32
AES256_CBC_NOPAD	分组加密算法	16	32
RC4	流加密算法	-	16
MD5	散列算法	-	-
SHA1	散列算法	-	-
SHA224	散列算法	-	-
SHA256	散列算法	-	-
SHA384	散列算法	-	-
SHA512	散列算法	-	-

说明： **以“NOPAD”结尾的加密算法需要用户保证原始数据长度是分组长度的整数倍，DM 不会自动填充。如果数据不一定是分组长度的整数倍，请选择不以“NOPAD”结尾的加密算法，以“NOPAD”结尾的加密算法主要用于数据页分片加密。**

DM 安装中自带 OPENSSL 的动态库，则查询 V$CIPHERS 时除了上表中列出的算法，还会查询到 OPENSSL 动态库的相关算法。

存储加密支持的加密和散列算法除了 DM 内置的算法，还包括第三方加密和散列算法，详情请参考 8 加密引擎。

DM 使用服务器主密钥 SVR_KEY 和数据库主密钥 DB_KEY 进行分级存储加密。SVR_KEY 为服务器主密钥，用于加密数据库主密钥；DB_KEY 用于加密库内密钥，如用户加密密钥、列加密密钥等。

DM 安全版提供了两个系统过程 SP_UPDATE_SVRKEY()和 SP_UPDATE_DBKEY()，分别用于更新密钥 SVR_KEY 和 DB_KEY。

例如，更新服务器主密钥 SVR_KEY。

```
SQL> SP_UPDATE_SVRKEY();
```

7.1 透明加密

在透明加密中，密钥生成、密钥管理和加解密过程由数据库管理系统自动完成，用户不可见。透明加密的目的主要是保证存储在数据文件中的敏感数据的安全，并不能保护合法用户的个人私密数据。

7.1.1 全库加密

对 DM 数据库进行全库加密需要在初始化数据库时通过 ENCRYPT_NAME 参数指定全库加密算法，加密密钥由 DM 自动生成。若初始化数据库时不指定 ENCRYPT_NAME 参数，则不进行全库加密。

指定全库加密时，在 DM 数据库服务器启动及运行的过程中，需要对处理的所有数据页通过指定的加密算法和 DM 自动生成的密钥进行加解密处理。如果数据页读入缓存，需要进行解密后才能使用，在缓存中的数据页进行刷盘时，需要对数据页进行加密后存储到数据文件中。

系统存储函数 SF_GET_ENCRYPT_NAME() 用于获取全库加密算法名。未使用全库加密时返回 NULL。

例如，使用了全库加密算法 xor1。

```
SQL> select SF_GET_ENCRYPT_NAME();

行号        SF_GET_ENCRYPT_NAME()
---------- ---------------------
1           xor1
```

7.1.2 表空间透明加密

DM 支持在创建表空间时指定进行透明加密，语法格式为：

```
CREATE TABLESPACE <表空间名><数据文件子句>[<数据页缓冲池子句>][<存储加密子句>]

<存储加密子句> ::= ENCRYPT WITH <加密算法> [BY <加密密码>]
```

说明： **此处主要说明存储加密相关的语法，其余语法的详细描述请参考《DM8_SQL 语言使用手册》，后续小节中相同，不再说明。**

使用说明：

1． 加密密码最大长度 32 字节，若未指定，由 DM 随机生成；

2． 如果已指定全库加密，则不再支持表空间加密。

7.1.3 表列透明加密

DM 支持对表的列进行透明加密，支持建表时设置加密列，以及修改表定义时设置加密列。存储加密支持所有的列类型，包括大字段类型。用透明加密的方式加密列上的数据时，在数据库中保存加密该列的密钥，执行 DML 语句的过程中系统能自动获取密钥。

建表或修改表时指定对列进行透明加密的语法格式为：

```
……
<列定义> ::= <不同类型列定义> [<列定义子句>] [<STORAGE 子句>][<透明存储加密子句>]
<透明存储加密子句> ::= <透明存储加密子句 1>|<透明存储加密子句 2>
<透明存储加密子句 1>::= ENCRYPT [<透明加密用法>]
<透明存储加密子句 2> ::= ENCRYPT <透明加密用法><散列选项>
<透明加密用法> ::= WITH <加密算法> |
                   WITH <加密算法> AUTO |
                   AUTO
<加密密码 1> ::= BY <口令>
<加密密码 2> ::= BY  WRAPPED '<口令密文>'
<散列选项> ::= HASH WITH<散列算法> [<加盐选项>]
<加盐选项> ::= [NO] SALT
```

说明： **此语法格式不完整，主要说明与表列透明加密相关的部分，其他建表相关语法请参考《DM8_SQL 语言使用手册》。**

使用说明：

1． 当使用列加密但没有指定加密算法时，缺省使用 AES256_CBC 加密算法；

2． 散列算法支持 MD5 和 SHA1，用于保证用户数据的完整性，若用户数据被非法修改，则能判断该数据不合法；

例如，以下是一些对表列进行透明加密的例子。

```
CREATE TABLE TEST_ENCRYPT1(C1 INT, C2 INT ENCRYPT);

CREATE TABLE TEST_ENCRYPT2(C1 INT, C2 INT ENCRYPT WITH DES_ECB);

CREATE TABLE TEST_ENCRYPT5(C1 INT, C2 INT ENCRYPT WITH DES_ECB  HASH WITH MD5
SALT);
```

7.1.4 其他数据库对象加密

DM 还支持对存储过程、存储函数、触发器、包、类、自定义类型等的定义进行加密，创建时在对象名称后加上“WITH ENCRYPTION ”即可。

例如，创建存储过程 proc_arg，对其定义进行加密。

```
CREATE OR REPLACE PROCEDURE proc_arg WITH ENCRYPTION(a IN INT, b INT)
AS
BEGIN
       a:=0;
       b:=A+1;
END;
```

7.1.5 重做日志文件加密

DM 支持对联机日志文件和归档日志文件进行加密，可在创建数据库时通过建库参数 RLOG_ENCRYPT_NAME 指定重做日志文件的加密算法，若未指定则不加密。

RLOG_ENCRYPT_NAME 支持使用第三方加密算法，但不支持工作模式为 WORK_MODE_CBC_NOPAD、WORK_MODE_ECB_NOPAD 或 WORK_MODE_EXTKEY 的加密算法，关于加密算法的工作模式可参考 8.1.1 节。

7.2 半透明加密

在 2.2 节中可以看到，创建用户时可以指定存储加密密钥，这个密钥就是为了进行半透明加密时使用的。如果在创建用户时并没有指定存储加密密钥，系统也会自动为用户生成一个默认的加密密钥。

如果在创建表或修改表时指定对表列进行半透明加密，DM 会使用用户的存储加密密钥对数据进行加密。半透明加密列的密文后追加了 4 个字节的 UID，用户查询数据时，若当前会话用户 ID 与列密文数据中存储的 UID 相同，则返回明文，否则返回 NULL。

建表或修改表时指定对列进半透明加密的语法格式为：

```
……
<列定义> ::= <不同类型列定义> [<列定义子句>] [<STORAGE 子句>][<半透明存储加密子句>]
<半透明存储加密子句> ::= ENCRYPT [WITH <加密算法>] MANUAL[<散列选项>]
<散列选项> ::= HASH WITH <散列算法> [<加盐选项>]
<加盐选项> ::= [NO] SALT
```

使用半透明加密时，用户仅能查看到自己插入的数据。

使用说明：

1．HUGE 表不支持半透明加密列；

2．半透明加密列不能作为索引列，索引列也不能修改为半透明加密列；

3．半透明加密列不允许添加引用约束；

4．若表中包含半透明加密列，则不允许创建聚集索引，同时不允许删除该表原有的聚集索引；

5．不允许修改半透明加密列的列定义；

6．当增加列、删除列或者修改列定义时，若表中包含半透明加密列，则 INI 参数 ALTER_TABLE_OPT 不能设置为 1，若参数 ALTER_TABLE_OPT 为 1，则系统自动按照参数 ALTER_TABLE_OPT 为 0 时处理，关于参数 ALTER_TABLE_OPT 的详细介绍请参考手册《DM8 系统管理员手册》；

7．当增加列且新增列的默认值非 NULL 时，若新增列为半透明加密列，则 INI 参数 ALTER_TABLE_OPT 不能设置为 3，若参数 ALTER_TABLE_OPT 为 3，则系统自动按照参数 ALTER_TABLE_OPT 为 0 时处理；

8．若表中包含半透明加密列，则不允许使用 REBUILD COLUMNS 子句修改表，关于 REBUILD COLUMNS 子句的详细介绍请参考手册《DM8_SQL 语言使用手册》；

9．物化视图的查询语句中不能涉及半透明加密列；

10．若表指定了 USING LONG ROW 存储选项，即支持超长记录存储时，如果表中包含半透明加密列，并且该列的数据类型为字符串类型，则系统将默认该列不支持超长记录存储；

11．半透明列不支持带默认值的快速加列；

12．半透明加密列支持 UPDATE，具体规则如下：

1) 用户通过 UPDATE 语句修改半透明加密列数据，执行提交或回滚操作前：当前用户可查询更新后表中 UID 与自身用户 ID 相匹配的半透明加密列数据，其他用户可查询更新前表中 UID 与自身用户 ID 相匹配的半透明加密列数据；

2) 用户通过 UPDATE 语句修改半透明加密列数据，执行回滚操作后：表中数据恢复到执行 UPDATE 操作前的状态，当前用户和其他用户均可查询更新前表中 UID 与自身用户 ID 相匹配的半透明加密列数据；

3) 用户通过 UPDATE 语句修改半透明加密列数据，执行提交操作后：当前用户和其他用户均可查询更新后表中 UID 与自身用户 ID 相匹配的半透明加密列数据。

例 1，以下是一些对表列进行半透明加密的例子。

```
CREATE TABLE TEST_ENCRYPT6(C1 INT, C2 INT ENCRYPT MANUAL);
CREATE TABLE TEST_ENCRYPT7(C1 INT, C2 INT ENCRYPT WITH DES_ECB MANUAL);
CREATE TABLE TEST_ENCRYPT8(C1 INT, C2 INT ENCRYPT WITH DES_ECB MANUAL HASH
WITH MD5 SALT);
CREATE TABLE TEST_ENCRYPT9(C1 INT, C2 INT ENCRYPT MANUAL HASH WITH MD5 SALT);
```

例 2，以下是对半透明加密列进行更新操作的示例。

创建用户 USER1 和 USER2。

```
CREATE USER USER1 IDENTIFIED BY 123456789;
CREATE USER USER2 IDENTIFIED BY 123456789;
```

```
GRANT DBA TO USER1;

GRANT DBA TO USER2;

COMMIT;
```

用户 USER1 创建包含半透明加密列的表 TEST 并插入数据，执行提交操作后，用户 USER1 和 USER2 对表中数据进行查询。

```
CONN USER1/123456789@LOCALHOST

CREATE TABLE TEST(C1 VARCHAR ENCRYPT MANUAL);

INSERT INTO TEST VALUES('AAA');

COMMIT;

SELECT * FROM USER1.TEST;

行号     C1

---------- ---

1          AAA


CONN USER2/123456789@LOCALHOST

SELECT * FROM USER1.TEST;

行号     C1

---------- ----

1          NULL
```

用户 USER2 对表 TEST 中的半透明加密列进行更新，执行提交操作后，用户 USER1 和 USER2 对表中数据进行查询。

```
CONN USER2/123456789@LOCALHOST

UPDATE USER1.TEST SET C1 = 'BBB';

COMMIT;

SELECT * FROM USER1.TEST;

行号     C1

---------- ---

1          BBB


CONN USER1/123456789@LOCALHOST
```

```
SELECT * FROM USER1.TEST;

行号     C1

---------- ----

1          NULL
```

用户 USER1 对表 TEST 中的半透明加密列进行更新，执行回滚操作后，用户 USER1 和 USER2 对表中数据进行查询。

```
CONN USER1/123456789@LOCALHOST

UPDATE USER1.TEST SET C1 = 'CCC';

ROLLBACK;

SELECT * FROM USER1.TEST;

行号     C1

---------- ----

1          NULL


CONN USER2/123456789@LOCALHOST

SELECT * FROM USER1.TEST;

行号     C1

---------- ---

1          BBB
```

7.3 非透明加密

DM 对非透明加密的支持是通过对用户提供加解密接口实现的。用户在使用非透明加密时，需要提供密钥并调用加解密接口。采用非透明加密可以保证个人私密数据不被包括 DBA 在内的其他人获取。

非透明加密通过用户调用存储加密函数来进行，DM 提供了一系列的存储加密函数，还提供了一个数据加密包 DBMS_OBFUSCATION_TOOLKIT。本节主要介绍 DM 的存储加密函数，关于包 DBMS_OBFUSCATION_TOOLKIT 的介绍请参看《DM8 系统包使用手册》的相关章节。

DM 提供了下列存储加密函数：

1. CFALGORITHMSENCRYPT

```
CFALGORITHMSENCRYPT(
    SRC             VARCHAR/TEXT/CLOB,
    ALGORITHM       INT,
    KEY             VARCHAR
)
```

参数说明：

SRC　　需要被加密明文的类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

功能说明：

对指定类型的明文进行加密，并返回密文。

返回值：

加密后的密文，密文数据类型和 SRC 类型相同

举例说明：

对数据进行加密：

```
CREATE TABLE enc_001(c1 VARCHAR(200));
INSERT INTO enc_001 VALUES(CFALGORITHMSENCRYPT('tt', 514, '仅供测试使用'));
```

这样就将加密后的数据存放到表列中。

2. CFALGORITHMSDECRYPT

```
CFALGORITHMSDECRYPT(
    SRC             VARCHAR/TEXT/CLOB,
    ALGORITHM       INT,
    KEY             VARCHAR
)
```

参数说明：

SRC　　需要被解密密文的数据类型

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

功能说明：

对密文进行解密，并得到加密前的相同数据类型的明文。

返回值：

解密后的明文，明文数据类型和 SRC 类型相同

举例说明：

对数据进行解密：

```
SELECT CFALGORITHMSDECRYPT(c1, 514, '仅供测试使用') FROM enc_001;

行号        CFALGORITHMSDECRYPT(C1,514,'仅供测试使用')

---------- ----------------------------------------

1           tt
```

3. SF_ENCRYPT_BINARY

```
SF_ENCRYPT_BINARY(
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 VARBINARY 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 VARBINARY 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_002(c1 VARBINARY(200));

INSERT INTO enc_002 VALUES(SF_ENCRYPT_BINARY(0x12345678EF, 514, '仅供测试使用',
NULL));
```

这样就将加密后的数据存放到表列中。

4. SF_DECRYPT_TO_BINARY

```
SF_DECRYPT_TO_BINARY(
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被解密的 VARBINARY 类型密文

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 VARBINARY 类型明文。

返回值：

解密后的明文，数据类型为 VARBINARY

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_BINARY(c1, 514, '仅供测试使用',NULL) FROM enc_002;
```

```
行号        SF_DECRYPT_TO_BINARY(C1,514,'仅供测试使用',NULL)
---------- ------------------------------------------------
1           0x12345678EF
```

5. SF_ENCRYPT_CHAR

```
SF_ENCRYPT_CHAR(
    SRC             VARCHAR,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 CHAR/VARCHAR 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 VARCHAR 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_003(c1 VARBINARY(200));
INSERT INTO enc_003 VALUES(SF_ENCRYPT_CHAR('测试数据', 514, '仅供测试使用',NULL));
```

这样就将加密后的数据存放到表列中。

6. SF_DECRYPT_TO_CHAR

```
SF_DECRYPT_TO_CHAR(
```

```
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被解密的 VARBINARY 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 VARCHAR 类型明文。

返回值：

解密后的明文，数据类型为 VARCHAR

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_CHAR(c1, 514, '仅供测试使用',NULL) FROM enc_003;
行号        SF_DECRYPT_TO_CHAR(C1,514,'仅供测试使用',NULL)
---------- ----------------------------------------------
1           测试数据
```

7. SF_ENCRYPT_DATE

```
SF_ENCRYPT_DATE(
    SRC             DATE,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 DATE 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 DATE 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_004(c1 VARBINARY(200));
INSERT INTO enc_004 VALUES(SF_ENCRYPT_DATE(cast('2011-1-1' as date), 514, '仅供测试使用',NULL));
```

这样就将加密后的日期类型数据存放到表列中。

8. SF_DECRYPT_TO_DATE

```
SF_DECRYPT_TO_DATE(
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被解密的 VARBINARY 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 DATE 类型明文。

返回值：

解密后的明文，数据类型为 DATE

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_DATE(c1, 514, '仅供测试使用',NULL) FROM enc_004;

行号        SF_DECRYPT_TO_DATE(C1,514,'仅供测试使用',NULL)

---------- ----------------------------------------------

1           2011-01-01
```

9. SF_ENCRYPT_DATETIME

```
SF_ENCRYPT_DATETIME(
    SRC             DATETIME,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 DATETIME 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 DATETIME 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_005(c1 VARBINARY(200));

INSERT INTO enc_005 VALUES(SF_ENCRYPT_DATETIME(cast('2011-12-12 11:11:11' as

datetime), 514, '仅供测试使用',NULL));
```

这样就将加密后的日期时间类型数据存放到表列中。

10. SF_DECRYPT_TO_DATETIME

```
SF_DECRYPT_TO_DATETIME(
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 DATETIME 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 DATETIME 类型明文。

返回值：

解密后的明文，数据类型为 DATETIME

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_DATETIME (c1, 514, '仅供测试使用',NULL) FROM enc_005;

行号        SF_DECRYPT_TO_DATETIME(C1,514,'仅供测试使用',NULL)

---------- ------------------------------------------------
```

```
1          2011-12-12 11:11:11.0
```

11. SF_ENCRYPT_DEC

```
SF_ENCRYPT_DEC(
    SRC             DEC,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被加密的 DEC 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 DEC 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_006(c1 VARBINARY(200));
INSERT INTO enc_006 VALUES(SF_ENCRYPT_DEC(cast('3.1415900000'as dec(15,10)),
514, '仅供测试使用',NULL));
```

这样就将加密后的 DEC 数据存放到表列中。

12. SF_DECRYPT_TO_DEC

```
SF_DECRYPT_TO_DEC(
    SRC             VARBINARY,
```

```
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被解密的 VARBINARY 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 DEC 类型明文。

返回值：

解密后的明文，数据类型为 DEC

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_DEC(c1, 514, '仅供测试使用',NULL) FROM enc_006;
行号        SF_DECRYPT_TO_DEC(C1,514,'仅供测试使用',NULL)
---------- ----------------------------------------------
1           3.141590000000
```

13. SF_ENCRYPT_TIME

```
SF_ENCRYPT_TIME(
    SRC             TIME,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCAHR
)
```

参数说明：

SRC　　需要被加密的 TIME 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对 TIME 类型明文进行加密，并返回密文。

返回值：

加密后的密文，数据类型为 VARBINARY

举例说明：

对数据进行加密：

```
CREATE TABLE enc_007(c1 VARBINARY(200));
INSERT INTO enc_007 VALUES(SF_ENCRYPT_TIME(cast('12:12:12' as time), 514, '仅供测试使用',NULL));
```

这样就将加密后的时间类型数据存放到表列中。

14.SF_DECRYPT_TO_TIME

```
SF_DECRYPT_TO_TIME(
    SRC             VARBINARY,
    ALGORITHM       INT,
    KEY             VARCHAR,
    IV              VARCHAR
)
```

参数说明：

SRC　　需要被解密的 VARBINARY 类型数据

ALGORITHM　　加密算法 ID，不可以为 NULL。加密算法对应的 ID 可通过查询 V$CIPHERS 得到

KEY　　采用的密钥，不可以为 NULL

IV　　采用的初始化矢量，可以为 NULL

功能说明：

对密文进行解密，并得到加密前的 TIME 类型明文。

返回值：

解密后的明文，数据类型为 TIME

举例说明：

对数据进行解密：

```
SELECT SF_DECRYPT_TO_TIME(c1, 514, '仅供测试使用',NULL) FROM enc_007;

行号        SF_DECRYPT_TO_TIME(C1,514,'仅供测试使用',NULL)

---------- ----------------------------------------------

1           12:12:12.0
```

15.SF_GET_CIPHER_NAME

```
SF_GET_CIPHER_NAME(

    CIPHER_ID IN INT

)
```

参数说明：

CIPHER_ID　　加密算法 ID

功能说明：

根据加密算法 ID，获取加密算法的名称

返回值：

加密算法的名称，最大长度 32767

举例说明：

```
CREATE TABLE T1(C1 INT ENCRYPT ,C2 INT ENCRYPT );

SELECT ID FROM SYSOBJECTS WHERE NAME = 'T1';

    //设返回 1152

SELECT * FROM SYS.SYSCOLCYT WHERE TID = 1152;

    //SYSCOLCYT 表中 ENC_ID 列对应的值，即为列的加密类型 ID，此处返回 2050

SELECT SF_GET_CIPHER_NAME(2050);

行号        SF_GET_CIPHER_NAME(2050)
```

```
---------- ------------------------

1          AES256_CBC
```

8 加密引擎

DM 系统中内置了常用的 DES，AES，RC4，SHA 等类型的加密和散列算法供用户使用，以此来保护数据的安全性。然而在有些特殊的环境下，这些算法可能不能满足用户的需求，用户可能希望使用自己特殊的或更高强度的加密和散列算法（例如：国密算法）。DM 的加密引擎功能则可以满足这样的需求。

用户只需要按照 DM 提供的加密引擎 C 语言编程接口，封装自己的加密和散列算法，并编译成第三方加密动态库，即可以在 DM 系统中使用自己的加密和散列算法。


注意：

用户使用自行封装的加密和散列算法前需确保该算法适用于当前应用场景。例如，用户使用自行封装的加密算法进行存储加密时，需要确保该算法适用于存储加密。

第三方加密动态库需要提供如下的对外接口，如下表所示。

表 8.1 第三方加密接口定义

接口名称	功能说明
cipher_get_count	获取实现的算法个数
cipher_get_info	获取算法的基本信息，当实现了 cipher_get_info_ex 时，cipher_get_info 自动失效
cipher_get_info_ex	获取算法的基本信息
cipher_encrypt_init	加密初始化
cipher_get_cipher_text_size	计算给定长度的明文，加密后得到的密文所占的字节数
cipher_encrypt	加密函数
cipher_cleanup	回收密码算法在处理过程中申请的系统资源
cipher_decrypt_init	解密初始化
cipher_decrypt	解密函数
cipher_hash_init	散列过程的初始化工作
cipher_hash_update	计算消息 msg 的散列值
cipher_hash_final	散列值的实际长度
crypto_login	登录设备
crypto_logout	退出设备
crypto_read_cert	读取用户证书
cipher_gen_random	产生随机数
cipher_asym_sign	用户私钥对数据进行签名

cipher_asym_verify	用户公钥对数据进行验签
crypto_get_name	获取加密引擎名字
crypto_get_type	获取加密引擎类型。0：第三方软加密；1：第三方硬件加密；2：ukey。
crypto_encrypt	加密
crypto_decrypt	解密。与 crypto_encrypt 对应

8.1 编程接口介绍

8.1.1 算法信息相关接口

1. cipher_get_count

```
ulint
cipher_get_count(
);
```

功能说明：

获取实现的算法个数。

返回值：

返回实现的算法个数。

2. cipher_get_info

```
dm_bool
cipher_get_info(
    ulint     seqno,
    ulint*    cipher_id,
    byte**    cipher_name,
    byte*     type,
    ulint*    blk_size,
    ulint*    kh_size
);
```

参数说明：

seqno	算法的序号，从 1 开始，小于等于算法的个数
cipher_id	算法的 ID，第三方加密算法的 ID 必须大于等于 5000
cipher_name	算法名
type	算法类型
blk_size	块大小
kh_size	key 或者 hash 大小

功能说明：

获取算法的基本信息，有 cipher_get_info_ex 存在时，cipher_get_info 失效。

返回值：

TRUE/FALSE	获取成功/获取失败

3. cipher_get_info_ex

```
dm_bool
cipher_get_info_ex(
    ulint     seqno,
    ulint*    cipher_id,
    byte**    cipher_name,
    byte*     type,
    ulint*    blk_size,
    ulint*    kh_size,
    byte*     work_mode
);
```

参数说明：

seqno	算法的序号，从 1 开始，小于等于算法的个数
cipher_id	算法的 ID，第三方加密算法的 ID 必须大于等于 5000
cipher_name	算法名
type	算法类型
blk_size	块大小
kh_size	key 或者 hash 大小

work_mode　　工作模式。work_mode 取值见下表

表 8.2 work_mode 取值

模式名取值	释义
0 或 WORK_MODE_ECB	ECB（Electronic CodeBook），电子密码本模式
2 或 WORK_MODE_CBC	CBC（Cipher-Block Chaining），密码块连接模式
4 或 WORK_MODE_CFB	CFB（Cipher FeedBack），密文反馈模式
8 或 WORK_MODE_OFB	OFB（Output FeedBack），输出反馈模式
16 或 WORK_MODE_CBC_NOPAD	NOPAD（NO PADDING），非填充模式
32 或 WORK_MODE_ECB_NOPAD	NOPAD（NO PADDING），非填充模式
64 或 WORK_MODE_EXTKEY	WORK_MODE_EXTKEY（EXTERNAL KEY），外部密钥模式，专用算法模式
67 或 WORK_MODE_KID	WORK_MODE_KID(KEY ID)，密钥 ID 模式， 专用算法模式。备份还原不支持使用 WORK_MODE_KID 的加密算法，该种加密算法也不支持作为 external_cipher_name 来进行数据库初始化

WORK_MODE_KID 配置文件为 dmkid.ini。dmkid.ini 是使用 WORK_MODE_KID 加密算法的必要文件，文件位于 DM 安装目录$DM_HOME/bin 目录下。dmkid.ini 详细参数如下表所示：

表 8.3 dmkid.ini 中控制文件相关参数

参数名	缺省值	说明
InstanceURL	http://127.0.0.1:8214/kmip	加密服务器 URL
InstanceFormat	1	格式：NONE = 1, JSON = 1, XML = 2
InstanceUser	KmipAdmin	用户名
InstancePwd	88888888	密码
InstanceCa	CA 证书	如果使用 HTTPS，则在这里指定 CA 证书
RETRY_TIMES	5	出现错误时，重试次数
RETRY_SLEEP	1000	每次重试等待时间(MS)
ENC_DEBUG	0	是否生成日志
ENC_LOG	ENC.LOG	日志文件完整路径
CONFIG_LOG_FILE	无	加密服务器生成日志完整路径

功能说明：

获取算法的基本信息，有 cipher_get_info_ex 存在时，cipher_get_info 失效。

返回值：

TRUE/FALSE　　获取成功/获取失败

4. cipher_get_para

```
dmbool cipher_get_para(
       ulint cipher_id,
       ulint  para_id,
       void* value
);
```

参数说明：

cipher_id　　算法的 ID，第三方加密算法的 ID 必须大于等于 5000

para_id　　参数 ID，目前仅支持两个参数，0 表示 WORKMODE，1 表示 EXTENDSIZE

value　　参数值。类型将根据 para_id 变化：para_id 为 0 时，value 为 unsigned char*；para_id 为 1 时，value 为 unsigned int*。用户须保证不会写越界.

功能说明：

获取当前加密算法的当前参数的值，目前支持获取 WORK_MODE 和 EXTEND_SIZE。WORK_MODE 表示加密算法工作模式；EXTEND_SIZE 表示加密算法最大扩展长度，即密文相对于明文一次加密最大扩展的长度，通常可以认为是一次加密(密文-明文)的最大值。

返回值：

TRUE/FALSE　　获取成功/获取失败

当获取失败时，将使用参数的默认值。WORK_MODE 默认值为 0；EXTEND_SIZE 默认值为通过 cipher_get_info 获得的 kh_size。

当同时使用 cipher_get_info_ex 和 cipher_get_para 对 work_mode 进行正确赋值时，最终结果为 cipher_get_para 所设置的值。

8.1.2 加密过程相关接口

1. cipher_encrypt_init

```
dm_bool
cipher_encrypt_init(
    ulint     inner_id,
    byte*     key,
    ulint     key_size,
    void**    encrypt_para
);
```

参数说明：

inner_id　　输入参数，调用的密码算法在加密引擎中的内部编号

key　　输入参数，加密密钥

key_size　　输入参数，密钥的字节数

encrypt_para　输入参数，加密过程的全局参数，由用户负责内存分配。该参数为加密过程全程使用的参数，由用户负责分配空间，其中可以携带 key、 key_size、用户自定义信息等。该参数需要在整个加密过程中保持有效，后续加密相关接口中的 encrypt_para 都是由该接口生成。

功能说明：

加密初始化。

返回值

TRUE/FALSE　　初始化成功/初始化失败

2. cipher_get_cipher_text_size

```
lint
cipher_get_cipher_text_size(
    ulint     inner_id,
    void*     encrypt_para,
    lint      plain_text_size
```

```
);
```

参数说明：

inner_id　　输入参数，密码算法在加密引擎中的内部编号

encrypt_para　输入参数，cipher_encrypt_init 的输出参数。ncrypt_para 是 cipher_encrypt_init 时产生的输出参数，如何利用其中信息由用户决定

plain_text_size　　输入参数，待加密的明文所占的字节数

功能说明

计算给定长度的明文加密后得到的密文所占的字节数。DM 照返回值分配密文缓冲区。

返回值

与明文对应的密文所占的字节数。

3. cipher_encrypt

```
lint
cipher_encrypt(
    ulint     inner_id,
    void*     encrypt_para,
    byte*     plain_text,
    ulint     plain_text_size,
    byte*     cipher_text,
    ulint     cipher_text_buf_size
);
```

参数说明：

inner_id　　输入参数，密码算法在加密引擎中的内部编号

encrypt_para　输入参数，密码算法的全局变量。ncrypt_para 是 cipher_encrypt_init 时产生的输出参数，如何利用其中信息由用户决定

plain_text　输入参数，明文

plain_text_size　　输入参数，明文所占的字节数

cipher_text　输出参数，密文

cipher_text_buf_size　　输入参数，密文缓冲区的字节数

功能说明

加密。用户需要保证根据提供的参数不会出现写越界等问题，密文长度不能超过所提供的空间，这里的密文缓冲区字节数不会小于 cipher_get_cipher_text_size 的返回值，用户还需确保 cipher_get_cipher_text_size 与该接口使用相同参数时，应该返回相同值。

返回值

加密后密文的长度。

4. cipher_cleanup

```
void
cipher_cleanup(
    ulint     inner_id,
    void*     encrypt_para
);
```

参数说明：

inner_id 输入参数，密码算法在加密引擎中的内部编号

encrypt_para 输入参数，密码算法的全局参数

功能说明

回收密码算法在处理过程中申请的系统资源。一般情况下，用户需要在这里释放在 cipher_encrypt_init 或 cipher_decrypt_init 中创建的 encrypt_para/decrypt_para。

返回值

无，假定该算法总是成功。

在使用一个密码算法进行加密之前应首先对该密码算法进行初始化处理，DM 数据库管理系统根据所指定的密码算法准备明文，并为密文分配密文缓冲区。然后调用加密函数进行加密，完成加密动作后回收加密过程中申请的系统资源。

8.1.3 解密过程相关接口

1. cipher_decrypt_init

```
dm_bool
cipher_decrypt_init(
    ulint     inner_id,
    byte*     key,
    ulint     key_size,
    void**    decrypt_para
);
```

参数说明：

inner_id	输入参数，调用的密码算法在加密引擎中的内部编号
key	输入参数，加密密钥
key_size	输入参数，密钥的字节数
decrypt_para	输入参数，解密过程的全局参数，由用户分配内存

功能说明

解密初始化。用法与加密的 cipher_encrypt_init 类似。

返回值

TRUE/FALSE	初始化成功/初始化失败

2. cipher_decrypt

```
lint
cipher_decrypt(
    ulint     inner_id,
    void*     decrypt_para,
    byte*     cipher_text,
    ulint     cipher_text_size,
    byte*     plain_text,
    ulint     plain_text_buf_size
```

```
);
```

参数说明：

inner_id　　输入参数，调用的密码算法在加密引擎中的内部编号

decrypt_para　输入参数，cipher_decrypt_init 的输出参数。该参数来自于 cipher_decrypt_init 的输出参数，如何利用其中数据由用户决定

cipher_text　输入参数，密文

cipher_text_size　输入参数，密文所占的字节数

plain_text　输出参数，明文

plain_text_buf_size　输入参数，明文缓冲区的长度

功能说明

解密。用户需要保证根据提供的参数不会出现写越界等问题，明文长度不能超过所提供的空间。

返回值

明文的实际长度。

在使用一个密码算法进行解密之前应首先对该密码算法进行初始化处理，然后调用解密函数进行解密，完成解密动作后回收解密过程中申请的系统资源。

8.1.4 散列过程相关接口

1. cipher_hash_init

```
dm_bool
cipher_hash_init(
    ulint     inner_id,
    void**    hash_para
);
```

参数说明：

inner_id　　输入参数，调用的密码算法在加密引擎中的内部编号

hash_para　　输入参数，散列过程的全局变量，由用户分配内存

功能说明

散列过程初始化。

返回值

TRUE/FALSE　　初始化成功/初始化失败

2. cipher_hash_update

```
void
cipher_hash_update(
    ulint     inner_id,
    void*     hash_para,
    byte*     msg,
    ulint     msg_size
);
```

参数说明：

inner_id　　输入参数，调用的密码算法在加密引擎中的内部编号

hash_para　　输入参数，散列过程中的全局参数

msg　　输入参数，待散列的消息

msg_size　　输入参数，消息长度

功能说明

计算消息的散列值。

返回值

无

3. cipher_hash_final

```
lint
cipher_hash_final(
    ulint     inner_id,
    void*     hash_para,
    byte*     digest,
```

```
    ulint     digest_buf_size
);
```

参数说明：

inner_id　　输入参数，调用的密码算法在加密引擎中的内部编号

hash_para　　输入参数，散列过程中的全局参数

digest　　输入参数，散列值

digest_buf_size　　输入参数，散列缓冲区的大小

功能说明

将整个散列过程得到的散列值存放到 digest 中。释放 hash_para 空间。

返回值

散列值的实际长度。

8.1.5 其他可选相关接口

1. cipher_asym_sign

```
dm_bool
cipher_asym_sign(
     ulint   inner_id,
     byte*   prikey,
     ulint   prikey_size,
     byte*   data,
     ulint   data_size,
     byte*   signdata,
     ulint*  signdata_buf_size
);
```

参数说明：

inner_id　　调用的密码算法在加密引擎中的内部编号

prikey　　私钥

prikey_size　　私钥长度

data　　需要进行签名的数据

data_size　　数据的长度

signdata　　数据的签名值

signdata_buf_size　签名值长度

功能说明

用户私钥对数据进行签名。

返回值

TRUE/FALSE　　签名成功/签名失败

2. cipher_asym_verify

```
dm_bool
cipher_asym_verify(
    ulint    inner_id,
    byte*    pubkey,
    ulint    pubkey_size,
    byte*    data,
    ulint    data_size,
    byte*    signdata,
    ulint    signdata_size
);
```

参数说明：

inner_id　　调用的密码算法在加密引擎中的内部编号

pubkey　　公钥

pubkey_size　公钥长度

data　　需要进行签名的数据

data_size　　数据的长度

signdata　　数据的签名值

signdata_buf_size　签名值长度

功能说明

用户公钥对数据进行验签。

返回值

TRUE/FALSE　　验签成功/验签失败

3. crypto_login

```
dm_bool
crypto_login(
    void**    cipher_para,
    byte*     pin,
    ulint     pin_len
);
```

参数说明：

cipher_para　　登录过程的全局参数，由用户负责内存分配

pin　　设备口令

pin_len　　口令长度

功能说明

登录设备。

返回值

TRUE/FALSE　　登录成功/登录失败

4. crypto_logout

```
dm_bool
crypto_logout(
    void*     cipher_para
);
```

参数说明：

cipher_para　　登录过程时使用的全局参数，此时需要释放

功能说明

退出设备。

返回值

TRUE/FALSE　　退出成功/退出失败

5. crypto_read_cert

```
dm_bool
crypto_read_cert(
    void*     cipher_para,
    byte*     cert,
    ulint*    cert_len
);
```

参数说明：

cipher_para　登录过程时使用的全局参数

cert　　输出参数，用户证书信息

cert_len　　输出参数，用户证书长度

功能说明

读取用户证书。

返回值

TRUE/FALSE　　读取成功/读取失败

6. cipher_gen_random

```
dm_bool
cipher_gen_random(
    byte*     random,
    ulint     random_length
);
```

参数说明：

random　输出参数，生成的随机数

random_length　　输入参数，需要产生随机数的长度

功能说明

产生随机数。

返回值

TRUE/FALSE　　随机数生成成功/随机数生成失败

7. cipher_get_name

```
dm_bool
crypto_get_name(
    byte**    crypto_name,
    ulint*    len
);
```

参数说明：

crypto_name　　输出参数，加密引擎的名字

len　　输出参数，加密引擎的名字的长度

功能说明

获取加密引擎名字。

返回值

TRUE/FALSE　　获取成功/获取失败

8. cipher_get_type

```
dm_bool
crypto_get_type(
    ulint*    crypto_type
);
```

参数说明：

crypto_type　　输出参数，加密引擎的类型

功能说明

获取加密引擎类型。

返回值

TRUE/FALSE　　获取成功/获取失败

9. crypto_encrypt

```
lint
crypto_encrypt(
    byte*     plain_text,
    ulint     plain_text_size,
    byte*     cipher_text,
    ulint     cipher_text_buf_size
);
```

参数说明：

plain_text　　输入参数，明文

plain_text_size　　输入参数，明文所占的字节数

cipher_text　　输出参数，密文

cipher_text_buf_size　　输入参数，密文缓冲区的字节数

功能说明

加密。由第三方动态加密库指定加密的算法和密钥 key，可以使用第三方设备（硬件加密卡、ukey）中的密钥 key。为了安全起见，设备中的密钥 key 可以不加载到内存中，加密的全程都在第三方设备内完成。

返回值

加密后密文的长度

10. crypto_decrypt

```
lint
crypto_decrypt(
    byte*     cipher_text,
    ulint     cipher_text_size,
    byte*     plain_text,
    ulint     plain_text_buf_size
);
```

参数说明：

cipher_text　输入参数，密文

cipher_text_buf_size　　输入参数，密文所区的字节数

plain_text　输出参数，明文

plain_text_size　　输入参数，明文缓冲区的字节数

功能说明

解密。与 crypto_encrypt 对应。

返回值

明文的实际长度

11. cipher_get_key_id

```
lint
cipher_get_key_id(
   ulint     cipher_id,
   byte*     key_id,
   ulint     key_id_size,
   ulint*    key_size
);
```

参数说明：

cipher_id　　输入参数，密文

key_id　　输出参数，key_id

key_id_size　输入参数，key_id 空间大小

key_size　　输出参数，key_id 实际大小

功能说明

生成加密 key 的标识符 key_id，需要对应加密算法工作模式为 WORK_MODE_KID。

返回值

是否生成成功

12. cipher_free_key_id

```
lint
```

```
cipher_free_key_id(
    ulint      cipher_id,
    byte*      key_id,
    ulint      key_id_size
);
```

参数说明：

cipher_id　　输入参数，密文

key_id　　输入参数，key_id

key_id_size　　输入参数，key_id 空间大小

功能说明

释放加密 key 的标识符 key_id，需要对应加密算法工作模式为 WORK_MODE_KID。

返回值

是否释放成功

8.2 接口库文件使用说明

想要使用自己的加密算法的用户需要用 C 语言编写实现 8.1 节中介绍的接口（8.1.5 节中的接口可选），并编译生成第三方动态库文件，如：Windows 操作系统下为 external_crypto.dll（LINUX 操作系统下为 libexternal_crypto.so）。

DM 支持同时加载多个第三方动态库文件。但这些 DLL 库文件必须放在 DM 数据库服务器所在 BIN 目录的 external_crypto_libs 子目录下。external_crypto_libs 目录需用户自己创建。如果是 DM 控制台工具，需将这些 DLL 库文件放在 CONSOLE 工具所在 TOOL 目录的 external_crypto_libs 子目录下。

为了保证加密算法有效性，所有第三方库文件中的加密方法名及 ID 不同。DM 提供一个动态视图 V$EXTERNAL_CIPHERS，用户可查看所有的第三方加密算法的 ID、名称、所在的库文件以及是否有效标记。

V$EXTERNAL_CIPHERS 的结构如下表所示。

表 8.4 V$EXTERNAL_CIPHERS 结构

序号	列	数据类型	说明
1	ID	INTEGER	算法 ID

2	NAME	VARCHAR(128)	算法名
3	LIB	VARCHAR(300)	算法所在的 lib 库文件名
4	VALID	CHAR	算法是否有效。'Y'：是；'N'：否

8.3 UKEY 使用说明

DM 支持使用 UKEY 进行身份鉴别和加密。使用 UKEY 进行身份鉴别时，应按照 8.1 节的介绍编写两个动态库，分别作用于客户端和服务器。编写动态库时应注意：

1. 客户端和服务器端的动态库 crypto_get_name()的返回值必须相同；

2. 客户端动态库的名字*.dll/*.so 中的*必须与 crypto_get_name()返回值相同，否则 JDBC 驱动无法完成 UKEY 鉴别，DPI 接口无此要求；

3. 客户端动态库的 crypto_get_type()应返回 2，表示移动设备 UKEY；服务器端动态库的 crypto_get_type()返回 1，表示软件或硬件加密卡。

8.4 DM 提供的第三方加密和散列算法

DM 提供了一些第三方加密和散列算法供用户使用，例如：SM3、SM4 算法。第三方加密和散列算法的动态库文件为位于 bin/external_crypto_libs 文件夹下的 openssl1.1.1_crypto.dll（Windows 操作系统）或 libopenssl_crypto.so（Linux 操作系统），此动态库依赖于 bin 文件夹下的 libeay32.dll（Windows 操作系统）或 libcrypto.so（Linux 操作系统）。

表 8.5 DM 提供的第三方加密和散列算法（SM3、SM4）

算法名称	算法类型	分组长度	密钥长度
OPENSSL_SM4_ECB	分组加密算法	16	16
OPENSSL_SM4_CBC	分组加密算法	16	16
OPENSSL_SM4_CFB	分组加密算法	16	16
OPENSSL_SM4_OFB	分组加密算法	16	16
OPENSSL_SM3	散列算法	-	-
OPENSSL_SM4_ECB_NOPAD	分组加密算法	16	16
OPENSSL_SM4_CBC_NOPAD	分组加密算法	16	16
OPENSSL_SM4_CFB_V1	分组加密算法	16	16
OPENSSL_SM4_OFB_V1	分组加密算法	16	16

SM3、SM4 算法可以通过查询动态视图 V$CIPHERS 查看。因为通过 external_crypto_libs 文件夹下的文件进行支持， 因此也可以通过查询动态视图 V$external_ciphers 进行查询。

```
select * from V$external_ciphers;


行号        ID           NAME                    LIB                       VALID
---------- ----------- ---------------------- ----------------------- -----
1           5201         OPENSSL_SM4_ECB        openssl1.1.1_crypto.dll Y
2           5202         OPENSSL_SM4_CBC        openssl1.1.1_crypto.dll Y
3           5203         OPENSSL_SM4_CFB        openssl1.1.1_crypto.dll Y
4           5204         OPENSSL_SM4_OFB        openssl1.1.1_crypto.dll Y
5           5207         OPENSSL_SM3            openssl1.1.1_crypto.dll Y
6           5205         OPENSSL_SM4_ECB_NOPAD openssl1.1.1_crypto.dll Y
7           5206         OPENSSL_SM4_CBC_NOPAD openssl1.1.1_crypto.dll Y
8           5208         OPENSSL_SM4_CFB_V1     openssl1.1.1_crypto.dll Y
9           5209         OPENSSL_SM4_OFB_V1     openssl1.1.1_crypto.dll Y
```

DM 提供的 SM3，SM4 算法是由 openssl 提供，DM 进行封装的。因此，如果有用户不使用 DM 提供的 openssl 第三方库，则用户需要保证使用版本号大于 1.1.1 的 openssl，大于 1.1.1 的版本才包含 SM3，SM4 算法。

SM3 为散列算法，与 DM 内置的散列算法用途一致，可以用于完整性校验。SM4 为分组加密算法，与 DM 内置的分组加密算法用途一致，可以用于通信加密、存储加密等场景。

其中，OPENSSL_SM4_CFB_V1、OPENSSL_SM4_OFB_V1 算法分别是 OPENSSL_SM4_CFB、OPENSSL_SM4_OFB 算法的新版本，不与老版本兼容，因此当用户需要使用上述算法时，建议优先选择新版本。

8.5 编程实例

本节用一个实例来说明 DM 加密引擎接口的编程方法。

例子中用户自定义加密算法为实现一个简单的异或加密算法和一个 HASH 算法。异或加密算法包括单字节的流加密（xor1、xor2）和 4 字节的块加密方式（xor3），HASH 算法为 MD5（名称为 new_md5）。

第一步，生成动态库。

使用 Microsoft Visual Studio 2008 创建新项目 mycrypto，将 xor.h 和 xor.c、hash.h 和 hash.c、crypto_engine.h 和 my_crypto.c 生成动态库 mycrypto.dll。其中 crypto_engine.h 被放在 DM 安装目录的 include 文件夹中。

涉及到的文件如下：

- xor.h

说明：异或加密算法函数的声明。

```
//xoh.h :declare the xor cipher
#ifndef CRYPTO_ENGINE_H
typedef unsigned char byte;
#endif
int
xor_encrypt(
    byte key,
    byte* plain_text,
    int plain_text_len,
    byte* cipher_text,
    int cipher_text_len);
int
xor_decrypt(
    byte key,
    byte* cipher_text,
    int cipher_text_len,
    byte* plain_text,
    int plain_text_len);
int
xor_data_block_encrypt(
```

```
    int key,
    byte* in_text,
    int in_text_len,
    byte* out_text,
    int out_text_len);
int
xor_data_block_decrypt(
    int key,
    byte* in_text,
    int in_text_len,
    byte* out_text,
    int out_text_len);
```

- **xor.c**

  说明：异或加密算法的实现。

```
// xor.c : implement the xor cipher.
#include <stdio.h>
#include <stdlib.h>
#include "xor.h"
int
xor_data(
    byte key,
    byte* in_text,
    int in_text_len,
    byte* out_text,
    int out_text_len)
{
    int i = 0;
    for (i = 0; i < in_text_len; i++)
    {
        out_text[i] = in_text[i] ^ key;
```

```
    }
    return in_text_len;
}
int
xor_data_block_encrypt(
               int key,
               byte* in_text,
               int in_text_len,
               byte* out_text,
               int out_text_len)
{
    int i = 0;
    int in_buf = 0;
int out_buf = 0;
    int out_len = 0;
    int pos = 0;
    int block_len = sizeof(int);
    byte* p = NULL;
    byte* t = NULL;
    int n = 0;
    int m = 0;
    int fill_num = 0;
    n = in_text_len / block_len + 1;
    m = in_text_len % block_len;
    out_len = block_len * n;
    p = (byte*)malloc(out_len);
    memset(p, 0, out_len);
    memcpy(p, in_text, in_text_len);
    fill_num = out_len - in_text_len;
    memset(p + in_text_len, fill_num, out_len - in_text_len);
```

```
    for (i = 0; i < out_len; i++)
    {
        p[i] = p[i] ^ key;
    }
    memcpy(out_text, p, out_len);
    free(p);
    return out_len;
}
int
xor_data_block_decrypt(
               int key,
               byte* in_text,
               int in_text_len,
               byte* out_text,
               int out_text_len)
{
    int i = 0;
    int in_buf = 0;
    int out_buf = 0;
    int out_len = 0;
    int pos = 0;
    int block_len = sizeof(int);
    byte* p = NULL;
    byte* t = NULL;
    int n = 0;
    int m = 0;
    int fill_num = 0;
    n = in_text_len / block_len;
    m = in_text_len % block_len;
    if (m != 0)
```

```
    {
        return -1;
    }
    p = (byte*)malloc(in_text_len);
    memcpy(p, in_text, in_text_len);
    for (i = 0; i < in_text_len; i++)
    {
        p[i] = p[i] ^ key;
    }
    fill_num = p[in_text_len - 1];
    memcpy(out_text, p, in_text_len - fill_num);
    free(p);
    return in_text_len - fill_num;
}
int
xor_encrypt(
    byte key,
    byte* plain_text,
    int plain_text_len,
    byte* cipher_text,
    int cipher_text_len)
{
    return xor_data(key, plain_text, plain_text_len, cipher_text,
cipher_text_len);
}
int
xor_decrypt(
    byte key,
    byte* cipher_text,
    int cipher_text_len,
```

```
    byte* plain_text,
    int plain_text_len)
{
    return xor_data(key, cipher_text, cipher_text_len, plain_text,
plain_text_len);
}
```

- **hash.h**

  说明：hash 算法函数的声明。

```
// hash.h
typedef struct {
    unsigned int state[4];
    unsigned int count[2];
    unsigned char buffer[64];
} MD5Context;
void MD5_Init(MD5Context * context);
void MD5_Update(MD5Context * context, unsigned char * buf, int len);
void MD5_Final(MD5Context * context, unsigned char digest[16]);
```

- **hash.c**

  说明：hash 算法实现。

```
#include <string.h>
#include "hash.h"
#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
```

```
#define S32 11

#define S33 16

#define S34 23

#define S41 6

#define S42 10

#define S43 15

#define S44 21

static unsigned char PADDING[64] =

{0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};

#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))

#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))

#define H(x, y, z) ((x) ^ (y) ^ (z))

#define I(x, y, z) ((y) ^ ((x) | (~z)))

#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

#define FF(a, b, c, d, x, s, ac)          \

   {                        \

    (a) += F((b), (c), (d)) + (x) + (unsigned int)(ac); \

    (a) = ROTATE_LEFT((a), (s));           \

    (a) += (b);                \

    }

#define GG(a, b, c, d, x, s, ac)          \

    {                        \

    (a) += G((b), (c), (d)) + (x) + (unsigned int)(ac); \

    (a) = ROTATE_LEFT((a), (s));           \

    (a) += (b);                \

    }

#define HH(a, b, c, d, x, s, ac)          \

    {                        \
```

```
    (a) += H((b), (c), (d)) + (x) + (unsigned int)(ac); \
    (a) = ROTATE_LEFT((a), (s));           \
    (a) += (b);                 \
    }
#define II(a, b, c, d, x, s, ac)          \
    {                       \
    (a) += I((b), (c), (d)) + (x) + (unsigned int)(ac); \
    (a) = ROTATE_LEFT((a), (s));           \
    (a) += (b);                 \
    }
static void MD5_Encode(unsigned char * output, unsigned int * input, int len)
{
    unsigned int i, j;
    for (i = 0, j = 0; j < (unsigned int)len; i++, j += 4)
    {
       output[j] = (unsigned char) (input[i] & 0xff);
       output[j + 1] = (unsigned char) ((input[i] >> 8) & 0xff);
       output[j + 2] = (unsigned char) ((input[i] >> 16) & 0xff);
       output[j + 3] = (unsigned char) ((input[i] >> 24) & 0xff);
    }
}
static void MD5_Decode(unsigned int * output, unsigned char * input, int len)
{
    unsigned int i, j;
    for (i = 0, j = 0; j < (unsigned int)len; i++, j += 4)
    {
       output[i] = ((unsigned int) input[j]) |
           (((unsigned int) input[j + 1]) << 8) |
           (((unsigned int) input[j + 2]) << 16) |
           (((unsigned int) input[j + 3]) << 24);
```

```
    }
}
static void MD5_Transform(unsigned int state[4], unsigned char block[64])
{
    unsigned int a = state[0], b = state[1], c = state[2], d = state[3],
x[16];
    MD5_Decode(x, block, 64);
    //Round 1
    FF(a, b, c, d, x[0], S11, 0xd76aa478);    /* 1 */
    FF(d, a, b, c, x[1], S12, 0xe8c7b756);    /* 2 */
    FF(c, d, a, b, x[2], S13, 0x242070db);    /* 3 */
    FF(b, c, d, a, x[3], S14, 0xc1bdceee);    /* 4 */
    FF(a, b, c, d, x[4], S11, 0xf57c0faf);    /* 5 */
    FF(d, a, b, c, x[5], S12, 0x4787c62a);    /* 6 */
    FF(c, d, a, b, x[6], S13, 0xa8304613);    /* 7 */
    FF(b, c, d, a, x[7], S14, 0xfd469501);    /* 8 */
    FF(a, b, c, d, x[8], S11, 0x698098d8);    /* 9 */
    FF(d, a, b, c, x[9], S12, 0x8b44f7af);    /* 10 */
    FF(c, d, a, b, x[10], S13, 0xffff5bb1);   /* 11 */
    FF(b, c, d, a, x[11], S14, 0x895cd7be);   /* 12 */
    FF(a, b, c, d, x[12], S11, 0x6b901122);   /* 13 */
    FF(d, a, b, c, x[13], S12, 0xfd987193);   /* 14 */
    FF(c, d, a, b, x[14], S13, 0xa679438e);   /* 15 */
    FF(b, c, d, a, x[15], S14, 0x49b40821);   /* 16 */
    // Round 2
    GG(a, b, c, d, x[1], S21, 0xf61e2562);    /* 17 */
    GG(d, a, b, c, x[6], S22, 0xc040b340);    /* 18 */
    GG(c, d, a, b, x[11], S23, 0x265e5a51);   /* 19 */
    GG(b, c, d, a, x[0], S24, 0xe9b6c7aa);    /* 20 */
    GG(a, b, c, d, x[5], S21, 0xd62f105d);    /* 21 */
```

```
GG(d, a, b, c, x[10], S22, 0x2441453);    /* 22 */
GG(c, d, a, b, x[15], S23, 0xd8a1e681);   /* 23 */
GG(b, c, d, a, x[4], S24, 0xe7d3fbc8);    /* 24 */
GG(a, b, c, d, x[9], S21, 0x21e1cde6);    /* 25 */
GG(d, a, b, c, x[14], S22, 0xc33707d6);   /* 26 */
GG(c, d, a, b, x[3], S23, 0xf4d50d87);    /* 27 */
GG(b, c, d, a, x[8], S24, 0x455a14ed);    /* 28 */
GG(a, b, c, d, x[13], S21, 0xa9e3e905);   /* 29 */
GG(d, a, b, c, x[2], S22, 0xfcefa3f8);    /* 30 */
GG(c, d, a, b, x[7], S23, 0x676f02d9);    /* 31 */
GG(b, c, d, a, x[12], S24, 0x8d2a4c8a);   /* 32 */
// Round 3
HH(a, b, c, d, x[5], S31, 0xfffa3942);    /* 33 */
HH(d, a, b, c, x[8], S32, 0x8771f681);    /* 34 */
HH(c, d, a, b, x[11], S33, 0x6d9d6122);   /* 35 */
HH(b, c, d, a, x[14], S34, 0xfde5380c);   /* 36 */
HH(a, b, c, d, x[1], S31, 0xa4beea44);    /* 37 */
HH(d, a, b, c, x[4], S32, 0x4bdecfa9);    /* 38 */
HH(c, d, a, b, x[7], S33, 0xf6bb4b60);    /* 39 */
HH(b, c, d, a, x[10], S34, 0xbebfbc70);   /* 40 */
HH(a, b, c, d, x[13], S31, 0x289b7ec6);   /* 41 */
HH(d, a, b, c, x[0], S32, 0xeaa127fa);    /* 42 */
HH(c, d, a, b, x[3], S33, 0xd4ef3085);    /* 43 */
HH(b, c, d, a, x[6], S34, 0x4881d05);         /* 44 */
HH(a, b, c, d, x[9], S31, 0xd9d4d039);    /* 45 */
HH(d, a, b, c, x[12], S32, 0xe6db99e5);   /* 46 */
HH(c, d, a, b, x[15], S33, 0x1fa27cf8);   /* 47 */
HH(b, c, d, a, x[2], S34, 0xc4ac5665);    /* 48 */
// Round 4
II(a, b, c, d, x[0], S41, 0xf4292244);    /* 49 */
```

```
    II(d, a, b, c, x[7], S42, 0x432aff97);    /* 50 */
    II(c, d, a, b, x[14], S43, 0xab9423a7);   /* 51 */
    II(b, c, d, a, x[5], S44, 0xfc93a039);    /* 52 */
    II(a, b, c, d, x[12], S41, 0x655b59c3);   /* 53 */
    II(d, a, b, c, x[3], S42, 0x8f0ccc92);    /* 54 */
    II(c, d, a, b, x[10], S43, 0xffeff47d);   /* 55 */
    II(b, c, d, a, x[1], S44, 0x85845dd1);    /* 56 */
    II(a, b, c, d, x[8], S41, 0x6fa87e4f);    /* 57 */
    II(d, a, b, c, x[15], S42, 0xfe2ce6e0);   /* 58 */
    II(c, d, a, b, x[6], S43, 0xa3014314);    /* 59 */
    II(b, c, d, a, x[13], S44, 0x4e0811a1);   /* 60 */
    II(a, b, c, d, x[4], S41, 0xf7537e82);    /* 61 */
    II(d, a, b, c, x[11], S42, 0xbd3af235);   /* 62 */
    II(c, d, a, b, x[2], S43, 0x2ad7d2bb);    /* 63 */
    II(b, c, d, a, x[9], S44, 0xeb86d391);    /* 64 */
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    memset((char *) x, 0, sizeof(x));
}
void MD5_Init(MD5Context * context)
{
    context->count[0] = context->count[1] = 0;
    context->state[0] = 0x67452301;
    context->state[1] = 0xefcdab89;
    context->state[2] = 0x98badcfe;
    context->state[3] = 0x10325476;
}
void MD5_Update(MD5Context * context, unsigned char * buf, int len)
```

```
{
    unsigned int i, index, partLen;
    index = (unsigned int) ((context->count[0] >> 3) & 0x3F);
    if ((context->count[0] += ((unsigned int) len << 3)) < ((unsigned int)
len << 3))
    context->count[1]++;
    context->count[1] += ((unsigned int) len >> 29);
    partLen = 64 - index;
    if ((unsigned int)len >= partLen)
    {
       memcpy((char *) &context->buffer[index], (char *) buf, partLen);
       MD5_Transform(context->state, context->buffer);
       for (i = partLen; i + 63 < (unsigned int)len; i += 64)
           MD5_Transform(context->state, &buf[i]);
       index = 0;
    }
    else
    {
        i = 0;
    }
    memcpy((char *) &context->buffer[index], (char *) &buf[i], len - i);
}
void MD5_Final(MD5Context * context, unsigned char digest[16])
{
    unsigned char bits[8];
    unsigned int index, padLen;
    MD5_Encode(bits, context->count, 8);
    index = (unsigned int) ((context->count[0] >> 3) & 0x3f);
    padLen = (index < 56) ? (56 - index) : (120 - index);
    MD5_Update(context, PADDING, padLen);
```

```
    MD5_Update(context, bits, 8);

    MD5_Encode(digest, context->state, 16);

    memset((char *) context, 0, sizeof(*context));

}
```

- **my_crypto.c**

  说明：封装供 DM 加载的函数接口。

```
//my_crypto.c

#include <stdlib.h>

#include "crypto_engine.h"

#include "xor.h"

#include "hash.h"

#define  MIN_EXTERNAL_CIPHER_ID         5000

#define CYT_TYPE_UNKNOWN                0            //类型, 未知算法

#define CYT_TYPE_SYM_BLOCK_ENCRYPT      1            //类型, 分组对称加密算法

#define CYT_TYPE_SYM_STREAM_ENCRYPT     2            //类型, 流式对称加密算法

#define CYT_TYPE_ASYM_ENCRYPT           3            //类型, 非对称加密算法, 保留

#define CYT_TYPE_HASH                   4            //类型, 散列算法

#define KEY1 ((byte)(0XAA))

#define KEY2 ((byte)(0x55))

#define KEY3 ((int)(0xCCCCCCCC))

DllExport

ulint

cipher_get_count(

)

{

    return 4;

}


DllExport

dm_bool
```

```
cipher_get_info(
    ulint    seqno,
    ulint*   cipher_id,
    byte**   cipher_name,
    byte*    type,
    ulint*   blk_size,
    ulint*   kh_size
)
{
    switch (seqno)
    {
    case 1:
        *cipher_id   = MIN_EXTERNAL_CIPHER_ID;
        *cipher_name = "xor1";
        *type        =  CYT_TYPE_SYM_STREAM_ENCRYPT;
        *blk_size    =  0;
        *kh_size     =  sizeof(KEY1);
        break;
    case 2:
        *cipher_id   = MIN_EXTERNAL_CIPHER_ID + 1;
        *cipher_name = "xor2";
        *type        =  CYT_TYPE_SYM_STREAM_ENCRYPT;
        *blk_size    =  0;
        *kh_size     =  sizeof(KEY1);
        break;
    case 3:
        *cipher_id   = MIN_EXTERNAL_CIPHER_ID + 2;
        *cipher_name = "xor3";
        *type        =  CYT_TYPE_SYM_BLOCK_ENCRYPT;
        *blk_size    =  4;
```

```
        *kh_size     =  sizeof(KEY1);
        break;
    case 4:
        *cipher_id   = MIN_EXTERNAL_CIPHER_ID + 3;
        *cipher_name = "new_md5";
        *type        =  CYT_TYPE_HASH;
        *blk_size    =  0;
        *kh_size     =  16;
        break;
    default:
        return DM_FALSE;
    }
    return DM_TRUE;
}

DllExport
dm_bool
cipher_get_para(
       ulint  cipher_id,
       ulint  para_id,
       void*  value
){
   switch (cipher_id)
   {
   case MIN_EXTERNAL_CIPHER_ID:
        switch (para_id)
          {
             case CYT_PARA_WORKMODE:
                    *(byte*)value = 0;
                    break;
```

```
            case CYT_PARA_EXTENDSIZE:
                    *(int*)value = 0;
                    break;
            default:
                    return DM_FALSE;
        }
    break;
case MIN_EXTERNAL_CIPHER_ID+1:
    switch (para_id)
      {
          case CYT_PARA_WORKMODE:
                    *(byte*)value = 0;
                    break;
          case CYT_PARA_EXTENDSIZE:
                    *(int*)value = 0;
                    break;
          default:
                    return DM_FALSE;
      }
    break;
case MIN_EXTERNAL_CIPHER_ID+2:
    switch (para_id)
      {
          case CYT_PARA_WORKMODE:
                    *(byte*)value = 0;
                    break;
          case CYT_PARA_EXTENDSIZE:
                    *(int*)value = 4;
                    break;
          default:
```

```
                    return DM_FALSE;
          }
        break;
    case MIN_EXTERNAL_CIPHER_ID+3:
        switch (para_id)
          {
              case CYT_PARA_WORKMODE:
                    *(byte*)value = 0;
                    break;
              case CYT_PARA_EXTENDSIZE:
                    *(int*)value = 0;
                    break;
              default:
                    return DM_FALSE;
          }
        break;
    default:
        return DM_FALSE;
}
    return DM_TRUE;
}

DllExport
dm_bool
cipher_encrypt_init(
    ulint    inner_id,
    byte*    key,
    ulint    key_size,
    void**   encrypt_para
)
```

```
{
    switch (inner_id)
    {
        case MIN_EXTERNAL_CIPHER_ID:
             break;
        case MIN_EXTERNAL_CIPHER_ID + 1:
             break;
        case MIN_EXTERNAL_CIPHER_ID + 2:
             break;
        default:
             return DM_FALSE;
    }
    return DM_TRUE;
}


DllExport
lint
cipher_get_cipher_text_size(
    ulint    inner_id,
    void*    cipher_para,
    lint     plain_text_size
)
{
    lint len = 0;
    lint block_len = sizeof(int);
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID:
             len = plain_text_size;
             break;
```

```
      case MIN_EXTERNAL_CIPHER_ID + 1:
            len = plain_text_size;
            break;
      case MIN_EXTERNAL_CIPHER_ID + 2:
            len = plain_text_size + block_len;
            break;
      default:
            return -1;
   }
   return len;
}

DllExport
lint
cipher_encrypt(
   ulint    inner_id,
   void*    cipher_para,
   byte*    plain_text,
   ulint    plain_text_size,
   byte*    cipher_text,
   ulint    cipher_text_buf_size
)
{
   int      len = plain_text_size;
   if (plain_text_size > cipher_text_buf_size)
   {
      return -1;
   }
   switch(inner_id)
   {
```

```
       case MIN_EXTERNAL_CIPHER_ID:
             len = xor_encrypt(KEY1, plain_text, plain_text_size, cipher_text,
cipher_text_buf_size);
              break;
       case MIN_EXTERNAL_CIPHER_ID + 1:
             len = xor_encrypt(KEY2, plain_text, plain_text_size, cipher_text,
cipher_text_buf_size);
              break;
       case MIN_EXTERNAL_CIPHER_ID + 2:
              len = xor_data_block_encrypt(KEY3, plain_text, plain_text_size,
cipher_text, cipher_text_buf_size);
        break;
       default:
              return -1;
    }
    return len;
}


DllExport
dm_bool
cipher_decrypt_init(
    ulint    inner_id,
    byte*    key,
    ulint    key_size,
    void**   decrypt_para
)
{
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID:
```

```
            break;
      case MIN_EXTERNAL_CIPHER_ID + 1:
            break;
      case MIN_EXTERNAL_CIPHER_ID + 2:
            break;
      default:
            return DM_FALSE;
   }
   return DM_TRUE;
}

DllExport
lint
cipher_decrypt(
   ulint     inner_id,
   void*     decrypt_para,
   byte*     cipher_text,
   ulint     cipher_text_size,
   byte*     plain_text,
   ulint     plain_text_buf_size
)
{
    dm_bool   bRet = DM_FALSE;
    int       len = cipher_text_size;
    if (cipher_text_size > plain_text_buf_size)
    {
            return -1;
    }
    switch(inner_id)
    {
```

```
       case MIN_EXTERNAL_CIPHER_ID:
              len = xor_decrypt(KEY1, cipher_text, cipher_text_size,
plain_text, plain_text_buf_size);
              break;
       case MIN_EXTERNAL_CIPHER_ID + 1:
              len = xor_decrypt(KEY2, cipher_text, cipher_text_size,
plain_text, plain_text_buf_size);
              break;
       case MIN_EXTERNAL_CIPHER_ID + 2:
           len = xor_data_block_decrypt(KEY3, cipher_text, cipher_text_size,
plain_text, plain_text_buf_size);
              break;
       default:
              return -1;
    }
    return len;
}


DllExport
dm_bool
cipher_hash_init(
    ulint    inner_id,
    void**   hash_para
)
{
    MD5Context* md5_context = NULL;
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID + 3:
              *hash_para = malloc(sizeof(MD5Context));
```

```
        md5_context = (MD5Context*)(*hash_para);
              MD5_Init(md5_context);
              break;
       default:
              return DM_FALSE;
    }
    return DM_TRUE;
}


DllExport
void
cipher_hash_update(
    ulint    inner_id,
    void*    hash_para,
    byte*    msg,
    ulint    msg_size
)
{
    MD5Context* md5_context = NULL;
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID + 3:
        md5_context = (MD5Context*)(hash_para);
              MD5_Update(md5_context, msg, msg_size);
              break;
       default:
              return;
    }
    return;
}
```

```
DllExport
lint
cipher_hash_final(
    ulint    inner_id,
    void*    hash_para,
    byte*    digest,
    ulint    digest_buf_size
)
{
    lint len = 0;
    MD5Context* md5_context = NULL;
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID + 3:
             md5_context = (MD5Context*)(hash_para);
             MD5_Final(md5_context, digest);
        len = 16; // MD5 后为一个 bit 的摘要
             free(md5_context);
             break;
       default:
             return -1;
    }
    return len;
}

DllExport
void
cipher_cleanup(
    ulint    inner_id,
```

```
    void*     cipher_para
)
{
    switch (inner_id)
    {
       case MIN_EXTERNAL_CIPHER_ID:
              break;
       case MIN_EXTERNAL_CIPHER_ID + 1:
              break;
       case MIN_EXTERNAL_CIPHER_ID + 2:
              break;
       case MIN_EXTERNAL_CIPHER_ID + 3:
              break;
       default:
              return;
    }
    return;
}

DllExport
dm_bool
cipher_asym_sign(
    ulint           inner_id,
    byte*           prikey,             //私钥签名
    ulint           prikey_size,
    byte*           data,
    ulint           data_size,
    byte*           signdata,
    ulint*       signdata_buf_size
)
```

```
{
    return DM_FALSE;
}

DllExport
dm_bool
cipher_asym_verify(
    ulint    inner_id,
    byte*    pubkey,                    //公钥验签
    ulint    pubkey_size,
    byte*    data,
    ulint    data_size,
    byte*    signdata,
    ulint    signdata_size
)
{
    return DM_FALSE;
}

DllExport
dm_bool
crypto_login(
    void**   cipher_para,
    byte*   pin,
    ulint    pin_len
)
{
    return DM_FALSE;
}
```

```
DllExport
dm_bool
crypto_logout(
    void*     cipher_para
)
{
    return DM_FALSE;
}

DllExport
dm_bool
crypto_read_cert(
    void*            cipher_para,
    byte*            cert,
    ulint *     cert_len
)
{
    return DM_FALSE;
}

DllExport
dm_bool
cipher_gen_random(
    byte*     random,
    ulint     random_length
)
{
    return DM_FALSE;
}
```

```
DllExport
dm_bool
crypto_get_name(
    byte**    crypto_name,
    ulint*    len
)
{
    *crypto_name = "mycrypto";
    *len = 8;
    return DM_TRUE;
}

DllExport
dm_bool
crypto_get_type(
    ulint*    crypto_type
)
{
    *crypto_type = 0;
    return DM_TRUE;
}
```

- **Crypto_engine.h**

  说明：库接口说明文件。

```
#ifndef CRYPTO_ENGINE_H
#define CRYPTO_ENGINE_H
#ifndef dm_h
typedef int   lint;
typedef short sint;
typedef signed char tint;
typedef unsigned int ulint;
```

```
typedef unsigned short usint;

typedef unsigned char byte;

#ifdef _WIN64

#define dm_int3264 __int64

#define dm_uint3264 unsigned __int64

#else

#define dm_int3264 long

#define dm_uint3264 unsigned long

#endif

#endif //#ifndef dm_h

#ifdef WIN32

#define DllImport __declspec( dllimport )

#define DllExport __declspec( dllexport )

#else

#define DllImport

#define DllExport

#endif

#define  MIN_EXTERNAL_CIPHER_ID         5000

#define  CYT_TYPE_UNKNOWN                0                  //type, unknown

arithmetic

#define       CYT_TYPE_SYM_BLOCK_ENCRYPT          1               //type,

symmetry encrypt arithmetic

#define  CYT_TYPE_SYM_STREAM_ENCRYPT    2                   //type, flow

encrypt arithmetic

#define  CYT_TYPE_ASYM_ENCRYPT              3               //type,

unsymmetrical encrypt arithmetic, reserved

#define  CYT_TYPE_HASH                              4               //type,

hash arithmetic

#ifdef __cplusplus

extern "C" {
```

```
#endif
#ifdef WIN32
#define dm_bool long
#else
#define dm_bool unsigned int
#endif
#define DM_FALSE    0
#define DM_TRUE     1
#define CYT_PARA_WORKMODE   0
#define CYT_PARA_EXTENDSIZE 1
DllExport
ulint
cipher_get_count(
);
DllExport
dm_bool
cipher_get_info(
    ulint    seqno,
    ulint*   cipher_id,
    byte**   cipher_name,
    byte*    type,
    ulint*   blk_size,
    ulint*   kh_size
);
DllExport
dm_bool
cipher_encrypt_init(
       ulint  inner_id,
       byte*  key,
       ulint  key_size,
```

```
	void** encrypt_para
);
DllExport
lint
cipher_get_cipher_text_size(
	ulint  inner_id,
	void*  cipher_para,
	lint   plain_text_size
);
DllExport
lint
cipher_encrypt(
	ulint  inner_id,
	void*  encrypt_para,
	byte*  plain_text,
	ulint  plain_text_size,
	byte*  cipher_text,
	ulint  cipher_text_buf_size
);
DllExport
dm_bool
cipher_decrypt_init(
	ulint  inner_id,
	byte*  key,
	ulint  key_size,
	void** decrypt_para
);
DllExport
lint
cipher_decrypt(
```

```
   ulint     inner_id,
   void*     decrypt_para,
   byte*     cipher_text,
   ulint     cipher_text_size,
   byte*     plain_text,
   ulint     plain_text_buf_size
);
DllExport
dm_bool
cipher_hash_init(
       ulint  inner_id,
       void** hash_para
);
DllExport
void
cipher_hash_update(
       ulint  inner_id,
       void*  hash_para,
       byte*  msg,
       ulint  msg_size
);
DllExport
lint
cipher_hash_final(
       ulint  inner_id,
       void*  hash_para,
       byte*  digest,
       ulint  digest_buf_size
);
DllExport
```

```
void
cipher_cleanup(
       ulint  inner_id,
       void*  hash_para
);
#ifdef __cplusplus
}
#endif
#endif
```

第二步，在安装目录 bin 下创建一个 external_crypto_libs 文件夹。将第一步中生成的动态库 mycrypto.dll 放入该文件夹中。

第三步，重启服务器。

第四步，至此，可以使用自定义的算法了。也可以通过 V$EXTERNAL_CIPHERS 查看到。

```
SQL> select * from V$EXTERNAL_CIPHERS;
行号        ID          NAME    LIB       VALID
---------- ----------- ------- --------- -----
1          5000        xor1    mycrypto.dll Y
2          5001        xor2    mycrypto.dll Y
3          5002        xor3    mycrypto.dll Y
4          5003        new_md5 mycrypto.dll Y
```

9 资源限制

DM 支持在创建和修改用户时指定对用户进行资源限制，具体语法参考本文 2.2 节的<资源限制子句>介绍。

用户的资源限制管理有两种方式：

方式一，将用户关联到某个已存在的 PROFILE 对象上，之后用户的资源限制由关联的 PROFILE 统一管理配置，且无法再使用方式二中的 LIMIT <资源设置>设置资源设置项。

方式二，未关联 PROFILE 的用户使用 LIMIT <资源设置>设置资源设置项。

新建用户默认使用方式二管理资源设置，在关联到 PROFILE 时，之前已设置的资源限制项会立即失效。

资源限制用于限制用户对 DM 数据库系统资源的使用，表 9.1 列出了资源限制所包含的设置项内容。

表 9.1 DM 资源限制一览表

资源设置项	说明	最大值	最小值	缺省值
SESSION_PER_USER	在一个实例中，一个用户可以同时拥有的会话数量	32768	1	安全版本中默认值为 4096；其他版本中默认值为系统所能提供的最大值
CONNECT_TIME	一个会话连接、访问和操作数据库服务器的时间上限（单位：1 分钟。若配置了 ini 参数 RESOURCE_FLAG=1，则单位为 1 秒钟）	1440 分或 86400 秒（1 天）	1	无限制
CONNECT_IDLE_TIME	会话最大空闲时间（单位：1 分钟。若配置了 ini 参数 RESOURCE_FLAG=1，则单位为 1 秒钟）	1440 分或 86400 秒（1 天）	1	无限制
FAILED_LOGIN_ATTEMPS	将引起一个账户被锁定的连续注册失败的次数	100	1	3，系统预设管理员用户默认无限制
CPU_PER_SESSION	一个会话允许使用的 CPU 时间	31536000	1	无限制

	上限（单位：秒）	（365 天）		
CPU_PER_CALL	用户的一个请求能够使用的 CPU 时间上限（单位：秒）	86400（1 天）	1	无限制
READ_PER_SESSION	会话能够读取的总数据页数上限	2147483646	1	无限制
READ_PER_CALL	每个请求能够读取的数据页数	2147483646	1	无限制
MEM_SPACE	会话占有的私有内存空间上限（单位：MB）	2147483647	1	无限制
PASSWORD_LIFE_TIME	一个口令在其终止前可以使用的天数	365	1	无限制
PASSWORD_REUSE_TIME	一个口令在可以重新使用前必须经过的天数	365	1	无限制
PASSWORD_REUSE_MAX	一个口令在可以重新使用前必须改变的次数	32768	1	无限制
PASSWORD_LOCK_TIME	如果超过 FAILED_LOGIN_ATTEMPS 设置值，一个账户将被锁定的分钟数	1440（1 天）	1	1，系统预设管理员用户默认无限制
PASSWORD_GRACE_TIME	以天为单位的口令过期宽限时间，过期口令超过该期限后，禁止执行除修改口令以外的其他操作	30	1	10，系统预设管理员用户默认无限制

注意：

当 PASSWORD_GRACE_TIME 为 UNLIMITED 时不能设置 PASSWORD_LIFE_TIME。

除上述资源设置项外，DM 还支持用户 IP 地址限制和用户时间段限制。

IP 地址设置项包括 ALLOW_IP（允许的 IP 地址）和 NOT_ALLOW_IP（不允许的 IP 地址），具体的限制规则如下表所示。

表 9.2 IP 地址限制规则

IP 地址限制项		集合状态			
IP 地址集合	不允许的 IP 地址	空	N	空	N
	允许的 IP 地址	Y	空	空	Y
IP 限制结果		只允许属于集合 Y 的 IP 地址	只允许属于集合 N 以外的所有 IP 地址	允许所有 IP，即无任何 IP 地址限制	只允许属于 Y-N（差集）集合的 IP 地址

允许 IP 和禁止 IP 用于控制此登录是否可以从某个 IP 访问数据库，其中禁止 IP 优先。在设置 IP 时，设置的允许和禁止 IP 需要用双引号括起来，中间用逗号隔开，如"192.168.0.29", "192.168.0.30"，也可以利用*来设置网段，如"192.168.0.*"。

用户时间段设置项包括 ALLOW_DATETIME（允许的时间段）和 NOT_ALLOW_DATETIME（不允许的时间段），具体的限制规则如下表所示：

表 9.3 用户时间段限制规则

时间段限制项		集合状态			
时间段集合	不允许的时间段	空	N	空	N
	允许的时间段	Y	空	空	Y
时间段限制结果		只允许属于集合 Y 的时间段	只允许属于集合 N 以外的所有时间段	允许所有时间段，即无任何时间段限制	只允许属于 Y-N （差集）集合的时间段

允许时间段和禁止时间段用于控制此登录是否可以在某个时间段访问数据库，其中禁止时间段优先。设置的时间段中的日期和时间要分别用双引号括起来。在设置时间段时，有两种方式：

- 具体时间段，如 2016 年 1 月 1 日 8：30 至 2016 年 2 月 1 日 17：00；
- 规则时间段，如每周一 8：30 至每周五 17：00。

例 1 创建对失败登录次数进行控制的用户，如果用户失败的登录次数达到 3 次，这个用户账号将被锁定。

```
CREATE USER USER1 IDENTIFIED BY USER1PASSWORD
LIMIT FAILED_LOGIN_ATTEMPS 3, PASSWORD_LOCK_TIME 5;
```

例 2 创建对修改口令进行限制的用户，要求用户在 30 天内必须把口令修改过 5 次后，才能使用过去用过的口令。

```
CREATE USER USER2 IDENTIFIED BY USER2PASSWORD
LIMIT PASSWORD_REUSE_TIME 30, PASSWORD_REUSE_MAX 5;
```

例 3 创建对允许 IP 和允许时间段进行限制的用户，要求用户在 192.168.0.*网段进行登录，且访问时间必须为工作时段。

```
CREATE USER USER3 IDENTIFIED BY USER3PASSWORD
ALLOW_IP "192.168.0.*" ALLOW_DATETIME MON "8:30:00" TO FRI "17:30:00";
```

例 4 创建对允许 IP 和允许时间段进行限制的用户，要求用户在 192.168.0.29 或 192.168.0.30 IP 进行登录，且访问时间段为 2016 年 1 月 1 日 8：30 至 2016 年 12 月 31 日 17：30。

```
CREATE USER USER4 IDENTIFIED BY USER4PASSWORD
ALLOW_IP "192.168.0.29", "192.168.0.30" ALLOW_DATETIME  "2016-1-1" "8:30:30"
to "2016-12-12" "17:30:00";
```

10 客体重用

在普通的环境下，数据库客体（主要指数据库对象、数据文件、缓存区）回收后不做处理，直接分配给新来的请求，但是有些窃密者会利用这一点编写特殊的非法进程通过数据库管理系统的内存泄露来获取数据库系统的信息。

DM 提供了客体重用安全功能。为防止非法进程利用数据库客体的内存泄露来攻击数据库，DM 主要从内存和文件两个方面进行了处理：

- ✓ 内存重用：DM 从系统分配内存及释放内存时均对内存内容进行清零，以保证不利用内存中前一进程所残留内容，且不泄漏 DM 的内容给其他进程。
- ✓ 文件重用：DM 在系统生成、扩展及删除文件时，对文件内容也进行了清零。

DM 提供了 INI 参数 ENABLE_OBJ_REUSE 用来控制是否启用客体重用功能，将该参数置为 1 表示启用客体重用，0 表示不启用，缺省为 0。

系统管理员可通过查询 V$PARAMETER 动态视图查询 ENABLE_OBJ_REUSE 的当前值。

```
SELECT * FROM V$PARAMETER WHERE NAME='ENABLE_OBJ_REUSE';
```

ENABLE_OBJ_REUSE 为静态参数，管理员可通过使用 DM 系统过程 SP_SET_PARA_VALUE 来修改该参数值，也可以使用 DM 控制台 Console 工具对该参数进行配置，不过配置后都需要重新启动 DM 数据库服务器才能生效。

11 登录用户名密码增强加密

非通信加密的情况下，为了实现对登录用户名、密码的安全管理，达梦对登录消息中的用户名密码进行了增强加、解密处理。

具体实现分为四步：一是通过 dmkey 工具生成公钥文件和私钥文件。二是配置服务器参数，使用私钥对客户端发送过来的登录名和密码进行解密。三是配置客户端参数，使用公钥文件对登录用户名和密码进行加密。四，启动数据库服务器。

11.1 dmkey 工具的使用

dmkey 是一个 RSA 非对称加解密的公钥文件、私钥文件生成工具。公钥文件和私钥文件分别用于对登录消息中的用户名、密码进行加密和解密。

11.1.1 启动 dmkey

安装好 DM 数据库管理系统后，在安装目录的“bin”子目录下可找到 dmkey 执行文件。

启动操作系统的命令行窗口，进入“dmkey”所在目录，可以准备启动 dmkey 工具了。

dmkey 的使用需要指定必要的参数。为 dmkey 指定参数的格式为：

```
dmkey [ keyword=value [keyword=value ...] ]
```

常见的使用搭配有两种：

一，生成公钥和私钥。

```
dmkey.exe PATH=path PUBKEY=pubkey
```

PATH：指定生成公钥和私钥的所在目录(缺省为 dmkey.exe 所在的目录)；

PUBKEY：指定生成公钥的文件名（缺省为 dm_login.pubkey），私钥为固定文件名 dm_login.prikey。公钥文件为文本方式存储，私钥文件则为二进制存储。

例如：

```
>dmkey PATH=D:\aa pubkey=test.pubkey

DMKEY V8.1.0.137-Build(2019.02.14-102842)ENT

generate public key file D:\aa\test.pubkey success!

generate private key file D:\aa\dm_login.prikey success!
```

二，显示私钥的相关信息(文件版本号、私钥 Guid、私钥的生成时间、dmkey 工具的版本号)。

该信息主要用于校验公钥、私钥是否匹配。同一套公钥文件、私钥文件的 Guid 一样的。文件版本号只有私钥文件有，公钥文件没有。

```
dmkey.exe PRIKEY=prikey
```

PRIKEY：私钥文件 dm_login.prikey 所在路径。

例如：

```
>dmkey PRIKEY=D:\aa\dm_login.prikey

DMKEY V8.1.0.137-Build(2019.02.14-102842)ENT


File version           : 0

Guid                   : E829B15F16444E2A2376A1143D4E65AC

Generation time        : 2019-02-25 10:48:42

Generation tool version: V8.1.0.137
```

11.1.2 查看 dmkey 参数

dmkey 使用较为灵活，参数较多，用户可以使用“dmkey help”查看 dmkey 版本信息以及各参数的简单信息。

```
./dmkey HELP

DMKEY V8

version: 1-2-101-21.12.16-153499-10000-ENT

格式: dmkey.exe KEYWORD=value

例程:

dmkey.exe       path=C:\data\DAMENG pubkey=dm_login.pubkey

dmkey.exe       prikey=C:\data\DAMENG\dm_login.prikey

关键字          说明

-------------------------------------------------------------

PATH            生成公钥和私钥文件的目标目录

PUBKEY          要生成的公钥文件名，缺省为 dm_login.pubkey
```

PRIKEY	要打印信息的私钥文件路径名
HELP	打印帮助信息

注意： 指定 PRIKEY 时仅打印所指私钥文件的相关信息，否则生成公钥文件和名为 dm_login.prikey 的私钥文件

11.2 服务器端配置

首先，将私钥文件拷贝到 ini 参数 SYSTEM_PATH 指定的目录下。

其次，配置 INI 参数 FORCE_CERTIFICATE_ENCRYPTION=1，即非加密通讯的情况下，开启用户名密码强制证书加密。该参数有效值为 0、1，缺省值为 0，运行中无法更改。

该项配置成功后，启动数据库服务器时，服务器会对登录的用户名密码强制使用 SYSTEM_PATH 目录下的私钥文件进行解密。

11.3 客户端配置

客户端配置有两种方式：一种是通过 dm_svc.conf 文件进行配置，供所有用户使用；另一种是通过 DPI 进行配置，供 DPI 编程用户使用。使用 DPI 编程的用户可以使用第一种方式设置，也可以使用第二种方式设置，当两种方式不一致时，第二种方式优先。

11.3.1 dm_svc.conf

使用配置项 LOGIN_CERTIFICATE 指定登录加密用户名密码公钥所在的路径。该配置项可全局配置也可在某个服务名下配置，一旦配置即认为开启了客户端的证书加密用户名密码模式。

11.3.2 DPI

指定 con 上属性 DSQL_ATTR_LOGIN_CERTIFICATE，用于指定加密登录用户名密码的公钥所在的路径。该属性和 dm_svc.conf 中配置项 LOGIN_CERTIFICATE 功能一样。两者不一致时，DSQL_ATTR_LOGIN_CERTIFICATE 设置优先。

11.4 启动数据库服务器

全部配置完成后，启动数据库服务器。服务器启动时自动加载 SYSTEM_PATH 目录下的私钥文件，对登录的用户名密码使用私钥文件 dm_login.prikey 进行解密。

11.5 应用实例

下面通过一个具体的实例，展示如何配置登录用户名和密码的增强加、解密功能。

第一步，通过 dmkey 生成公钥文件、私钥文件。

```
>dmkey PATH=D:\DAMENG pubkey=test.pubkey

DMKEY V8.1.0.137-Build(2019.02.14-102842)ENT

generate public key file D:\DAMENG\test.pubkey success!

generate private key file D:\DAMENG\dm_login.prikey success!
```

第二步，首先将私钥文件 dm_login.prikey 拷贝到数据库 SYSTEM PATH 目录下，然后配置服务器 ini 参数 FORCE_CERTIFICATE_ENCRYPTION=1。

第三步，配置 dm_svc.conf 文件的 LOGIN_CERTIFICATE 配置项，指定公钥文件所在位置。dm_svc.conf 文件内容如下：

```
TIME_ZONE=(480)

LANGUAGE=(cn)

LOGIN_CERTIFICATE=(D:\DAMENG\test.pubkey)
```

第四步，启动数据库服务器。

至此，配置完毕。

12 登录用户名密码外部存储

为了避免用户登录数据库时直接接触数据库登录密码，DM 提供了一种登录用户名密码外部存储方式，系统管理员可将用户名和密码存储在外部加密的密码文件（wallet 文件）中，用户登录数据库时只需提供配置好的服务名便可成功登录数据库。

具体实现分为两步：一是创建 wallet 文件，用于存储服务名、用户名和登录密码。二是配置 dm_svc.conf 文件，设置 wallet 文件路径以及服务名对应的数据库连接地址（IP 和 PORT）。配置完成后用户便可直接通过服务名成功登录数据库。

以上数据库登录方式存在以下限制：

- LOGIN 命令不支持利用 wallet 文件登录数据库
- DM 管理工具和 JDBC 不支持利用 wallet 文件登录数据库

DM 提供了一个数据库密码管理工具 dmmkstore，系统管理员可通过 dmmkstore 工具创建、访问或修改 wallet 文件。

12.1 dmmkstore 工具的使用

dmmkstore 是 DM 数据库密码管理工具。系统管理员可以通过 dmmkstore 工具创建、访问或修改 wallet 文件，wallet 文件中存储有数据库登录信息。支持跨平台或跨编码下发 wallet 文件。

数据库登录信息在 wallet 文件中以凭据（credential）的格式存储。每个凭据包含三个条目（entry）：服务名、用户名、密码。系统管理员可以利用 dmmkstore 工具增加、删除或修改 wallet 文件中的凭据。

12.1.1 启动 dmmkstore

安装好 DM 数据库管理系统后，在安装目录的“bin”子目录下可找到 dmmkstore 执行文件。

启动操作系统的命令行窗口，进入 dmmkstore 所在目录，可以准备启动 dmmkstore 工具了。

dmmkstore 的使用需要指定必要的参数。为 dmmkstore 指定参数的格式为：

```
dmmkstore -wrl <wallet_location> {keyword [<value>]}
```

或

```
dmmkstore -help
```

-wrl：指定 wallet 文件路径。

keyword：参数关键字。关键字不区分大小写，多个参数之间使用空格间隔。

<value>：参数取值。参数关键字与参数取值之间使用空格间隔。

-help：显示帮助信息。

例如：

在/home/test/dmdbms/wallet 目录下创建一个 wallet 文件，文件密码为 Wallet_123。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -create
输入口令: Wallet_123
再次输入口令: Wallet_123
```

如果创建成功，则/home/test/dmdbms/wallet 目录下会生成一个加密的 dmwallet.prikey 文件。

在 wallet 文件中创建凭据：

```
./dmmkstore -wrl /home/test/dmdbms/wallet -createCredential dm_user01 USER01
123456789
输入 Wallet 口令: Wallet_123
Create credential DM.security.client.connect_string1
```

查看 wallet 文件中的凭据：

```
./dmmkstore -wrl /home/test/dmdbms/wallet -listCredential
输入 Wallet 口令: Wallet_123
List credential (index: connect_string username)
1: dm_user01 USER01
```

12.1.2 查看 dmmkstore 参数

dmmkstore 使用较为灵活，参数较多。用户可使用“-help”命令快速查看 dmmkstore 版本信息以及各参数信息。

```
./dmmkstore -help
DM Secret Store Tool: V8

version: 1-2-101-21.12.16-153499-10000-ENT
dmmkstore [-wrl wrl] [-create] [-list] [-viewEntry alias] [-modifyEntry alias
secret] [-createCredential connect_string username password] [-listCredential]
[-modifyCredential connect_string username password] [-deleteCredential
connect_string] [-cipherName name] [-hashName name] [-help]
```

12.1.3 dmmkstore 参数详解

下面将详细介绍 dmmkstore 工具各参数的功能以及使用方法。

12.1.3.1 -wrl

指定 wallet 文件路径。

语法如下：

```
dmmkstore -wrl <wallet_location>
```

<wallet_location>：wallet 文件路径。

该参数单独使用时无任何实际效果，必须搭配 dmmkstore 工具的其他参数（除-help 参数）一起使用。

当-wrl 参数与其他参数一起使用时，-wrl 参数必须作为第一个参数，否则报错。

12.1.3.2 -create

创建 wallet 文件。

语法如下：

```
dmmkstore -wrl <wallet_location> -create
```

创建 wallet 文件时，需要指定 wallet 文件的密码，该密码长度范围为 9~48，并且必须同时包含大写字母、小写字母、数字以及特殊字符。

创建成功后将在<wallet_location>指定路径下生成一个加密的dmwallet.prikey文件。若指定路径下已存在dmwallet.prikey文件，则报错。

例如：

在/home/test/dmdbms/wallet目录下创建一个wallet文件，文件密码为Wallet_123。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -create
输入口令：Wallet_123
再次输入口令：Wallet_123
```

12.1.3.3 -createCredential

在wallet文件中创建凭据。

语法如下：

```
dmmkstore -wrl <wallet_location> -createCredential <server_name> <username>
<password>
```

<server_name>：服务名。必须与dm_svc.conf文件中配置的服务名相匹配。

<username>：用户名。

<password>：密码。

若wallet文件中已存在服务名相同的凭据，则报错。

wallet文件最大为64M，若创建凭据时wallet文件大小超过64M，则报错。

由于修改wallet文件过程中可能发生错误导致wallet文件受到损坏，因此修改wallet文件之前系统会自动在wallet文件路径下生成一个备份文件dmwallet.prikey.bak，若wallet文件修改失败，用户可以利用备份文件手动恢复wallet文件；若wallet文件修改成功，则系统自动删除备份文件。

例如：

在wallet文件中创建凭据，服务名为dm_user01，用户名为USER01，登录密码为123456789。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -createCredential dm_user01 USER01
123456789
```

12.1.3.4 -listCredential

显示 wallet 文件中的凭据。

语法如下：

```
dmmkstore -wrl <wallet_location> -listCredential
```

执行结果中会显示凭据的序号、服务名、用户名。

例如：

查看/home/test/dmdbms/wallet 路径下 wallet 文件中的凭据。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -listCredential
输入 Wallet 口令: Wallet_123
List credential (index: connect_string username)
1: dm_user01 USER01
```

12.1.3.5 -modifyCredential

修改 wallet 文件中指定凭据的用户名和密码。

语法如下：

```
dmmkstore -wrl <wallet_location> -modifyCredential <server_name> <username>
<password>
```

<server_name>：待修改凭据的服务名。

<username>：修改后的用户名。

<password>：修改后的密码。

若 wallet 文件中不存在服务名为<server_name>的凭据，则报错。

例如：

将 wallet 文件中服务名为 dm_user01 的凭据的用户名改为 SYSDBA，密码改为 SYSDBA。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -modifyCredential dm_user01 SYSDBA
SYSDBA
输入 Wallet 口令: Wallet_123
Modify credential
Modify 1
```

12.1.3.6 -deleteCredential

删除 wallet 文件中指定的凭据。

语法如下：

```
dmmkstore -wrl <wallet_location> -deleteCredential <server_name>
```

<server_name>：待删除凭据的服务名。

若 wallet 文件中不存在服务名为<server_name>的凭据，则报错。

例如：

删除 wallet 文件中服务名为 dm_user01 的凭据。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -deleteCredential dm_user01
输入 Wallet 口令: Wallet_123
Delete credential
Delete 1
```

12.1.3.7 -list

显示 wallet 文件中所有凭据条目的别名。

语法如下：

```
dmmkstore -wrl <wallet_location> -list
```

凭据中各条目的别名如下表所示：

表 12.1 凭据中各条目的别名

条目	别名
服务名	DM.security.client.connect_string%d
用户名	DM.security.client.username%d
密码	DM.security.client.password%d

其中，%d 表示该条目所在凭据的序号。

例如：

查看 wallet 文件中的所有凭据。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -listCredential

输入 Wallet 口令: Wallet_123

List credential (index: connect_string username)

1: dm_user01 USER01

2: dm_admin SYSDBA
```

可以看到，此时 wallet 文件中共有两条凭据，服务名分别为 dm_user01 和 dm_admin，凭据序号分别为 1 和 2。

查看 wallet 文件中所有凭据条目的别名。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -list

输入 Wallet 口令: Wallet_123

DM 密钥存储条目:

DM.security.client.connect_string1

DM.security.client.connect_string2

DM.security.client.password1

DM.security.client.password2

DM.security.client.username1

DM.security.client.username2
```

12.1.3.8 -viewEntry

显示凭据中指定条目的值。

语法如下：

```
dmmkstore -wrl <wallet_location> -viewEntry <alias>
```

<alias>：指定条目的别名。条目别名不区分大小写，关于条目别名的详细介绍请参考 12.1.3.7 -list。

若指定的条目为密码，则将以明文形式显示密码。

若指定的条目不存在，则报错。

例如：

查看别名为 DM.security.client.password1 的条目的值。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -viewEntry

DM.security.client.password1

输入 Wallet 口令: Wallet_123

DM.security.client.password1 = 123456789
```

12.1.3.9 -modifyEntry

修改凭据中指定条目的值。

语法如下：

```
dmmkstore -wrl <wallet_location> -modifyEntry <alias> <value>
```

<alias>：指定条目的别名。条目别名不区分大小写，关于条目别名的详细介绍请参考 12.1.3.7 -list。

<value>：修改后条目的值。

若指定的条目不存在，则报错。

例如：

将别名为 DM.security.client.password1 的条目的值修改为“987654321”。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -modifyEntry

DM.security.client.password1 987654321

输入 Wallet 口令: Wallet_123
```

查看别名为 DM.security.client.password1 的条目的值。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -viewEntry

DM.security.client.password1

输入 Wallet 口令: Wallet_123

DM.security.client.password1 = 987654321
```

12.1.3.10 -cipherName

设置加密算法。

语法如下：

```
dmmkstore -wrl <wallet_location> -cipherName <cipher_name> -create
```

<cipher_name>：加密算法名称。默认使用 DESEDE_CFB 加密算法。

参数-cipherName 和-create 必须一起使用，且-cipherName 必须位于-create 的前面。

若-cipherName 参数指定的不是加密算法，或指定的加密算法为 NOPAD/EXTKEY 工作模式，则报错。

例如：

指定使用 AES256_CFB 加密算法创建 wallet 文件。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -cipherName AES256_CFB -create
输入口令：Wallet_123
再次输入口令：Wallet_123
```

12.1.3.11 -hashName

设置杂凑算法（哈希算法）。

语法如下：

```
dmmkstore -wrl <wallet_location> -hashName <hash_name> -create
```

<hash_name>：杂凑算法名称。默认使用 SHA512 杂凑算法。

参数-hashName 和-create 必须一起使用，且-hashName 必须位于-create 的前面。

若-hashName 参数指定的不是杂凑算法，则报错。

例如：

指定使用 SHA512 杂凑算法创建 wallet 文件。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -hashName SHA512 -create
输入口令：Wallet_123
再次输入口令：Wallet_123
```

12.1.3.12 -help

显示帮助信息。

语法如下：

```
dmmkstore -help
```

12.2 dm_svc.conf 文件配置

使用全局配置项 WALLET_LOCATION 指定 wallet 文件所在路径，并配置服务名对应的数据库连接地址（IP 和 PORT）。仅当全局配置项 WALLET_LOCATION 非空时，才可通过 wallet 文件登录数据库。

12.3 应用实例

下面列出一个完整的 dmmkstore 工具应用实例以供参考。

1. 用户准备

数据库中新建用户 user01 和 user02。

```
CREATE USER USER01 IDENTIFIED BY 123456789;
CREATE USER USER02 IDENTIFIED BY 987654321;
COMMIT;
```

2. 配置 dm_svc.conf 文件

dm_svc.conf 文件中的配置内容如下：

```
WALLET_LOCATION=(/home/test/dmdbms/wallet)
dm_user01=(192.168.100.163:5254)
dm_user02=(192.168.100.163:5254)
```

3. 创建 wallet 文件

在/home/test/dmdbms/wallet 路径下创建一个 wallet 文件，文件密码为 Wallet_123。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -create
输入口令: Wallet_123
```

```
再次输入口令: Wallet_123
```

4. 创建凭据

在 wallet 文件中创建两条凭据，服务名分别为 dm_user01 和 dm_user02。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -createCredential dm_user01 USER01
123456789
输入 Wallet 口令: Wallet_123
Create credential DM.security.client.connect_string1
./dmmkstore -wrl /home/test/dmdbms/wallet -createCredential dm_user02 USER02
987654321
输入 Wallet 口令: Wallet_123
Create credential DM.security.client.connect_string2
```

5. 查看凭据

查看 wallet 文件中的凭据。

```
./dmmkstore -wrl /home/test/dmdbms/wallet -listCredential
输入 Wallet 口令: Wallet_123
List credential (index: connect_string username)
1: dm_user01 USER01
2: dm_user02 USER02
```

6. 使用客户端工具连接数据库

disql 客户端工具通过 dm_user01 服务名登录数据库。

```
./disql /@dm_user01
服务器[192.168.100.163:5254]:处于普通打开状态
登录使用时间 : 9.232(ms)
disql V8
SQL> SELECT USER();

行号     USER()
---------- ------
1          USER01
```

关于各客户端工具登录数据库方法的详细介绍请参考各客户端工具使用手册。

12.4 dmmkstore 错误码汇编

运行 DM 数据库密码管理工具 dmmkstore 时，可能会就使用过程中的一些错误进行报错提示。

表 12.2 dmmkstore 错误码

代码	解释
-86500	未指定 wallet 文件的位置
-86501	命令无效
-86502	命令中缺少参数
-86503	无法创建目录
-86504	系统找不到指定的路径
-86505	系统指定的路径过长
-86506	系统找不到指定的文件
-86507	系统指定路径下已存在 wallet 文件
-86508	口令无效
-86509	口令不匹配
-86513	内存空间不足
-86514	写文件失败
-86515	无法创建文件
-86516	无法打开文件
-86517	无法删除文件
-86518	无法拷贝文件
-86519	无法截断文件
-86520	加密模块加载失败
-86521	生成杂凑值失败
-86522	杂凑值校验失败
-86523	加密失败
-86524	解密失败
-86525	命令中参数值无效
-86526	缺少必要命令
-86527	命令重复
-86528	无法关闭文件
-86529	文件校验失败
-86530	需要检查加密模块
-86531	wallet 文件验证失败
-86532	凭据已存在
-86533	凭据不存在

-86534	-cipherName 和-hashName 命令只能在-create 命令之前指定
-86535	wallet 文件最大为 64M
-86537	指定的别名不存在
-86538	转码失败

附录 1 “三权分立”预设角色权限列表

说明：表中所列权限均为在同一数据库类型中的权限，如 DBA 具有 SELECT ANY TABLE 的权限，但是不能查询 SYSAUDITOR.SYSAUDIT 表；而 DB_AUDIT_ADMIN 具有 CREATE USER 权限，创建的用户也只能是 AUDIT 类型用户。

预设角色	预设数据库权限
DBA	CREATE SESSION
	ALTER DATABASE
	RESTORE DATABASE
	CREATE USER
	ALTER USER
	DROP USER
	CREATE ROLE
	CREATE SCHEMA
	CREATE TABLE
	CREATE VIEW
	CREATE PROCEDURE
	CREATE SEQUENCE
	CREATE TRIGGER
	CREATE INDEX
	CREATE CONTEXT INDEX
	BACKUP DATABASE
	CREATE LINK
	CREATE REPLICATE
	CREATE PACKAGE
	CREATE SYNONYM
	CREATE PUBLIC SYNONYM
	ALTER REPLICATE
	DROP REPLICATE
	DROP ROLE
	ADMIN ANY ROLE
	ADMIN ANY DATABASE PRIVILEGE
	GRANT ANY OBJECT PRIVILEGE
	CREATE ANY SCHEMA
	DROP ANY SCHEMA
	CREATE ANY TABLE
	ALTER ANY TABLE
	DROP ANY TABLE

INSERT TABLE
INSERT ANY TABLE
UPDATE TABLE
UPDATE ANY TABLE
DELETE TABLE
DELETE ANY TABLE
SELECT TABLE
SELECT ANY TABLE
REFERENCES TABLE
REFERENCES ANY TABLE
DUMP TABLE
DUMP ANY TABLE
GRANT TABLE
GRANT ANY TABLE
CREATE ANY VIEW
ALTER ANY VIEW
DROP ANY VIEW
INSERT VIEW
INSERT ANY VIEW
UPDATE VIEW
UPDATE ANY VIEW
DELETE VIEW
DELETE ANY VIEW
SELECT VIEW
SELECT ANY VIEW
GRANT VIEW
GRANT ANY VIEW
CREATE ANY PROCEDURE
DROP ANY PROCEDURE
EXECUTE PROCEDURE
EXECUTE ANY PROCEDURE
GRANT PROCEDURE
GRANT ANY PROCEDURE
CREATE ANY SEQUENCE
ALTER ANY SEQUENCE
DROP ANY SEQUENCE
SELECT SEQUENCE
SELECT ANY SEQUENCE
GRANT SEQUENCE
GRANT ANY SEQUENCE
CREATE ANY TRIGGER
DROP ANY TRIGGER

CREATE ANY INDEX
ALTER ANY INDEX
DROP ANY INDEX
CREATE ANY CONTEXT INDEX
ALTER ANY CONTEXT INDEX
DROP ANY CONTEXT INDEX
CREATE ANY PACKAGE
DROP ANY PACKAGE
EXECUTE PACKAGE
EXECUTE ANY PACKAGE
GRANT PACKAGE
GRANT ANY PACKAGE
CREATE ANY LINK
DROP ANY LINK
CREATE ANY SYNONYM
DROP ANY SYNONYM
DROP PUBLIC SYNONYM
ADMIN REPLAY
ADMIN BUFFER
CREATE TABLESPACE
ALTER TABLESPACE
DROP TABLESPACE
ALTER ANY TRIGGER
CREATE MATERIALIZED VIEW
CREATE ANY MATERIALIZED VIEW
DROP ANY MATERIALIZED VIEW
ALTER ANY MATERIALIZED VIEW
SELECT MATERIALIZED VIEW
SELECT ANY MATERIALIZED VIEW
CREATE ANY DOMAIN
DROP ANY DOMAIN
CREATE DOMAIN
GRANT ANY DOMAIN
GRANT DOMAIN
USAGE ANY DOMAIN
USAGE DOMAIN
CREATE ANY CONTEXT
DROP ANY CONTEXT
GRANT ANY CONTEXT
COMMENT ANY TABLE
CREATE ANY DIRECTORY
DROP ANY DIRECTORY

	ADMIN JOB
	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
RESOURCE	CREATE SCHEMA
	CREATE TABLE
	CREATE VIEW
	CREATE PROCEDURE
	CREATE SEQUENCE
	CREATE TRIGGER
	CREATE INDEX
	CREATE CONTEXT INDEX
	CREATE LINK
	CREATE PACKAGE
	CREATE SYNONYM
	CREATE PUBLIC SYNONYM
	INSERT TABLE
	UPDATE TABLE
	DELETE TABLE
	SELECT TABLE
	REFERENCES TABLE
	DUMP TABLE
	GRANT TABLE
	INSERT VIEW
	UPDATE VIEW
	DELETE VIEW
	SELECT VIEW
	GRANT VIEW
	EXECUTE PROCEDURE
	GRANT PROCEDURE
	SELECT SEQUENCE
	GRANT SEQUENCE
	EXECUTE PACKAGE
	GRANT PACKAGE
	CREATE MATERIALIZED VIEW
	SELECT MATERIALIZED VIEW
	CREATE DOMAIN
	GRANT DOMAIN
	USAGE DOMAIN
PUBLIC	INSERT TABLE
	UPDATE TABLE
	DELETE TABLE

	SELECT TABLE
	REFERENCES TABLE
	GRANT TABLE
	INSERT VIEW
	UPDATE VIEW
	DELETE VIEW
	SELECT VIEW
	GRANT VIEW
	EXECUTE PROCEDURE
	GRANT PROCEDURE
	SELECT SEQUENCE
	GRANT SEQUENCE
	EXECUTE PACKAGE
	GRANT PACKAGE
	SELECT MATERIALIZED VIEW
	GRANT DOMAIN
	USAGE DOMAIN
	DUMP TABLE
DB_AUDIT_ADMIN	CREATE USER
	ALTER USER
	DROP USER
	AUDIT DATABASE
DB_AUDIT_OPER	AUDIT DATABASE
DB_AUDIT_PUBLIC	无
DB_POLICY_ADMIN	CREATE USER
	ALTER USER
	DROP USER
	LABEL DATABASE
DB_POLICY_OPER	LABEL_DATABASE
DB_POLICY_PUBLIC	无

附录 2 “四权分立”预设角色权限列表

说明：表中所列权限均为在同一数据库类型中的权限，如 DBA 具有 SELECT ANY TABLE 的权限，但是不能查询 SYSAUDITOR.SYSAUDIT 表；而 DB_AUDIT_ADMIN 具有 CREATE USER 权限，创建的用户也只能是 AUDIT 类型用户。

预设角色	预设数据库权限
DBA	ALTER DATABASE
	BACKUP DATABASE
	RESTORE DATABASE
	CREATE USER
	ALTER USER
	DROP USER
	CREATE ROLE
	DROP ROLE
	ADMIN ANY ROLE
	CREATE TABLESPACE
	ALTER TABLESPACE
	DROP TABLESPACE
	CREATE REPLICATE
	ALTER REPLICATE
	DROP REPLICATE
	VERIFY DATABASE
	ADMIN REPLAY
	ADMIN BUFFER
	ADMIN JOB
	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
RESOURCE	CREATE ROLE
	DROP ROLE
PUBLIC	无
DB_OBJECT_ADMIN	CREATE USER
	ALTER USER
	DROP USER
	CREATE ROLE
	DROP ROLE
	ADMIN ANY ROLE
	CREATE SCHEMA
	CREATE TABLE
	INSERT TABLE

	UPDATE TABLE
	DELETE TABLE
	SELECT TABLE
	REFERENCES TABLE
	DUMP TABLE
	GRANT TABLE
	CREATE VIEW
	INSERT VIEW
	UPDATE VIEW
	DELETE VIEW
	SELECT VIEW
	GRANT VIEW
	CREATE DOMAIN
	GRANT DOMAIN
	USAGE DOMAIN
	CREATE PROCEDURE
	EXECUTE PROCEDURE
	GRANT PROCEDURE
	CREATE SEQUENCE
	SELECT SEQUENCE
	GRANT SEQUENCE
	CREATE TRIGGER
	CREATE INDEX
	CREATE CONTEXT INDEX
	CREATE PACKAGE
	EXECUTE PACKAGE
	GRANT PACKAGE
	CREATE SYNONYM
	CREATE PUBLIC SYNONYM
	DROP PUBLIC SYNONYM
	CREATE LINK
	CREATE ANY CONTEXT
	DROP ANY CONTEXT
	GRANT ANY CONTEXT
	COMMENT ANY TABLE
	CREATE ANY DIRECTORY
	DROP ANY DIRECTORY
	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
DB_OBJECT_OPER	CREATE SCHEMA
	CREATE TABLE
	INSERT TABLE
	UPDATE TABLE

	DELETE TABLE
	SELECT TABLE
	REFERENCES TABLE
	DUMP TABLE
	GRANT TABLE
	CREATE VIEW
	INSERT VIEW
	UPDATE VIEW
	DELETE VIEW
	SELECT VIEW
	GRANT VIEW
	CREATE DOMAIN
	CREATE PROCEDURE
	EXECUTE PROCEDURE
	GRANT PROCEDURE
	CREATE SEQUENCE
	SELECT SEQUENCE
	GRANT SEQUENCE
	CREATE TRIGGER
	CREATE INDEX
	CREATE CONTEXT INDEX
	CREATE PACKAGE
	EXECUTE PACKAGE
	GRANT PACKAGE
	CREATE SYNONYM
	CREATE PUBLIC SYNONYM
	CREATE LINK
DB_OBJECT_PUBLIC	INSERT TABLE
	UPDATE TABLE
	DELETE TABLE
	SELECT TABLE
	REFERENCES TABLE
	DUMP TABLE
	GRANT TABLE
	INSERT VIEW
	UPDATE VIEW
	DELETE VIEW
	SELECT VIEW
	GRANT VIEW
	EXECUTE PROCEDURE
	GRANT PROCEDURE
	SELECT SEQUENCE
	GRANT SEQUENCE
	EXECUTE PACKAGE

	GRANT PACKAGE
DB_AUDIT_ADMIN	CREATE USER
	ALTER USER
	DROP USER
	AUDIT DATABASE
	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
DB_AUDIT_OPER	AUDIT DATABASE
DB_AUDIT_PUBLIC	无
DB_POLICY_ADMIN	CREATE USER
	ALTER USER
	DROP USER
	LABEL DATABASE
	CREATE PROFILE
	ALTER PROFILE
	DROP PROFILE
DB_POLICY_OPER	LABEL_DATABASE
DB_POLICY_PUBLIC	无

咨询热线：400-991-6599

技术支持：dmtech@dameng.com

官网网址：www.dameng.com

武汉达梦数据库股份有限公司
Wuhan Dameng Database Co.,Ltd.

地址：武汉市东湖新技术开发区高新大道999号未来科技大厦C3栋16—19层

16th-19th Floor, Future Tech Building C3, No.999 Gaoxin Road, Donghu New Tech Development Zone,Wuhan,Hubei Province,China

电话：(+86) 027-87588000　传真：(+86) 027-87588810