目录

第 1 章 结构化查询语言 DM_SQL 简介

结构化查询语言 SQL(Structured Query Language)是在 1974 年提出的一种关系数据库语言。由于 SQL 语言接近英语的语句结构，方便简洁、使用灵活、功能强大，倍受用户及计算机工业界的欢迎，被众多计算机公司和数据库厂商所采用，经各公司的不断修改、扩充和完善，SQL 语言最终发展成为关系数据库的标准语言。

SQL 的第一个标准是 1986 年 10 月由美国国家标准化组织(ANSI)公布的 ANSI X3.135-1986 数据库语言 SQL，简称 SQL-86，1987 年国际标准化组织(ISO)也通过了这一标准。以后通过对 SQL-86 的不断修改和完善，于 1989 年第二次公布了 SQL 标准 ISO/IEC 9075-1989(E)，即 SQL-89。1992 年又公布了 SQL 标准 ISO/IEC 9075：1992，即 SQL-92。最新的 SQL 标准是 SQL-3(也称 SQL-99)，1999 年作为 ISO/IEC 9075：1999《信息技术——数据库语言 SQL》发布。我国也相继公布了数据库语言 SQL 的国家标准。

SQL 成为国际标准以后，其影响远远超出了数据库领域。例如在 CAD、软件工程、人工智能、分布式等领域，人们不仅把 SQL 作为检索数据的语言规范，而且也把 SQL 作为检索图形、图象、声音、文字等信息类型的语言规范。目前，世界上大型的著名数据库管理系统均支持 SQL 语言，如 Oracle、Sybase、SQL Server、DB2 等。在未来相当长的时间里，SQL 仍将是数据库领域以至信息领域中数据处理的主流语言之一。

由于不同的 DBMS 产品，大都按自己产品的特点对 SQL 语言进行了扩充，很难完全符合 SQL 标准。目前在 DBMS 市场上已将 SQL 的符合率作为衡量产品质量的重要指标，并研制成专门的测试软件，如 NIST。目前，DM SQL-92 入门级和过渡级的符合率均达到 100%，并且部分支持更新的 SQL-99 标准。同时 DM 还兼容 Oracle 11g 和 SQL Server 2008 的部分语言特性。本章主要介绍 DM 系统所支持的 SQL 语言——DM_SQL 语言。

1.1 DM_SQL 语言的特点

DM_SQL 语言符合结构化查询语言 SQL 标准，是标准 SQL 的扩充。它集数据定义、数据查询、数据操纵和数据控制于一体，是一种统一的、综合的关系数据库语言。它功能强大，使用简单方便、容易为用户掌握。DM_SQL 语言具有如下特点：

1. 功能一体化

DM_SQL 的功能一体化表现在以下两个方面：

1) DM_SQL 支持多媒体数据类型，用户在建表时可直接使用。DM 系统在处理常规数据与多媒体数据时达到了四个一体化：一体化定义、一体化存储、一体化检索、一体化处理，最大限度地提高了数据库管理系统处理多媒体的能力和速度；
2) DM_SQL 语言集数据库的定义、查询、更新、控制、维护、恢复、安全等一系列操作于一体，每一项操作都只需一种操作符表示，格式规范，风格一致，简单方便，很容易为用户所掌握。

2. 两种用户接口使用统一语法结构的语言

DM_SQL 语言既是自含式语言，又是嵌入式语言。作为自含式语言，它能独立运行于联机交互方式。作为嵌入式语言，DM_SQL 语句能够嵌入到 C 和 C++语言程序中，将高级语言(也称主语言)灵活的表达能力、强大的计算功能与 DM_SQL 语言的数据处理功能相结

合，完成各种复杂的事务处理。而在这两种不同的使用方式中，DM_SQL 语言的语法结构是一致的，从而为用户使用提供了极大的方便性和灵活性。

3. 高度非过程化

DM_SQL 语言是一种非过程化语言。用户只需指出“做什么”，而不需指出“怎么做”，对数据存取路径的选择以及 DM_SQL 语句功能的实现均由系统自动完成，与用户编制的应用程序与具体的机器及关系 DBMS 的实现细节无关，从而方便了用户，提高了应用程序的开发效率，也增强了数据独立性和应用系统的可移植性。

4. 面向集合的操作方式

DM_SQL 语言采用了集合操作方式。不仅查询结果可以是元组的集合，而且一次插入、删除、修改操作的对象也可以是元组的集合，相对于面向记录的数据库语言(一次只能操作一条记录)来说，DM_SQL 语言的使用简化了用户的处理，提高了应用程序的运行效率。

5. 语言简洁，方便易学

DM_SQL 语言功能强大，格式规范，表达简洁，接近英语的语法结构，容易为用户所掌握。

1.2 保留字与标识符

标识符的语法规则兼容标准 GJB 1382A-9X，标识符分为正规标识符和定界标识符两大类。

正规标识符以字母、_、$、#或汉字开头，后面可以跟随字母、数字、_、$、#或者汉字，正规标识符的最大长度是 128 个英文字符或 64 个汉字。正规标识符不能是保留字。

正规标识符的例子：A，test1，_TABLE_B，表 1。

定界标识符的标识符体用双引号括起来时，标识符体可以包含任意字符，特别地，其中使用连续两个双引号转义为一个双引号。

定界标识符的例子："table"，"A"，"!@#$"。

保留字的清单参见附录 1。

1.3 DM_SQL 语言的功能及语句

DM_SQL 语言是一种介于关系代数与关系演算之间的语言，其功能主要包括数据定义、查询、操纵和控制四个方面，通过各种不同的 SQL 语句来实现。按照所实现的功能，DM_SQL 语句分为以下几种：

1. 用户、模式、基表、视图、索引、序列、全文索引、存储过程和触发器的定义和删除语句，基表、视图、全文索引的修改语句，对象的更名语句；
2. 查询(含全文检索)、插入、删除、修改语句；
3. 数据库安全语句。包括创建角色语句、删除角色语句，授权语句、回收权限语句，修改登录口令语句，审计设置语句、取消审计设置语句等。

在嵌入方式中，为了协调 DM_SQL 语言与主语言不同的数据处理方式，DM_SQL 语言引入了游标的概念。因此在嵌入方式下，除了数据查询语句(一次查询一条记录)外，还有几种与游标有关的语句：

1. 游标的定义、打开、关闭、拨动语句；

2. 游标定位方式的数据修改与删除语句。

为了有效维护数据库的完整性和一致性，支持 DBMS 的并发控制机制，DM_SQL 语言提供了事务的回滚(ROLLBACK)与提交(COMMIT)语句。同时 DM 允许选择实施事务级读一致性，它保证同一事务内的可重复读，为此 DM 提供用户多种手动上锁语句，和设置事务隔离级别语句。

1.4 DM_SQL 所支持的数据类型

数据类型是可表示值的集。值的逻辑表示是<字值>。值的物理表示依赖于实现。DM 系统具有 SQL-92 的绝大部分数据类型，以及部分 SQL-99 和 SQL Server 2000 的数据类型。

1.4.1 常规数据类型

1. 字符数据类型

CHAR 类型

语法：CHAR[(长度)]

功能：CHAR 数据类型指定定长字符串。在基表中，定义 CHAR 类型的列时，可以指定一个不超过 8188 的正整数作为字符长度，例如：CHAR(100)。如果未指定长度，缺省为 1。DM 确保存储在该列的所有值都具有这一长度。CHAR 数据类型的最大长度由数据库页面大小决定，字符类型最大长度和页面大小的对应关系请见下表 1.4.1。DM 支持按字节存放字符串。

表 1.4.1

数据库页面大小	最大长度
4K	1900
8K	3900
16K	8000
32K	8188

CHARACTER 类型

语法：CHARACTER[(长度)]

功能：与 CHAR 相同。

VARCHAR 类型/VARCHAR2 类型

语法：VARCHAR[(长度)]

功能：VARCHAR 数据类型指定变长字符串，用法类似 CHAR 数据类型，可以指定一个不超过 8188 的正整数作为字符长度，例如：VARCHAR (100)。如果未指定长度，缺省为 8188。在 DM 系统中，VARCHAR 数据类型的实际最大长度由数据库页面大小决定，具体最大长度算法如表 1.4.2。CHAR 同 VARCHAR 的区别在于前者长度不足时，系统自动填充空格，而后者只占用实际的字节空间。

表 1.4.2

数据库页面大小	实际最大长度
4K	1900
8K	3900
16K	8000

32K	8188

注：这个限制长度只针对建表的情况，在定义变量的时候，可以不受这个限制长度的限制。

VARCHAR2 类型和 VARCHAR 类型用法相同。

2. 数值数据类型

1. 精确数值数据类型

NUMERIC 类型

语法：NUMERIC[(精度 [, 标度])]

功能：NUMERIC数据类型用于存储零、正负定点数。其中：精度是一个无符号整数，定义了总的数字数，精度范围是1至38。标度定义了小数点右边的数字位数。一个数的标度不应大于其精度。例如：NUMERIC(4,1)定义了小数点前面3位和小数点后面1位，共4位的数字，范围在-999.9到999.9。所有NUMERIC数据类型，如果其值超过精度，DM会返回一个出错信息，如果超过标度，则多余的位会被截断。

如果不指定精度和标度，则正数表示范围为：$1 * 10^{-129}$ to $9.99...9 * 10^{126}$ ；负数表示范围为： $-1 * 10^{-129}$ to $-9.99...99 * 10^{126}$。

DECIMAL类型

语法：DECIMAL[(精度 [, 标度])]

功能：与 NUMERIC 相似。

DEC 类型

语法：DEC[(精度[, 标度])]

功能：与 DECIMAL 相同。

NUMBER 类型

语法：NUMBER[(精度[, 标度])]

功能：与 NUMERIC 相同。

INTEGER 类型

语法：INTEGER

功能：用于存储有符号整数，精度为 10，标度为 0。取值范围为：-2147483648 (-2^{31})～+2147483647(2^{31}-1)。

INT 类型

语法：INT

功能：与 INTEGER 相同。

BIGINT 类型

语法：BIGINT

功能：用于存储有符号整数,精度为 19，标度为 0。取值范围为：-9223372036854775808(-2^{63})～9223372036854775807(2^{63}-1)。

TINYINT 类型

语法：TINYINT

功能：用于存储有符号整数，精度为 3，标度为 0。取值范围为：-128 ～ +127。

BYTE 类型

语法：BYTE

功能：与 TINYINT 相似，精度为 3，标度为 0。

SMALLINT 类型

语法：SMALLINT

功能：用于存储有符号整数，精度为 5，标度为 0。取值范围为：-32768(-2^{15} ～

+32767(2^{15}-1)。

BINARY 类型

语法：BINARY[(长度)]

功能：BINARY 数据类型用来存储定长二进制数据。缺省长度为 1 个字节。最大长度由数据库页面大小决定，具体最大长度算法与 CHAR 类型的相同。BINARY 常量以 0x 开始，后面跟着数据的十六进制表示。例如 0x2A3B4058。

VARBINARY 类型

语法：VARBINARY[(长度)]

功能：VARBINARY 数据类型用来存储变长二进制数据，用法类似 BINARY 数据类型，可以指定一个不超过 8188 的正整数作为数据长度。缺省长度为 8188 个字节。VARBINARY 数据类型的实际最大长度由数据库页面大小决定，具体最大长度算法与 VARCHAR 类型的相同。

2. 近似数值数据类型

FLOAT 类型

语法：FLOAT[(精度)]

功能：FLOAT 是带二进制精度的浮点数。DM 中 FLOAT 的精度设置用于保证数据移植的兼容性，实际精度在达梦内部是固定的：FLOAT 精度为 24 或低于 24 的时候，达梦系统内部自动将其转化为 REAL，即二进制精度为 24，十进制精度为 7；当 FLOAT 精度大于 24 时，FLOAT 的二进制精度为 53，十进制精度为 15。如省略精度，则二进制精度为 53，十进制精度为 15。精确到一位有效数字情况下，取值范围为-1.7*10^{308} ～ 1.7*10^{308}。

DOUBLE 类型

语法：DOUBLE[(精度)]

功能：DOUBLE 是带二进制精度的浮点数。DM 中 DOUBLE 的精度设置用于保证数据移植的兼容性，实际二进制精度为 53，十进制精度为 15。如省略精度，则二进制精度为 53，十进制精度为 15。精确到一位有效数字情况下，取值范围为-1.7*10^{308} ～ 1.7*10^{308}。

REAL 类型

语法：REAL

功能：REAL 是带二进制的浮点数，但它不能由用户指定使用的精度，系统指定其二进制精度为 24，十进制精度为 7。取值范围-3.4*10^{38} ～ 3.4*10^{38}。

DOUBLE PRECISION 类型

语法：DOUBLE PRECISION

功能：该类型指明双精度浮点数，其二进制精度为 53，十进制精度为 15。取值范围-1.7*10^{308} ～ 1.7*10^{308}。

1.4.2 位串数据类型

BIT 类型

语法：BIT

功能：BIT 类型用于存储整数数据 1、0 或 NULL，只有 0 才转换为假，其他非空、非 0 值都会自动转换为真，可以用来支持 ODBC 和 JDBC 的布尔数据类型。DM 的 BIT 类型与 SQL SERVER2000 的 BIT 数据类型相似。

功能与 ODBC 和 JDBC 的 bool 相同。

1.4.3 日期时间数据类型

日期时间数据类型分为一般日期时间数据类型、时间间隔数据类型和时区数据类型三类，用于存储日期、时间和它们之间的间隔信息。

1. 一般日期时间数据类型

DATE 类型

语法：DATE

功能：DATE 类型包括年、月、日信息，定义了'0001-01-01'和'9999-12-31'之间任何一个有效的格里高利日期。

DM 支持 SQL92 标准或 SQL SERVER 的 DATE 字值，例如：DATE '1999-10-01'、'1999/10/01'或'1999.10.01'都是有效的 DATE 值，且彼此等价。年月日中第一个非 0 数值前的 0 亦可省略，例如'0001-01-01'等价于'1-1-1'。

TIME 类型

语法：TIME[(小数秒精度)]

功能：TIME 类型包括时、分、秒信息，定义了一个在'00:00:00.000000'和'23:59:59.999999'之间的有效时间。TIME 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 0。

DM 支持 SQL92 标准或 SQL SERVER 的 TIME 字值，例如：TIME '09:10:21', '09:10:21'或'9:10:21'都是有效的 TIME 值，且彼此等价。

TIMESTAMP 类型

语法：TIMESTAMP[(小数秒精度)]

功能：TIMESTAMP 类型包括年、月、日、时、分、秒信息，定义了一个在'0001-01-01 00:00:00.000000'和'9999-12-31 23:59:59.999999'之间的有效格里高利日期时间。TIMESTAMP 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。

DM 支持 SQL92 标准或 SQL SERVER 的 TIMESTAMP 字值，例如：TIMESTAMP '2002-12-12 09:10:21'或'2002-12-12 9:10:21'或'2002/12/12 09:10:21'或'2002.12.12 09:10:21' 都是有效的 TIMESTAMP 值，且彼此等价。

语法中，TIMESTAMP 也可以写为 DATETIME。

2. 时间间隔数据类型

DM 支持两类十三种时间间隔类型：两类是年-月间隔类和日-时间隔类，它们通过时间间隔限定符区分，前者结合了日期字段年和月，后者结合了时间字段日、时、分、秒。由时间间隔数据类型所描述的值总是有符号的。

需要说明的是，使用时间间隔数据类型时，如果使用了其引导精度的默认精度，要注意保持精度匹配，否则会出现错误。如果不指定精度，那么将使用默认精度 6。

1) 年-月间隔类

INTERVAL YEAR TO MONTH 类型

语法：INTERVAL YEAR［(引导精度)］TO MONTH

功能：描述一个若干年若干月的间隔，引导精度规定了年的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。月的取值范围在 0 到 11 之间。例如：INTERVAL YEAR(4) TO MONTH，其中 YEAR(4)表示年的精度为 4，表示范围为负 9999 年零 12 月到正 9999 年零 12 月。一个合适的字值例子是：INTERVAL '0015-08' YEAR TO MONTH。

INTERVAL YEAR 类型

语法：INTERVAL YEAR［(引导精度)］

功能：描述一个若干年的间隔，引导精度规定了年的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。例如：INTERVAL YEAR(4)，其中 YEAR(4)表示年的精度为 4，表示范围为负 9999 年到正 9999 年。一个合适的字值例子是：INTERVAL '0015' YEAR。

INTERVAL MONTH 类型

语法：INTERVAL MONTH［(引导精度)］

功能：描述一个若干月的间隔，引导精度规定了月的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。例如：INTERVAL MONTH(4)，其中 MONTH(4)表示月的精度为 4，表示范围为负 9999 月到正 9999 月。一个合适的字值例子是：INTERVAL '0015' MONTH。

2) 日-时间隔类

INTERVAL DAY 类型

语法：INTERVAL DAY［(引导精度)］

功能：描述一个若干日的间隔，引导精度规定了日的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。例如：INTERVAL DAY(3)，其中 DAY (3)表示日的精度为 3，表示范围为负 999 日到正 999 日。一个合适的字值例子是：INTERVAL '150' DAY。

INTERVAL DAY TO HOUR 类型

语法：INTERVAL DAY［(引导精度)］ TO HOUR

功能：描述一个若干日若干小时的间隔，引导精度规定了日的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。而时的取值范围在 0 到 23 之间。例如：INTERVAL DAY(1) TO HOUR，其中 DAY(1)表示日的精度为 1，表示范围为负 9 日零 23 小时到正 9 日零 23 小时。一个合适的字值例子是：INTERVAL '9 23' DAY TO HOUR。

INTERVAL DAY TO MINUTE 类型

语法：INTERVAL DAY［(引导精度)］ TO MINUTE

功能：描述一个若干日若干小时若干分钟的间隔，引导精度规定了日的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。而小时的取值范围在 0 到 23 之间，分钟的取值范围在 0 到 59 之间。例如：INTERVAL DAY(2) TO MINUTE，其中 DAY(2)表示日的精度为 2，表示范围为负 99 日零 23 小时零 59 分到正 99 日零 23 小时零 59 分。一个合适的字值例子是：INTERVAL '09 23:12' DAY TO MINUTE。

INTERVAL DAY TO SECOND 类型

语法：INTERVAL DAY［(引导精度)］ TO SECOND [(小数秒精度)]

功能：描述一个若干日若干小时若干分钟若干秒的间隔，引导精度规定了日的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果不定义小数秒精度默认精度为 6。小时的取值范围在 0 到 23 之间，分钟的取值范围在 0 到 59 之间。例如：INTERVAL DAY(2) TO SECOND(1)，其中 DAY(2)表示日的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 日零 23 小时零 59 分零 59.9 秒到正 99 日零 23 小时零 59 分零 59.9 秒。一个合适的字值例子是：INTERVAL '09 23:12:01.1' DAY TO SECOND。

INTERVAL HOUR 类型

语法：INTERVAL HOUR［(引导精度)］

功能：描述一个若干小时的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。例如：INTERVAL HOUR(3)，其中 HOUR (3)表示时的精度为 3，表示范围为负 999 小时到正 999 小时。一个合适的字值例子是：INTERVAL '150' HOUR。

INTERVAL HOUR TO MINUTE 类型

语法：INTERVAL HOUR［(引导精度)］ TO MINUTE

功能：描述一个若干小时若干分钟的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。而分钟的取值范围在 0 到 59 之间。例如：INTERVAL HOUR(2) TO MINUTE，其中 HOUR(2)表示小时的精度为 2，表示范围为负 99 小时零 59 分到正 99 小时零 59 分。一个合适的字值例子是：INTERVAL '23:12' HOUR TO MINUTE。

INTERVAL HOUR TO SECOND 类型

语法：INTERVAL HOUR［(引导精度)］ TO SECOND [(小数秒精度)]

功能：描述一个若干小时若干分钟若干秒的间隔，引导精度规定了小时的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。分钟的取值范围在 0 到 59 之间。例如：INTERVAL HOUR(2) TO SECOND(1)，其中 HOUR(2)表示小时的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 小时零 59 分零 59.9 秒到正 99 小时零 59 分零 59.9 秒。一个合适的字值例子是：INTERVAL '23:12:01.1' HOUR TO SECOND。

INTERVAL MINUTE 类型

语法：INTERVAL MINUTE［(引导精度)］

功能：描述一个若干分钟的间隔，引导精度规定了分钟的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。例如：INTERVAL MINUTE(3)，其中 MINUTE(3)表示分钟的精度为 3，表示范围为负 999 分钟到正 999 分钟。一个合适的字值例子是：INTERVAL '150' MINUTE。

INTERVAL MINUTE TO SECOND 类型

语法：INTERVAL MINUTE［(引导精度)］ TO SECOND [(小数秒精度)]

功能：描述一个若干分钟若干秒的间隔，引导精度规定了分钟的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。例如：INTERVAL MINUTE(2) TO SECOND(1)，其中 MINUTE(2)表示分钟的精度为 2，SECOND(1)表示秒的小数点后面取 1 位，表示范围为负 99 分零 59.9 秒到正 99 分零 59.9 秒。一个合适的字值例子是：INTERVAL '12:01.1' MINUTE TO SECOND。

INTERVAL SECOND 类型

语法：INTERVAL SECOND［(引导精度 [,小数秒精度])]

功能：描述一个若干秒的间隔，引导精度规定了秒整数部分的取值范围。引导精度取值范围为 0～9，如果未定义，缺省精度为 2。小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。例如：INTERVAL SECOND(2,1)，表示范围为负 99.9 秒到正 99.9 秒。一个合适的字值例子是：INTERVAL '51.1' SECOND。

3. 时区数据类型

DM 支持两种时区类型：带时区的 TIME 类型和带时区的 TIMESTAMP 类型。

1) TIME WITH TIME ZONE 类型

语法：TIME［(小数秒精度)］WITH TIME ZONE

功能：描述一个带时区的 TIME 值，其定义是在 TIME 类型的后面加上时区信息。TIME 值部分与 1.4.2 节中的描述一致，时区部分的实质是 INTERVAL HOUR TO MINUTE 类型，其值应在-12:59 与+13:00 之间。一个合适的字值例子是：’19:01:01.0000 +02:03’。 TIME 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 0。

DM 支持 SQL92 标准或 SQL SERVER 的 TIME 字值，例如：TIME '09:10:21 +8:00',

'09:10:21 +8:00'或'9:10:21 +8:00'都是有效的 TIME WITH TIME ZONE 值，且彼此等价。

例如：

```
CREATE TABLE T1 (C1 TIME(2) WITH TIME ZONE);
INSERT INTO T1 VALUES(TIME '09:10:21 +8:00');
```

2) TIMESTAMP WITH TIME ZONE 类型

语法：TIMESTAMP［(小数秒精度)］WITH TIME ZONE

功能：描述一个带时区的 TIMESTAMP 值，其定义是在 TIMESTAMP 类型的后面加上时区信息。TIMESTMP 值部分与 1.4.2 节中的描述一致，时区部分的实质是 INTERVAL HOUR TO MINUTE 类型，其值应在-12:59 与+13:00 之间。一个合适的字值例子是：’2009-10-11 19:03:05.0000 -02:10’。TIMESTAMP 类型的小数秒精度规定了秒字段中小数点后面的位数，取值范围为 0～6，如果未定义，缺省精度为 6。

DM 支持 SQL92 标准或 SQL SERVER 的 TIMESTAMP 字值，例如：TIMESTAMP '2002-12-12 09:10:21 +8:00'或'2002-12-12 9:10:21 +8:00'或'2002/12/12 09:10:21 +8:00'或'2002.12.12 09:10:21 +8:00' 都是有效的 TIMESTAMP WITH TIME ZONE 值，且彼此等价。

语法中，TIMESTAMP 也可以写为 DATETIME。

例如：

```
CREATE TABLE T2(C1 TIMESTAMP(2) WITH TIME ZONE);
INSERT INTO T2 VALUES(TIMESTAMP '2002-12-12 09:10:21 +8:00');
```

1.4.4 多媒体数据类型

1. TEXT 类型

语法：TEXT

功能：TEXT 为变长字符串类型。其字符串的长度最大为 2G-1。DM 利用它存储长的文本串。

2. LONGVARCHAR(又名 TEXT)类型

语法：LONGVARCHAR

功能：与 TEXT 相同。

3. IMAGE 类型

语法：IMAGE

功能：IMAGE 用于指明多媒体信息中的图像类型。图像由不定长的象素点阵组成，长度最大为 2G-1 字节。该类型除了存储图像数据之外，还可用于存储任何其它二进制数据。

4. LONGVARBINARY(又名 IMAGE)类型

语法：LONGVARBINARY

功能：与 IMAGE 相同。

5. BLOB 类型

语法：BLOB

功能：BLOB 类型用于指明变长的二进制大对象，长度最大为 2G-1 字节。

6. CLOB 类型

语法：CLOB

功能：CLOB 类型用于指明变长的字母数字字符串，长度最大为 2G-1 字节。

注：多媒体数据类型的字值有两种格式，一种类似于字符串类型，例如：’ABCD’，一种类似于 BINARY，如 0x61626364；不支持为多媒体数据类型的字段指定精度。

1.5 DM_SQL 语言支持的表达式

DM 支持多种类型的表达式，包括数值表达式、字符串表达式、时间值表达式、时间间隔值表达式等。本节中引用的数据库实例请参见第 2 章。

1.5.1 数值表达式

1. 一元算符 + 和 -

语法：+exp 、-exp

(exp 代表表达式，下同)

当单独使用时，+ 和 – 代表表达式的正负号。

例

```
select -(-5), +NOWPRICE from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

5 6.1000

注：在 SQL 中由于两短横即“--”，表示“注释开始”，则双负号必须是-(-5)，而不是--5。

2. 一元算符 ~

语法：~exp

按位非算符，要求参与运算的操作数都为整数数据类型。

例

```
select ~10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

-11

3. 二元算符 +、-、*、/

语法：exp1+exp2 、exp1- exp2 、exp1*exp2 、exp1/exp2

(exp1 代表表达式 1，exp2 代表表达式 2，下同)

当在表达式之间使用+、-、*、/ 时，分别表示加、减、乘、除运算。对于结果的精度规定如下：

1) 只有精确数值数据类型的运算

两个相同类型的整数运算的结果类型不变，两个不同类型的整数运算的结果类型转换为范围较大的那个整数类型。

整数与 NUMERIC，DEC 等类型运算时，SMALLINT 类型的精度固定为 5，标度为 0；INTEGER 类型的精度固定为 10，标度为 0；BIGINT 类型的精度固定为 19，标度为 0。

exp1+exp2 结果的精度为二者整数部分长度(即精度－标度)的最大值与二者标度的最大值之和，标度是二者标度的最大值；

exp1-exp2 结果的精度为二者整数部分长度的最大值与二者标度的最大值之和，标度是二者标度的最大值；

exp1*exp2 结果的精度为二者的精度之和，标度也是二者的标度之和；

exp1/exp2 结果的标度为下面两个值中的较大者：

值一：exp1 标度 + exp2 精度 - exp2 标度 + 1。

值二：exp2 标度 + 1。

exp1/exp2 结果的精度为：上面求得的标度 + exp1 精度 - exp1 标度 + exp2 标度。

例

```
select NOWPRICE+1, NOWPRICE-1 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

7.1 5.1

例

```
select NOWPRICE*10, NOWPRICE/10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

61 0.61

2) 有近似数值数据类型的运算

对于 exp1+exp2 、exp1- exp2 、exp1*exp2 、exp1/exp2 中 exp1 和 exp2 只要有一个为近似数值数据类型，则结果为近似数值数据类型。

例

```
select NOWPRICE+3, NOWPRICE-3, NOWPRICE*3, NOWPRICE/3 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

9.1000 3.1000 18.3000 2.0333

3) 只有字符数据类型的运算

当 exp1 和 exp2 为字符串的时候，仅支持+运算符，用途为字符串连接，结果为字符串类型。

例

```
select 'asd' + 'zxc';
```

结果是：

asdzxc

4. 二元算符 &

```
语法：exp1 & exp2
```

按位与算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 & 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

0

5. 二元算符 |

```
语法：exp1 | exp2
```

按位或算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 | 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

30

6. 二元算符 ^

```
语法：exp1 ^ exp2
```

按位异或算符，要求参与运算的操作数都为整数数据类型。

例

```
select 20 ^ 10 from PRODUCTION.PRODUCT WHERE NOWPRICE<10;
```

结果是：

30

1.5.2 字符串表达式

连接 ||

```
语法：STR1 || STR2
```

(STR1 代表字符串 1，STR2 代表字符串 2)

连接操作符对两个运算数进行运算，其中每一个都是对属于同一字符集的字符串的求值。它以给定的顺序将字符串连接在一起，并返回一个字符串。其长度等于两个运算数长度之和。如果两个运算数中有一个是 NULL，则 NULL 等价为空串。

例

```
select '武汉' || ADDRESS1 from PERSON.ADDRESS WHERE ADDRESSID=3;
```

结果是：

武汉青山区青翠苑 1 号

1.5.3 时间值表达式

时间值表达式的结果为时间值类型，包括日期(DATE)类型，时间(TIME)类型和时间戳(TIMESTAMP)间隔类型。DM SQL 不是对于任何的日期时间和间隔运算数的组合都可以计算。如果任何一个运算数是 NULL，运算结果也是 NULL。下面列出了有效的可能性和结果的数据类型。

1. 日期+间隔，日期-间隔和间隔+日期，得到日期

日期表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的日期，表达式将出错。参与运算的间隔类型只能是 INTERVAL YEAR、INTERVAL MONTH、INTERVAL YEAR TO MONTH、INTERVAL DAY。

如果间隔运算数是年-月间隔，则没有从运算数的 DAY 字段的进位。

例

```
select PUBLISHTIME + INTERVAL '1' YEAR, PUBLISHTIME - INTERVAL '1' YEAR
from PRODUCTION.PRODUCT
where PRODUCTID=1;
```

结果是：

2006-04-01 2004-04-01

2. 时间+间隔，时间-间隔和间隔+时间，得到时间

时间表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的时间，表达式将出错。参与运算的间隔类型只能是 INTERVAL DAY、INTERVAL HOUR、INTERVAL MINUTE、INTERVAL SECOND、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE TO SECOND。

当结果的小时值大于等于 24 时，时间表达式是对 24 模的计算。

例

```
SELECT TIME '19:00:00'+INTERVAL '9' HOUR,
TIME '19:00:00'-INTERVAL '9' HOUR;
```

结果是：

04:00:00 10:00:00

3. 时间戳记+间隔，时间戳记-间隔和间隔+时间戳记，得到时间戳记

时间戳记表达式的计算是根据有效格里高利历日期的规则。如果结果是一个无效的时间戳记，表达式将出错。参与运算的间隔类型只能是 INTERVAL YEAR、INTERVAL MONTH、INTERVAL YEAR TO MONTH、INTERVAL DAY、INTERVAL HOUR、INTERVAL MINUTE、INTERVAL SECOND、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE TO SECOND。

与时间的计算不同，当结果的小时值大于等于 24 时，结果进位到天。

例

```
SELECT TIMESTAMP'2007-07-15 19:00:00'+INTERVAL'9'HOUR,
TIMESTAMP'2007-07-15 19:00:00'-INTERVAL'9'HOUR;
```

结果是：

2007-07-16 04:00:00　　2007-07-15 10:00:00

注：在含有 SECOND 值的运算数之间的一个运算的结果具有等于运算数的小数秒精度的小数秒精度。

4. 日期+数值，日期-数值和数值+日期，得到日期

日期与数值的运算，等价于日期与一个 INTERVAL ‘数值’ DAY 的时间间隔的运算。

例

```
SELECT CURDATE();
```

结果是：

2011-09-29 /* 假设该查询操作发生在 2011 年 9 月 29 日 */

例

```
SELECT CURDATE() + 2;
```

结果是：

2011-10-01

例

```
SELECT CURDATE() - 100;
```

结果是：

2011-06-21

5. 时间戳记+数值，时间戳记-数值和数值+时间戳记，得到时间戳记

时间戳记与数值的运算，将数值看作以天为单位，转化为一个 INTERVAL DAY TO SECOND(6)的时间间隔，然后进行运算。

例 时间戳加上 2.358 天。

```
SELECT TIMESTAMP '2003-09-29 08:59:59.123' + 2.358;
```

结果是：

2003-10-01 17:35:30.323000

1.5.4 时间间隔值表达式

1. 日期-日期，得到间隔

由于得到的结果可能会是“年-月-日”间隔，而这是不支持的间隔类型，故要对结果强制使用语法：

(日期表达式-日期表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的日期字段决定。

例

```
SELECT (PUBLISHTIME-DATE'1990-01-01')YEAR TO MONTH
FROM PRODUCTION.PRODUCT
WHERE PRODUCTID=1;
```

结果是：

INTERVAL '15-3' YEAR(9) TO MONTH

2. 时间－时间，得到间隔

要对结果强制使用语法：

(时间表达式-时间表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的时间字段决定。

例

```
SELECT (TIME'19:00:00'-TIME'10:00:00') HOUR;
```

结果是：

INTERVAL '9' HOUR(9)

3. 时间戳记 - 时间戳记，得到间隔

要对结果强制使用语法：

(时间戳记表达式-时间戳记表达式)<时间间隔限定符>

结果由<时间间隔限定符>中最不重要的日期时间字段决定。

例

```
SELECT (TIMESTAMP '2007-07-15 19:00:00'- TIMESTAMP '2007-01-15 19:00:00') HOUR;
```

结果是：

INTERVAL '4344' HOUR(9)

4. 年月间隔 + 年月间隔和年月间隔 - 年月间隔，得到年月间隔

参加运算的两个间隔必须有相同的数据类型，若得出无效的间隔的表达式将出错。结果的子类型包括运算数子类型所有的域，关于结果的引导精度规定如下：

如果二者的子类型相同，则为二者引导精度的最大值；如果二者的子类型不同，则为与结果类型首字段相同的那个运算数的引导精度。

例

```
SELECT INTERVAL'2007-07'YEAR(4) TO MONTH + INTERVAL'7'MONTH,
INTERVAL'2007-07'YEAR(4) TO MONTH - INTERVAL'7'MONTH;
```

结果是：

INTERVAL '2008-2' YEAR(9) TO MONTH INTERVAL '2007-0' YEAR(9) TO MONTH

5. 日时间隔 + 日时间隔和日时间隔 - 日时间隔，得到日时间隔

参加运算的两个间隔必须有相同的数据类型，若得出无效的间隔表达式将出错。结果的子类型包含运算数子类型所有的域，结果的小数秒精度为两运算数的小数秒精度的最大值，关于结果的引导精度规定如下：

如果二者的子类型相同，则为二者引导精度的最大值；如果二者的子类型不同，则为与结果类型首字段相同的那个运算数的引导精度。

例

```
SELECT INTERVAL'7 15'DAY TO HOUR + INTERVAL'10:10'MINUTE TO SECOND,
INTERVAL'7 15'DAY TO HOUR - INTERVAL'10:10'MINUTE TO SECOND;
```

结果是：

INTERVAL '7 15:10:10.000000' DAY(9) TO SECOND(6)

INTERVAL '7 14:49:50.000000' DAY(9) TO SECOND(6)

6. 间隔 * 数字，间隔 / 数字和数字 * 间隔，得到间隔

若得出无效的间隔表达式将出错。结果的子类型、小数秒精度、引导精度和原间隔相同。

例

```
SELECT INTERVAL'10:10'MINUTE TO SECOND * 3,
INTERVAL'10:10'MINUTE TO SECOND / 3;
```

结果是：

INTERVAL '30:30.000000' MINUTE(9) TO SECOND(6)

INTERVAL '3:23.333333' MINUTE(9) TO SECOND(6)

1.5.5 运算符的优先级

当一个复杂的表达式有多个运算符时，运算符优先性决定执行运算的先后次序。运算符有下面这些优先等级(从高到低排列)。在较低等级的运算符之前先对较高等级的运算符进行求值。

```
( )
+(一元正)、-(一元负)、~(一元按位非)
*(乘)、/(除)
+(加)、||(串联)、-(减)
^(按位异)、&(按位与)、|(按位或)
```

1.6 DM_SQL 语言支持的数据库模式

DM_SQL 语言支持关系数据库的三级模式，外模式对应于视图和部分基表，模式对应于基表，基表是独立存在的表。一个或若干个基表存放于一个存贮文件中，存贮文件中的逻辑结构组成了关系数据库的内模式。DM_SQL 语言本身不提供对内模式的操纵语句。

视图是从基表或其它视图上导出的表，DM 只将视图的定义保存在数据字典中。该定义实际为一查询语句，再为该查询语句取一名字即为视图名。每次调用该视图时，实际上是执行其对应的查询语句，导出的查询结果即为该视图的数据。所以视图并无自己的数据，它是一个虚表，其数据仍存放在导出该视图的基表之中。当基表中的数据改变时，视图中查询的数据也随之改变，因此，视图象一个窗口，用户透过它可看到自己权限内的数据。视图一旦定义也可以为多个用户所共享，对视图作类似于基表的一些操作就象对基表一样方便。

综上所述，SQL 语言对关系数据库三级模式的支持如图 1.6.1 所示。

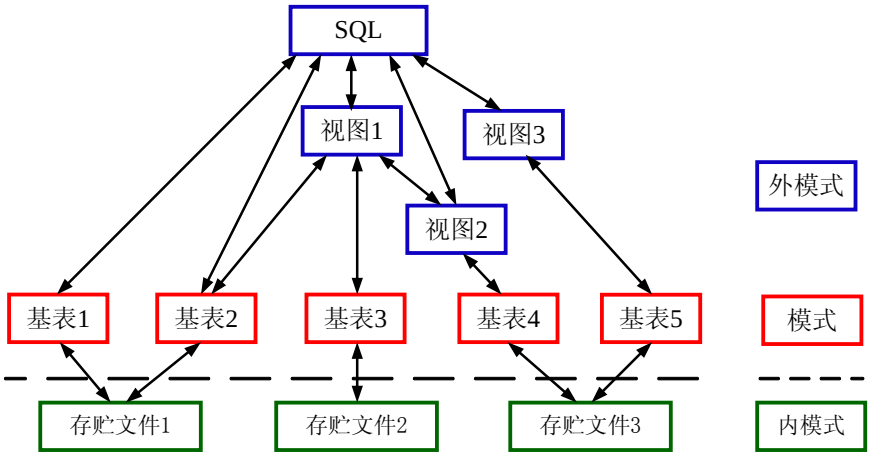


图 1.6.1 关系数据库的三级模式

第 2 章 手册中的实例说明

为方便读者阅读并尽快学会使用 DM 系统，本手册在介绍利用 DM 建立、维护数据库以及对数据库进行的各种操作中，使用了实例库 BOOKSHOP。本章将对该实例库进行说明。

2.1 实例库说明

实例库 BOOKSHOP 模拟武汉代理图书的某销售公司，该公司欲建立在线购物平台来拓展其代理产品的销售渠道，该在线购物平台支持网上产品信息浏览、订购等服务(仅限同城内销售及送货)。该销售公司的雇员、部门信息、业务方案等是所有实例的基础。

实例库 BOOKSHOP 所包含的模式、表如下：

表 2.1.1 BOOKSHOP

模式	包含相关对象	包含的表
RESOURCES	公司基本信息表	EMPLOYEE EMPLOYEE_ADDRESS DEPARTMENT EMPLOYEE_DREPARTMENT
PERSON	各个客户、雇员、供应商的名称和地址	ADDRESS ADDRESS_TYPE PERSON PERSON_TYPE
PRODUCTION	产品销售的信息	PRODUCT PRODUCT_CATEGORY PRODUCT_SUBCATEGORY PRODUCT_INVENTORY LOCATION PRODUCT_REVIEW PRODUCT_VENDOR
PURCHASING	供应商列表(采购订单等)	PURCHASEORDER_HEADER PURCHASEORDER_DETIAL VENDOR VENDOR_ADDRESS VENDOR_PERSON
SALES	与客户销售相关的数据(销售定单等)	CUSTMER CUSTMER_ADDRESS SALESORDER_HEADER SALESORDER_DETAIL SALESPERSON
OTHER	用到的其他表	DEPARTMENT EMPSALARY ACCOUNT

		ACTIONS READER READERAUDIT DEPTTAB EMPTAB SALGRADE COMPANYHOLIDAYS

EMPLOYEE

该公司的雇员信息表，包含在 RESOURCES 模式中。

表 2.1.2 EMPLOYEE

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	主键，自增列
NATIONALNO	VARCHAR(18)	非空	身份证号码
PERSONID	INT	非空	指向 PERSON.PERSONID 的外键
LOGINID	VARCHAR(256)	非空	用户登录 ID
TITLE	VARCHAR(50)	非空	职位
MANAGERID	INT	空	直接上司 ID，指 EMPLOYEE.EMPLOYEEID 的外键
BIRTHDATE	DATE	非空	出生日期
MARITALSTATUS	CHAR(1)	非空	S=未婚 M=已婚
PHOTO	IMAGE	空	照片
HAIRDATE	DATE	非空	入职时间

EMPLOYEE_ADDRESS

将雇员映射到 ADDRESS 表中的地址信息，包含在 RESOURCES 模式中。

表 2.1.3 EMPLOYEE_ADDRESS

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	指向 EMPLOYEE.EMPLOYEEID 的外键
ADDRESSID	INT	非空	指向 ADDRESS.ADDRESSID 的外键

EMPLOYEE_DEPARTMENT

将雇员映射到 DEPARTMENT 表中的部门信息，包含在 RESOURCES 模式中。

表 2.1.4 EMPLOYEE_DEPARTMENT

列	数据类型	是否为空	说明
EMPLOYEEID	INT	非空	指向 EMPLOYEE.EMPLOYEEID 的外键
DEPARTMENTID	INT	非空	员工所在部门，指向 DEPARTMENT. DEPARTMENTID 的外键
STARTDATE	DATE	非空	在该部门开始工作的日期
ENDDATE	DATE	空	离开该部门的日期 空=雇员在当前部门

DEPARTMENT

该公司的部门信息，包含在 RESOURCES 模式中。

表 2.1.5 DEPARTMENT

列	数据类型	是否为空	说明

DEPARTENTID	INT	非空	主键，自增列
NAME	VARCHAR(50)	非空	部门名称

PERSON

该公司所有雇员、供应商、客户的姓名信息表，包含在 PERSON 模式中。

表 2.1.6 PERSON

列	数据类型	是否为空	说明
PERSONID	INT	非空	主键，自增列，聚集索引
NAME	VARCHAR(50)	非空	姓名
SEX	CHAR(1)	非空	M=男 F=女
EMAIL	VARCHAR(50)	空	电子邮件地址
PHONE	VARCHAR(25)	空	电话

PERSON_TYPE

存储 PERSON 表中的联系人类型，客户中的联系人类型有采购经理、采购代表，供应商的联系人类型有销售经理、销售代表。包含在 PERSON 模式中。

表 2.1.7 PERSON_TYPE

列	数据类型	是否为空	说明
PERSON_TYPEID	INT	非空	主键，自增列
NAME	VARCHAR(50)	非空	联系人类型说明

ADDRESS

雇员、客户、供应商的地址信息，包含在 PERSON 模式中。

表 2.1.8 ADDRESS

列	数据类型	是否为空	说明
ADDRESSID	INT	非空	主键，自增列
ADDRESS1	VARCHAR(60)	非空	第一通讯地址
ADDRESS2	VARCHAR(60)	空	第二通讯地址
CITY	VARCHAR(30)	非空	市/县名称
POSTALCODE	VARCHAR(15)	非空	邮政编码

ADDRESS_TYPE

为雇员、客户、供应商定义地址类型的表，包含在 PERSON 模式中。

表 2.1.9 ADDRESS_TYPE

列	数据类型	是否为空	说明
ADDRESS_TYPEID	INT	非空	主键,自增列
NAME	VARCHAR(50)	非空	地址类型的说明如开票地址、家庭地址、送货地址、公司地址等

CUSTOMER

当前客户信息，包含在 SALES 模式中。

表 2.1.10 CUSTOMER

列	数据类型	是否为空	说明
CUSTOMERID	INT	非空	主键，自增列
PERSONID	INT	非空	指向 PERSON.PERSONID 的外键

CUSTOMER_ADDRESS

将客户映射到某个地址或多个地址,包含在 SALES 模式中。

表 2.1.11 CUSTOMER_ADDRESS

列	数据类型	是否为空	说明
CUSTOMERID	INT	非空	主键，指向 CUSTOMER.CUSTOMERID 的外键
ADDRESSID	INT	非空	主键，指向 ADDRESS.ADDRESSID 的外键
ADDRESS_TYPEID	INT	非空	地址类型，指向 ADDRESS_TYPE.ADDRESS_TYPEID 的外键

SALESPERSON

销售代表的销售统计信息，包含在 SALES 模式中。

表 2.1.12 SALESPERSON

列	数据类型	是否为空	说明
SALESPERSONID	INT	非空	主键, 自增列
EMPLOYEEID	INT	非空	该销售代表对应的雇员号,指向 EMPLOYEE.EMPLOYEEID 的外键
SALESTHISYEAR	DEC(19,4)	非空	今年到目前为止销售总额(万)
SALESLASTYEAR	DEC(19,4)	非空	去年销售总额(万)

VENDOR

所有供应商公司信息，包含在 PURCHASING 模式中。

表 2.1.13 VENDOR

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键, 自增列
ACCOUNTNO	VARCHAR(15)	非空	供应商账户号
NAME	VARCHAR(50)	非空	供应商名称
ACTIVEFLAG	BIT	非空	0=不再使用供应商提供的产品 1=正在使用供应商提供的产品 CHECK 约束
WEBURL	VARCHAR(1024)	空	供应商服务 URL
CREDIT	INT	非空	1=高级 2=很好 3=较好 4=一般 5=较差 CHECK 约束

VENDOR_ADDRESS

所有供应商地址信息，包含在 PURCHASING 模式中。

表 2.1.14 VENDOR_ADDRESS

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键, 指向 VENDOR.VENDORID 的外键
ADDRESSID	INT	非空	主键,指向 ADDRESS.ADDRESSID 外键
ADDRESS_TYPEID	INT	非空	地址类型,指向 ADDRESS_TYPE.ADDRESS_TYPEID 外键

VENDOR_PERSON

供应商联系人信息表。

表 2.1.15 VENDOR_PERSON

列	数据类型	是否为空	说明
VENDORID	INT	非空	主键,指向 VENDOR.VENDORID 外键
PERSONID	INT	非空	主键，指向 PERSON.PERSONID 外键
PERSON_TYPEID	INT	非空	联系人的类型，指向 PERSON_TYPE.PERSON_TYPEID 的外键

PRODUCT

该公司售出的所有产品(图书)，包含在 PRODUCTION 模式中。

表 2.1.16 PRODUCT

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	主键,自增列
NAME	VARCHAR(50)	非空	产品名称
AUTHOR	VARCHAR(25)	非空	作者
PUBLISHER	VARCHAR(50)	非空	出版社
PUBLISHTIME	DATE	非空	出版时间
PRODUCT_SUBCATEGORYID	INT	非空	产品所属子类别，PRODUCT_SUBCATEGORY .PRODUCT_SUBCATEGORYID 的外键
PRODUCTNO	VARCHAR(25)	非空	唯一产品标识号, unique 约束
DESCRIPTION	TEXT	空	产品的说明\简介
PHOTO	IMAGE	空	产品的照片
SATETYSTOCKLEVEL	SMALLINT	非空	最小库存量
ORIGINALPRICE	DEC(19,4)	非空	原价(初始价格)
NOWPRICE	DEC(19,4)	非空	当前销售价格
DISCOUNT	DECIMAL(2,1)	非空	折扣
TYPE	VARCHAR(5)	空	产品开本规格,如 16 开
PAPERTOTAL	INT	空	总页数
WORDTOTAL	INT	空	总字数
SELLSTARTTIME	DATE	非空	开始销售日期
SELLENDTIME	DATE	空	停止销售的日期

PRODUCT_CATEGORY

产品分类表，包含在 PRODUCTION 模式中。

表 2.1.17 PRODUCT_CATEGORY

列	数据类型	是否为空	说明
PRODUCT_CATEGORYID	INT	非空	产品类别 ID，主键,自增列
NAME	VARCHAR(50)	非空	产品类别名称

PRODUCT_SUBCATEGORY

产品子分类表，包含在 PRODUCTION 模式中。

表 2.1.18 PRODUCT_SUBCATEGORY

列	数据类型	是否为空	说明
PRODUCT_SUBCATEGORYID	INT	非空	产品子类别 ID，主键,自增列
PRODUCT_CATEGORYID	INT	非空	产品类别 ID
NAME	VARCHAR(50)	非空	产品类别名称

PRODUCT_INVENTORY

产品的库存信息，包含在 PRODUCTION 模式中。

表 2.1.19 PRODUCT_INVENTORY

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	产品标识，指向 PRODUCT.PRODUCTID 的外键
LOCATIONID	INT	非空	库存位置标识，指向 LOCATION.LOCATIONID 的外键
QUANTITY	INT	非空	库存位置的产品数量

PRODUCT_VENDOR

产品分类表，包含在 PRODUCTION 模式中。

表 2.1.20 PRODUCT_VENDOR

列	数据类型	是否为空	说明
PRODUCTID	INT	非空	主键,指向 PRODUCT.PRODUCTID 外键
VENDORID	INT	非空	主键,指向 VENDOR.VENDORID 外键
STANDARDPRICE	DEC(19,4)	非空	通常价格
LASTPRICE	DEC(19,4)	空	上次采购价格
LASTDATE	DATE	空	上次收到供应商产品的日期
MINQTY	INT	非空	应订购的最小定购数量
MAXQTY	INT	非空	应订购的最大定购数量
ONORDERQTY	INT	空	当前定购的数量

PRODUCT_REVIEW

已售产品的评论表，包含在 PRODCUTION 模式中。

表 2.1.21 PRODUCT_REVIEW

列	数据类型	是否为空	说明
PRODUCT_REVIEWID	INT	非空	主键,自增列
PRODUCTID	INT	非空	指向 PRODUCT.PRODUCTID 外键
NAME	VARCHAR(50)	非空	评论人姓名
REVIEWDATE	DATE	非空	提交评论的日期
EMAIL	VARCHAR(50)	非空	评论人的 EMAIL 地址
RATING	INT	非空	评论人给出的产品等级 范围 1-5，5 为最高级，CHECK 约束
COMMENTS	TEXT	空	评论人的注释

LOCATION

产品的库存地址信息，包含在 PRODUCTION 模式中。

表 2.1.22 LOCATION

列	数据类型	是否为空	说明
LOCATIONID	INT	非空	主键,自增列
NAME	VARCHAR(50)	非空	地点说明
PRODUCT_SUBCATEGORYID	INT	非空	指向 PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID)的外键

SALESORDER_HEADER

销售订单常规信息表，包含在 SALES 模式中。

表 2.1.23 SALESORDER_HEADER

列	数据类型	是否为空	说明
SALESORDERID	INT	非空	订单 ID，主键,自增列
ORDERDATE	DATE	非空	创建订单的日期
DUEDATE	DATE	非空	客户订单到期的日期
STATUS	TINYINT	非空	订单状态, CHECK 约束 1=待用户确认 2=待发货 3=已发货 4=已完成 5=意外终结
ONLINEORDERFLAG	BIT	非空	0=销售人员下的订单(包括电话订单) 1=在线订单
CUSTOMERID	INT	非空	客户标识号，指向 CUSTOMER.CUSTOMERID 的外键
SALESPERSONID	INT	非空	销售代表 ID，指向 SALESPERSON.SALESPERSONID 的外键
ADDRESSID	INT	非空	客户收货地址，指向 ADDRESS.ADDRESSID 外键
SHIPMETHOD	BIT	非空	发货方法 0=免费送货(仅限中心城区) 1=有偿送货(>=400 元，免费送货；<400,运费 10 元)
SUBTOTAL	DEC(19,4)	非空	销售小计 计算方式:SUM(SALESORDER_DETAIL.LINETOTAL)
FREIGHT	DEC(19,4)	非空	运费(非中城区,subtotal>=400,freight=0;<400, freight=10;中心城区, freight=0)
TOTAL	DEC(19,4)	非空	应付款总计 SUBTOTAL+FREIGHT
COMMENTS	TEXT	空	销售代表备注说明

SALESORDER_DETAIL

与特定销售订单关联的各个产品信息表，包含在 SALES 模式中。

表 2.1.24 SALESORDER_DETAIL

列	数据类型	是否为空	说明
SALESORDERID	INT	非空	主键，指向 SALESORDER_HEADER.SALESORDERID 的外键
SALESORDER_DETAILID	INT	非空	主键,确保数据唯一性的连续编号
CARRIERNO	VARCHAR(25)	非空	发货跟踪号
PRODUCTID	INT	非空	定购产品的 ID，指向 PRODUCT.PRODUCTID 的外键
ORDERQTY	INT	非空	每件产品定购数量
LINETOTAL	DEC(19,4)	非空	产品的销售小计

PURCHASEORDER_HEADER

采购订单常规信息，包含在 PURCHASING 模式中。

表 2.1.25 PURCHASEORDER_HEADER

列	数据类型	是否为空	说明
PURCHASEORDERID	INT	非空	订单 ID，主键,自增列
ORDERDATE	DATE	非空	创建订单的日期
STATUS	TINYINT	非空	订单状态, CHECK 约束 0=等待批准 1=已批准 2=已拒绝 3=已完成
EMPLOYEEID	INT	非空	创建采购订单的雇员 ID，指向 EMPLOYEE.EMPLOYEEID 的外键
VENDORID	INT	非空	所采购产品的供应商 ID，指向 VENDOR.VENDORID 的外键
SHIPMETHOD	VARCHAR(50)	非空	发货方法
SUBTOTAL	DEC(19,4)	非空	产品价格小计
TAX	DEC(19,4)	非空	税额
FREIGHT	DEC(19,4)	非空	运费
TOTAL	DEC(19,4)	非空	应向供应商付款总计(SUBTOTAL+ TAX+ FREIGHT)

PURCHASEORDER_DETAIL

与特定采购订单相关联的每个产品的采购信息，包含在 PURCHASING 模式中。

表 2.1.26 PURCHASEORDER_DETAIL

列	数据类型	是否为空	说明
PURCHASEORDERID	INT	非空	主键，指向 PURCHASEORDER_HEADER .PURCHASEORDERID 的外键
PURCHASEORDER_DETAILID	INT	非空	主键,确保数据唯一性的连续编号
DUEDATE	DATE	非空	希望从供应商收到产品的日期
PRODUCTID	INT	非空	定购产品的 ID，指向 PRODUCT.PRODUCTID 的外键
ORDERQTY	INT	非空	定购数量
PRICE	DEC(19,4)	非空	单件产品的价格
SUBTOTAL	DEC(19,4)	非空	产品价格小计(RPICE*ORDERQTY)
RECEIVEDQTY	DECIMAL(8,2)	非空	实际从供应商收到的数量
REJECTEDQTY	DECIMAL(8,2)	非空	检查时拒收的数量
STOCKEDQTY	DECIMAL(8,2)	非空	纳入库存的数量

2.2 参考脚本

2.2.1 创建实例库

--创建实例库

安装 DM 时，如果选择安装示例库，系统会自动安装一个名为 BOOKSHOP 的示例库。如果安装时没有选择安装示例库，可以通过 SQL 语句自行创建。

```
CREATE TABLESPACE BOOKSHOP DATAFILE 'BOOKSHOP.DBF' size 150;
```

2.2.2 创建模式及表

```
--创建模式
CREATE SCHEMA RESOURCES AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PERSON AUTHORIZATION SYSDBA;
/
CREATE SCHEMA SALES AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PRODUCTION AUTHORIZATION SYSDBA;
/
CREATE SCHEMA PURCHASING AUTHORIZATION SYSDBA;
/
CREATE SCHEMA OTHER AUTHORIZATION SYSDBA;
/

--创建表
--CREATE ADDRESS
      CREATE TABLE PERSON.ADDRESS
(ADDRESSID INT IDENTITY(1,1) PRIMARY KEY,
ADDRESS1 VARCHAR(60) NOT NULL,
ADDRESS2 VARCHAR(60),
CITY VARCHAR(30) NOT NULL,
POSTALCODE VARCHAR(15) NOT NULL) STORAGE (on BOOKSHOP);

--CREATE ADDRESS_TYPE
CREATE TABLE PERSON.ADDRESS_TYPE
(ADDRESS_TYPEID INT IDENTITY(1,1) PRIMARY KEY,
NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PERSON
CREATE TABLE PERSON.PERSON
(
```

```
    PERSONID    INT IDENTITY(1,1) CLUSTER PRIMARY KEY,
    SEX CHAR(1) NOT NULL,
    NAME VARCHAR(50) NOT NULL,
    EMAIL VARCHAR(50),
    PHONE VARCHAR(25)) STORAGE (ON BOOKSHOP);

    --CREATE PERSON_TYPE
    CREATE TABLE PERSON.PERSON_TYPE
    (PERSON_TYPEID INT IDENTITY(1,1) PRIMARY KEY,
    NAME VARCHAR(256) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE DEPARTMENT
    CREATE TABLE RESOURCES.DEPARTMENT(DEPARTMENTID INT IDENTITY(1,1) PRIMARY
KEY,NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE EMPLOYEE
    CREATE TABLE RESOURCES.EMPLOYEE(EMPLOYEEID INT IDENTITY(1,1) PRIMARY KEY ,
                NATIONALNO VARCHAR(18) NOT NULL,
                PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID),
                LOGINID VARCHAR(256) NOT NULL,
                TITLE VARCHAR(50) NOT NULL,
                MANAGERID INT,
                BIRTHDATE DATE NOT NULL,
                MARITALSTATUS CHAR(1) NOT NULL,
                PHOTO IMAGE,
                HAIRDATE DATE NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE EMPLOYEE_ADDRESS
    CREATE TABLE RESOURCES.EMPLOYEE_ADDRESS
    (ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
    EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID))
STORAGE (ON BOOKSHOP);

    --CREATE EMPLOYEE_DEPARTMENT
    CREATE TABLE RESOURCES.EMPLOYEE_DEPARTMENT
    (EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
                DEPARTMENTID INT NOT NULL REFERENCES
RESOURCES.DEPARTMENT(DEPARTMENTID),
                STARTDATE DATE NOT NULL,
                ENDDATE DATE) STORAGE (ON BOOKSHOP);

    --CREATE CUSTOMER
    CREATE TABLE SALES.CUSTOMER(CUSTOMERID INT IDENTITY(1,1) PRIMARY
KEY ,PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID)) STORAGE (ON
```

```
BOOKSHOP);

    --CREATE CUSTOMER_ADDRESS
    CREATE TABLE SALES.CUSTOMER_ADDRESS
    (CUSTOMERID INT REFERENCES SALES.CUSTOMER(CUSTOMERID),
    ADDRESSID INT REFERENCES PERSON.ADDRESS(ADDRESSID),
    ADDRESS_TYPEID INT NOT NULL REFERENCES PERSON.ADDRESS_TYPE(ADDRESS_TYPEID),
    PRIMARY KEY (CUSTOMERID,ADDRESSID)) STORAGE (ON BOOKSHOP);

    --CREATE PRODUCT_CATEGORY
    CREATE TABLE PRODUCTION.PRODUCT_CATEGORY
    (PRODUCT_CATEGORYID INT IDENTITY(1,1) PRIMARY KEY,
      NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE    PRODUCT_SUBCATEGORY
    CREATE TABLE PRODUCTION.PRODUCT_SUBCATEGORY
    (PRODUCT_SUBCATEGORYID INT IDENTITY(1,1) PRIMARY KEY,
    PRODUCT_CATEGORYID INT NOT NULL ,
    NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

    --CREATE PRODUCT
    CREATE TABLE PRODUCTION.PRODUCT
    (PRODUCTID INT IDENTITY(1,1) PRIMARY KEY,
      NAME VARCHAR(50) NOT NULL,
      AUTHOR VARCHAR(25) NOT NULL,
      PUBLISHER VARCHAR(50) NOT NULL,
      PUBLISHTIME DATE NOT NULL,
      PRODUCT_SUBCATEGORYID INT NOT NULL REFERENCES
PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID),
      PRODUCTNO VARCHAR(25) NOT NULL,
      SATETYSTOCKLEVEL SMALLINT NOT NULL,
      ORIGINALPRICE DEC(19,4) NOT NULL,
      NOWPRICE DEC(19,4) NOT NULL,
      DISCOUNT DECIMAL(2,1) NOT NULL,
      DESCRIPTION TEXT,
      PHOTO IMAGE,
      TYPE VARCHAR(5),
      PAPERTOTAL INT,
      WORDTOTAL INT,
      SELLSTARTTIME DATE NOT NULL,
      SELLENDTIME DATE,
      UNIQUE(PRODUCTNO)) STORAGE (ON BOOKSHOP);

    --CREATE LOCATION
```

```
CREATE TABLE PRODUCTION.LOCATION
(LOCATIONID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCT_SUBCATEGORYID INT NOT NULL REFERENCES
PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_SUBCATEGORYID),
NAME VARCHAR(50) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PRODUCT_INVENTORY
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PRODUCT_REVIEW
CREATE TABLE PRODUCTION.PRODUCT_REVIEW
(PRODUCT_REVIEWID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
NAME VARCHAR(50) NOT NULL,
REVIEWDATE DATE NOT NULL,
EMAIL VARCHAR(50) NOT NULL,
RATING INT NOT NULL CHECK(RATING IN(1,2,3,4,5)),
COMMENTS TEXT) STORAGE (ON BOOKSHOP);

--CREATE VENDOR
CREATE TABLE PURCHASING.VENDOR
(VENDORID INT IDENTITY(1,1) PRIMARY KEY,
ACCOUNTNO VARCHAR(15) NOT NULL,
NAME VARCHAR(50) NOT NULL,
ACTIVEFLAG BIT NOT NULL,
WEBURL VARCHAR(1024),
CREDIT INT NOT NULL CHECK(CREDIT IN(1,2,3,4,5))) STORAGE (ON BOOKSHOP);

--CREATE VENDOR_ADDRESS
CREATE TABLE PURCHASING.VENDOR_ADDRESS
(VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
ADDRESS_TYPEID INT NOT NULL REFERENCES PERSON.ADDRESS_TYPE(ADDRESS_TYPEID),
PRIMARY KEY (VENDORID,ADDRESSID)) STORAGE (ON BOOKSHOP);

--CREATE VENDOR_PERSON
CREATE TABLE PURCHASING.VENDOR_PERSON
(VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
PERSONID INT NOT NULL REFERENCES PERSON.PERSON(PERSONID),
PERSON_TYPEID INT NOT NULL REFERENCES PERSON.PERSON_TYPE(PERSON_TYPEID),
PRIMARY KEY (VENDORID,PERSONID)) STORAGE (ON BOOKSHOP);
```

```
--CREATE PRODUCT_VENDOR
CREATE TABLE PRODUCTION.PRODUCT_VENDOR
(PRODUCTID INT REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
  VENDORID INT REFERENCES PURCHASING.VENDOR(VENDORID),
  STANDARDPRICE DEC(19,4) NOT NULL,
  LASTPRICE DEC(19,4),
  LASTDATE DATE,
  MINQTY INT NOT NULL,
  MAXQTY INT NOT NULL,
  ONORDERQTY INT,
  PRIMARY KEY(PRODUCTID,VENDORID)) STORAGE (ON BOOKSHOP);

--CREATE SALESPERSON
CREATE TABLE SALES.SALESPERSON
(SALESPERSONID INT IDENTITY(1,1) PRIMARY KEY,
EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
SALESTHISYEAR DEC(19,4) NOT NULL,
SALESLASTYEAR DEC(19,4) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PURCHASEORDER_HEADER
CREATE TABLE PURCHASING.PURCHASEORDER_HEADER
(PURCHASEORDERID INT IDENTITY(1,1) PRIMARY KEY,
ORDERDATE DATE NOT NULL,
STATUS TINYINT NOT NULL CHECK(STATUS IN(0,1,2,3)),
EMPLOYEEID INT NOT NULL REFERENCES RESOURCES.EMPLOYEE(EMPLOYEEID),
VENDORID INT NOT NULL REFERENCES PURCHASING.VENDOR(VENDORID),
SHIPMETHOD VARCHAR(50) NOT NULL,
SUBTOTAL DEC(19,4) NOT NULL,
TAX DEC(19,4) NOT NULL,
FREIGHT DEC(19,4) NOT NULL,
TOTAL DEC(19,4) NOT NULL) STORAGE (ON BOOKSHOP);

--CREATE PURCHASEORDER_DETAIL
CREATE TABLE PURCHASING.PURCHASEORDER_DETAIL
(PURCHASEORDERID INT NOT NULL REFERENCES
PURCHASING.PURCHASEORDER_HEADER(PURCHASEORDERID),
PURCHASEORDER_DETAILID INT NOT NULL,
DUEDATE DATE NOT NULL,
PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
ORDERQTY INT NOT NULL,
PRICE DEC(19,4) NOT NULL,
SUBTOTAL DEC(19,4) NOT NULL,
RECEIVEDQTY INT NOT NULL,
```

```
    REJECTEDQTY INT NOT NULL,
    STOCKEDQTY INT NOT NULL,
    PRIMARY KEY(PURCHASEORDERID,PURCHASEORDER_DETAILID)) STORAGE (ON
BOOKSHOP);

    --CREATE SALESORDER_HEADER
    CREATE TABLE SALES.SALESORDER_HEADER
    (SALESORDERID INT IDENTITY(1,1) PRIMARY KEY,
    ORDERDATE DATE NOT NULL,
    DUEDATE DATE NOT NULL,
    STATUS TINYINT NOT NULL CHECK(STATUS IN(0,1,2,3,4,5)),
    ONLINEORDERFLAG BIT NOT NULL,
    CUSTOMERID INT NOT NULL REFERENCES SALES.CUSTOMER(CUSTOMERID),
    SALESPERSONID INT NOT NULL REFERENCES SALES.SALESPERSON(SALESPERSONID),
    ADDRESSID INT NOT NULL REFERENCES PERSON.ADDRESS(ADDRESSID),
    SHIPMETHOD BIT NOT NULL,
    SUBTOTAL DEC(19,4) NOT NULL,
    FREIGHT DEC(19,4) NOT NULL,
    TOTAL DEC(19,4) NOT NULL,
    COMMENTS TEXT) STORAGE (ON BOOKSHOP);

    --CREATE SALESORDER_DETAIL
    CREATE TABLE SALES.SALESORDER_DETAIL
    (SALESORDERID INT NOT NULL REFERENCES
    SALES.SALESORDER_HEADER(SALESORDERID),
    SALESORDER_DETAILID INT NOT NULL,
    CARRIERNO VARCHAR(25) NOT NULL,
    PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
    ORDERQTY INT NOT NULL,
    LINETOTAL DEC(19,4) NOT NULL,
    PRIMARY KEY(SALESORDERID,SALESORDER_DETAILID)) STORAGE (ON BOOKSHOP);

    --CREATE OTHER.DEPARTMENT
    CREATE TABLE OTHER.DEPARTMENT
    (
    HIGH_DEP VARCHAR(50),
    DEP_NAME VARCHAR(50)) STORAGE (ON BOOKSHOP);

    --CREATE OTHER.EMPSALARY
    CREATE TABLE OTHER.EMPSALARY
    (
    ENAME CHAR(10),
    EMPNO NUMERIC(4),
    SAL NUMERIC(4)) STORAGE (ON BOOKSHOP);
```

```
--CREATE OTHER.ACCOUNT
CREATE TABLE OTHER.ACCOUNT
(
"ACCOUNT_ID" INTEGER NOT NULL,
"BAL" DEC(10,2),
PRIMARY KEY("ACCOUNT_ID"))STORAGE (ON BOOKSHOP);

--CREATE OTHER.ACTIONS
CREATE TABLE OTHER.ACTIONS
(
"ACCOUNT_ID" INTEGER NOT NULL,
"OPER_TYPE" CHAR(1),
"NEW_VALUE" DEC(10,2),
"STATUS" VARCHAR(50),
PRIMARY KEY("ACCOUNT_ID"))STORAGE (ON BOOKSHOP);

--CREATE OTHER.READER
CREATE TABLE OTHER.READER
(
READER_ID     INT PRIMARY KEY,
NAME          VARCHAR(30),
AGE       SMALLINT,
GENDER        CHAR,
MAJOR         VARCHAR(30)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.READERAUDIT
CREATE TABLE OTHER.READERAUDIT
(
CHANGE_TYPE      CHAR NOT NULL,
CHANGED_BY VARCHAR(8) NOT NULL,
OP_TIMESTAMP     DATE NOT NULL,
OLD_READER_ID    INT,
OLD_NAME     VARCHAR(30),
OLD_AGE      SMALLINT,
OLD_GENDER  CHAR,
OLD_MAJOR    VARCHAR(30),
NEW_READER_ID    INT,
NEW_NAME     VARCHAR(30),
NEW_AGE          SMALLINT,
NEW_GENDER CHAR,
NEW_MAJOR    VARCHAR(30)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.DEPTTAB
```

```
CREATE TABLE OTHER.DEPTTAB
(
DEPTNO    INT PRIMARY KEY,
DNAME    VARCHAR(15),
LOC    VARCHAR(25)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.EMPTAB
CREATE TABLE OTHER.EMPTAB
(
EMPNO    INT PRIMARY KEY,
ENAME    VARCHAR(15) NOT NULL,
JOB    VARCHAR(10),
SAL    FLOAT,
DEPTNO    INT) STORAGE (on BOOKSHOP);

--CREATE OTHER.SALGRADE
CREATE TABLE OTHER.SALGRADE
(
LOSAL                FLOAT,
HISAL                FLOAT,
JOB_CLASSIFICATION VARCHAR(10)) STORAGE (ON BOOKSHOP);

--CREATE OTHER.COMPANYHOLIDAYS
CREATE TABLE OTHER.COMPANYHOLIDAYS
(
HOLIDAY    DATE) STORAGE (ON BOOKSHOP);
```

2.2.3 插入数据

```
--插入数据
--INSERT ADDRESS
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区 369 号金地太阳城 56-1-202','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区 369 号金地太阳城 57-2-302','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('青山区青翠苑 1 号','','武汉市青山区','430080');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('武昌区武船新村 115 号','','武汉市武昌区','430063');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('汉阳大道熊家湾 15 号','','武汉市汉阳区','430050');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区保利花园 50-1-304','','武汉市洪山区','430073');
```

```
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区保利花园 51-1-702','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 51-1-702','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 561 号','','武汉市江汉区','430023');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 555 号','','武汉市江汉区','430023');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('武昌区武船新村 1 号','','武汉市武昌区','430063');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('江汉区发展大道 423 号','','武汉市江汉区','430023');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 55-1-202','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 10-1-202','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区关山春晓 11-1-202','','武汉市洪山区','430073');
INSERT INTO PERSON.ADDRESS(ADDRESS1,ADDRESS2,CITY,POSTALCODE) VALUES('洪山区光谷软件园 C1_501','','武汉市洪山区','430073');

--INSERT ADDRESS_TYPE
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('发货地址');
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('送货地址');
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('家庭地址');
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('公司地址');

--INSERT DEPARTMENT
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('采购部门');
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('销售部门');
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('人力资源');
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('行政部门');
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('广告部');

--INSERT PERSON
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F',' 李 丽','lily@sina.com','02788548562');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','王刚','','02787584562');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','李勇','','02782585462');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','郭艳','','02787785462');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','孙丽','','13055173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','黄非','','13355173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王菲','','13255173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','张平','','13455173012');
```

```
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('M','张红','','13555173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','刘佳','','13955173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王南','','15955173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','李飞','','15954173012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','张大海','','15955673012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','王宇轩','','15955175012');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','桑泽恩','','15955173024');
INSERT INTO PERSON.PERSON(SEX,NAME,EMAIL,PHONE) VALUES('F','刘青','','15955173055');
INSERT INTO PERSON.PERSON(SEX,NAME,PHONE) VALUES('F','杨凤兰','02785584662');

--INSERT PERSON_TYPE
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('采购经理');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('采购代表');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售经理');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售代表');

--INSERT CUSTOMER
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='刘青'));
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='桑泽恩'));
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='王宇轩'));
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='张大海'));
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='李飞'));
INSERT INTO SALES.CUSTOMER(PERSONID) VALUES((SELECT PERSONID FROM
PERSON.PERSON WHERE NAME='王南'));

--INSERT CUSTOMER_ADDRESS
INSERT INTO SALES.CUSTOMER_ADDRESS(CUSTOMERID,ADDRESSID,ADDRESS_TYPEID)
SELECT CUSTOMERID,11,2 FROM SALES.CUSTOMER;
INSERT INTO SALES.CUSTOMER_ADDRESS(CUSTOMERID,ADDRESSID,ADDRESS_TYPEID)
SELECT CUSTOMERID,12,3 FROM SALES.CUSTOMER;

--INSERT EMPLOYEE
INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
VALUES('420921197908051523',1,'L1','总经理','','1979-08-05','S','','2002-05-02');

INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
```

```
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198008051523',2,'L2',' 销 售 经 理 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='总经理'),'1980-08-05','S','','2002-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198408051523',3,'L3',' 采 购 经 理 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='总经理'),'1981-08-05','S','','2002-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198208051523',4,'L4',' 销 售 代 表 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='销售经理'),'1982-08-05','S','','2002-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198308051523',5,'L5',' 销 售 代 表 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='销售经理'),'1983-08-05','S','','2002-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198408051523',6,'L6',' 采 购 代 表 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='采购经理'),'1984-08-05','S','','2005-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921197708051523',7,'L7',' 人 力 资 源 部 经 理 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='总经理'),'1977-08-05','M','','2002-05-02');

    INSERT INTO
RESOURCES.EMPLOYEE(NATIONALNO,PERSONID,LOGINID,TITLE,MANAGERID,BIRTHDATE,MARI
TALSTATUS,PHOTO,HAIRDATE)
    VALUES('420921198008071523',8,'L8',' 系 统 管 理 员 ',(SELECT EMPLOYEEID FROM
RESOURCES.EMPLOYEE WHERE TITLE='人力资源部经理'),'1980-08-07','S','','2004-05-02');

    --INSERT EMPLOYEE_DEPARTMENT
    INSERT INTO
RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'2','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE
```

```
RESOURCES.EMPLOYEE.TITLE='销售代表' OR RESOURCES.EMPLOYEE.TITLE='销售经理';
    INSERT INTO RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'1','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE RESOURCES.EMPLOYEE.TITLE='采购代表' OR RESOURCES.EMPLOYEE.TITLE='采购经理';
    INSERT INTO RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'3','2005-02-01',null FROM RESOURCES.EMPLOYEE WHERE RESOURCES.EMPLOYEE.TITLE='系统管理员';
    INSERT INTO RESOURCES.EMPLOYEE_DEPARTMENT(EMPLOYEEID,DEPARTMENTID,STARTDATE,ENDDATE)
    SELECT EMPLOYEEID,'4','2001-02-01',null FROM RESOURCES.EMPLOYEE WHERE RESOURCES.EMPLOYEE.TITLE='总经理';

    --INSERT EMPLOYEE_ADDRESS
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(1,1);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(2,2);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(3,3);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(4,4);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(5,5);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(6,6);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(7,7);
    INSERT INTO RESOURCES.EMPLOYEE_ADDRESS(EMPLOYEEID,ADDRESSID) VALUES(8,8);

    --INSERT SALES.SALESPERSON
    INSERT INTO SALES.SALESPERSON(EMPLOYEEID,SALESTHISYEAR,SALESLASTYEAR)
    SELECT EMPLOYEEID,8,10 FROM RESOURCES.EMPLOYEE WHERE TITLE='销售代表';

    --INSERT VENDOR
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','上海画报出版社','1','','2');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','长江文艺出版社','1','','2');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','北京十月文艺出版社','1','','1');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','人民邮电出版社','1','','1');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','清华大学出版社','1','','1');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','中华书局','1','','1');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
    VALUES('00','广州出版社','1','','1');
    INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
```

```
VALUES('00','上海出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','21 世纪出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','外语教学与研究出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','机械工业出版社','1','','1');
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT)
VALUES('00','文学出版社','1','','1');

--INSERT VENDOR_ADDRESS
INSERT INTO PURCHASING.VENDOR_ADDRESS(VENDORID,ADDRESSID,ADDRESS_TYPEID)
SELECT VENDORID,9,1 FROM PURCHASING.VENDOR;
INSERT INTO PURCHASING.VENDOR_ADDRESS(VENDORID,ADDRESSID,ADDRESS_TYPEID)
SELECT VENDORID,10,4 FROM PURCHASING.VENDOR;

--INSERT VENDOR_PERSON
INSERT INTO PURCHASING.VENDOR_PERSON(VENDORID,PERSONID,PERSON_TYPEID)
SELECT VENDORID,9,4 FROM PURCHASING.VENDOR;

--INSERT PRODUCT_CATEGORRY
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('小说');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('文学');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('计算机');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('英语');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('管理');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('少儿');
INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES('金融');

--INSERT PRODUCT_SUBCATEGORY
INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='小说'),'世界名著');
INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='小说'),'武侠');
INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='小说'),'科幻');
INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='小说'),'四大名著');
INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
```

```
WHERE NAME='小说'),'军事');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='小说'),'社会');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
VALUES(10,'历史');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'文集');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'纪实文学');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'文学理论');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'中国古诗词');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'中国现当代诗');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'戏剧');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='文学'),'民间文学');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'计算机理论');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'计算机体系结构');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'操作系统');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'程序设计');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'数据库');
```

```
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'软件工程');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'信息安全');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='计算机'),'多媒体');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语词汇');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语语法');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语听力');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语口语');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语阅读');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='英语'),'英语写作');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'行政管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'项目管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'质量管理与控制');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'商业道德');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'经营管理');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
```

```
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='管理'),'财务管理');

    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'幼儿启蒙');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'益智游戏');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'童话');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'卡通');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'励志');
    INSERT INTO PRODUCTION.PRODUCT_SUBCATEGORY(PRODUCT_CATEGORYID,NAME)
    VALUES((SELECT PRODUCT_CATEGORYID FROM PRODUCTION.PRODUCT_CATEGORY
WHERE NAME='少儿'),'少儿英语');

    --INSERT PRODUCT
    INSERT                                                    INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBC
ATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,T
YPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('红楼梦','曹雪芹，高鹗','中华书局','2005-4-1','9787101046120',(SELECT
PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='四大
名著'),'10','19','15.2','8.0','曹雪芹，是中国文学史上最伟大也是最复杂的作家，《红楼梦》也是中国文学史上
最伟大而又最复杂的作品。《红楼梦》写的是封建贵族的青年贾宝玉、林黛玉、薛宝钗之间的恋爱和婚姻悲
剧，而且以此为中心，写出了当时具有代表性的贾、王、史、薛四大家族的兴衰，其中又以贾府为中心，
揭露了封建社会后期的种种黑暗和罪恶，及其不可克服的内在矛盾，对腐朽的封建统治阶级和行将崩溃的
封建制度作了有力的批判，使读者预感到它必然要走向覆灭的命运。本书是一部具有高度思想性和高度艺
术性的伟大作品，从本书反映的思想倾向来看，作者具有初步的民主主义思想，他对现实社会包括宫廷及
官场的黑暗，封建贵族阶级及其家庭的腐朽，封建的科举制度、婚姻制度、奴婢制度、等级制度，以及与
此相适应的社会统治思想即孔孟之道和程朱理学、社会道德观念等等，都进行了深刻的批判并且提出了朦
胧的带有初步民主主义性质的理想和主张。这些理想和主张正是当时正在滋长的资本主义经济萌芽因素的
曲折反映。','','16','943','933000','2006-03-20','');

    INSERT                                                    INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBC
ATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,T
YPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
```

VALUES(' 水 浒 传 ',' 施 耐 庵 ， 罗 贯 中 ',' 中 华 书 局 ','2005-4-1','9787101046137',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='四大名著'),'10','19','14.3','7.5','《水浒传》是宋江起义故事在民间长期流传基础上产生出来的，吸收了民间文学的营养。《水浒传》是我国人民最喜爱的古典长篇白话小说之一。它产生于明代，是在宋、元以来有关水浒的故事、话本、戏曲的基础上，由作者加工整理、创作而成的。全书以宋江领导的农民起义为主要题材，艺术地再现了中国古代人民反抗压迫、英勇斗争的悲壮画卷。作品充分暴露了封建统治阶级的腐朽和残暴，揭露了当时尖锐对立的社会矛盾和“官逼民反”的残酷现实，成功地塑造了鲁智深、李逵、武松、林冲、阮小七等一批英雄人物。小说故事情节曲折，语言生动，人物性格鲜明，具有高度的艺术成就。但作品歌颂、美化宋江，鼓吹“忠义”和“替天行道”，表现出严重的思想局限。','','16','922','912000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 老 人 与 海 ',' 海 明 威 ',' 上 海 出 版 社 ','2006-8-1','9787532740093',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='世界名著'),'10','10','6.1','6.1','海明威(1899 一 1961)，美国著名作家、诺贝尔文学奖获得者。《老人与海》是他最具代表性的作品之一。','','16','98','67000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 射 雕 英 雄 传 (全 四 册)',' 金 庸 ',' 广 州 出 版 社 ','2005-12-1','9787807310822',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='武侠'),'10','32','21.7','6.8','自幼家破人亡的郭靖，随母流落蒙古大漠，这傻头傻脑但有情有义的小伙子倒也逝有福气，他不但习得了江南六怪的绝艺、全真教马钰的内功、洪七公的隆龙十八掌、双手互博之术、九阴真经等盖世武功，还让古灵精怪的小美女黄蓉这辈子跟定了他。这部原名『大漠英雄传』的小说是金庸小说中最广为普罗大众接受、传颂的一部，其中出了许多有名又奇特的人物，东邪西毒南帝北丐中神通，还有武功灵光、脑袋不灵光的老顽童周伯通，他们有特立独行的性格、作为和人生观，让人叹为观止。书中对历史多有着墨，中原武林及蒙古大漠的生活情形随着人物的生长环境变迁而有不同的叙述，异族统治之下的小老百姓心情写来丝丝入扣，本书对情的感觉是很含蓄的，尤其是郭靖与拖雷、华筝无猜的童年之谊，他与江南六怪的师生之谊等等，还有全真七子中长春子丘处机的侠义行为及其与郭杨二人风雪中的一段情谊，也有很豪气的叙述。神算子瑛姑及一灯大师和周伯通的一场孽恋，是最出乎人意料的一段，成人世界的恋情可比小儿女的青涩恋燕还复杂多了。郭靖以扭胜巧的人生经历和「为国为民，侠之大者」的儒侠风范，也是书中最大要旨。距离这本书完成的时间已有四十年了，书中的单纯诚朴的人物性格还深深的留在读者心中，本书故事也多改编成电影、电视剧等，受欢迎程度可见一斑。','','16','','1153000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 鲁 迅 文 集 (小 说 、 散 文 、 杂 文) 全 两 册 ',' 鲁 迅 ','','2006-9-1','9787509000724',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='文集

'),'10','39.8','20','5.0','','','16','684','680000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 长 征 ',' 王 树 增 ',' 人 民 文 学 出 版 社 ','2006-9-1','9787020057986',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='纪实文学'),'10','53','37.7','6.4','','','16','683','670000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 数 据 结 构 (C 语 言 版)(附 光 盘)',' 严 蔚 敏 ， 吴 伟 民 ',' 清 华 大 学 出 版 社 ','2007-3-1','9787302147510',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='计算机理论'),'10','30','25.5','8.5','《数据结构》(C 语言版)是为“数据结构”课程编写的教材，也可作为学习数据结构及其算法的 C 程序设计的参数教材。本书的前半部分从抽象数据类型的角度讨论各种基本类型的数据结构及其应用','','8','334','493000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES(' 工 作 中 无 小 事 ',' 陈 满 麒 ',' 机 械 工 业 出 版 社 ','2006-1-1','9787111182252',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='行政管理'),'10','16.8','11.4','6.8','本书立足于当今企业中常见的轻视小事，做事浮躁等现象，从人性的弱点这一独特角度，挖掘出员工轻视小事的根本原因，具有深厚的人文关怀，极易引起员工的共鸣。它有助于员工端正心态，摒弃做事贪大的浮躁心理，把小事做好做到位，从而提高整个企业的工作质量。当重视小事成为员工的一种习惯，当责任感成为一种生活态度，他们将会与胜任、优秀、成功同行，责任、忠诚、敬业也将不再是一句空洞的企业宣传口号。本书是一本提升企业竞争力、建设企业文化的指导手册，一本员工素质培训的完美读本，一本所有公务员、公司职员的必读书。','','8','152','70000','2006-03-20','');

INSERT INTO PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)

VALUES('突破英文基础词汇','刘毅','外语教学与研究出版社','2003-8-1','9787560035024',(SELECT PRODUCT_SUBCATEGORYID FROM PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语词汇'),'10','15.9','11.1','7.0','本书所列单词共计 1300 个，加上各词的衍生词、同义词及反义词，则实际收录约 3000 词，均为平时最常用、最容易接触到的单词。详细列出各词的国际音标、词性说明及中文解释，省却查字典的麻烦。每一课分为五个部分，以便于分段记忆。在课前有预备测验，每一部分之后有习题，课后有效果检测，可借助于重复测验来加深对单词的印象，并学习如何活用单词。','','8','350','','2006-03-20','');

```
    INSERT INTO
PRODUCTION.PRODUCT(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBC
ATEGORYID,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,DESCRIPTION,PHOTO,T
YPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
    VALUES('噼里啪啦丛书(全 7 册)','(日)佐佐木洋子','21 世纪出版社','1901-01-01','9787539125992',(SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='幼儿启蒙'),'10','58','42','6.1','噼里啪啦系列丛书包括：我要拉巴巴》《我去刷牙》《我要洗澡》《你好》《草莓点心》《车来了》《我喜欢游泳》共 7 册。 这是日本画家佐佐木洋子编绘的，分别描绘孩子在刷牙、洗澡、游玩、吃点心等各种时候所碰到的问题，以风趣的方式教会他们人生的最初的知识。书中的图形不仅夸张诱人，而且采用了一些局部折叠的方式，在书页中可以不时翻开一些折叠面，让人看到图画内部的东西，这是很符合低幼儿童的阅读心理的。','','8','','','2006-03-20','');

    --INSERT LOCATION
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='世界名著'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='武侠'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='科幻'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='军事'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='社会'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='文集'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='纪实文学'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='文学理论'),'库存 1-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='中国古诗词'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
```

```
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='中国现当代诗'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='戏剧'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='民间文学'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='计算机理论'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='计算机体系结构'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='操作系统'),'库存 1-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='程序设计'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='数据库'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='软件工程'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='信息安全'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='多媒体'),'库存 1-货架 3');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语词汇'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语语法'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语听力'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语口语'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
```

```
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语阅读'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='英语写作'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='行政管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='项目管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='质量管理与控制'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='商业道德'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='经营管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='财务管理'),'库存 1-货架 4');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='幼儿启蒙'),'库存 2-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='益智游戏'),'库存 2-货架 1');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='童话'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='卡通'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='励志'),'库存 2-货架 2');
    INSERT INTO PRODUCTION.LOCATION(PRODUCT_SUBCATEGORYID,NAME)
    VALUES((SELECT PRODUCT_SUBCATEGORYID FROM
PRODUCTION.PRODUCT_SUBCATEGORY WHERE NAME='少儿英语'),'库存 2-货架 2');

    --INSERT PRODUCT_INVENTORY
    INSERT INTO PRODUCTION.PRODUCT_INVENTORY(PRODUCTID,LOCATIONID,QUANTITY)
```

```
    SELECT T1.PRODUCTID,T2.LOCATIONID,100 FROM PRODUCTION.PRODUCT
T1,PRODUCTION.LOCATION T2
    WHERE T1.PRODUCT_SUBCATEGORYID=T2.PRODUCT_SUBCATEGORYID;

    --INSERT PRODUCT_REVIEW
    INSERT INTO
PRODUCTION.PRODUCT_REVIEW(PRODUCTID,NAME,REVIEWDATE,EMAIL,RATING,COMMENTS)
    SELECT PRODUCTID,' 刘 青 ','2007-05-06','zhangping@sina.com','1',' 送 货 快 ' from
PRODUCTION.PRODUCT;
    INSERT INTO
PRODUCTION.PRODUCT_REVIEW(PRODUCTID,NAME,REVIEWDATE,EMAIL,RATING,COMMENTS)
    SELECT PRODUCTID,' 桑 泽 恩 ','2007-05-06','zhangping@sina.com','1',' 服 务 态 度 好 ' from
PRODUCTION.PRODUCT;

    --INSERT PRODUCT_VENDOR
    INSERT INTO
PRODUCTION.PRODUCT_VENDOR(PRODUCTID,VENDORID,STANDARDPRICE,LASTPRICE,LASTDA
TE,MINQTY,MAXQTY,ONORDERQTY)
    SELECT PRODUCTID,VENDORID,25,'','10','100','' FROM
PRODUCTION.PRODUCT,PURCHASING.VENDOR WHERE
PRODUCTION.PRODUCT.PUBLISHER=PURCHASING.VENDOR.NAME;

    --INSER SALESORDER_HEADER
    INSERT INTO
SALES.SALESORDER_HEADER(ORDERDATE,DUEDATE,STATUS,ONLINEORDERFLAG,CUSTOMERID,
SALESPERSONID,ADDRESSID,SHIPMETHOD,SUBTOTAL,FREIGHT,TOTAL,COMMENTS)
    VALUES ('2007-05-06','2007-5-07',2,1,1,2,3,0,36.9,0,36.9,'上午送到');
    INSERT INTO
SALES.SALESORDER_HEADER(ORDERDATE,DUEDATE,STATUS,ONLINEORDERFLAG,CUSTOMERID,
SALESPERSONID,ADDRESSID,SHIPMETHOD,SUBTOTAL,FREIGHT,TOTAL,COMMENTS)
    VALUES ('2007-05-07','2007-5-07',1,1,1,1,1,0,36.9,0,36.9,'上午送到');

    -- INSERT SALESORDER_DETAIL
    INSERT INTO
SALES.SALESORDER_DETAIL(SALESORDERID,SALESORDER_DETAILID,CARRIERNO,PRODUCTID,
ORDERQTY,LINETOTAL)
    SELECT SALESORDERID,'1','2007052',1,1,15.2 FROM SALES.SALESORDER_HEADER;
    INSERT INTO
SALES.SALESORDER_DETAIL(SALESORDERID,SALESORDER_DETAILID,CARRIERNO,PRODUCTID,
ORDERQTY,LINETOTAL)
    SELECT SALESORDERID,'2','2007053',3,1,21.7 FROM SALES.SALESORDER_HEADER;

    UPDATE SALES.SALESPERSON SET SALESLASTYEAR = 20.0000 WHERE SALESPERSONID = 2;
```

```
INSERT INTO PURCHASING.PURCHASEORDER_HEADER(ORDERDATE,STATUS,EMPLOYEEID,VENDORID,SHIPMETHOD,SUBTOTAL,TAX,FREIGHT,TOTAL)
VALUES('2006-7-21',1,6,5,'快递',5000.00,600.00,800.00,6400.00);

-- INSERT DEPARTMENT
INSERT INTO OTHER.DEPARTMENT VALUES(NULL, '总公司');
INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '采购部');
INSERT INTO OTHER.DEPARTMENT VALUES('总公司', '财务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '网络服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '读者服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('服务部', '企业服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('读者服务部', '书籍借阅服务部');
INSERT INTO OTHER.DEPARTMENT VALUES('读者服务部', '书籍阅览服务部');

-- INSERT EMPSALARY
INSERT INTO OTHER.EMPSALARY VALUES ('KING',7839,5000);
INSERT INTO OTHER.EMPSALARY VALUES ('SCOTT',7788,3000);
INSERT INTO OTHER.EMPSALARY VALUES ('FORD',7902,3000);
INSERT INTO OTHER.EMPSALARY VALUES ('JONES',7566,2975);
INSERT INTO OTHER.EMPSALARY VALUES ('BLAKE',7698,2850);
INSERT INTO OTHER.EMPSALARY VALUES ('CLARK',7782,2450);
INSERT INTO OTHER.EMPSALARY VALUES ('ALLEN',7499,1600);
INSERT INTO OTHER.EMPSALARY VALUES ('TURNER',7844,1500);
INSERT INTO OTHER.EMPSALARY VALUES ('MILLER',7934,1300);
INSERT INTO OTHER.EMPSALARY VALUES ('WARD',7521,1250);
INSERT INTO OTHER.EMPSALARY VALUES ('MARTIN',7654,1250);
INSERT INTO OTHER.EMPSALARY VALUES ('ADAMS',7876,1100);
INSERT INTO OTHER.EMPSALARY VALUES ('JAMES',7900,950);
INSERT INTO OTHER.EMPSALARY VALUES ('SMITH',7369,800);

-- INSERT ACCOUNT
INSERT INTO OTHER.ACCOUNT VALUES(1,1000);
INSERT INTO OTHER.ACCOUNT VALUES(2,2000);
INSERT INTO OTHER.ACCOUNT VALUES(3,1500);
INSERT INTO OTHER.ACCOUNT VALUES(4,6500);
INSERT INTO OTHER.ACCOUNT VALUES(5,500);

-- INSERT ACTIONS
INSERT INTO OTHER.ACTIONS VALUES(3,'U',599,NULL);
INSERT INTO OTHER.ACTIONS VALUES(6,'I',20099,NULL);
INSERT INTO OTHER.ACTIONS VALUES(5,'D',NULL,NULL);
INSERT INTO OTHER.ACTIONS VALUES(7,'U',1599,NULL);
```

```
INSERT INTO OTHER.ACTIONS VALUES(1,'I',399,NULL);
INSERT INTO OTHER.ACTIONS VALUES(9,'D',NULL,NULL);
INSERT INTO OTHER.ACTIONS VALUES(10,'X',NULL,NULL);

--INSERT READER
INSERT INTO OTHER.READER VALUES(10, 'Bill', 19, 'M', 'Computer');
INSERT INTO OTHER.READER VALUES(11, 'Susan', 18, 'F', 'History');
INSERT INTO OTHER.READER VALUES(12, 'John', 19, 'M', 'Computer');
```

第 3 章 数据定义语句

3.1 数据库修改语句

一个数据库创建成功后，可以增加和重命名日志文件，修改日志文件大小。可以修改数据库的状态和模式。

语法格式

```
ALTER DATABASE <修改数据库语句>;
<修改数据库语句>::= RESIZE LOGFILE <文件路径> TO <文件大小>|
ADD LOGFILE <文件说明项>{,<文件说明项>}|
RENAME LOGFILE <文件路径>{,<文件路径>} TO <文件路径>{,<文件路径>}|
MOUNT|SUSPEND|OPEN [FORCE]|NORMAL|RIMARY|STANDBY|ARCHIVELOG|NOARCHIVELOG|
<ADD|MODIFY|DELETE> ARCHIVELOG <归档配置语句>
<文件说明项> ::= <文件路径> SIZE <文件大小>[<自动扩展子句>]
<自动扩展子句> ::= AUTOEXTEND <ON [<每次扩展大小子句>][<最大大小子句> |OFF>
<每次扩展大小子句> ::= NEXT <扩展大小>
<最大大小子句> ::= MAXSIZE <文件最大大小>
<归档配置语句>::= 'DEST = <归档目标>,TYPE = <归档类型>'
<归档类型>::=LOCAL [,FILE_SIZE = <文件大小>][,SPACE_LIMIT = <空间大小限制>] |
REALTIME|SYNC |ASYNC ,TIMER_NAME = <定时器名称> | MARCH;
```

参数

1．<文件路径>　指明被操作的数据文件在操作系统下的路径＋数据文件名；

2．<文件大小>　整数值，单位为 M；

3．<扩展大小>　整数值，指明新增数据文件的每次扩展大小 (单位 MB)，取值范范围为 0-2048；

4．<文件最大大小> 整数值，指明新增数据文件的最大大小(单位 MB)，不能小于此文件的初始大小；

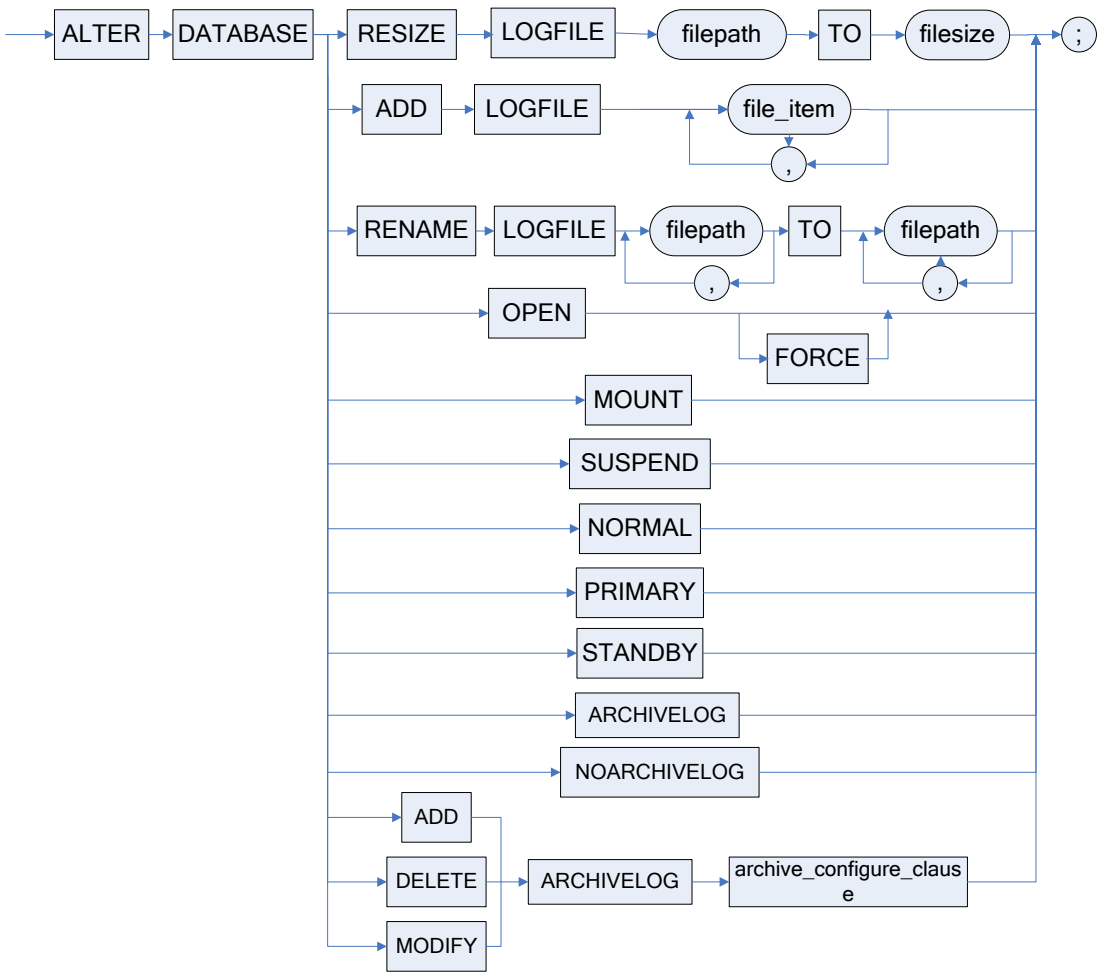
5．<归档目标>　指归档日志所在位置，若本地归档，则本地归档目录；若远程归档，则为远程服务实例名；删除操作，只需指定归档目标；

6．<归档类型>　指归档操作类型，包括 REALTIME/ASYNC/SYNC/LOCAL/MARCH；

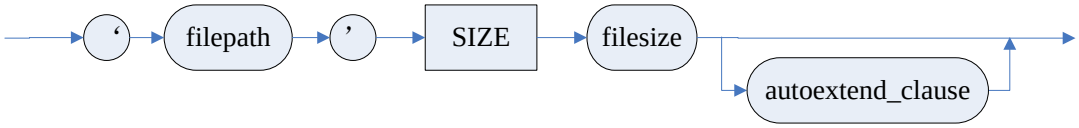
7．<空间大小限制>　整数值，范围（1024~4294967294），若设为 0，表示不限制，仅本地归档有效；

8．<定时器名称>　异步归档中指定的定时器名称，仅异步归档有效。

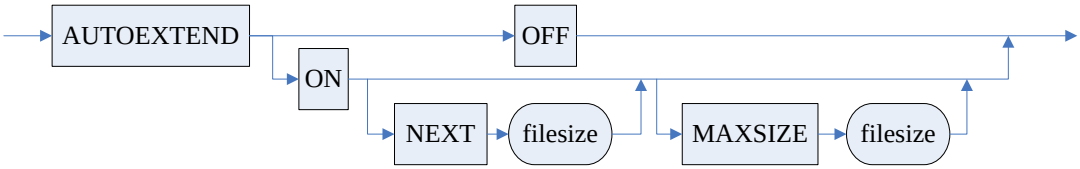
图例



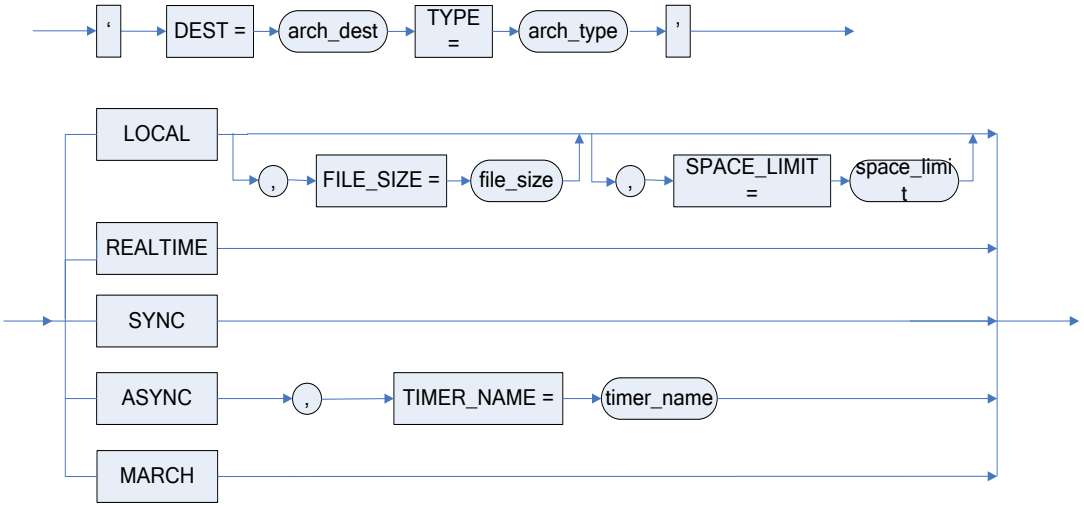
file_item



autoextend_clause



archive configure_clause



语句功能

供具有 DBA 权限的用户修改数据库。

使用说明

1．修改日志文件大小时，只能增加文件的大小，否则失败；

2．只有 MOUNT 状态下才能启用或关闭归档，添加、修改、删除归档，重命名日志文件。

举例说明

假设数据库 BOOKSHOP 页面大小为 8K，数据文件存放路径为 C:\DMDBMS\data。

例 1 给数据库增加一个日志文件 C:\DMDBMS\data\dmlog_0.log，其大小为 200M。

```
ALTER DATABASE ADD LOGFILE 'C:\DMDBMS\data\dmlog_0.log'    SIZE 200;
```

例 2 扩展数据库中的日志文件 C:\DMDBMS\data\dmlog_0.log，使其大小增大为 300M。

```
ALTER DATABASE RESIZE LOGFILE 'C:\DMDBMS\data\dmlog_0.log'    TO 300;
```

例 3 设置数据库状态为 MOUNT。

```
ALTER DATABASE MOUNT;
```

例 4 设置数据库状态为 OPEN。

```
ALTER DATABASE OPEN;
```

例 5 设置数据库状态为 SUSPEND。

```
ALTER DATABASE SUSPEND;
```

例 6 重命名日志文件 C:\DMDBMS\data\dmlog_0.log 为 d:\dmlog_1.log。

```
ALTER DATABASE MOUNT;
ALTER DATABASE RENAME LOGFILE 'C:\DMDBMS\data\dmlog_0.log' TO 'd:\dmlog_1.log';
ALTER DATABASE OPEN;
```

例 7 设置数据库模式为 PRIMARY。

```
ALTER DATABASE MOUNT;
ALTER DATABASE PRIMARY;
ALTER DATABASE OPEN FORCE;
```

例 8 设置数据库模式为 STANDBY。

```
ALTER DATABASE MOUNT;
ALTER DATABASE STANDBY;
ALTER DATABASE OPEN FORCE;
```

例 9 设置数据库模式为 NORMAL。

```
ALTER DATABASE MOUNT;
ALTER DATABASE NORMAL;
ALTER DATABASE OPEN;
```

例 10 设置数据库归档模式为归档。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ARCHIVELOG;
```

例 11 设置数据归档模式为非归档。

```
ALTER DATABASE MOUNT;
ALTER DATABASE NOARCHIVELOG;
```

例 12 增加本地归档配置，归档目录为 c:\arch_local，文件大小为 128MB，空间限制为 1024MB。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = c:\arch_local, TYPE = local, FILE_SIZE = 128, SPACE_LIMIT = 1024';
```

例 13 增加一个实时归档配置，远程服务实例名为 realtime，需事先配置 mail。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = realtime, TYPE = REALTIME';
```

例 14 增加一个同步归档配置，远程服务实例名为 syn，需事先配置 mail。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = syn, TYPE = SYNC ';
```

例 15 增加一个异步归档配置，远程服务实例名为 asyn，定时器名为 timer1，需事先配置好 mail 和 timer。

```
ALTER DATABASE MOUNT;
ALTER DATABASE ADD ARCHIVELOG 'DEST = asyn, TYPE = ASYNC, TIMER_NAME = timer1';
```

例 16 修改实时归档为同步归档，同步归档只能存在一个，若已经存在，则需删除（沿用前面的例子）。

```
ALTER DATABASE MOUNT;
ALTER DATABASE DELETE ARCHIVELOG 'DEST = syn';
ALTER DATABASE MODIFY ARCHIVELOG 'DEST = realtime, TYPE = SYNC';
```

3.2 管理用户

3.2.1 用户定义语句

在数据库中创建新的用户，DM7 中直接用 USER 与数据库服务器建立连接。

语法格式

```
CREATE USER <用户名> IDENTIFIED <身份验证模式>  [PASSWORD_POLICY <口令策略>] [<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>
<外部身份验证模式> ::= EXTERNALLY
<口令策略> ::=  口令策略项的任意组合
```

```
<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> ::= LIMIT <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
<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>
```

参数

1. <用户名> 指明要创建的用户名称，用户名称最大长度 128 字节；
2. <参数设置>用于限制用户对 DM 数据库服务器系统资源的使用；
3. 系统在创建用户时，必须指定一种身份验证模式：<数据库身份验证模式>或者<外部身份验证模式>；
4. 如要利用基于 OS 的身份验证，则在创建登录或修改登录时，要利用 IDENTIFIED EXTERNALLY 关键字。另外，基于 OS 的身份验证分为本机验证和远程验证，本机验证在任何情况下都可以使用，而远程验证则需要将配置文件 dm.ini 的 ENABLE_REMOTE_OSAUTH 项设置为 1(缺省为 0)，表示支持远程验证，同时还要将配置文件 dm.ini 的 ENABLE_ENCRYPT 项设置为 1，表示采用 SSL 安全连接；
5. <口令策略>可以为以下值，或其任何组合：
   0 无策略；
   1 禁止与用户名相同；
   2 口令长度不小于 6；
   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)。

若为其他数字，则表示以上设置值的和，如 3＝1＋2,表示同时启用第 1 项和第 2 项策略。当设置为 0 时，表示设置口令没有限制，但总长度不得超过 48 个字节。另外，若不指定该项，则默认采用系统配置文件中 PWD_POLICY 所设值。

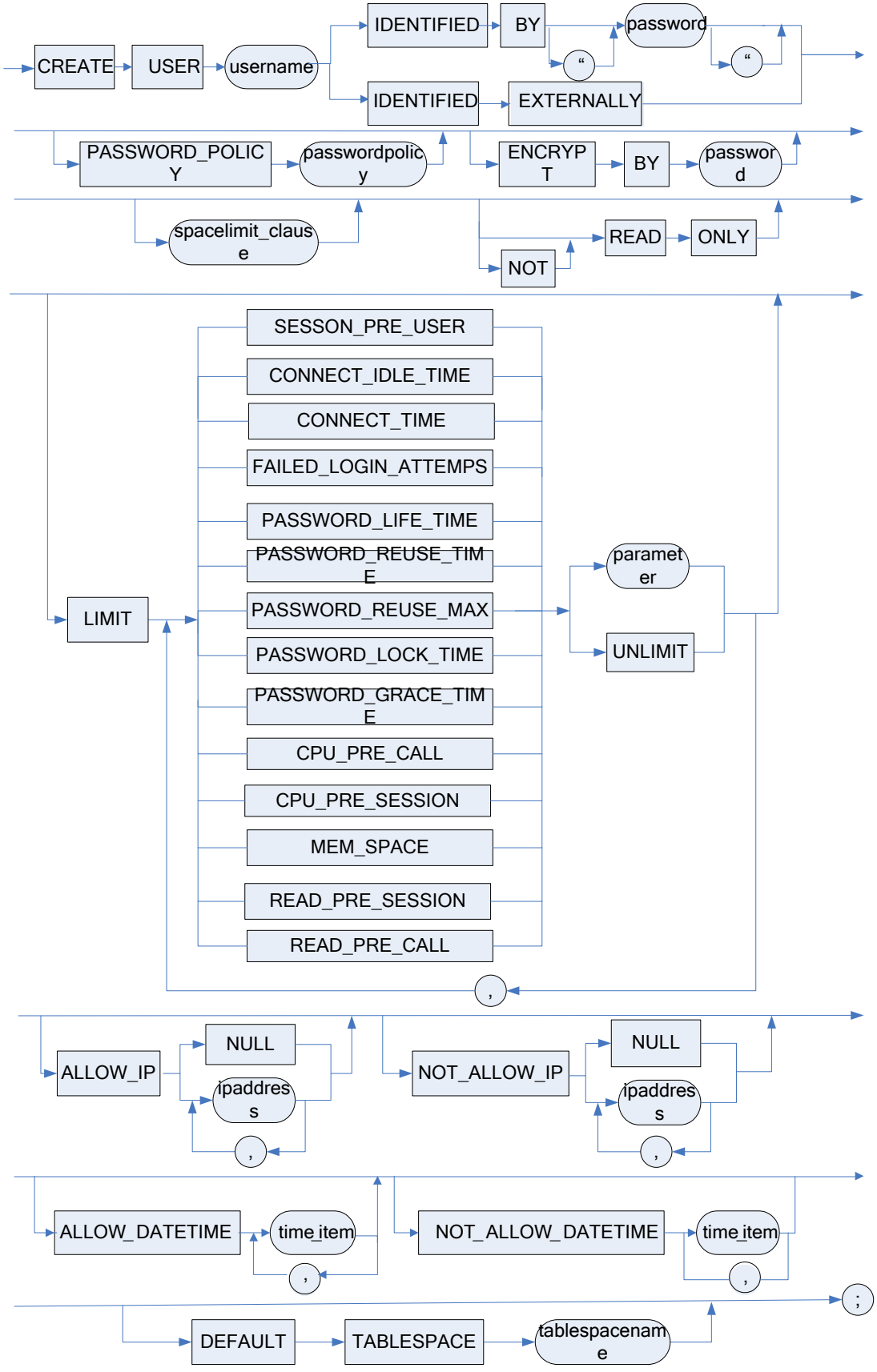
6. 存储加密密钥用于与半透明加密配合使用，缺省情况下系统自动生成一个密钥；
7. 只读标志表示该登录是否只能对数据库作只读操作，默认为可读写；
8. 资源设置项的各参数设置说明见下表：

表 3.2.1 资源设置项说明

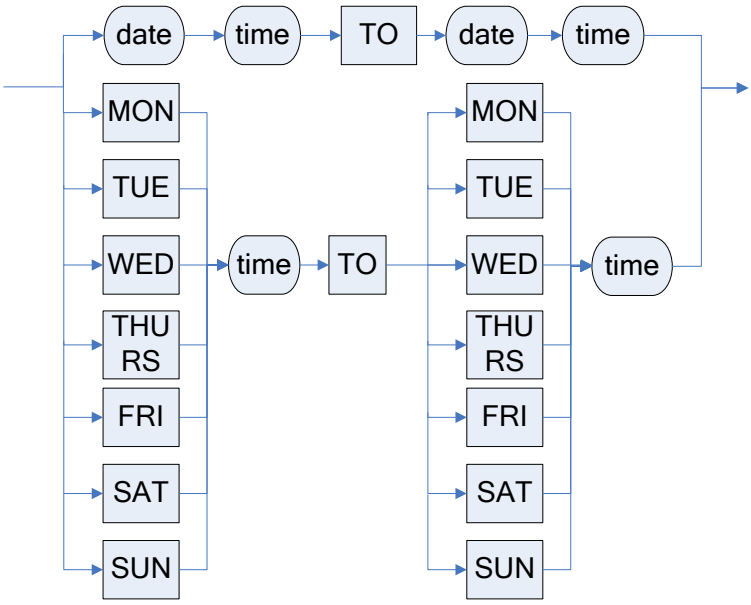
资源设置项	说明	最大值	最小值	缺省值
SESSION_PER_USER	在一个实例中，一个用户可以同时拥有的会话数量	32768	1	系统所能提供的最大值
CONNECT_TIME	一个会话连接、访问和操作数据库服务器的时间上限（单位：10 分钟）	144（1 天）	1	无限制
CONNECT_IDLE_TIME	会话最大空闲时间（单位：10 分钟）	144（1 天）	1	无限制
FAILED_LOGIN_ATTEMPS	将引起一个帐户被锁定的连续注册失败的次数	100	1	3
CPU_PER_SESSION	一个会话允许使用的 CPU 时间上限(单位：秒）	31536000（365 天）	1	无限制
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，一个帐户将被锁定的分钟数	1440（1 天）	1	1
PASSWORD_GRACE_TIME	以天为单位的口令过期宽限时间	30	1	10

9. 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，可以利用*来设置网段，如 192.168.0.*；
10. 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。在设置时间段时，有两种方式：
    1） 具体时间段，如 2006 年 1 月 1 日 8：30 至 2006 年 2 月 1 日 17：00；
    2） 规则时间段，如 每周一 8：30 至 每周五 17：00。
11. 口令策略、允许 IP、禁止 IP、允许时间段、禁止时间段和外部身份验证功能只在安全版本中提供；
12. 用户默认表空间不能使用 RLOG，ROLL，TEMP 表空间。

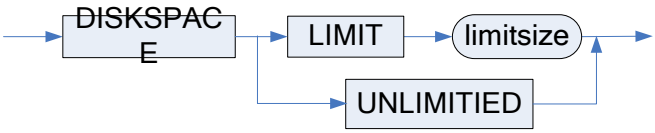
图例


CREATE
USER
username
IDENTIFIED
BY
password
“
“
IDENTIFIED
EXTERNALLY
PASSWORD_POLICY
passwordpolicy
ENCRYPT
BY
password
spacelimit_clause
NOT
READ
ONLY
LIMIT
SESSON_PRE_USER
CONNECT_IDLE_TIME
CONNECT_TIME
FAILED_LOGIN_ATTEMPS
PASSWORD_LIFE_TIME
PASSWORD_REUSE_TIME
PASSWORD_REUSE_MAX
PASSWORD_LOCK_TIME
PASSWORD_GRACE_TIME
CPU_PRE_CALL
CPU_PRE_SESSION
MEM_SPACE
READ_PRE_SESSION
READ_PRE_CALL
parameter
UNLIMIT
,
ALLOW_IP
NULL
ipaddress
,
NOT_ALLOW_IP
NULL
ipaddress
,
ALLOW_DATETIME
time_item
,
NOT_ALLOW_DATETIME
time_item
,
;
DEFAULT
TABLESPACE
tablespacename

time_item



spacelimit_clause



语句功能

创建新的用户。

使用说明

1. 用户名在服务器中必须唯一；
2. 系统为一个用户存储的信息主要有：用户名、口令、资源限制；
3. 用户口令以密文形式存储；
4. 系统预先设置了三个用户，分别为 SYSDBA、SYSAUDITOR 和 SYSSSO，其中 SYSDBA 具备 DBA 角色，SYSAUDITOR 具备 DB_AUDIT_ADMIN 角色，而 SYSSSO 具备 DB_POLICY_ADMIN 系统角色；
5. DM 提供三种身份验证模式来保护对服务器访问的安全，即数据库身份验证模式、外部身份验证模式和混合身份验证模式。数据库身份验证模式需要利用数据库口令；外部身份验证模式既支持基于操作系统(OS)的身份验证又提供口令管理策略；混合身份验证模式是同时支持数字证书和数据库身份的双重验证；
6. 如要利用基于 OS 的身份验证，则在创建用户或修改用户时，要利用 IDENTIFIED EXTERNALLY 关键字。

举例说明

例 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;
```

例 创建用户名为 BOOKSHOP_OS_USER、基于操作系统身份验证的用户。

```
CREATE USER BOOKSHOP_OS_USER IDENTIFIED EXTERNALLY;
```

3.2.2 修改用户语句

修改数据库中的用户。

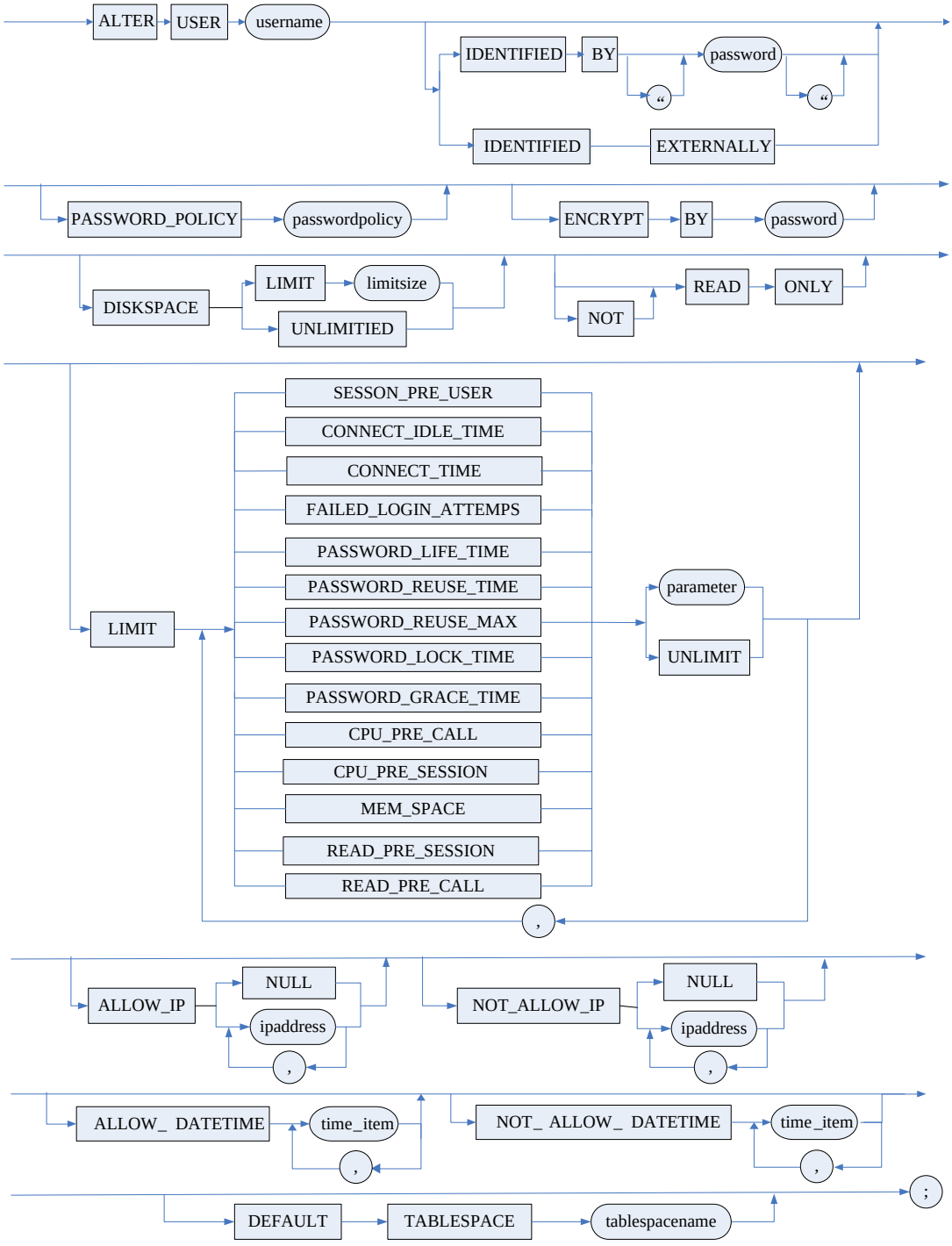
语法格式

```
ALTER USER <用户名> [IDENTIFIED <身份验证模式>] [PASSWORD_POLICY <口令策略>] [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>
<外部身份验证模式> ::= EXTERNALLY
<口令策略> ::=  口令策略项的任意组合
[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> ::= LIMIT <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
<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>
```

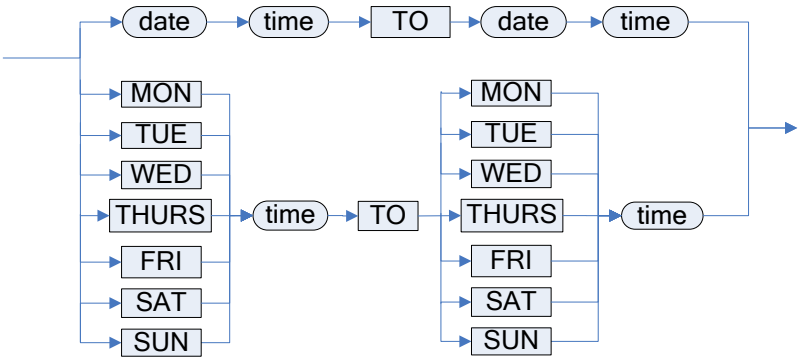
参数

同 CREATE USER 的参数规定一样。

图例


ALTER
USER
username
IDENTIFIED
BY
password
“
“
IDENTIFIED
EXTERNALLY
PASSWORD_POLICY
passwordpolicy
ENCRYPT
BY
password
DISKSPACE
LIMIT
limitsize
UNLIMITIED
NOT
READ
ONLY
LIMIT
SESSON_PRE_USER
CONNECT_IDLE_TIME
CONNECT_TIME
FAILED_LOGIN_ATTEMPS
PASSWORD_LIFE_TIME
PASSWORD_REUSE_TIME
PASSWORD_REUSE_MAX
PASSWORD_LOCK_TIME
PASSWORD_GRACE_TIME
CPU_PRE_CALL
CPU_PRE_SESSION
MEM_SPACE
READ_PRE_SESSION
READ_PRE_CALL
,
parameter
UNLIMIT
ALLOW_IP
NULL
ipaddress
,
NOT_ALLOW_IP
NULL
ipaddress
,
ALLOW_ DATETIME
time_item
,
NOT_ ALLOW_ DATETIME
time_item
,
;
DEFAULT
TABLESPACE
tablespacename

time_item



语句功能

修改用户。

使用说明

1．每个用户均可修改自身的口令(在数据库验证方式下)；

2．只有具备 ALTER USER 权限的用户才能修改其身份验证模式、系统角色及资源限制项；

3. 不论 dm.ini 的 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。

4．系统固定用户不能修改其系统角色和资源限制项；

5．其他参数的取值、意义与 CREATE USER 中的要求一样。

举例说明

例 修改用户 BOOKSHOP_USER，会话空闲期为无限制，最大连接数为 10。

```
ALTER USER BOOKSHOP_USER LIMIT SESSION_PER_USER 10, CONNECT_IDLE_TIME UNLIMITED;
```

3.2.3 用户删除语句

删除用户。

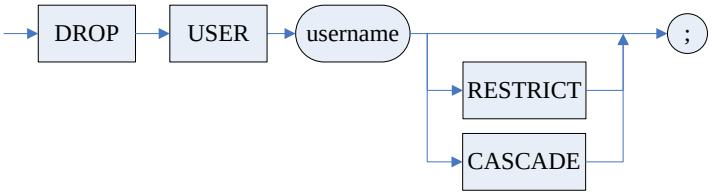
语法格式

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

参数

<用户名> 指明被删除的用户。

图例



语句功能

删除指定用户。

使用说明

1．系统自动创建的三个系统用户 SYSDBA、SYSAUDITOR 和 SYSSSO 不能被删除；

2．具有相应的 DROP USER 权限的用户即可进行删除用户操作；

3．执行此语句将导致 DM 删除数据库中该用户建立的所有对象，且不可恢复。如果要

保存这些实体，请参考 REVOKE 语句；

4．如果未使用 CASCADE 选项，若该用户建立了数据库对象 (如表、视图、过程或函数)，或其他用户对象引用了该用户的对象，或在该用户的表上存在其它用户建立的视图，DM 将返回错误信息，而不删除此用户；

5．如果使用了 CASCADE 选项，除数据库中该用户及其创建的所有对象被删除外，如果其他用户创建的表引用了该用户表上的主关键字或唯一关键字，或者在该表上创建了视图，DM 还将自动删除相应的引用完整性约束及视图依赖关系；

6．正在使用中的用户可以被删除，删除后重登录或者做操作会报错。

举例说明

例 删除用户 BOOKSHOP_USER 的语句。

```
DROP USER BOOKSHOP_USER;
```

例 删除用户 BOOKSHOP_OS_USER 的语句。

```
DROP USER BOOKSHOP_OS_USER CASCADE;
```

3.3 管理模式

3.3.1 模式定义语句

模式定义语句创建一个架构，并且可以在概念上将其看作是包含表、视图和权限定义的对象。在 DM 中，一个用户可以创建多个模式，一个模式中的对象(表、视图)可以被多个用户使用。

系统为每一个用户自动建立了一个与用户名同名的模式作为默认模式，用户还可以用模式定义语句建立其它模式。

语法格式

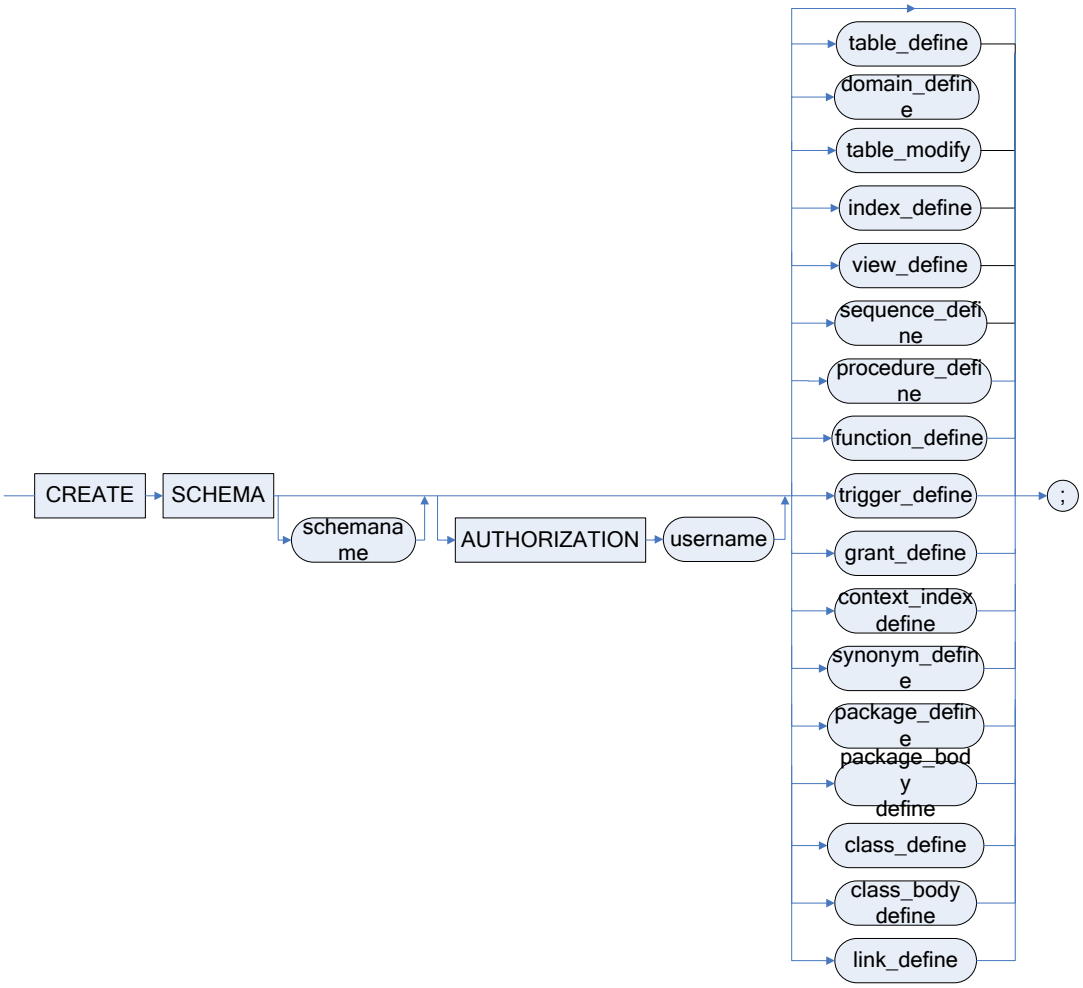
```
CREATE SCHEMA [<模式名>] [AUTHORIZATION <用户名>]
[<基表定义> | <域定义>| <基表修改> | <索引定义> | <视图定义> | <序列定义>|
<存储过程定义> | <存储函数定义> | <触发器定义> | <特权定义>|
<全文索引定义> | <同义词定义>|<包定义>|<包体定义>|<类定义>|<类体定义>|<外部链接定义>];
```

参数

1. <模式名> 创建模式的名字，模式名最大长度 128 字节；
2. <基表定义> 建表语句；
3. <域定义> 域定义语句；
4. <基表修改> 基表修改语句；
5. <索引定义> 索引定义语句；
6. <视图定义> 建视图语句；
7. <序列定义> 建序列语句；
8. <存储过程定义> 存储过程定义语句；
9. <存储函数定义> 存储函数定义语句；
10. <触发器定义> 建触发器语句；
11. <特权定义> 授权语句；
12. <全文索引定义> 全文索引定义语句；
13. <同义词定义> 同义词定义语句；
14. <包定义> 包定义语句；

15. <包体定义> 包体定义语句；
16. <类定义> 类定义语句；
17. <类体定义> 类体定义语句；
18. <外部链接定义> 外部链接定义语句。

图例



语句功能

供具有 DBA 或 CREATE SCHEMA 权限的用户在指定数据库中定义模式。

使用说明

1. <模式名>不可与其所在数据库中其它模式名相同；在创建新的模式时，如果存在同名的模式，那么跳过该语句执行；
2. AUTHORIZATION <用户名>标识了拥有该模式的用户；它是为其他用户创建模式时使用的；
3. 使用该语句的用户必须具有 DBA 或 CREATE SCHEMA 权限；
4. 定义模式时，用户可以用单条语句同时建多个表、视图，同时进行多项授权；模式一旦定义，该用户所建基表、视图等均属该模式，其它用户访问该用户所建立的基表、视图等均需在表名、视图名前冠以模式名，而建表者访问自己所建表、视图时模式名可省，这时系统自动以用户名作为模式名；
5. 模式定义语句中的基表修改子句只允许添加表约束；
6. 模式定义语句中的索引定义子句不能定义聚集索引；
7. 模式未定义之前，其它用户访问该用户所建的基表、视图等均需在表名前冠以建表

者名；

8. 模式定义语句不允许与其它 SQL 语句一起执行；
9. 在 disql 中使用该语句必须以“/”结束。

举例说明

例 下面是用户 SYSDBA 建立模式的例子，建立的模式属于 SYSDBA。

```
CREATE SCHEMA SCHEMA1 AUTHORIZATION SYSDBA;
```

3.3.2 设置当前模式语句

设置当前模式。

语法格式

```
SET SCHEMA <模式名>;
```

图例

使用说明

只能设置到属于自己的模式。

举例说明

例 SYSDBA 用户将当前的模式从 SYSDBA 换到 SALES 模式。

```
SET SCHEMA SALES;
```

3.3.3 模式删除语句

在 DM 系统中，允许用户删除整个模式。

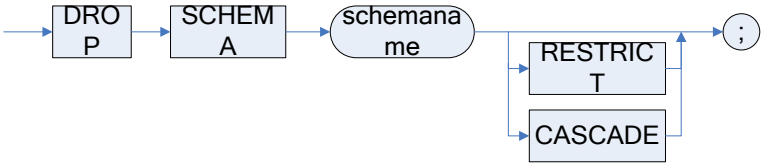
语法格式

```
DROP SCHEMA <模式名> [RESTRICT | CASCADE];
```

参数

<模式名> 指要删除的模式名。

图例



语句功能

供具有 DBA 角色的用户或该模式的拥有者删除模式。

使用说明

1. <模式名>必须是当前数据库中已经存在的模式；
2. 用该语句的用户必须具有 DBA 权限或是该模式的所有者；
3. 如果使用 RESTRICT 选项，只有当模式为空时删除才能成功，否则，当模式中存在数据库对象时则删除失败。默认选项为 RESTRICT 选项；
4. 如果使用 CASCADE 选项，则将整个模式、模式中的对象，以及与该模式相关的依赖关系都删除。

举例说明

例 以 SYSDBA 身份登录数据库后，删除 BOOKSHOP 库中模式 SCHEMA1。

```
DROP SCHEMA SCHEMA1 CASCADE;
```

3.4 管理表空间

3.4.1 表空间定义语句

创建表空间。

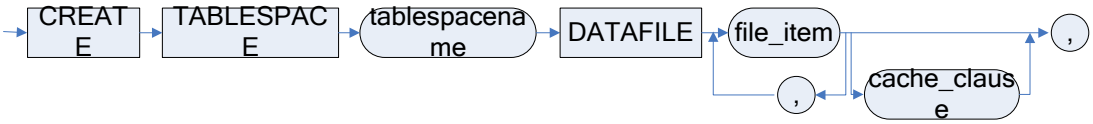
语法格式

```
CREATE TABLESPACE <表空间名> <数据文件子句>[<数据页缓冲池子句>]
<数据文件子句> ::= DATAFILE <文件说明项>{,<文件说明项>}
<文件说明项> ::= <文件路径> [ MIRROR <文件路径>] SIZE <文件大小>[<自动扩展子句>]
<自动扩展子句> ::= AUTOEXTEND <ON [<每次扩展大小子句>][<最大大小子句> |OFF>
<每次扩展大小子句> ::= NEXT <扩展大小>
<最大大小子句> ::= MAXSIZE <文件最大大小>
<数据页缓冲池子句>::= CACHE = <缓冲池名>
```

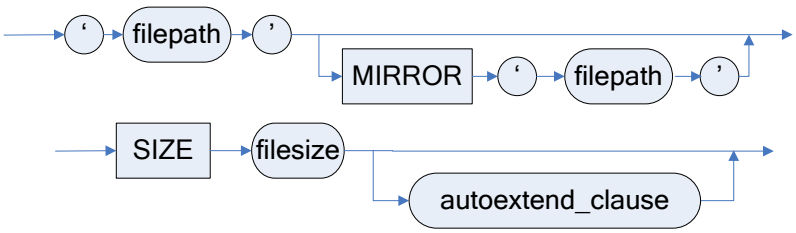
参数

1. <表空间名> 表空间的名称，表空间名称最大长度 128 字节；
2. <文件路径> 指明新生成的数据文件在操作系统下的路径＋新数据文件名。数据文件的存放路径符合 DM 安装路径的规则，且该路径必须是已经存在的；
3. MIRROR 数据文件镜像，用于在数据文件出现损坏时替代数据文件进行服务。
4. <文件大小> 整数值，指明新增数据文件的大小(单位 MB)，取值范围 4096*页大小~2147483647*页大小；
5. <缓冲池名> 系统数据页缓冲池名 NORMAL 或 KEEP。

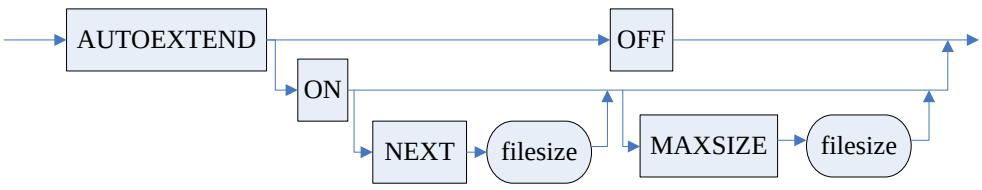
图例



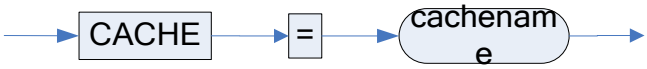
file_item



autoextend_clause



cache_clause



语句功能

供具有 DBA 角色的用户创建表空间。

使用说明

1. 表空间名在服务器中必须唯一；
2. 用该语句的用户必须具有 DBA 权限；
3. 表空间 ID 取值范围为 0~32767，ID 由系统自动分配，ID 不能重复使用，即使删除掉已有表空间，也无法重复使用已用 ID 号；
4. HFS 表空间不支持创建 MIRROR 镜像文件。

举例说明

例 以 SYSDBA 身份登录数据库后，创建表空间 TS1，指定数据文件 TS1.dbf，大小 128M。

```
CREATE TABLESPACE TS1 DATAFILE 'd:\TS1.dbf' SIZE 128;
```

3.4.2 修改表空间语句

修改表空间。

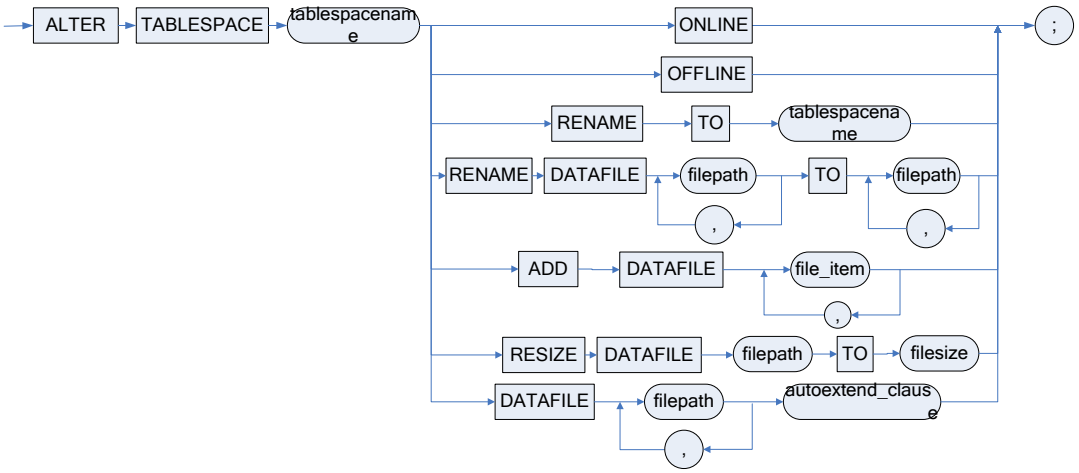
语法格式

```
ALTER TABLESPACE <表空间名> [ONLINE丨OFFLINE丨<表空间重命名子句>丨<数据文件重命名子句>|<增加数据文件子句>|<修改文件大小子句>|<修改文件自动扩展子句>|<数据页缓冲池子句>
<表空间重命名子句> ::= RENAME TO <表空间名>
<数据文件重命名子句>::= RENAME DATAFILE <文件路径>{,<文件路径>} TO <文件路径>{,<文件路径>}
<增加数据文件子句> ::= ADD <数据文件子句>
<数据文件子句>见上一节表空间定义语句
<修改文件大小子句>::= RESIZE DATAFILE <文件路径> TO <文件大小>
<修改文件自动扩展子句>::= DATAFILE <文件路径>{,<文件路径>}[<自动扩展子句>]
<自动扩展子句>::=见 3.1 节说明
<数据页缓冲池子句>::=见 3.1 节说明
```

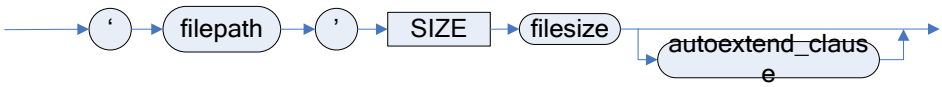
参数

1. <表空间名> 表空间的名称；
2. <文件路径> 指明数据文件在操作系统下的路径＋新数据文件名。数据文件的存放路径符合 DM 安装路径的规则，且该路径必须是已经存在的；
3. <文件大小> 整数值，指明新增数据文件的大小(单位 MB)；
4. <缓冲池名> 系统数据页缓冲池名 NORMAL 或 KEEP。

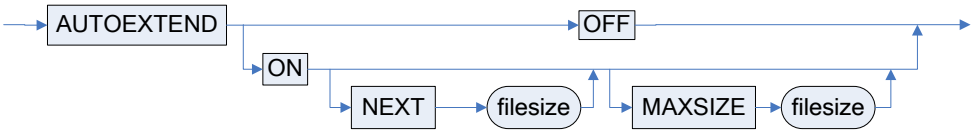
图例



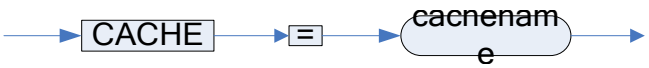
file_item



autoextend_clause



cache_clause



语句功能

供具有 DBA 角色的用户修改表空间。

使用说明

1. 用该语句的用户必须具有 DBA 权限；
2. 修改表空间数据文件大小时，其大小必须大于自身大小；
3. 如果表空间有未提交事务时，表空间不能修改为 OFFLINE 状态；
4. 重命名表空间数据文件时，表空间必须处于 OFFLINE 状态，修改成功后再将表空间修改为 ONLINE 状态。

举例说明

例 1 将表空间 TS1 名字修改为 TS2。

```
ALTER TABLESPACE TS1 RENAME TO TS2;
```

例 2 增加一个路径为 d:\TS1_1.dbf，大小为 128M 的数据文件到表空间 TS1。

```
ALTER TABLESPACE TS1 ADD DATAFILE   'd:\TS1_1.dbf' SIZE 128;
```

例 3 修改表空间 TS1 中数据文件 d:\TS1.dbf 的大小为 200M。

```
ALTER TABLESPACE TS1 RESIZE DATAFILE   'd:\TS1.dbf' TO 200;
```

例 4 重命名表空间 TS1 的数据文件 d:\TS1.dbf 为 e:\TS1_0.dbf。

```
ALTER TABLESPACE TS1 OFFLINE;
ALTER TABLESPACE TS1 RENAME DATAFILE 'd:\TS1.dbf' TO 'e:\TS1_0.dbf' ;
ALTER TABLESPACE TS1 ONLINE;
```

例 5 修改表空间 TS1 的数据文件 d:\TS1.dbf 自动扩展属性为每次扩展 10M，最大文件大小为 1 G。

```
ALTER TABLESPACE TS1 DATAFILE   'd:\TS1.dbf' AUTOEXTEND ON NEXT 10 MAXSIZE 1000;
```

例 6 修改表空间 TS1 缓冲池名字为 KEEP。

```
ALTER TABLESPACE TS1 CACHE=KEEP;
```

3.4.3 表空间删除语句

删除表空间。

语法格式

```
DROP TABLESPACE <表空间名>
```

参数

<表空间名>　所要删除的表空间的名称。

图例

语句功能

供具有 DBA 角色的用户删除表空间。

使用说明

1. SYSTEM、RLOG、ROLL 和 TEMP 表空间不允许删除；
2. 用该语句的用户必须具有 DBA 权限；
3. 系统处于 SUSPEND 或 MOUNT 状态时不允许删除表空间，系统只有处于 OPEN 状态下才允许删除表空间。

举例说明

例　以 SYSDBA 身份登录数据库后，删除表空间 TS1。

```
DROP TABLESPACE TS1;
```

3.5 管理 HTS 表空间

创建 HFS 表之前，必须要先创建一个 HUGE TABLESPACE（HTS）。如果不创建，只能使用系统表空间 HMAIN。

3.5.1 创建 HTS 表空间

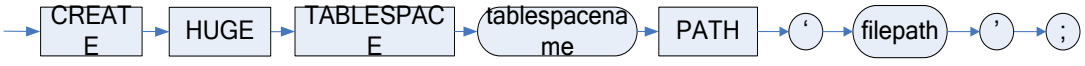
语法格式

```
CREATE HUGE TABLESPACE <表空间名> PATH <表空间路径>;
```

参数

1. <表空间名>　表空间的名称，表空间名称最大长度 128 字节；
2. <表空间路径>　指明新生成的表空间在操作系统下的路径。

图例



语句功能

供具有 DBA 角色的用户创建 HTS 表空间。

使用说明

HUGE 表空间 ID 取值范围为 0~32767，ID 由系统自动分配，ID 不能重复使用，即使删除掉已有 HUGE 表空间，也无法重复使用已用 ID 号。

举例说明

例 创建表空间 HTS_NAME。

```
CREATE HUGE TABLESPACE HTS_NAME PATH 'e:\HTSSPACE';
```

3.5.2 删除 HTS 表空间

删除 HTS 表空间。

语法格式

```
DROP HUGE TABLESPACE <表空间名>
```

参数

<表空间名> 所要删除的 HTS 表空间的名称。

图例

语句功能

供具有 DBA 角色的用户删除 HTS 表空间。

使用说明

1. 用该语句的用户必须具有 DBA 权限；
2. 该表空间必须未被使用才能被删除。

举例说明

例 以 SYSDBA 身份登录数据库后，删除 HTS 表空间 HTS_NAME。

```
DROP HUGE TABLESPACE HTS_NAME;
```

3.6 管理表

3.6.1 基表定义语句

用户数据库建立后，就可以定义基表来保存用户数据的结构。达梦数据库中基表可以分为两类，分别为数据库表和外部表，数据库表由数据库管理系统自行组织管理，而外部表在数据库的外部组织，是操作系统文件。手册中如无明确说明基表均指数据库表。下面分别对这两类基表的创建与使用进行详细描述。

3.6.1.1 数据库基表

用户数据库建立后，就可以定义基表来保存用户数据的结构。需指定如下信息：

1. 表名、表所属的模式名；
2. 列定义；
3. 完整性约束。

语法格式

```
CREATE [[GLOBAL | LOCAL] TEMPORARY] [VERTICAL] TABLE <表名定义> <表结构定义>[ON COMMIT <DELETE | PRESERVE>  ROWS] [<PARTITION 子句>][<空间限制子句>] [<STORAGE 子句>][<
```

```
压缩子句] [<DISTRIBUTE 子句>];
    <表名定义> ::= [<模式名>.] <表名>
    <表结构定义>::=<表结构定义 1> | <表结构定义 2>
    <表结构定义 1>::= (<列定义> {,<列定义>} [<表级约束定义>{,<表级约束定义>}])
    <表结构定义 2>::= [<空间限制子句>]AS <不带 INTO 的 SELECT 语句>[<DISTRIBUTE 子句>];
    <列定义> ::= <列名> <数据类型> [<列定义子句>] [<STORAGE 子句>][<存储加密子句>]
    <列定义子句> ::= DEFAULT <列缺省值表达式> |
                     <IDENTITY 子句> |
                     <列级约束定义> |
                     DEFAULT <列缺省值表达式> <IDENTITY 子句> |
                     DEFAULT <列缺省值表达式> <列级约束定义> |
                     <IDENTITY 子句> DEFAULT <列缺省值表达式> |
                     <IDENTITY 子句> <列级约束定义> |
                     <列级约束定义> DEFAULT <列缺省值表达式> |
                     <列级约束定义> <IDENTITY 子句> |
                     DEFAULT <列缺省值表达式> <IDENTITY 子句> <列级约束定义> |
                     DEFAULT <列缺省值表达式> <列级约束定义> <IDENTITY 子句> |
                     <IDENTITY 子句> DEFAULT <列缺省值表达式> <列级约束定义> |
                     <IDENTITY 子句> <列级约束定义> DEFAULT <列缺省值表达式> |
                     <列级约束定义> DEFAULT <列缺省值表达式> <IDENTITY 子句> |
                     <列级约束定义> <IDENTITY 子句> DEFAULT <列缺省值表达式>
    <IDENTITY 子句>::=IDENTITY [(<种子>,<增量>)]
    <列级约束定义>::=<列级完整性约束>{,<列级完整性约束>}[<约束属性>]
    <列级完整性约束> ::= [CONSTRAINT <约束名>] [NOT] NULL |<唯一性约束选项>|<引用约束> |
                         CHECK (<检验条件>)
    <唯一性约束选项>::=[PRIMARY KEY ]| [[NOT] CLUSTER PRIMARY KEY] |[CLUSTER [UNIQUE]
KEY] | UNIQUE |
    <引用约束> ::= REFERENCES [PENDANT] [<模式名>.]<表名>[(<列名>{[,<列名>]})]
    [MATCH   <FULL | PARTIAL>] [<引用触发动作>] [WITH INDEX]
    <引用触发动作> ::=
    [<UPDATE  规则>] [<DELETE  规则>] |
    [<DELETE  规则>] [<UPDATE  规则>]
    <UPDATE  规则> ::= ON UPDATE <引用动作>
    <DELETE  规则> ::= ON DELETE <引用动作>
    <引用动作> ::= CASCADE | SET NULL | SET DEFAULT | NO ACTION
    <约束属性> ::= { [<约束检查时间>][<延迟选项>] }| { [<延迟选项>][<约束检查时间>]}
    <延迟选项> ::= DEFERRABLE
    <约束检查时间> ::= INITIALLY <DEFERRABLE | IMMEDIATE>

    <STORAGE 子句> ::= STORAGE(<STORAGE 项> {,<STORAGE 项>})
    <STORAGE 项> ::=
    [INITIAL <初始簇数目>] |
    [NEXT <下次分配簇数目>] |
    [MINEXTENTS <最小保留簇数目>] |
```

```
[ON <表空间名>] |
[FILLFACTOR <填充比例>]|
[BRANCH    <BRANCH 数>]|
[BRANCH    (<BRANCH 数>, <NOBRANCH 数>)]|
[NOBRANCH ]|
[<CLUSTERBTR>]|
[SECTION (<区数>)]|
[STAT NONE]

<存储加密子句> ::= <存储加密子句 1>|<存储加密子句 2>
<存储加密子句 1> ::= ENCRYPT [<加密用法>|<加密用法><加密模式>|<加密模式>]
<存储加密子句 2> ::= ENCRYPT { <加密用法>|<加密用法><加密模式>|<加密模式>}<散列选项>
<加密用法> ::= WITH <加密算法>
<加密模式> ::= <透明加密模式> | <半透明加密模式>
<透明加密模式> ::= AUTO [<加密密码>]
<加密密码> ::= BY <口令>
<半透明加密模式> ::= MANUAL
<散列选项> ::= HASH WITH [<密码引擎名>].<散列算法> [<加盐选项>]
<加盐选项> ::= [NO] SALT
<加密算法> ::= <DES_ECB | DES_CBC | DES_CFB|DES_OFB|DESEDE_ECB|
                DESEDE_CBC | DESEDE_CFB|DESEDE_OFB | AES128_ECB |
                AES128_CBC | AES128_CFB | AES128_OFB | AES192_ECB |
                AES192_CBC | AES192_CFB | AES192_OFB | AES256_ECB |
                AES256_CBC | AES256_CFB | AES256_OFB | RC4>
 <散列算法> ::= <MD5 | SHA1>

<表级约束定义>::=[CONSTRAINT <约束名>] <表级完整性约束>[<约束属性>]
<表级完整性约束> ::=
<唯一性约束选项> (<列名> {,<列名>}) |
FOREIGN KEY (<列名>{,<列名>}) <引用约束> |
CHECK (<检验条件>)
<PARTITION 子句> ::= PARTITION BY <PARTITION 项>
<PARTITION 项> ::=
RANGE (<列名>{,<列名>}) [<子分区模板>{,<子分区模板>}](<范围分区项> {,<范围分区项>})
| HASH  (<列名>{,<列名>})  [<子分区模板>{,<子分区模板>}]{PARTITIONS  <分区数>  [<STORAGE
HASH  子句>]
| HASH(<列名>{,<列名>})[<子分区模板>{,<子分区模板>}] (<HASH 分区项> {,<HASH 分区项>})
| LIST(<列名>)[<子分区模板>{,<子分区模板>}](<LIST 分区项> {,<LIST 分区项>})
| COLUMN (<列名> {,<列名>}) [AS <别名>]
| COLUMN (<列分区项> {,<列分区项>})
<范围分区项>::= PARTITION <分区名> VALUES [EQU OR] LESS THAN (<表达式|MAXVALUE>>{,<
表达式|MAXVALUE>}) [<STORAGE 子句>][<子分区描述项>]
<列分区项>::= (<列名>{,<列名>}) [AS <别名>] [<STORAGE 子句>]
<HASH 分区项>::= PARTITION <分区名> [<STORAGE 子句>][<子分区描述项>]
```

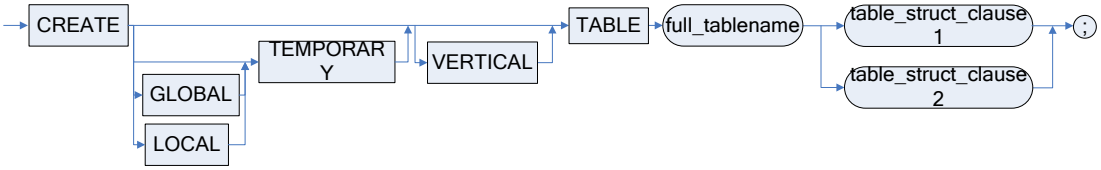
```
<LIST 分区项>::= PARTITION <分区名> VALUES (DEFAULT|<<表达式>,{<表达式>}>) [<STORAGE 子句>][<子分区描述项>]
<子分区描述项> ::=
(<范围子分区描述项>{,<范围子分区描述项>})
|(<HASH 子分区描述项>{,<HASH 子分区描述项>})
| SUBPARTITIONS  <分区数> [<STORAGE HASH  子句>]
|(<LIST 子分区描述项>{,<LIST 子分区描述项>})
<范围子分区描述项>::= <范围子分区项>[<子分区描述项>]
<HASH 子分区描述项>::= <HASH 子分区项>[<子分区描述项>]
<LIST 子分区描述项>::= <LIST 子分区项>[<子分区描述项>]
<范围子分区项>::=
SUBPARTITION <分区名> VALUES [EQU OR] LESS THAN (<表达式|MAXVALUE>){,<表达式|MAXVALUE>}) [<STORAGE 子句>]
<HASH 子分区项>::= SUBPARTITION <分区名> [<STORAGE 子句>]
<LIST 子分区项>::= SUBPARTITION <分区名> VALUES (DEFAULT|<<表达式>,{<表达式>}>) [<STORAGE 子句>]
<子分区模板> ::= <范围子分区模板>|<HASH 子分区模板>|<LIST 子分区模板>
<范围子分区模板> ::= SUBPARTITION BY RANGE (<列名>{,<列名>})[SUBPARTITION TEMPLATE (<范围分区项> {,<范围分区项>})]
<HASH 子分区模板> ::=
SUBPARTITION BY HASH (<列名>{,<列名>})
|SUBPARTITION BY HASH (<列名>{,<列名>})SUBPARTITION TEMPLATE SUBPARTIONS <分区数> [<STORAGE HASH  子句>]
|SUBPARTITION BY HASH (<列名>{,<列名>})SUBPARTITION TEMPLATE (<HASH 分区项>{,<HASH 分区项>})
<LIST 子分区模板> ::= SUBPARTITION BY LIST (<列名>{,<列名>})[SUBPARTITION TEMPLATE (<LIST 分区项> {,<LIST 分区项>})]
<STORAGE HASH  子句>::= STORE IN (<表空间名列表>)
<空间限制子句> ::= DISKSPACE LIMIT <空间大小>|
                   DISKSPACE UNLIMITED
<压缩子句> ::=
COMPRESS |
COMPRESS (<列名> {,<列名>}) |
COMPRESS EXCEPT (<列名> {,<列名>})
<DISTRIBUTE 子句>::=DISTRIBUTED[<RANDOMLY>|<FULLY>]
  |DISTRIBUTED BY [<HASH>](<列名> {,<列名>})
  |DISTRIBUTED BY RANGE (<列名> {,<列名>})(<范围分布项> {,<范围分布项>})
  |DISTRIBUTED BY LIST (<列名> {,<列名>})(<列表分布项> {,<列表分布项>})
<范围分布项>::= VALUES LESS THAN (<表达式>{,<表达式>}) ON <实例名>
                |VALUES EQU OR LESS THAN (<表达式>{,<表达式>}) ON <实例名>
<列表分布项>::= VALUES (<表达式>{,<表达式>}) ON <实例名>
```

参数

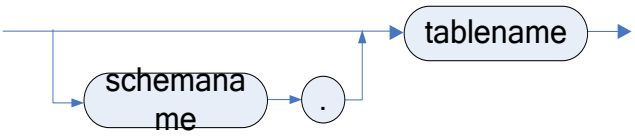
1. <模式名> 指明该表属于哪个模式，缺省为当前模式；
2. <表名> 指明被创建的基表名，基表名最大长度 128 字节；

3. <列名> 指明基表中的列名，列名最大长度 128 字节；
4. <数据类型> 指明列的数据类型；
5. <列缺省值表达式> 如果随后的 INSERT 语句省略了插入的列值，那么此项为列值指定一个缺省值，可以通过 DEFAULT 指定一个值；
6. <列级完整性约束定义>中的参数：
   1) NULL 指明指定列可以包含空值，为缺省选项。
   2) NOT NULL 非空约束，指明指定列不可以包含空值；
   3) UNIQUE 唯一性约束，指明指定列作为唯一关键字；
   4) PRIMARY KEY 主键约束，指明指定列作为基表的主关键字；
   5) CLUSTER PRIMARY KEY 主键约束，指明指定列作为基表的聚集索引主关键字；
   6) NOT CLUSTER PRIMARY KEY 主键约束，指明指定列作为基表的非聚集索引主关键字；
   7) REFERENCES 指明指定列的引用约束，如果有 WITH INDEX 选项，则为引用约束建立索引，否则不建立索引，通过其他内部机制保证约束正确性；
   8) CHECK 检查约束，指明指定列必须满足的条件。
7. <表级完整性约束>中的参数：
   1) UNIQUE 唯一性约束，指明指定列或列的组合作为唯一关键字；
   2) PRIMARY KEY 主键约束，指明指定列或列的组合作为基表的主关键字。指明 CLUSTER，表明是主关键字上聚集索引；指明 NOT CLUSTER，表明是主关键字上非聚集索引；
   3) FOREIGN KEY 指明表级的引用约束，如果使用 WITH INDEX 选项，则为引用约束建立索引，否则不建立索引，通过其他内部机制保证约束正确性；
   4) CHECK 检查约束，指明基表中的每一行必须满足的条件；
   5) 与列级约束之间不应该存在冲突。
8. <检验条件> 指明表中一列或多列可以/不可以接受的数据值或格式；
9. <不带 INTO 的 SELECT 语句> 定义请查看数据查询章节；
10. 组合水平分区表层次最多支持八层；
11. 组合分区表支持自定义子分区描述项，自定义的子分区描述项分区类型与分区列必须与子分区模板一致。如果子分区模板和自定义子分区描述项均指定了分区信息则按自定义分区描述项处理；
12. 垂直分区表不支持组合分区。普通表、LIST 表、列存储表和 HUGE 表均支持组合分区。

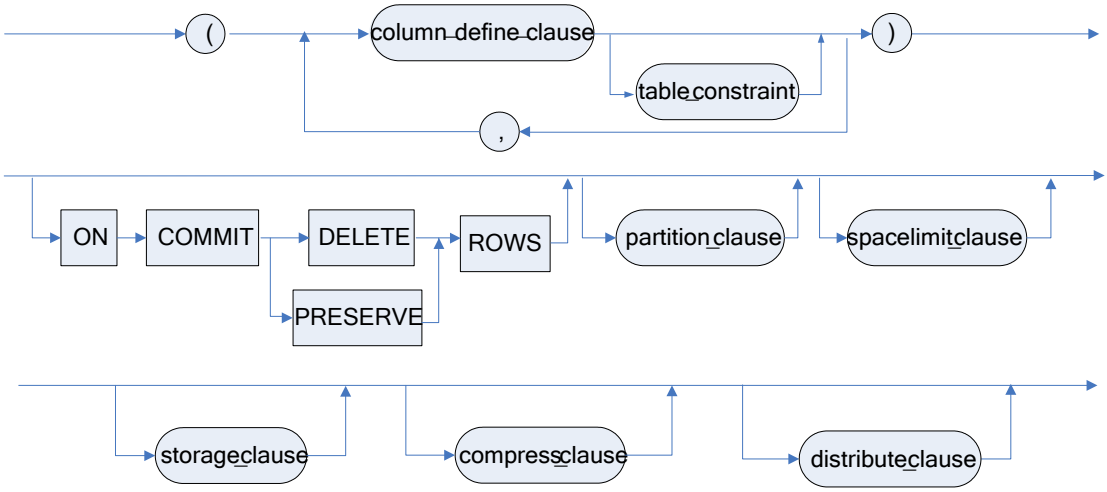
图例



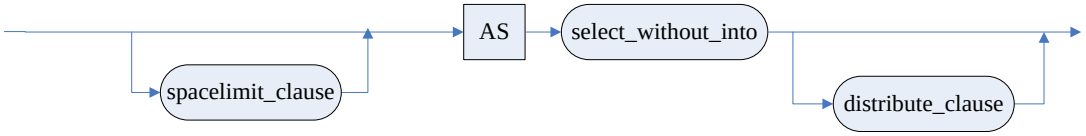
full_tablename



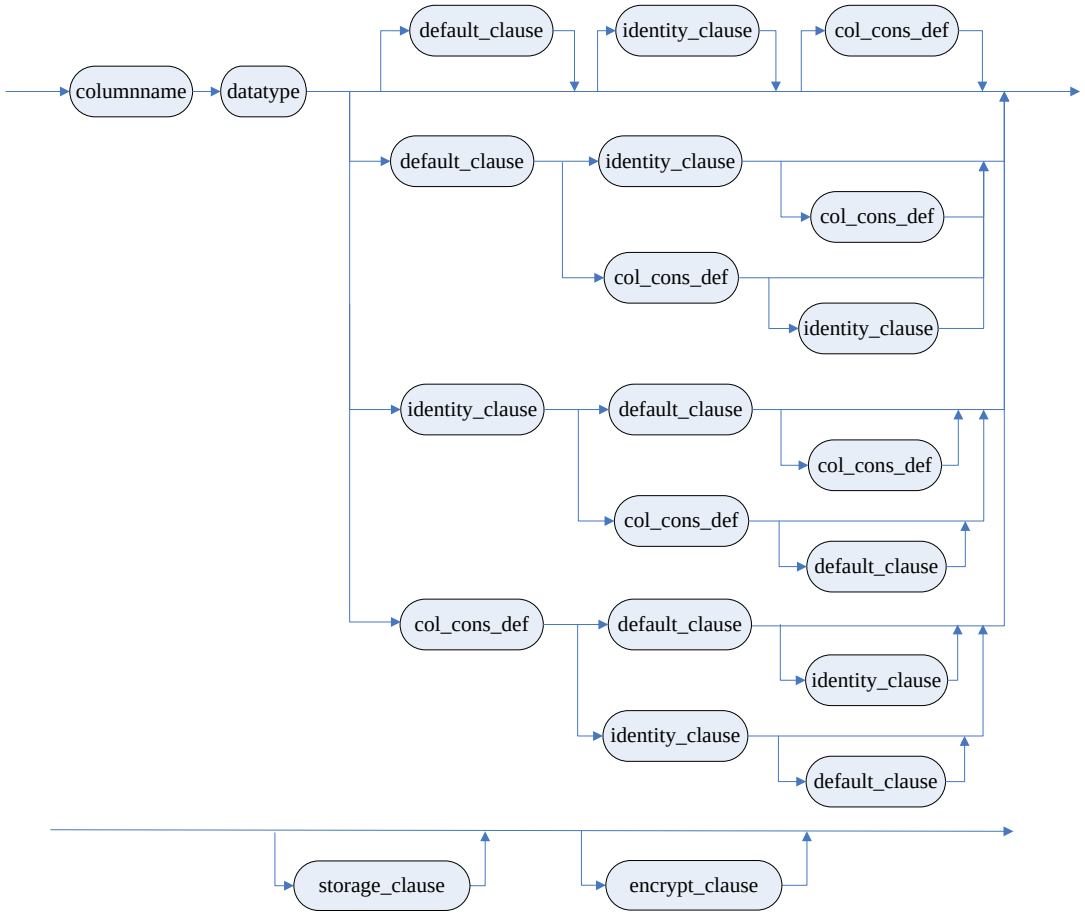
table_struct_clause1



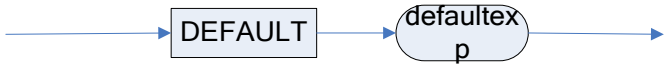
table_struct_clause2



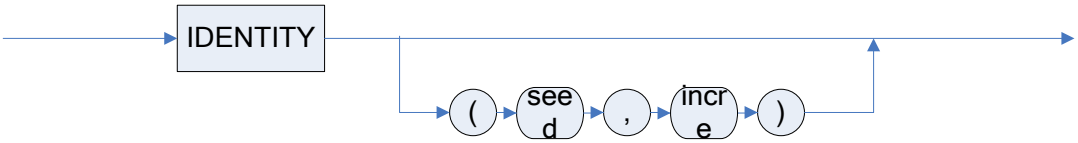
column_define_clause



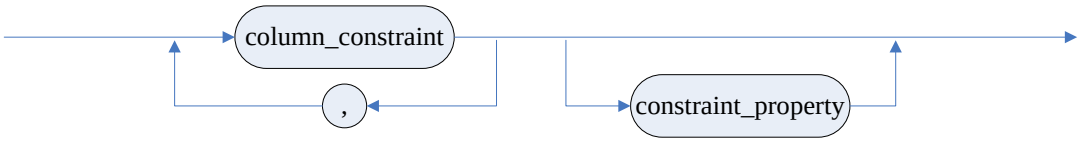
default_clause



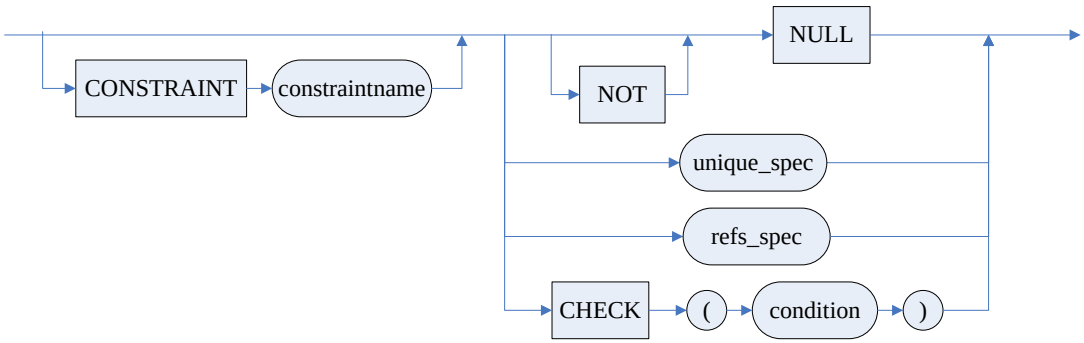
identity_clause



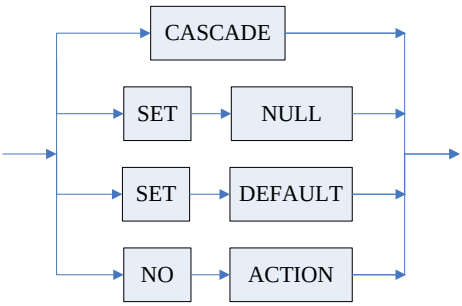
col_cons_def



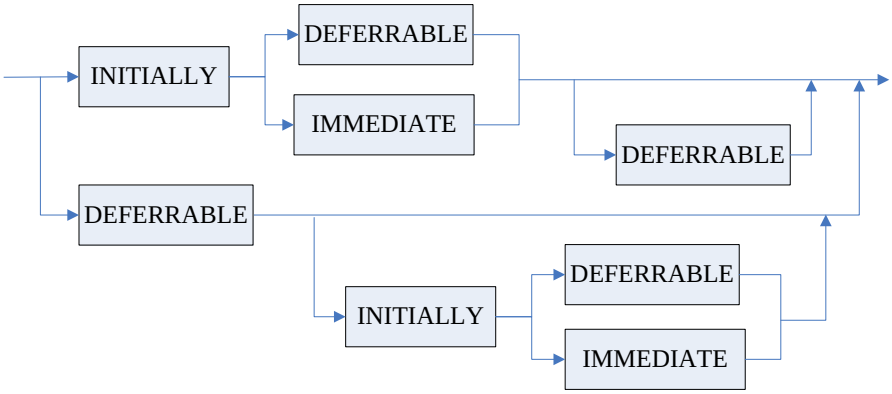
column_constraint



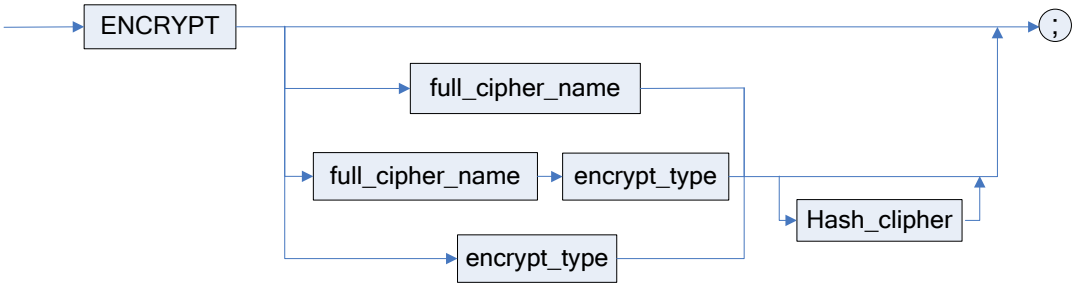
ref_action



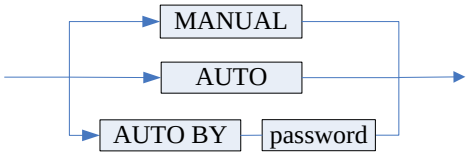
constraint_property



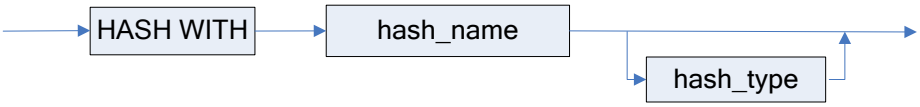
encrypt_caluse



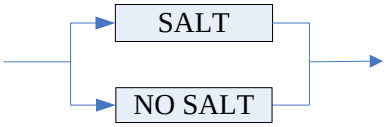
encrypt_type



hash_clipher

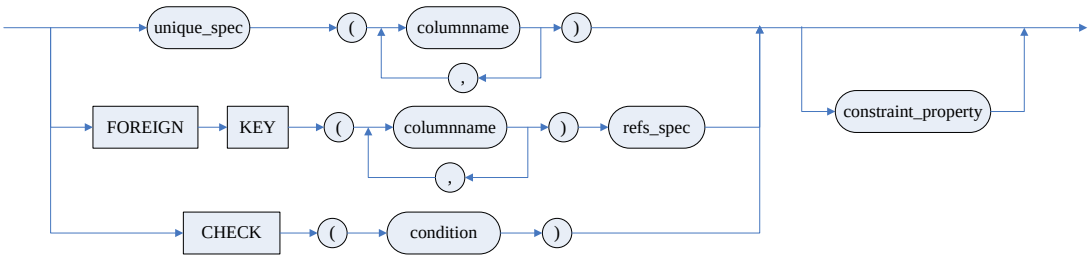


hash_type

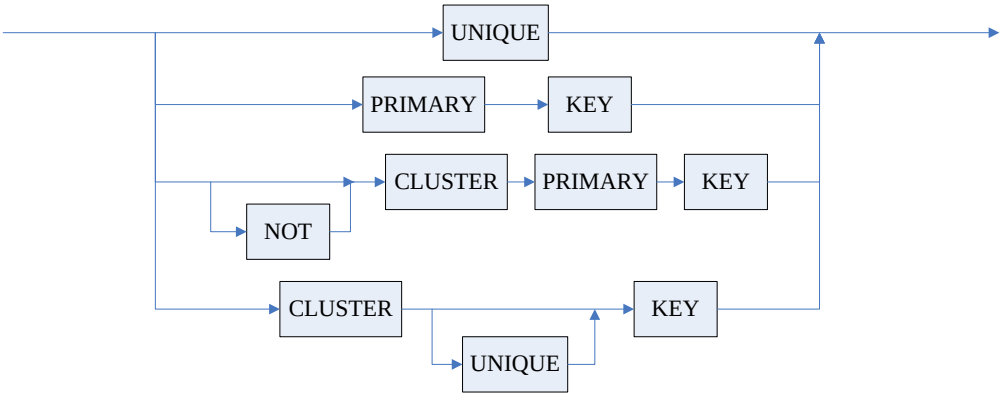


table_constraint

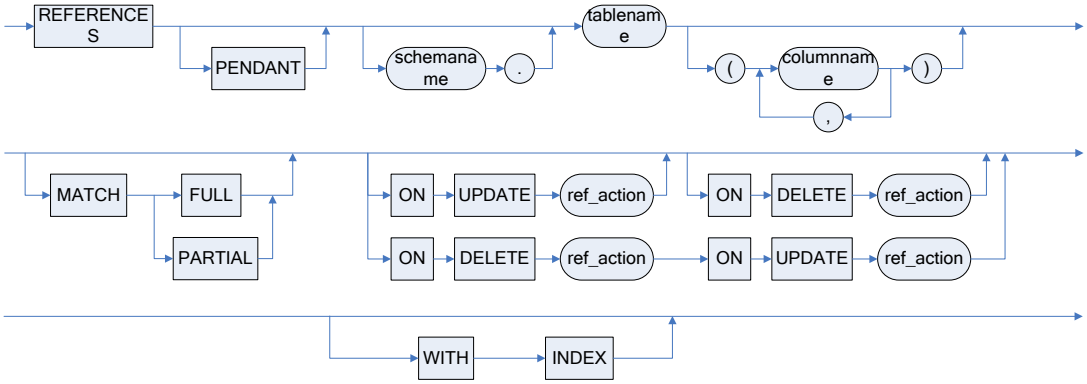
table_constraint_clause



unique_spec

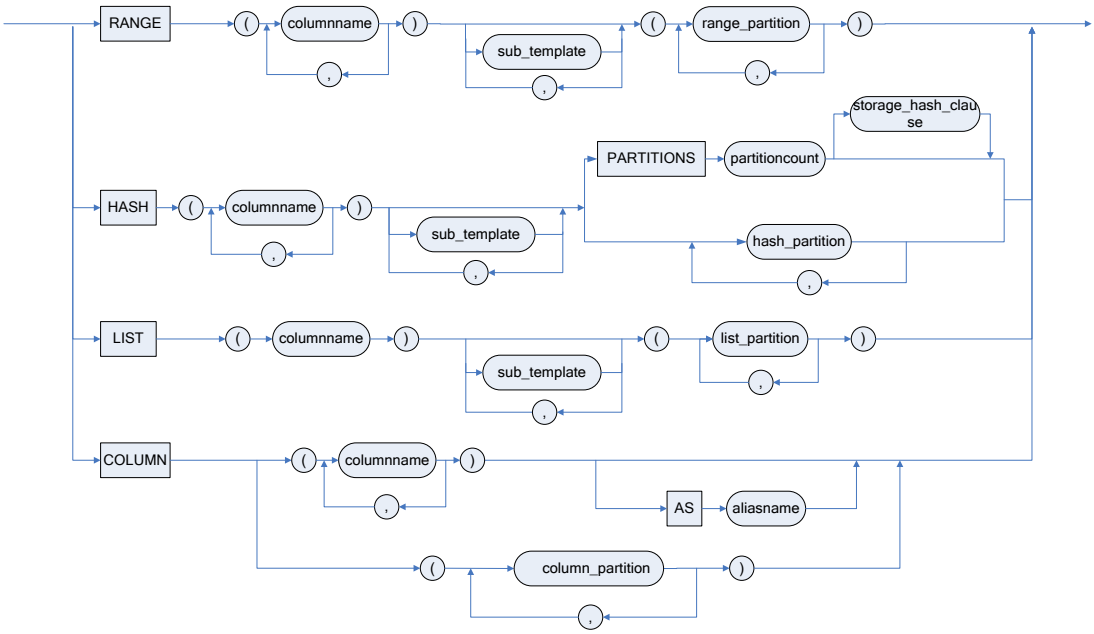


refs_spec

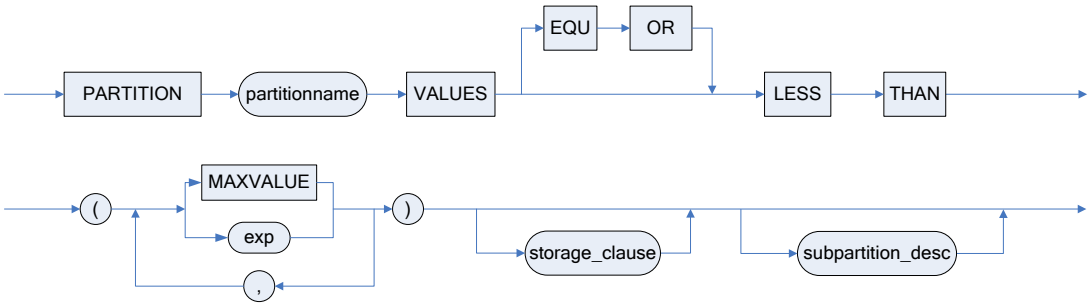


partition_clause

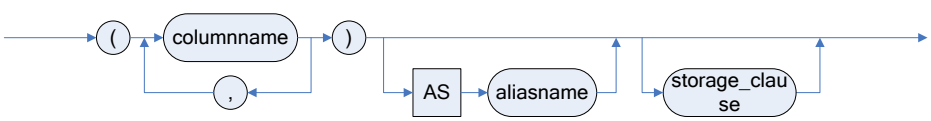
partition_item



range_partition

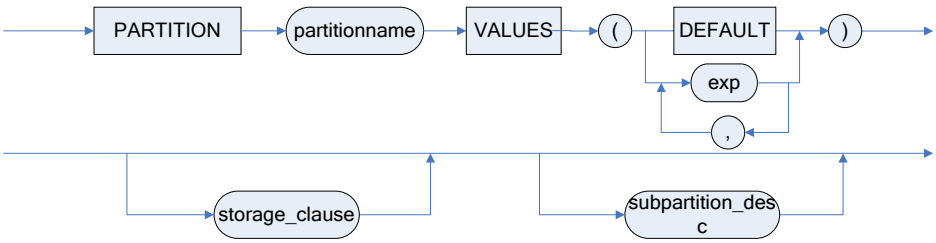


column_partition

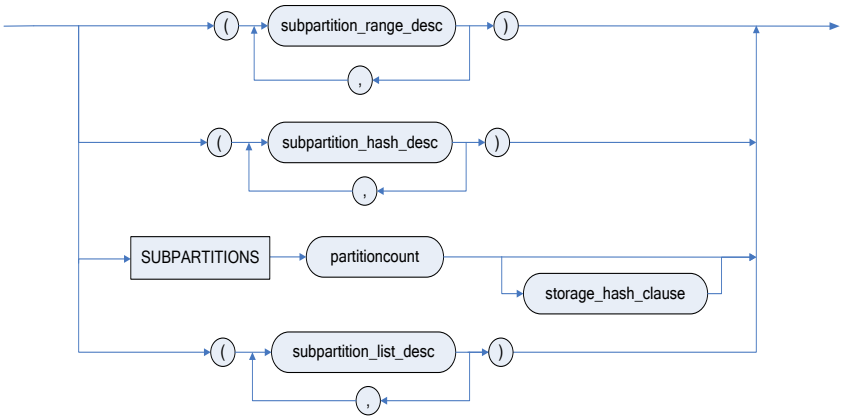


hash_partition

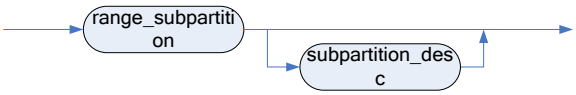
list_partition



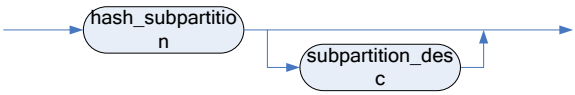
subpartition_desc



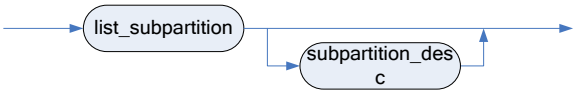
range_subpartition_dec



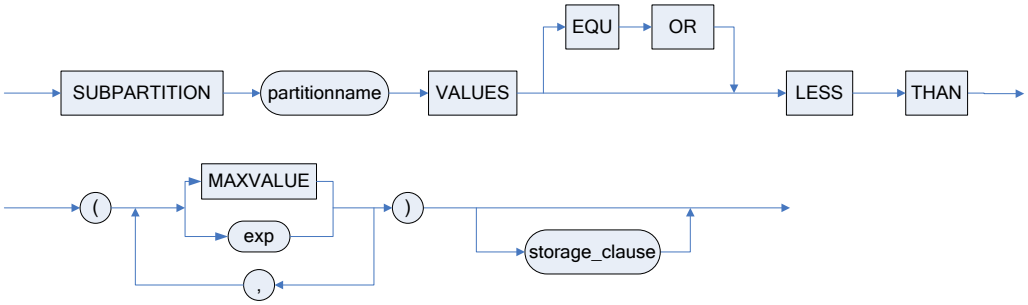
hash_subpartition_dec



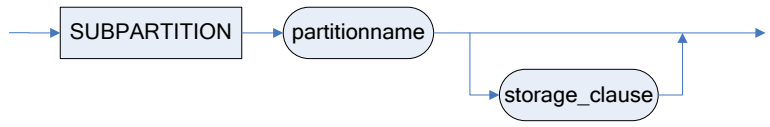
list_subpartition_dec



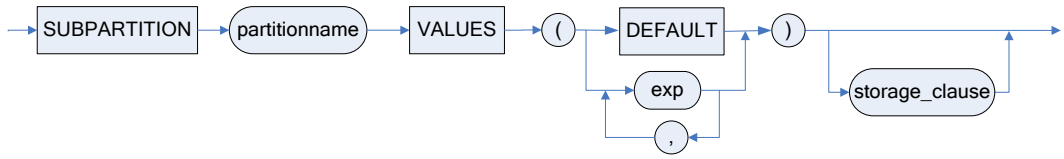
range_subpartition



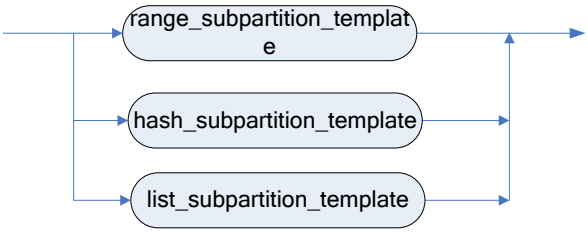
hash_subpartition



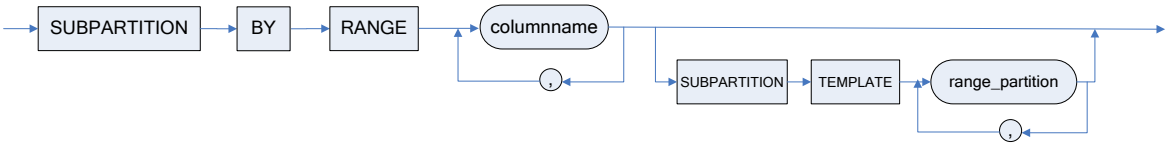
list_subpartition



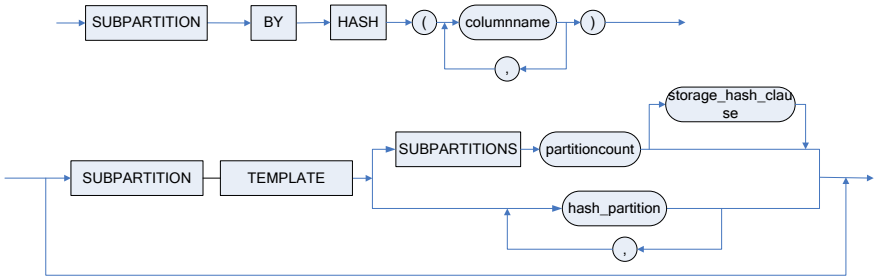
sub_template



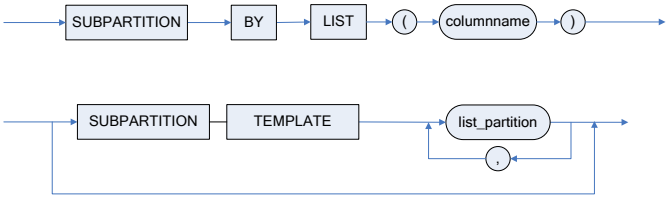
range_subpartition_template



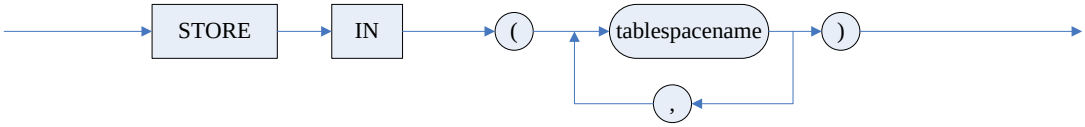
hash_subpartition_template



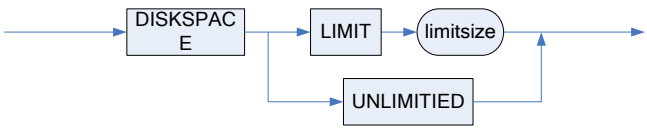
list_subpartition_template



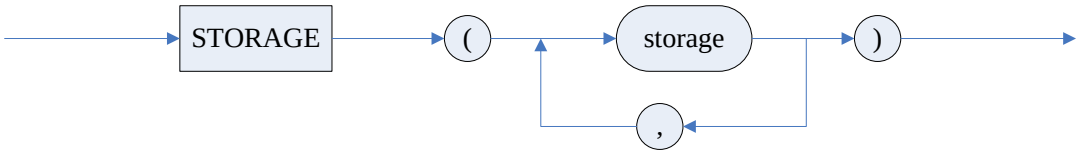
storage_hash_clause



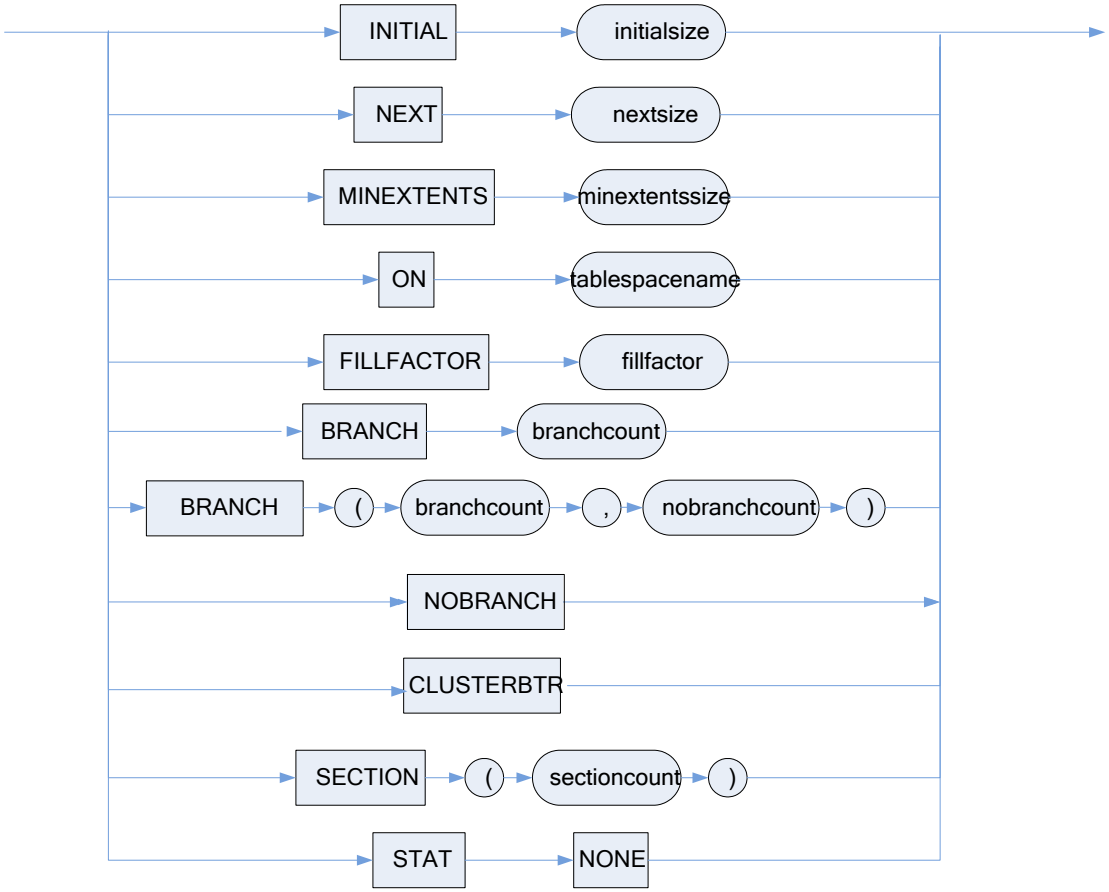
spacelimit_clause



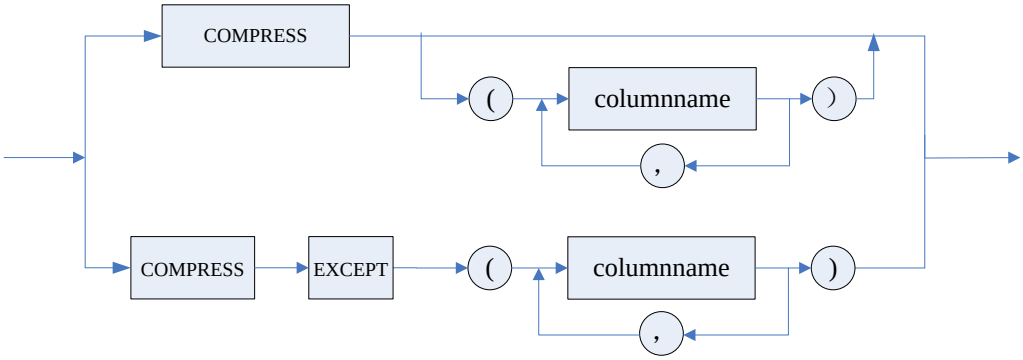
storage_clause



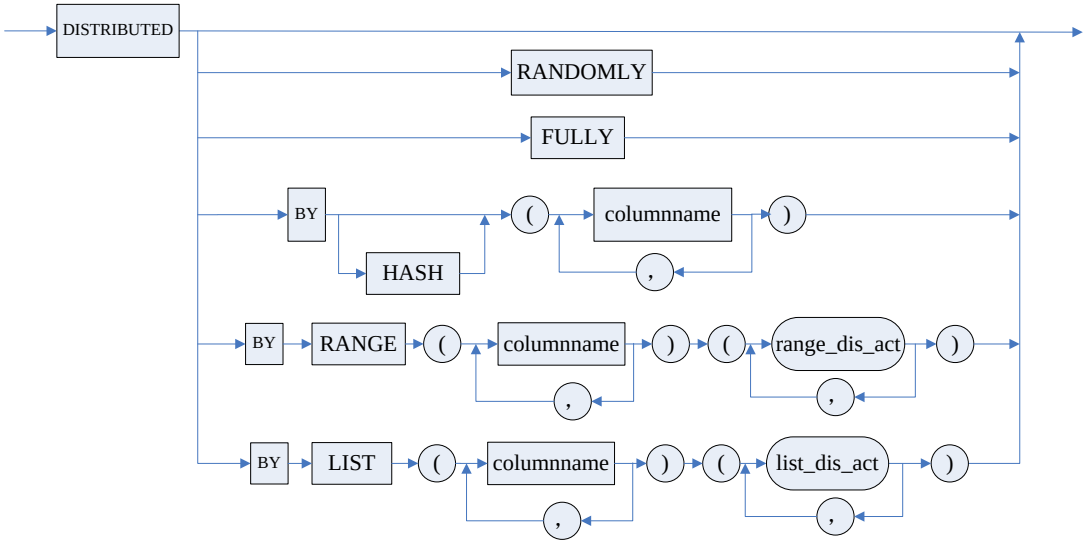
storage



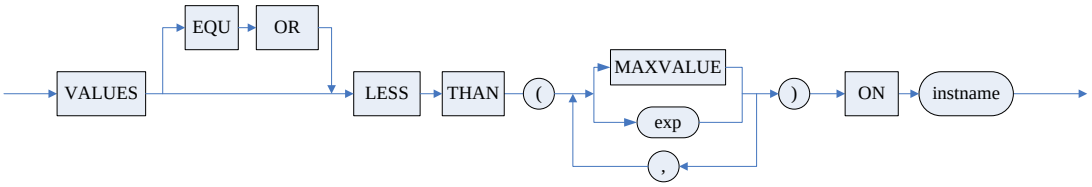
compress_clause



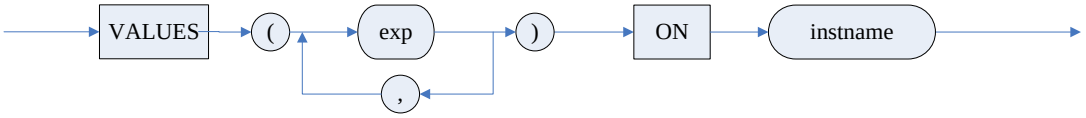
distribute_clause



range_dis_act



list_dis_act



语句功能

供 DBA 或具有 CREATE_TABLE 权限的用户定义基表。

使用说明

1. <表名>指定了所要建立的基表名。在一个<模式>中，<基表名>、<视图名>均不相同。如果<模式名>缺省，则缺省为当前模式。若指定 TEMPORARY，则表示该表为一个临时表，只在一个会话中有效，当一个会话结束，该临时表被自动清空，同时临时表不支持多媒体数据类型。表名需要是合法的标识符，且满足 SQL 语法要求。当表名以“##”开头时，则该表为全局临时表；
2. 表名最大长度为 128 个字符；
3. 所建基表至少要包含一个<列名>指定的列，在一个基表中，各<列名>不得相同。一张基表中至多可以包含 2048 列；
4. <DEFAULT 子句>指定列的缺省值。如果列数据类型为 DATE 类型时，指定无效的缺省值，如 DEFAULT'2005-13-26'，不会对数据进行有效性检查；而设置格式为 DEFAULT DATE '2005-13-26'，则会对数据进行有效性检查；
5. 如果未指明 NOT NULL，也未指明<DEFAULT 子句>，则隐含为 DEFAULT NULL；
6. 自增列不能使用<DEFAULT 子句>；
7. <列缺省值表达式>的数据类型必须与本列的<数据类型>一致；
8. 如果列定义为 NOT NULL，则当该列插入空值时会报错；
9. 约束被 DM 用来对数据实施业务规则，完成对数据完整性的控制。DM_SQL 中主要定义了以下几种类型的约束：非空约束、唯一性约束、主键约束、引用约束和检

查约束。如果完整性约束只涉及当前正在定义的列，则既可定义成列级完整性约束，也可以定义成表级完整性约束；如果完整性约束涉及到该基表的多个列，则只能在语句的后面定义成表级完整性约束。

定义与该表有关的列级或表级完整性约束时，可以用 CONSTRAINT<约束名>子句对约束命名，系统中相同模式下的约束名不得重复。如果不指定约束名，系统将为此约束自动命名。经定义后的完整性约束被存入系统的数据字典中，用户操作数据库时，由 DBMS 自动检查该操作是否违背这些完整性约束条件；

1） 非空约束主要用于防止向一列添加空值，这就确保了该列在表中的每一行都存在一个有意义的值；
   a) 该约束仅用于列级；
   b) 如果定义了列约束为 NOT NULL，则其<列缺省值表达式>不能将该列指定为 NULL；
   c) 空值即为未知的值，没有大小，不可比较。除关键字列外，其列可以取空值。不可取空值的列要用 NOT NULL 进行说明。
2） 唯一性约束主要用于防止一个值或一组值在表中的特定列里出现不止一次，确保数据的完整性；
   a) 唯一性约束是通过唯一索引来实现的。创建了一个唯一索引也就创建了一个唯一性约束；同样的，创建了一个唯一性约束，也就同时创建了一个唯一索引，这种情况下唯一索引是由系统自动创建的；
   b) NULL 值是不参加唯一性约束的检查的。DM 系统允许插入多个 NULL 值。对于组合的唯一性约束，只要插入的数据中涉及到唯一性约束的列有一个或多个 NULL 值，系统则认为这笔数据不违反唯一性约束。
3） 主键约束确保了表中构成主键的一列或一组列的所有值是唯一的。主键主要用于识别表中的特定行。主键约束是唯一性约束的特例；
   a) 主键约束涉及的列必须为非空。通常情况下，DM 系统会自动在主键约束涉及的列上自动创建非空约束；
   b) 每个表中只能有一个主键；
   c) 主键约束是通过创建唯一索引来实现的。DM 系统允许用户自己定义创建主键时，通过 CLUSTER 或 NOT CLUSTER 关键字来指明创建索引的类型。CLUSTER 指明该主键是创建在聚集索引上的，NOT CLUSTER 指明该主键是创建在非聚集索引上的。缺省情况下，主键是创建在非聚集索引上的。
4） 引用约束用于保证相关数据的完整性。引用约束中构成外键的一列或一组列，其值必须至少匹配其参照的表中的一行的一个主键或唯一键值。我们把这种数据的相关性称为引用关系，外键所在的表称为引用表，外键参照的表称为被引用表；
   a) 引用约束指明的被引用表上必须已经建立了相关主键或唯一索引。也就是说，必须保持引用约束所引用的数据必须是唯一的；
   b) 引用约束的检查规则：
      i. 插入规则：外键的插入值必须匹配其被引用表的某个键值。
      ii. 更新规则：外键的更新值必须匹配其被引用表的某个键值。而被引用表数据更新的时候，必须保证所有引用表上的外键值必须有匹配的键值。
      iii. 删除规则：当从被引用表中删除一行数据时，如果定义约束时的选项

是 NO ACTION，就不删除引用表上的相关外键值；如果定义的是 SET NULL 则将引用表上的相关外键值置为 NULL；如果定义的是 CASCADE，那么引用表上的相关外键值将被删除；如里定义的是 SET DEFAULT，则把每个引用列置为“<列缺省值表达式>”规则中所指定的缺省值；

c) NULL 值不参加引用约束的检查。受引用约束的表，如果要插入的涉及到引用约束的列值有一个或多个 NULL 则认为插入值不违反引用约束；

d) MPP 环境下，引用列和被引用列都必需包含分布列，且分布情况完全相同；

e) MPP 环境下，不支持创建 SET NULL 或 SET DEFAULT 约束检查规则的引用约束。

5） 检查约束用于对将要插入的数据实施指定的检查，从而保证表中的数据都符合指定的限制。<检验条件>必须是一个有意义的布尔表达式，其中的每个列名必须是本表中定义的列，但列的类型不得为多媒体数据类型，并且不应包含子查询、集函数。

10. 可以使用 PARTITION 子句指定水平分区和垂直分区，水平分区包括范围分区、HASH 分区和列表分区三种。但是，系统不支持同时进行水平分区和垂直分区；

1) 范围分区：按照分区列的数据范围，确定实际数据存放位置的划分方式；

2) HASH 分区：对分区列值进行 HASH 运算后，确定实际数据存放位置的划分方式，主要用来确保数据在预先确定数目的分区中平均分布；

3) 列表分区：通过指定表中的某个列的离散值集，来确定应当存储在一起的数据；

4) 水平分区使用说明：

a) 分区列类型必须是数值型、字符型或日期型，不支持 BLOB、TEXT、ROWID、BIT、BINARY、VARBINARY、时间间隔等类型和用户自定义类型为分区列；

b) 范围分区和哈希分区的分区键可以多个，最多不超过 16 列；列表分区的分区键必须唯一；

c) 范围分区支持 MAXVALUE 范围值的使用，MAXVALUE 是一个比任何值都大的值；

d) 对于范围分区，增加分区必须在最后一个分区范围值的后面添加，要想在表的开始范围或中间增加分区，应使用 SPLIT PARTITION 语句。

e) 水平分区表指定主键和唯一约束时，分区键必须都包含在主键和唯一约束中，但是全局唯一索引不受此约束；

f) 在水平分区表上执行更新分区键，不允许更新后数据导致跨分区的移动，即不能发生行迁移；

g) 水平分区表不支持临时表；

h) 不能在水平分区表上建立自引用约束；

i) 普通环境中，水平分区表的分区数的上限是 65535；MPP 环境下，水平分区表的分区数上限取决于 INI 参数 MAX_EP_SITES，为 $\log_2$MAX_EP_SITES 的结果向上取整，当 MAX_EP_SITES 为默认值 64 时，分区数上限为 1024；

j) 可以定义主表的 BRANCH 选项，但不能对水平分区子表进行 BRANCH 项设置，子表的 BRANCH 项只能通过主表 继承得到；

k) 水平分区表不支持自增列；

1） 不允许引用水平分区子表作为外键约束。

5) 垂直分区使用说明：
   a) 包含 IDENTITY 列的表不允许定义垂直分区表；
   b) 垂直分区表不允许建立触发器；
   c) 除 CLUSTER KEY 列外，其他任何列只能出现在一个分区中；
   d) 垂直分区表上允许建立引用类型为 NO ACTION 的外键约束；
   e) 垂直分区表上不允许建立除 UNIQUE 外的 CHECK 约束；
   f) 不允许跨分区定义索引；
   g) 一张垂直分区表最多允许有 32 个分区；
   h) 禁止用户对垂直分区表子表的 INSERT/UPDATE/DELETE 直接操作，但允许直接 SELECT；
   i) 禁止通过 ALTER TABLE 方式为垂直分区表增加 PK 约束；
   j) ALTER TABLE 目前只支持 ALTER TABLE RENAME 表名（主表或子表），其他均报错返回；
   k) 垂直分区表不支持建表的 BRANCH 选项；
   l) 垂直分区表上视图不支持更新 CLUSTER KEY 列；
   m) 垂直分区表中所有的大字段列必须位于同一个子表中；
   n) 垂直分区表子表必须至少包含一个非 CLUSTER KEY 的列；
   o) 垂直分区表在建表后不能更改聚集索引；如果建表时没有指定聚集索引，后续也不能再指定；
   p) 垂直分区表上视图的 CHECK 约束不能跨分区，并且此时视图不能嵌套。

11. VERTICAL 表使用说明：
    1) VERTICAL 表可以在建表的时候指定主键；
    2) VERTICAL 表不支持大字段列和类对象的列字段类型；
    3) VERTICAL 表不支持建立除 NULL、NOT NULL 和 UNIQUE 外的任何约束，包括索引；
    4) VERTICAL 表不支持自增列；
    5) VERTICAL 表不允许建立触发器；
    6) VERTICAL 表列支持设置缺省值。
12. LIST 表使用说明：
    1) LIST 表支持水平分区表，不支持垂直分区表和 VERTICAL 表；
    2) LIST 类型的水平分区表各子表必须位于同一个表空间；
    3) LIST 表不支持聚簇索引。
13. 可以使用空间限制子句来限制表的最大存储空间，以 M 为单位，取值范围为 1 到 1048576，关键字 UNLIMITED 表示无限制。系统不支持查询建表情况下指定空间限制；
14. 可以使用 STORAGE 子句指定表的存储信息：
    1) 初始簇数目：指建立表时分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
    2) 下次分配簇数目：指当表空间不够时，从数据文件中分配的簇个数，必须为整数，最小值为 1，最大值为 256，缺省为 1；
    3) 最小保留簇数目：当删除表中的记录后，如果表使用的簇数目小于这个值，就不再释放表空间，必须为整数，最小值为 1，最大值为 256，缺省为 1；
    4) 表空间名：在指定的表空间上建表，表空间必须已存在，缺省为该用户的默认

表空间；

5) 填充比例：指定存储数据时每个数据页和索引页的充满程度，取值范围从 0 到 100。默认值为 0，等价于 100，表示全满填充。插入数据时填充比例的值越低，可由新数据使用的空间就越多；更新数据时填充比例的值越大，更新导致出现的页分裂的几率越大。同样，创建索引时，填充比例的值越低，可由新索引项使用的空间也就越多；
6) BRANCH 和 NOBRANCH：指定 BRANCH 和 NOBRANCH 的个数；
7) 区数：指定 VERTICAL 表的区数，只对 VERTICAL 表有效；
8) CLUSTERBTR：当 INI 参数 LIST_TABLE = 1 时，指定 CLUSTERBTR，则建立的表为普通 B 树表而非 LIST 表。

15. 记录的列长度总和不超过块长的一半，VARCHAR 数据类型的长度是指数据定义长度，实际是否越界还需要判断实际记录的长度，而 CHAR 类型的长度是实际数据长度。与此类似的还有 VARBINARY 和 BINARY 数据类型。因此，对于 16K 的块，可以定义 CREATE TABLE TEST(C1 VARCHAR(8000),C2 INT)，但是不能定义 CREATE TABLE TEST(C1 CHAR(8000)，C2 INT)；
16. DM 具备自动断句功能；
17. 在对列指定存储加密属性时，用于保护用户的数据保存在物理介质之前使用指定的加密算法被加密，防止数据泄露；
18. 加密算法可以是系统中已经存在的算法名称，可选的算法可以在 v$ciphers 中获取，也可以使用第三方加密库中的算法，第三方加密库的实现可参考《DM 程序员手册》的《DM 加密引擎接口编程指南》章节，将已实现的第三方加密动态库(external_crypto.dll 或 libexternal_crypto.so)放到 bin 目录下后重启 DM 服务器即可引用其中的算法；
19. 散列算法。散列算法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。加盐选项可以与散列算法配合使用；
20. 透明加密模式。用透明加密的方式加密列上的数据，在数据库中保存加密该列的密钥，执行 DML 语句的过程中自动获取密钥；
21. 对于列上设置的密钥，系统会根据指定加密算法的密钥长度自动使用相应长度，如果用户设置的密钥长度大于加密算法密钥，实际加解密时会截断，否则在后面添 0 补足；
22. 只能在 MPP 模式下才能建分布表，如果未指定列则默认为随机分布表；
23. 对于复制分布表，不支持更新和删除操作；
24. 对于列表分布表，分布列只能为一列。范围分布表和列表分布表，分布列与分布列值列表必须一致，并且指定的实例名不能重复。

举例说明

例 1 首先回顾一下第二章中定义的基表，它们均是用列级完整性约束定义的格式写出，也可以将唯一性约束、引用约束和检查约束以表级完整性约束定义的格式写出的。假定用户为 SYSDBA，下面以产品的评论表为例进行说明。

```
CREATE  TABLE PRODUCTION.PRODUCT_REVIEW
(
 PRODUCT_REVIEWID INT IDENTITY(1,1),
 PRODUCTID INT NOT NULL,
 NAME VARCHAR(50) NOT NULL,
 REVIEWDATE DATE NOT NULL,
```

```
    EMAIL VARCHAR(50) NOT NULL,
    RATING INT NOT NULL,
    COMMENTS TEXT,
    PRIMARY KEY(PRODUCT_REVIEWID),
    FOREIGN    KEY(PRODUCTID)   REFERENCES   PRODUCTION.PRODUCT(PRODUCTID),
    CHECK(RATING IN(1,2,3,4,5))
);
--注：该语句的执行需在“产品的信息表”已经建立的前提下
```

系统执行建表语句后，就在数据库中建立了相应的基表，并将有关基表的定义及完整性约束条件存入数据字典中。需要说明的是，由于被引用表要在引用表之前定义，本例中的产品的信息表被产品的评论表引用，所以这里应先定义产品的信息表，再定义产品的评论表，否则就会出错。

例 2 建表时指定存储信息，表 PERSON 建立在表空间 FG_PERSON 中，初始簇大小为 5，最小保留簇数目为 5，下次分配簇数目为 2，填充比例为 85。

```
CREATE TABLESPACE FG_PERSON DATAFILE 'FG_PERSON.DBF' SIZE 128;

CREATE TABLE PERSON.PERSON
( PERSONID    INT IDENTITY(1,1) CLUSTER PRIMARY KEY,
SEX CHAR(1) NOT NULL,
NAME VARCHAR(50) NOT NULL,
EMAIL VARCHAR(50),
PHONE VARCHAR(25))
STORAGE
( INITIAL      5,
MINEXTENTS     5,
NEXT       2,
ON          FG_PERSON,
FILLFACTOR     85);
```

例 3 建立如下范围分区表后，表 PRODUCT_INVENTORY 将按照 QUANTITY 列值，被分成 4 个分区。

PARTITION	**P1**	**P2**	**P3**	**P4**
VALUES	QUANTITY<=1	1<QUANTITY<=100	100<QUANTITY<=10000	10000<QUANTITY

```
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL)
PARTITION BY RANGE (QUANTITY)
(
      PARTITION P1 VALUES EQU OR LESS THAN (1),
      PARTITION P2 VALUES EQU OR LESS THAN (100),
      PARTITION P3 VALUES EQU OR LESS THAN (10000),
      PARTITION P4 VALUES LESS THAN (MAXVALUE)          --亦可将 MAXVALUE 替换成 99999
);
```

使用 SPLIT PARTITION 语句在上述范围表中增加分区：

```
alter table PRODUCTION.PRODUCT_INVENTORY SPLIT partition p4 at(20000) INTO(PARTITION P5,PARTITION P6);
```

例 4 建立如下垂直分区表后，将系统内部将创建 5 张表，分别是垂直分区基表 ADDRESS，垂直分区子表 ADDRESS00DMPART、ADDRESS01DMPART、ADDRESS02DMPART 和 ADDRESS03DMPART。分区子表的命名规则是，在主表名之后附加后缀 NDMPART，其中 N 为 00、01、02…。

```
CREATE TABLE PERSON.ADDRESS
    (ADDRESSID INT CLUSTER PRIMARY KEY,
    ADDRESS1 VARCHAR(60) NOT NULL,
    ADDRESS2 VARCHAR(60),
    CITY VARCHAR(30) NOT NULL,
    POSTALCODE VARCHAR(15) NOT NULL)
    PARTITION BY COLUMN((ADDRESS1,ADDRESS2), (CITY, POSTALCODE));
```

例 5 建立如下范围分布表后，表 PRODUCT_INVENTORY 将按照 QUANTITY 列值，被分布到 2 个站点上。

```
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL)
DISTRIBUTED BY RANGE (QUANTITY)
(
    VALUES EQU OR LESS THAN (100) ON EP01,
    VALUES EQU OR LESS THAN (MAXVALUE) ON EP02
);
```

例 6 建立如下列表分布表后，表 PRODUCT_INVENTORY 将按照 LOCATIONID 列值，被分布到 2 个站点上，1，2，3，4 在 EP01 上，5，6，7，8 在 EP02，如果有插入其它值时则报错。

```
CREATE TABLE PRODUCTION.PRODUCT_INVENTORY
(PRODUCTID INT NOT NULL REFERENCES PRODUCTION.PRODUCT(PRODUCTID),
LOCATIONID INT NOT NULL REFERENCES PRODUCTION.LOCATION(LOCATIONID),
QUANTITY INT NOT NULL)
DISTRIBUTED BY LIST (LOCATIONID)
(
    VALUES (1,2,3,4) ON EP01,
    VALUES (5,6,7,8) ON EP02
);
```

例 7 建立如下复制分布表后，表 LOCATION 被分布到 MPP 各个站点上，每个站点上的数据都保致一致。

```
CREATE TABLE PRODUCTION.LOCATION
(LOCATIONID INT IDENTITY(1,1) PRIMARY KEY,
PRODUCT_SUBCATEGORYID INT NOT NULL,
NAME VARCHAR(50) NOT NULL)
DISTRIBUTED  FULLY;
```

3.6.1.2 外部表

需指定如下信息：

1. 表名、表所属的模式名；
2. 列定义；
3. 控制文件路径。

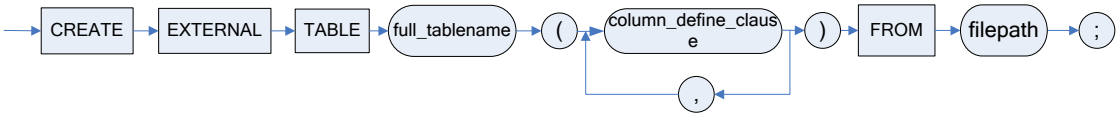
语法格式

```
CREATE EXTERNAL TABLE <表名定义> <表结构定义>;
<表名定义> ::=[<模式名>.]<表名>
<表结构定义>::= (<列定义> {,<列定义>}) FROM <控制文件路径>
<列定义> ::= <列名> <数据类型>
<列定义>  参见 3.6.1 节说明
```

参数

1. <模式名>　　指明该表属于哪个模式，缺省为当前模式；
2. <表名>　　指明被创建的外部基表名；
3. <列名>　　指明基表中的列名；
4. <数据类型>　　指明列的数据类型，暂不支持多媒体类型；
5. <控制文件路径>　　指明使用的控制文件的路径的字符串。

图例



语句功能

供 DBA 或具有 CREATE_TABLE 权限的用户定义外部基表。

使用说明

1. <表名>指定了所要建立的外部基表名。如果<模式名>缺省，则缺省为当前模式。表名需要是合法的标识符。且满足 SQL 语法要求；
2. 外部表的表名最大长度为 128 个字符；
3. 所建外部基表至少要包含一个<列名>指定的列，在一个外部基表中，各<列名>不得相同。一张外部基表中至多可以包含 2048 列；
4. 外部基表不能存在大字段列；
5. 外部基表不能存在任何约束条件；
6. 外部基表不能为临时表，不能建立分区；
7. 外部基表上不能建立任何索引；
8. 外部基表是只读的，不存在表锁，不允许任何针对外部表的增删改数据操作，不允许 truncate 外部表操作；
9. 控制文件路径，以及数据文件路径建议采用绝对路径；
10. 控制文件的格式为：

```
LOAD [DATA]
INFILE [LIST] <path_name>|<path_name_list>
INTO TABLE tablename
FIELDS <delimiter>
```

其中：

<path_name_list> 指明使用的数据文件列表；

tablename　　指明表名，创建外部表时，表名可以与此不同；

<delimiter>　　指明同一行中各个列的分隔符；

11. 数据文件中一行数据必须以回车结束；

12. 外部表支持查询 ROWID、USER 和 UID 伪列，不支持查询 TRXID 伪列。

举例说明

例 1 指定操作系统的一个文本文件作为数据文件，编写控制文件及建表语句。

数据文件（f:\ext_table\data.txt），首行数据如下：

```
a|abc|varchar_data|12.34|12.34|12.34|12.34|0|1|1|1234|1234|1234|100|11|1234|1|1|14.2|
12.1|12.1|1999-10-01|9:10:21|2002-12-12|15
```

控制文件（f:\ext_table\ctrl.txt）如下：

```
LOAD DATA
INFILE   'f:\ext_table\data.txt'
INTO TABLE EXT
FIELDS '|'
```

建表语句：

```
DROP TABLE EXT;
CREATE EXTERNAL TABLE EXT (
L_CHAR CHAR(1),
L_CHARACTER CHARACTER(3),
L_VARCHAR VARCHAR(20),
L_NUMERIC NUMERIC(6,2),
L_DECIMAL DECIMAL(6,2),
L_DEC DEC(6,2),
L_MONEY DECIMAL(19,4),
L_BIT BIT,
L_BOOL BIT,
L_BOOLEAN BIT,
L_INTEGER INTEGER,
L_INT INT,
L_BIGINT BIGINT,
L_TINYINT TINYINT,
L_BYTE BYTE,
L_SMALLINT SMALLINT,
L_BINARY BINARY,
L_VARBINARY VARBINARY,
L_FLOAT FLOAT,
L_DOUBLE DOUBLE,
L_REAL REAL,
L_DATE DATE,
L_TIME TIME,
L_TIMESTAMP TIMESTAMP,
L_INTERVAL INTERVAL YEAR
)FROM 'f:\ext_table\ctrl.txt';
```

系统执行建表语句后，就在数据库中建立了相应的外部基表。查询 ext_table 表：

```
SELECT * FROM EXT;
```

查询结果：

```
L_CHAR L_CHARACTER L_VARCHAR L_NUMERIC L_DECIMAL L_DEC L_MONEY L_BIT L_BOOL L_BOOLEAN L_INTEGER L_INT L_BIGINT L_TINYINT L_BYTE L_SMALLINT L_BINARY L_VARBINARY L_FLOAT L_DOUBLE L_REAL L_DATE L_TIME L_TIMESTAMP L_INTERVAL
a abc varchar_data 12.34 12.34 12.34 12.3400 0 1 1 1234 1234 1234 100 11 1234 0x01 0x01 14.2 12.1 12.1 1999-10-01 09:10:21 2002-12-12 00:00:00.0 INTERVAL '15' YEAR(2)
```

3.6.1.3 HFS 表

语法格式

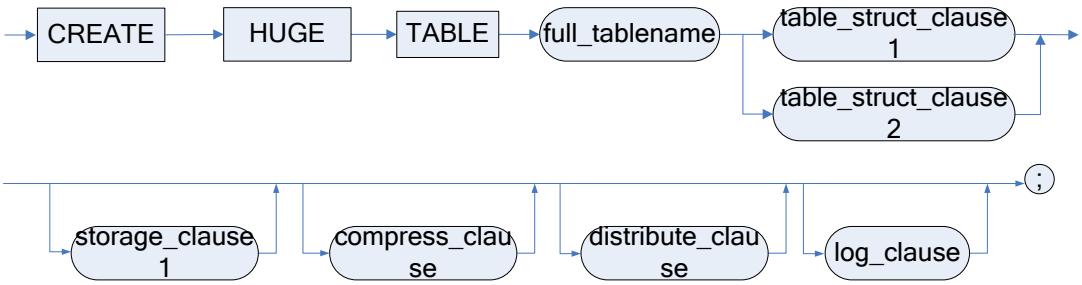
```
CREATE HUGE TABLE <表名定义> <表结构定义>[<PARTITION 子句>] [<STORAGE 子句 1>][<压缩子句] [<DISTRIBUTE 子句>][<日志属性>];
<表名定义> ::= [<模式名>.] <表名>
<表结构定义>::=<表结构定义 1> | <表结构定义 2>
<表结构定义 1>::= (<列定义> {,<列定义>} [<表级约束定义>{,<表级约束定义>}])
<表结构定义 2>::= AS <不带 INTO 的 SELECT 语句>[<DISTRIBUTE 子句>]
<列定义> ::=<列名> <数据类型>[DEFAULT<列缺省值表达式>][<列级约束定义>][<STORAGE 子句 2>][<存储加密子句>]
<表级约束定义>::=[CONSTRAINT <约束名>] <表级完整性约束>[<约束属性>]
<表级完整性约束> ::=
<唯一性约束选项> (<列名> {,<列名>}) |
FOREIGN KEY (<列名>{,<列名>}) <引用约束> |
CHECK (<检验条件>)
<列级约束定义>::=<列级完整性约束>{,<列级完整性约束>}[<约束属性>]
<列级完整性约束> ::= [CONSTRAINT <约束名>] [NOT] NULL |<唯一性约束选项>
<约束属性> ::= { [<约束检查时间>][<延迟选项>] }| { [<延迟选项>][<约束检查时间>]}
<延迟选项> ::= DEFERRABLE
<约束检查时间> ::= INITIALLY <DEFERRABLE | IMMEDIATE>
<唯一性约束选项>::=[PRIMARY KEY] |UNIQUE
<存储加密子句> ::= <存储加密子句 1>|<存储加密子句 2>
<存储加密子句 1> ::= ENCRYPT [<加密用法>|<加密用法><加密模式>|<加密模式>]
<存储加密子句 2> ::= ENCRYPT { <加密用法>|<加密用法><加密模式>|<加密模式>}<散列选项>
<加密用法> ::= WITH <加密算法>
<加密模式> ::= <透明加密模式> | <半透明加密模式>
<透明加密模式> ::= AUTO [<加密密码>]
<加密密码> ::= BY <口令>
<半透明加密模式> ::= MANUAL
<散列选项> ::= HASH WITH [<密码引擎名>].<散列算法> [<加盐选项>]
<加盐选项> ::= [NO] SALT
<加密算法> ::= <DES_ECB | DES_CBC | DES_CFB|DES_OFB|DESEDE_ECB|
                DESEDE_CBC | DESEDE_CFB|DESEDE_OFB | AES128_ECB |
                AES128_CBC | AES128_CFB | AES128_OFB | AES192_ECB |
                AES192_CBC | AES192_CFB | AES192_OFB | AES256_ECB |
```

```
                AES256_CBC | AES256_CFB | AES256_OFB | RC4>
<散列算法> ::= <MD5 | SHA1>
<PARTITION 子句> 参见 3.6.1.1 节
<STORAGE 子句 1> ::= STORAGE([SECTION (<区大小>)][FILESIZE (<文件大小>)][STAT NONE]
ON <HTS 表空间名>)
<STORAGE 子句 2> ::= STORAGE(STAT NONE)
<压缩子句> ::=COMPRESS|COMPRESS (<列名>{,<列名>})|COMPRESS EXCEPT (<列名>{,<列名>})
<DISTRIBUTE 子句>::=DISTRIBUTED|DISTRIBUTED BY (<列名> {,<列名>})
<日志属性>::=LOG NONE|LOG LAST|LOG ALL
```

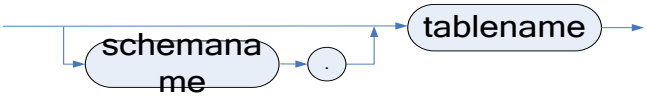
参数

1. <表名> 指明被创建的 HFS 表名，HFS 表名最大长度 124 字节；
2. <区大小> 指一个区的数据行数。区的大小必须是 2 的多少次方，如果不是则向上对齐。取值范围：1024 行~1024*1024 行。不指定则默认值为 65536 行；
3. [STAT NONE] 指定不记录区统计信息，即在修改时不做数据的统计。不指定情况下，默认为做统计信息；
4. <HTS 表空间名> 指要创建的 HFS 表所属的 HTS 表空间。不指定则存储于默认系统 HFS 表空间 HMAIN 中；
5. <文件大小> 指创建 HFS 表时指定的单个文件的大小。取值范围为 16M~1024*1024M。文件大小必须是 2 的多少次方，如果不是则向上对齐。不指定则默认为 64M；
6. <日志属性> 支持通过做日志来保证数据的完整性。完整性保证策略主要是通过数据的镜像来实现的，镜像的不同程度可以实现不同程度的完整性恢复。三种选择：1）LOG NONE：不做镜像。相当于不做数据一致性的保证，如果出错只能手动通过系统函数 SF_REPAIR_HFS_TABLE(模式名,表名)来修复表数据。2）LOG LAST：做部分镜像。但是在任何时候都只对当前操作的区做镜像，如果当前区的操作完成了，这个镜像也就失效了，并且可能会被下一个被操作区覆盖，这样做的好处是镜像文件不会太大，同时也可以保证数据是完整的。但有可能遇到的问题是：一次操作很多的情况下，有可能一部分数据已经完成，另一部分数据还没有来得及做的问题。3）LOG ALL：全部做镜像。在操作过程中，所有被修改的区都会被记录下来，当一次操作修改的数据过多时，镜像文件有可能会很大，但能够保证操作完整性。默认选择为 LOG LAST。

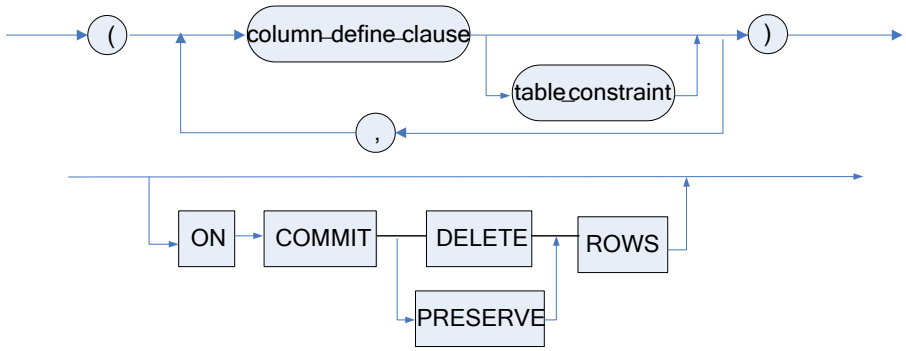
图例



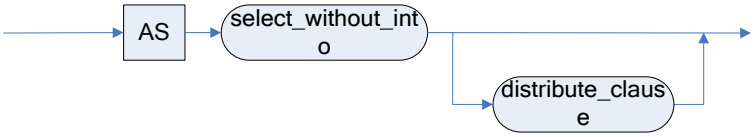
full_tablename



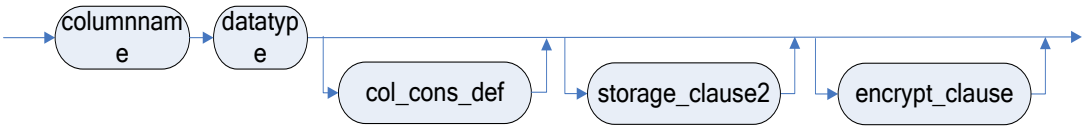
table_struct_clause1



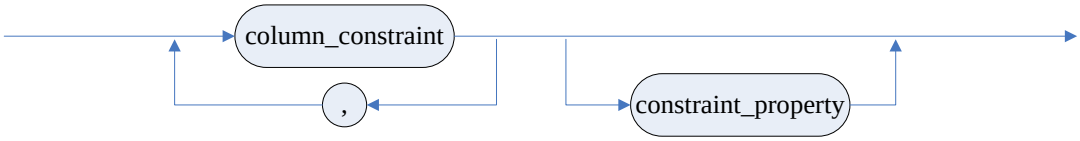
table_struct_clause2



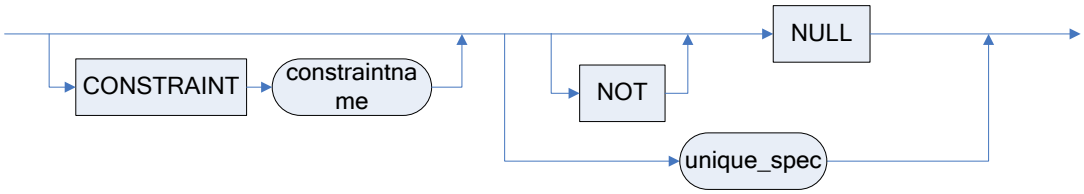
column_define_clause



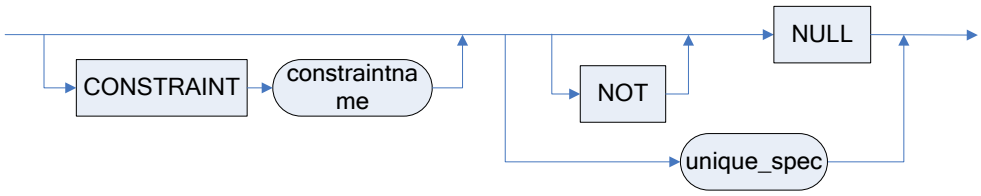
col_cons_def



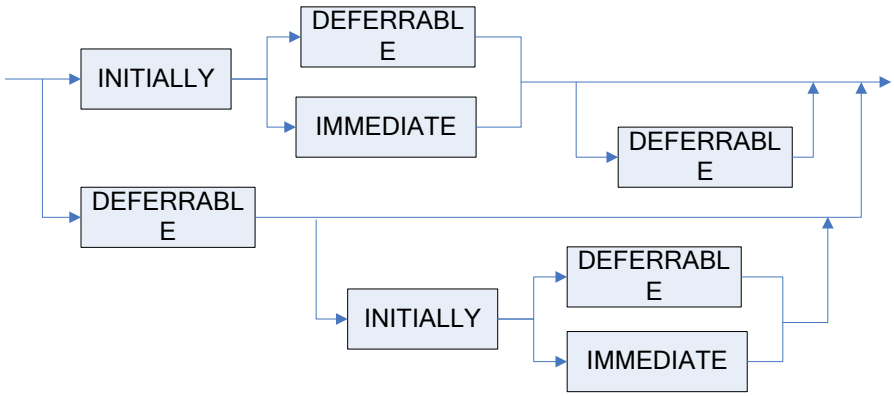
column_constraint



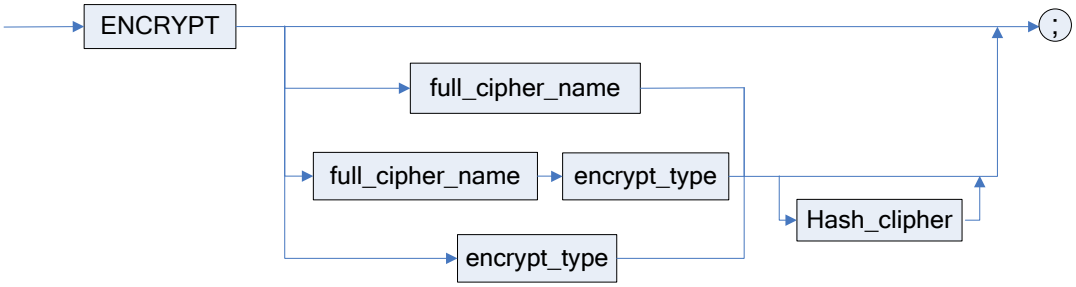
column_constraint



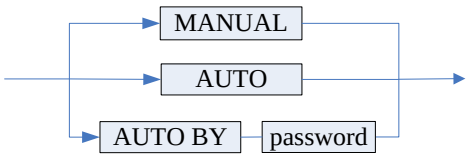
constraint_property



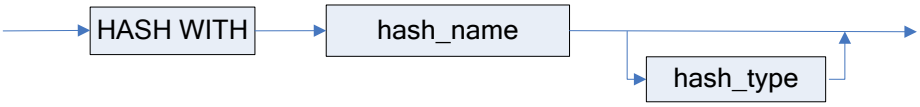
encrypt_caluse



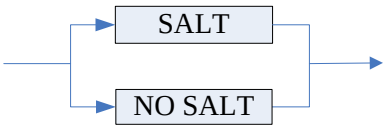
encrypt_type



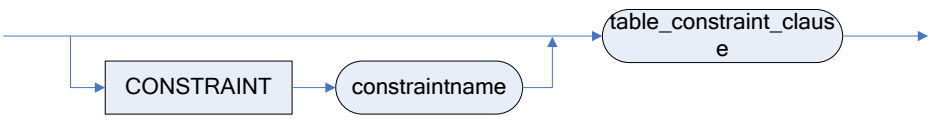
hash_clipher



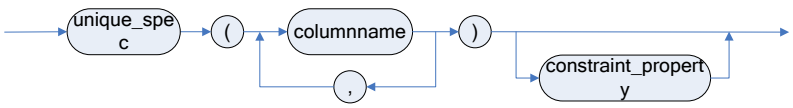
hash_type



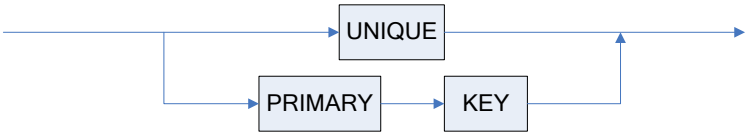
table_constraint



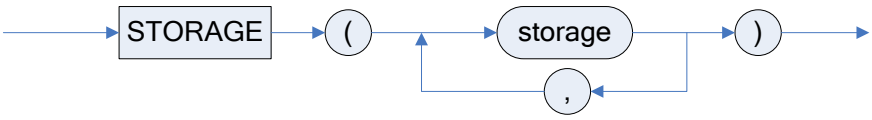
table_constraint_clause



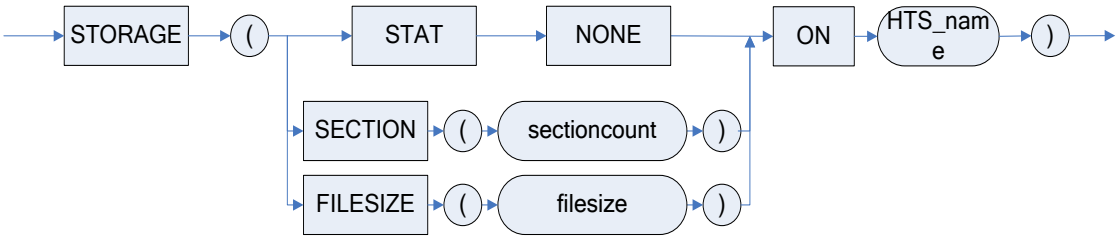
unique_spec



storage_clause1

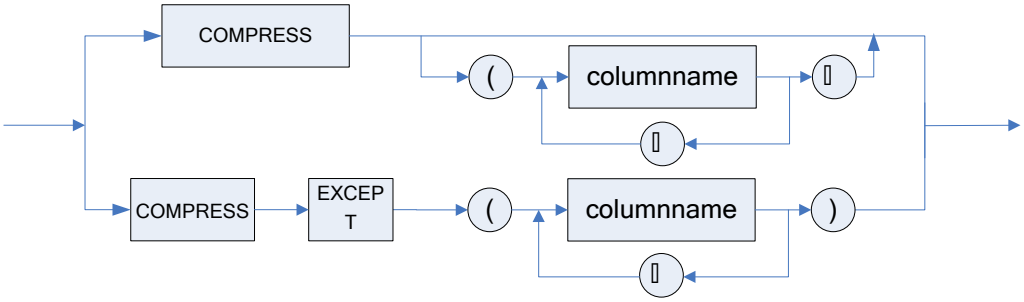


storage

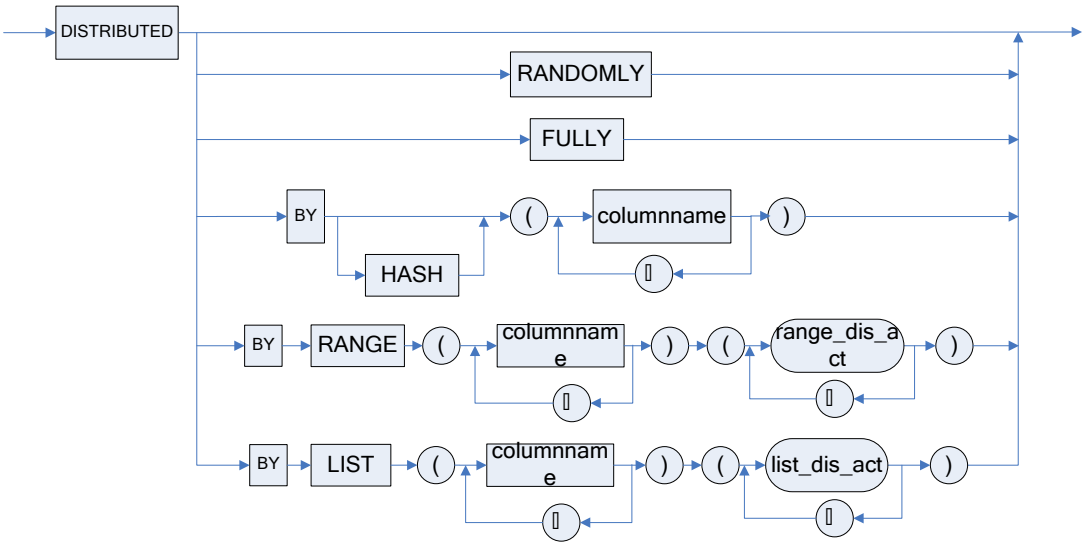


storage_clause2

compress_clause



distribute_clause



语句功能

供 DBA 或具有 CREATE HUGE TABLE 权限的用户创建 HFS 表。

使用说明

1. HFS 表操作时封锁粒度较大，且不支持多版本并发控制；
2. HFS 表的插入、删除与更新操作处理都不能进行回滚；
3. 建 HFS 表时仅支持定义 NULL、NOT NULL、UNIQUE 约束以及 PRIMARY KEY，

后两种约束也可以通过 ALTER TABLE 的方式添加，但这两种约束不检查唯一性；

4. HFS 允许建立二级索引，其中 UNIQUE 索引不检查唯一性；
5. HFS 表不支持事务，没有事务的特性；
6. 不支持 SPACE LIMIT（空间限制）；
7. 不支持 IDENTITY 自增列；
8. 不支持大字段列。

举例说明

例 以 SYSDBA 身份登录数据库后，创建 HFS 表 orders。

```
CREATE HUGE TABLE orders
(
    o_orderkey          INT,
    o_custkey           INT,
    o_orderstatus       CHAR(1),
    o_totalprice        FLOAT,
    o_orderdate         DATE,
    o_orderpriority     CHAR(15),
    o_clerk             CHAR(15),
    o_shippriority      INT,
    o_comment           VARCHAR(79) STORAGE(stat none)
)STORAGE(section(65536) , filesize(64), on HTS_NAME) log all;
```

在这个例子中，ORDERS 表的区大小为 65536 行，文件大小为 64M，指定所在的表空间为 HTS_NAME，做完整镜像，o_comment 列指定的区大小为不做统计信息，其它列（默认）都做统计信息。

3.6.2 基表修改语句

为了满足用户在建立应用系统的过程中需要调整数据库结构的要求，DM 系统提供基表修改语句，对基表的结构进行全面的修改，包括修改基表名、列名、增加列、删除列、修改列类型、增加表级约束、删除表级约束、设置列缺省值、设置触发器状态等一系列修改。系统不提供外部基表的修改语句，如果想更改外部基表的表结构，可以通过重建外部基表来实现。

语法格式

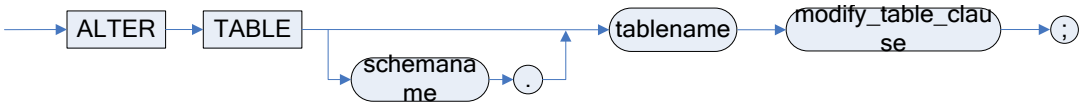
```
ALTER TABLE [<模式名>.]<表名> <修改表定义子句>
<修改表定义子句> ::=
MODIFY <列定义> |
ADD [COLUMN] <列定义> |
DROP [COLUMN] <列名> [RESTRICT | CASCADE] |
ADD [CONSTRAINT [<约束名>] ] <表级约束定义> [<CHECK 选项>] |
DROP CONSTRAINT <约束名> [RESTRICT | CASCADE] |
ALTER [COLUMN] <列名> SET DEFAULT <列缺省值表达式>|
ALTER [COLUMN] <列名> DROP DEFAULT |
ALTER [COLUMN] <列名> RENAME TO <列名> |
ALTER [COLUMN] <列名> SET STAT NONE
ALTER [COLUMN] <列名> SET <NULL | NOT NULL>|
```

RENAME TO <表名> |
ENABLE ALL TRIGGERS |
DISABLE ALL TRIGGERS |
MODIFY <空间限制子句>|
MODIFY CONSTRAINT <约束名> TO <表级约束定义> [<CHECK 选项>]|
ADD <水平分区项>|
ADD<LIST 分区项>|
DROP PARTITION <分区名>|
EXCHANGE PARTITION <分区名> WITH TABLE <表名>|
SPLIT PARTITION <分区名> AT (<表达式>{,<表达式>}) INTO (PARTITION <分区名称 1> <STORAGE 子句>, PARTITION <分区名称 2> <STORAGE 子句>)|
MERGE PARTITIONS <分区编号>,<分区编号> INTO PARTITION <分区名称>|
MERGE PARTITIONS <分区名称>,<分区名称> INTO PARTITION <分区名称>|
DROP IDENTITY|
ADD [COLUMN] <列名>[<IDENTITY 子句>]|
ENABLE CONSTRAINT <约束名> [<CHECK 选项>]|
DISABLE CONSTRAINT <约束名> [RESTRICT | CASCADE] |
<列定义><表级约束定义><范围分区项><LIST 分区项><HASH 分区项><IDENTITY 子句>请参考本章 3.6.1 节相关内容
<CHECK 选项>::=CHECK|NOT CHECK

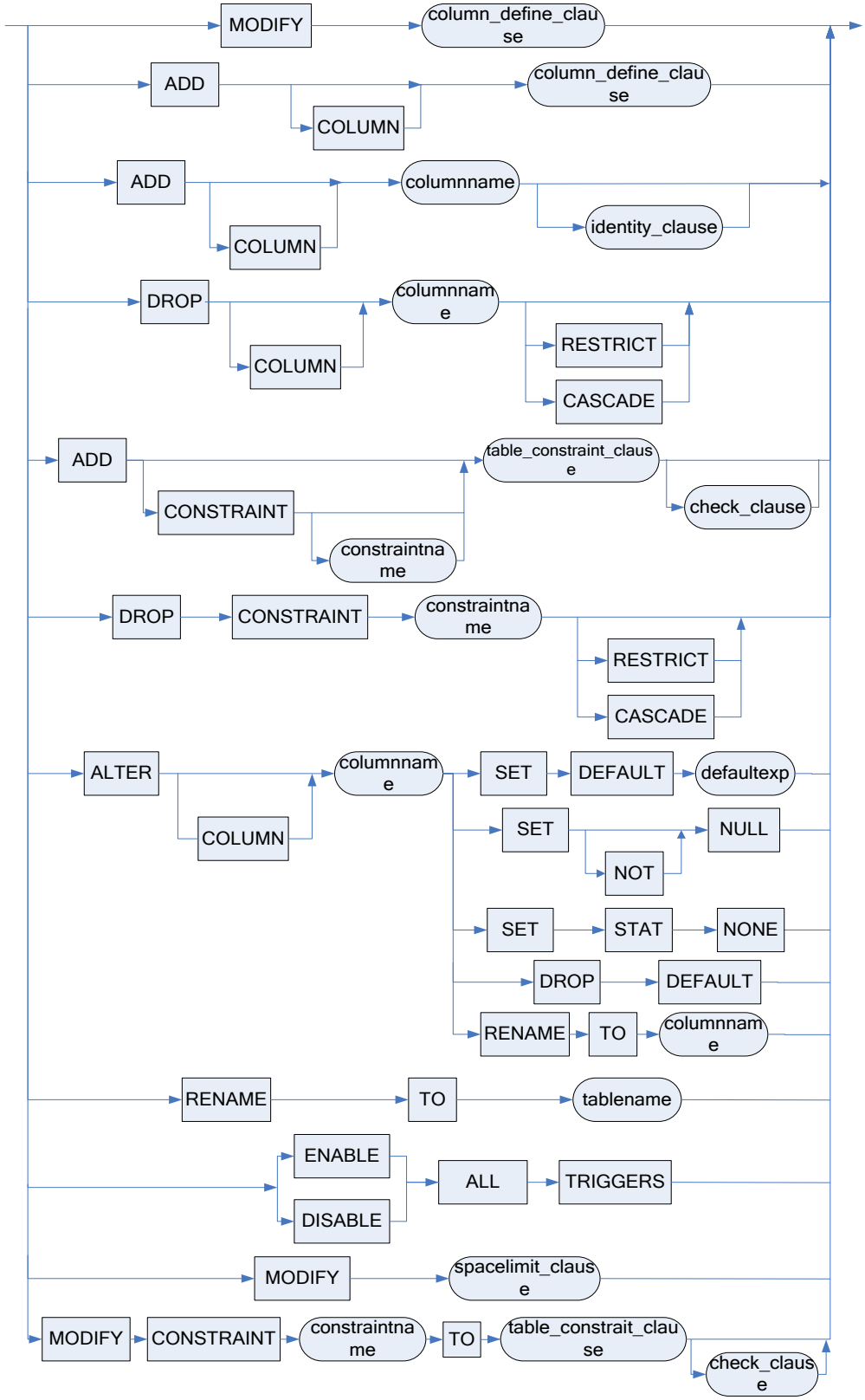
参数

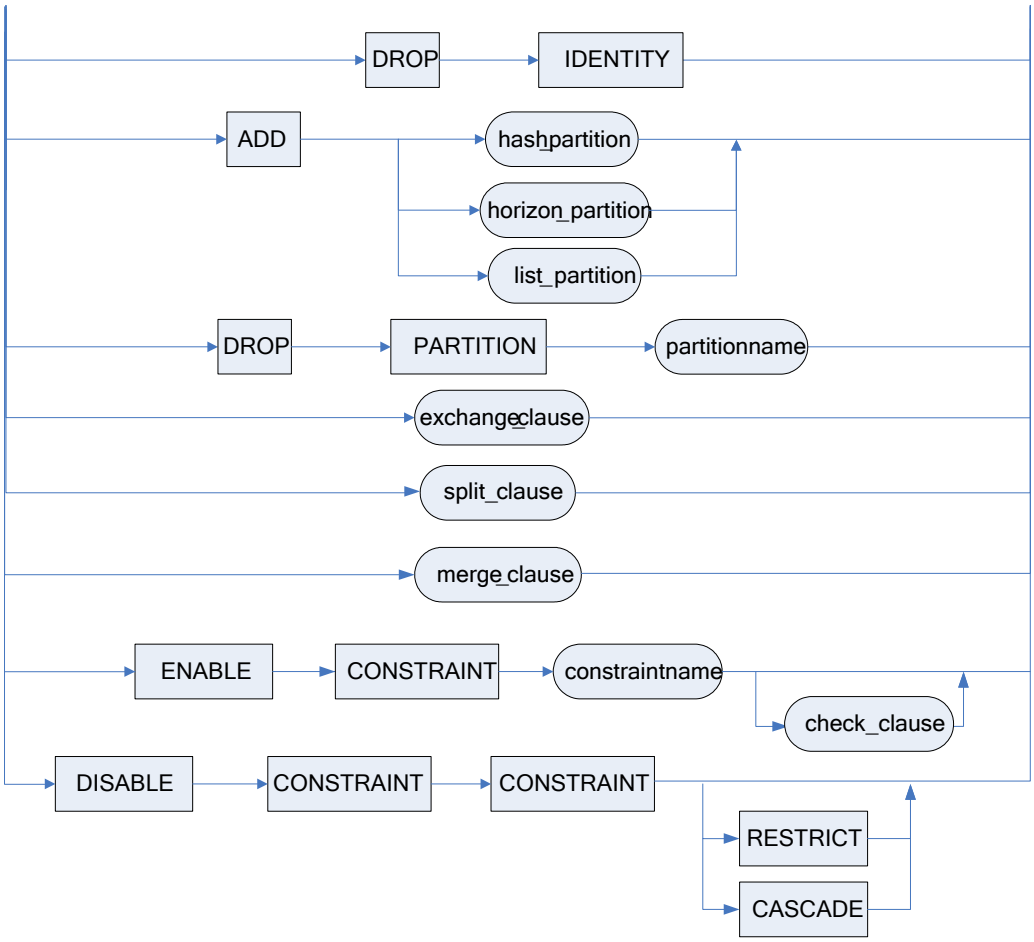
1. <模式名> 指明被操作的基表属于哪个模式，缺省为当前模式；
2. <表名> 指明被操作的基表的名称；
3. <列名> 指明修改、增加或被删除列的名称；
4. <数据类型> 指明修改或新增列的数据类型；
5. <列缺省值> 指明新增/修改列的缺省值，其数据类型与新增/修改列的数据类型一致；
6. <列定义> 相关约束定义，请参考本章 3.6.1 节相关内容；
7. <表级约束定义> 相关约束定义，请参考本章 3.6.1 节相关内容；
8. <CHECK 选项> 设置在添加外键约束的时候，是否对表中的数据进行约束检查；在添加约束和修改约束时，不指明 CHECK 属性，默认 CHECK；而在使约束生效时，不指明 CHECK 属性，默认是 NOT CHECK 的；
9. <分区编号> 从 1 开始，2、3、4……以此类推，编号最大值为：实际分区数。

图例

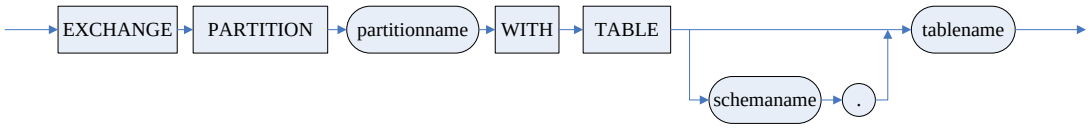


modify_table_clause

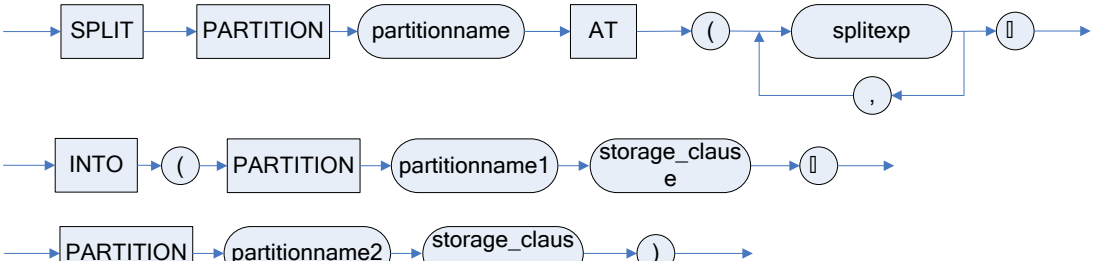

MODIFY
column_define_clause
ADD
COLUMN
column_define_clause
ADD
COLUMN
columnname
identity_clause
DROP
COLUMN
columnname
RESTRICT
CASCADE
ADD
CONSTRAINT
constraintname
table_constraint_clause
check_clause
DROP
CONSTRAINT
constraintname
RESTRICT
CASCADE
ALTER
COLUMN
columnname
SET
DEFAULT
defaultexp
SET
NOT
NULL
SET
STAT
NONE
DROP
DEFAULT
RENAME
TO
columnname
RENAME
TO
tablename
ENABLE
DISABLE
ALL
TRIGGERS
MODIFY
spacelimit_clause
MODIFY
CONSTRAINT
constraintname
TO
table_constrait_clause
check_clause



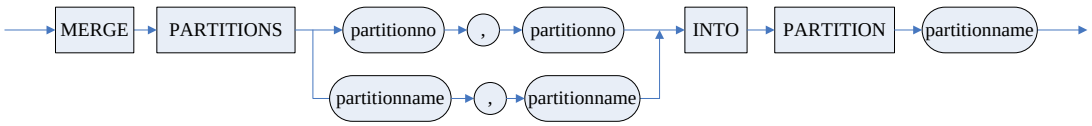
exchange_clause



split_clause



merge_clause



语句功能

供拥有 DBA 权限的用户或该表的建表者对表的定义进行修改，修改包括：

1．修改一列的数据类型、精度、刻度，设置列上的 DEFAULT、NOT NULL、NULL；

2．增加一列及该列上的列级约束；

3．删除一列；
4．增加表上的约束；
5．删除表上的约束；
6．表名/列名的重命名；
7．修改触发器状态；
8．修改表的最大存储空间限制；
9．拆分水平分区；
10．合并水平分区。

使用说明

1. 使用 MODIFY COLUMN 时，不能更改聚集索引的列或者引用约束中引用和被引用的列；
2. 使用 MODIFY COLUMN 时，一般不能更改用于 CHECK 约束的列。只有当该 CHECK 列，都为字符串时，且新列的长度大于旧列长度；或是都为整型时，新列的类型能够完全覆盖旧列的类型（如：char（1）到 char（20），tiny 到 int）
3. 使用 MODIFY COLUMN 子句不能在列上增加 CHECK 约束，能修改的约束只有列上的 NULL，NOT NULL 约束；如果某列现有的值均非空，则允许添加 NOT NULL；属于聚集索引包含的列不能被修改；自增列不允许被修改；
4. 使用 MODIFY COLUMN 修改可更改列的数据类型时，若该表中无元组，则可任意修改其数据类型、长度、精度或量度；若表中有元组，则系统会尝试修改其数据类型、长度、精度或量度，如果修改不成功，则会报错返回；

特殊说明：无论表中有、无元组，多媒体数据类型和非多媒体数据类型都不能相互转换。

5. 修改有默认值的列的数据类型时，原数据类型与新数据类型必须是可以转换的，否则即使数据类型修改成功，但在进行插入等其他操作时，仍会出现数据类型转换错误；
6. 使用 ADD COLUMN 时，新增列名之间、新增列名与该基表中的其它列名之间均不能重复。若新增列跟有缺省值，则已存在的行的新增列值是其缺省值。添加新列对于任何涉及表的约束定义没有影响，对于涉及表的视图定义会自动增加。例如：如果用“*”为一个表创建一个视图，那么后加入的新列会自动地加入该视图；
7. 使用 ADD COLUMN 时，还有以下限制条件：
   1) 列定义中如果带有列约束，只能是对该新增列的约束，但不支持引用约束；列级约束可以带有约束名，系统中同一模式下的约束名不得重复，如果不带约束名，系统自动为此约束命名；
   2) 如果表上没有元组，列可以指定为 NOT NULL；如果表中有元组，对于已有列可以指定同时有 DEFAULT 和 NOT NULL，新增列不能指定 NOT NULL；
   3) 该列可指定为 CHECK；
   4) 该列可指定为 FOREIGN KEY；
   5) 允许向空数据的表中，添加自增列。
8. ADD CONSTRAINT 子句用于添加表级约束。表级约束包括：主键约束（PRIMARY KEY），唯一性约束（UNIQUE），引用约束（REFERENCES），检查约束（CHECK）。添加表级约束时可以带有约束名，系统中同一模式下的约束名不得重复，如果不带约束名，系统自动为此约束命名；

用 ADD CONSTRAINT 子句添加约束时，对于该基表上现有的全部元组要进行约束违规验证：

   1) 添加一个主键约束时，要求将成为关键字的字段上无重复值且值非空，并且表

上没有定义主关键字；

2) 添加一个 UNIQUE 约束时，要求将成为唯一性约束的字段上不存在重复值，但允许有空值；
3) 添加一个 REFERENCES 约束时，要求将成为引用约束的字段上的值满足该引用约束。
4) 添加一个 CHECK 约束或外键时，要求该基表中全部的元组满足该约束。

9. 用 DROP COLUMN 子句删除一列有两种方式：RESTRICT 和 CASCADE。RESTRICT 方式为缺省选项，确保只有不被其他对象引用的列才被删除。无论哪种方式，表中的唯一列不能被删除。RESTRICT 方式下，下列类型的列不能被删除：被引用列、建有视图的列、有 check 约束的列。删除列的同时将删除该列上的约束。CASCADE 方式下，将删除这一列上的引用信息和被引用信息、引用该列的视图、索引和约束；系统允许直接删除 PK 列。但被删除列为 CLUSTER PRIMARY KEY 类型时除外，此时不允许删除；
10. DROP CONSTRAINT 子句用于删除表级约束，表级约束包括：主键约束（PRIMARY KEY）、唯一性约束（UNIQUE）、引用约束（REFERENCES）、检查约束（CHECK）。用 DROP CONSTRAINT 子句删除一约束时，同样有 RESTRICT 和 CASCADE 两种方式。当删除主键或唯一性约束时，系统自动创建的索引也将一起删除。如果打算删除一个主键约束或一个唯一性约束而它有外部约束，除非指定 CASCADE 选项，否则将不允许删除。也就是说，指定 CASCADE 时，删除的不仅仅是用户命名的约束，还有任何引用它的外部约束；
11. 各个子句中都可以含有单个或多个列定义（约束定义），单个列定义（约束定义）和多个列定义（约束定义）应该在它们的定义外加一层括号，括号要配对；
12. 具有 DBA 权限的用户或该表的建表者才能执行此操作；
13. 修改表的最大存储空间限制时，空间大小以 M 为单位，取值范围在表的已占用空间和 1024*1024 之间，还可以利用 UNLIMITED 关键字去掉表的空间限制。如果建表时指定表空间为无限制，则以后不能设置表的空间大小；创建表时如果没有设定空间大小，且以后无法进行限制；
14. 合并分区，将相邻的两个范围分区合并为一个分区。合并分区通过指定分区号进行，相邻的两个分区的表空间没有特殊要求，合并分区后，被合并分区（分区号小的分区）的数据加入合并分区（分区号大的分区）；
15. 拆分分区，将某一个范围分区拆分为相邻的两个分区。拆分分区时指定的常量表达式值不能是原有的分区表范围值。拆分分区时，可以指定分区表空间，也可以不指定；指定的分区表空间不要求一定与原有分区相同；如果不指定表空间缺省使用拆分前分区的表空间；
16. 对于 list 表分区，不论是拆分还是添加分区，都不能改变分区表空间，各子表必须和原表位于同一表空间；
17. 对于哈希分区，除了表名重命名外，不允许其他修改操作；
18. 对范围分区增加分区值必须是递增的，即只能在最后一个分区后添加分区。列表分区增加分区值不能存在于其他已在分区；
19. 当水平分区数仅剩一个时，不允许进行删除分区；
20. 与分区进行分区交换的普通表，必须与分区表拥有相同的列及索引，但交换分区并不会对数据进行校验，即交换后的数据并不能保证数据完整性，如 CHECK 约束；
21. 垂直分区表只支持修改表名，其它修改操作均不支持；
22. 使用 MODIFY COLUMN 不能修改类型为多媒体数据类型的列，也不能将列修改为

多媒体数据类型。

举例说明

例1 产品的评论表中COMMENTS、PRODUCT_REVIEWID、PRODUCTID、RATING几列都不允许修改，分别因为：COMMENTS为多媒体数据类型；PRODUCT_REVIEWID上定义有关键字，属于用于索引的列；PRODUCTID用于引用约束（包括引用列和被引用列）；RATING用于CHECK约束。另外，关联有缺省值的列也不能修改。而其他列都允许修改。假定用户为SYSDBA，如将评论人姓名的数据类型改为VARCHAR(8)，并指定该列为NOT NULL，且缺省值为'刘青'。

```
ALTER TABLE PRODUCTION.PRODUCT_REVIEW MODIFY NAME VARCHAR(8) DEFAULT '刘青' NOT NULL;
```

此语句只有在表中无元组的情况下才能成功。

例2 具有DBA权限的用户需要对EMPLOYEE_ADDRESS表增加一列，列名为ID（序号），数据类型为INT，值小于10000。

```
ALTER TABLE RESOURCES.EMPLOYEE_ADDRESS ADD ID INT PRIMARY KEY CHECK (ID <10000);
```

如果该表上没有元组，且没有PRIMARY KEY，则可以将新增列指定为 PRIMARY KEY。表上没有元组时也可以将新增列指定为UNIQUE，但同一列上不能同时指定PRIMARY KEY和UNIQUE两种约束。

例3 具有DBA权限的用户需要对ADDRESS表增加一列，列名为PERSONID，数据类型为INT，定义该列为DEFAULT和NOT NULL。

```
ALTER TABLE PERSON.ADDRESS ADD PERSONID INT DEFAULT 10 NOT NULL;
```

如果表上没有元组，新增列可以指定为 NOT NULL；如果表上有元组且都不为空，该列可以指定同时有 DEFAULT 和 NOT NULL，不能单独指定为 NOT NULL。

例 4 具有 DBA 权限的用户需要删除 PRODUCT 表的 PRODUCT_SUBCATEGORYID 一列。

```
ALTER TABLE PRODUCTION.PRODUCT DROP PRODUCT_SUBCATEGORYID CASCADE;
```

删除PRODUCT_SUBCATEGORYID这一列必须采用CASCADE方式，因为该列引用了PRODUCT_SUBCATEGORY表的PRODUCT_SUBCATEGORYID。

例5 具有DBA权限的用户需要在PRODUCT表上增加UNIQUE约束，UNIQUE字段为NAME。

```
ALTER TABLE PRODUCTION.PRODUCT ADD CONSTRAINT CONS_PRODUCTNAME UNIQUE(NAME);
```

用ADD CONSTRAINT子句添加约束时，对于该基表上现有的全部元组要进行约束违规验证。在这里，分为三种情况：

1. 如果表商场登记里没有元组，则上述语句一定执行成功；
2. 如果表商场登记里有元组，并且欲成为唯一性约束的字段商场名上不存在重复值，则上述语句执行成功；
3. 如果表商场登记里有元组，并且欲成为唯一性约束的字段商场名上存在重复值，则上述语句执行不成功，系统报错“无法建立唯一性索引”。

如果语句执行成功，用户通过查询

```
SELECT TABLEDEF('PRODUCTION', 'PRODUCT');
```

可以看到，修改后的商场登记的表结构显示为：

```
CREATE TABLE "PRODUCTION"."PRODUCT"
(
```

```
    "PRODUCTID" INTEGER IDENTITY(1, 1) NOT NULL,
    "NAME" VARCHAR(50) NOT NULL,
    "AUTHOR" VARCHAR(25) NOT NULL,
    "PUBLISHER" VARCHAR(50) NOT NULL,
    "PUBLISHTIME" DATE NOT NULL,
    "PRODUCT_SUBCATEGORYID" INTEGER NOT NULL,
    "PRODUCTNO" VARCHAR(25) NOT NULL,
    "SATETYSTOCKLEVEL" SMALLINT NOT NULL,
    "ORIGINALPRICE" DEC(19,4) NOT NULL,
    "NOWPRICE" DEC(19,4) NOT NULL,
    "DISCOUNT" DEC(2,1) NOT NULL,
    "DESCRIPTION" CLOB,
    "PHOTO" BLOB,
    "TYPE" VARCHAR(5),
    "PAPERTOTAL" INTEGER,
    "WORDTOTAL" INTEGER,
    "SELLSTARTTIME" DATE NOT NULL,
    "SELLENDTIME" DATE,
    PRIMARY KEY("PRODUCTID"),
    UNIQUE("PRODUCTNO"),
    CONSTRAINT "CONS_PRODUCTNAME" UNIQUE("NAME"),
    FOREIGN KEY("PRODUCT_SUBCATEGORYID") REFERENCES
"PRODUCTION"."PRODUCT_SUBCATEGORY"("PRODUCT_SUBCATEGORYID")) STORAGE(ON
"MAIN", CLUSTERBTR) ;
```

例6 假定具有DBA权限的用户需要删除PRODUCT表上的NAME的UNIQUE约束。当前的PRODUCT表结构请参见例5。

删除表约束，首先需要得到该约束对应的约束名，用户可以查询系统表 SYSCONS，如下所示。

```
    SELECT NAME FROM SYS.SYSOBJECTS WHERE SUBTYPE$='CONS' AND PID =
    (SELECT ID FROM SYS.SYSOBJECTS WHERE NAME = 'PRODUCT' AND TYPE$='SCHOBJ' AND
SCHID=( SELECT ID FROM SYS.SYSOBJECTS    WHERE NAME='PRODUCTION' AND TYPE$='SCH'));
```

该系统表显示商场登记表上的所有PRIMARY KEY，UNIQUE，CHECK约束。查询得到NAME上UNIQUE约束对应的约束名，这里为CONS_PRODUCTNAME。

然后，可采用以下的语句删除指定约束名的约束。

```
    ALTER TABLE PRODUCTION.PRODUCT DROP CONSTRAINT CONS_PRODUCTNAME;
```

语句执行成功。

例 7 合并分区表，修改分区表。

```
    ALTER TABLE PRODUCTION.PRODUCT_INVENTORY MERGE PARTITIONS P1,P2 INTO
PARTITION P5;
```

执行后，分区结构如下：

```
    PARTITIONNO   P5      P3      P4
    VALUES      QUANTITY<=100      100<UANTITY<=10000     QUANTITY<99999
```

例 8 在例 7 合并分区表基础上，重新拆分分区表，修改分区表。

ALTER TABLE PRODUCTION.PRODUCT_INVENTORY SPLIT PARTITION P3 AT(666)

INTO(PARTITION P6,PARTITION P7);

执行后，分区结构如下：

```
PARITIONNO      P5      P6      P7      P4
VALUES QUANTITY<=100 100<QUANTITY<=666 666<QUANTITY<=10000 10000< QUANTITY
```

3.6.3 基表删除语句

DM 系统允许用户随时从数据库中删除基表。

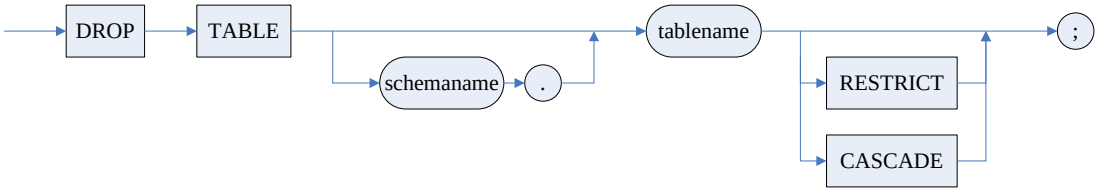
语法格式

```
DROP TABLE [<模式名>.]<表名> [RESTRICT|CASCADE];
```

参数

1. <模式名> 指明被删除基表所属的模式，缺省为当前模式；
2. <表名> 指明被删除基表的名称。

图例



语句功能

供具有 DBA 权限的用户或该表的拥有者删除基表。

使用说明

1. 表删除有两种方式：RESTRICT/CASCADE方式（外部基表除外）。其中RESTRICT为缺省值。如果以CASCADE方式删除该表，将删除表中唯一列上和主关键字上的引用完整性约束，当设置dm.ini中的参数DROP_CASCADE_VIEW值为1时，还可以删除所有建立在该基表上的视图。如果以RESTRICT方式删除该表，要求该表上已不存在任何视图以及引用完整性约束，否则DM返回错误信息，而不删除该表；
2. 该表删除后，在该表上所建索引也同时被删除；
3. 该表删除后，所有用户在该表上的权限也自动取消，以后系统中再建同名基表是与该表毫无关系的表；
4. 删除外部基表无需指定RESTRICT和CASCADE关键字。

举例说明

例 1 用户 SYSDBA 删除 PERSON 表。

```
DROP TABLE PERSON.PERSON CASCADE;
```

例 2 假设当前用户为用户 SYSDBA。现要删除 PERSON_TYPE 表。为此，必须先删除 VENDOR_PERSON 表，因为它们之间存在着引用关系，VENDOR_PERSON 表为引用表，PERSON_TYPE 表为被引用表。

```
DROP TABLE PURCHASING.VENDOR_PERSON;
DROP TABLE PERSON.PERSON_TYPE;
```

也可以使用 CASCADE 强制删除 PERSON_TYPE 表，但是 VENDOR_PERSON 表仍然存在，只是删除了 PERSON_TYPE 表的引用约束。

```
DROP TABLE PERSON.PERSON_TYPE CASCADE;
```

3.6.4 基表数据删除语句

DM 可以从表中删除所有记录。

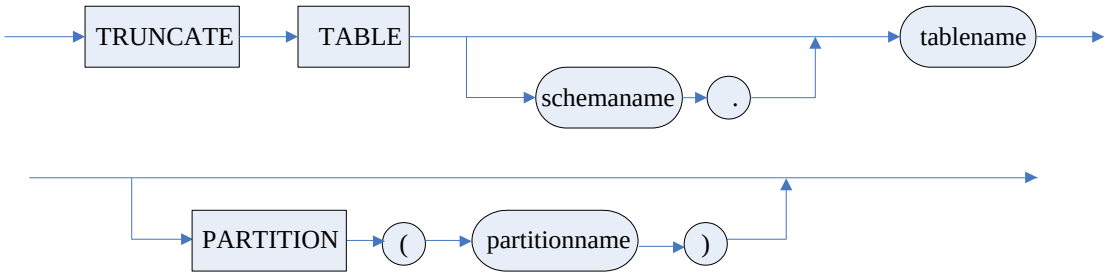
语法格式

```
TRUNCATE TABLE [<模式名>.]<表名>[PARTITION(<分区名>)];
```

参数

1. <模式名> 指明表所属的模式，缺省为当前模式；
2. <表名> 指明被删除记录的表的名称。
3. <分区名> 指明被删除分区表的名称。

图例



语句功能

供具有 DBA 权限的用户或该表的拥有者从表中删除所有记录。

使用说明

1. 用TRUNCATE命令删除记录，它一次性删除指定表的所有记录，比用DELETE命令删除记录快。但TRUNCATE不会触发表上的DELETE触发器；
2. 如果TRUNCATE删除的表上有被引用关系，则此语句失败；
3. TRUNCATE不同于DROP命令，因为它保留了表结构及其上的约束和索引信息；
4. TRUNCATE 命令只能用来删除表的所有的记录，而DELETE命令可以只删除表的部分记录。

举例说明

例 假定删除模式 PRODUCTION 中的 PRODUCT_REVIEW 表中所有的记录。

```
TRUNCATE TABLE PRODUCTION.PRODUCT_REVIEW;
```

3.7 管理索引

3.7.1 索引定义语句

为了提高系统的查询效率，DM 系统提供了索引。但也需要注意，索引会降低那些影响索引列值的命令的执行效率，如 INSERT、UPDATE、DELETE 的性能，因为 DM 不但要维护基表数据还要维护索引数据。

语法格式

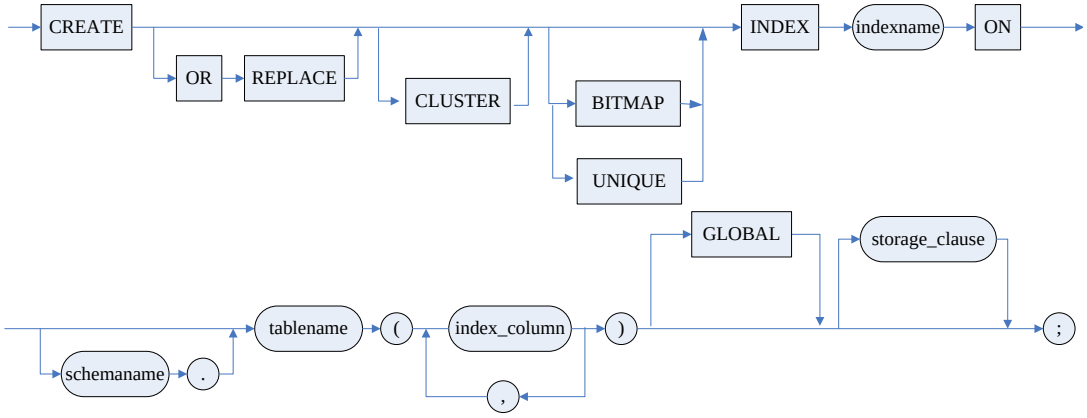
```
CREATE [OR REPLACE] [CLUSTER][UNIQUE | BITMAP] INDEX <索引名>
ON [<模式名>.]<表名>(<索引列定义>{,<索引列定义>}) [GLOBAL] [<STORAGE 子句>];
<索引列定义>::= <索引列表达式>[ASC|DESC]
<STORAGE 子句>::=<STORAGE 子句 1>|<STORAGE 子句 2>
```

```
<STORAGE 子句 1>::= STORAGE(<STORAGE1 项> {,<STORAGE1 项>})
<STORAGE1 项> ::=
[INITIAL <初始簇数目>] |
[NEXT <下次分配簇数目>] |
[MINEXTENTS <最小保留簇数目>] |
[ON <表空间名>] |
[FILLFACTOR  <填充比例>]|
[BRANCH  <BRANCH 数>]|
[BRANCH  (<BRANCH 数>, <NOBRANCH 数>)]|
[NOBRANCH ]|
[<CLUSTERBTR>]|
[SECTION (<区数>)]|
[STAT NONE]
<STORAGE 子句 2>::= STORAGE(<STORAGE2 项> {,<STORAGE2 项>})
<STORAGE2 项> ::=  [ON <表空间名>][STAT NONE]
```

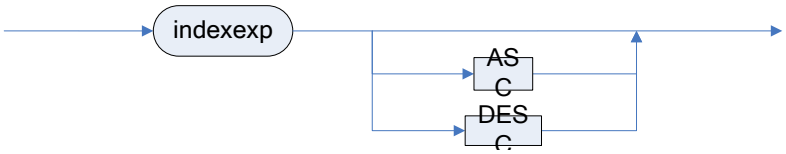
参数

1. UNIQUE　指明该索引为唯一索引；
2. BITMAP　指明该索引为位图索引；
3. CLUSTER　指明该索引为聚簇索引，不能应用到函数索引中；
4. <索引名>　指明被创建索引的名称，索引名称最大长度128字节；
5. <模式名>　指明被创建索引的基表属于哪个模式，缺省为当前模式；
6. <表名>　指明被创建索引的基表的名称；
7. <索引列表达式>　指明被创建的索引列可以为表达式；
8. GLOBAL　指明该索引为全局索引，仅LIST表的水平分区表支持该选项，非水平分区表忽略该选项；
9. ASC　递增顺序；
10. DESC　递减顺序；
11. <STORAGE子句>　普通表的索引参考<STORAGE子句1>，HFS表的索引参考<STORAGE子句2>。

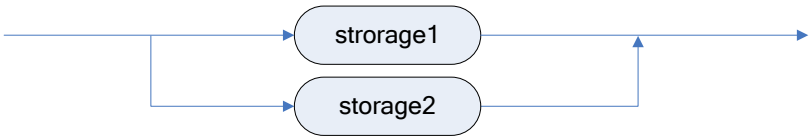
图例



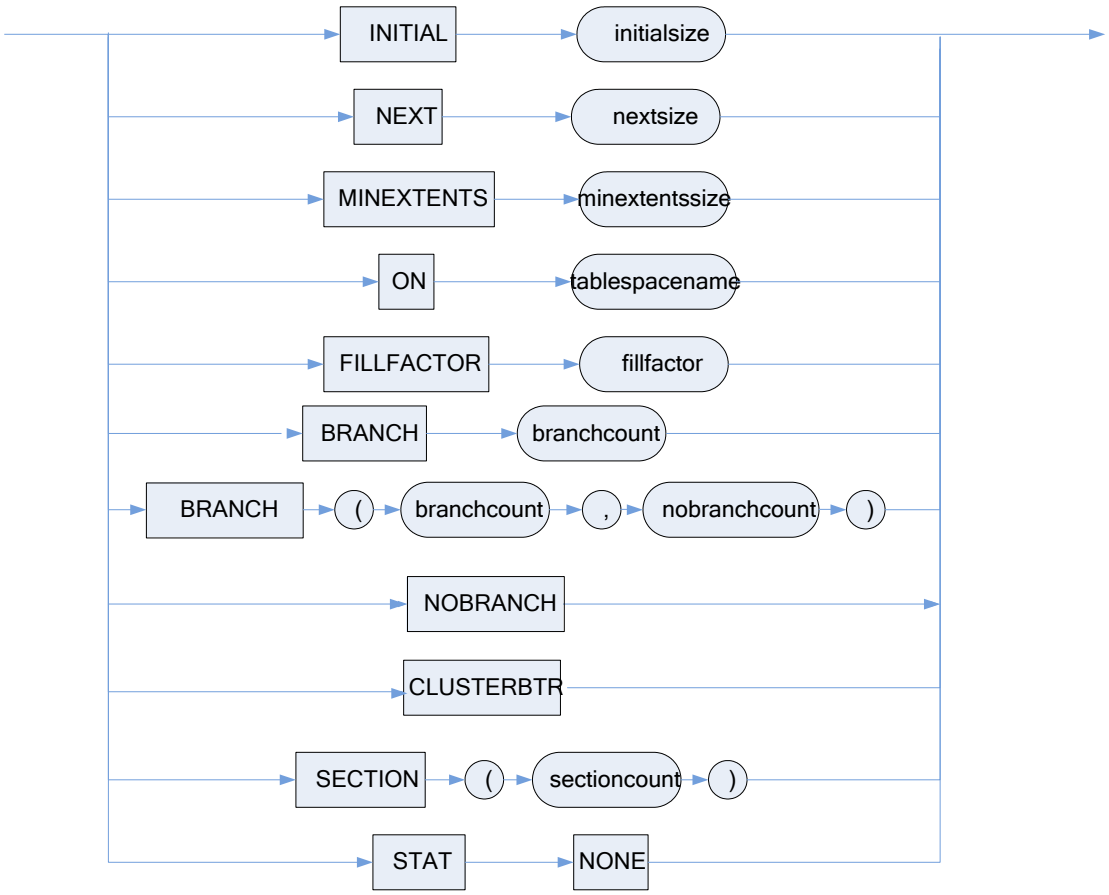
index_column



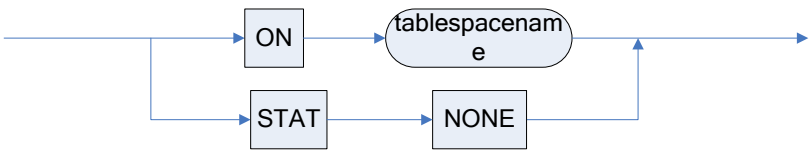
storage



storage1



storage2



语句功能

供 DBA 或该索引所属基表的拥有者且具有 CREATE INDEX 权限的用户定义索引。

使用说明

1. 索引列不得重复出现且数据类型不得为多媒体数据类型；
2. <索引名>不得与该模式中其它索引的名字相同；
3. 可以使用STORAGE子句指定索引的存储信息。它的参数说明参见CREATE TABLE 语句；

4. 索引的默认表空间与其基表的表空间一致；
5. 索引各字段值相加得到的总数据值长度不得超过1020；
6. 在下列情况下，DM利用索引可以提高性能：
   1） 用指定的索引列值来搜索记录；
   2） 用索引列的顺序来存取基表。
7. 每张表中只允许有一个聚集索引；
8. 如果之前已经指定过CLUSTER INDEX或者指定了CLUSTER PK，则必须先删除原有的CLUSTER INDEX方可重新建立CLUSTER INDEX；
9. 指定CLUSTER INDEX操作需要重建表上的所有索引，包括PK索引；
10. 删除聚集索引时，缺省以ROWID排序，自动重建所有索引；
11. 本地索引：其分区方式与其所在基础表的分区方式一模一样的索引。本地索引的每个分区仅对应于其所在基础表的一个分区；
12. 函数索引：创建方式与普通索引一样，并且支持UNIQUE和STORAGE设置项，对于以表达式为过滤的查询，创建合适的函数索引会提升查询效率；
    函数索引具有以下约束：
    1) 表达式可以由多列组成，不同的列不能超过 63 个；
    2) 表达式里面不允许出现大字段列；
    3) 不支持建立分区函数索引；
    4) 函数索引表达式的长度理论值不能超过 816 个字符（包括生成后的指令和字符串）；
    5) 表达式中必须包含表列；
    6) 函数索引不能为 CLUSTER 或 PRIMARY KEY 类型；
    7) 表达式不支持集函数和不确定函数，不确定函数为每次执行得到的结果不确定，系统中不确定函数包括：RAND、SOUNDEX、CURDATE、CURTIME、CURRENT_DATE、CURRENT_TIME、CURRENT_TIMESTAMP、GETDATE、NOW、SYSDATE、CUR_DATABASE、DBID、EXTENT、PAGE、SESSID、UID、USER、TABLEDEF、VIEWDEF、VSIZE、SET_TABLE_OPTION、SET_INDEX_OPTION、UNLOCK_LOGIN、CHECK_LOGIN、GET_AUDIT、CFALGORITHMSENCRYPT、LABEL_TO_CHAR、CFALGORITHMSDECRYPT、BFALGORITHMSENCRYPT、LABEL_FROM_CHAR、LABEL_STR_CMP、BFALGORITHMSDECRYPT、LABEL_CMP；
    8) 快速装载不支持含有函数索引的表；
    9) 当表中含有行前触发器并且该触发器会修改函数索引涉及列的值时，不能建立函数索引。
13. 垂直分区表不允许跨分区定义索引；
14. 在水平分区表中建立唯一索引，分区键必须都包含在索引键中；不能对水平分区表建立唯一函数索引；不能对水平分区子表单独建立索引；
15. 非聚集索引和聚集索引不能使用or replace选项互相转换；
16. 位图索引：创建方式和普通索引一致，对低基数的列创建位图索引，能够有效提高基于该列的查询效率，位图索引具有以下约束：
    1) 支持普通表和 LIST 表创建位图索引；
    2) 不支持对大字段创建位图索引；
    3) 不支持对计算表达式列创建位图索引；

4) 不支持在 UNIQUE 列和 PRIMARY KEY 上创建位图索引；
5) 不支持对存在 CLUSTER KEY 的表创建位图索引；
6) 仅支持单列或者不超过 63 个组合列上创建位图索引；
7) MPP 环境下不支持位图索引的创建；
8) 不支持快速装载建有位图索引的表；
9) 不支持全局位图索引。

举例说明

例 1 假设具有 DBA 权限的用户在 PURCHASING 表中，以 VENDORID 为索引列建立索引 S1，以 ACCOUNTNO，NAME 为索引列建立唯一索引 S2。

```
CREATE INDEX S1 ON PURCHASING.VENDOR (VENDORID);
CREATE UNIQUE INDEX S2 ON PURCHASING.VENDOR (ACCOUNTNO, NAME);
```

例 2 假设具有 DBA 权限的用户在 SALESPERSON 表中，需要查询比去年销售额超过 20 万的销售人员信息，该过滤条件无法使用到单列上的索引，每次查询都需要进行全表扫描，效率较低。如果在 SALESTHISYEAR-SALESLASTYEAR 上创建一个函数索引，则可以较大程度提升查询效率。

```
CREATE INDEX INDEX_FBI ON SALES.SALESPERSON(SALESTHISYEAR-SALESLASTYEAR);
```

3.7.2 索引删除语句

DM 系统允许用户在建立索引后还可随时删除索引。

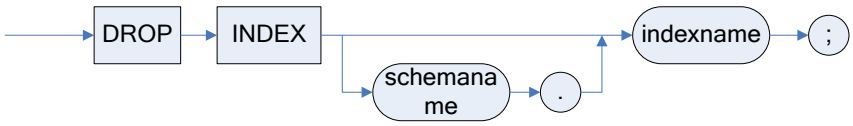
语法格式

```
DROP INDEX [<模式名>.]<索引名>;
```

参数

1. <模式名> 指明被删除索引所属的模式，缺省为当前模式；
2. <索引名> 指明被删除索引的名称。

图例



语句功能

供具有 DBA 角色的用户或该索引所属基表的拥有者删除索引。

使用说明

使用者应拥有 DBA 权限或是该索引所属基表的拥有者。

举例说明

例 具有 DBA 权限的用户需要删除 S1 索引可用以下语句实现。

```
DROP INDEX PURCHASING.S1;
```

3.8 管理位图连接索引

3.8.1 位图连接索引定义语句

位图连接索引是一种提高通过连接实现海量数据查询效率的有效方式，主要用于数据仓

库环境中。它是针对两个或者多个表连接的位图索引，同时保存了连接的位图结果。对于列中的每一个值，该索引保存了索引表中对应行的 ROWID。

语法格式

CREATE [OR REPLACE] BITMAP INDEX <索引名>

ON bitmap_join_index_clause [<STORAGE 子句>];

bitmap_join_index_clause::=[<模式名>.]<表名>(<索引列定义>{,<索引列定义>})FROM [<模式名>.]<基表名>[别名]{,[<模式名>.]<基表名>[别名]} WHERE <条件表达式>;

<STORAGE 子句>请参考本章 3.6.1 节相关内容。

<索引列定义>::= [[<模式名>.]<表名|别名>]<索引列表达式>[ASC|DESC]

参数

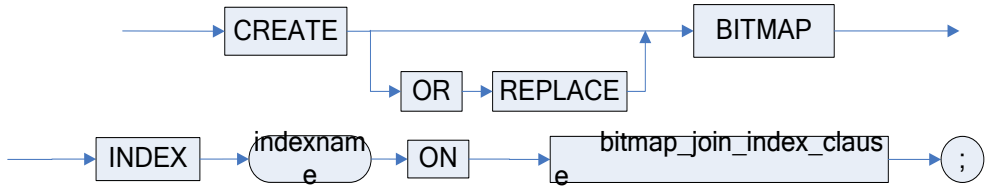
ON 子句：指定的表为事实表，括号内的列既可以是事实表的列也可以是维度表的列；

FROM 子句：指定参与连接的表；

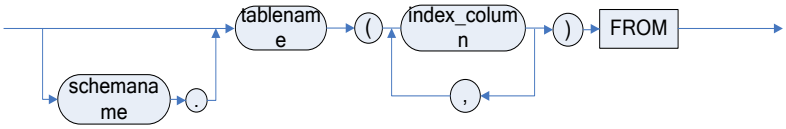
WHERE 子句：指定连接条件；

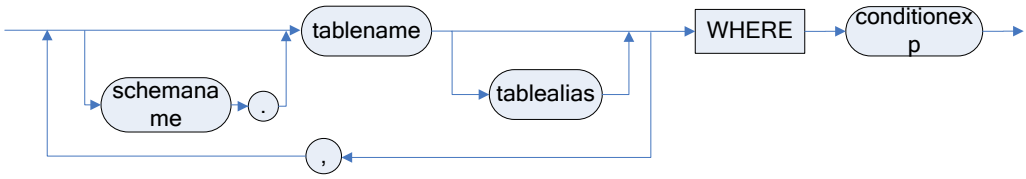
其他参数说明请参考本章 3.7 节。

图例

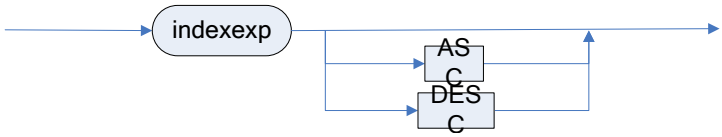


bitmap_join_index_clause





index_column



语句功能

供 DBA 或该索引所属基表的拥有者且具有 CREATE INDEX 权限的用户定义索引。

使用说明

1. 适用于常规索引的基本限制也适用于位图连接索引；
2. 用于连接的列必须是维度表中的主键或存在唯一约束；如果是复合主键，则必须使用复合主键中的所有列；
3. 当多个事务同时使用位图连接索引时，同一时间只允许更新一个表；
4. 连接索引创建时，基表只允许出现一次；
5. 不允许对存在 cluster key 的表创建位图连接索引；

6. 位图连接索引表（命名为 BMJ$_索引 ID）仅支持 select 操作，其他操作都不支持：如 insert、delete、update、alter、drop 和建索引等；
7. 位图连接索引表和位图连接索引所在的事实表以及维度表都不支持表级备份还原；
8. 不支持位图连接索引表、位图连接索引以及虚索引的导出导入；
9. 位图连接索引及其相关表不支持快速装载；
10. 位图连接索引名称的长度限制为：事实表名的长度+索引名称长度+6<128；
11. 仅支持普通表及 list 表；
12. WHERE 条件只能是列与列之间的等值连接，并且必须含有所有表；
13. mpp 环境下暂不支持位图连接索引的创建与查询；
14. 事实表上聚集索引和位图连接索引不能同时存在；
15. 不支持对含有位图连接索引的表中的数据执行 DML，如需要执行 DML,则先删除该索引；
16. 不支持对含有位图连接索引的表执行部分修改操作：删除、修改表约束，删除、修改列，更改表名；
17. 不允许对含有位图连接索引的表并发操作。

举例说明

创建位图连接索引：

```
create bitmap index SALES_CUSTOMER_NAME_IDX
on SALES.SALESORDER_HEADER(SALES.CUSTOMER.PERSONID)
from     SALES.CUSTOMER, SALES.SALESORDER_HEADER
where    SALES.CUSTOMER.CUSTOMERID = SALES.SALESORDER_HEADER.CUSTOMERID;
```

执行查询：

```
Select TOTAL
from SALES.CUSTOMER, SALES.SALESORDER_HEADER
where SALES.CUSTOMER.CUSTOMERID = SALES.SALESORDER_HEADER.CUSTOMERID
and SALES.CUSTOMER.PERSONID = '12';
```

3.8.2 位图连接索引删除语句

如果不需要位图连接索引可以使用删除语句删除。

删除（位图连接）索引语句格式：

```
DROP INDEX [<模式名>.]<索引名>;
```

参数

参数说明请参考本章 3.7 节。

例如：

```
DROP INDEX sales.SALES_CUSTOMER_NAME_IDX;
```

3.8.3 位图连接索引的更新语句

如果用户对该表的数据进行修改，影响到了索引列的数据，则需要更新位图索引，确保数据的一致性。位图索引是实时更新的，即增加、删除和更新操作成功后会及时地自动更新位图索引信息。

增加：在索引列上增加一条数据，能够通过位图索引查找得到该插入的数据；

删除：在基表中删除一条数据，不能够通过位图索引查找该记录；

更新：对基表中修改一条数据，通过位图索引查找，可以查找到修改后的数据。

3.8.4 位图连接索引回滚与并发

由于用户会对基表数据进行增删改查，而这些操作如果影响到二级索引，即需要对二级索引进行更新，此时如果执行回滚，那么二级索引也必须支持自动回滚操作，否则数据就不能保证一致性。

增加：在索引列上增加一条数据，然后执行回滚操作，通过位图索引查找，该记录不存在；

删除：在基表中删除一条数据，然后执行回滚操作，通过位图索引查找，该记录可以查到；

更新：在基表中更新一条数据，然后执行回滚，通过位图索引查找，查找得到数据是更新前的数据；

3.9 管理全文索引

3.9.1 全文索引定义语句

用户可以在指定的表的文本列上建立全文索引。

语法格式

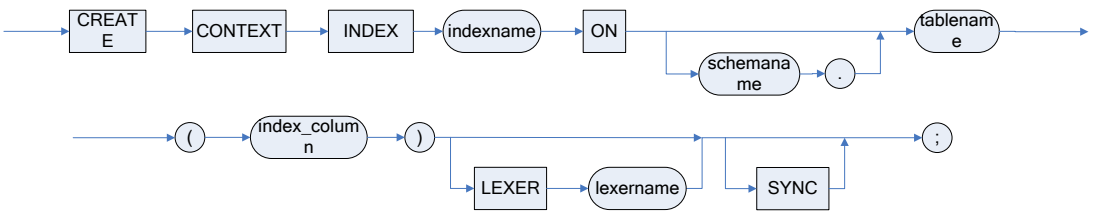
```
CREATE CONTEXT INDEX <索引名> ON  [<模式名>.] <表名> (<索引列定义>) [LEXER <分词参数>] [SYNC];
```

<索引列定义>请参考本章 3.15 节相关内容

参数

1. <索引名> 指明要创建的全文索引的名称；
2. <模式名> 指明要创建全文索引的基表属于哪个模式，缺省为当前模式；
3. <表名> 指明要创建全文索引的基表的名称；
4. <列名> 指明基表中要创建全文索引的列的名称；
5. <分词参数> 指明全文索引分词器的分词参数。

图例



语句功能

供 DBA 或该全文索引基表的拥有者且具有 CREATE CONTEXT INDEX 权限的用户，在指定的表的文本列上建立全文索引。

使用说明

1. 全文索引必须在一般用户表上定义，而不能在系统表、视图、临时表、列存储表、

分区表、LIST表和外部表上定义，同时，表必须未自定义聚集主键；

2. 一条全文索引作用于表的一个文本列，不允许为组合列和计算列；
3. 同一列只允许创建一个全文索引；
4. <列名>为文本列，类型可为CHAR、CHARACTER、VARCHAR、LONGVARCHAR、TEXT或CLOB；
5. TEXT、CLOB类型的列可存储二进制字符流数据。如果用于存储DM全文检索模块能识别的格式简单的文本文件(如.txt，html等)，则可为其建立全文索引；
6. 全文索引支持简体中文和英文；
7. 分词参数有4种：CHINESE_LEXER，中文最少分词；CHINESE_VGRAM_LEXER，中文最多分词；ENGLISH_LEXER，英文分词；DEFAULT_LEXER，中英文最少分词，也是默认分词。中文分词可以划分英文，但是指定英文分词不可以划分中文；
8. 创建全文索引时，如果设置SYNC属性，会同步填充索引信息，如果没有设置该属性，则需要用全文索引修改语句填充索引信息，之后才能进行全文检索；SYNC属性只能保证创建全文索引时和基表数据同步，如果之后表数据发生改变，仍需要填充索引信息；在指定表的指定列上建立全文索引完成后，全文索引信息会保存在CTISYS模式下的SYSCONTEXTINDEXES系统表中；
9. 不支持快速装载建有全文索引的表；
10. MPP环境下不支持全文索引的创建；
11. 不支持对建有位图索引的表进行表级备份恢复。

举例说明

例 用户 SYSDBA 需要在 PERSON 模式下的 ADDRESS 表的 ADDRES1 列上创建全文索引，可以用下面的语句：

```
CREATE CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS(ADDRESS1) LEXER CHINESE_LEXER;
```

3.9.2 全文索引修改语句

完全填充/增量填充全文索引。全文索引不能随数据的改变而自动更新，必须定期更新。若表数据变化而没有更新索引，可能引起查询结果不正确。

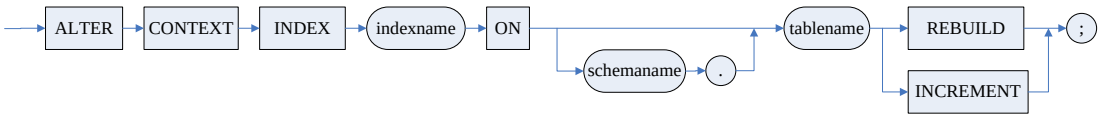
语法格式

```
ALTER CONTEXT INDEX <索引名> ON [<模式名>.] <表名> <REBUILD | INCREMENT>;
```

参数

1. <索引名> 指明被操作的全文索引的名称；
2. <模式名> 指明被操作的全文索引属于哪个模式，缺省为当前模式；
3. <表名> 指明被操作的基表的名称。

图例



语句功能

REBUILD 完全填充时，首先会将词表清空，再将基表中指定建立全文索引的列逐个记录取出，根据分词算法获得结果词，即字/词出现的位置信息(记录＋记录中的位置)，保存在词表中。INCREMENT 增量填充只是将建立全文索引后的基表数据发生变化的记录执行分词并保存词结果。

使用说明

1. 创建 CONTEXT INDEX 后，需要调用此语句，然后才能进行全文检索。这时该修改语句起到填充全文索引信息的作用；
2. 当该列数据大量更新后，为了对更新后的数据进行检索，需要再次调用该语句 REBUILD 重建全文索引信息，以确保查询的正确性；
3. DM 服务器启动时，不会自动加载词库，而是在第一次执行全文索引修改时加载，之后直到服务器停止才释放；
4. 在完全更新全文索引后，如果表的数据再次发生变化，就需要增量式更新全文索引，之后全文检索才可以获得正确的结果。

举例说明

例 用户 SYSDBA 需要在 PERSON 模式下的 ADDRESS 表的 ADDRES1 列上完全填充全文索引，可以用下面的语句：

```
ALTER CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS REBUILD;
```

3.9.3 全文索引删除语句

删除全文索引。

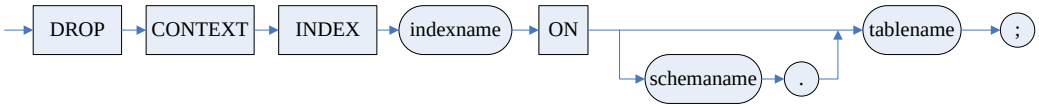
语法格式

```
DROP CONTEXT INDEX <索引名> ON [<模式名>.] <表名>;
```

参数

1. <索引名> 指明被操作的全文索引的名称；
2. <模式名> 指明被操作的全文索引属于哪个模式，缺省为当前模式；
3. <表名> 指明被操作的基表的名称。

图例



语句功能

供具有全文索引删除权限的用户或该全文索引所属基表的拥有者删除全文索引，包括删除数据字典中的相应信息和全文索引内容。

使用说明

除了该语句可删除全文索引外，当数据库模式发生如下改变时，系统将自动调用全文索引删除模块：

1. 删除表时，删除表上的全文索引；
2. 删除建立了全文索引的列时，删除列上的全文索引；
3. 修改建立了全文索引的列时，删除列上的全文索引。

举例说明

例 用户 SYSDBA 需要删除在 PERSON 模式下 ADDRESS 表的全文索引，可以用下面的语句：

```
DROP CONTEXT INDEX INDEX0001 ON PERSON.ADDRESS;
```

3.10 管理序列

3.10.1 序列定义语句

序列是一个数据库实体，通过它多个用户可以产生唯一整数值，可以用序列来自动地生成主关键字值。

语法格式

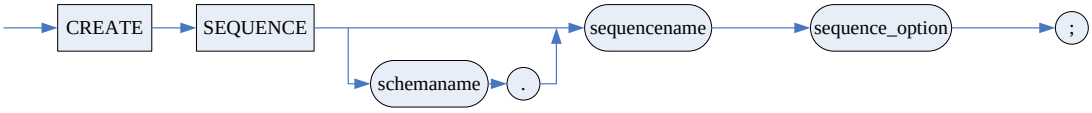
```
CREATE SEQUENCE [ <模式名>.] <序列名>  [ <序列选项列表>];
<序列选项列表> ::=  <序列选项>{<序列选项>}
<序列选项> ::=
INCREMENT BY <增量值>|
START WITH <初值> |
MAXVALUE <最大值> | NOMAXVALUE |
MINVALUE <最小值> | NOMINVALUE|
CYCLE | NOCYCLE|
CACHE <缓存值>| NOCACHE|
ORDER | NOORDER
```

参数

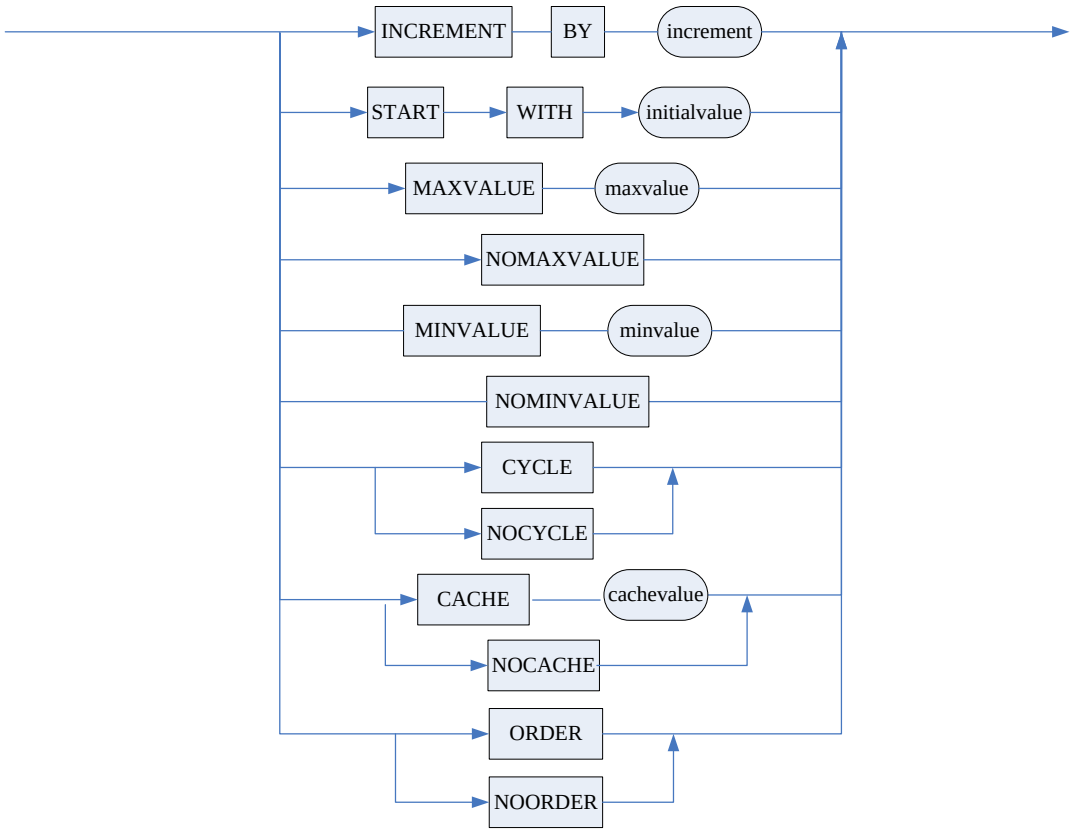
1. <模式名>　指明被创建的序列属于哪个模式，缺省为当前模式；
2. <序列名>　指明被创建的序列的名称，序列名称最大长度128字节；
3. <增量值>　指定序列数之间的间隔，这个值可以是任意的DM正整数或负整数，但不能为0。如果此值为负，序列是下降的，如果此值为正，序列是上升的。如果忽略INCREMENT BY子句，则间隔缺省为1；
4. <初值>　指定被生成的第一个序列数，可以用这个选项来从比最小值大的一个值开始升序序列或比最大值小的一个值开始降序序列。对于升序序列，缺省值为序列的最小值，对于降序序列，缺省值为序列的最大值；
5. <最大值>　指定序列能生成的最大值，如果忽略MAXVALUE子句，则降序序列的最大值缺省为-1，升序序列的最大值为9223372036854775806(0x7FFFFFFFFFFFFFE)。非循环序列在到达最大值之后，将不能继续生成序列数；
6. <最小值>　指定序列能生成的最小值，如果忽略MINVALUE子句，则升序序列的最小值缺省为1，降序序列的最小值为-9223372036854775808(0x8000000000000000)。循环序列在到达最小值之后，将不能继续生成序列数；
7. CYCLE　该关键字指定序列为循环序列：当序列的值达到最大值/最小值时，序列将从最小值/最大值计数；
8. NOCYCLE　该关键字指定序列为非循环序列：当序列的值达到最大值/最小值时，序列将不再产生新值；
9. CACHE　该关键字表示序列的值是预先分配，并保持在内存中，以便更快地访问；<缓存值>指定预先分配的值的个数，最小值为2；最大值为50000；且缓存值不能大于(<最大值> - <最小值>)/<增量值>；
10. NOCACHE　该关键字表示序列的值是不预先分配；
11. ORDER　该关键字表示以保证请求顺序生成序列号；

12. NOORDER 该关键字表示不保证请求顺序生成序列号。

图例



sequence_option



语句功能

创建一个序列生成器。只有 DBA 或该序列的拥有者且具有 CREATE SEQUENCE 权限的用户才能创建序列。

使用说明

1. 一旦序列生成，就可以在 SQL 语句中用以下伪列来存取序列的值；
   1） CURRVAL 返回当前的序列值；
   2） NEXTVAL 如果为升序序列，序列值增加并返回增加前的值；如果为降序序列，序列值减少并返回减少前的值。
2. 缺省序列：如果在序列中什么也没有指出则缺省生成序列，一个从 1 开始增量为 1 且无限上升(最大值为 9223372036854775806)的升序序列；仅指出 INCREMENT BY -1，将创建一个从-1 开始且无限下降(最小值为-9223372036854775808)的降序序列。

举例说明

例 创建序列 SEQ_QUANTITY，将序列的前两个值插入表 PRODUCTION.PRODUCT_INVENTORY 中。

```
(1)创建序列 SEQ_QUANTITY
CREATE  SEQUENCE  SEQ_QUANTITY  INCREMENT  BY 10;
(2)将序列的第一个值插入表 PRODUCT_INVENTORY 中
```

```
INSERT INTO PRODUCTION.PRODUCT_INVENTORY  VALUES(1,1, SEQ_QUANTITY.NEXTVAL);
SELECT  *  FROM  PRODUCTION.PRODUCT_INVENTORY;
```

查询结果为：表 PRODUCT_INVENTORY 增加一行，列 QUANTITY 的值为 1

(3)将序列的第二个值插入表 PRODUCT_INVENTORY 中

```
INSERT INTO PRODUCTION.PRODUCT_INVENTORY  VALUES(1,1, SEQ_QUANTITY.NEXTVAL);
SELECT  *  FROM  PRODUCTION.PRODUCT_INVENTORY;
```

查询结果为：表 PRODUCT_INVENTORY 增加两行，列 QUANTITY 的值分别为 1，11

3.10.2 序列删除语句

DM 系统允许用户在建立序列后还可随时删除序列。

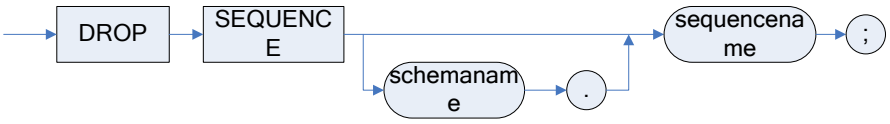
语法格式

```
DROP SEQUENCE [ <模式名>.]<序列名>;
```

参数

1. <模式名> 指明被删除序列所属的模式，缺省为当前模式；
2. <序列名> 指明被删除序列的名称。

图例



语句功能

供具有 DBA 角色的用户或该序列的拥有者从数据库中删除序列生成器。

使用说明

一种重新启动序列生成器的方法就是删除它然后再重新创建，例如有一序列生成器当前值为 150，而且用户想要从值 27 开始重新启动此序列生成器，他可以：

1) 删除此序列生成器；
2) 重新以相同的名字创建序列生成器，START WITH 选项值为 27。

举例说明

例 用户 SYSDBA 需要删除序列 SEQ_QUANTITY，可以用下面的语句：

```
DROP SEQUENCE SEQ_QUANTITY;
```

3.11 管理 SQL 域

为了支持 SQL 标准中的域对象定义与使用，DM 支持 DOMAIN 的创建、删除以及授权 DDL 语句，并支持在表定义中使用 DOMAIN。

域（DOMAIN）是一个可允许值的集合。域在模式中定义，并由<域名>标识。域是用来约束由各种操作存储于基表中某列的有效值集。域定义说明一种数据类型，它也能进一步说明约束域的有效值的<域约束>，还可说明一个<缺省子句>，该子句规定没有显式指定值时所要用的值或列的缺省值。

3.11.1 创建 DOMAIN

CREATE DOMAIN 创建一个新的数据域。 定义域的用户成为其所有者。DOMAIN 为模式类型对象，其名称在模式内唯一。

语法格式

```
<domain definition>::=CREATE DOMAIN <domain name> [ AS ] <数据类型> [ <default clause> ]
[ <domain constraint>... ]
<domain constraint>::=[<constraint name definition>]<check constraint definition>
<constraint name definition>::=CONSTRAINT <约束名>
<check constraint definition>::= CHECK (expression)
```

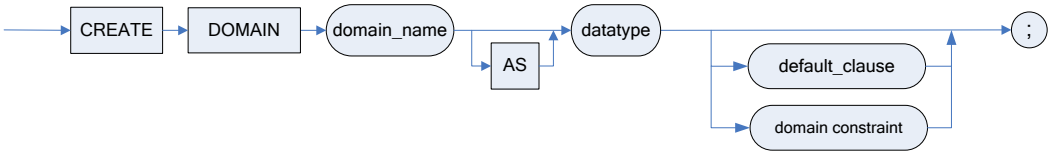
语句功能

供具有 CREATE DOMAIN 和 CREATE ANY DOMAIN 系统权限的用户创建 DOMAIN。

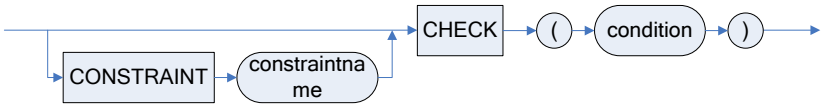
参数

1. <domain name> 要创建的域名字（可以有模式前缀）。如果在 CREATE SCHEMA 语句中定义 DOMAIN，则<domain name>中的模式前缀（如果有）必须与创建的模式名一致。
2. <data type> 域的数据类型。仅支持定义标准的 SQL 数据类型。
3. <default clause> DEFAULT 子句为域数据类型的字段声明一个缺省值。该值是任何不含变量的表达式（但不允许子查询）。 缺省表达式的数据类型必需匹配域的数据类型。如果没有声明缺省值， 那么缺省值就是空值。缺省表达式将用在任何为该字段声明数值的插入操作。 如果为特定的字段声明了缺省值，那么它覆盖任何和该域相关联的缺省值。
4. <constraint name definition> 一个约束的可选名称。如果没有名称，系统生成一个名字。
5. <check constraint definition> CHECK 子句声明完整性约束或者是测试，域的数值必须满足这些要求。 每个约束必须是一个生成一个布尔结果的表达式。它应该使用名字 VALUE 来引用被测试的数值。CHECK 表达式不能包含子查询，也不能引用除 VALUE 之外的变量。

图例



domain constraint



举例说明

```
CREATE DOMAIN DA INT CHECK (VALUE < 100);
```

3.11.2 使用 DOMAIN

在表定义语句中，支持为表列声明使用域。如果列声明的类型定义使用域引用，则此列定义直接继承域中的数据类型、缺省值以及 CHECK 约束。如果列定义使用域，然后又自己定义了缺省值，则最终使用自己定义的缺省值。

用户可以使用自己的域。如果要使用其它用户的域，则必须被授予了该域的 USAGE 权限。DBA 角色默认拥有此权限。

例 在 T 表中使用第 1 节中创建的域 DA。

```
CREATE TABLE T(ID DA);
```

列定义虽然使用了域后，其 SYSCOLUMNS 系统表中类型相关字段记录域定义的数据类型。也就是说，从 SYSCOLUMNS 系统表中不会表现出对域的引用。

使用说明

使用某个域的用户必须具有该域的 USAGE DOMAIN 或 USAGE ANY DOMAIN 权限。

3.11.3 删除 DOMAIN

语法格式

```
<drop domain statement> ::= DROP DOMAIN <domain name> <drop behavior>
<drop behavior> ::= RESTRICT | CASCADE
```

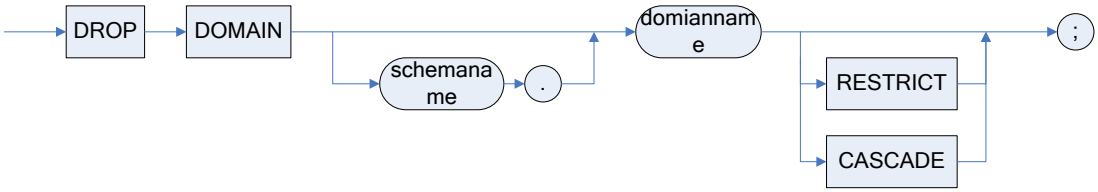
语句功能

删除一个用户定义的域。用户可以删除自己拥有的域，具有 DROP ANY TABLE 系统权限的用户则可以删除任意模式下的域。

参数说明

RESTRICT 表示仅当 DOMAIN 未被表列使用时才可以被删除；CASCADE 表示级联删除。

图例



举例说明

```
DROP DOMAIN DA CASCADE;
```

3.12 约束的启用与禁用

启用表上的约束。

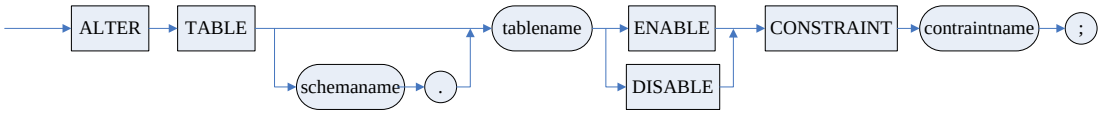
语法格式

```
ALTER TABLE  [<模式名>.]<表名>  ENABLE|DISABLE CONSTRAINT <约束名>;
```

参数

1. <模式名> 指明要启用约束的基表所属的模式，缺省为当前模式；
2. <表名> 指明要启用约束的基表的名称。
3. <约束名> 指明要启用约束的基表某列的约束名称。

图例



语句功能

供具有 DBA 权限的用户或该表的拥有者启用或禁用表中的约束。

使用说明

使用者应拥有 DBA 权限或者是该表的拥有者。

举例说明

例 创建、启用、禁用、删除 PERSON 表约束 unq。

```
ALTER TABLE PERSON.PERSON ADD CONSTRAINT unq UNIQUE(PHONE);
ALTER TABLE PERSON.PERSON ENABLE CONSTRAINT unq;
ALTER TABLE PERSON.PERSON DISABLE CONSTRAINT unq;
ALTER TABLE PERSON.PERSON DROP CONSTRAINT unq;
```

3.13 设置当前会话时区信息

设置当前会话时区信息

语法格式

```
SET TIME ZONE <时区>;
<时区>::= <LOCAL>|<'[+|-]整数'>|INTERVAL'[+|-]整数'<间隔类型>
```

参数

<时区> 指明要设置的时区信息；

图例

语句功能

设置当前会话时区信息。

使用说明

只能当前会话有效。

举例说明

例 设置当前会话时区为'+9:00'。

```
SET TIME ZONE   '+9:00';
```

例 设置当前会话时区为服务器所在地时区。

```
SET TIME ZONE LOCAL;
```

3.14 注释语句

可以通过注释语句来创建或修改表、视图或它们的列的注释信息。表和视图上的注释信息可以通过查询字典表 SYSTABLECOMMENTS 进行查看，列的注释信息可以通过查询字典表 SYSCOLUMNSCOMMENTS 进行查看。

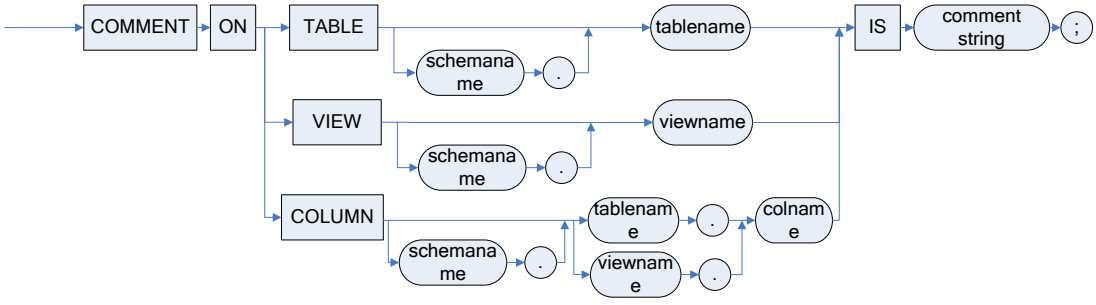
语法格式

```
COMMENT ON TABLE|VIEW|COLUMN <对象名称> IS <注释字符串>
<对象名称> ::= <表名定义> | <视图名定义> | <列名定义>
<表名定义> ::= [<模式名>.]<表名>
<视图名定义> ::= [<模式名>.]<视图名>
<列名定义> ::= [<模式名>.]<表名|视图名>.<列名>
```

参数

<注释字符串> 指明要设置的注释信息；

图例



使用说明

1. 注释字符串最大长度为1024；
2. 在已有注释的对象上再次执行此语句将直接覆盖之前的注释。

举例说明

例 为表 PERSON 创建注释信息。

```
COMMENT ON TABLE PERSON IS 'PERSON IS A SIMPLE TABLE';
```

例 为表 PERSON 的列 NAME 创建注释信息。

```
COMMENT ON COLUMN PERSON.NAME IS 'SYSDBA.PERSON.NAME';
```

第 4 章 数据查询语句

数据查询是数据库的核心操作，DM_SQL 语言提供了功能丰富的查询方式，满足实际应用需求。几乎所有的数据库操作均涉及到查询，因此熟练掌握查询语句的使用是数据库从业人员必须掌握的技能。

在 DM_SQL 语言中，有的语法支持包含查询语句，如视图定义语句、游标定义语句等。为了区别，我们将这类出现在其它定义语句中的查询语句称查询说明。

每种查询都有适用的场景，使用得当会大大提高查询效率。为方便用户的使用，本章对 DM_SQL 语言支持的查询方式进行讲解，测例中所用基表及各基表中预先装入的数据参见第 2 章，各例的建表者均为用户 SYSDBA。

查询语句的语法如下：

```
<查询表达式>;
<查询表达式>::= <不带 INTO 查询表达式> | <带 INTO 查询表达式>
<不带 INTO 查询表达式>::=<子查询表达式>[ <ORDER BY 子句>][ <FOR UPDATE 子句>]
<子查询表达式>::=<查询表达式> | <复合查询表达式> | [( ] <子查询表达式> [ )]
<查询表达式>::=
SELECT [ALL | DISTINCT] [TOP N1 [, N2 ]]
<选择列表>[ FROM <表引用> {,<表引用>}
 [<WHERE 子句>]
 [<CONNECT BY 子句>]
 [<GROUP BY 子句>]
 [<HAVING 子句>]]
<复合查询表达式>::=<子查询表达式> UNION [ALL | DISTINCT]| EXCEPT | MINUS | INTERSECT <子查询表达式>
<带 INTO 查询表达式>::=< select_with_into1>| < select_with_into 2>
< select_with_into 1>::=SELECT [ALL | DISTINCT] [TOP N1[, N2 ]]
<选择列表> INTO <存贮列表>[ FROM <表引用> [{,<表引用>}] [<WHERE 子句>]
 [<GROUP BY 子句>] [<HAVING 子句>] ][ <ORDER BY 子句>][<限制条件>]
<存贮列表> ::= <存贮对象>{, <存贮对象>}
<存贮对象>>::=<变量名> | <参数> | <伪列>
<伪列>::=:OLD|NEW|<别名>.列名
<限制条件>::=LIMIT <大小> [ OFFSET <大小>]
<选择列表> ::= [[<模式名>.]<基表名> | <视图名> .] * | <值表达式> [[AS] <列别名>]
{,[[<模式名>.]<基表名> | <视图名>.] * | <值表达式> [[AS] <列别名>]}
< select_with_into 2>::= SELECT <存贮对象>=<值表达式>{,<存贮对象>=<值表达式>}FROM <表引用>{,<表引用>} [<WHERE 子句>]
<表引用> ::=<普通表> | <连接表>
<普通表>::=<普通表 1> | <普通表 2> | <普通表 3>
<普通表 1>::= <对象名> [AS] [别名]
<普通表 2>::= (<不带 INTO 查询表达式>) [[AS] <表别名> [<新生列>]]
<普通表 3>::= [<模式名>.]<基表名> | <视图名> (<选择列>) [[AS] <表别名> [<派生列表>]]
<对象名>::=<本地对象> | <索引> | <分区表>
```

```
<本地对象>::=[<模式名>.]<基表名> | <视图名>
<索引>::=[<模式名>.]<基表名> INDEX <索引名>
<分区表>::=[<模式名>.]<基表名> PARTITION (<分区表名>)
<选择列>::= <列名> [ { , <列名> }]
<派生列表> ::=( <列名> [ { , <列名> }])
<连接表>::=<交叉连接>|<限定连接>|<左括号><连接表><右括号>
<交叉连接>::=<表引用> CROSS JOIN <表引用> | (<连接表>)
<限定连接>::=<限定连接 1> | <限定连接 2>
<限定连接 1>::=<表引用> [NATURAL] [<连接类型>] JOIN <表引用> | (<连接表>)
<限定连接 2>::=<表引用> [<连接类型>] [HASH | MERG] JOIN <表引用> <连接条件>
<连接类型>::=INNER|[<外连接类型>[OUTER]]
<外连接类型>::=LEFT|RIGHT|FULL
<连接条件>::=<条件匹配> | <列匹配>
<条件匹配>::=ON <搜索条件>
<列匹配>::=USING (<列名>{, <列名>})
<WHERE 子句> ::= WHERE <搜索条件>
<CONNECT BY 子句> ::= CONNECT BY [NOCYCLE] <连接条件> [ START WITH <起始条件> ] |
                      START WITH <起始条件> CONNECT BY [NOCYCLE] <连接条件>
<连接条件>::= <表达式> = PRIOR <表达式> | PRIOR <表达式> = <表达式>
<GROUP BY 子句> ::= GROUP BY <分组项> | <ROLLUP 项> | <CUBE 项> | <GROUPING SETS 项>
<HAVING 子句> ::= HAVING <搜索条件>
<ORDER BY 子句> ::= ORDER BY
<无符号整数> | <列说明> | <值表达式> [ASC | DESC] [nulls first|last]
{,<无符号整数> | <列说明> | <值表达式> [ASC | DESC] [nulls first|last]}
<搜索条件>::=<条件判断表达式>
```

参数

1．ALL　　返回所有被选择的行，包括所有重复的拷贝，缺省值为 ALL；

2．DISTINCT　从被选择出的具有重复行的每一组中仅返回一个这些行的拷贝。对于集合算符：UNION，缺省值为 DISTINCT；对于 EXCEPT/MINUS 和 INTERSCET：操作的两个表中数据类型和个数要完全一致。其中，EXCEPT 和 MINUS 集合算符功能完全一样，返回两个集合的差集；INTERSCET 返回两个集合的交集（去除重复记录）。

3．<模式名>　被选择的表和视图所属的模式，缺省为当前模式；

4．<基表名>　被选择数据的基表的名称；

5．<视图名>　被选择数据的视图的名称；

6．*　　指定对象的所有的列；

7．<值表达式>　可以为一个<列引用>、<集函数>、<函数>、<标量子查询>或<计算表达式>等等；

8．<列别名>　为列表达式提供不同的名称，使之成为列的标题，列别名不会影响实际的名称，别名在该查询中被引用；

9．<相关名>　给表、视图提供不同的名字，经常用于求子查询和相关查询的目的；

10．<列名>　指明列的名称；

11．<WHERE 子句>　限制被查询的行必须满足条件，如果忽略该子句，DM 从在 FROM 子句中的表、视图中选取所有的行；

12．<HAVING 子句>　限制所选择的行组所必须满足的条件，缺省为恒真，即对所有

的组都满足该条件；

13．<无符号整数>　　指明了要排序的<值表达式>在 SELECT 后的序列号；

14．<列说明>　　排序列的名称；

15．ASC　　指明为升序排列，缺省为升序；

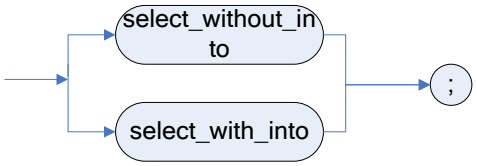
16．DESC　　指明为降序排列；

17. NULLS FIRST　指定排序列的 NULL 值放在最前面，不受 ASC 和 DESC 的影响，缺省的是 NULLS FIRST；

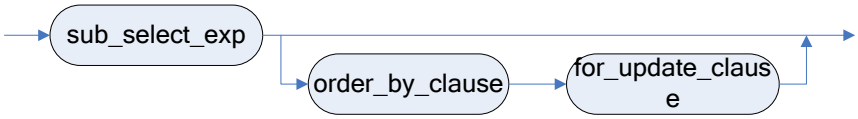
18. NULLS LAST　指定排序列的 NULL 值放在最后面，不受 ASC 和 DESC 的影响。

图例

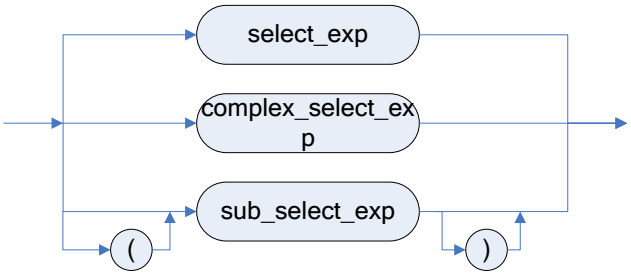
select_without_into



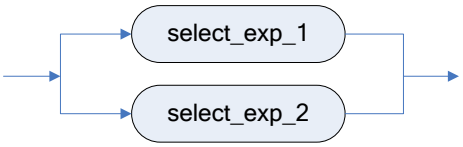
sub_select_exp



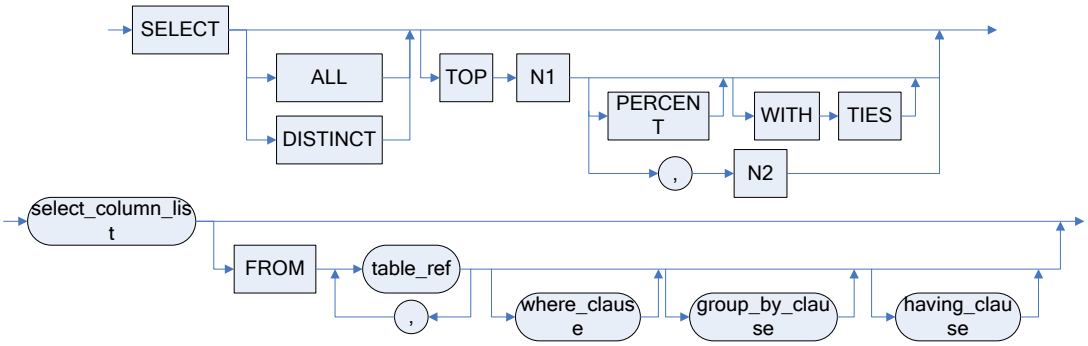
select_exp



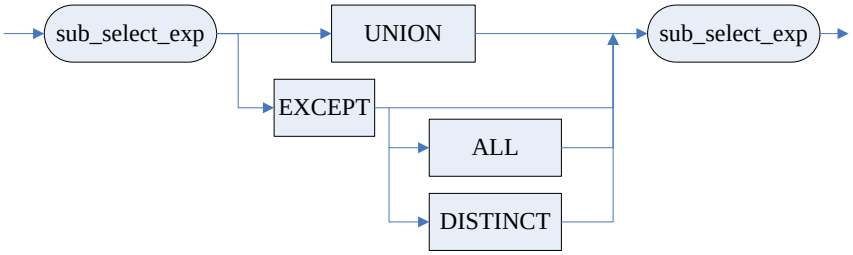
select_exp_1



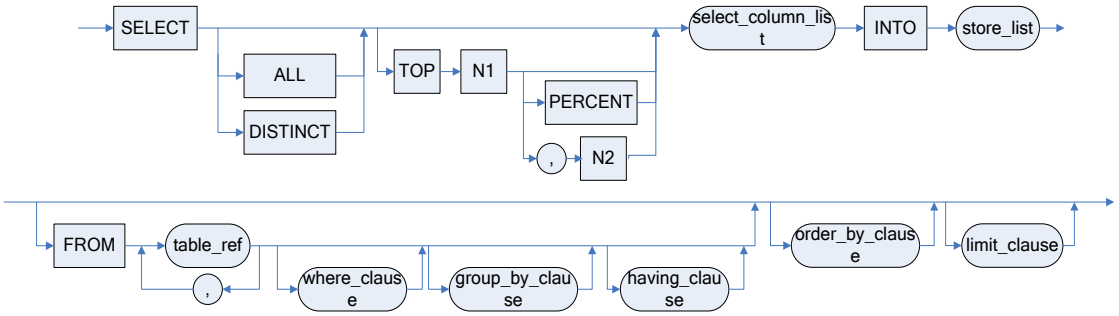
complex_select_exp



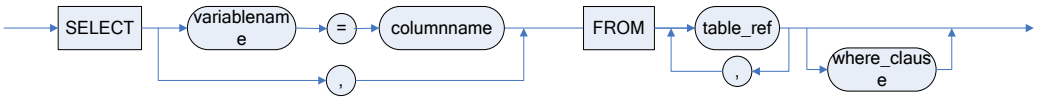
sub_select_exp



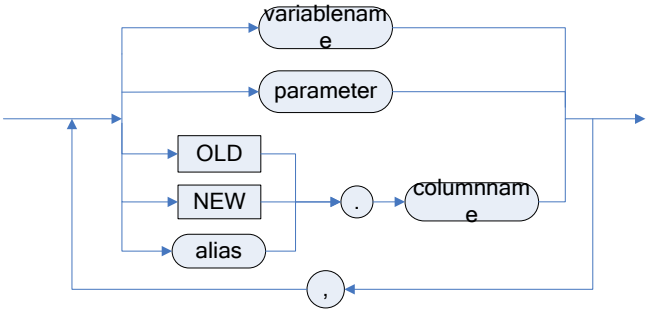
select_with_into1



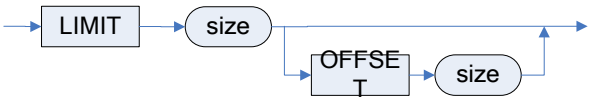
select_with_into2



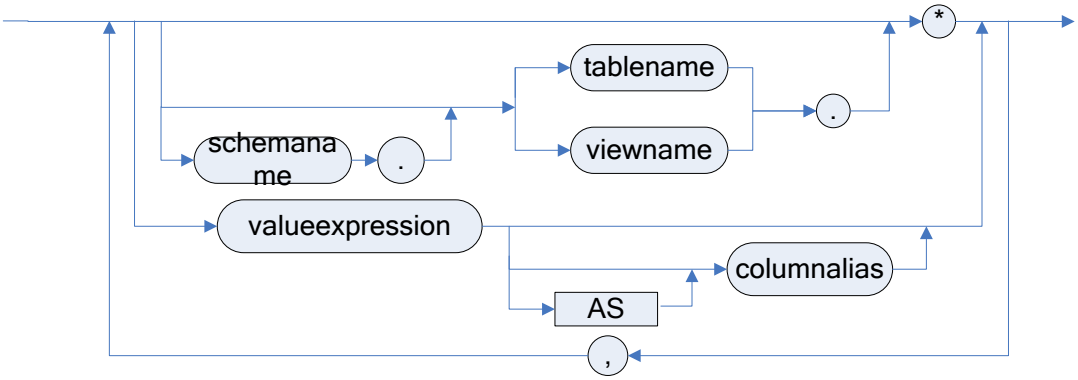
store_list



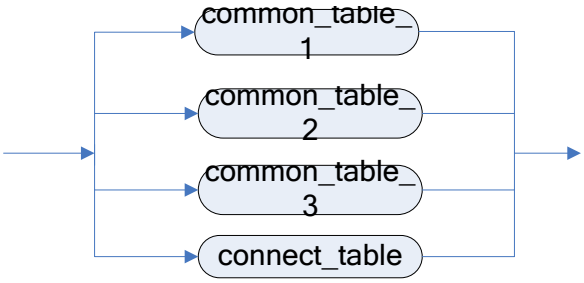
limit_clause



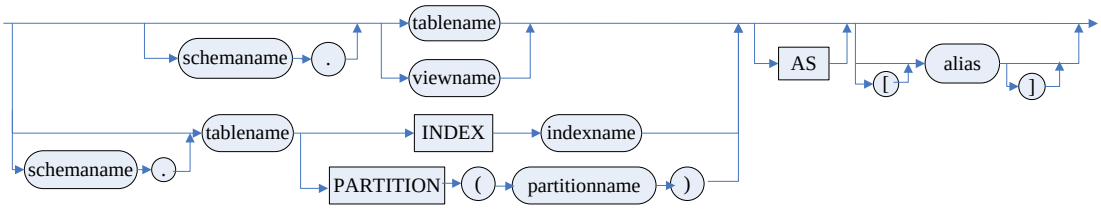
select_column_list



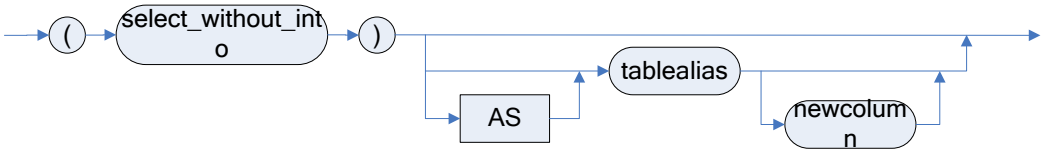
table_ref



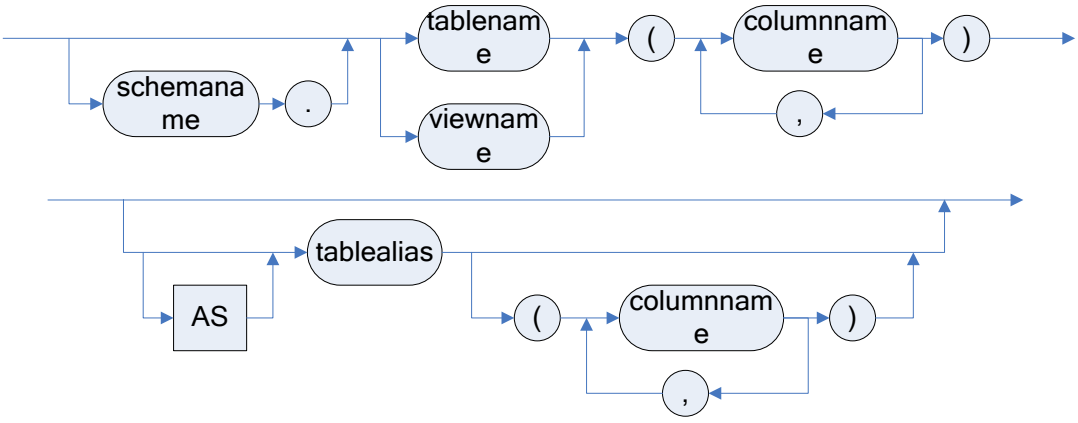
common_table_1



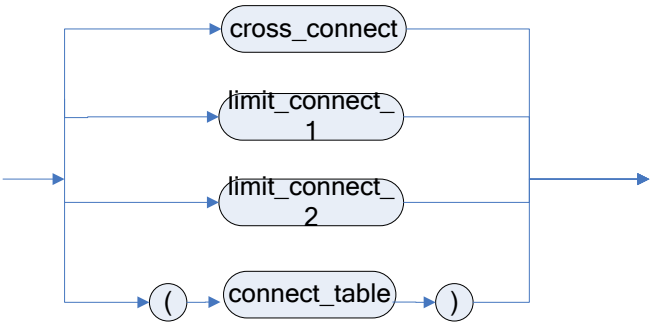
ommon_table_2



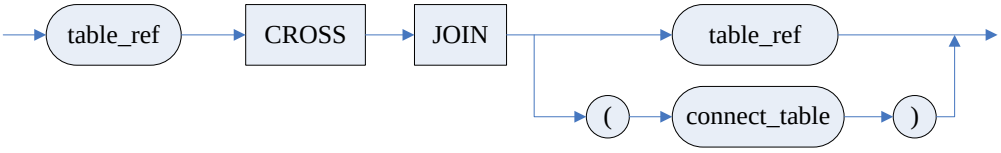
common_table_3



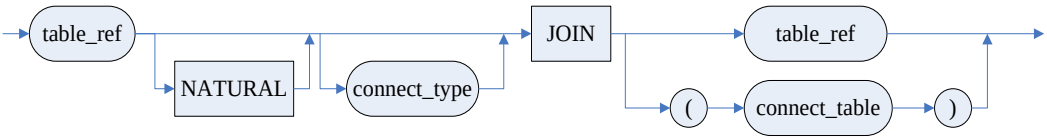
connect_table



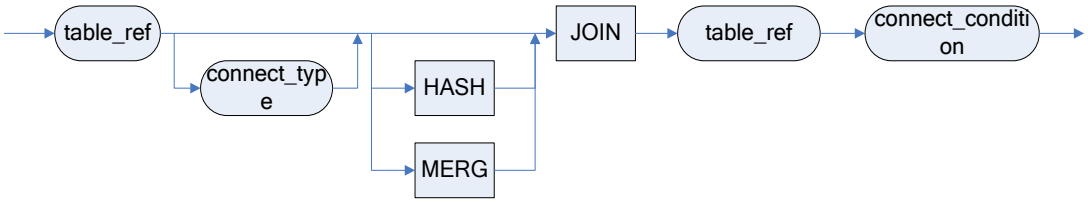
cross_connect



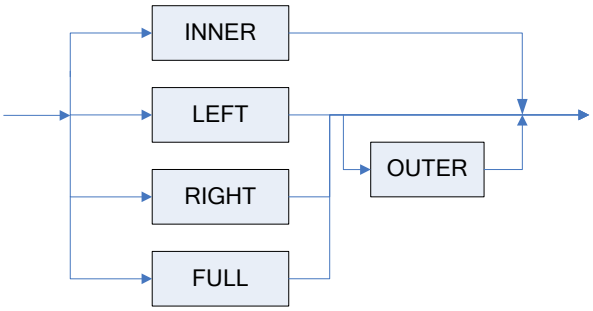
limit_connect_1



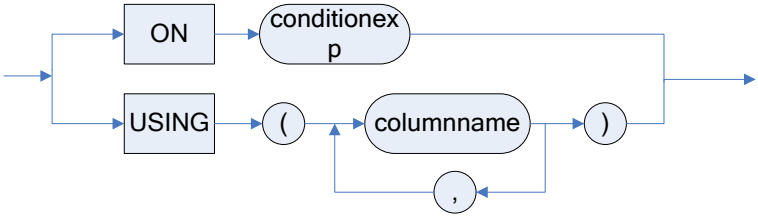
limit_connect_2



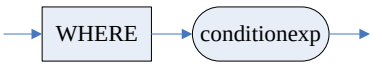
connect_type



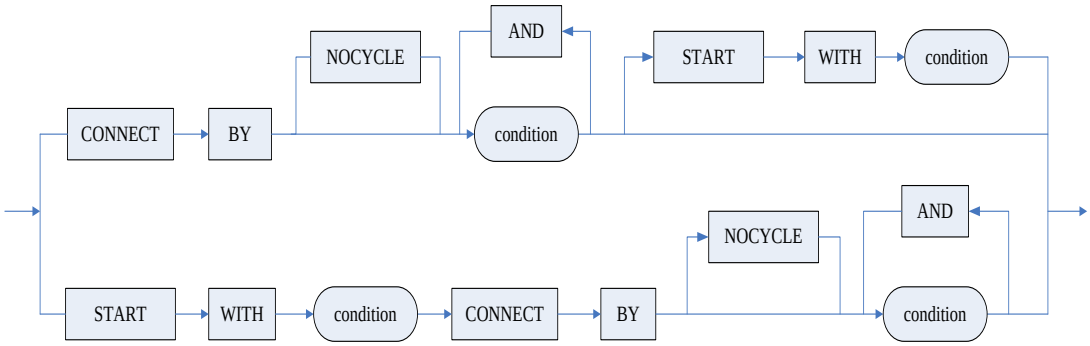
connect_condition



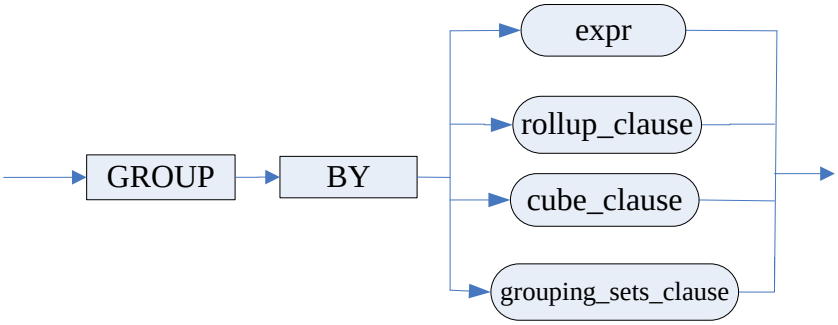
where_clause



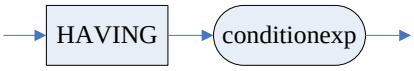
connect_by_clause



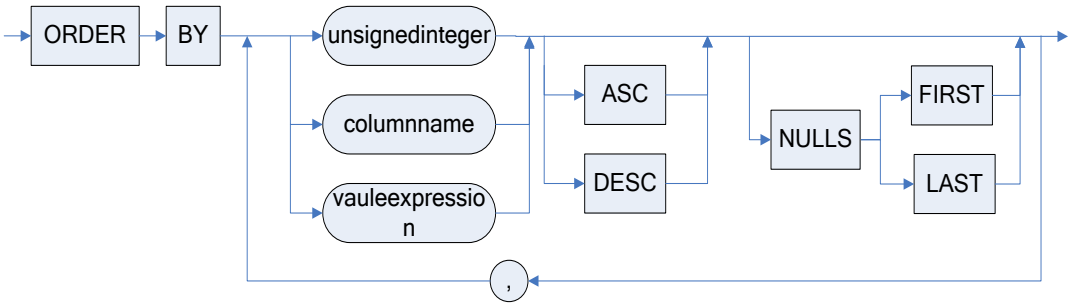
group_by_clause



having_clause



order_by_clause



使用说明

1. <选择列表>中最多可包含 1024 个查询项，且查询记录的长度限制不能超过块长的一半；
2. <FROM 子句>中最多可引用 50 张表。

4.1 单表查询

本节着重于较简单的查询语句——单表查询，即 SELECT 语句仅从一个表/视图中检索数据。以下是一个单表查询语句的语法：

```
SELECT <选择列表>
FROM [<模式名>.]<基表名> | <视图名> <相关名>
[<WHERE 子句>]
[<CONNECT BY 子句>]
```

```
[<GROUP BY 子句>]
[<HAVING 子句>]
[ORDER BY 子句];
```

可选项 WHERE 子句用于设置对于行的检索条件。不在规定范围内的任何行都从结果表中去除。GROUP BY 子句逻辑地将由 WHERE 子句返回的临时结果重新编组。结果是行的集合，一组内一个成组列的所有值都是相同的。HAVING 子句用于为组设置检索条件。本节将详细介绍 WHERE 子句，后两种子句将在本章第 4.5 节中详细介绍。

4.1.1 简单查询

例 查询所有图书的名字、作者及当前销售价格，并消去重复。

```
SELECT  DISTINCT  NAME, AUTHOR, NOWPRICE  FROM  PRODUCTION .PRODUCT;
```

其中，DISTINCT 保证重复的行将从结果中去除。若允许有重复的元组，改用 ALL 来替换 DISTINCT，或直接去掉 DISTINCT 即可。

查询结果如表 4.1.1 所示。(注：除带 Order By 的查询外，本书所示查询结果中各元组的顺序与实际输出结果中的元组顺序不一定一致。)

表 4.1.1 查询结果

NAME	AUTHOR	NOWPRICE
长征	王树增	37.7000
工作中无小事	陈满麒	11.4000
红楼梦	曹雪芹,高鹗	15.2000
老人与海	海明威	6.1000
鲁迅文集(小说、散文、杂文)全两册	鲁迅	20.0000
射雕英雄传(全四册)	金庸	21.7000
数据结构(C 语言版)(附光盘)	严蔚敏，吴伟民	25.5000
水浒传	施耐庵，罗贯中	14.3000
突破英文基础词汇	刘毅	11.1000
噼里啪啦丛书(全 7 册)	(日)佐佐木洋子	42.0000

有时，用户需要查出各个列的数据，且各列的显示顺序也与基表中列的顺序相同时，为了方便用户提高工作效率，SQL 语言允许用户将 SELECT 后的<值表达式>省略为*。

```
SELECT  *  FROM  PERSON.PERSON;
```

等价于：

```
SELECT  PERSONID, NAME, SEX, EMAIL, PHONE  FROM  PERSON.PERSON;
```

其查询结果是模式 PERSON 中基表 PERSON 的一份拷贝，结果从略。

例 1）查询 tt 表中有的，kk 表中没有的数据；2）查询 tt 表和 kk 表都有的数据。

```
CREATE TABLE TT(A INT);
INSERT INTO TT VALUES(5);
INSERT INTO TT VALUES(6);
INSERT INTO TT VALUES(7);

CREATE TABLE KK(A INT);
INSERT INTO KK VALUES(5);
```

```
INSERT INTO KK VALUES(5);
INSERT INTO KK VALUES(6);
INSERT INTO KK VALUES(8);
```

1) 使用 MINUS 或 EXCEPT 查询 tt 表中有的，kk 表中没有的数据。

```
SELECT * FROM TT MINUS SELECT * FROM KK;
```

等价于

```
SELECT * FROM TT EXCEPT SELECT * FROM KK;
```

其查询结果是：

```
A
7
```

2）使用 INTERSECT 查询 TT 表中和 KK 表中都有的数据。

```
SELECT * FROM TT INTERSECT SELECT * FROM KK;
```

其查询结果是：

```
A
5
6
```

4.1.2 带条件查询

带条件查询是指在指定表中查询出满足条件的元组。该功能是在查询语句中使用 WHERE 子句实现的。WHERE 子句常用的查询条件由谓词和逻辑运算符组成。谓词指明了一个条件，该条件求解后，结果为一个布尔值：真、假或未知。

逻辑算符有：AND，OR，NOT。

谓词包括比较谓词(＝、＞、＜、＞＝、＜＝、＜＞)，BETWEEN 谓词、IN 谓词、LIKE 谓词、NULL 谓词、EXISTS 谓词。

1．使用比较谓词的查询

当使用比较谓词时，数值数据根据它们代数值的大小进行比较，字符串的比较则按序对同一顺序位置的字符逐一进行比较。若两字符串长度不同，短的一方应在其后增加空格，使两串长度相同后再作比较。

例 给出当前销售价格在 10～20 元之间的所有图书的名字、作者、出版社和当前价格。

```
SELECT  NAME, AUTHOR, PUBLISHER, NOWPRICE  FROM  PRODUCTION.PRODUCT
WHERE  NOWPRICE>=10  AND  NOWPRICE<=20;
```

查询结果如表 4.1.2 所示。

表 4.1.2

NAME	AUTHOR	PUBLISHER	NOWPRICE
红楼梦	曹雪芹,高鹗	中华书局	15.2000
水浒传	施耐庵，罗贯中	中华书局	14.3000
鲁迅文集(小说、散文、杂文)全两册	鲁迅		20.0000
工作中无小事	陈满麒	机械工业出版社	11.4000
突破英文基础词汇	刘毅	外语教学与研究出版社	11.1000

2．使用 BETWEEN 谓词的查询

例 给出当前销售价格在 10～20 元之间的所有图书的名字、作者、出版社和当前价格。

```
SELECT NAME, AUTHOR, PUBLISHER, NOWPRICE FROM PRODUCTION.PRODUCT
WHERE NOWPRICE BETWEEN 10 AND 20;
```

此例查询与上例完全等价，查询结果如上表所示。在 BETWEEN 谓词前面可以使用 NOT，以表示否定。

3．使用 IN 谓词的查询

谓词 IN 可用来查询某列值属于指定集合的元组。

例 查询出版社为中华书局或人民文学出版社出版的图书名称与作者信息。

```
SELECT NAME, AUTHOR FROM PRODUCTION.PRODUCT
WHERE PUBLISHER IN ('中华书局', '人民文学出版社');
```

查询结果如表 4.1.3 所示。

表 4.1.3

NAME	AUTHOR
红楼梦	曹雪芹，高鹗
水浒传	施耐庵，罗贯中
长征	王树增

在 IN 谓词前面也可用 NOT 表示否定。

4．使用 LIKE 谓词的查询

LIKE 谓词一般用来进行字符串的匹配。我们先用实例来说明 LIKE 谓词的使用方法。

例 查询第一通讯地址中第四个字开始为“关山”且以 202 结尾的地址。

```
SELECT ADDRESSID, ADDRESS1, CITY, POSTALCODE FROM PERSON.ADDRESS
WHERE ADDRESS1 LIKE '___关山%202';
```

查询结果如表 4.1.4 所示。

表 4.1.4

ADDRESSID	ADDRESS1	CITY	POSTALCODE
13	洪山区关山春晓 55-1-202	武汉市洪山区	430073
14	洪山区关山春晓 10-1-202	武汉市洪山区	430073
15	洪山区关山春晓 11-1-202	武汉市洪山区	430073

由上例可看出，LIKE 谓词的一般使用格式为：

```
<列名> LIKE <匹配字符串常数>
```

其中，<列名>必须是可以转化为字符类型的数据类型的列。对于一个给定的目标行，如果指定列值与由<匹配字符串常数>给出的内容一致，则谓词结果为真。<匹配字符串常数>中的字符可以是一个完整的字符串，也可以是百分号“%”和下划线“_”，“%”和“_”称通配符。“%”代表任意字符串(也可以是空串)；“_”代表任何一个字符。

因此，上例中的 SELECT 语句将从 ADDRESS 表中检索出第一通讯地址中第四个字开始为“关山”且以 202 结尾的地址情况。从该例我们可以看出 LIKE 谓词是非常有用的。使用 LIKE 谓词可以找到所需要的但又记不清楚的那样一些信息。这种查询称模糊查询或匹配查询。为了加深对 LIKE 谓词的理解，下面我们再举几例：

```
ADDRESS1 LIKE '%洪山%'
```

如果 ADDRESS1 的值含有字符“洪山”，则该谓词取真值。

```
POSTALCODE LIKE '43__7_'
```

如果 POSTALCODE 的值由六个字符组成且前两个字符为 43，第五个字符为 7，则该谓词取真值。

```
CITY  LIKE  '%汉阳_'
```

如果 CITY 的值中倒数第三和第二个字为汉阳，则该谓词取真值。

```
ADDRESS1  NOT  LIKE  '洪山%'
```

如果 ADDRESS1 的值的前两个字不是洪山，则该谓词取真值。

阅读以上的例子，读者可能就在想这样一个问题：如果<匹配字符串常数>中所含“%”和“_”不是作通配符，而只是作一般字符使用应如何表达呢？为解决这一问题，SQL 语句对 LIKE 谓词专门提供了对通配符“%”和“_”的转义说明，这时 LIKE 谓语使用格式为：

```
<列名>  LIKE  '<匹配字符串常数>'  [ESCAPE  <转义字符>]
```

其中，<转义字符>指定了一个字符，当该字符出现在<匹配字符串常数>中时，用以指明紧跟其后的“%”或“_”不是通配符而仅作一般字符使用。

例 查询第一通讯地址以 C1_501 结尾的地址，则 LIKE 谓词应为：

```
SELECT  ADDRESSID,  ADDRESS1,  CITY,  POSTALCODE  FROM  PERSON.ADDRESS
WHERE  ADDRESS1  LIKE  '%C1*_501'  ESCAPE '*';
```

在此例中，*被定义为转义字符，因而在<匹配字符串常数>中*号后的下划线不再作通配符，而是普通字符。

查询结果如表 4.1.5 所示。

表 4.1.5

ADDRESSID	ADDRESS1	CITY	POSTALCODE
16	洪山区光谷软件园 C1_501	武汉市洪山区	430073

为避免错误，转义字符一般不要选通配符“%”、“_”或在<匹配字符串常数>中已出现的字符。

5．使用.ROW 进行 LIKE 谓词的查询

LIKE 谓词除支持使用列的计算外，还支持通过 ROW 保留字对表或视图进行 LIKE 计算。该查询依次对表或视图中所有字符类型类型的列进行 LIKE 计算，只要有一列符合条件，则返回 TRUE。

其语法的一般格式为

```
<表名>.ROW  LIKE  <匹配字符串>  [ ESCAPE  <转义字符>]
```

例 查询评论中哪些与曹雪芹有关

```
SELECT * FROM PRODUCTION. PRODUCT_REVIEW WHERE  PRODUCT_REVIEW.ROW LIKE '%曹雪芹%';
```

该语句等价于

```
SELECT * FROM PRODUCTION. PRODUCT_REVIEW WHERE NAME LIKE '%曹雪芹%' OR EMAIL LIKE '%曹雪芹%' OR COMMENTS LIKE '%曹雪芹%';
```

6．使用 NULL 谓词的查询

空值是未知的值。当列的类型为数值类型时，NULL 并不表示 0；当列的类型为字符串类型时，NULL 也并不表示空串。因为 0 和空串也是确定值。NULL 只能是一种标识，表示它在当前行中的相应列值还未确定或未知，对它的查询也就不能使用比较谓词而须使用 NULL 谓词。

例 查询哪些人员的 EMAIL 地址为 NULL。

```
SELECT  NAME, SEX, PHONE  FROM  PERSON.PERSON
```

```
WHERE    EMAIL    IS NULL;
```

在 NULL 谓词前，可加 NOT 表示否定。

7. 组合逻辑

可以用逻辑算符(AND，OR，NOT)与各种谓词相组合生成较复杂的条件查询。

例 查询当前销售价格低于15元且折扣低于7或出版社为人民文学出版社的图书名称和作者。

```
SELECT    NAME, AUTHOR    FROM    PRODUCTION.PRODUCT
WHERE    NOWPRICE < 15 AND DISCOUNT < 7 OR PUBLISHER='人民文学出版社';
```

查询结果如表 4.1.6 所示。

表 4.1.6

NAME	AUTHOR
老人与海	海明威
长征	王树增
工作中无小事	陈满麒

4.1.3 集函数

为了进一步方便用户的使用，增强查询能力，SQL 语言提供了多种内部集函数。集函数又称库函数，当根据某一限制条件从表中导出一组行集时，使用集函数可对该行集作统计操作。集函数可分为四类：

1. COUNT(*)；
2. 相异集函数　AVG|MAX|MIN|SUM|COUNT(DISTINCT<列名>)；
3. 完全集函数　AVG|MAX|MIN| COUNT|SUM([ALL]<值表达式>)；
4. 方差集函数 var_pop、var_samp、variance、stddev_pop、stddev_samp、stddev；
5. 求区间范围内最大值集函数 AREA_MAX。
6. FIRST/LAST 集函数 AVG|MAX|MIN| COUNT|SUM([ALL] <值表达式>) KEEP (DENSE_RANK FIRST|LAST ORDER BY 子句)；ORDER BY 子句语法参考第 4.7 节。

相异集函数与完全集函数的区别是：相异集函数是对表中的列值消去重复后再作集函数运算，而完全集函数是对包含列名的值表达式作集函数运算且不消去重复。

在使用集函数时要注意以下几点：

1. 缺省情况下，集函数均为完全集函数；
2. 集函数中的自变量不允许是集函数，即不能嵌套使用；
3. 要注意 DISTINCT 的用法；
4. **AVG、SUM** 的参数必须为数值类型；**MAX、MIN** 的结果数据类型与参数类型保持一致；对于 **SUM** 函数，如果参数类型为 BYTE、BIT、SMALLINT 或 INTEGER，那么结果类型为 INTEGER，如果参数类型为 NUMERIC、DECIMAL、FLOAT、DOUBLE PRECISION 和 MONEY，那么结果类型为 DOUBLE PRECISION；COUNT 结果类型统一为 INTEGER；

   对于 **AVG** 函数，其参数类型与结果类型对应关系如表 4.1.7 所示：

表 4.1.7

参数类型	结果类型

tinyint	dec(3,1)
smallint	dec(5,1)
int	dec(10,1)
bigint	bigint
float	double
double	double
dec(x,y)	dec(x1,y1)

说明：对于 dec 类型，如果 y<6，则 y1=6，否则 y1=y;如果 x<19,则 x1=x+19，否则 x1=38。

5. 方差集函数中参数 expr 为<列名>或<值表达式>，具体用法如下：

1) **var_pop(expr)** 返回 expr 的总体方差。其计算公式为：

$$\frac{SUM(\exp r^2)-\frac{(SUM(\exp r))^2}{COUNT(\exp r)}}{COUNT(\exp r)}$$

2) **var_samp(expr)** 返回 expr 的样本方差，如果 expr 的行数为 1，则返回 NULL。其计算公式为：

$$\frac{SUM(\exp r^2)-\frac{(SUM(\exp r))^2}{COUNT(\exp r)}}{COUNT(\exp r)-1}$$

3) **variance(expr)** 返回 expr 的方差，如果 expr 的行数为 1，则返回为 0，行数大于 1 时，与 var_samp 函数的计算公式一致；

4) **stddev_pop(expr)** 返回 expr 的标准差，返回的结果为总体方差的算术平方根，即 var_pop 函数结果的算术平方根。公式如下：

$$\sqrt{\frac{SUM(\exp r^2)-\frac{(SUM(\exp r))^2}{COUNT(\exp r)}}{COUNT(\exp r)}}$$

5) **stddev_samp(expr)** 返回expr的标准差,返回的结果为样本方差的算术平方根，即 var_samp 函数结果的算术平方根，所以如果 expr 的行数为 1，stddev_samp 返回 NULL；6) stddev(expr) 与 stddev_samp 基本一致，差别在于，如果 expr 的行数为 1，stddev 返回 0，即 variance 函数结果的算术平方根。公式如下：

$$\sqrt{\frac{SUM(\exp r^2)-\frac{(SUM(\exp r))^2}{COUNT(\exp r)}}{COUNT(\exp r)-1}}$$

6. **AREA_MAX(EXP, LOW, HIGH)** 在区间[LOW, HIGH]的范围内取 EXP 的最大值。如果 EXP 不在该区间内，则返回 LOW 值。如果 LOW 或 HIGH 为 NULL，则返回 NULL。EXP 为<变量>、<常量>、<列名>或<值表达式>。参数 EXP 类型为 TINYINT、SMALLINT、INT、BIGINT、DEC、FLOAT、DOUBLE、DATE、TIME、DATETIME、BINARY、VARBINARY、INTERVAL YEAR TO MONTH、INTERVAL DAY TO HOUR、TIME WITH TIME ZONE、DATETIEM WITH TIME ZONE。LOW 和 HIGH 的数据类型和 EXP 的类型一致，如果不一致，则转换为 EXP 的类型，不能转换则

报错。AREA_MAX 集函数返回值定义如下：

表 4.1.8 没有 GROUP BY 的情况

EXP 集合	是否有在[LOW, HIGH]区间内的非空值	结果
空集	-	LOW
非空	否	LOW
非空	是	在[LOW,HIGH]区间的最大值

表 4.1.9 有 GROUP BY 的情况

分组前结果	在[LOW, HIGH]区间内是否非空值	结果
空集	--	整个结果为空集
非空集	是	在[LOW,HIGH]区间的最大值
非空集	否	LOW

7. **FIRST/LAST 集函数** 首先根据 sql 语句中的 group by 分组(如果没有指定分组则所有结果集为一组)，然后在组内进行排序。根据 FIRST/LAST 计算第一名（最小值）或者最后一名（最大值）的集函数值，排名按照奥林匹克排名法

下面按集函数的功能分别举例说明。

1．求最大值集函数 MAX 和求最小值集函数 MIN

例 查询折扣小于 7 的图书中现价最低的价格。

```
SELECT  MIN(NOWPRICE)  FROM  PRODUCTION.PRODUCT
WHERE  DISCOUNT < 7;
```

查询结果为：6.1000

需要说明的是：SELECT 后使用集函数 MAX 和 MIN 得到的是一个最大值和最小值，因而 SELECT 后不能再有列名出现，如果有只能出现在集函数中。如：

```
SELECT  NAME,MIN(NOWPRICE)  FROM  PRODUCTION.PRODUCT;
```

DM 系统会报错，因为 NAME 是一个行集合，而最低价格是唯一值。

至于 MAX 的使用格式与 MIN 是完全一样的，读者可以自己举一反三。

2．求平均值集函数 AVG 和总和集函数 SUM

例 求折扣小于 7 的图书的平均现价。

```
SELECT  AVG(NOWPRICE)  FROM     PRODUCTION.PRODUCT
WHERE  DISCOUNT < 7;
```

查询结果为：23.15

3．求总个数集函数 COUNT

例 查询已登记供应商的个数。

```
SELECT  COUNT(*)  FROM  PURCHASING.VENDOR;
```

查询结果为：12

由此例可看出，COUNT(*)的结果是 VENDOR 表中的总行数，由于主关键字不允许有相同值，因此，它不需要使用保留字 DISTINCT。

例 查询目前销售的图书的出版商的个数。

```
SELECT COUNT(DISTINCT PUBLISHER) FROM PRODUCTION.PRODUCT;
```

查询结果为：9

由于一个出版商可出版多种图书，因而此例中一定要用 DISTINCT 才能得到正确结果。

4．求方差集函数 VARIANCE、标准差函数 STDDEV 和样本标准差函数 STDDEV_SAMP

例 求图书的现价方差。

```
SELECT VARIANCE(NOWPRICE) FROM PRODUCTION.PRODUCT;
```

查询结果为：1.3664888888888888888888888888888889E2

例 求图书的现价标准差。

```
SELECT STDDEV(NOWPRICE) FROM PRODUCTION.PRODUCT;
```

查询结果为：11.689692

例 求图书的现价样本标准差。

```
SELECT STDDEV_SAMP(NOWPRICE) FROM PRODUCTION.PRODUCT;
```

查询结果为：11.689692

5. 求区间范围内的最大值函数 AREA_MAX

例 求图书的现价在 20~30 之间的最大值。

```
SELECT area_max(NOWPRICE,20,30) FROM PRODUCTION.PRODUCT;
```

查询结果为：25.5000

6. 求 FIRST/LAST 集函数

例 求每个用户最早定的商品中花费最多和最少的金额。

```
select CUSTOMERID, max(TOTAL) keep (dense_rank first order by ORDERDATE) max_val, min(TOTAL) keep (dense_rank first order by ORDERDATE) min_val from SALES.SALESORDER_HEADER group by CUSTOMERID;
```

查询结果为：

```
CUSTOMERID    MAX_VAL MIN_VAL
-----------    -------    -------
1              36.90      36.90
```

4.1.4 分析函数

DM7 分析函数为用户分析数据提供了一种更加简单高效的处理方式。如果不使用分析函数，则必须使用连接查询、子查询或者视图，甚至复杂的存储过程实现。引入分析函数后，只需要简单的 SQL 语句，并且执行效率方面也有大幅提高。

分析函数主要用于计算基于组的某种聚合值。与集函数的主要区别是，分析函数对于每组返回多行，而集函数对于每个分组只返回一行。多行形成的组称为窗口，窗口决定了执行当前行的计算范围，窗口的大小可以由组中定义的行数或者范围值滑动。

分析函数的分析子句语法如下：

```
<分析函数>::=(<参数>) OVER (<分析子句>)
<分析子句>::=( [<PARTITION BY 项>] [<ORDER BY 项> ] [<窗口子句>])
```

参数

1.分析函数名：包括 8 种方式 AVG、COUNT、MAX、MIN、SUM、RANK、DENSE_RANK、

ROW_NUMBER；

2.函数参数：对于 AVG、COUNT、MAX、MIN、SUM 这 5 类分析函数的参数和作为集函数时的参数一致，目前不支持 DISTINCT<列名>；

3. PARTITION BY 项：为常量表达式或者列名；

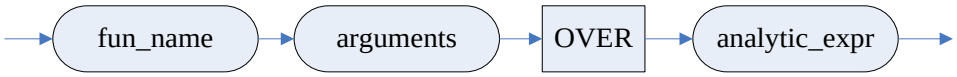
4. ORDER BY 项：为常量表达式或者列名；

5.窗口子句：窗口子句包括 5 类：

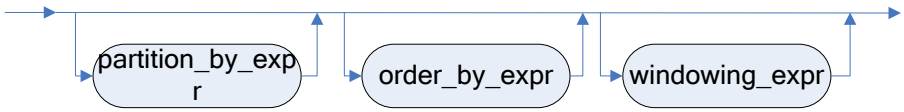
1) ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW，表示该组的第一行到当前行；
2) ROWS BETWEEN CURRENT ROW AND CURRENT ROW，表示当前行到当前行；
3) ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING，表示该组的第一行到最后一行；
4) ROWS UNBOUNDED PRECEDING，和 1)等价；
5) ROWS CURRENT ROW，和 2)等价。

图例

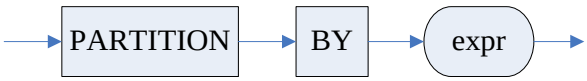
分析函数语法如下：



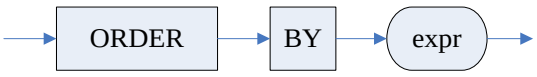
analytic_expr



partition_by_expr



order_by_expr



使用说明

1. 分析函数只能出现在选择项或者 ORDER BY 子句中；
2. 分析函数参数、PARTITION BY 项和 ORDER BY 项中不允许使用分析函数，即不允许嵌套；
3. 当 PARTITION BY 项包含常量表达式时，表示以整个结果集分区；ORDER BY 项中包含常量表达式时，表示以该常量排序，即保持原来结果集顺序；
4. AVG、COUNT、MAX、MIN、SUM 这 5 类分析函数的参数和返回的结果集的数据类型与对应的集函数保持一致，详细参见集函数部分。

下面按分析函数的功能分别进行介绍。

1．分析函数 MAX 和集函数 MIN

例 查询折扣大于 7 的图书作者以及最大折扣。

```
SELECT    AUTHOR,    MAX(DISCOUNT) OVER (PARTITION BY AUTHOR)    AS MAX
FROM     PRODUCTION.PRODUCT
WHERE    DISCOUNT > 7;
```

查询结果如表 4.1.10 所示：

表 4.1.10

AUTHOR	MAX
曹雪芹,高鹗	8.0
施耐庵,罗贯中	7.5
严蔚敏,吴伟民	8.5

需要说明的是：如果使用的是集函数 MAX，那么得到的是所有图书中折扣的最大值，并不能查询出作者，使用了分析函数，就可以对作者进行分区，得到每个作者所写的图书中折扣最大的值。MIN 的含义和 MAX 类似。

2．分析函数 AVG 和分析函数 SUM

例 求折扣小于 7 的图书作者和平均价格。

```
SELECT    AUTHOR,    AVG(NOWPRICE) OVER (PARTITION BY AUTHOR) as AVG
FROM      PRODUCTION.PRODUCT
WHERE     DISCOUNT < 7;
```

查询结果如表 4.1.11 所示：

表 4.1.11

AUTHOR	AVG
(日)佐佐木洋子	42
陈满麒	114
海明威	6.1
金庸	21.7
鲁迅	20
王树增	37.7

例 求折扣大于 8 的图书作者和书的总价格。

```
SELECT     AUTHOR,    SUM(NOWPRICE) OVER (PARTITION BY AUTHOR)    as SUM
FROM      PRODUCTION.PRODUCT
WHERE     DISCOUNT >8;
```

查询结果如表 4.1.12 所示：

表 4.1.12

AUTHOR	SUM
严蔚敏,吴伟民	25.5

3．分析函数 COUNT

例 查询信用级别为“很好”的已登记供应商的名称和个数。

```
SELECT    NAME,    COUNT(*) OVER (PARTITION    BY    CREDIT) AS CNT
FROM     PURCHASING.VENDOR
WHERE    CREDIT = 2;
```

查询结果如表 4.1.13 所示：

表 4.1.13

NAME	CNT
长江文艺出版社	2

上海画报出版社	2

由此例可看出，COUNT(*)的结果是 VENDOR 表中的按 CREDIT 分组后的总行数。

4．排名分析函数 RANK、DENSE_RANK 和 ROW_NUMBER

例 求按销售额排名的销售代表对应的雇员号和排名。

```
SELECT  EMPLOYEEID,  RANK() OVER (ORDER BY SALESLASTYEAR) as RANK
FROM  SALES.SALESPERSON;
```

查询结果如表 4.1.14 所示：

表 4.1.14

EMPLOYEEID	RANK
4	1
5	2

RANK()排名函数按照指定 ORDER BY 项进行排名，如果值相同，则排名相同，例如销售额相同的排名相同，该函数使用非密集排名，例如两个第 1 名后，下一个就是第 3 名；与之对应的是 DENSE_RANK()，表示密集排名，例如两个第 1 名之后，下一个就是第 2 名。ROW_NUMBER()表示按照顺序编号，不区分相同值，即从 1 开始编号。

5．窗口的使用

例 按照作者分类，求到目前为止图书价格最贵的作者和价格。

```
SELECT AUTHOR,
    MAX(NOWPRICE) OVER(PARTITION  BY  AUTHOR  ORDER  BY  NOWPRICE ROWS
    UNBOUNDED  PRECEDING)  AS  MAX_PRICE
FROM  PRODUCTION.PRODUCT;
```

查询结果如表 4.1.15 所示：

表 4.1.15

AUTHOR	MAX_PRICE
(日)佐佐木洋子	42.0000
曹雪芹,高鹗	15.2000
陈满麒	11.4000
海明威	6.1000
金庸	21.7000
刘毅	11.1000
鲁迅	20.0000
施耐庵,罗贯中	14.3000
王树增	37.7000
严蔚敏,吴伟民	25.5000

分析函数中的窗口限定了计算的范围，ROWS UNBOUNDED PRECEDING 表示该组的第一行开始到当前行，等价于 ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW。

4.1.5 情况表达式

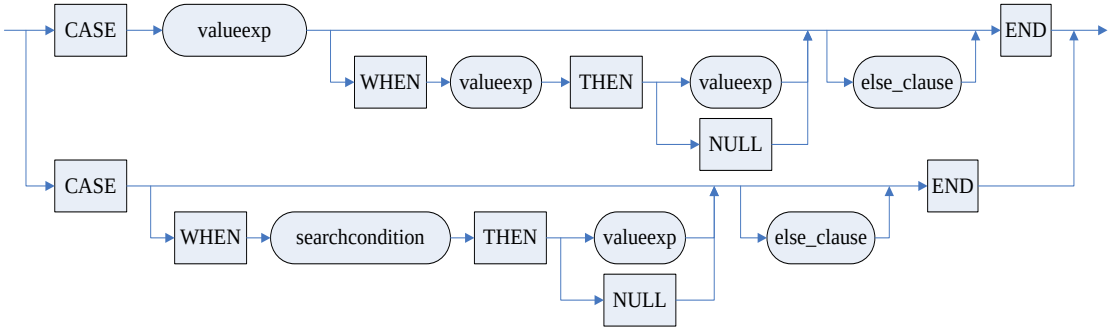
<值表达式>可以为一个<列引用>、<集函数>、<标量子查询>或<情况表达式>等等。<情况表达式>包括<情况缩写词>和<情况说明>两大类。<情况缩写词>包括函数 NULLIF 和 COALESCE，在 DM 中被划分为空值判断函数。具体函数说明请见 8.4 节。下面详细介绍<情况说明>表达式。

<CASE 情况说明>的语法和语义如下：

语法格式

```
<情况说明> ::= <简单情况> | <搜索情况>

<简单情况> ::= CASE
<值表达式>
{<简单 WHEN 子句> }
[<ELSE 子句>]
END

<搜索情况> ::= CASE
[<搜索 WHEN 子句>]
[<ELSE 子句>]
END
<简单 WHEN 子句> ::= WHEN <值表达式> THEN <结果>
<搜索 WHEN 子句> ::= WHEN <搜索条件> THEN <结果>
<结果> ::= <值表达式> | NULL
```

图例



功能

指明一个条件值。将搜索条件作为输入并返回一个标量值。

使用说明

1．在<情况说明>中至少有一个<结果>应该指明<值表达式>；

2．如果未指明<ELSE 子句>，则隐含 ELSE NULL；

3. <简单情况>中，CASE 运算数的数据类型必须与<简单 WHEN 子句>中的<值表达式>的数据类型是可比较的，且与 ELSE 子句的结果也是可比较的；

4．<情况说明>的数据类型由<结果>中的所有<值表达式>的数据类型确定；

1) 如果<结果>指明 NULL，则它的值是空值；

2) 如果<结果>指明<值表达式>，则它的值是该<值表达式>的值。

5．如果在<情况说明>中某个<搜索 WHEN 子句>的<搜索条件>为真，则<情况说明>的

值是其<搜索条件>为真的第一个<搜索 WHEN 子句>的<结果>的值，并按照<情况说明>的数据类型来转换；

6．如果在<情况说明>中没有一个<搜索条件>为真，则<情况表达式>的值是其显式或隐式的<ELSE 子句>的<结果>的值，并按照<情况说明>的数据类型来转换；

7．CASE 表达式查询列名为“CASE……END”这部分，最大长度 124 字节，如果大于 124 字节则后面部分截断。

举例说明

例 查询图书信息，如果当前销售价格大于 20 元，返回‘昂贵’，如果当前销售价格小于等于 20 元，大于等于 10 元，返回‘普通’，如果当前销售价格小于 10 元，返回‘便宜’。

```
SELECT NAME,
        CASE
            WHEN NOWPRICE > 20 THEN    '昂贵'
            WHEN NOWPRICE <= 20 AND NOWPRICE >= 10 THEN '普通'
            ELSE '便宜'
        END AS  选择
FROM PRODUCTION.PRODUCT;
```

查询结果如表 4.1.16 所示：

表 4.1.16

NAME	CASEWHENNOWPRICE>20THEN'昂贵'WHEN(NOWPRICE<=20ANDNOWPRICE>=10)THEN'普通'ELSE'便宜'END
红楼梦	普通
水浒传	普通
老人与海	便宜
射雕英雄传(全四册)	昂贵
鲁迅文集(小说、散文、杂文)全两册	普通
长征	昂贵
数据结构(C 语言版)(附光盘)	昂贵
工作中无小事	普通
突破英文基础词汇	普通
噼里啪啦丛书(全 7 册)	昂贵

例 在 VERDOR 中如果 NAME 为中华书局或清华大学出版社，且 CREDIT 为 1 则返回'采购'，否则返回'考虑'。

```
SELECT NAME,
    CASE
            WHEN (NAME = '中华书局' OR NAME = '清华大学出版社') AND CREDIT = 1 THEN '采购'
            ELSE '考虑'
    END    AS  选择
FROM PURCHASING.VENDOR;
```

查询结果如表 4.1.17 所示：

表 4.1.17

NAME	选择

上海画报出版社	考虑
长江文艺出版社	考虑
北京十月文艺出版社	考虑
人民邮电出版社	考虑
清华大学出版社	采购
中华书局	采购
广州出版社	考虑
上海出版社	考虑
21 世纪出版社	考虑
外语教学与研究出版社	考虑
社械工业出版社	考虑
文学出版社	考虑

例 在上述表中将 NAME 为中华书局，CREDIT 为 1 的元组返回。

```
SELECT NAME, CREDIT FROM PURCHASING.VENDOR
WHERE NAME IN (SELECT CASE
                    WHEN CREDIT = 1 THEN '中华书局'
                    ELSE 'NOT EQUAL'
                    END
                    FROM PURCHASING.VENDOR);
```

查询结果如表 4.1.18 所示：

表 4.1.18

NAME	CREDIT
中华书局	1

例 在上述表中，若 CREDIT 大于 1 则修改该值为 1。

```
UPDATE PURCHASING.VENDOR SET CREDIT = CASE
                                WHEN CREDIT > 1 THEN 1
                                ELSE CREDIT
                                END;
SELECT NAME, CREDIT    FROM PURCHASING.VENDOR;
```

查询结果如表 4.1.19 所示：

表 4.1.19

NAME	CREDIT
上海画报出版社	1
长江文艺出版社	1
北京十月文艺出版社	1
人民邮电出版社	1
清华大学出版社	1
中华书局	1
广州出版社	1
上海出版社	1
21 世纪出版社	1

外语教学与研究出版社	1
机械工业出版社	1
文学出版社	1

4.2 连接查询

如果一个查询包含多个表(>=2)，则称这种方式的查询为连接查询。DM 的连接查询方式包括：交叉连接(cross join)、内连接、自然连接、外连接、哈希连接。连接查询语法如下：

```
SELECT <选择列表><FROM 子句> [<WHERE 子句> | <CONNECT BY 子句> | <GROUP BY 子句> |
<HAVING 子句>]
<FROM 子句>::= FROM <表项>
<表项>::=<单表> | <连接表项>
<连接表项>::=<交叉连接>|<限定连接>|(<连接表>)
<交叉连接>::=<表引用> CROSS JOIN <表引用>
<限定连接>::=<表引用>[NATURAL][<连接类型>] [<强制连接类型>][JOIN] <表引用>[<连接条件>]
<连接类型>::=INNER|<外连接类型>[OUTER]
<外连接类型>::=LEFT | RIGHT | FULL
<强制连接类型>::=HASH
<连接条件>::=ON <搜索条件> | USING ( <连接列列名> [{,<连接列列名>}] )
```

下面分别举例说明。

4.2.1 交叉连接

1. 无过滤条件

对连接的两张表记录做笛卡尔集，产生最终结果输出。

例 SALESPERSON 和 EMPLOYEE 通过交叉连接查询 HAIRDATE 和 SALESLASTYEAR。

```
SELECT T1.HAIRDATE, T2.SALESLASTYEAR
FROM    RESOURCES.EMPLOYEE T1 CROSS JOIN SALES.SALESPERSON T2;
```

查询结果如表 4.2.1 所示。

表 4.2.1

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000

2005-5-2	10.0000
2005-5-2	20.0000
2002-5-2	10.0000
2002-5-2	20.0000
2004-5-2	10.0000
2004-5-2	20.0000

2. 有过滤条件

对连接的两张表记录做笛卡尔集，根据 where 条件进行过滤，产生最终结果输出。

例 查询性别为男性的员工的姓名与职务。

```
SELECT  T1.NAME, T2.TITLE
FROM PERSON.PERSON T1, RESOURCES.EMPLOYEE T2
WHERE T1.PERSONID = T2.PERSONID AND T1.SEX = 'M';
```

查询结果如表 4.2.2 所示。

表 4.2.2

NAME	TITLE
王刚	销售经理
李勇	采购经理
黄非	采购代表
张平	系统管理员

本例中的查询数据必须来自 PERSON 和 EMPLOYEE 两个表。因此，应在 FROM 子句中给出这两个表的表名(为了简化采用了别名)，在 WHERE 子句中给出连接条件(即要求两个表中 PERSONID 的列值相等)。当参加连接的表中出现相同列名时，为了避免混淆，可在这些列名前加表名前缀。

该例的查询结果是 PERSON 和 EMPLOYEE 在 PERSONID 列上做等值连接产生的。条件“T1.PERSONID = T2.PERSONID”称为连接条件或连接谓词。当连接运算符为“=”号时，称为等值连接，使用其它运算符则称非等值连接。

测例说明：

1) 连接谓词中的列类型必须是可比较的，但不一定要相同，只要可以隐式转换即可；
2) 不要求连接谓词中的列同名；
3) 连接谓词中的比较操作符可以是>、>=、<、<=、=、< >；
4) WHERE 子句中可同时包含连接条件和其它非连接条件。

4.2.2 自然连接(NATURAL JOIN)

把两张连接表中的同名列作为连接条件，进行等值连接，我们称这样的连接为自然连接。自然连接具有以下特点：

1. 连接表中存在同名列；
2. 如果有多个同名列，则会产生多个等值连接条件；
3. 如果连接表中的同名列类型不匹配，则报错处理。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT T1.HAIRDATE,  T2.SALESLASTYEAR
FROM  RESOURCES.EMPLOYEE  T1  NATURAL  JOIN  SALES.SALESPERSON  T2;
```

查询结果如表 4.2.3 所示。

表 4.2.3

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.3 JOIN … USING

这是自然连接的另一种写法，JOIN 关键字指定连接的两张表，USING 指明连接列。要求 USING 中的列存在于两张连接表中。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT HAIRDATE,  SALESLASTYEAR
FROM  RESOURCES.EMPLOYEE  JOIN  SALES.SALESPERSON  USING(EMPLOYEEID);
```

查询结果如表 4.2.4 所示。

表 4.2.4

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.4 JOIN…ON

这是一种连接查询的常用写法，说明是一个连接查询。JOIN 关键字指定连接的两张表，ON 子句指定连接条件表达式。具体采用何种连接方式，由数据库内部分析确定。

例 查询销售人员的入职时间和去年销售总额。

```
SELECT  T1.HAIRDATE,
T2.SALESLASTYEAR
FROM  RESOURCES.EMPLOYEE  T1 JOIN  SALES.SALESPERSON  T2
ON  T1.EMPLOYEEID=T2.EMPLOYEEID;
```

查询结果如表 4.2.5 所示。

表 4.2.5

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.2.5 自连接

数据表与自身进行连接，我们称这种连接为自连接。

自连接查询至少要对一张表起别名，否则，服务器无法识别要处理的是哪张表。

例 对 PURCHASING.VENDOR 表进行自连接查询

```
SELECT T1.NAME,  T2.NAME, T1.ACTIVEFLAG
FROM  PURCHASING.VENDOR T1, PURCHASING.VENDOR T2
```

```
WHERE    T1.NAME = T2.NAME;
```

查询结果如表 4.2.6 所示。

表 4.2.6

NAME	NAME	ACTIVEFLAG
上海画报出版社	上海画报出版社	1
长江文艺出版社	长江文艺出版社	1
北京十月文艺出版社	北京十月文艺出版社	1
人民邮电出版社	人民邮电出版社	1
清华大学出版社	清华大学出版社	1
中华书局	中华书局	1
广州出版社	广州出版社	1
上海出版社	上海出版社	1
21 世纪出版社	21 世纪出版社	1
外语教学与研究出版社	外语教学与研究出版社	1
机械工业出版社	机械工业出版社	1
文学出版社	文学出版社	1

4.2.6 内连接(INNER JOIN)

根据连接条件，结果集仅包含满足全部连接条件的记录，我们称这样的连接为内连接。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和子目录名称。

```
SELECT T1.NAME, T2.NAME
FROM    PRODUCTION.PRODUCT_CATEGORY T1    INNER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.7 所示。

表 4.2.7

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学

计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语

因为 PRODUCT_CATEGORY 中的 NAME 为金融的没有对应的子目录，所以结果集中没有金融类的图书信息。

4.2.7 外连接(OUTER JOIN)

对结果集进行了扩展，会返回一张表的所有记录，对于另一张表无法匹配的字段用 NULL 填充返回，我们称这种连接为外连接。DM 数据库支持三种方式的外连接：左外连接、右外连接、全外连接。

外连接中常用到的术语：左表、右表。根据表所在外连接中的位置来确定，位于左侧的表，称为左表；位于右侧的表，称为右表。例如 select * from T1 left join T2 on T1.c1=T2.d1 T1 表为左表，T2 表为右表。

返回所有记录的表根据外连接的方式而定。

1. 左外连接：返回左表所有记录；
2. 右外连接：返回右表所有记录；
3. 全外连接：返回两张表所有记录。处理过程为分别对两张表进行左外连接和右外连接，然后合并结果集。

下面举例说明。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的所有目录名称和子目录名称，包括没有子目录的目录。

```
SELECT T1.NAME, T2.NAME
FROM    PRODUCTION.PRODUCT_CATEGORY T1    LEFT OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.8 所示。

表 4.2.8

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙

少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语
金融	NULL

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和所有子目录名称，包括没有目录的子目录。

```
SELECT T1.NAME, T2.NAME
FROM    PRODUCTION.PRODUCT_CATEGORY T1    RIGHT OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.9 所示。

表 4.2.9

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇
英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理

管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语
NULL	历史

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的所有目录名称和所有子目录名称。

```
SELECT T1.NAME, T2.NAME
FROM    PRODUCTION.PRODUCT_CATEGORY T1    FULL OUTER JOIN
PRODUCTION.PRODUCT_SUBCATEGORY T2
ON T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID;
```

查询结果如表 4.2.10 所示。

表 4.2.10

NAME	NAME
小说	世界名著
小说	武侠
小说	科幻
小说	四大名著
小说	军事
小说	社会
文学	文集
文学	纪实文学
文学	文学理论
文学	中国古诗词
文学	中国现当代诗
文学	戏剧
文学	民间文学
计算机	计算机理论
计算机	计算机体系结构
计算机	操作系统
计算机	程序设计
计算机	数据库
计算机	软件工程
计算机	信息安全
计算机	多媒体
英语	英语词汇

英语	英语语法
英语	英语听力
英语	英语口语
英语	英语阅读
英语	英语写作
管理	行政管理
管理	项目管理
管理	质量管理与控制
管理	商业道德
管理	经营管理
管理	财务管理
少儿	幼儿启蒙
少儿	益智游戏
少儿	童话
少儿	卡通
少儿	励志
少儿	少儿英语
金融	NULL
NULL	历史

外连接还有一种写法，在连接条件或 where 条件中，在列后面增加(+)指示左外连接或者右外连接。如果表 A 和表 B 连接，连接条件或者 where 条件中，A 的列带有(+)后缀，则认为是 B left join A。如果用户的(+)指示引起了外连接环，则报错。下面举例说明。

例 从 PRODUCT_CATEGORY、PRODUCT_SUBCATEGORY 中查询图书的目录名称和所有子目录名称，包括没有目录的子目录。

```
SELECT T1.NAME, T2.NAME
FROM PRODUCTION.PRODUCT_CATEGORY T1, PRODUCTION.PRODUCT_SUBCATEGORY T2
WHERE T1.PRODUCT_CATEGORYID(+) = T2.PRODUCT_CATEGORYID;
```

查询结果如上表 4.2.9 所示。

4.2.8 哈希连接(HASH JOIN)

DM 数据库支持对查询强制指定连接方式，可以指定为哈希连接。哈希连接处理过程为对一张数据表以连接列为哈希键，构造哈希表，另一张表使用连接列进行哈希探测，返回满足连接条件的记录。语法如下：

```
<限定连接>::=<表引用>NATURAL <连接类型> JOIN <表引用>
或 <限定连接>::=<表引用> <连接类型> <强制连接类型>JOIN <表引用>[<连接条件>]
<连接类型>::=INNER|<外连接类型>[OUTER]
<外连接类型>::=LEFT | RIGHT | FULL
<强制连接类型>::=HASH
<连接条件>::=ON <搜索条件> | USING ( <连接列列名> [{,<连接列列名>}] )
```

从语法看出，<强制连接类型>中显式指定为 HASH，则会采用哈希连接。

例 查询销售人员的入职时间和去年销售总额

```
SELECT   T1.HAIRDATE,
```

```
T2.SALESLASTYEAR
FROM RESOURCES.EMPLOYEE T1 INNER HASH JOIN SALES.SALESPERSON T2
ON T1.EMPLOYEEID=T2.EMPLOYEEID;
```

查询结果如表 4.2.11 所示。

表 4.2.11

HAIRDATE	SALESLASTYEAR
2002-5-2	10.0000
2002-5-2	20.0000

4.3 子查询

在 DM_SQL 语言中，一个 SELECT-FROM-WHERE 语句称为一个查询块，如果在一个查询块中嵌套一个或多个查询块，我们称这种查询为子查询。子查询会返回一个值(标量子查询)或一个表(表子查询)。它通常采用(SELECT…)的形式嵌套在表达式中。子查询语法如下：

```
<子查询> ::= ( <查询表达式>)
```

即子查询是嵌入括弧的<查询表达式>，而这个<查询表达式>通常是一个 SELECT 语句。它有下列限制：

1. 在子查询中不得有 ORDER BY 子句；
2. 子查询允许 TEXT 类型与 CHAR 类型值比较。比较时，取出 TEXT 类型字段的最多 8188 字节与 CHAR 类型字段进行比较；如果比较的两字段都是 TEXT 类型，则最多取 300*1024 字节进行比较；
3. 子查询不能包含在集函数中；
4. 在子查询中允许嵌套子查询。

按子查询返回结果的形式，DM 子查询可分为两大类：

1. 标量子查询：只返回一行一列；
2. 表子查询：可返回多行多列。

4.3.1 标量子查询

标量子查询是一个普通 SELECT 查询，它只应该返回一条记录。如果返回结果多于一行则会提示无效子查询。

下面是一个标量子查询的例子(请先关闭自动提交功能，否则 COMMIT 与 ROLLBACK 会失去效果)：

```
SELECT 'VALUE IS', (SELECT ADDRESS1 FROM PERSON.ADDRESS WHERE ADDRESSID = 1)
FROM PERSON.ADDRESS_TYPE;
-- 子查询只有一列，结果正确
SELECT 'VALUE IS', LEFT((SELECT ADDRESS1 FROM PERSON. ADDRESS WHERE ADDRESSID =
1), 8) FROM PERSON.ADDRESS_TYPE;

-- 函数+标量子查询，结果正确
        SELECT 'VALUE IS', (SELECT ADDRESS1, CITY FROM PERSON.ADDRESS WHERE
ADDRESSID = 1) FROM PERSON.ADDRESS_TYPE;
        -- 返回列数不为 1，报错

        SELECT 'VALUES IS', (SELECT ADDRESS1 FROM PERSON.ADDRESS)
```

```
        FROM    PERSON.ADDRESS_TYPE;
        -- 子查询返回行值多于一个，报错
           DELETE FROM SALES.SALESORDER_DETAIL;
        SELECT 'VALUE IS', (SELECT ORDERQTY FROM SALES.SALESORDER_DETAIL)
FROM    SALES.CUSTOMER;
        -- 子查询有 0 行，结果返回 NULL

UPDATE PRODUCTION.PRODUCT SET PUBLISHER =
(SELECT NAME FROM PURCHASING.VENDOR WHERE VENDORID = 2)
WHERE PRODUCTID = 5;

UPDATE PRODUCTION.PRODUCT_VENDOR SET STANDARDPRICE =
(SELECT AVG(NOWPRICE) FROM PRODUCTION.PRODUCT)
WHERE PRODUCTID = 1;
-- Update 语句中允许使用标量子查询

INSERT INTO PRODUCTION.PRODUCT_CATEGORY(NAME) VALUES
(( SELECT NAME FROM PRODUCTION.PRODUCT_SUBCATEGORY
WHERE PRODUCT_SUBCATEGORYID= 41));
-- Insert 语句中允许使用标量子查询
```

例如，查询通常价格最小的供应商的名称和最小价格：

```
SELECT NAME, (SELECT    MIN(STANDARDPRICE)
                    FROM    PRODUCTION.PRODUCT_VENDOR T1
                    WHERE T1.VENDORID = T2.VENDORID)
FROM    PURCHASING.VENDOR T2;
```

4.3.2 表子查询

表子查询经常类似标量子查询，单列构成了子查询的选择清单，但它的查询结果允许返回多行。可以从上下文中区分出表子查询：在其前面始终有一个只对表子查询的算符：<比较算符>ALL、<比较算符>ANY(或是其同义词<比较算符> SOME)、IN、EXISTS 或 UNIQUE。

其中，对于 IN/NOT IN 表子查询，DM 支持多列操作。

例 查询职务为销售代表的员工的编号、今年销售总额和去年销售总额。

```
SELECT    EMPLOYEEID, SALESTHISYEAR, SALESLASTYEAR
FROM      SALES.SALESPERSON
WHERE     EMPLOYEEID IN
( SELECT    EMPLOYEEID
   FROM    RESOURCES.EMPLOYEE
   WHERE    TITLE = '销售代表'
);
```

查询结果如表 4.3.1 所示。

表 4.3.1

EMPLOYEEID	SALESTHISYEAR	SALESLASTYEAR
4	8.0000	10.0000
5	8.0000	20.0000

该查询语句的求解方式是：首先通过子查询“SELECT EMPLOYEEID FROM RESOURCES.EMPLOYEE WHERE TITLE = ‘销售代表’”查到职务为销售代表的 EMPLOYEEID 的集合，然后，在 SALESPERSON 表中找到与子查询结果集中的 EMPLOYEEID 所对应员工的的 SALESTHISYEAR 和 SALESLASTYEAR。

在带有子查询的查询语句中，通常也将子查询称内层查询或下层查询。由于子查询还可以嵌套子查询，相对于下一层的子查询，上层查询又称为父查询或外层查询。

由于 DM_SQL 语言所支持的嵌套查询功能可以将一系列简单查询构造成复杂的查询，从而有效地增强了 DM_SQL 语句的查询功能。以嵌套的方式构造语句是 DM_SQL 的“结构

化”的特点。

需要说明的是：上例的外层查询只能用 IN 谓词而不能用比较算符“=”，因为子查询的结果包含多个元组，除非能确定子查询的结果只有一个元组时，才可用等号比较。测例语句也可以用连接查询的方式实现。

```
SELECT T1.EMPLOYEEID, T1.SALESTHISYEAR, T1.SALESLASTYEAR
FROM    SALES.SALESPERSON T1 , RESOURCES.EMPLOYEE T2
WHERE T1.EMPLOYEEID = T2.EMPLOYEEID AND T2.TITLE = '销售代表';
```

例 查询对目录名为小说的图书进行评论的人员名称和评论日期。

采用子查询嵌套方式写出以下查询语句：

```
SELECT    DISTINCT NAME, REVIEWDATE
FROM      PRODUCTION.PRODUCT_REVIEW
WHERE    PRODUCTID    IN
( SELECT    PRODUCTID
   FROM    PRODUCTION.PRODUCT
   WHERE    PRODUCT_SUBCATEGORYID IN
   ( SELECT    PRODUCT_SUBCATEGORYID
      FROM      PRODUCTION.PRODUCT_SUBCATEGORY
      WHERE    PRODUCT_CATEGORYID IN
      ( SELECT    PRODUCT_CATEGORYID
         FROM    PRODUCTION.PRODUCT_CATEGORY
         WHERE    NAME = '小说'
      )
   )
);
```

查询结果如表 4.3.2 所示。

表 4.3.2

NAME	REVIEWDATE
刘青	2007-05-06
桑泽恩	2007-05-06

该语句采用了四层嵌套查询方式，首先通过最内层子查询从 PRODUCT_CATEGORY 中查出目录名为小说的目录编号，然后从 PRODUCT_SUBCATEGORY 中查出这些目录编号对应的子目录编号，接着从 PRODUCT 表中查出这些子目录编号对应的图书的编号，最后由最外层查询查出这些图书编号对应的评论人员和评论日期。

此例也可用四个表的连接来完成。

从上例可以看出，当查询涉及到多个基表时，嵌套子查询与连接查询相比，前者由于是逐步求解，层次清晰，易于阅读和理解，具有结构化程序设计的优点。

在许多情况下，外层子查询与内层子查询常常引用同一个表，如下例所示。

例 查询当前价格低于红楼梦的图书的名称、作者和当前价格。

```
SELECT    NAME, AUTHOR,    NOWPRICE
FROM    PRODUCTION.PRODUCT
WHERE    NOWPRICE < ( SELECT NOWPRICE FROM PRODUCTION.PRODUCT
                          WHERE NAME = '红楼梦');
```

查询结果如表 4.3.3 所示。

表 4.3.3

NAME	AUTHOR	NOWPRICE
水浒传	施耐庵，罗贯中	14.3000
老人与海	海明威	6.1000
工作中无小事	陈满麒	11.4000
突破英文基础词汇	刘毅	11.1000

此例的子查询与外层查询尽管使用了同一表名，但作用是不一样的。子查询是在该表中

红楼梦的图书价格，而外查询是在 PRODUCT 表 NOWPRICE 列查找小于该值的集合，从而得到这些值所对应的名称和作者。DM_SQL 语言允许为这样的表引用定义别名：

```
SELECT  NAME, AUTHOR,  NOWPRICE
FROM  PRODUCTION.PRODUCT  T1
WHERE  T1.NOWPRICE < ( SELECT  T2.NOWPRICE
FROM  PRODUCTION.PRODUCT  T2
                              WHERE  T2.NAME = '红楼梦');
```

该语句也可以采用连接方式实现：

```
SELECT  T1.NAME, T1.AUTHOR, T1.NOWPRICE
FROM  PRODUCTION.PRODUCT T1 ,  PRODUCTION.PRODUCT T2
WHERE  T2.NAME = '红楼梦' AND T1.NOWPRICE < T2.NOWPRICE;
```

例 查询图书的出版社和产品供应商名称相同的图书编号和名称。

```
SELECT  T1.PRODUCTID, T1.NAME
FROM  PRODUCTION.PRODUCT T1, PRODUCTION.PRODUCT_VENDOR T2
WHERE  T1.PRODUCTID = T2.PRODUCTID  AND T1.PUBLISHER = ANY
( SELECT  NAME  FROM  PURCHASING.VENDOR T3
  WHERE  T2.VENDORID = T3.VENDORID);
```

其结果如表 4.3.4 所示。

表 4.3.4

PRODUCTID	NAME
1	红楼梦
2	水浒传
3	老人与海
4	射雕英雄传(全四册)
7	数据结构(C 语言版)(附光盘)
8	工作中无小事
9	突破英文基础词汇
10	噼里啪啦丛书(全 7 册)

此例有一点需要注意：子查询的 WHERE 子句涉及到 PRODUCT_VENDOR.VENDORID(即 T2.VENDORID)，但是其 FROM 子句中却没有提到 PRODUCT_VENDOR。在外部子查询 FROM 子句中命名了 PRODUCT_VENDOR——这就是外部引用。当一个子查询含有一个外部引用时，它就与外部语句相关联，称这种子查询为相关子查询。

例 查询图书的出版社和产品供应商名称不相同的图书编号和名称。

```
SELECT  T1.PRODUCTID, T1.NAME
FROM  PRODUCTION.PRODUCT T1
WHERE  T1.PUBLISHER <> ALL(SELECT  NAME  FROM  PURCHASING.VENDOR );
```

其结果如表 4.3.5 所示。

表 4.3.5

PRODUCTID	NAME
5	鲁迅文集(小说、散文、杂文)全两册
6	长征

4.3.3 派生表子查询

派生表子查询是一种特殊的表子查询。所谓派生表是指 FROM 子句中的 SELECT 语句，并以别名引用这些派生表。在 SELCET 语句的 FROM 子句中可以包含一个或多个派生

表。

说明：在派生表中，如果有重复列名，DM 系统将自动修改其列名。

例 查询每个目录的编号、名称和对应的子目录的数量，并按数量递减排列。

```
SELECT    T1.PRODUCT_CATEGORYID, T1.NAME, T2.NUM
FROM    PRODUCTION.PRODUCT_CATEGORY    T1,
( SELECT PRODUCT_CATEGORYID, COUNT(PRODUCT_SUBCATEGORYID)
   FROM    PRODUCTION.PRODUCT_SUBCATEGORY
   GROUP BY PRODUCT_CATEGORYID
)   AS    T2(PRODUCT_CATEGORYID,NUM)
WHERE T1.PRODUCT_CATEGORYID = T2.PRODUCT_CATEGORYID
ORDER BY T2.NUM
DESC;
```

查询结果如表 4.3.6 所示。

表 4.3.6

PRODUCT_CATEGORYID	NAME	NUM
3	计算机	8
2	文学	7
6	少儿	6
1	小说	6
5	管理	6
4	英语	6

4.3.4 定量比较

量化符 ALL、SOME、ANY 可以用于将一个<数据类型>的值和一个由表子查询返回的值的集合进行比较。

1．ALL

ALL 定量比较要求的语法如下：

```
<标量表达式> <比较算符> ALL <表子查询>
```

其中：

1) <标量表达式>可以是对任意单值计算的表达式；

2) <比较算符>包括＝、＞、＜、＞＝、＜＝或＜＞。

若表子查询返回 0 行或比较算符对表子查询返回的每一行都为 TRUE，则返回 TRUE。若比较算符对于表子查询返回的至少一行是 FALSE，则 ALL 返回 FALSE。

例 查询没有分配部门的员工的编号、姓名和身份证号码。

```
SELECT    T1.EMPLOYEEID, T2.NAME, T1.NATIONALNO
FROM    RESOURCES.EMPLOYEE T1 ,   PERSON.PERSON T2
WHERE    T1.PERSONID = T2.PERSONID    AND    T1.EMPLOYEEID <> ALL
            ( SELECT    EMPLOYEEID    FROM    RESOURCES.EMPLOYEE_DEPARTMENT);
```

查询结果如表 4.3.7 所示。

表 4.3.7

EMPLOYEEID	NAME	NATIONALNO
7	王菲	420921197708051523

例 查询比中华书局所供应的所有图书都贵的图书的编号、名称和现在销售价格。

```
SELECT    PRODUCTID, NAME, NOWPRICE
FROM    PRODUCTION.PRODUCT
WHERE    NOWPRICE    > ALL
```

```
( SELECT   T1.NOWPRICE
   FROM   PRODUCTION.PRODUCT   T1 , PRODUCTION.PRODUCT_VENDOR T2
   WHERE T1.PRODUCTID = T2.PRODUCTID AND T2.VENDORID =
   ( SELECT   VENDORID   FROM   PURCHASING.VENDOR
      WHERE   NAME = '中华书局'
    )
)
AND   PRODUCTID   <>   ALL
( SELECT   T1.PRODUCTID
   FROM   PRODUCTION.PRODUCT_VENDOR   T1 , PURCHASING.VENDOR   T2
   WHERE   T1.VENDORID =   T2.VENDORID   AND T2.NAME = '中华书局'
);
```

查询结果如表 4.3.8 所示。

表 4.3.8

PRODUCTID	NAME	NOWPRICE
4	射雕英雄传(全四册)	21.7000
5	鲁迅文集(小说、散文、杂文)全两册	20.0000
6	长征	37.7000
7	数据结构(C 语言版)(附光盘)	25.5000
10	噼里啪啦丛书(全 7 册)	42.0000

2．ANY 或 SOME

ANY 或 SOME 定量比较要求的语法如下：

```
<标量表达式> <比较算符> ANY | SOME <表子查询>
```

SOME 和 ANY 是同义词。如果它们对于表子查询返回的至少一行为 TRUE，则返回为 TRUE。若表子查询返回 0 行或比较算符对表子查询返回的每一行都为 FALSE，则返回 FALSE。

ANY 和 ALL 与集函数的对应关系如表 4.3.9 所示。

表 4.3.9　ANY 和 ALL 与集函数的对应关系

	=	<>	<	<=	>	>=
ANY	IN	不存在	<MAX	<=MAX	>MIN	>=MIN
ALL	不存在	NOT　IN	<MIN	<=MIN	>MAX	>=MAX

在具体使用时，读者完全可根据自己的习惯和需要选用。

4.3.5 带 EXISTS 谓词的子查询

带 EXISTS 谓词的子查询语法如下：

```
<EXISTS 谓词> ::= [NOT] EXISTS <表子查询>
```

EXISTS 判断是对非空集合的测试并返回 TRUE 或 FALSE。若表子查询返回至少一行，则 EXISTS 返回 TRUE，否则返回 FALSE。若表子查询返回 0 行，则 NOT EXISTS 返回 TRUE，否则返回 FALSE。

例 查询职务为销售代表的员工的编号和入职时间。

```
SELECT   T1.EMPLOYEEID , T1.STARTDATE
FROM   RESOURCES.EMPLOYEE_DEPARTMENT T1
WHERE   EXISTS
( SELECT   *   FROM   RESOURCES.EMPLOYEE T2
   WHERE   T2.EMPLOYEEID = T1.EMPLOYEEID   AND T2.TITLE = '销售代表');
```

查询结果如表 4.3.10 所示。

表 4.3.10

EMPLOYEEID	STARTDATE
5	2005-02-01
4	2005-02-01

此例查询需要 EMPLOYEE_DEPARTMENT 表和 EMPLOYEE 表中的数据，其执行方式为：首先在 EMPLOYEE_DEPARTMENT 表的第一行取 EMPLOYEEID 的值为 2，这样对内层子查询则为：

```
(SELECT * FROM RESOURCES.EMPLOYEE T2
WHERE T2.EMPLOYEEID='2' AND T2.TITLE='销售代表');
```

在 EMPLOYEE 表中，不存在满足该条件的行，子查询返回值为假，说明不能取 EMPLOYEE_DEPARTMENT 表的第一行作为结果。系统接着取 EMPLOYEE_DEPARTMENT 表的第二行，又得到 EMPLOYEEID 的值为 4，执行内层查询，此时子查询返回值为真，说明可以取该行作为结果。重复以上步骤……。只有外层子查询 WHERE 子句结果为真时，方可将 EMPLOYEE_DEPARTMENT 表中的对应行送入结果表，如此继续，直到把 EMPLOYEE_DEPARTMENT 表的各行处理完。

从以上分析得出，EXISTS 子查询的查询结果与外表相关，即连接条件中包含内表和外表列，我们称这种类型的子查询为相关子查询；反之，子查询的连接条件不包含外表列，即查询结果不受外表影响，我们称这种类型的子查询为非相关子查询。

4.3.6 多列表子查询

为了满足应用需求，DM 数据库扩展了子查询功能，目前支持多列 IN / NOT IN 子查询。子查询可以是值列表或者查询块。

例 查询活动标志为 1 且信誉为 2 的供应商编号和名称。

```
SELECT VENDORID, NAME
FROM PURCHASING.VENDOR
WHERE (ACTIVEFLAG, CREDIT) IN ((1, 2));
```

查询结果如表 4.3.11 所示。

表 4.3.11

VENDORID	NAME
1	上海画报出版社
2	长江文艺出版社

测例中子查询的选择清单为多列，而看到子查询算符后面跟着的形如((1，2))的表达式我们称之为多列表达式链表，这个多列表达式链表以一个或多个多列数据集构成的集合构成。上述的例子中的多列表达式链表中的元素有两个。

例 查询作者为海明威且出版社为上海出版社或作者为王树增且出版社为人民文学出版社的图书名称和现在销售价格。

```
SELECT NAME, NOWPRICE
FROM PRODUCTION.PRODUCT
WHERE (AUTHOR, PUBLISHER) IN
(( '海明威', '上海出版社'), ('王树增', '人民文学出版社'));
```

查询结果如表 4.3.12 所示。

表 4.3.12

NAME	NOWPRICE
老人与海	6.1000

长征	37.7000

子查询为值列表时，需要注意以下三点：

1. 值列表需要用括号；
2. 值列表之间以逗号分割；
3. 值列表的个数与查询列个数相同。

子查询为查询块的情况如下例所示：

例 查询由采购代表下的供应商是清华大学出版社的订单的创建日期、状态和应付款总额。

```
SELECT    ORDERDATE, STATUS, TOTAL
FROM      PURCHASING.PURCHASEORDER_HEADER
WHERE     (EMPLOYEEID, VENDORID)    IN
(SELECT    T1.EMPLOYEEID,    T2.VENDORID
                FROM    RESOURCES.EMPLOYEE    T1,    PURCHASING.VENDOR    T2
                WHERE    T1.TITLE = '采购代表' AND T2.NAME = '清华大学出版社');
```

查询结果如表 4.3.13 所示。

表 4.3.13

ORDERDATE	STATUS	TOTAL
2006-07-21	1	6400

由例子可以看到，WHERE 子句中有两个条件列，NOT IN 子查询的查询项也由两列构成。

DM 对多列子查询的支持，满足了更多的应用场景。

4.4 公用表表达式

嵌套 SQL 语句如果层次过多，会使 SQL 语句难以阅读和维护。如果将子查询放在临时表中，会使 SQL 语句更容易维护，但同时也增加了额外的 I/O 开销，因此，临时表并不太适合数据量大且频繁查询的情况。为此，在 DM7 中引入了公用表表达式(CTE，COMMON TABLE EXPRESSION)，使用 CTE 可以提高 SQL 语句的可维护性，同时 CTE 要比临时表的效率高很多。

CTE 与派生表类似，具体表现在不存储为对象，并且只在查询期间有效。与派生表的不同之处在于，CTE 可自引用，还可在同一查询中引用多次。

WITH AS 短语，也叫做子查询部分（SUBQUERY FACTORING），它定义一个 SQL 片断，该 SQL 片断会被整个 SQL 语句所用到。它可以有效提高 SQL 语句的可读性，也可以用在 UNION ALL 的不同部分，作为提供数据的部分。

4.4.1 公用表表达式的作用

公用表表达式（CTE）是一个在查询中定义的临时命名结果集，将在 FROM 子句中使用它。每个 CTE 仅被定义一次（但在其作用域内可以被引用任意次），并且在该查询生存期间将一直生存，而且可以使用 CTE 来执行递归操作。

因为 UNION ALL 的每个部分可能相同，但是如果每个部分都去执行一遍的话，则成本太高，所以可以使用 WITH AS 短语，则只要执行一遍即可。如果 WITH AS 短语所定义的表名被调用两次以上，则优化器会自动将 WITH AS 短语所获取的数据放入一个临时表里，如果只是被调用一次则不会，很多查询通过这种方法都可以提高速度。

4.4.2 公用表表达式的使用

语法格式

```
WITH <公用表表达式名 [ ( <列名>{,<列名>} ) ]  AS  ( 公用表表达式子查询语句)>
{, <公用表表达式名 [ ( <列名>{,<列名>} ) ]  AS  ( 公用表表达式子查询语句)>}
```

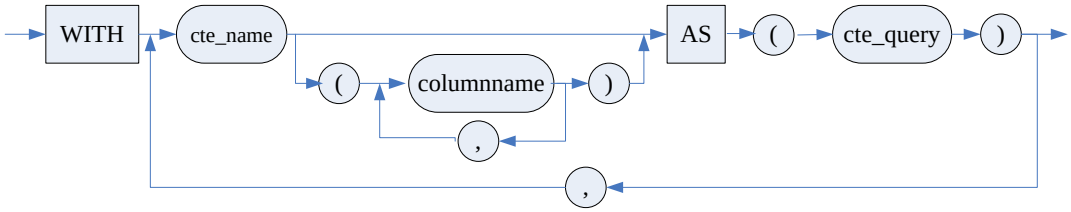
参数

1．<公用表表达式名>　　公用表表达式的有效标识符；

2．<列名>　　指明被创建的公用表表达式中列的名称；

3．<公用表表达式子查询语句> 标识公用表表达式所基于的表的行和列，其语法遵照 SELECT 语句的语法规则。

图例



语句功能

供用户定义公用表表达式，也就是 WITH AS 语句。

使用说明

1．<公用表表达式名>必须与在同一 WITH 子句中定义的任何其他公用表表达式的名称不同，但公用表表达式名可以与基表或基视图的名称相同。在查询中对公用表表达式名的任何引用都会使用公用表表达式，而不使用基对象；

2．<列名>在一个 CTE 定义中不允许出现重复的列名。指定的列名数必须与<公用表表达式子查询语句>结果集中列数匹配。只有在查询定义中为所有结果列都提供了不同的名称时，列名称列表才是可选的；

3．<公用表表达式子查询语句>指定一个结果集填充公用表表达式的 SELECT 语句。除了 CTE 不能定义另一个 CTE 以外，<公用表表达式子查询语句> 的 SELECT 语句必须满足与创建视图时相同的要求；

4．公用表表达式后面必须直接跟使用 CTE 的 SQL 语句，否则无效。

权限

该语句的使用者必须对<查询说明>中的每个表均具有 SELECT 权限。

举例说明

公用表表达式可以认为是在单个 SELECT、INSERT、UPDATE、DELETE 或 CREATE VIEW 语句的执行范围内定义的临时结果集。

例 创建一个表 TEST1 和表 TEST2，并利用公用表表达式对它们进行连接运算。

```
CREATE TABLE TEST1(I INT);
INSERT INTO TEST1 VALUES(1);
INSERT INTO TEST1 VALUES(2);

CREATE TABLE TEST2(J INT);
INSERT INTO TEST2 VALUES(5);
INSERT INTO TEST2 VALUES(6);
INSERT INTO TEST2 VALUES(7);

WITH CTE1(K) AS(SELECT I FROM TEST1 WHERE I > 1),
CTE2(G) AS(SELECT J FROM TEST2 WHERE J > 5)
```

```
SELECT K, G FROM CTE1, CTE2;

运算结果：
K                G
1        2                6
2        2                7
```

例 利用公用表表达式将表 TEST2 中的记录插入到 TEST1 表中。

```
INSERT INTO TEST2 WITH CTE1 AS(SELECT * FROM TEST1)
SELECT * FROM CTE1;

SELECT * FROM TEST2;
运行结果：
        J
----------------
        5
        6
        7
        1
        2
```

4.5 合并查询结果

DM 提供了一种集合运算符：UNION。这种算符将两个或多个查询块的结果集合并为一个结果集输出。语法如下：

语法格式

```
<查询表达式>
UNION [ALL][DISTINCT]
[ ( )<查询表达式> [ ) ];
```

使用说明

1. 每个查询块的查询列数目必须相同；
2. 每个查询块对应的查询列的数据类型必须兼容；
3. 不允许查询列含有 BLOB、CLOB 或 IMAGE、TEXT 等大字段类型；
4. 在 UNION 后的可选项关键字 ALL 的意思是保持所有重复，而没有 ALL 的情况下表示删除所有重复；
5. 在 UNION 后的可选项关键字 DISTINCT 的意思是删除所有重复。

例 查询所有图书的出版商，查询所有图书供应商的名称，将两者连接，并去掉重复行。

```
SELECT PUBLISHER FROM PRODUCTION.PRODUCT
UNION
SELECT NAME FROM PURCHASING.VENDOR ORDER BY 1;
```

查询结果如表 4.5.1 所示。

表 4.5.1

PUBLISHER
21 世纪出版社
北京十月文艺出版社
长江文艺出版社
广州出版社
机械工业出版社
清华大学出版社 （注：末尾含空格）
清华大学出版社（注：末尾不含空格）

人民文学出版社
人民邮电出版社
上海出版社
上海画报出版社
外语教学与研究出版社
文学出版社
中华书局

例 UNION ALL

```
SELECT PUBLISHER FROM PRODUCTION.PRODUCT
UNION ALL
SELECT NAME FROM PURCHASING.VENDOR ORDER BY 1;
```

查询结果如表 4.5.2 所示。

表 4.5.2

PUBLISHER
21 世纪出版社
21 世纪出版社
北京十月文艺出版社
长江文艺出版社
广州出版社
广州出版社
机械工业出版社
机械工业出版社
清华大学出版社
清华大学出版社
人民文学出版社
人民邮电出版社
上海出版社
上海出版社
上海画报出版社
外语教学与研究出版社
外语教学与研究出版社
文学出版社
中华书局
中华书局
中华书局

4.6 GROUP BY 和 HAVING 子句

4.6.1 GROUP BY 子句的使用

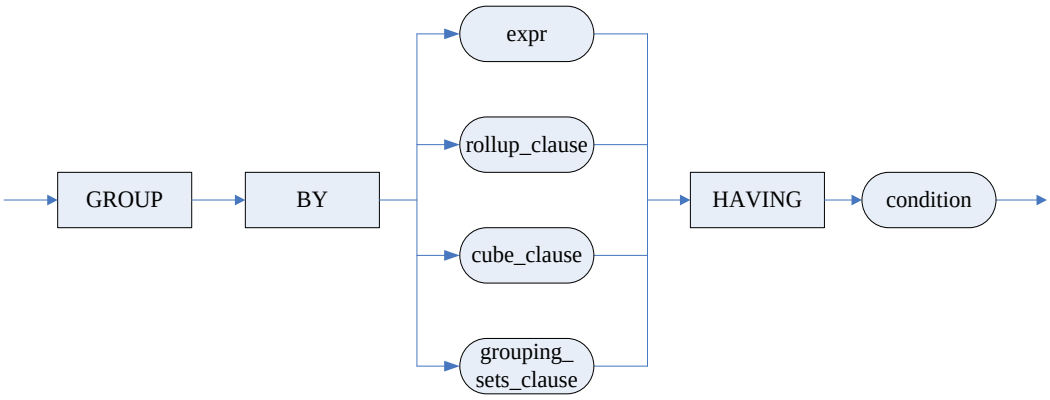
GROUP BY 子句是 SELECT 语句的可选项部分。它定义了成组表。GROUP BY 子句语

法如下：

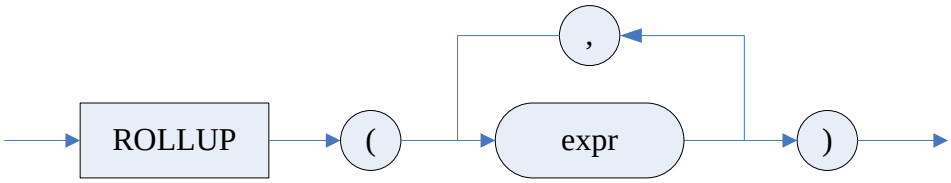
```
<GROUP BY 子句> ::= GROUP BY <分组项> | <ROLLUP 项> | <CUBE 项> | <GROUPING SETS 项>
<分组项>::= <列名> | <值表达式>{,<列名> | <值表达式>}
<ROLLUP 项>::=ROLLUP (<分组项>)
<CUBE 项>::=CUBE (<分组项>)
<GROUPING SETS 项>::=GROUPING SETS (<分组项> | (<分组项>){,<分组项> | (<分组项>)})
```

图例

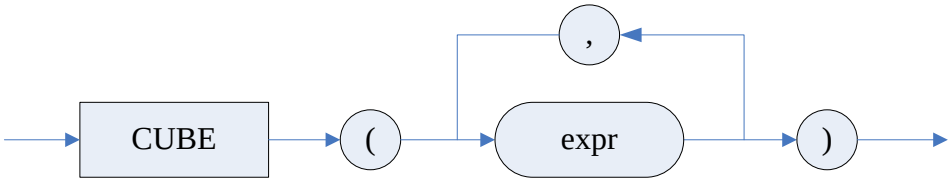
GROUP BY 子句



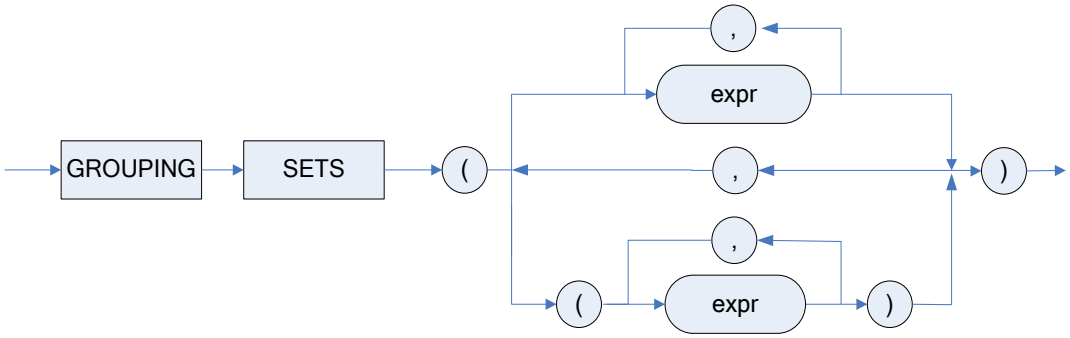
rollup_clause



cube_clause



grouping_sets_clause



GROUP BY 定义了成组表：行组的集合，其中每一个组由其中所有成组列的值都相等的行构成。

例 统计每个部门的员工数。

```
SELECT DEPARTMENTID,COUNT(*) FROM RESOURCES.EMPLOYEE_DEPARTMENT GROUP BY DEPARTMENTID;
```

查询结果如表 4.6.1 所示。

表 4.6.1

DEPARTMENTID	COUNT(*)
4	1
3	1
2	3
1	2

系统执行此语句时，首先将 EMPLOYEE_DEPARTMENT 表按 DEPARTMENTID 列进行分组，相同的 DEPARTMENTID 为一组，然后对每一组使用集函数 COUNT(*)，统计该组内的记录个数，如此继续，直到处理完最后一组，返回查询结果。

如果存在 WHERE 子句，系统先根据 WHERE 条件进行过滤，然后对满足条件的记录进行分组。

此外，GROUP BY 不会对结果集排序。如果需要排序，可以使用 ORDER BY 子句。

例 求小说类别包含的子类别所对应的产品数量，并按子类别编号的升序排列。

```
SELECT A1.PRODUCT_SUBCATEGORYID AS 子分类编号,A3.NAME AS 子分类名,count(*)AS 数量
FROM PRODUCTION.PRODUCT A1,
PRODUCTION.PRODUCT_CATEGORY A2,
PRODUCTION.PRODUCT_SUBCATEGORY A3
WHERE A1.PRODUCT_SUBCATEGORYID=A3.PRODUCT_SUBCATEGORYID
AND A2.PRODUCT_CATEGORYID=A3.PRODUCT_CATEGORYID
AND A2.NAME='小说'
GROUP BY A1.PRODUCT_SUBCATEGORYID,A3.NAME
ORDER BY A1.PRODUCT_SUBCATEGORYID
```

查询结果如表 4.6.2 所示。

表 4.6.2

子分类编号	子分类名	数量
1	世界名著	1
2	武侠	1
4	四大名著	2

使用 GROUP BY 要注意以下问题：

1．在 GROUP BY 子句中的每一列必须明确地命名属于在 FROM 子句中命名的表的一列。成组列的数据类型不能是多媒体数据类型；

2．分组列不能为集函数表达式或者在 SELECT 子句中定义的别名；

3．当分组列值包含空值时，则空值作为一个独立组；

4．当分组列包含多个列名时，则按照 GROUP BY 子句中列出现的顺序进行分组；

5．GROUP BY 子句中至多可包含 128 个分组列。

4.6.2 ROLLUP 的使用

ROLLUP 主要用于统计分析，对分组列以及分组列的部分子集进行分组，输出用户需要的结果。语法如下：

```
<GROUP BY 子句> ::= GROUP BY ROLLUP (<列名>{,<列名>})
```

假如 ROLLUP 分组列为(A, B, C)的话，首先对(A,B,C)进行分组，然后对(A,B)进行分组，

接着对(A)进行分组，最后对全表进行查询，无分组列，其中查询项中出现在 ROLLUP 中的列设为 NULL。查询结果是把每种分组的结果集进行 union all 合并输出。如果分组列为 n 列，则共有 n+1 种组合方式。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT CITY , ADDRESS1, COUNT(*) as NUMS FROM PERSON.ADDRESS GROUP BY ROLLUP(CITY, ADDRESS1);
```

查询结果如表 4.6.3 所示。

表 4.6.3

CITY	ADDRESS1	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
武汉市洪山区	洪山区保利花园 50-1-304	1
武汉市洪山区	洪山区保利花园 51-1-702	1
武汉市洪山区	洪山区关山春晓 51-1-702	1
武汉市洪山区	洪山区关山春晓 55-1-202	1
武汉市洪山区'	洪山区关山春晓 10-1-202	1
武汉市洪山区	洪山区关山春晓 11-1-202	1
武汉市洪山区	洪山区光谷软件园 C1_501	1
武汉市青山区	青山区青翠苑 1 号	1
武汉市武昌区	武昌区武船新村 115 号	1
武汉市武昌区	武昌区武船新村 1 号	1
武汉市汉阳区	汉阳大道熊家湾 15 号	1
武汉市江汉区	江汉区发展大道 561 号	1
武汉市江汉区	江汉区发展大道 555 号	1
武汉市江汉区	江汉区发展大道 423 号	1
武汉市洪山区	NULL	9
武汉市青山区	NULL	1
武汉市武昌区	NULL	2
武汉市汉阳区	NULL	1
武汉市江汉区	NULL	3
NULL	NULL	16

测例中的查询等价于：

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CITY, ADDRESS1
UNION ALL
SELECT CITY , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CITY
UNION ALL
SELECT NULL , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS;
```

4.6.3 CUBE 的使用

CUBE 的使用场景与 ROLLUP 类似，常用于统计分析，对分组列以及分区列的所有子集进行分组，输出所有分组结果。语法如下：

```
<GROUP BY 子句> ::= GROUP BY CUBE (<列名>{,<列名>})
```

假如，CUBE 分组列为(A, B, C)，则首先对(A,B,C)进行分组，然后依次对(A,B)、(A,C)、

(A)、(B,C)、(B)、(C)6 六种情况进行分组，最后对全表进行全表进行查询，无分组列，其中查询项存在于 cube 列表的列设置为 NULL。输出为每种分组的结果集进行 union all。CUBE 分组共有 2^n 种组合方式。CUBE 最多支持 8 列。

例 按小区住址、所属行政区域统计员工居住分布情况。

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CUBE(CITY, ADDRESS1);
```

查询结果如表 4.6.4 所示。

表 4.6.4

CITY	ADDRESS1	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
武汉市洪山区	洪山区保利花园 50-1-304	1
武汉市洪山区	洪山区保利花园 51-1-702	1
武汉市洪山区	洪山区关山春晓 51-1-702	1
武汉市洪山区	洪山区关山春晓 55-1-202	1
武汉市洪山区	洪山区关山春晓 10-1-202	1
武汉市洪山区	洪山区关山春晓 11-1-202	1
武汉市洪山区	洪山区光谷软件园 C1_501	1
武汉市青山区	青山区青翠苑 1 号	1
武汉市武昌区	武昌区武船新村 115 号	1
武汉市武昌区	武昌区武船新村 1 号	1
武汉市汉阳区	汉阳大道熊家湾 15 号	1
武汉市江汉区	江汉区发展大道 561 号	1
武汉市江汉区	江汉区发展大道 555 号	1
武汉市江汉区	江汉区发展大道 423 号	1
武汉市洪山区	NULL	9
武汉市青山区	NULL	1
武汉市武昌区	NULL	2
武汉市汉阳区	NULL	1
武汉市江汉区	NULL	3
NULL	洪山区 369 号金地太阳城 56-1-202	1
NULL	洪山区 369 号金地太阳城 57-2-302	1
NULL	洪山区保利花园 50-1-304	1
NULL	洪山区保利花园 51-1-702	1
NULL	洪山区关山春晓 51-1-702	1
NULL	洪山区关山春晓 55-1-202	1
NULL	洪山区关山春晓 10-1-202	1
NULL	洪山区关山春晓 11-1-202	1
NULL	洪山区光谷软件园 C1_501	1
NULL	青山区青翠苑 1 号	1
NULL	武昌区武船新村 115 号	1
NULL	武昌区武船新村 1 号	1
NULL	汉阳大道熊家湾 15 号	1
NULL	江汉区发展大道 561 号	1

NULL	江汉区发展大道 555 号	1
NULL	江汉区发展大道 423 号	1
NULL	NULL	16

测例中的查询等价于：

```
SELECT CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CITY, ADDRESS1
UNION ALL
SELECT CITY , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY CITY
UNION ALL
SELECT NULL , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY ADDRESS1
UNION ALL
SELECT NULL , NULL, COUNT(*) AS NUMS FROM PERSON.ADDRESS;
```

4.6.4 GROUPING 的使用

GROUPING 可以看作为集函数，用来标识某列是否为分组列。如果是分组列，GROUPING 值为 0；否则为 1。语法如下：

```
<GROUPING 项>::=GROUPING <列名>
```

使用约束说明：

1. GROUPING 中只能包含一列；
2. GROUPING 只能与 rollup\cube 一起使用；
3. GROUPING 支持表达式运算。例如 GROUPING(c1) + GROUPING(c2)。

例 按小区住址和所属行政区域统计员工居住分布情况。

```
SELECT GROUPING(CITY) AS G_CITY,GROUPING(ADDRESS1) AS G_ADD, CITY , ADDRESS1, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY ROLLUP(CITY, ADDRESS1);
```

查询结果如表 4.6.5 所示。

表 4.6.5

G_CITY	G_ADD	CITY	ADDRESS1	NUMS
0	0	武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	1
0	0	武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	1
0	0	武汉市洪山区	洪山区保利花园 50-1-304	1
0	0	武汉市洪山区	洪山区保利花园 51-1-702	1
0	0	武汉市洪山区	洪山区关山春晓 51-1-702	1
0	0	武汉市洪山区	洪山区关山春晓 55-1-202	1
0	0	武汉市洪山区'	洪山区关山春晓 10-1-202	1
0	0	武汉市洪山区	洪山区关山春晓 11-1-202	1
0	0	武汉市洪山区	洪山区光谷软件园 C1_501	1
0	0	武汉市青山区	青山区青翠苑 1 号	1
0	0	武汉市武昌区	武昌区武船新村 115 号	1
0	0	武汉市武昌区	武昌区武船新村 1 号	1
0	0	武汉市汉阳区	汉阳大道熊家湾 15 号	1
0	0	武汉市江汉区	江汉区发展大道 561 号	1
0	0	武汉市江汉区	江汉区发展大道 555 号	1
0	0	武汉市江汉区	江汉区发展大道 423 号	1
0	1	武汉市洪山区	NULL	9

0	1	武汉市青山区	NULL	1
0	1	武汉市武昌区	NULL	2
0	1	武汉市汉阳区	NULL	1
0	1	武汉市江汉区	NULL	3
1	1	NULL	NULL	16

4.6.5 GROUPING SETS 的使用

GROUPING SETS 是对 GROUP BY 的扩展，可以指定不同的列进行分组，每个分组列集作为一个分组单元。使用 GROUPING SETS，用户可以灵活的指定分组方式，避免 ROLLUP/CUBE 过多的分组情况，满足实际应用需求。

GROUPING SETS 的分组过程为依次按照每一个分组单元进行分组，最后把每个分组结果进行 UNION ALL 输出最终结果。如果查询项不属于分组列，则用 NULL 代替。语法如下：

```
<GROUP BY 子句> ::= GROUP BY GROUPING SETS (<分组项>)
<分组项>::= <列名> | <值表达式>{,<列名> | <值表达式>}
```

例 按照邮编、住址和行政区域统计员工住址分布情况

```
SELECT CITY , ADDRESS1, POSTALCODE, COUNT(*) AS NUMS FROM PERSON.ADDRESS GROUP BY GROUPING SETS((CITY, ADDRESS1), POSTALCODE);
```

查询结果如表 4.6.6 所示。

表 4.6.6

CITY	ADDRESS1	POSTALCODE	NUMS
武汉市洪山区	洪山区 369 号金地太阳城 56-1-202	NULL	1
武汉市洪山区	洪山区 369 号金地太阳城 57-2-302	NULL	1
武汉市洪山区	洪山区保利花园 50-1-304	NULL	1
武汉市洪山区	洪山区保利花园 51-1-702	NULL	1
武汉市洪山区	洪山区关山春晓 51-1-702	NULL	1
武汉市洪山区	洪山区关山春晓 55-1-202	NULL	1
武汉市洪山区'	洪山区关山春晓 10-1-202	NULL	1
武汉市洪山区	洪山区关山春晓 11-1-202	NULL	1
武汉市洪山区	洪山区光谷软件园 C1_501	NULL	1
武汉市青山区	青山区青翠苑 1 号	NULL	1
武汉市武昌区	武昌区武船新村 115 号	NULL	1
武汉市武昌区	武昌区武船新村 1 号	NULL	1
武汉市汉阳区	汉阳大道熊家湾 15 号	NULL	1
武汉市江汉区	江汉区发展大道 561 号	NULL	1
武汉市江汉区	江汉区发展大道 555 号	NULL	1
武汉市江汉区	江汉区发展大道 423 号	NULL	1
NULL	NULL	430073	9
NULL	NULL	430080	1
NULL	NULL	430063	2
NULL	NULL	430050	1
NULL	NULL	430023	3

测例中的查询等价于：

```
SELECT CITY , ADDRESS1, NULL , COUNT(*) AS NUMS  FROM  PERSON.ADDRESS  GROUP
BY  CITY, ADDRESS1
UNION ALL
SELECT  NULL  ,  NULL,  POSTALCODE  ,COUNT(*)  AS  NUMS  FROM  PERSON.ADDRESS
GROUP  BY  POSTALCODE;
```

4.6.6 HAVING 子句的使用

HAVING 子句是 SELECT 语句的可选项部分，它也定义了一个成组表。HAVING 子句语法如下：

```
<HAVING 子句> ::= HAVING <搜索条件>
<搜索条件>::= <表达式>
```

HAVING 子句定义了一个成组表，其中只含有搜索条件为 TRUE 的那些组，且通常跟随一个 GROUP BY 子句。HAVING 子句与组的关系正如 WHERE 子句与表中行的关系。WHERE 子句用于选择表中满足条件的行，而 HAVING 子句用于选择满足条件的组。

例 统计出同一子类别的产品数量大于 1 的子类别名称，数量，并按数量从小到大的顺序排列。

```
SELECT A2.NAME AS 子分类名, COUNT (*)AS 数量
FROM PRODUCTION.PRODUCT A1,
PRODUCTION.PRODUCT_SUBCATEGORY A2
WHERE A1.PRODUCT_SUBCATEGORYID=A2.PRODUCT_SUBCATEGORYID
GROUP  BY  A2.NAME
HAVING COUNT(*)>1
ORDER  BY  2
```

查询结果如表 4.6.7 所示。

表 4.6.7

子分类名	数量
四大名著	2

系统执行此语句时，首先将 PRODUCT 表和.PRODUCT_SUBCATEGORY 表中的各行按相同的 SUBCATEGORYID 作连接，再按子类别名的取值进行分组，相同的子类别名为一组，然后对每一组使用集函数 COUNT(*)，统计该组内产品的数量，如此继续，直到最后一组。再选择产品数量大于 1 的组作为查询结果。

4.7 ORDER BY 子句

ORDER BY 子句可以选择性地出现在<查询表达式>之后，它规定了当行由查询返回时应具有的顺序。ORDER BY 子句的语法如下：

```
<ORDER BY 子句> ::= ORDER BY
<无符号整数> | <列说明> | <值表达式> [ASC | DESC] [NULLS FIRST|LAST]
{,<无符号整数> | <列说明> | <值表达式> [ASC | DESC] [NULLS FIRST|LAST]}
```

例 将 RESOURCES.DEPARTMENT 表中的资产总值按从大到小的顺序排列。

```
SELECT  *  FROM  RESOURCES.DEPARTMENT  ORDER  BY  DEPARTMENTID  DESC;
```

等价于：

```
SELECT  *  FROM  RESOURCES.DEPARTMENT  ORDER  BY  1  DESC;
```

查询结果如表 4.7.1 所示。

表 4.7.1

DEPARTMENTID	NAME
5	广告部

4	行政部门
3	人力资源
2	销售部门
1	采购部门

例

```
SELECT * FROM RESOURCES.DEPARTMENT ORDER BY 3;
```

系统报错：无效的 ORDER BY 语句。

需要说明的是：

1．ORDER BY 子句为 DBMS 提供了要排序的项目清单和他们的排序顺序：递增顺序(ASC，默认)或是递减顺序(DESC)。它必须跟随<查询表达式>，因为它是在查询计算得出的最终结果上进行操作的；

2．排序键可以是任何在查询清单中的列的名称，或者是对最终结果表的列计算的表达式(即使这一列不在选择清单中)，也可以是子查询，但不允许为集函数。对于 UNION 查询语句，排序键必须在第一个查询子句中出现；

3.<无符号整数>为<值表达式>在 SELECT 后的序列号。当用<无符号整数>代替列名时，<无符号整数>不应大于 SELECT 后<值表达式>的个数。如上面例句中 ORDER BY 5，因查询结果列只有 4 列，无法进行排序，系统将会报错。若采用其他常量表达式(如：-1，3×6)作为排序列，将不影响最终结果表的行输出顺序；

4．无论采用何种方式标识想要排序的结果列，它们都不应为多媒体数据类型(如 IMAGE、TEXT、BLOB 和 CLOB)；

5．当排序列值包含 NULL 时，不论按升序排列，还是按降序排列，含该空值的行始终都排在最前面；

6．当排序列包含多个列名时，系统则按列名从左到右排列的顺序，先按左边列将查询结果排序，当左边排序列值相等时，再按右边排序列排序……如此右推，逐个检查调整，最后得到排序结果；

7．由于 ORDER BY 只能在最终结果上操作，不能将其放在查询中或一个集函数前；

8．ORDER BY 子句中至多可包含 64 个排序列。

4.8 TOP 子句

在 DM 中，可以使用 TOP 子句来筛选结果。语法如下：

```
<TOP 子句>::=TOP <记录数> | <记录数>,<记录数>
<记录数>::= <整数>
```

目前支持两种方式：

1. SELECT TOP N ：选择结果的前 N 条记录

2. SELECT TOP M,N：选择第 M 条记录之后的 N 条记录。

其中 M、N 都为整数(>=0)。

例 查询现价最贵的两种产品的编号和名称。

```
SELECT TOP 2 PRODUCTID,NAME FROM PRODUCTION.PRODUCT
ORDER BY NOWPRICE DESC;
```

其结果如表 4.8.1 所示：

表 4.8.1

PRODUCTID	NAME
10	噼里啪啦丛书(全 7 册)

6	长征

例 查询现价第二贵的产品的编号和名称。

```
SELECT TOP 1,1 PRODUCTID,NAME FROM PRODUCTION.PRODUCT
ORDER BY NOWPRICE DESC;
```

其结果如表 4.8.2 所示：

表 4.8.2

PRODUCTID	NAME
6	长征

4.9 LIMIT 子句

在 DM 中，可以使用限定条件对结果集做出筛选，按顺序选取结果集中某条记录开始的 N 条记录。语法如下

```
<LIMIT 子句>::=LIMIT <记录数> | <记录数>,<记录数> | <记录数> OFFSET <偏移量>
<记录数>::=<整数>
<偏移量>::=<整数>
```

共支持三种方式：

1. LIMIT N：选择前 N 条记录；
2. LIMIT M,N：选择第 M 条记录之后的 N 条记录；
3. LIMIT M OFFSET N：选择第 N 条记录之后的 M 条记录。

注意：LIMIT 不能与 TOP 同时出现在查询语句中。

例 查询前 2 条记录

```
SELECT PRODUCTID , NAME    FROM    PRODUCTION.PRODUCT    LIMIT 2;
```

其结果如表 4.9.1 所示：

表 4.9.1

PRODUCTID	NAME
1	红楼梦
2	水浒传

例 查询第 3，4 个登记的产品的编号和名称。

```
SELECT PRODUCTID, NAME    FROM    PRODUCTION.PRODUCT    LIMIT    2    OFFSET    2;
```

其结果如表 4.9.2 所示：

表 4.9.2

PRODUCTID	NAME
3	老人与海
4	射雕英雄传(全四册)

例 查询前第 5，6，7 个登记的姓名。

```
SELECT PERSONID,NAME FROM PERSON.PERSON LIMIT 4,3;
```

其结果如表 4.9.3 所示：

表 4.9.3

PERSONID	NAME
5	孙丽
6	黄非

7	王菲

4.10 全文检索

4.10.1 全文检索的使用

DM 数据库提供多文本数据检索服务，包括全文索引和全文检索。全文索引为在字符串数据中进行复杂的词搜索提供了有效支持。全文索引存储关于词和词在特定列中的位置信息，全文检索利用这些信息，可以快速搜索包含某个词或某一组词的记录。

执行全文检索涉及到以下这些任务：

1. 对需要进行全文检索的表和列进行注册；
2. 对注册了的列的数据建立全文索引；
3. 对注册了的列查询填充后的全文索引。

执行全文检索步骤如下：

1．建立全文索引；

2．修改（填充）全文索引；

3．使用带 CONTAINS 谓词的查询语句进行全文检索；

4．当数据表的全文索引列数据发生变化，则需要进行增量或者完全填充全文索引，以便可以查询到更新后的数据；

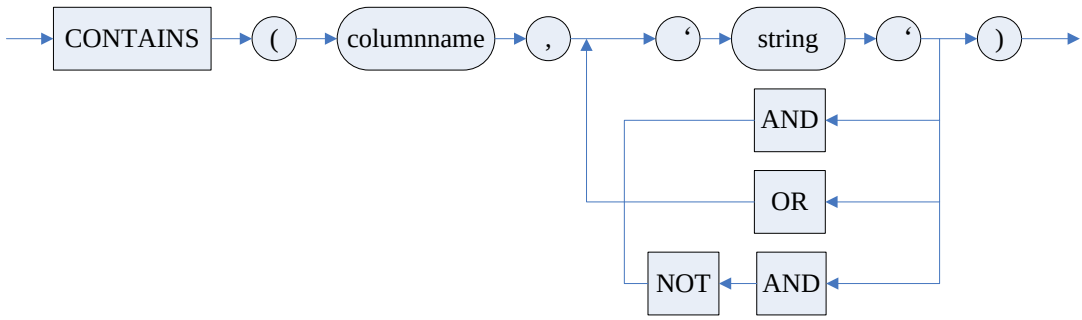
5．若不再需要全文索引，可以删除该索引；

6．在全文索引定义并填充后，才可进行全文检索。

全文检索通过在查询语句中使用 CONTAINS 子句进行。CONTAINS 子句格式说明如下：

```
CONTAINS ( <列名> , <检索条件> )
<检索条件>::= <布尔项> | <检索条件> <AND | OR | AND NOT> <布尔项>
<布尔项>::= '字符串'
```

contains_clause 的图例



当用户使用 CONTAINS 子句查询时，<列名>必须是已经建立了全文索引并填充后的列，否则，系统会报错。

有关 DM 全文检索的几点说明：

1．支持精确字、词、短语及一段文字的查询，CONTAINS 谓词内支持 AND | AND NOT | OR 的使用，AND 的优先级高于 OR 的优先级；

2．支持对每个精确词（单字节语言中没有空格或标点符号的一个或多个字符）或短语（单字节语言中由空格和可选的标点符号分隔的一个或多个连续的词）的匹配。对词或短语中字符的搜索不区分大小写；

3．对于短语或一段文字的查询，根据词库，单个查找串被分解为若干个关键词，忽略

词库中没有的词和标点符号，在索引上进行（关键词 AND 关键词）匹配查找。因而，不一定是精确查询；

4．英文查询不区分大小写和全角半角中英文字符；

5．不提供 Noise 文件，即不考虑忽略词或干扰词；

6．不支持通配符“*”；

7．不提供对模糊词或变形词的查找；

8．不支持对结果集的相关度排名；

9．检索条件子句可以和其他子句共同组成 WHERE 的检索条件。

例 全文检索综合实例，以 PRODUCT 表为例。

(1)在 DESCRIPTION 列上定义全文索引。

```
CREATE CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT(DESCRIPTION) LEXER CHINESE_VGRAM_LEXER;
```

(2)完全填充全文索引。

```
ALTER CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT REBUILD;
```

(3)进行全文检索，查找描述里有“语言”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION, '语言');
```

结果如表 4.10.1 所示。

表 4.10.1

PRODUCTID	NAME
2	水浒传
7	数据结构(C 语言版)(附光盘)

(4) 进行全文检索，查找描述里有“语言”及“中国”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'语言' AND '中国');
```

结果如表 4.10.2 所示。

表 4.10.2

PRODUCTID	NAME
2	水浒传

(5)进行全文检索，查找描述里有“语言”或“中国”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'语言' OR '中国');
```

结果如表 4.10.3 所示。

表 4.10.3

PRODUCTID	NAME
1	红楼梦
2	水浒传
7	数据结构(C 语言版)(附光盘)

(6)进行全文检索，查找描述里无“中国”字样的雇员的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE NOT CONTAINS(DESCRIPTION, '中国');
```

结果如表 4.10.4 所示。

表 4.10.4

PRODUCTID	NAME
3	老人与海

4	射雕英雄传(全四册)
5	鲁迅文集(小说、散文、杂文)全两册
6	长征
7	数据结构(C 语言版)(附光盘)
8	工作中无小事
9	突破英文基础词汇
10	噼里啪啦丛书(全 7 册)

(7)进行全文检索，查找描述里有“C 语言”字样的产品的编号和名称。

```
SELECT PRODUCTID, NAME FROM PRODUCTION.PRODUCT WHERE CONTAINS(DESCRIPTION,'C 语言');
```

结果如表 4.10.5 所示。

表 4.10.5

PRODUCTID	NAME
7	数据结构(C 语言版)(附光盘)

(8)对不再需要的全文索引进行删除。

```
DROP CONTEXT INDEX INDEX1 ON PRODUCTION.PRODUCT;
```

4.10.2 全文检索中文词库的自定义

全文检索中的词库包含：系统默认词库和用户自定义词库，前者不允许修改，后者可以修改。所以试图修改系统默认词库时，会报错。用户可以定义多个词库，每个词库的修改都必须重新加载后才起作用。

自定义词库的所有接口都通过系统函数的方式实现，具有使用系统函数权限的用户可以通过这些函数实现对应的词库修改操作。

1． 创建词库

语法格式

```
SF_WORD_LIB_CREATE(
lib_path        varchar(256)
)
```

语句功能

创建一个新的词库文件。

参数说明

lib_path：词库文件的路径，包括文件名。如果该文件不存在，会自动创建。不指定目录情况下为默认工作目录。

返回值

创建成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_CREATE('user_word_lib.lib');
```

2． 添加新词

语法格式

```
SF_WORD_LIB_ADD_WORD(
lib_path        varchar(256),
word        varchar(256)
)
```

语句功能

向自定义词库添加新词。增加的新词不允许为空串或者 NULL。

参数说明

lib_path：词库文件的路径，包括文件名；

word：新词文本。

返回值

添加成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_ADD_WORD ('user_word_lib.lib','电监会');
```

3. 删除词

语法格式

```
SF_WORD_LIB_DELETE_WORD(
lib_path        varchar(256),
word            varchar(256)
)
```

语句功能

删除一个词。

参数说明

lib_path：词库文件的路径，包括文件名；

word：待删除的词文本。

返回值

删除成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_DELETE_WORD ('user_word_lib.lib','电监会');
```

4. 修改词

语法格式

```
SF_WORD_LIB_MODIFY_WORD(
lib_path        varchar(256),
old_word        varchar(256),
new_word        varchar(256)
)
```

语句功能

修改一个词。

参数说明

lib_path：词库文件的路径，包括文件名；

old_word：原词文本；

new_word：新词文本。

返回值

修改成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_MODIFY_WORD ('user_word_lib.lib','电监会','电会');
```

5. 删除词库

语法格式

```
SF_WORD_LIB_DELETE(
lib_path        varchar(256)
)
```

语句功能

删除自定义的词库，系统默认词库不允许删除。

参数说明

lib_path：词库文件的路径，包括文件名。

返回值

删除成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_DELETE('user_word_lib.lib');
```

6. 清空词库

语法格式

```
SF_WORD_LIB_CLEAR(
lib_path        varchar(256)
)
```

语句功能

清空自定义的词库文件。

参数说明

lib_path：词库文件的路径，包括文件名。

返回值

清空成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_ CLEAR ('user_word_lib.lib');
```

7. 批量添加新词

语法格式

```
SF_WORD_LIB_BATCH_ADD_WORDS(
lib_path        varchar(256),
words_path          varchar(256)
)
```

语句功能

批量添加新词。

参数说明

lib_path：词库文件的路径，包括文件名；

words_path：新词的文件路径，包括文件名。

返回值

添加成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_BATCH_ADD_WORDS ('user_word_lib.lib','new_words.txt');
```

加入新词的文件（.txt）中，每个新词占一行，并且都为合法字符。增加的新词不允许为空串或者 NULL。如果存在重复的词，则只插入一次。具体内容格式为：

```
新词 1
新词 2
…
```

8. 批量删除词

语法格式

```
SF_WORD_LIB_BATCH_DELETE_WORDS(
lib_path        varchar(256),
words_path          varchar(256)
)
```

语句功能

批量删除词。删除词的文件格式和批量插入词的文件格式相同。如果待删除的词不存在，则不会删除。

参数说明

lib_path：词库文件的路径，包括文件名；

words_path：词的文件路径，包括文件名。

返回值

删除成功，返回 1；否则，报错。

举例说明

```
SELECT SF_WORD_LIB_BATCH_DELETE_WORDS ('user_word_lib.lib','new_words.txt');
```

举例说明

(1) 准备测试数据

```
drop table t1;
create table t1(c0 int, c1 char(200), c2 varchar(1000));
create context index cti_t1 on t1(c1)LEXER CHINESE_LEXER;
insert into t1(c1) values('达梦数据库公司');
alter context index cti_t1 on t1 rebuild;
```

查看分词结果：

```
select word from cti$cti_t1$i;
行号          WORD
--------------------------
1             公司
2             数据库
3             梦
4             达
5             达梦
```

(2) 创建一个词库文件

```
SELECT SF_WORD_LIB_CREATE('user_word_lib.lib');
```

(3) 添加一批新词

```
SELECT SF_WORD_LIB_BATCH_ADD_WORDS ('user_word_lib.lib', 'new_words.txt');
```

“new_words.txt”的文件格式为：

```
达梦
```

(4) 重启服务器后

```
alter context index cti_t1 on t1 rebuild;
```

查看分词结果：

```
select word from cti$cti_t1$i;
行号          WORD
----------------------------
1             公司
2             数据库
3             达梦
```

从结果中可以看出，“达梦”作为一个已知词被分出来。

(5) 使用全文检索查询

```
select * from t1 where contains(c1, '达梦');
```

查询结果：

```
C0                    C1                    C2
-------------------------------------------------------------
NULL          达梦数据库公司          NULL
```

4.11 层次查询

可通过 CONNECT BY 子句进行层次查询，得到数据间的层次关系。在使用 CONNECT BY 子句时，可以使用层次查询相关的伪列、函数或操作符来明确层次查询结果中的相应层次信息。

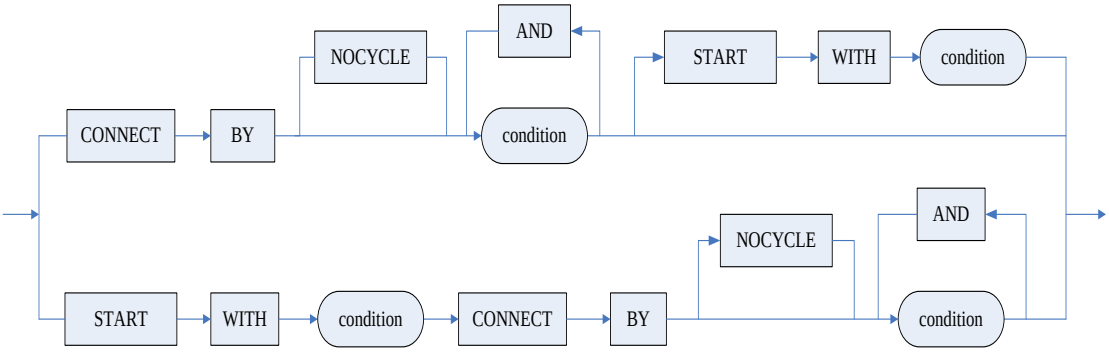
4.11.1 层次查询子句

语法格式

```
<层次查询子句> ::=
        CONNECT BY [NOCYCLE] <连接条件> [ START WITH <起始条件> ] |
```

```
START WITH <起始条件> CONNECT BY [NOCYCLE] <连接条件>
<连接条件>::=<表达式> = PRIOR <表达式> | PRIOR <表达式> = <表达式>
```

图例



参数

1．<连接条件>　逻辑表达式，指明层次数据间的层次连接关系。在其中必须用操作符 PRIOR 指明父节点的元组；

2．<起始条件>　逻辑表达式，指明选择层次数据根数据的条件；

3.　NOCYCLE 关键字用于指定数据导致环的处理方式，如果在 CONNECT BY 子句中指定 NOCYCLE 关键字，会忽略导致环元组的的儿子数据。否则，返回错误。

4.11.2 层次查询相关伪列

在使用层次查询子句时，可以通过相关的伪列来明确数据的层次信息。层次查询相关的伪列有：

1．LEVEL

该伪列表示当前元组在层次数据形成的树结构中的层数。LEVEL 的初始值为 1，即层次数据的根节点数据的 LEVEL 值为 1，之后其子孙节点的 LEVEL 依次递增。

2．CONNECT_BY_ISLEAF

该伪列表示当前元组在层次数据形成的树结构中是否是叶节点(即该元组根据连接条件不存在子结点)。是叶节点时为 1，否则为 0。

3．CONNECT_BY_ISCYCLE

该伪列表示当前元组是否会将层次数据形成环，该伪列只有在层次查询子句中表明 NOCYCLE 关键字时才有意义。如果元组的存在会导致层次数据形成环，该伪列值为 1，否则为 0。

4.11.3 层次查询相关操作符

1．PRIOR

PRIOR 操作符主要使用在 CONNECT BY 子句中，指明逻辑表达式中的父节点。

2．CONNECT_BY_ROOT

该操作符作为查询项，查询在层次查询结果中根节点的某列的值。

4.11.4 层次查询相关函数

语法格式

```
SYS_CONNECT_BY_PATH(col_name,char)
```

语句功能

层次查询。

使用说明

该函数得到从根节点到当前节点路径上所有节点名为 col_name 的某列的值，之间用 char 指明的字符分隔开。

4.11.5 层次查询层内排序

语法格式

```
<ORDER BY 子句> ::= order siblings by
<无符号整数> | <列说明> | <值表达式> [ASC | DESC]
{,<无符号整数> | <列说明> | <值表达式> [ASC | DESC]}
```

语句功能

层次查询。

使用说明

在层次查询中使用 order siblings by，可用于指定层次查询中相同层次数据返回的顺序。

4.11.6 层次查询的限制

1. START WITH 子句中不能使用层次查询的所有伪列、层次查询函数、操作符以及 ROWNUM；
2. ORDER SIBLINGS BY 子句中不能使用层次查询的所有伪列、层次查询函数、操作符、ROWNUM 以及子查询；
3. CONNECT BY 子句不能使用伪列 CONNECT_BY_ISLEAF 、CONNECT_BY_ISCYCLE 和 SYS_CONNECT_BY_PATH 伪函数，但必须包含 PRIOR 表达式；
4. PRIOR、CONNECT_BY_ROOT 操作符后以及 SYS_CONNECT_BY_PATH 第一个参数不能使用层次查询的所有伪列、层次查询函数、操作符、ROWNUM 以及子查询。SYS_CONNECT_BY_PATH 第二个参数必须是常量字符串；
5. 在多表连接的层次查询中，WHERE 条件中不能包含 OR；
6. 函数 SYS_CONNECT_BY_PATH 的最大返回长度为 8188，超长就会报错。函数 SYS_CONNECT_BY_PATH 在一个查询语句中最多使用个数为 64；
7. INI 参数 CNNTB_MAX_LEVEL 表示支持层次查询的最大层次，默认为 20000。该参数的有效取值为[1, 100000]。

例

对 OTHER.DEPARTMENT 数据进行层次查询，HIGH_DEP 表示上级部门；DEP_NAME 表示部门名称。

层次数据所建立起来的树形结构如下图：

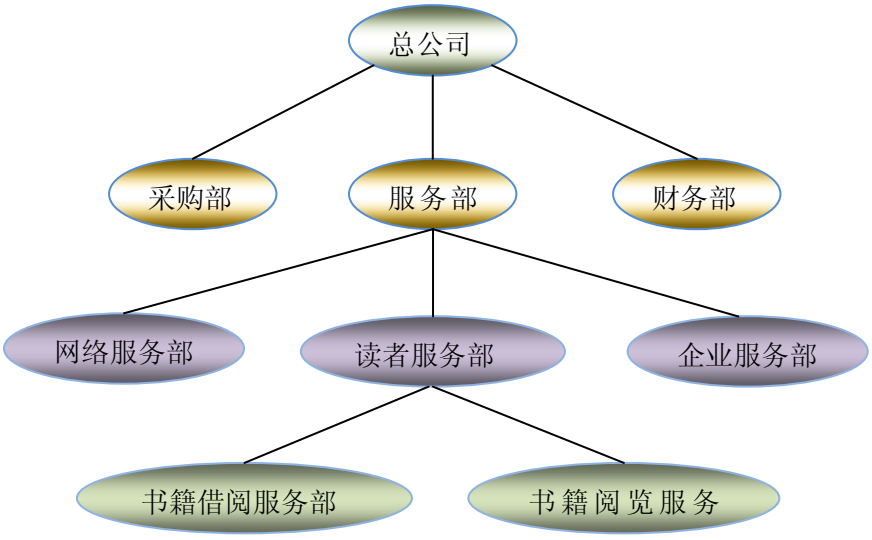


图 4.11.1 层次数据树形结构图

1．不带起始选择根节点起始条件的层次查询

```
SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME = HIGH_DEP;
```

查询结果如表 4.11.1 所示。

表 4.11.1

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	总公司	服务部
3	服务部	网络服务部
4	服务部	读者服务部
5	读者服务部	书籍借阅服务部
6	读者服务部	书籍阅览服务部
7	服务部	企业服务部
8	总公司	采购部
9	总公司	财务部
10	总公司	财务部
11	总公司	采购部
12	总公司	服务部
13	服务部	网络服务部
14	服务部	读者服务部
15	读者服务部	书籍借阅服务部
16	读者服务部	书籍阅览服务部
17	服务部	企业服务部
18	服务部	企业服务部
19	服务部	读者服务部
20	读者服务部	书籍借阅服务部
21	读者服务部	书籍阅览服务部
22	服务部	网络服务部
23	读者服务部	书籍阅览服务部
24	读者服务部	书籍借阅服务部

结果是以表中所有的节点为根节点进行先根遍历进行层次查询。

2．带起始选择根节点起始条件的层次查询

```
SELECT HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司';
```

查询结果如表 4.11.2 所示。

表 4.11.2

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	总公司	服务部
3	服务部	网络服务部
4	服务部	读者服务部
5	读者服务部	书籍借阅服务部
6	读者服务部	书籍阅览服务部
7	服务部	企业服务部
8	总公司	采购部
9	总公司	财务部

3．层次查询伪列的使用

在层次查询中,伪列的使用可以更明确层次数据之间的关系。

```
SELECT LEVEL,
CONNECT_BY_ISLEAF ISLEAF ,
CONNECT_BY_ISCYCLE ISCYCLE,
HIGH_DEP, DEP_NAME FROM OTHER.DEPARTMENT
CONNECT BY PRIOR DEP_NAME=HIGH_DEP
START WITH DEP_NAME='总公司';
```

查询结果如表 4.11.3 所示。

表 4.11.3

序号	LEVEL	ISLEAF	ISCYCLE	HIGH_DEP	DEP_NAME
1	1	0	0	NULL	总公司
2	2	0	0	总公司	服务部
3	3	1	0	服务部	网络服务部
4	3	0	0	服务部	读者服务部
5	4	1	0	读者服务部	书籍借阅服务部
6	4	1	0	读者服务部	书籍阅览服务部
7	3	1	0	服务部	企业服务部
8	2	1	0	总公司	采购部
9	2	1	0	总公司	财务部

通过伪列，可以清楚的看到层次数据之间的层次结构。

4．含有过滤条件的层次查询

在层次查询中加入过滤条件，将会先进行层次查询，然后进行过滤。

```
SELECT LEVEL,* FROM OTHER.DEPARTMENT WHERE HIGH_DEP = '总公司' CONNECT BY PRIOR DEP_NAME=HIGH_DEP;
```

查询结果如表 4.11.4 所示。

表 4.11.4

序号	LEVEL	HIGH_DEP	DEP_NAME
1	2	总公司	服务部
2	2	总公司	采购部
3	2	总公司	财务部
4	1	总公司	财务部
5	1	总公司	采购部
6	1	总公司	服务部

5．含有排序子句的层次查询

在层次查询中加入排序，查询将会按照排序子句指明的要求排序，不再按照层次查询的排序顺序排序。

```
SELECT * FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ORDER BY HIGH_DEP;
```

查询结果如表 4.11.5 所示。

表 4.11.5

序号	HIGH_DEP	DEP_NAME
1	NULL	总公司
2	读者服务部	书籍阅览服务部
3	读者服务部	书籍借阅服务部
4	服务部	读者服务部
5	服务部	企业服务部
6	服务部	网络服务部
7	总公司	财务部
8	总公司	采购部
9	总公司	服务部

6. 含层内排序子句的层次查询

在层次查询中加入 ORDER SIBLINGS BY，查询会对相同层次的数据进行排序后，深度优先探索返回数据，即 LEVEL 相同的数据进行排序。

```
SELECT HIGH_DEP, DEP_NAME, LEVEL FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司' ORDER SIBLINGS BY DEP_NAME;
```

查询结果如表 4.11.6 所示。

表 4.11.6

序号	HIGH_DEP	DEP_NAME	LEVEL
1	NULL	总公司	1
2	总公司	财务部	2
3	总公司	采购部	2
4	总公司	服务部	2
5	服务部	读者服务部	3
6	读者服务部	书籍借阅服务部	4
7	读者服务部	书籍阅览服务部	4
8	服务部	企业服务部	3

9	服务部	网络服务部	3

7．CONNECT_BY_ROOT 操作符的使用

CONNECT_BY_ROOT 操作符之后跟某列的列名，例如：

```
CONNECT_BY_ROOT DEP_NAME
```

进行如下查询

```
SELECT CONNECT_BY_ROOT DEP_NAME,* FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司';
```

查询结果如表 4.11.7 所示。

表 4.11.7

序号	CONNECT_BY_ROOT DEP_NAME	HIGH_DEP	DEP_NAME
1	总公司	NULL	总公司
2	总公司	总公司	服务部
3	总公司	服务部	网络服务部
4	总公司	服务部	读者服务部
5	总公司	读者服务部	书籍借阅服务部
6	总公司	读者服务部	书籍阅览服务部
7	总公司	服务部	企业服务部
8	总公司	总公司	采购部
9	总公司	总公司	财务部

8．SYS_CONNECT_BY_PATH 函数的使用

函数的使用方式，如：

```
SYS_CONNECT_BY_PATH(DEP_NAME, '/')
```

进行如下查询：

```
SELECT SYS_CONNECT_BY_PATH(DEP_NAME, '/') PATH,* FROM OTHER.DEPARTMENT CONNECT BY PRIOR DEP_NAME=HIGH_DEP START WITH DEP_NAME='总公司';
```

查询结果如表 4.11.8 所示。

表 4.11.8

序号	PATH	HIGH_DEP	DEP_NAME
1	/总公司	NULL	总公司
2	/总公司/服务部	总公司	服务部
3	/总公司/服务部/网络服务部	服务部	网络服务部
4	/总公司/服务部/读者服务部	服务部	读者服务部
5	/总公司/服务部/读者服务部/书籍借阅服务部	读者服务部	书籍借阅服务部
6	/总公司/服务部/读者服务部/书籍阅览服务部	读者服务部	书籍阅览服务部
7	/总公司/服务部/企业服务部	服务部	企业服务部
8	/总公司/采购部	总公司	采购部
9	/总公司/财务部	总公司	财务部

4.12 并行查询

达梦支持并行查询技术。首先，设置好如下 3 个 ini 参数；其次，执行 sql 语句。即可

执行并行查询。用到的三个 ini 参数解释如下表。

表 4.12.1 并行查询相关参数

参数名	缺省值	说明
MAX_PARALLEL_DEGREE	1	用来设置默认并行任务个数。取值范围：1~128。缺省值 1，表示无并行任务。全局有效。
PARALLEL_POLICY	0	用来设置并行策略。取值范围：0、1 和 2，缺省为 0。其中，0 表示不支持并行；1 表示自动配置并行工作线程个数（与物理 CPU 核数相同）；2 表示手动设置并行工作线程数。
PARALLEL_THRD_NUM	10	用来设置并行工作线程个数。取值范围：1~1024。仅当 PARALLEL_POLICY 值为 2 时才启用此参数。

注：当处于 PL/SQL 调试状态时，并行查询的相关设置均无效。

其中，并行任务数，也可以在 SQL 语句中使用“PARALLEL”关键字特别指定。如果单条查询语句没有特别指定，则依然使用默认并行任务个数。 “PARALLEL”关键字的用法是在数据查询语句的 SELECT 关键字后，增加 HINT 子句来实现。

语法格式如下：

```
/*+ PARALLEL(表名 并行任务个数) */
```

对于无特殊要求的并行查询用户，可以使用默认并行任务数 MAX_PARALLEL_DEGREE。只需要在 ini 参数中设置好如下 3 个参数，然后执行 SQL 查询语句，就可以启用并行查询。

例如，可以将 3 个参数设置为如下：

```
MAX_PARALLEL_DEGREE 3
PARALLEL_POLICY 2
PARALLEL_THRD_NUM 4
```

或者将 PARALLEL_POLICY 设置为 1，此时，只要设置 2 个参数就可以：

```
MAX_PARALLEL_DEGREE 3
PARALLEL_POLICY 1
```

或者将 PARALLEL_POLICY 设置为 0，此时，不支持并行查询：

```
MAX_PARALLEL_DEGREE 3
PARALLEL_POLICY 0
```

然后，执行语法格式类似“SELECT * FROM SYSOBJECTS;”的 SQL 语句即可，本条语句使用默认并行任务数 3。

当然，如果单条查询语句不想使用默认并行任务数，可以通过在 SQL 语句中增加 HINT，通过“PARALLEL”关键字来特别指定。本条语句使用特别指定的并行任务数 4，例如：

```
SELECT /*+ PARALLEL(SYSOBJECTS 4) */ * FROM SYSOBJECTS;
```

4.13 ROWNUM

ROWNUM 是一个虚假的列，表示从表中查询的行号，或者连接查询的结果集行数。它将被分配为 1，2，3，4，...N，N 是行的数量。通过使用 ROWNUM 可以限制查询返回的行数。例如，以下语句执行只会返回前 5 行数据。

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE rownum < 6;
```

一个 ROWNUM 值不是被永久的分配给一行。表中的某一行并没有标号，不可以查询 ROWNUM 值为 5 的行。ROWNUM 值只有当被分配之后才会增长，并且初始值为 1。即只有满足一行后，ROWNUM 值才会加 1，否则只会维持原值不变。因此，以下语句在任何时候都不能返回数据。

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM > 11;
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM = 5;
```

ROWNUM 一个重要作用是控制返回结果集的规模，可以避免查询在磁盘中排序。因为，ROWNUM 值的分配是在查询的谓词解析之后，任何排序和聚合之前进行的。因此，在排序和聚合使用 ROWNUM 时需要注意，可能得到并非预期的结果，例如：

```
SELECT * FROM RESOURCES.EMPLOYEE WHERE ROWNUM < 11 ORDER BY NAME;
```

以上语句只会对 EMPLOYEE 表前 10 行数据按 NAME 排序输出，并不是表的所有数据按 NAME 排序后输出前 10 行，要实现后者，需要使用如下语句：

```
SELECT * FROM (SELECT * FROM RESOURCES.EMPLOYEE ORDER BY NAME) WHERE ROWNUM < 11;
SELECT TOP 10 * FROM RESOURCES.EMPLOYEE ORDER BY NAME;
```

ROWNUM 的限制：

1．在查询中，ROWNUM 可与任何数字类型表达式进行比较及运算，但不能出现在含 OR 的布尔表达式中，否则报错处理；

2．ROWNUM 只能在非相关子查询中出现，不能在相关子查询中使用，否则报错处理；

3．在非相关子查询中，ROWNUM 只能实现与 TOP 相同的功能，因此子查询不能含 ORDER BY 和 GROUP BY；

4．ROWNUM 所处的子谓词只能为如下形式： ROWNUM op exp，exp 的类型只能是立即数、参数和变量值，op ∈ {<, <=, >, >=, =, <>}。

4.14 数组查询

在 DM 中，可以通过查询语句查询数组信息。语法如下：

```
FROM ARRAY <数组>
```

目前 DM 只支持一维数组的查询。

数组类型可以是记录类型和普通数据库类型。记录类型数组查询出来的列名为记录类型每一个属性的名字。普通数据库类型查询出来的列名均为”C”。

例 1 查看数组

```
SELECT * FROM ARRAY NEW INT[2]{1};
```

可以看到返回结果:

```
C
1
NULL
```

例 2 数组与表的连接

```
DECLARE
    TYPE rrr IS RECORD (x INT, y INT);
    TYPE ccc IS ARRAY rrr[];
    c ccc;
BEGIN
    c = NEW rrr[2];
    FOR i IN 1..2 LOOP
        c[i].x = i;
        c[i].y = i*2;
    END LOOP;
    SELECT arr.x, o.name    FROM ARRAY c arr, SYSOBJECTS o WHERE arr.x = o.id;
END;
```

可以看到返回结果:

```
X                    NAME
1                    SYSINDEXES
2                    SYSCOLUMNS
```

4.15 查看执行计划与执行跟踪统计

在 DISQL 命令行执行 EXPLAIN 命令可以查看 DML 语句的执行计划。

语法说明：

```
EXPLAIN <SQL 语句>;
<SQL 语句> ::= <删除语句> | <插入语句> | <查询语句> | <更新语句>
```

参数

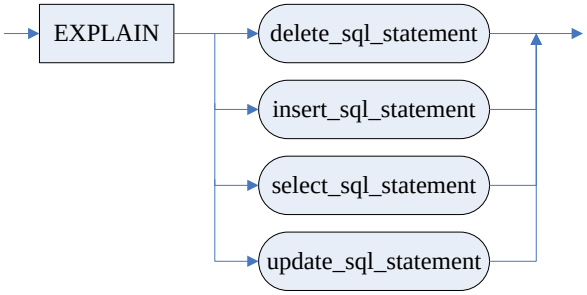
1．<删除语句>　指数据删除语句；

2．<插入语句>　指数据插入语句；

3．<查询语句>　指查询语句；

4．<更新语句>　指数据更新语句。

图例



语句功能

供用户查看执行计划。

举例说明：

例 显示如下语句的查询计划：

```
EXPLAIN SELECT NAME,schid
FROM SYSOBJECTS
WHERE SUBTYPE$='STAB' AND NAME
    NOT IN (
SELECT NAME FROM SYSOBJECTS WHERE NAME IN (SELECT NAME FROM SYSOBJECTS WHERE SUBTYPE$='STAB') AND TYPE$='DSYNOM')
```

可以看到返回结果：

```
1    #NSET2: [2, 1,142]
2      #PRJT2: [2, 1, 142]; exp_num(2), is_atom(FALSE)
3        #HASH LEFT SEMI JOIN2: [2, 1,142];  (ANTI),
4          #SLCT2: [0, 20, 142]; SYSOBJECTS.SUBTYPE$ = STAB
5            #CSCN2: [0, 807, 142]; SYSINDEXSYSOBJECTS(SYSOBJECTS)
6          #PRJT2: [1, 1, 138]; exp_num(1), is_atom(FALSE)
7            #HASH LEFT SEMI JOIN2: [1, 1, 138]
8                                    #CSEK2:  [0,  20,  138];  scan_type(ASC),
SYSINDEXSYSOBJECTS(SYSOBJECTS),
scan_range[(DSYNOM,min,min),(DSYNOM,max,max))
9                #SLCT2: [0, 20, 138]; SYSOBJECTS.SUBTYPE$ = STAB
10                 #CSCN2: [0, 807, 138]; SYSINDEXSYSOBJECTS(SYSOBJECTS)
```

4.16 查看当前会话时区信息

可通过查询语句查看当前会话时区信息

例 查看查看当前会话时区信息：

```
SELECT TIME_ZONE FROM V$SESSIONs WHERE SESS_ID=( SELECT cast(SESSID AS binary(8)));
```

可以看到返回结果:

```
+8:00
```

第 5 章 数据的插入、删除和修改

DM_SQL 语言的数据更新语句包括：数据插入、数据修改和数据删除三种语句，其中数据插入和修改二种语句使用的格式要求比较严格。在使用时要求对相应基表的定义，如列的个数、各列的排列顺序、数据类型及关键约束、唯一性约束、引用约束、检查约束的内容均要了解得很清楚，否则就很容易出错。下面将分别对这三种语句进行讨论。在讨论中，如不特别说明，各例均使用示例库 BOOKSHOP，用户均为建表者 SYSDBA。

5.1 数据插入语句

数据插入语句用于往已定义好的表中插入单个或成批的数据。

INSERT 语句有二种形式。一种形式是值插入，即构造一行或者多行，并将它们插入到表中；另一种形式为查询插入，它通过返回一个查询结果集以构造要插入表的一行或多行。

数据插入语句的语法格式如下：

语法格式

```
[@] INSERT [INTO]   <表引用>   [(<列名>{,<列名>})]
VALUES(<插入值>{,<插入值>});|(<插入值>{,<插入值>}){, (<插入值>{,<插入值>})};|
<查询说明>[<ORDER BY  表达式>];|
DEFAULT VALUES [RETURN <列名>{,<列名>} INTO <结果对象>,{<结果对象>}];
<结果对象>::<数组>|<变量>
<表引用>::= [<模式名>.]<基表或视图名>
<基表或视图名>::= <基表名>|<视图名>
```

参数

1．<模式名>　　指明该表或视图所属的模式，缺省为当前模式；

2．<基表名>　　指明被插入数据的基表的名称；

3．<视图名>　　指明被插入数据的视图的名称，实际上 DM 将数据插入到视图引用的基表中；

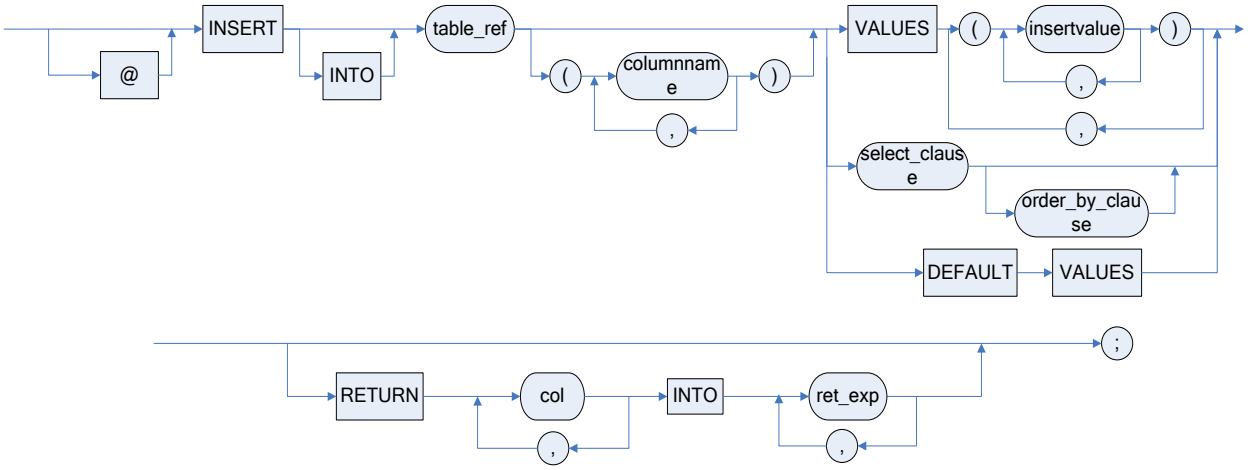
4．<列名>　　表或视图的列的名称。在插入的记录中，这个列表中的每一列都被 VALUES 子句或查询说明赋一个值。如果在此列表中省略了表的一个列名，则 DM 用先前定义好的缺省值插入到这一列中。如果此列表被省略，则在 VALUES 子句和查询中必须为表中的所有列指定值；

5．<插入值>　　指明在列表中对应的列的插入的列值，如果列表被省略了，插入的列值按照基表中列的定义顺序排列；

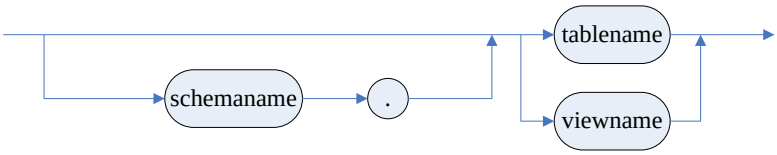
6．<查询说明>　将一个 SELECT 语句所返回的记录插入表或视图的基表中，子查询中选择的列表必须和 INSERT 语句中列名清单中的列具有相同的数量。查询说明必须遵照的语法规则请参照 SELECT 语句的有关说明；

7. @　当插入的是大数据数据文件时，启用@。同时对应的<插入值>格式为：@'path'。比如：@insert into t1 values(@'e: \DSC_1663.jpg');

图例



table_ref



使用说明

1．<基表名>或<视图名>后所跟的<列名>必须是该表中的列，且同一列名不允许出现两次，但排列顺序可以与定义时的顺序不一致；

2．<插入值>的个数、类型和顺序要与<列名>一一对应；

3．如果某一<列名>未在 INTO 子句后面出现，则新插入的行在这些列上将取空值或缺省值，如该列在基表定义时说明为 NOT NULL 时将会出错；

4．如果<基表名>或<视图名>后没指定任何<列名>，则隐含指定该表或视图的所有列，这时，新插入的行必须在每个列上均有<插入值>；

5．如果两表之间存在引用和被引用关系时，应先插入被引用表的数据，再插入引用表的数据；

6．<查询说明>是指用查询语句得到的一个结果集插入到插入语句中<表名>指定的表中，因此该格式的使用可供一次插入多个行，但插入时要求结果集的列与目标表要插入的列是一一对应的，不然会报错；

7．插入在指定值的时候，可以同时指定多行值，这种叫做多行插入或者批量插入。最多可以同时指定 65534 行数据；对于存在行触发器的表，多行插入时每一行都会触发相关的触发器；同样如果目标表具有约束，那么每一行都会进行相应的约束检查，只要有一行不满足约束，所有的值都不能插入成功；

8．在嵌入方式下工作时，<插入值>可以为主变量；

9．如果插入对象是视图，同时在这个视图上建立了 INSTEAD OF 触发器，则会将插入操作转换为触发器所定义的操作；如果没有触发器，则需要判断这个视图是否可更新，如果不可更新则报错，否则是可以插入成功的。

10. RETURN INTO 返回列支持返回 ROWID。

11. RETURN INTO 语句中返回结果对象支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1，且记录数组属性类型与个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数和数组元素类型与返回列一致；返回结果不支持变量、普通数组和记录数组混和使用。

举例说明

例 在 VENDOR 表中插入一条供应商信息：帐户号为 00，名称为华中科技大学出版社，活动标志为 1，URL 为空，信誉为 2。

```
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO, NAME, ACTIVEFLAG, WEBURL, CREDIT)
VALUES ('00', '华中科技大学出版社', 1, '', 2);
```

如果需要同时多行插入，则可以用如下的 SQL 语句实现：

```
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO, NAME, ACTIVEFLAG, WEBURL, CREDIT)
VALUES ('00', '华中科技大学出版社', 1, '', 2), ('00', '清华大学出版社', 1, '', 3);
```

在定义 VENDOR 表时，设定了检查约束：CHECK(CREDIT IN(1,2,3,4,5))，说明 CREDIT 只能是 1,2,3,4,5。在插入新数据或修改供应商的 CREDIT 时，系统则按检查约束进行检查，如果不满足条件，系统将会报错，多行插入中，每一行都会做检查。

由于 DM 支持标量子查询，标量子查询允许用在标量值合法的地方，因此在数据插入语句的<插入值>位置允许出现标量子查询。

例 将书名为长征的图书的出版社插入到 VENDOR 表中。

```
INSERT INTO PURCHASING.VENDOR(ACCOUNTNO, NAME, ACTIVEFLAG, WEBURL, CREDIT)
VALUES('00',
(SELECT PUBLISHER FROM PRODUCTION.PRODUCT WHERE NAME ='长征'),1, '', 1);
```

若是需要插入一批数据时，可使用带<查询说明>的插入语句，如下例所示。

例 构造一个新的基表，表名为 PRODUCT_SELL，用来显示出售的商品名称和购买用户名称，并将查询的数据插入此表中。

```
CREATE TABLE PRODUCTION.PRODUCT_SELL
( PRODUCTNAME VARCHAR(50) NOT NULL,
  CUSTOMERNAME VARCHAR(50) NOT NULL);
INSERT INTO PRODUCTION.PRODUCT_SELL
SELECT DISTINCT T1.NAME , T5.NAME
FROM PRODUCTION.PRODUCT T1, SALES.SALESORDER_DETAIL T2,
          SALES.SALEsORDER_HEADER T3, SALES.CUSTOMER T4,
          PERSON.PERSON T5
WHERE T1.PRODUCTID = T2.PRODUCTID and T2.SALESORDERID = T3.SALESORDERID
           AND T3.CUSTOMERID = T4.CUSTOMERID AND T4.PERSONID = T5.PERSONID;
```

该插入语句将已销售的商品名称和购买该商品的用户名称插入到新建的 PRODUCT_SELL 表中。查询结果如下表 5.1.1 所示。

表 5.1.1

PRODUCTNAME	CUSTOMERNAME
红楼梦	刘青
老人与海	刘青

值得一提的是，BIT 数据类型值的插入与其他数据类型值的插入略有不同：其他数据类型皆为插入的是什么就是什么，而 BIT 类型取值只能为 1/0，又同时能与整数、精确数值类型、不精确数值类型和字符串类型相容(可以使用这些数据类型对 BIT 类型进行赋值和比较)，取值时有一定的规则。

数值类型常量向 BIT 类型插入的规则是：非 0 数值转换为 1，数值 0 转换为 0。例如：

```
CREATE TABLE T10 (C BIT);
INSERT INTO T10 VALUES(1);        --插入 1
INSERT INTO T10 VALUES(0);        --插入 0
INSERT INTO T10 VALUES(1.2);      --插入 1
```

字符串类型常量向 BIT 类型插入的规则是：

全部由 0 组成的字符串转换为 0，其他全数字字符串(例如 123)，转换为 1。非全数字字符串(例如：1e1, 2a5, 3.14)也转换为 1。

```
INSERT INTO T10 VALUES(000);      --插入 0
INSERT INTO T10 VALUES(0);        --插入 0
```

```
INSERT INTO T10 VALUES(10);        --插入 1
INSERT INTO T10 VALUES(1.0);       --插入 1
```

5.2 数据修改语句

数据修改语句用于修改表中已存在的数据。

语法格式

```
UPDATE <表引用>{<单列修改子句>|<多列修改子句>}
<单列修改子句>::= SET<列名>=<<值表达式>|DEFAULT>{,<列名>=<<值表达式>|DEFAULT>}[FROM <表引用>{,<表引用>}][WHERE <条件表达式>][RETURN <列名>{,<列名>} INTO <结果对象>];
<多列修改子句>::= SET <列名>{,<列名>}= <subquery>;
<表引用>::= [ <模式名>.]<基表或视图名>
<基表或视图名>::= <基表名>|<视图名>
<结果对象>::= <数组>|<变量>
```

参数

1．<模式名>　　指明该表或视图所属的模式，缺省为当前用户的缺省模式；

2．<基表名>　　指明被修改数据的基表的名称；

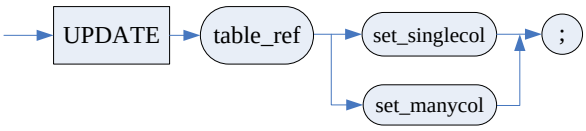
3．<视图名>　　指明被修改数据的视图的名称，实际上 DM 对视图的基表更新数据；

4．<列名>　　表或视图中被更新列的名称，如果 SET 子句中省略列的名称，列的值保持不变；

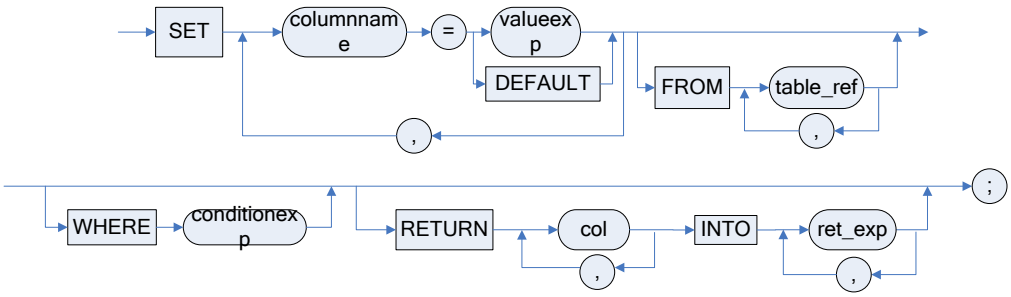
5．<值表达式>　指明赋予相应列的新值；

6．<条件表达式>指明限制被更新的行必须符合指定的条件，如果省略此子句，则修改表或视图中所有的行。

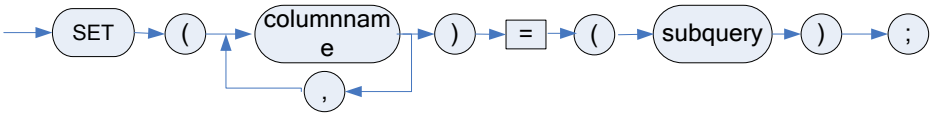
图例



set_singlecol



set_manycol



使用说明

1．SET 后的<列名>不能重复出现；

2．WHERE 子句也可以包含子查询。如果省略了 WHERE 子句，则表示要修改所有的元组；

3．如果<列名>为被引用列，只有被引用列中未被引用列引用的数据才能被修改；如果<列名>为引用列，引用列的数据被修改后也必须满足引用完整性。在 DM 系统中，以上引用完整性由系统自动检查；

4．执行基表的 UPDATE 语句触发任何与之相联系的 UPDATE 触发器；

5．如果视图的定义查询中含有以下结构则不能更新视图：

1) 联结运算；
2) 集合运算符；
3) GROUP BY 子句；
4) 集函数；
5) INTO 语句；
6) 分析函数；
7) HAVING 语；
8) CONNECT BY 语句。

6．如果更新对象是视图，同时在这个视图上建立了 INSTEAD OF 触发器，则会将更新操作转换为触发器所定义的操作；如果没有触发器，则需要判断这个视图是否可更新，如果不可更新则报错，否则可以继续更新，如果上面的条件都满足，则可以更新成功；

7. RETURN INTO 不支持返回 ROWID 列。

8. RETURN INTO 语句中返回列如果是更新列，则返回值为列的新值。返回结果对象支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1.且记录数组属性类型和个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数与数组元素类型与返回列一致;返回结果不支持变量、普通数组和记录数组混和使用。

9. UPDATE 语句支持一次进行多列修改，多列修改存在以下限制：

1) 仅支持非相关子查询；
2) 集合操作情况（UNION 等）:只有当查询语句为非相关子查询才支持集合操作；
3) 多列修改不支持 explain 操作；
4) 要 UPDATE 的表要有且只能有 1 行数据。子查询的结果不能多于 1 行数据。

举例说明

例 将出版社为中华书局的图书的现在销售价格增加 1 元。

```
UPDATE    PRODUCTION.PRODUCT    SET    NOWPRICE = NOWPRICE + 1.0000
WHERE    PUBLISHER = '中华书局';
```

例 由于标量子查询允许用在标量值合法的地方，因此在数据修改语句的<值表达式>位置也允许出现标量子查询。下例将折扣高于 7.0 且出版社不是中华书局的图书的折扣设成出版社为中华书局的图书的平均折扣。

```
UPDATE    PRODUCTION.PRODUCT    SET    DISCOUNT =
( SELECT    AVG(DISCOUNT)
      FROM    PRODUCTION.PRODUCT
WHERE PUBLISHER = '中华书局')
WHERE    DISCOUNT > 7.0    AND    PUBLISHER != '中华书局';
```

注：自增列的修改例外，它一经插入，只要该列存储于数据库中，其值为该列的标识，不允许修改。关于自增列修改的具体情况，请参见 5.6 节——自增列的使用。

例 带 RETRUN INTO 的更新语句。

```
DECLARE
   TYPE rrr IS RECORD(x INT, y INT);
   TYPE ccc IS ARRAY rrr[];
   a INT;
   c ccc;
BEGIN
   c = NEW rrr[2];
   UPDATE t1 SET c2=4 WHERE c3 = 2 RETURN c1 INTO a;
```

```
    PRINT a;
    UPDATE t1 SET c2=5 WHERE c3 = 2 RETURN c1,c2 INTO c;
    SELECT * FROM ARRAY c;
END;
```

例 使用一次进行多列修改的更新语句。

```
UPDATE PURCHASING.PURCHASEORDER_HEADER SET(TAX,FREIGHT)=(select ORIGINALPRICE, NOWPRICE from PRODUCTION.PRODUCT where NAME='长征');
```

5.3 数据删除语句

数据删除语句用于删除表中已存在的数据。

语法格式

```
DELETE [FROM] <表引用>
[WHERE <条件表达式>][RETURN <列名>{,<列名>} INTO <结果对象>,{<结果对象>}];;
<表引用>::= [<模式名>.]<基表或视图名>
<基表或视图名>::= <基表名>|<视图名>
<结果对象>::<数组>|<变量>
```

参数

1．<模式名> 指明该表或视图所属的模式，缺省为当前模式；

2．<基表名> 指明被删除数据的基表的名称；

3．<视图名> 指明被删除数据的视图的名称，实际上 DM 将从视图的基表中删除数据；

4．<条件表达式> 指明基表或视图的基表中被删除的记录须满足的条件。

使用说明

1．如果不带 WHERE 子句，表示删除表中全部元组，但表的定义仍在字典中。因此，DELETE 语句删除的是表中的数据，并未删除表结构；

2．由于 DELETE 语句一次只能对一个表进行删除，因此当两个表存在引用与被引用关系时，要先删除引用表里的记录，只有引用表中无记录时，才能删被引用表中的记录，否则系统会报错；

3．执行与表相关的 DELETE 语句将触发所有定义在表上的 DELETE 触发器；

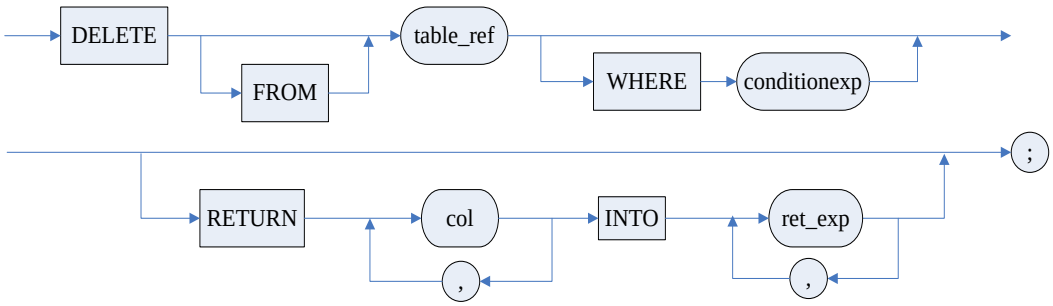
4．如果视图的定义查询中包含以下结构之一，就不能从视图中删除记录：

1) 联结运算；
2) 集合运算符；
3) GROUP BY 子句；
4) 集函数；
5) INTO 语句；
6) 分析函数；
7) HAVING 语；
8) CONNECT BY 语句。

5. RETURN INTO 不支持返回 ROWID 列。

6. RETURN INTO 返回结果对象支持变量和数组。如果返回列为记录数组，则返回结果数只能为 1.且记录数组属性类型和个数须与返回列一致；如果为变量，则变量类型与个数与返回列一致；如果返回普通数组，则数组个数与数组元素类型与返回列一致；返回结果不支持变量、普通数组和记录数组混和使用。

图例



例 将没有分配部门的员工的住址信息删除。

```
DELETE FROM RESOURCES.EMPLOYEE_ADDRESS
WHERE EMPLOYEEID IN
( SELECT EMPLOYEEID
FROM RESOURCES.EMPLOYEE
WHERE EMPLOYEEID NOT IN
( SELECT EMPLOYEEID FROM RESOURCES.EMPLOYEE_DEPARTMENT));
```

由于 DELETE 语句也是一次只能对一个表进行删除，因此当两个表存在引用与被引用关系时，要先删除引用表里的记录，只有引用表中无此记录时，才能删除被引用表中的记录，否则系统会报错。

5.4 MERGE INTO 语句

使用 MERGE INTO 语法可合并 UPDATE 和 INSERT 语句。通过 MERGE 语句，根据一张表（或视图）的连接条件对另外一张表（或视图）进行查询，连接条件匹配上的进行 UPDATE（可能含有 DELETE），无法匹配的执行 INSERT。其中，数据表包括：普通表、分区表、加密表、压缩表和 LIST 表。

语法格式

```
MERGE INTO <表引用> [<表别名>] USING <from 表引用>ON (<条件判断表达式>)
| <[<merge_update_clause >] [< merge_insert_clause>] >
< merge_update_clause >::=WHEN MATCHED THEN UPDATE SET <set_value_list><where_clause1>
<merge_insert_clause>::=WHEN NOT MATCHED THEN INSERT<column_list_null>VALUES
(<ins_value_list >) <where_clause2>;

<表引用>::= [<模式名>.]<基表或视图名>
<from 表引用>::=<普通表> | <连接表>
<普通表>::=<普通表 1> | <普通表 2> | <普通表 3>
<普通表 1>::= <对象名> [AS] [别名]
<普通表 2>::= (<不带 INTO 查询表达式>) [[AS] <表别名> [<新生列>]]
<普通表 3>::= [<模式名>.]<基表名> | <视图名> (<选择列>) [[AS] <表别名> [<派生列表>]]
<对象名>::=<本地对象> | <索引>
<本地对象>::=[<模式名>.]<基表名> | <视图名>
<索引>::=[<模式名>.]<基表名> INDEX <索引名>
<选择列>::=[ <列名> { , <列名> }]
<派生列表> ::=( [ <列名> { , <列名> }])

<set_value_list> ::= <列名>=<值表达式| DEFAULT> {,<列名>=<值表达式| DEFAULT>}
<where_clause_null > ::= [WHERE <条件表达式>][DELETE] [WHERE <条件表达式>]

<column_list_null>::= [(<列名>{,<列名>})]
<ins_value_list >::= (<插入值>{,<插入值>});|(<插入值>{,<插入值>}){, (<插入值>{,<插入值>})
```

参数

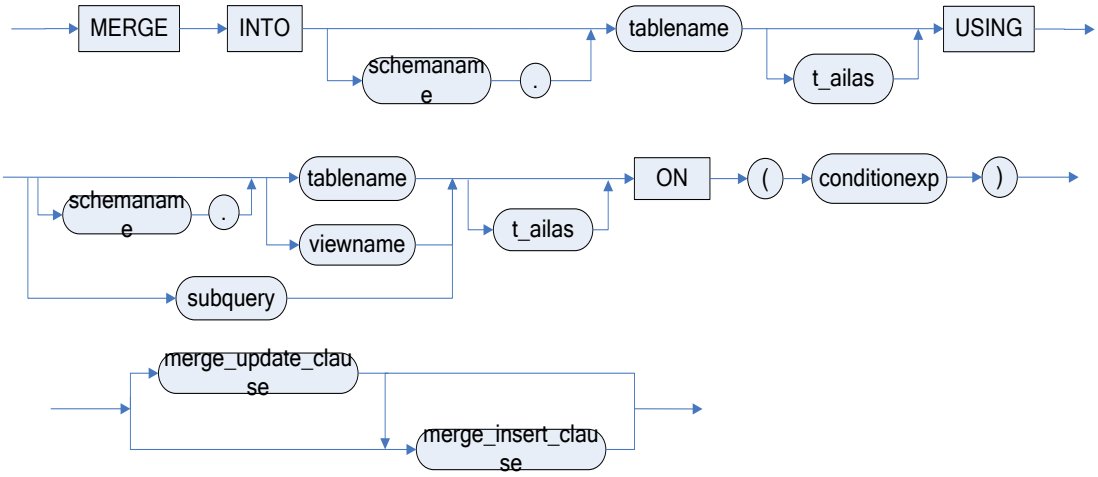
1．<模式名>　　指明该表或视图所属的模式，缺省为当前用户的缺省模式；

2．<基表名>　　指明被修改数据的基表的名称；

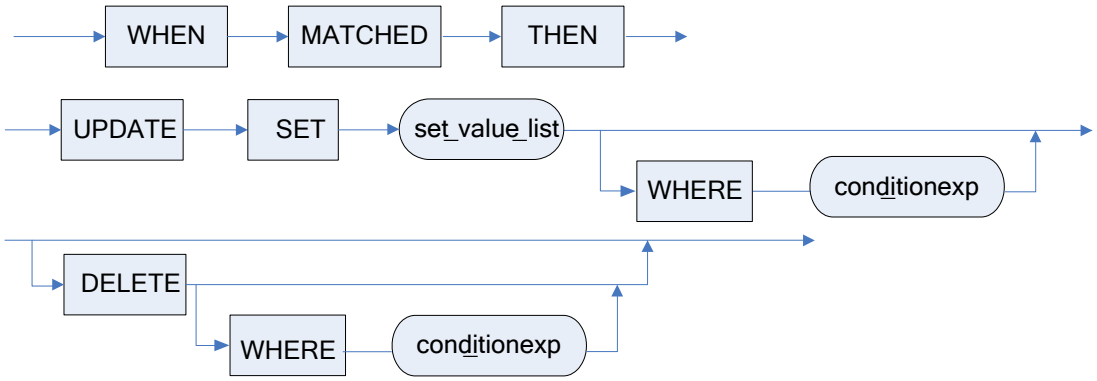
3．<视图名>　　指明被修改数据的视图的名称，实际上 DM 对视图的基表更新数据；

4．<条件表达式>指明限制被操作执行的行必须符合指定的条件，如果省略此子句，则对表或视图中所有的行进行操作。

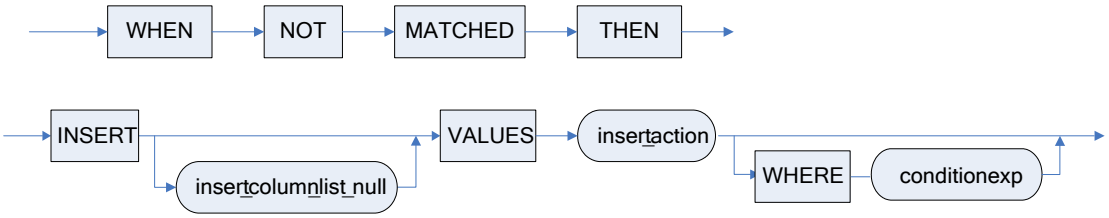
图例



merge_update_clause



merge_insert_clause



使用说明

1．INTO 后为目标表，表示待更新、插入的表（可更新视图）；

2．USING 后为源表（普通表或可更新视图），表示用于和目标表匹配、更新或插入的数据源；

3．ON (<条件判断表达式>)表示目标表和源表的连接条件，如果目标表有匹配连接条件的记录则执行更新该记录，如果没有匹配到则执行插入源表数据；

4．MERGE_UPDATE_CLAUSE：当目标表和源表的 JOIN 条件为 TRUE 时，执行该语句；

1) 如果更新执行，更新语句会触发所有目标表上的 UPDATE 触发器，也会进行约束检查；

2) 可以指定更新条件，如果不符合条件就不会执行更新操作。更新条件既可以和源表相关，也可以和目标表相关，或者都相关；

3) DELETE 子句只删除目标表和源表的 JOIN 条件为 TRUE，并且是更新后的符合删除条件的记录，DELETE 子句不影响 INSERT 项插入的行。删除条件作用在更新后的记录上，既可以和源表相关，也可以和目标表相关，或者都相关。如果 JOIN 条件为 TRUE，但是不符合更新条件，并没有更新数据，那么 DELETE 将不会删除任何数据。当执行了删除操作，会触发目标表上的 DELETE 触发器，也会进行约束检查。

5．MERGE_INSERT_CLAUSE：当目标表和源表的 JOIN 条件为 FALSE 时，执行该语句。同时会触发目标表上的 INSERT 触发器，也会进行约束检查。可指定插入条件，插入条件只能在源表上设置；

6．MERGE_UPDATE_CLAUSE 和 MERGE_INSERT_CLAUSE 既可以同时指定，也可以只出现其中任何一个；

7．需要有对源表的 SELECT 权限，对目标表的 UPDATE/INSERT 权限，如果 UPDATE 子句有 DELETE，还需要有 DELETE 权限；

8．UPDATE 子句不能更新在 ON 连接条件中出现的列；

9．如果匹配到，源表中的匹配行必须唯一，否则报错；

10．Insert_action 不能包含目标表列；

11．插入的 where 条件只能包含源表列。

举例说明

下面的例子把 T1 表中 C1 值为 2 的记录行中的 C2 列更新为表 T2 中 C3 值的记录中 C4 列的值，同时把 T2 中 C3 列为 4 的记录行插入到了 T1 中。

```
DROP TABLE T1;
DROP TABLE T2;
CREATE TABLE T1 (C1 INT,    C2 VARCHAR(20));
CREATE TABLE T2 (C3 INT,    C4 VARCHAR(20));
INSERT INTO T1 VALUES(1,'T1_1');
INSERT INTO T1 VALUES(2,'T1_2');
INSERT INTO T1 VALUES(3,'T1_3');
INSERT INTO T2 VALUES(2,'T2_2');
INSERT INTO T2 VALUES(4,'T2_4');
COMMIT;

MERGE INTO T1 USING T2 ON (T1.C1=T2.C3)
WHEN MATCHED THEN UPDATE SET T1.C2=T2.C4
WHEN NOT MATCHED THEN INSERT (C1,C2) VALUES(T2.C3, T2.C4);
```

下面的例子把 T1 表中 C1 值为 2,4 的记录行中的 C2 列更新为表 T2 中 C3 值的记录中 C4 列的值，同时把 T2 中 C3 列为 5 的记录行插入到了 T1 中。由于 UPDATE 带了 DELETE 子句，且 T1 中 C1 列值为 2 和 4 的记录行被更新过，而 C1 为 4 的行符合删除条件，最终该行会被删除掉。

```
DROP TABLE T1;
DROP TABLE T2;
CREATE TABLE T1 (C1 INT,    C2 VARCHAR(20));
CREATE TABLE T2 (C3 INT,    C4 VARCHAR(20));
INSERT INTO T1 VALUES(1,'T1_1');
INSERT INTO T1 VALUES(2,'T1_2');
INSERT INTO T1 VALUES(3,'T1_3');
INSERT INTO T1 VALUES(4,'T1_4');
INSERT INTO T2 VALUES(2,'T2_2');
INSERT INTO T2 VALUES(4,'T2_4');
INSERT INTO T2 VALUES(5,'T2_5');
COMMIT;

MERGE INTO T1 USING T2 ON (T1.C1=T2.C3)
```

```
WHEN MATCHED THEN UPDATE SET T1.C2=T2.C4 WHERE T1.C1 > 2 DELETE WHERE T1.C1=4
WHEN NOT MATCHED THEN INSERT (C1,C2) VALUES(T2.C3, T2.C4);
```

5.5 伪列的使用

除了 4.10.2 和 4.11 节中介绍的伪列外，DM 中还提供包括 ROWID、UID、USER、TRXID 等伪列。

5.5.1 ROWID

伪列从语法上和表中的列很相似，查询时能够返回一个值，但实际上在表中并不存在。用户可以对伪列进行查询，但不能插入、更新和删除它们的值。DM 支持的伪列有：ROWID，USER，UID，TRXID、ROWNUM 等。

DM 中行标识符 ROWID 用来标识数据库基表中每一条记录的唯一键值，标识了数据记录的确切的存储位置。如果用户在选择数据的同时从基表中选取 ROWID，在后续的更新语句中，就可以使用 ROWID 来提高性能。如果在查询时加上 FOR UPDATE 语句，该数据行就会被锁住，以防其他用户修改数据，保证查询和更新之间的一致性。例如：

```
SELECT    ROWID, VENDORID, NAME, CREDIT
FROM    PURCHASING.VENDOR
WHERE    NAME = '广州出版社';
--假设查询的 ROWID=CF06000000
UPDATE    PURCHASING.VENDOR    SET    CREDIT    = 2
WHERE    ROWID = 0XCF06000000;
```

5.5.2 UID 和 USER

伪列 USER 和 UID 分别用来表示当前用户的用户名和用户标识。

5.5.3 TRXID

伪列 TRXID 用来表示当前事务的事务标识。

5.6 DM 自增列的使用

5.6.1 DM 自增列定义

1．自增列功能定义

在表中创建一个自增列。该属性与 CREATE TABLE 语句一起使用，一个表只能有一个自增列。

语法格式

```
IDENTITY [ (种子，增量) ]
```

参数

1．种子　装载到表中的第一个行所使用的值；

2．增量　增量值，该值被添加到前一个已装载的行的标识值上。增量值可以为正数或负数，但不能为 0。

使用说明

1．IDENTITY 适用于 int(-2147483648～+2147483647)、bigint(-2^63～+2^63-2)类型的列；每个表只能创建一个自增列；

2．不能对自增列使用 DEFAULT 约束；

3．必须同时指定种子和增量值，或者二者都不指定。如果二者都未指定，则取默认值(1,1)；若种子或增量为小数类型，报错；

4．最大值和最小值为该列的数据类型的边界；

5．建表种子和增量大于最大值或者种子和增量小于最小值时报错；

6．自增列一旦生成，无法更新，不允许用 Update 语句进行修改。

2．自增列查询函数

1)　IDENT_SEED(函数)

语法格式：

```
IDENT_SEED ('tablename')
```

功能：返回种子值，该值是在带有自增列的表中创建自增列时指定的。

参数：tablename：是带有引号的字符串常量，也可以是变量、函数或列名。tablename 的数据类型为 char 或 varchar。其含义是表名，可带模式名前缀。

返回类型：返回数据类型为 int / NULL

2)　IDENT_INCR(函数)

语法格式：

```
IDENT_INCR ('tablename')
```

功能：返回增量值，该值是在带有自增列的表中创建自增列时指定的。

参数：tablename：是带有引号的字符串常量，也可以是变量、函数或列名。tablename 的数据类型为 char 或 varchar。其含义是表名，可带模式名前缀。

返回类型：返回数据类型为 int / NULL

例 用自增列查询函数获得表 PERSON_TYPE 的自增列的种子和增量信息。

```
SELECT　IDENT_SEED('PERSON.PERSON_TYPE');
查询结果为：1
SELECT IDENT_INCR('PERSON.PERSON_TYPE');
查询结果为：1
```

5.6.2 SET IDENTITY_INSERT 属性

设置是否允许将显式值插入表的自增列中。

语法格式

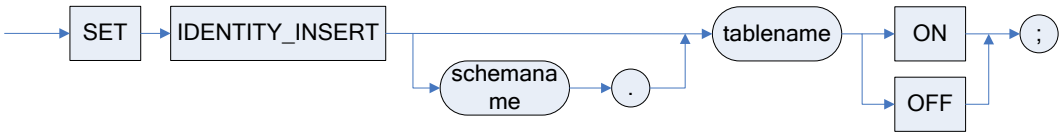
```
SET IDENTITY_INSERT [<模式名>.]<表名> ON | OFF;
```

参数

1．<模式名>　指明表所属的模式，缺省为当前模式；

2．<表名>　指明含有自增列的表名。

图例



使用说明

1．IDENTITY_INSERT 属性的默认值为 OFF。SET IDENTITY_INSERT 的设置是在执行或运行时进行的。当一个连接结束，IDENTITY_INSERT 属性将被自动还原为 OFF；

2．DM 要求一个会话连接中只有一个表的 IDENTITY_INSERT 属性可以设置为 ON，当设置一个新的表 IDENTITY_INSERT 属性设置为 ON 时，之前已经设置为 ON 的表会自动还原为 OFF。当一个表的 IDENTITY_INSERT 属性被设置为 ON 时，该表中的自动增量列的值由用户指定。如果插入值大于表的当前标识值(自增列当前值)，则 DM 自动将新插入值作为当前标识值使用，即改变该表的自增列当前值；否则，将不影响该自增列当前值；

3．当设置一个表的 IDENTITY_INSERT 属性为 OFF 时，新插入行中自增列的当前值由系统自动生成，用户将无法指定；

4．自增列一经插入，无法修改；

5．手动插入自增列除了要求将 IDENTITY_INSERT 设置为 ON 以外，还要求在插入列表中明确指定待插入的自增列列名，插入方式与非 IDENTITY 表是完全一样的。

举例说明

例 SET IDENTITY_INSERT 的使用

1) PERSON_TYPE 表中的 PERSON_TYPEID 列是自增列，目前拥有的数据如表 5.6.1 所示。

表 5.6.1

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

2) 在该表中插入数据，自增列的值由系统自动生成。

```
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('销售总监');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('人力资源部经理');
```

插入结果如表 5.6.2 所示：

表 5.6.2

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理

3) 当插入数据并且要指定自增列的值时，必须要通过语句将 IDENTITY_INSERT 设置为 ON 时，插入语句中可以指定 PERSON_TYPE 中要插入的列，也可以不指定，不指定则是为每一列赋值。例如：

```
SET IDENTITY_INSERT PERSON.PERSON_TYPE ON;
```

```
INSERT INTO PERSON.PERSON_TYPE(PERSON_TYPEID, NAME) VALUES( 8, '广告部经理');
INSERT INTO PERSON.PERSON_TYPE VALUES( 9, '财务部经理');
```

插入结果如表 5.6.3 所示：

表 5.6.3

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理
8	广告部经理
9	财务部经理

4) 不允许用户修改自增列的值。

```
UPDATE  PERSON.PERSON_TYPE SET PERSON_TYPEID = 9 WHERE NAME = '广告部经理';
```

修改失败。对于自增列，不允许 UPDATE 操作。

5) 还原 IDENTITY_INSERT 属性。

```
SET IDENTITY_INSERT PERSON.PERSON_TYPE OFF;
```

6) 插入后再次查询。注意观察自增列当前值的变化。

```
INSERT  INTO  PERSON.PERSON_TYPE(NAME)  VALUES('市场总监');
```

表 5.6.4

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	销售总监
6	人力资源部经理
8	广告部经理
9	财务部经理
10	市场总监

第 6 章 视图

视图是从一个或几个基表(或视图)导出的表，它是一个虚表，即数据字典中只存放视图的定义(由视图名和查询语句组成)，而不存放对应的数据，这些数据仍存放在原来的基表中。当需要使用视图时，则执行其对应的查询语句，所导出的结果即为视图的数据。因此当基表中的数据发生变化时，从视图中查询出的数据也随之改变了，视图就像一个窗口，透过它可以看到数据库中用户感兴趣的数据和变化。由此可见，视图是关系数据库系统提供给用户以多种角度观察数据库中数据的重要机制，体现了数据库本身最重要的特色和功能，它简化了用户数据模型，提供了逻辑数据独立性，实现了数据共享和数据的安全保密。视图是数据库技术中一个十分重要的功能。

视图一经定义，就可以和基表一样被查询、修改和删除，也可以在视图之上再建新视图。由于对视图数据的更新均要落实到基表上，因而操作起来有一些限制，读者应注意如何才能在视图中正确更新数据。在本章各例中，如不特别说明，以下例子均使用 BOOKSHOP 实例库，用户均为建表者 SYSDBA。

6.1 视图的作用

视图是提供给用户以多种角度观察数据库中数据的重要机制。尽管在对视图作查询和更新时有各种限制，但只要用户对 DM_SQL 语言熟悉，合理使用视图对用户建立自己的管理信息系统会带来很多的好处和方便，归纳起来，主要有以下几点：

1．用户能通过不同的视图以多种角度观察同一数据

可针对不同需要建立相应视图，使他们从不同的需要来观察同一数据库中的数据。

2．简化了用户操作

由于视图是从用户的实际需要中抽取出来的虚表，因而从用户角度来观察这种数据库结构必然简单清晰。另外，由于复杂的条件查询已在视图定义中一次给定，用户再对该视图查询时也简单方便得多了。

3．为需要隐蔽的数据提供了自动安全保护

所谓“隐蔽的数据”是指通过某视图不可见的数据。由于对不同用户可定义不同的视图，使需要隐蔽的数据不出现在不应该看到这些数据的用户视图上，从而由视图机制自动提供了对机密数据的安全保密功能。

4．为重构数据库提供了一定程度的逻辑独立性

在建立调试和维护管理信息系统的过程中，由于用户需求的变化、信息量的增长等原因，经常会出现数据库的结构发生变化，如增加新的基表，或在已建好的基表中增加新的列，或需要将一个基表分解成两个子表等，这称为数据库重构。数据的逻辑独立性是指当数据库重构时，对现有用户和用户程序不产生任何影响。

在管理信息系统运行过程中，重构数据库最典型的示例是将一个基表垂直分割成多个表。将经常要访问的列放在速度快的服务器上，而不经常访问的列放在较慢的服务器上。

例如将 PRODUCT 表分解成两个基表。

```
PRODUCT(PRODUCTID,NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCT_CATEGORYID,PRODUCTNO,DESCRIPTION,PHOTO,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME),
```

分解为两个基表：

```
PRODUCT_1(PRODUCTID,NAME,AUTHOR,PUBLISHER,NOWPRICE)
PRODUCT_2(PRODUCTID,PUBLISHERTIME,PRODUCT_CATEGORYID,PRODUCTNO,DESCRIPTION,PHOTO,SATETYSTOCKLEVEL,ORIGINALPRICE,NOWPRICE,DISCOUNT,TYPE,PAPERTOTAL,WORDTOTAL,SELLSTARTTIME,SELLENDTIME)
```

并将 PRODUCT 表中的数据分别插入这两个新建表中，再删去 PRODUCT 表。这样一来，原有用户程序中有 PRODUCT 表的操作就均无法进行了。为了减少对用户程序影响，这时可在 PRODUCT_1 和 PRODUCT_2 两基表上建立一个名字为 PRODUCT 的视图，因为新建视图维护了用户外模式的原状，用户的应用程序不用修改仍可通过视图查询到数据，从而较好支持了数据的逻辑独立性。

6.2 视图的定义

语法格式

```
CREATE [OR REPALCE] VIEW
[<模式名>.]<视图名>[(<列名> {,<列名>})]
AS <查询说明>
[WITH CHECK OPTION];
<查询说明>::=<表查询> | <表连接>
<表查询>::=<子查询表达式>[ORDER BY 子句]
```

参数

1．<模式名>　　指明被创建的视图属于哪个模式，缺省为当前模式；

2．<视图名>　　指明被创建的视图的名称；

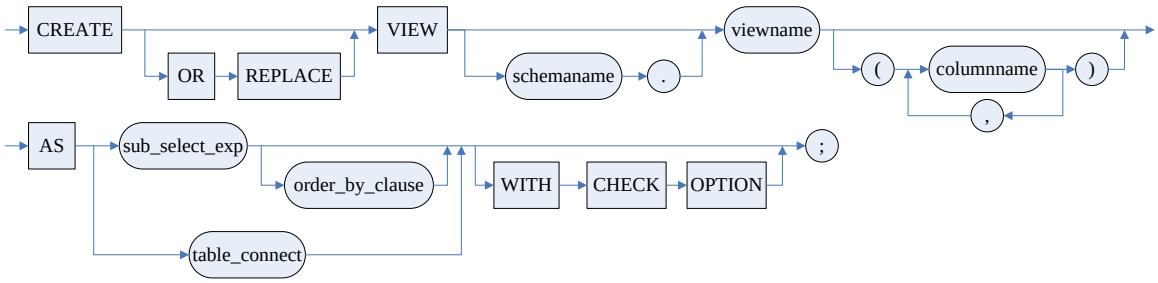
3．<列名>　　指明被创建的视图中列的名称；

4．<子查询表达式> 标识视图所基于的表的行和列。其语法遵照 SELECT 语句的语法规则；

5．<表连接>　　请参看第四章表连接查询部分；

6．WITH CHECK OPTION　指明往该视图中插入或修改数据时，插入行或更新行的数据必须满足视图定义中<查询说明>所指定的条件，如果不带该选项，则插入行或更新行的数据不必满足视图定义中<查询说明>所指定的条件。

图例



语句功能

供 DBA 或该视图的拥有者且具有 CREATE VIEW 权限的用户定义视图。

使用说明

1．<视图名>后所带<列名>不得同名，个数必须与<查询说明>中 SELECT 后的<值表达式>的个数相等。如果<视图名>后不带<列名>，则隐含该视图中的列由<查询说明>中 SELECT 后的各<值表达式>组成，但这些<值表达式>必须是单纯列名。如果出现以下三种情况之一，<视图名>后的<列名>不能省：

1) <查询说明>中 SELECT 后的<值表达式>不是单纯的列名，而包含集函数或运算表达式；

2) <查询说明>包含了多表连接，使得 SELECT 后出现了几个不同表中的同名列作为

视图列；

3) 需要在视图中为某列取与<查询说明>中 SELECT 后<列名>不同的名字。

最后要强调的是：<视图名>后的<列名>必须全部省略或全部写明。

2．为了防止用户通过视图更新基表数据时，无意或故意更新了不属于视图范围内的基表数据，在视图定义语句的子查询后提供了可选项 WITH CHECK OPTION。如选择，表示往该视图中插入或修改数据时，要保证插入行或更新行的数据满足视图定义中<查询说明>所指定的条件，不选则可不满足；

3．如果视图查询中包含下列结构：连接、集合运算符、GROUP BY 子句，则在视图上不能进行插入、修改和删除操作；

4．视图是一个逻辑表，它自己不包含任何数据；

5．视图上可以建立 INSTEAD OF 触发器（只允许行级触发），但不允许创建 BEFORE/AFTER 触发器。

权限

该语句的使用者必须对<查询说明>中的每个表均具有 SELECT 权限。

举例说明

例 对 VENDOR 表创建一个视图，名为 VENDOR_EXCELLENT,保存信誉等级为 1 的供应商，列名有：VENDORID,ACCOUNTNO,NAME,ACTIVEFLAG。

```
CREATE  VIEW  PURCHASING.VENDOR_EXCELLENT  AS
SELECT  VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT
FROM  PURCHASING.VENDOR
WHERE  CREDIT = 1;
```

由于视图列名与查询说明中 SELECT 后的列名相同，所以视图名后的列名可省。

运行该语句，AS 后的查询语句并未执行，系统只是将所定义的<视图名>及<查询说明>送数据字典保存。对用户来说，就像在数据库中已经有 VENDOR_EXCELLENT 这样一个表。

如果对该视图作查询：

```
SELECT  *  FROM  PURCHASING.VENDOR_EXCELLENT;
```

查询结果见下表 6.1.1。

表 6.1.1

VENDORID	ACCOUNTNO	NAME	ACTIVEFLAG	CREDIT
3	00	北京十月文艺出版社	1	1
4	00	人民邮电出版社	1	1
5	00	清华大学出版社	1	1
6	00	中华书局	1	1
7	00	广州出版社	1	1
8	00	上海出版社	1	1
9	00	21 世纪出版社	1	1
10	00	外语教学与研究出版社	1	1
11	00	社械工业出版社	1	1
12	00	文学出版社	1	1

用户可以在该表上作数据库的查询、插入、删除、修改等操作。在建好的视图之上还可以再建立视图。

由于以上定义包含可选项 WITH CHECK OPTION，以后对该视图作插入、修改和删除操作时，系统均会自动用 WHERE 后的条件作检查，不满足条件的数据，则不能通过该视图更新相应基表中的数据。

例 视图也可以建立在多个基表之上。构造一视图，名为 SALESPERSON_INFO，用来

保存销售人员的信息，列名有：SALESPERSONID,TITLE,NAME,SALESLASTYEAR。

```
CREATE VIEW SALES.SALESPERSON_INFO AS
SELECT T1.SALESPERSONID, T2.TITLE, T3.NAME, T1.SALESLASTYEAR
FROM SALES.SALESPERSON T1, RESOURCES.EMPLOYEE T2, PERSON.PERSON T3
WHERE T1.EMPLOYEEID = T2.EMPLOYEEID AND T2.PERSONID = T3.PERSONID;
```

如果对该视图作查询：

```
SELECT * FROM SALES.SALESPERSON_INFO;
```

查询结果见下表 6.1.2。

表 6.1.2

SALESPERSONID	TITLE	NAME	SALESLASTYEAR
1	销售代表	郭艳	10.0000
2	销售代表	孙丽	20.0000

由前面的介绍可知，基表中的数据均是基本数据。为了减少数据冗余，由基本数据经各种计算统计出的数据一般是不存贮的，但这样的数据往往又要经常使用，这时可将它们定义成视图中的数据。

例 在 PRODUCT_VENDOR 上建立一视图，用于统计数量。

```
CREATE VIEW PRODUCTION.VENDOR_STATIS(VENDORID, PRODUCT_COUNT) AS
SELECT VENDORID, COUNT(PRODUCTID)
FROM PRODUCTION.PRODUCT_VENDOR
GROUP BY VENDORID
ORDER BY VENDORID;
```

在该语句中，由于 SELECT 后出现了集函数 COUNT(PRODUCTID)，不属于单纯的列名，所以视图中的对应列必须重新命名，即在<视图名>后明确说明视图的各个列名。

由于该语句中使用了 GROUP BY 子句，所定义的视图也称分组视图。分组视图的<视图名>后所带<列名>不得包含集函数。

如果对该视图作查询：

```
SELECT * FROM PRODUCTION.VENDOR_STATIS;
```

查询结果如下表 6.1.3 所示。

表 6.1.3

VENDORID	PRODUCT_COUNT
5	1
6	2
7	1
8	1
9	1
10	1
11	1

6.3 视图的删除

一个视图本质上是基于其他基表或视图上的查询，我们把这种对象间关系称为依赖。用户在创建视图成功后，系统还隐式地建立了相应对象间的依赖关系。在一般情况下，当一个视图不被其他对象依赖时可以随时删除视图。

语法格式

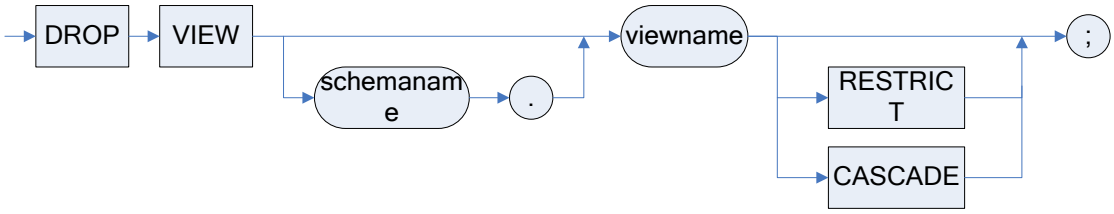
```
DROP VIEW [<模式名>.]<视图名> [RESTRICT | CASCADE];
```

参数

1．<模式名>　　指明被删除视图所属的模式，缺省为当前模式；

2．<视图名>　　指明被删除视图的名称。

图例



使用说明

1．视图删除有两种方式：RESTRICT/CASCADE 方式。其中 RESTRICT 为缺省值。当设置 dm.ini 中的参数 DROP_CASCADE_VIEW 值为 1 时，如果在该视图上建有其它视图，必须使用 CASCADE 参数才可以删除所有建立在该视图上的视图，否则删除视图的操作不会成功；当设置 dm.ini 中的参数 DROP_CASCADE_VIEW 值为 0 时，RESTRICT 和 CASCADE 方式都会成功，且只会删除当前视图，不会删除建立在该视图上的视图；

2．如果没有删除参考视图的权限，那么两个视图都不会被删除；

3．该视图删除后，用户在其上的权限也均自动取消，以后系统中再建的同名视图，是与他毫无关系的视图。

权限

使用者必须拥有 DBA 权限或是该视图的拥有者。

举例说明

例 删除视图 VENDOR_EXCELLENT，可使用下面的语句：

```
DROP　VIEW　PURCHASING.VENDOR_EXCELLENT;
```

当该视图对象被其他对象依赖时，用户在删除视图时必须带 CASCADE 参数，系统会将依赖于该视图的其他数据库对象一并删除，以保证数据库的完整性。

例 删除视图 SALES.SALESPERSON_INFO，同时删除此视图上的其他视图，可使用下面的语句：

```
DROP　VIEW　SALES.SALESPERSON_INFO CASCADE;
```

6.4 视图的查询

视图一旦定义成功，对基表的所有查询操作都可用于视图。对于用户来说，视图和基表在进行查询操作时没有区别。

例 从 VENDOR_EXCELLENT 中查询 ACTIVEFLAG 为 1 的供应商的编号和名称。

```
SELECT　VENDORID, NAME
FROM　PURCHASING.VENDOR_EXCELLENT
WHERE　ACTIVEFLAG = 1;
```

系统执行该语句时，先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，按定义语句查询基表，得到视图表，再根据条件:ACTIVEFLAG = 1 查询视图表，选择所需列名，得到结果如下表 6.3.1 所示。

表 6.3.1

VENDORID	NAME
3	北京十月文艺出版社

4	人民邮电出版社
5	清华大学出版社
6	中华书局
7	广州出版社
8	上海出版社
9	21 世纪出版社
10	外语教学与研究出版社
11	社械工业出版社
12	文学出版社

视图尽管是虚表，但它仍可与其它基表或视图作连接查询，也可以出现在子查询中。

例 查询信誉等级为 1 的供应商供应的图书编号、名称、通常价格和供应商名称。

```
SELECT  T1.PRODUCTID, T1.NAME, T2.STANDARDPRICE, T3.NAME
FROM  PRODUCTION.PRODUCT  T1,  PRODUCTION.PRODUCT_VENDOR  T2,
        PURCHASING.VENDOR_EXCELLENT  T3
WHERE  T1.PRODUCTID  =  T2.PRODUCTID  AND T2.VENDORID = T3.VENDORID;
```

系统执行该语句时，先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，按定义语句查询基表，得到视图表，再将 PRODUCT、PRODUCT_VENDOR 和视图表按连接条件作连接，选择所需列名，得到最后结果。

得到结果如下表 6.3.2 所示。

表 6.3.2

PRODUCTID	NAME	STANDARDPRICE	NAME
1	红楼梦	25.0000	中华书局
2	水浒传	25.0000	中华书局
3	老人与海	25.0000	上海出版社
4	射雕英雄传(全四册)	25.0000	广州出版社
7	数据结构(C 语言版)(附光盘)	25.0000	清华大学出版社
9	突破英文基础词汇	25.0000	外语教学与研究出版社
10	噼里啪啦丛书(全 7 册)	25.0000	21 世纪出版社
11	工作中无小事	25.0000	机械工业出版社

6.5 视图的编译

一个视图依赖于其基表或视图，如果基表定义发生改变，如增删一列，或者视图的相关权限发生改变，可能导致视图无法使用。在这种情况下，可对视图重新编译，检查视图的合法性。

语法格式

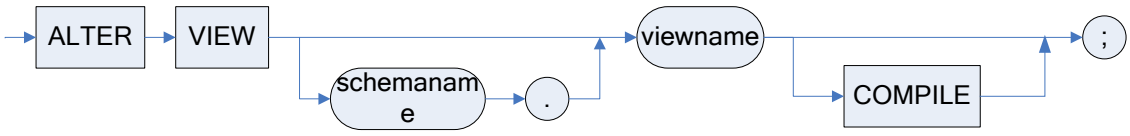
```
ALTER VIEW [<模式名>.]<视图名> COMPILE;
```

参数

1．<模式名>　　指明被删除视图所属的模式，缺省为当前模式；

2．<视图名>　　指明被删除视图的名称；

图例



使用说明

对视图的定义重新进行分析和编译，如果编译出错，则报错，可以此判断视图依赖的基表是否已被删除或修改了表定义。

权限

使用者必须拥有 DBA 权限或是该视图的拥有者。

举例说明

例 重新编译视图 PURCHASING.VENDOR_EXCELLENT。

```
ALTER VIEW PURCHASING.VENDOR_EXCELLENT COMPILE;
```

6.6 视图数据的更新

视图数据的更新包括插入(INSERT)、删除(DELETE)和修改(UPDATE)三类操作。由于视图是虚表，并没有实际存放数据，因此对视图的更新操作均要转换成对基表的操作。在 SQL 语言中，对视图数据的更新语句与对基表数据的更新语句在格式与功能方面是一致的。

例 从视图 VENDOR_EXCELLENT 中将名称为人民邮电出版社的 ACTIVEFLAG 改为 0。

```
UPDATE  PURCHASING.VENDOR_EXCELLENT
SET  ACTIVEFLAG = 0 WHERE NAME = '人民邮电出版社';
```

系统执行该语句时，首先从数据字典中取出视图 VENDOR_EXCELLENT 的定义，将其中的查询说明与对视图的修改语句结合起来，转换成对基表的修改语句，然后再执行这个转换后的更新语句。

```
UPDATE  PURCHASING.VENDOR
SET  ACTIVEFLAG = 0
WHERE  NAME = '人民邮电出版社'  AND  CREDIT = 1;
```

例 往视图 VENDOR_EXCELLENT 中插入一个新的记录，其中 ACCOUNTNO 为 00，NAME 为电子工业出版社，ACTIVEFLAG 为 1，WEBURL 为空。则相应的插入语句为：

```
INSERT  INTO  PURCHASING.VENDOR_EXCELLENT(ACCOUNTNO,  NAME,  ACTIVEFLAG,
CREDIT)
VALUES('00 ', '电子工业出版社', 1, 1);
```

例 从视图 VENDOR_EXCELLENT 中删除名称为中华书局的供应商信息。

```
DELETE  FROM  PURCHASING.VENDOR_EXCELLENT  WHERE  NAME = '中华书局';
```

系统将该语句与 VENDOR_EXCELLENT 视图的定义相结合，转换成对基表的语句：

```
DELETE  FROM  PURCHASING.VENDOR
WHERE  NAME = '中华书局'  AND  CREDIT = 1;
```

系统执行该语句，会报告违反约束错误，因为 VENDOR_EXCELLENT 尽管是视图，在做更新时一样要考虑基表间的引用关系。PRODUCT_VENDOR 表与 VENDOR 表存在着引用关系，PRODUCT_VENDOR 表为引用表，VENDOR 表为被引用表，只有当引用表中没有相应 VENDORID 时才能删除 VENDOR 表中相应记录。

在关系数据库中，并不是所有视图都是可更新的，即并不是所有的视图更新语句均能有意义地转换成相应的基表更新语句，有些甚至是根本不能转换。例如对视图 VENDOR_STATIS：

```
UPDATE    PRODUCTION.VENDOR_STATIS
SET    PRODUCT_COUNT    = 3
WHERE VENDORID = 5;
```

由于产品数量是查询结果按供应商编号分组后各组所包含的行数，这是无法修改的。像这样的视图为不可更新视图。

目前，不同的关系数据库管理系统产品对更新视图的可操作程度均有差异。DM 系统有这样的规定：

1. 如果视图由两个以上的基表导出时，则该视图不允许更新；
2. 如果视图建在单个基表或单个可更新视图上，且该视图包含了表中的全部聚集索引键，则该视图为可更新视图；
3. 如果视图列是集函数，或视图定义中的查询说明带了 GROUP BY 子句或 HAVING 子句，则该视图不允许更新；
4. 在不允许更新视图之上建立的视图也不允许更新。

应该说明的是：只有当视图是可更新的时候，才可以选择 WITH CHECK OPTION 项。

第 7 章 物化视图

物化视图是从一个或几个基表导出的表，同视图相比，它存储了导出表的真实数据。当基表中的数据发生变化时，物化视图所存储的数据将变得陈旧，用户可以通过手动刷新或自动刷新来对数据进行同步。

在本章各例中，如不特别说明，以下例子均用户均为建表者 SYSDBA。

7.1 物化视图的定义

语法格式

```
CREATE MATERIALIZED VIEW [<模式名>.]<物化视图名>[(<列名>{,<列名>})][BUILD IMMEDIATE|BUILD DEFERRED][<STORAGE 子句>][<物化视图刷新选项>][<查询改写选项>]AS<查询说明>
<查询说明>::= <表查询> | <表连接>
<表查询>::=<子查询表达式>[ORDER BY 子句]
<物化视图刷新选项> ::= { REFRESH {
{FAST | COMPLETE | FORCE } | { ON DEMAND | ON COMMIT } | { START WITH datetime_expr | NEXT datetime_expr } | WITH { PRIMARY KEY | ROWID }} NEVER REFRESH }
<查询改写选项>::={ DISABLE | ENABLE } QUERY REWRITE
<datetime_expr>::= SYSDATE[+数值常量]
```

参数

1．<模式名>　指明被创建的视图属于哪个模式，缺省为当前模式；

2．<物化视图名>　指明被创建的物化视图的名称；

3．<列名>　指明被创建的物化视图中列的名称；

4．[BUILD IMMEDIATE|BUILD DEFERRED]　指明 BUILD IMMEDIATE 为立即填充数据，默认为立即填充；BUILD DEFERRED 为延迟填充，使用这种方式要求第一次刷新必须为 COMPLETE 完全刷新。

5．<子查询表达式>　标识物化视图所基于的表的行和列。其语法遵照 SELECT 语句的语法规则；

6．<表连接>　请参看第四章表连接查询部分；

7．定义查询中的 ORDER BY 子句仅在创建物化视图时使用，此后 ORDER BY 被忽略；

8．刷新模式

- FAST

  根据相关表上的数据更改记录进行增量刷新。普通 DML 操作生成的记录存在于物化视图日志。使用 FAST 刷新之前，必须先建好物化视图日志。

- COMPLETE

  通过执行物化视图的定义脚本进行完全刷新。

- FORCE

  默认选项。当快速刷新可用时采用快速刷新否则采用完全刷新。

9．刷新时机

- ON COMMIT

  在相关表上事务提交时进行快速刷新，这会增加 COMMIT 完成的时间。DM7 目前仅语法支持 ON COMMIT，实际功能并未实现。

约束：

含有对象类型的不支持；

包含远程表的不支持。

- START WITH … NEXT

START WITH 用于指定首次刷新物化视图的时间，NEXT 指定自动刷新的间隔；

如果省略 START WITH 则首次刷新时间为当前时间加上 NEXT 指定的间隔；

如果指定 START WITH 省略 NEXT 则物化视图只会刷新一次；

如果二者都未指定物化视图不会自动刷新。

- ON DEMAND

由用户通过 REFRES 语法进行手动刷新。如果指定了 START WITH 和 NEXT 子句就没有必要指定 ON DEMAND。

- NEVER REFRESH

物化视图从不进行刷新。可以通过 ALTER MATERALIZED VIEW <物化视图名> FRESH 进行更改。

10．刷新选项

- WITH PRIMARY KEY

默认选项。除 WITH ROWID 之外都要选择此选项。

  - 必须含有 PRIMARY KEY 约束，选择列必须直接含有所有的 PRIMARY KEY（UPPER(col_name)的形式不可接受）
  - 不能含有对象类型

- WITH ROWID

必须基于单表，并且不能包含如下内容：

  - DISTINCT 或聚集函数
  - GROUP BY 或 CONNECT BY 子句
  - 子查询
  - 连接
  - 集合运算
  - 不能含有对象类型

如果使用 WITH ROWID 的同时使用快速刷新，则必须将 ROWID 提取出来，和其他列名一起，以别名的形式显示。

11．QUERY REWRITE 选项

- ENABLE

允许物化视图用于查询改写。

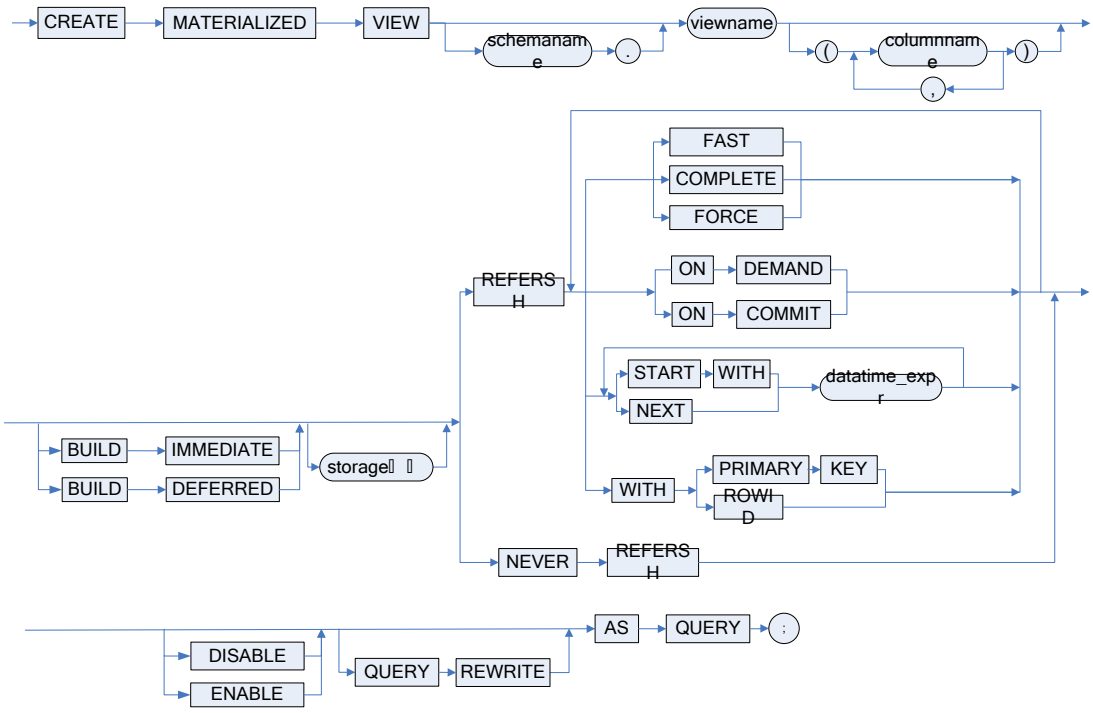
- DISABLE

禁止物化视图用于查询改写。

目前 DM7 仅语法支持查询改写选项，实际功能未实现。

12．datetime_expr 只能是日期常量表达式，SYSDATE[+数值常量]或日期间隔。

图例



datetime_expr

语句功能

供 DBA 或该物化视图的拥有者且具有 CREATE MATERIALIZED VIEW 权限的用户定义物化视图。

使用说明

1．创建物化视图时，会产生两个字典对象：物化视图和物化视图表，后者用于存放真实的数据；如果创建时指定了定时刷新选项则还可能存在一个 JOB 对象；

2．快速刷新物化视图的限制见本章第 8 节《物化视图的限制》；

3．由于受物化视图表的命名规则所限，物化视图基表名称长度必须小于 123 个字符；

4．用户可以直接通过 DM 管理工具中“代理”—>“作业”来修改定时刷新选项。

权限

1．在自己模式下创建物化视图时，该语句的使用者必须被授予 CREATE MATERIALIZED VIEW 系统权限，且至少拥有 CREATE TABLE 或者 CREATE ANY TABLE 两个系统权限中的一个；

2．在其他用户模式下创建物化视图时，该语句的使用者必须具有 CREATE ANY MATERIALIZED VIEW 系统权限，且物化视图的拥有者必须拥有 CREATE TABLE 系统权限；

3．物化视图的拥有者必须对<查询说明>中的每个表均具有 SELECT 权限或者具有 SELECT ANY TABLE 系统权限。

举例说明

例 对 VENDOR 表创建一个物化视图，名为 MV_VENDOR_EXCELLENT，保存信誉等级为 1 的供应商，列名有：VENDORID、ACCOUNTNO、NAME、ACTIVEFLAG、CREDIT。不允许查询改写，依据 ROWID 刷新且刷新间隔为一天。

```
CREATE MATERIALIZED VIEW    PURCHASING.MV_VENDOR_EXCELLENT
REFRESH WITH ROWID START WITH SYSDATE NEXT SYSDATE + 1 AS
SELECT    VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT
FROM    PURCHASING.VENDOR
WHERE    CREDIT = 1;
```

如果使用 WITH ROWID 的同时，后面还要使用快速刷新，则此处的语句应写为：

```
CREATE MATERIALIZED VIEW    PURCHASING.MV_VENDOR_EXCELLENT
REFRESH WITH ROWID START WITH SYSDATE NEXT SYSDATE + 1 AS
SELECT    VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT,ROWID AS X
FROM    PURCHASING.VENDOR
WHERE    CREDIT = 1;
```

运行该语句后，DM 服务器将得到：1）物化视图：MV_VENDOR_EXCELLENT；2）物化视图表：MTAB$_MV_VENDOR_EXCELLENT；3）定时刷新的物化视图作业：MJOB_REFRESH_MVIEW_1670（假定 MTAB$_MV_VENDOR_EXCELLENT 对象的 ID 是 1670）。

对该物化视图进行查询：

```
SELECT    *    FROM    PURCHASING.MV_VENDOR_EXCELLENT;
```

查询结果见下表 7.1.1。

表 7.1.1

VENDORID	ACCOUNTNO	NAME	ACTIVEFLAG	CREDIT
3	00	北京十月文艺出版社	1	1
4	00	人民邮电出版社	1	1
5	00	清华大学出版社	1	1
6	00	中华书局	1	1
7	00	广州出版社	1	1
8	00	上海出版社	1	1
9	00	21 世纪出版社	1	1
10	00	外语教学与研究出版社	1	1
11	00	社械工业出版社	1	1
12	00	文学出版社	1	1

7.2 物化视图的修改

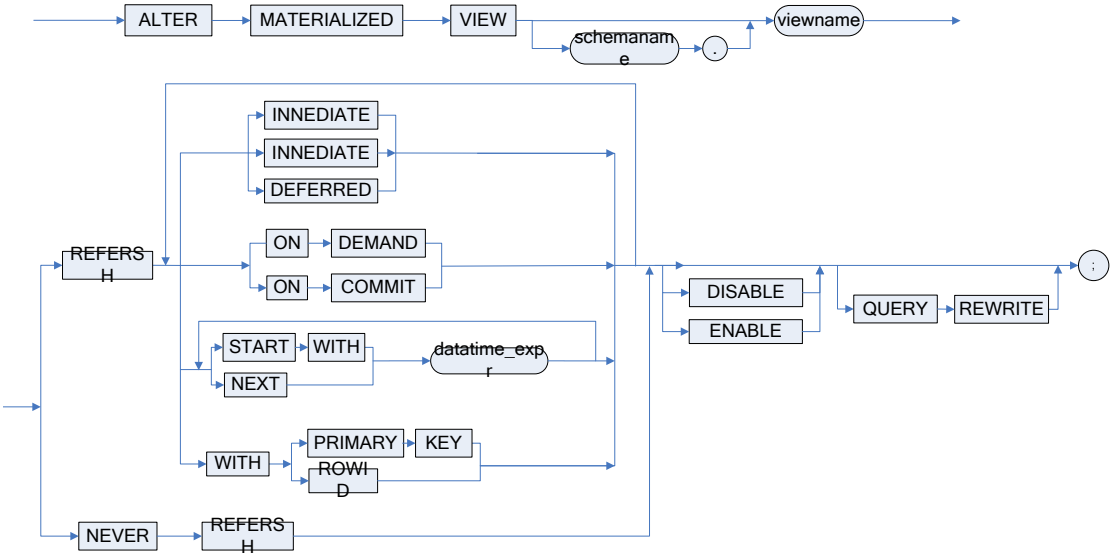
语法格式

```
ALTER MATERIALIZED VIEW [<模式名>.]<物化视图名>
[<物化视图刷新选项>]
[<查询改写选项>]
```

参数

<物化视图刷新选项>和<查询改写选项>参见本章第 1 节《物化视图的定义》。

图例



权限

使用者必须是该物化视图的拥有者或者拥有 ALTER ANY MATERIALIZED VIEW 系统权限。

举例说明

例 修改物化视图 MV_VENDOR_EXCELLENT，使之可以用于查询改写。

```
ALTER MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT ENABLE QUERY REWRITE;
```

例 修改物化视图 MV_VENDOR_EXCELLENT 为完全刷新。

```
ALTER MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT REFRESH COMPLETE;
```

7.3 物化视图的删除

语法格式

```
DROP MATERIALIZED VIEW [<模式名>.]<物化视图名>;
```

参数

1．<模式名>　指明被删除视图所属的模式，缺省为当前模式；

2．<物化视图名>　指明被删除物化视图的名称；

图例

使用说明

1．物化视图删除时会清除物化视图，物化视图表及定时刷新作业（如果存在的话）；

2．物化视图删除后，用户在其上的权限也均自动取消，以后系统中再建的同名物化视图，是与它毫无关系的物化视图；

3．用户不能直接删除物化视图表对象。

权限

使用者必须是物化视图的拥有者或者拥有 DROP ANY MATERIALIZED VIEW 系统权限。

举例说明

例 删除物化视图 MV_VENDOR_EXCELLENT，可使用下面的语句：

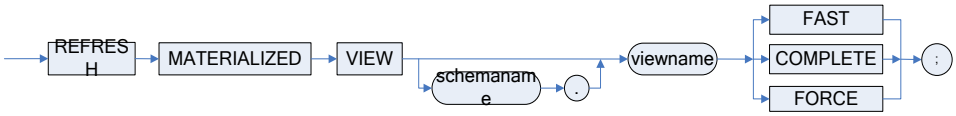
```
DROP MATERIALIZED VIEW PURCHASING.MV_VENDOR_EXCELLENT;
```

7.4 物化视图的更新

语法格式

```
REFRESH MATERIALIZED VIEW  [<模式名>.] <物化视图名>
[FAST                |
COMPLETE|
FORCE]
```

图例



权限

1．如果是基于物化视图日志的更新，则使用者必须是物化视图日志的拥有者或者具有 SELECT ANY TABLE 系统权限；

2．使用者必须是物化视图的拥有者或者具有 SELECT ANY TABLE 系统权限。

举例说明

例 采用完全方式刷新物化视图 MV_VENDOR_EXCELLENT。

```
REFRESH MATERIALIZED VIEW  PURCHASING.MV_VENDOR_EXCELLENT FAST; --使用快速刷新前，必须先建好物化视图日志
```

7.5 物化视图允许的操作

对物化视图进行查询或建立索引时这两种操作都会转为对其物化视图表的处理。用户不能直接对物化视图及物化视图表进行插入、删除、更新和 TRUNCATE 操作，对物化视图数据的修改只能通过刷新物化视图语句进行。

7.6 物化视图日志的定义

物化视图的快速刷新依赖于基表上的物化视图日志，物化视图日志记录了基表的变化信息。

语法格式

```
CREATE MATERIALIZED VIEW  LOG ON [<模式名>.]<表名>
[<STORAGE 子句>][<WITH 子句>][<PURGE 选项>]
<WITH  子句>::= WITH { PRIMARY KEY| ROWID | (<列名> [, <列名>])}
<PURGE 选项>::= PURGE { IMMEDIATE [ SYNCHRONOUS | ASYNCHRONOUS ]
        | START WITH datetime_expr [ NEXT datetime_expr | REPEAT INTERVAL interval_expr ]
```

参数

1．<模式名> 指明物化视图日志基表所属的模式，缺省为当前模式；

2．<表名> 指明创建日志的基表；

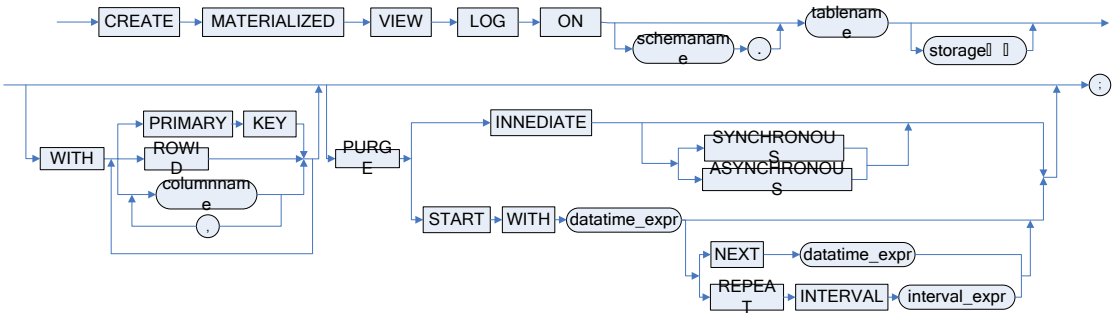
3．<WITH 子句> 基表中的哪些列将被包含到物化视图日志中；

4．<PURGE 选项> 指定每隔多长时间对物化视图日志中无用的记录进行一次清除。分两种情况：一是 IMMEDIATE 立即清除；二是 START WITH 定时清除。缺省是 PURGE

IMMEDIATE，即每次利用完物化视图日志后判断能否对日志表中的记录进行清除。SYNCHRONOUS 为同步清除；ASYNCHRONOUS 为异步清除。其中，ASYNCHRONOUS 仅语法支持，功能未实现；

5．datetime_expr　只能是日期常量表达式，SYSDATE[+数值常量]或日期间隔。

图例



使用说明

1．在表 T 上创建物化视图日志后会生成 MLOG$_T 的表和 MTRG$_T 的触发器；用户可以对 MLOG$_T 进行查询但是不能进行插入、删除和更新，触发器由系统维护,用户无法修改删除；

2．由于物化视图日志表的命名规则所限，日志基表名称长度必须小于 123 个字符；

3．用户可以直接通过 DM 管理工具中“代理”—>“作业”来修改定时刷新选项。

权限

1．如果是物化视图日志基表的拥有者，使用者必须拥有 CREATE TABLE 系统权限；

2．如果是物化视图日志基表的是其它模式下的表，使用者必须拥有 CREATE TABLE 系统权限，且物化视图日志的拥有者必须对<查询说明>中的每个表均具有 SELECT 权限或者具有 SELECT ANY TABLE 系统权限。

举例说明

例 在 PURCHASING. VENDOR 上创建物化视图日志，每天定时 PURGE。

```
CREATE MATERIALIZED VIEW LOG ON PURCHASING.VENDOR WITH ROWID(ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT) PURGE START WITH SYSDATE + 5 REPEAT INTERVAL '1' DAY;
```

注：间隔一天 PURGE 也可以写成 PURGE NEXT SYSDATE + 1。

7.7 物化视图日志的删除

语法格式

```
DROP MATERIALIZED VIEW　LOG ON [<模式名>.]<物化视图名>
```

物化视图日志删除时会同时 DROP 掉日志表对象和触发器对象。

图例

权限

使用者必须拥有删除表的权限。

7.8 物化视图的限制

7.8.1 物化视图的一般限制

1．物化视图不能包含集合操作，不能包含视图，查询语句不能包含有派生表；

2．物化视图定义不能含有垂直分区表；

3．不能对视图建立物化视图；

4．对物化视图日志、物化视图只能进行查询和建索引，不支持插入、删除、更新、MERGE INTO 和 TRUNCATE；

5．同一表上最多允许建立 127 个物化视图；

6．包含物化视图的普通视图及游标是不能更新的；

7．如果对某明细表进行了 TRUNCATE 操作，那么依赖于它的物化视图必须先进行一次完全刷新后才可以使用快速刷新。

7.8.2 物化视图的分类

依据物化视图定义中查询语句的不同分为以下五种。

1．SIMPLE：无 GROUP BY，无聚集函数，无连接操作；

2．AGGREGATE：仅包含有 group BY 和聚集函数；

3．JOIN：仅包含有多表连接；

4．Sub-Query：仅包含有子查询；

5．COMPLEX：除上述四种外的物化视图类型。

用户可以通过查看系统视图 SYS.USER_MVIEWS 的 MVIEW_TYPE 列来了解所定义物化视图的分类。

7.8.3 快速刷新通用约束

1．快速刷新物化视图要求每个基表都包含有物化视图日志，并且物化视图日志的创建时间不得晚于物化视图的最后刷新时间；

2．不能含有不确定性函数，如 SYSDATE 或 ROWNUM；

3．不能含有大字段类型；

4．查询项不能含有分析函数；

5．查询不能含有 HAVING 子句；

6．不能包含 ANY、ALL 及 NOT EXISTS；

7．不能含有层次查询；

8．不能在多个站点含有相关表；

9．同一张表上最多允许建立 127 个快速刷新的物化视图；

10．不能含有 UNION，UNION ALL，MINUS 等集合运算；

11．不能含有子查询；

12．只能基于普通表（视图，外部表，派生表等不支持）；

13．WITH PRIMAY KEY 时物化视图定义里只能是单表并且日志表里有 PK，多表时必

须是 WITH ROWID 并且日志表里有 ROWID，WITH ROWID 刷新时定义物化视图可以是单表，前提是日志表里有 ROWID；

14．对于 WITH ROWID 的快速刷新单表和多表时则需要一一选择 ROWID 并给出别名；

15．单表 WITH PK 刷新时，视图定义中必须包含所有的 PK 列；

16．如果日志定义中没有 WITH PRIMARY KEY 而扩展列又包含了，那么 DM7 认为这个和建立日志时指定 WITH PRIMARY KEY 效果相同。也就是说，基于这个日志建立 WITH PK 的快速刷新物化视图是允许的；

17．DM7 目前仅支持简单类型物化视图的快速刷新。

7.8.4 物化视图信息查看

用户可以通过系统视图 SYS.USER_MVIEWS 查看系统中所有物化视图的相关信息，视图定义如下：

列名	数据类型	备注
SCHID	INTEGER	物化视图所在模式 ID
MVIEW_NAME	Varchar(128)	物化视图名
QUERY	CLOB	物化视图定义查询语句
QUERY_LEN	INTEGER	物化视图定义查询语句长度
REWRITE_ENABLED	VARCHAR(128)	是否允许查询改写：’Y’允许；’N’不允许；
REFRESH_MODE	VARCHAR(128)	刷新模式，DEMAND 或 COMMIT
REFRESH_METHOD	VARCHAR(128)	刷新方法，FORCE、FAST 或 COMPLETE
MVIEW_TYPE	VARCHAR(128)	物化视图的类型
LAST_REFRESH_TYPE	VARCHAR(128)	最后一次刷新的方法
STALENESS	VARCHAR(128)	物化视图的状态： UNUSEABLE：物化视图数据从未构造和刷新； FRESH：物化视图数据和基表的数据一致； NEEDS_COMPILE：物化视图状态曾经是 FRESH 但现在未能保持一致，可以尝试使用快速刷新； COMPILE_ERROR：物化视图的定义语句执行出错； NEEDS_FULL_REFRESH：物化视图状态曾经是 FRESH 但现在未能保持一致，并且只能使用完全刷新；例如物化视图的刷新方法属性曾经是 NEVER REFRESH
LAST_REFRESH_DATE	VARCHAR(128)	最后一次刷新时间，需要注意的是对于从来没有刷新过的物化视图，其取值是’1900-1-1 0:0:0’

第 8 章 嵌入式 SQL

DM_SQL 尽管功能强，使用方便灵活，但由于它本身没有过程性结构，大多数语言都是独立执行、与上下文无关，很难满足大型管理信息系统的需要。为了解决这一问题，DM_SQL 语言提供了两种工作方式：交互方式和嵌入方式。这样一来，既发挥了高级语言数据类型丰富、处理方便灵活的优势，又以 DM_SQL 语言弥补了高级语言难以描述数据库操作的不足，从而为用户提供了建立大型管理信息系统和处理复杂事务所需要的工作环境。在这种方式下使用的 DM_SQL 语言称为 DM 嵌入式 SQL，而 DM 嵌入式 SQL 的高级语言称为主语言或宿主语言。

嵌入在主语言程序中的 DM_SQL 语句并不能直接被主语言编译程序所识别，必须对这些 SQL 语句进行预处理，将其翻译成主语言语句，生成由主语言语句组成的目标文件，然后再由编译程序编译成可执行文件。为了能够识别嵌入在主语言中的 SQL 语句，必须按一定的规则来编写嵌入式 SQL 代码。

1．以对主语言合适的 SQL 前缀开始每一个 SQL 语句；

2．以对主语言合适的 SQL 终结符结束每一个 SQL 语句；

3．说明所有的将在一些特殊 DECLARE 段与 SQL 共享的主语言程序变量；

4．说明用于错误处理的附加程序变量。

在本章各例中，如不特别说明，各例均使用实例库 BOOKSHOP，用户均为建表者 SYSDBA。

8.1 SQL 前缀和终结符

嵌入式 SQL 语句必须具有一个前缀和一个终结符。DM 嵌入式 SQL 语句的前缀是 EXEC SQL，而终结符是分号。在 EXEC SQL 和分号之间必须是有效的 SQL，不能有主语言语句及其注释。DM 嵌入式 SQL 前缀语法如下：

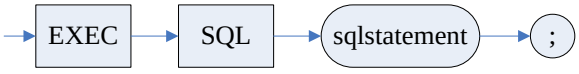
语法格式

```
EXEC SQL <SQL 语句>;
```

参数

<SQL 语句>　指明使用的嵌入式 DM_SQL 语句。

图例



使用说明

1．该语句只能在嵌入式方式中使用。EXEC 和 SQL 必须在一起出现，不能在分开的行中；

2．DM 嵌入式 SQL 语句包括说明性嵌入式 SQL 语句和可执行嵌入式 SQL 语句两类。说明性语句包括：嵌入变量声明节的开始和结束语句、异常声明语句和游标声明语句。除此之外，其它的数据操纵语句皆为可执行的 SQL 语句，可执行语句又分为数据定义、数据控制、数据操纵三种；

3．除特别声明的，在嵌入方式下有不同语法的语句(如 SELECT 语句)以外，所有本手册中定义的语句都可以作为 DM 嵌入式 SQL 语句。

8.2 宿主变量

嵌入式 SQL 可以在任何可放置标量表达式的地方含有宿主语言变量。宿主变量可用来在程序和 SQL 数据之间传送数据。在 SQL 语句中，<宿主变量名>必须前置一个冒号以同 SQL 对象名相区别。宿主变量只能返回标量值。

声明节是任何嵌入式 SQL 程序的一个重要部分。变量说明出现在 EXEC SQL BEGIN DECLARE SECTION 和 EXEC SQL END DECLARE SECTION 之间。声明节语法如下：

语法格式

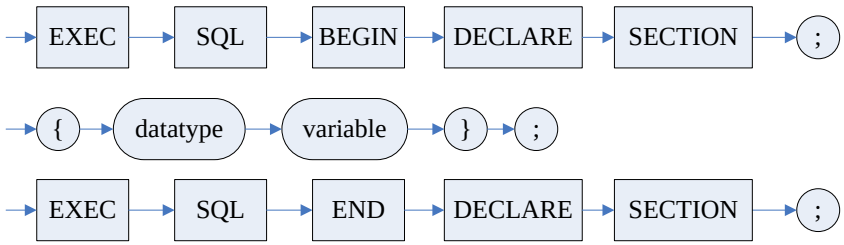
```
EXEC SQL BEGIN DECLARE SECTION;
{<宿主变量定义语句>;}
EXEC SQL END DECLARE SECTION;
<宿主变量定义语句> ::= <宿主变量数据类型> <宿主变量名>
```

参数

1．<宿主变量名>　指明在 DM 嵌入式 SQL 中使用的宿主变量的名称；

2．<宿主变量数据类型>　指明在 DM 嵌入式 SQL 中使用的宿主变量的数据类型。

图例



使用说明

1．该语句只能在嵌入式方式中使用；

2．在声明节中定义的变量可以被 C 语句使用，而不必要在声明节之外重新定义这些变量，否则会引起变量重复定义的错误；

3．在 DM_SQL 中所使用的宿主变量均必须在 DM 嵌入式 SQL 声明节中加以说明；

4．一个程序模块中可以出现多个声明节，各声明节可出现在 C 程序的说明语句可出现的位置上。PROC*C 规定，一个模块中所有的声明节中的宿主变量不得同名，而不论其实质是全局变量还是局部变量。预编译时，相应声明节的 C 语句是去掉了“EXEC SQL BEGIN DECLARE SECTION”和“EXEC SQL END DECLARE SECTION”两个语句后的变量定义语句，其位置顺序均保持不变。

举例说明

例　定义宿主变量。

```
EXEC SQL BEGIN DECLARE SECTION;
INT       num;
CHAR    string[20];
EXEC SQL END DECLARE SECTION;
```

该例中 num 和 string 是宿主变量。它们根据宿主语言(在此是 C)的规定定义。预编译器要求知道每个宿主变量的数据类型、大小和名称。

8.2.1 输入和输出变量

宿主变量分为输入宿主变量和输出宿主变量两种。输入和输出都是从 DBMS 的角度而言的，“输入”指输入到 DBMS，而“输出”指从 DBMS 输出。找到嵌入式 SQL 程序中的

宿主变量很容易，因为它们的名称前面都带有冒号。语法格式为：

```
: <宿主变量>
```

如下例所示：

```
EXEC SQL CREATE TABLE T1 ( C1 INT, C2 CHAR(20) );
num = 1;
strcpy(string, “1234”);
EXEC SQL INSERT INTO T1 VALUES ( :num, :string );
```

注意：宿主变量的数据类型必须与 SQL 语句中的列类型相容。

8.2.2 指示符变量

为了能够处理 NULL 值，在主语言中引入了指示符变量。它是一个跟在宿主变量后的数字变量，标记该宿主变量的数据是否为空。在 DM 嵌入式 SQL 语句中，指示符语法如下：

```
: <宿主变量> [:<指示符名>]
```

如下例所示：

```
EXEC SQL BEGIN DECLARE SECTION;
… … /*其他宿主变量的定义*/
short str_indicator;
EXEC SQL END DECLARE SECTION;
… …
str_indicator = -1;
EXEC SQL INSERT INTO T1 VALUES ( :num, :string :str_indicator);
/* 插入结果：对 T1 的 C2 列插入一个 NULL 值 */
```

对于输入宿主变量，预置指示符变量的值为-1，表明插入一个空值。

对于输出宿主变量，指示符表示返回结果是否为空或有截取的情况。一个指示变量的取值有三种：取值为 0，说明返回值非空且赋给宿主变量时不发生截取；取值为-1，说明返回的值为空值；取值>0，说明返回的值为非空值，但在赋给宿主变量时，字符串的值被截断，这时指示变量的值为截断之前的长度。

8.3 服务器登录与退出

在 DM 嵌入方式下，用户与 DBMS 进行交互前，必须先用命令 LOGIN 登录 DM 服务器。在与服务器交互结束后，用 LOGOUT 命令与服务器断开连接。

8.3.1 登录服务器

嵌入方式下，供用户向系统报告自己的登录名和口令，以便系统检查是否为合法用户。

语法格式

```
LOGIN <嵌入式变量 1>  PASSWORD <嵌入式变量 2>  SERVER <嵌入式变量 3>;
```

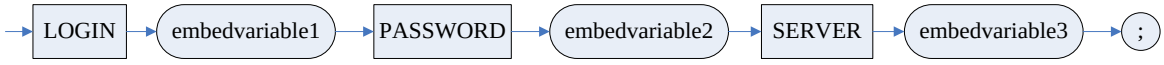
参数

1．<嵌入式变量 1> 指明登录系统的用户使用的用户的名称，该用户当前须具有数据库级权限；

2．<嵌入式变量 2> 指明与登录系统的用户名对应的口令；

3．<嵌入式变量 3> 指明所登录系统服务器的主机名。

图例



使用说明

1．该语句只能在嵌入式方式中使用；

2．<嵌入式变量 1>、<嵌入式变量 2>和<嵌入式变量 3>必须是在嵌入程序说明节中说明过的字符串变量；

3．嵌入式变量 1、嵌入式变量 2 为长度不超过 129 的字符串指针(在 DM 中登录名和密码最多为 128 个字符)；

4．在用户程序中，登录语句的逻辑位置必须先于所有可执行的 SQL 语句，只有执行登录语句之后，才能执行其它可执行的 SQL 语句。

8.3.2 退出服务器

嵌入方式下，用户用来断开与数据库的连接。

语法格式

```
LOGOUT;
```

使用说明

1．该语句只能在嵌入式方式中使用；

2．在程序结束时，需要执行 LOGOUT 语句，以断开与服务器的连接，回收服务器的资源。

举例说明

例 在 PRO * C 嵌入式环境中，用户 SYSDBA 登录到服务器 DMSERVER 上。

```
#include <stdio.h>
#include <string.h>
EXEC SQL BEGIN DECLARE SECTION;
        char LOGINNAME[20] = "SYSDBA";
        char USERPWD[20] = "SYSDBA";
        char APPSRV[20] = "DMSERVER";
EXEC SQL END DECLARE SECTION;
long    SQLCODE;
dm_hdbc    dm_con;
dm_hdiag   dm_diag;
void    main( )
{
EXEC SQL LOGIN :LOGINNAME PASSWORD :USERPWD SERVER :APPSRV;
if (SQLCODE != 0)
{
        printf("用户%s 登录失败\n\n", LOGINNAME);
        return ;
}
else
        printf("用户%s 登录成功\n\n", LOGINNAME);
EXEC SQL LOGOUT;
printf("用户%s 断开成功\n\n", LOGINNAME);
}
```

8.4 游标的定义与操纵

由于 DM_SQL 语言是面向集合的语言，一条 SQL 语句可以产生或处理多条记录，而高级语言是面向记录的，一组主变量一次只能存放一条记录，所以仅使用主变量并不能满足 SQL 语句向应用程序输出数据的要求。为解决这一问题，SQL 语言引用了游标来协调这两种不同的工作方式，从而为应用程序逐个处理查询结果提供了强有力的手段。DM_SQL 语言提供了四条有关游标的语句：定义游标语句、打开游标语句、拨动游标语句和关闭游标语

句。

使用游标必须先定义，定义游标实际上是定义了一个游标工作区，并给该工作区分配了一个指定名字的指针(即游标)。游标工作区用以存放满足查询条件行的集合，因此它是一说明性语句，是不可执行的。在打开游标时，就可从指定的基表中取出所有满足查询条件的行送入游标工作区并根据需要分组排序，同时将游标置于第一行的前面以备读出该工作区中的数据。当对行集合操作结束后，应关闭游标，释放与游标有关的资源。下面分述这几个语句的使用格式及使用中的注意事项。

8.4.1 定义游标语句

定义一个游标，给它一个名称和与之相联系的 SELECT 语句。

完整定义游标的语法格式为：

```
格式一：
DECLARE
  CURSOR 游标名 IS|FOR SELECT 语句;
    BEGIN
        OPEN 游标名;
        拨动游标语句;
        关闭游标;
    END;
格式二：
DECLARE
  游标名 CURSOR;|CURSOR 游标名;
    BEGIN
        OPEN 游标名 FOR SELECT 语句;
        拨动游标语句;
        关闭游标;
    END;
```

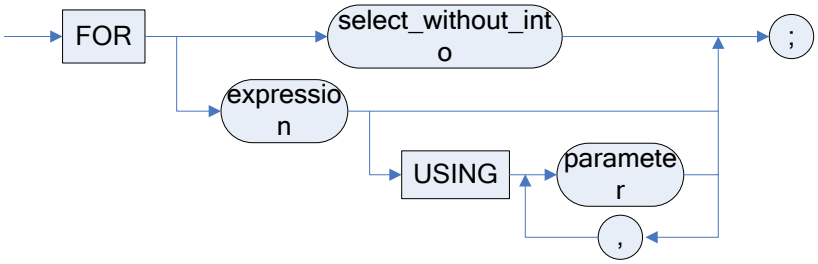
参数

1．<游标名>　　指明被定义的游标的名称；

2．<查询说明>　　指明对应于被定义的游标的 SELECT 语句。

其中，SELECT 语句图例如下：

图例



使用说明

1．该语句只能在嵌入方式或过程中使用；

2.<查询说明>的语法请参考 SELECT 语句的语法，并且该 SELECT 语句不能包括 INTO 子句；

3．用户必须在其他的嵌入式 SQL 语句引用该游标之前定义它。游标说明的范围在整个预编译单元内，并且每一个游标的名字必须在此范围内唯一。

举例说明

例　定义—游标，游标名为 CHEAP_BOOK，用来查询现在销售价格低于 15 元的图书的

名称、作者、出版社和现在销售价格信息。

```
DECLARE  CURSOR  CHEAP_BOOK  FOR
    SELECT  NAME, AUTHOR, PUBLISHER, NOWPRICE
    FROM    PRODUCTION.PRODUCT
    WHERE   NOWPRICE < 15.0000;
```

该语句定义了名字为 CHEAP_BOOK 的游标，该游标要从 PRODUCT 表中查出价格低于 15 元的图书的名称、作者、出版社和现在销售价格信息。定义了存取这个行集合的工作区及取数指针 CHEAP_BOOK。

8.4.2 打开游标语句

打开一个已经定义过的游标。

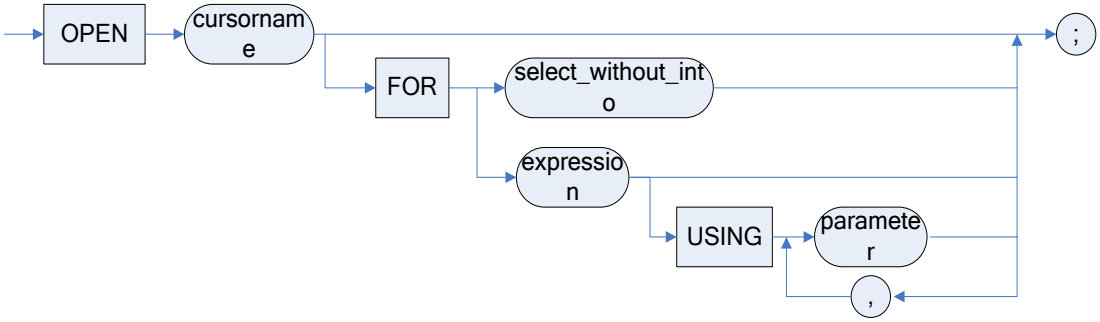
语法格式

```
OPEN <游标名>
或
OPEN <游标变量> FOR <不带 INTO 查询表达式> | <表达式子句>;
<表达式子句>::=<表达式> [USING <绑定参数> {, <绑定参数>}]
```

参数

<游标名>　指明被打开的游标的名称。

图例



使用说明

1．打开游标必须是打开已定义过的游标；

2．该语句的执行将根据定义游标语句中的查询说明所指出的条件，从表中取出符合条件的行进入游标工作区，然后将游标置于第一行之前的位置。

举例说明

例　打开 8.4.1 节例中定义的游标 CHEAP_BOOK。

```
OPEN  CHEAP_BOOK;
```

系统执行该语句，则将查询结果送入游标工作区，并将游标置于第一行之前，这时游标工作区的状态如下表 8.4.1 所示。

CHEAP_BOOK→　　表 8.4.1

NAME	AUTHOR	PUBLISHER	NOWPRICE
水浒传	施耐庵，罗贯中	中华书局	14.3000
老人与海	海明威	上海出版社	6.1000
工作中无小事	陈满麒	机械工业出版社	11.4000
突破英文基础词汇	刘毅	外语教学与研究出版社	11.1000

8.4.3 拨动游标语句

使游标移动到指定的一行，若游标名后跟随有 INTO 子句，则将游标当前指示行的内容取出分别送入 INTO 后的各变量中。

语法格式

```
FETCH [[NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n] [FROM] ] <游标名>
[INTO <赋值对象>{,<赋值对象>>}];
<赋值对象>>::=<变量名> | <参数> | <伪列>
<伪列>::=:OLD|NEW|<别名>.列名
```

参数

1．NEXT　　游标下移一行；

2．PRIOR　　游标前移一行；

3．FIRST　　游标移动到第一行；

4．LAST　　游标移动到最后一行；

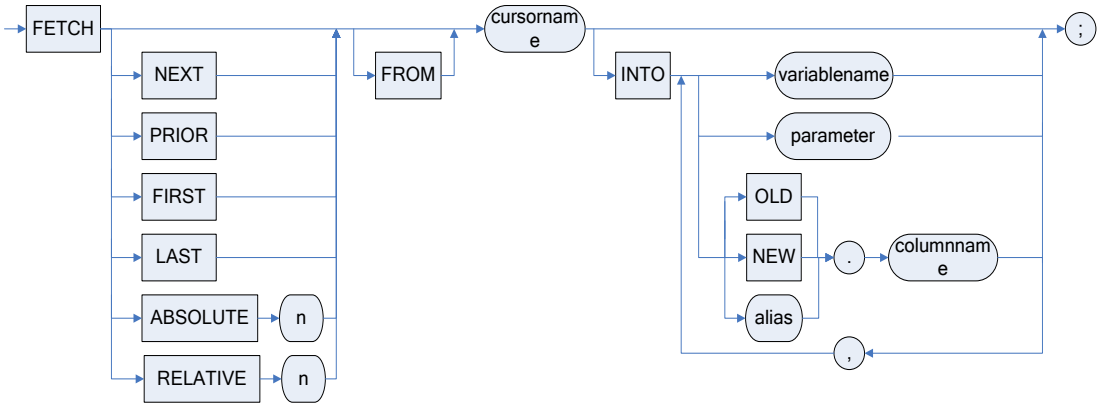
5．ABSOLUTE n　游标移动到第 n 行；

6．RELATIVE n　游标移动到当前指示行后的第 n 行；

7．<游标名>　　指明被拨动的游标的名称；

8．<别名>　　指的是 OLD 和 NEW 的别名(别名的指定请参看触发器)。

图例



使用说明

1．待拨动的游标必须是已打开过的游标；

2．DM 通过增加的[NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n]属性设置，可以方便地将游标拨动到结果集任意位置获得数据。游标拨动的默认属性为 NEXT。但这些增强属性只能在过程中使用；

3．当游标被打开后，若不指定游标的移动位置，第一次执行 FETCH 语句时，游标下移，指向工作区中的第一行，以后每执行一次 FETCH 语句，游标均顺序下移一行，使这一行成为当前行；

4．INTO 后的变量个数、类型必须与定义游标语句中、SELECT 后各值表达式的个数、类型一一对应；

5．在 PROC*C 中进行游标查询时，当返回值的数据类型与宿主变量的数据类型不一致时，DM 系统将返回值转换成宿主变量的类型。这种转换只局限于数值转换。不论数据类型如何，如果返回给宿主变量的值是 NULL，那么相应指示符变量被置为-1。如果没有与之相应的指示符变量，那么 SQLCODE 被设置为-5000。数值型数据转换，包括 short、int、double、float 四种数值型数据的转换，若发生溢出错误，将给出警告信息；

6．如果当前游标已经指向查询的最后一条记录，在 PROC*C 中使用 FETCH 语句将会导致返回错误代码(SQLCODE=100)；若是在存储过程中使用，将返回一个状态码告知用户已经到达最后一条记录。

举例说明

例 1 对于 8.4.2 节中刚打开的游标，当执行以下语句两次后，INTO 后各变量的值为多少(设 INTO 后的主变量与定义游标语句中 SELECT 后的值表达式在个数，类型上均一一对应)。

```
FETCH  CHEAP_BOOK  INTO  A, B, C, D;
```

由于是两次连续执行 FETCH 语句且结果均放在变量 A、B、C、D 中，这样，第一次的结果已被第二次的结果冲掉，最后的结果为：

A='老人与海'；B='海明威'；C='上海出版社'；D=6.1000

该例说明：当需要连续取出工作区的多行数据时，应将 FETCH 语句置入高级语言的循环结构中。

例 2 当设置 ABSOLUTE 属性时，n 值是从 1 开始的。如在 8.4.2 节中的游标，执行如下语句：

```
FETCH ABSOLUTE 3 CHEAP_BOOK  INTO A, B, C, D;
```

结果为：

A='突破英文基础词汇'; B='刘毅'; C='外语教学与研究出版社'； D=11.1000

如果执行：

```
FETCH ABSOLUTE 0 CHEAP_BOOK  INTO :A, :B, :C, :D;
```

则游标的 NOTFOUND 将被置为 TRUE。

8.4.4 关闭游标语句

关闭指定的游标，收回它所占的资源。

语法格式

```
CLOSE  <游标名>;
```

参数

<游标名> 指明被关闭的游标的名称。

使用说明

1．该语句只能在嵌入方式或过程中使用；

2．待关闭的游标必须是已打开过的游标；

3．游标一旦关闭，就不能再对该游标使用 FETCH 语句，否则应重新打开游标。

举例说明

例 要关闭前面已打开的 CHEAP_BOOK 游标，应使用以下语句：

```
CLOSE  CHEAP_BOOK;
```

8.4.5 关于可更新游标

在嵌入方式或过程中，通过游标对基表进行修改和删除时要求该游标表必须是可更新的。可更新游标的条件是：游标定义中给出的查询说明必须是可更新的。DM 系统对查询说明是可更新的有这样的规定：

1. 查询说明的 FROM 后只带一个表名，且该表必须是基表或者是可更新视图；
2. 查询说明是单个基表或单个可更新视图的行列子集，SELECT 后的每个值表达式只能是单纯的列名，如果基表上有聚集索引键，则必须包含所有聚集索引键；

3. 查询说明不能带 GROUP BY 子句、HAVING 子句、ORDER BY 子句；

4. 查询说明不能嵌套子查询。

不满足以上条件的游标表是不可更新的，其实，可更新游标的条件与可更新视图的条件是一致的。8.4.1 节例子中定义的游标 CHEAP_BOOK 就是一个可更新游标。

8.4.6 游标定位删除语句

DM 系统除了提供一般的数据删除语句外，还提供了游标定位删除语句。

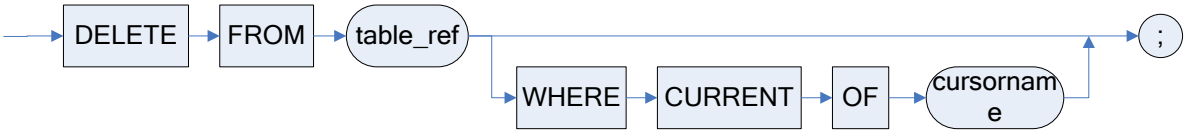
语法格式

```
DELETE FROM <表引用> [WHERE CURRENT OF <游标名>];
<表引用>::= [<模式名>.] <基表或视图名> | <外部连接表>
<基表或视图名>::=<基表名> | <视图名>
```

参数

1. <模式名>　　指明游标所对应表或视图所属模式名，缺省为当前模式名；
2. <基表名>　　指明游标所对应表名；
3. <视图名>　　指明游标所对应视图名；
4. <游标名>　　指明使用的游标的名称。

图例



使用说明

1. 语句中的游标在程序里已定义并被打开；
2. 指定的游标表应是可更新的；
3. 该基表应是游标定义中第一个 FROM 子句中所标识的表；
4. 游标结果集必须确定，否则 WHERE CURRENT OF <游标名>无法定位。

举例说明

例　已知游标的定义如 8.4.1 节定义的游标所示，用户的程序中有以下语句，请分析执行结果。

```
OPEN　CHEAP_BOOK;
FETCH　ABSOLUTE　2　CHEAP_BOOK;
DELETE　FROM　PRODUCTION.PRODUCT　WHERE　CURRENT　OF　CHEAP_BOOK;
```

用户是建表者，拥有该表上的所有权限。游标打开后，指针指在游标表的第一行的前面，游标工作区的状态如下表 8.4.1 所示。

执行 FETCH 语句后，使游标下移两行，再执行 DELETE 语句，删除了指针所指的第二行，游标顺序下移，最后的结果如下表 8.4.2 所示。

表 8.4.2

NAME	AUTHOR	PUBLISHER	NOWPRICE
水浒传	施耐庵，罗贯中	中华书局	14.3000
工作中无小事	陈满麒	机械工业出版社	11.4000

→

突破英文基础词汇	刘毅	外语教学与研究出版社	11.1000

8.4.7 游标定位修改语句

DM 系统除了提供一般的数据修改语句外，还提供了游标定位修改语句。

语法格式

```
UPDATE  <表引用>
SET <列名>=<值表达式>{,<列名>=<值表达式>}
[WHERE CURRENT OF <游标名>];
<表引用>::= [<模式名>.] <基表或视图名> | <外部连接表>
<基表或视图名>::=<基表名> | <视图名>
```

参数

1．<模式名>　　指明游标所对应表或视图所属模式名，缺省为当前模式名；

2．<基表名>　　指明游标所对应表名；

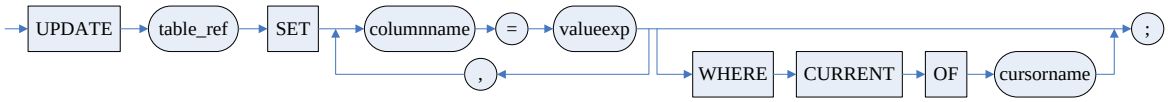
3．<视图名>　　指明游标所对应视图名；

4．<游标名>　　指明游标的名称；

5．<列名>　　指明表或视图中被更新列的名称，如果 SET 子句中省略列的名称，列的值保持不变；

6．<值表达式>　指明赋予相应列的新值。

图例



使用说明

1．语句中的游标在程序里应已被定义并打开；

2．指定的游标表应是可更新的；

3．该表应是游标定义中第一个 FROM 子句中所标识的表，所指的<列名>必须是表中的一个列，且不应在语句中多次出现；

4．语句中的值表达式不应包含集函数说明；

5．如果指定的表是可更新视图，其视图定义中使用了 WITH CHECK OPTION 子句，则该语句所给定的列值不应产生使视图定义中 WHERE 条件为假的行；

6．游标结果集必须确定，否则 WHERE CURRENT OF <游标名>无法定位。

举例说明

例　已知游标的定义如 8.4.1 节定义的游标，用户的程序中有以下语句，请分析执行结果。

```
OPEN  CHEAP_BOOK;
FETCH ABSOLUTE 3 CHEAP_BOOK;
UPDATE  PRODUCTION.PRODUCT
SET  NOWPIRCE=13.0000  WHERE  CURRENT  OF  CHEAP_BOOK;
```

用户是建表者，拥有该表上的所有权限。游标打开后，指针指在游标表的第一行的前面，再执行 FETCH 语句，游标下移三行，再执行 UPDATE 语句，将游标所指的第三行中的价格修改为 13.0000，最后的结果如下表 8.4.3 所示。

表 8.4.3

NAME	AUTHOR	PUBLISHER	NOWPRICE
水浒传	施耐庵，罗贯中	中华书局	14.3000
老人与海	海明威	上海出版社	6.1000
工作中无小事	陈满麒	机械工业出版社	11.4000
突破英文基础词汇	刘毅	外语教学与研究出版社	13.0000

8.5 单元组查询语句

游标为应用程序逐个处理 DM_SQL 语言产生的多条查询结果提供了强有力的手段，但当用户明知查询结果唯一时(如主关键字查询，查询结果或唯一或不存在)，仍使用游标处理会比较麻烦。为方便用户操作，DM_SQL 语言提供了单元组查询语句：从指定表中查询满足条件的一行，并将取到的数据赋给对应的变量。它的语法格式如下：

语法格式

```
SELECT [ALL | DISTINCT] <值表达式> {, <值表达式>}
        INTO <主变量名>{, <主变量名>}
        FROM   [<表引用> [[AS] <相关名>]
                  {,[<表引用> [[AS]<相关名>] }
        [WHERE <搜索条件>]
<表引用>::= [<模式名>.] <基表或视图名>
<基表或视图名>::=<基表名> | <视图名>
```

参数

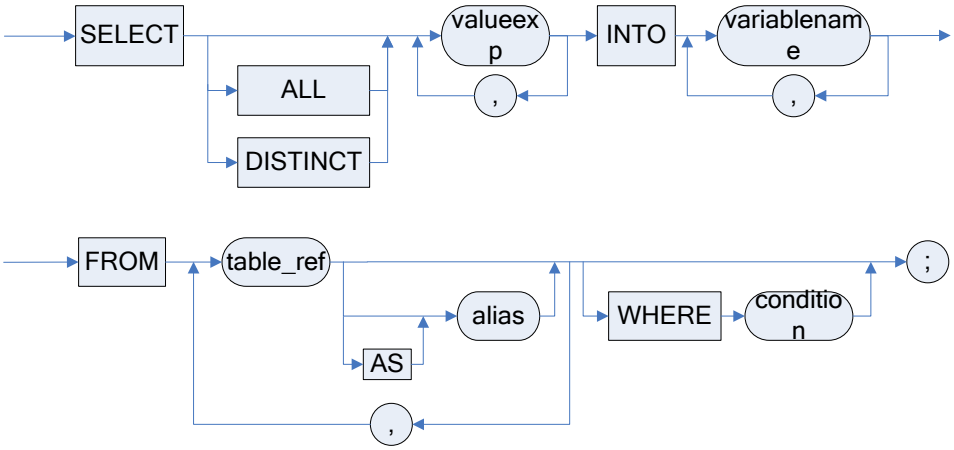
1．ALL　　返回所有被选择的行，包括所有重复的拷贝，缺省值为 ALL；

2．DISTINCT　从被选择出的具有重复行的每一组中仅返回一个这些行的拷贝；

3．<值表达式>　从在 FROM 子句中列出的表、视图中选择一表达式，通常是基于列的值。如果表、视图具有指定的用户名限定，则列要用用户名限定；

4．<主变量名>　指明存储数据的变量名。

图例



使用说明

1．用户对该语句中包含的每个<表名>应具有 SELECT 特权；

2．该语句中不应包含<GROUP BY 子句>或<HAVING 子句>，且不应出现分组视图；

3．INTO 后的主变量个数、类型必须与 SELECT 后<值表达式>的个数、类型一一对应；

4．WHERE 子句中的查询条件不得带集函数；

5．不处理多媒体数据类型；

6．如果没有行满足 WHERE 子句条件，则没有行能被获取并且 DM 返回一个错误码。

举例说明

例 分析以下语句的执行结果：

```
EXEC SQL BEGIN DECLARE SECTION;
char A[50];
char B[50];
double C;

EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL SELECT    NAME，AUTHOR，NOWPRICE
INTO       :A,:B,:C
FROM       PRODUCTION.PRODUCT
WHERE      PRODUCTID=2;
```

查询结果为：

```
A='水浒传'；  B='施耐庵，罗贯中'；  C=14.3000
```

例 分析以下语句的执行结果：

```
EXEC SQL BEGIN DECLARE SECTION;
……
int B;
EXEC SQL END DECLARE SECTION;
……
EXEC SQL SELECT    COUNT(*)
INTO       :B
FROM       PRODUCTION.PRODUCT
WHERE      PUBLISHER='中华书局';
```

查询结果为：

```
B=2。
```

8.6 动态 SQL

静态 SQL 语句提供的编程灵活性在许多情况下仍显得不足，有时候需要编写更为通用的程序，如当查询条件是不确定的，或者要查询的属性列也是不确定的，就无法用一条静态 SQL 语句实现了。因为在这些情况下，SQL 语句不能完整地写出来，而且这类语句在每次执行时都还有可能变化，只有在程序执行时才能构造完整。象这种在程序执行过程中临时生成的 SQL 语句叫动态 SQL 语句。

实际上，如果在预编译时下列信息不能确定，就必须使用动态 SQL 技术：

1．SQL 语句正文；

2．主变量个数；

3．主变量的数据类型；

4．SQL 语句中引用的数据库对象(例如：列、索引、基本表、视图等)。

动态 SQL 方法允许在程序运行过程中临时“组装”SQL 语句，主要有三种形式：

1．语句可变；

2．条件可变；

3．数据对象、查询条件均可变。

这几种动态形式几乎覆盖所有的可变要求。为了实现上述三种可变形式，在 DM 嵌入式 SQL 中提供了相应的语句：EXECUTE IMMEDIATE、PREPARE、EXECUTE。下面分别介绍这三种语句。

8.6.1 EXECUTE IMMEDIATE 立即执行语句

用立即执行语句执行动态 SQL 语句。

语法格式

```
EXECUTE IMMEDIATE <SQL 动态语句文本>;
[INTO <赋值对象> {,<赋值对象>}]
[USING <绑定参数> {,<绑定参数>}]
```

参数

1．<SQL 动态语句文本> 指明立即执行的动态语句文本；

2．<赋值对象>　用来存储查询语句返回的数据，可以是变量，也可以是 OUT 或 IN OUT 类型的参数，其个数与类型应与查询返回结果的各列相对应；

3．<绑定参数>　每个参数对应的实参，它可以是存储模块的参数或者变量，其个数与出现顺序应与形参对应。

图例

功能

动态地准备和执行一条语句。

使用说明

1．该语句首先分析动态语句文本，检查是否有错误，如果有错误则不执行它，并在 SQLCODE 中返回错误码；如果没发现错误则执行它；

2．SQL 动态语句中可以有参数，每个形参的形式为一个'?'符号。通过 USING 子句可以指定每个参数对应的实参值；

3．INTO 子句仅用于动态 SQL 语句为单行查询的情况；

4．用该方法处理的 SQL 语句一定不是 SELECT 语句，而且不包含任何虚拟的输入宿主变量；

5．一般来说，应该使用一个字符串变量来表示一个动态 SQL 语句的文本，下列语句不能作动态 SQL 语句：CLOSE、DECLARE、FETCH、OPEN、WHENEVER；

6．如果动态 SQL 语句的文本中有多条 SQL 语句，那么只执行第一条语句；

7．对于仅执行一次的动态语句，用立即执行语句较合适；

8．用户可以通过绑定不同的实参来重复执行一条动态 SQL 语句。但是要知道每次重复执行时，系统都会重新准备该语句后再执行，所以这样做并不能降低系统开销。

举例说明

例　假定变量 SQLSTMT 是声明节中定义的字符数组：

```
strcpy(SQLSTMT,"DELETE FROM SALES.SALESPERSON;");
EXEC SQL EXECUTE IMMEDIATE :SQLSTMT;
```

8.6.2 PREPARE 准备语句

嵌入方式下，为 EXECUTE 语句的执行准备一条语句。

语法格式

```
PREPARE <SQL 语句名> FROM <SQL 动态语句文本>;
```

参数

1．<SQL 语句名>　　指明被分析的动态语句文本的标识符；

2．<SQL 动态语句文本>　指明被分析的动态语句文本。

图例

使用说明

1．该语句只能在嵌入式方式中使用；

2．<SQL 语句名>标识被分析的动态 SQL 语句，它是供预编译程序使用的标识符，而不是宿主变量；

3．SQL 动态语句文本中的 SQL 语句，不能是 SELECT 语句；

4．该语句可能包含虚拟输入宿主变量(用问号表示)，而且变量的类型是已知的；

5. 如果 SQL 动态语句文本中含有多条 SQL 语句，那么只执行第一条 SQL 语句(第一条以分号结束的 SQL 语句)。

8.6.3 EXCUTE 执行语句

嵌入方式下，执行一个由 PREPARE 准备好的动态 SQL 语句。

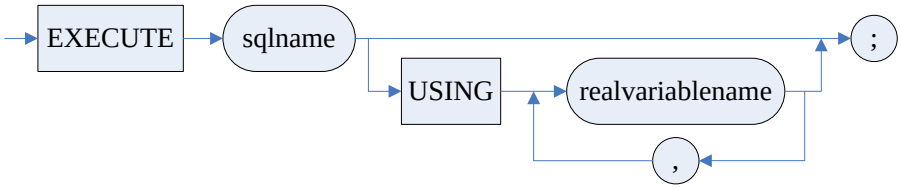
语法格式

```
EXECUTE <SQL 语句名> [<结果使用子句>];
<结果使用子句> ::= USING <实宿主变量名>{,<实宿主变量名>}
```

参数

1. <SQL 语句名>　　指明准备执行的动态语句文本的标识符；
2. <结果使用子句>　指出一个实宿主变量表，用于替换虚宿主变量；
3. <实宿主变量名>　指明用于替换虚宿主变量的相应的实宿主变量的名称。

图例



使用说明

1. 该语句只能在嵌入式方式中使用；
2. 该语句执行前，必须先使用 PREPARE 语句准备一个 SQL 语句并获得 SQL 语句名；
3. <结果使用子句>中的实宿主变量要与被分析的动态 SQL 语句中的虚宿主变量在类型、次序上相对应，个数相匹配。

举例说明

例 假定变量 shop_no 和 SQLSTMT 是声明节中已定义的字符数组，下面 4 条语句的功能相当于执行删除语句“DELETE FROM SALES.SALESPERSON WHERE SALESPERSONID= 2;”。

```
strcpy(salespersonid, "2");
strcpy(SQLSTMT, "DELETE FROM SALES.SALESPERSON WHERE SALESPERSONID=? ");
EXEC SQL PREPARE STMT FROM :SQLSTMT;
EXEC SQL EXECUTE STMT USING :salespersonid;
```

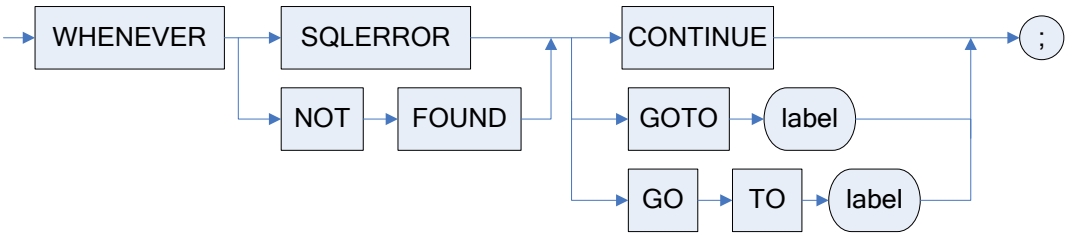
8.7 异常处理

在每个嵌入式 SQL 语句后，都可以加上一段代码来询问 DBMS 是否执行正确，DBMS 会返回 SQLSTATE 诊断信息码来告知用户是否出错及错误的类别。WHENEVER 语句可以用来说明在该异常申明处理语句的作用域内每个 SQL 操纵语句在发生操作异常时的处理方法。它的格式如下：

语法格式

```
WHENEVER SQLERROR | NOT FOUND
CONTINUE |
GOTO <标号> |
GO TO <标号>;
```

图例



参数

1．SQLERROR　指明当返回码 SQLCODE<0 即语句执行错误时，进行异常处理；

2．NOT FOUND　指明当返回码 SQLCODE=100 即查询语句没有结果返回时，进行异常处理；

3．CONTINUE　指明即使产生异常，也不需要作异常处理。其主要作用是取消前面与之具有相同条件的异常申明处理语句的作用；

4．GOTO 子句　指明程序转移到标号处继续执行。GOTO 可以书写为 GO TO；

5．<标号>　指明跳转的位置，它为标准的 C 语言的标号。

使用说明

1．该语句只能在嵌入式方式中使用；

2．一个 WHENEVER 语句的作用域是从该语句出现的位置开始，到下一个指明相同条件的 WHENEVER 语句出现(若没有下一个相同条件的 WHENEVER 语句，则到文件结束)之前的所有 SQL 语句。这种起始、终止位置是指源程序的物理位置，与该程序逻辑执行顺序无关。

举例说明

例 下面举例说明 WHENEVER 命令在 PROC*C 嵌入式 SQL 程序中的用法。

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
……
sql_error:
EXEC SQL WHENEVER SQLERROR CONTINUE;
```

第 9 章 函数

在值表达式中，除了可以使用常量、列名、集函数等之外，还可以使用函数作为组成成份。DM 中支持的函数分为数值函数、字符串函数、日期时间函数、空值判断函数、类型转换函数等。本手册还给出了 DM 部分系统函数的详细介绍。下列各表给出了函数的简要说明。在本章各例中，如不特别说明，各例均使用实例库 BOOKSHOP，用户均为建表者 SYSDBA。

表 9.1 数值函数

序号	函数名	功能简要说明
01	ABS(n)	求数值 n 的绝对值
02	ACOS(n)	求数值 n 的反余弦值
03	ASIN(n)	求数值 n 的反正弦值
04	ATAN(n)	求数值 n 的反正切值
05	ATAN2(n1,n2)	求数值 n1/n2 的反正切值
06	CEIL(n)	求大于或等于数值 n 的最小整数
07	CEILING(n)	求大于或等于数值 n 的最小整数，等价于 CEIL
08	COS(n)	求数值 n 的余弦值
09	COSH(n)	求数值 n 的双曲余弦值
10	COT(n)	求数值 n 的余切值
11	DEGREES(n)	求弧度 n 对应的角度值
12	EXP(n)	求数值 n 的自然指数
13	FLOOR(n)	求小于或等于数值 n 的最大整数
14	GREATEST(n1,n2,n3)	求 n1、n2 和 n3 中的最大浮点数
15	LEAST(n1,n2,n3)	求 n1、n2 和 n3 中的最小浮点数
16	LN(n)	求数值 n 的自然对数
17	LOG(n1[,n2])	求数值 n2 以 n1 为底数的对数
18	LOG10(n)	求数值 n 以 10 为底的对数
19	MOD(m,n)	求数值 m 被数值 n 除的余数
20	PI()	得到常数 π
21	POWER(n1,n2)	求数值 n2 以 n1 为基数的指数
22	RADIANS(n)	求角度 n 对应的弧度值
23	RAND([n])	求一个 0 到 1 之间的随机浮点数
24	ROUND(n[,m])	求四舍五入值函数
25	SIGN(n)	判断数值的数学符号
26	SIN(n)	求数值 n 的正弦值
27	SINH(n)	求数值 n 的双曲正弦值
28	SQRT(n)	求数值 n 的平方根
29	TAN(n)	求数值 n 的正切值
30	TANH(n)	求数值 n 的双曲正切值
31	TO_NUMBER (char [,fmt])	将 CHAR、VARCHAR、VARCHAR2 等类型的字符串

		转换为 DECIMAL 类型的数值
32	TRUNC(n[,m])	截取数值函数
33	TRUNCATE(n[,m])	截取数值函数，等价于 TRUNC
34	TO_CHAR(n [, fmt [, 'nls']])	将数值类型的数据转换为 VARCHAR 类型输出

表 9.2 字符串函数

序号	函数名	功能简要说明
01	ASCII(char)	返回字符对应的整数
02	BIT_LENGTH(char)	求字符串的位长度
03	CHAR(n)	返回整数 n 对应的字符
04	CHAR_LENGTH(char)/ CHARACTER_LENGTH(char)	求字符串的串长度
05	CHR(n)	返回整数 n 对应的字符,等价于 CHAR
06	CONCAT(char1,char2,char3,…)	顺序联结多个字符串成为一个字符串
07	DIFFERENCE(char1,char2)	以整数返回两个字符串的 SOUNDEX 值之差
08	INITCAP(char)	将字符串中单词的首字符转换成大写的字符
09	INS(char1,begin,n,char2)	删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置
10	INSERT(char1,n1,n2,char2) / INSSTR(char1,n1,n2,char2)	将字符串 char1 从 n1 的位置开始删除 n2 个字符，并将 char2 插入到 char1 中 n1 的位置
11	INSTR(char1,char2[,n,[m]])	从输入字符串 char1 的第 n 个字符开始查找字符串 char2 的第 m 次的出现，以字符计算
12	INSTRB(char1,char2[,n,[m]])	从输入字符串 char1 的第 n 个字符开始查找字符串 char2 的第 m 次的出现，以字节计算
13	LCASE(char)	将大写的字符串转换为小写的字符串
14	LEFT(char,n) / LEFTSTR(char,n)	返回字符串最左边的 n 个字符组成的字符串
15	LEN(char)	返回给定字符串表达式的字符(而不是字节)个数（汉字为一个字符），其中不包含尾随空格
16	LENGTH(char)	返回给定字符串表达式的字符(而不是字节)个数（汉字为一个字符），其中包含尾随空格
17	LENGTHB(char)/ OCTET_LENGTH(char)	返回输入字符串的字节数
18	LOCATE(char1,char2[,n])	返回 char1 在 char2 中首次出现的位置
19	LOWER(char)	将大写的字符串转换为小写的字符串
20	LPAD(char1,n,char2)	在输入字符串的左边填充上 char2 指定的字符，将其拉伸至 n 个字符长
21	LTRIM(char1,char2)	从输入字符串中删除所有的前导字符，这些前导字符由 char2 来定义
22	POSITION(char1,/ IN char2)	求串 1 在串 2 中第一次出现的位置
23	REPEAT(char,n) / REPEATSTR(char,n)	返回将字符串重复 n 次形成的字符串
24	REPLACE(char,search_string[,replacement_string])	将输入字符串中所有出现的 search_s tring 都替换成 replace_string 字符串

25	REPLICATE(char,times)	把字符串 char 自己复制 times 份
26	REVERSE(char)	将字符串反序
27	RIGHT / RIGHTSTR(char,n)	返回字符串最右边 n 个字符组成的字符串
28	RPAD(char1,n,char2)	类似 LPAD 函数，只是向右拉伸该字符串使之达到 n 个字符串长
29	RTRIM(char1,char2)	从输入字符串的右端开始删除 char2 参数中的字符
30	SOUNDEX(char)	返回一个表示字符串发音的字符串
31	SPACE(n)	返回一个包含 n 个空格的字符串
32	STRPOSDEC(char)	把字符串 char 中最后一个字符的值减一
33	STRPOSDEC(char,pos)	把字符串 char 中指定位置 pos 上的字符值减一
34	STRPOSINC(char)	把字符串 char 中最后一个字符的值加一
35	STRPOSINC(char,pos)	把字符串 char 中指定位置 pos 上的字符值加一
36	STUFF(char1,begin,n,char2)	删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置
37	SUBSTR(char,m,n) / SUBSTRING(char FROM m [FOR n])	返回 char 中从字符位置 m 开始的 n 个字符
38	SUBSTRB(char,n,m)	SUBSTR 函数等价的单字节形式
39	TO_CHAR(DATE[,fmt])	将日期数据类型 DATE 转换为一个在日期语法 fmt 中指定语法的 VARCHAR 类型字符串
40	TRANSLATE(char,from,to)	将所有出现在搜索字符集中的字符转换成字符集中的相应字符
41	TRIM([LEADING|TRAILING|BOTH] [exp] [] FROM char2])	删去字符串 char2 中由串 char1 指定的字符
42	UCASE(char)	将小写的字符串转换为大写的字符串
43	UPPER(char)	将小写的字符串转换为大写的字符串
44	REGEXP	根据符合 POSIX 标准的正则表达式进行字符串匹配
45	OVERLAY(char1 PLACING char2 FROM int [FOR int])	字符串覆盖函数，用 char2 覆盖 char1 中指定的子串，返回修改后的 char1
46	TEXT_EQU	返回两个 LONGVARCHAR 类型的值的比较结果，相同返回 1，否则返回 0
47	BLOB_EQU	返回两个 LONGVARBINARY 类型的值的比较结果，相同返回 1，否则返回 0
48	NLSSORT	返回对汉字排序的编码

表 9.3 日期时间函数

序号	**函数名**	**功能简要说明**
01	ADD_DAYS(date,n)	返回日期加上 n 天后的新日期
02	ADD_MONTHS(date,n)	在输入日期上加上指定的几个月返回一个新日期
03	ADD_WEEKS(date,n)	返回日期加上 n 个星期后的新日期
04	CURDATE()	返回系统当前日期
05	CURTIME()	返回系统当前时间
06	CURRENT_DATE()	返回系统当前日期

07	CURRENT_TIME(n)	返回系统当前时间
08	CURRENT_TIMESTAMP(n)	返回系统当前带会话时区信息的时间戳
09	DATEADD(datepart,n,date)	向指定的日期加上一段时间
10	DATEDIFF(datepart,date1,date2)	返回跨两个指定日期的日期和时间边界数
11	DATEPART(datepart,date)	返回代表日期的指定部分的整数
12	DAYNAME(date)	返回日期的星期名称
13	DAYOFMONTH(date)	返回日期为所在月份中的第几天
14	DAYOFWEEK(date)	返回日期为所在星期中的第几天
15	DAYOFYEAR(date)	返回日期为所在年中的第几天
16	DAYS_BETWEEN(date1,date2)	返回两个日期之间的天数
17	EXTRACT(时间字段 FROM date)	抽取日期时间或时间间隔类型中某一个字段的值
18	GETDATE()	返回系统当前时间戳
19	GREATEST(n1,n2,n3)	求 n1、n2 和 n3 中的最大日期
20	HOUR(time)	返回时间中的小时分量
21	LAST_DAY(date)	返回输入日期所在月份最后一天的日期
22	LEAST(n1,n2,n3)	求 n1、n2 和 n3 中的最小日期
23	MINUTE(time)	返回时间中的分钟分量
24	MONTH(date)	返回日期中的月份分量
25	MONTHNAME(date)	返回日期中月分量的名称
26	MONTHS_BETWEEN(date1,date2)	返回两个日期之间的月份数
27	NEXT_DAY(date1,char2)	返回输入日期指定若干天后的日期
28	NOW()	返回系统当前时间戳
29	QUARTER(date)	返回日期在所处年中的季节数
30	SECOND(time)	返回时间中的秒分量
31	ROUND (date1,char2)	把日期四舍五入到最接近格式元素指定的形式
32	TIMESTAMPADD(interval,n,timestamp)	返回时间 timestamp 加上 n 个 interval 类型时间间隔的结果
33	TIMESTAMPDIFF(interval,timeStamp1,timestamp2)	返回一个表明 timestamp2 与 timestamp1 之间的 interval 类型时间间隔的整数
34	SYSDATE()	返回系统的当前日期
35	TO_DATE(CHAR[,fmt])	字符串转换为日期数据类型
36	TRUNC(date[,format])	把日期截断到最接近格式元素指定的形式
37	WEEK(date)	返回日期为所在年中的第几周
38	WEEKDAY(date)	返回当前日期的星期值
39	WEEKS_BETWEEN(date1,date2)	返回两个日期之间相差周数
40	YEAR(date)	返回日期的年分量
41	YEARS_BETWEEN(date1,date2)	返回两个日期之间相差年数
42	LOCALTIME()	返回系统当前时间
43	LOCALTIMESTAMP()	返回系统当前时间戳
44	OVERLAPS	返回两两时间段是否存在重叠
45	TO_CHAR(DATE[,fmt])	将日期数据类型 DATE 转换为一个在日期语法 fmt 中指定语法的 VARCHAR 类型字符串。
46	TIMESTAMP()	返回系统当前带数据库时区信息的时间戳

表 9.4 空值判断函数

序号	函数名	功能简要说明
01	COALESCE(n1,n2,…nx)	返回第一个非空的值
02	IFNULL(n1,n2)	当 n1 为非空时，返回 n1；若 n1 为空，则返回 n2
03	ISNULL(n1,n2)	当 n1 为非空时，返回 n1；若 n1 为空，则返回 n2
04	NULLIF(n1,n2)	如果 n1=n2 返回 NULL，否则返回 n1
05	NVL(n1,n2)	返回第一个非空的值
06	NULL_EQU	返回两个类型相同的值的比较

表 9.5 类型转换函数

序号	函数名	功能简要说明
01	CAST(value AS 类型说明)	将 value 转换为指定的类型
02	CONVERT(类型说明,value)	将 value 转换为指定的类型
03	HEXTORAW(exp)	将 exp 转换为 BLOB 类型
04	RAWTOHEX(exp)	将 exp 转换为 VARCHAR 类型

表 9.6 杂类函数

序号	函数名	功能简要说明
01	DECODE(exp, search1, result1, … searchn, resultn [,default])	查表译码
02	ISDATE(exp)	判断表达式是否为有效的日期
03	ISNUMERIC(exp)	判断表达式是否为有效的数值

9.1 数值函数

数值函数接受数值参数并返回数值作为结果。

1．函数 ABS

语法格式：

```
ABS(n)
```

功能：返回 n 的绝对值。n 必须是数值类型。

例 查询现价小于 10 元或大于 20 元的信息。

```
SELECT PRODUCTID,NAME FROM PRODUCTION.PRODUCT
WHERE ABS(NOWPRICE-15)>5;
```

查询结果如下表 9.1.1 所示。

表 9.1.1

PRODUCTID	NAME
3	老人与海
4	射雕英雄传(全四册)
6	长征
7	数据结构(C 语言版)(附光盘)

10	噼里啪啦丛书(全 7 册)

2．函数 ACOS

语法格式：

```
ACOS(n)
```

功能：返回 n 的反余弦值。n 必须是数值类型，且取值在-1 到 1 之间，函数结果从 0 到 π。

例

```
SELECT acos(0);
```

查询结果为：1.5707963268E+000

3．函数 ASIN

语法格式：

```
ASIN(n)
```

功能：返回 n 的反正弦值。n 必须是数值类型，且取值在-1 到 1 之间，函数结果从-π/2 到 π/2。

例

```
SELECT asin(0);
```

查询结果为：0.0000000000E+000

4．函数 ATAN

语法格式：

```
ATAN(n)
```

功能：返回 n 的反正切值。n 必须是数值类型，取值可以是任意大小，函数结果从-π/2 到 π/2。

例

```
SELECT atan(1);
```

查询结果为：7.8539816340E-001

5．函数 ATAN2

语法格式：

```
ATAN2(n, m)
```

功能：返回 n/m 的反正切值。n,m 必须是数值类型，取值可以是任意大小，函数结果从-π/2 到 π/2。

例

```
SELECT atan2(0.2,0.3);
```

查询结果为：5.8800260355E-001

6．函数 CEIL

语法格式：

```
CEIL(n)
```

功能：返回大于等于 n 的最小整数。n 必须是数值类型。返回类型与 n 的类型相同。

例

```
SELECT    CEIL(15.6);
```

查询结果为：1.6000000000E+001

```
SELECT     CEIL(-16.23);
```

查询结果为：-16.0

7．函数 CEILING

语法格式：

```
CEILING(n)
```

功能：返回大于等于 n 的最小整数。等价于函数 CEIL()。

8．函数 COS

语法格式：

```
COS(n)
```

功能：返回 n 的余弦值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT cos(14.78);
```

查询结果为：-5.9946542619E-001

9．函数 COSH

语法格式：

```
COSH(n)
```

功能：返回 n 的双曲余弦值。

例

```
SELECT COSH(0)"Hyperbolic cosine of 0";
```

查询结果为：1.0000000000E+000

10．函数 COT

语法格式：

```
COT(n)
```

功能：返回 n 的余切值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT COT(20 * 3.1415926/180);
```

查询结果为：2.7474774704E+000

11．函数 DEGREES

语法格式：

```
DEGREES(n)
```

功能：返回弧度 n 对应的角度值，返回值类型与 n 的类型相同。

例

```
SELECT DEGREES(1.0);
```

查询结果为：5.7295779513E+001

12．函数 EXP

语法格式：

```
EXP(n)
```

功能：返回 e 的 n 次幂。

例

```
SELECT EXP(4) "e to the 4th power";
```

查询结果为：54.598150

13．函数 FLOOR

语法格式：

```
FLOOR(n)
```

功能：返回小于等于 n 的最大整数值。n 必须是数值类型。返回类型与 n 的类型相同。

例

```
SELECT    FLOOR(15.6);
```

查询结果为：15.0

```
SELECT    FLOOR(-16.23);
```

查询结果为：-17.0

14．函数 GREATEST

语法格式：

```
GREATEST(n1,n2,n3)
```

功能：求 n1、n2 和 n3 中的最大浮点数。

例

```
SELECT GREATEST(1.2,3.4,2.1);
```

查询结果：3.4

15．函数 LEAST

语法格式：

```
LEAST(n1,n2,n3)
```

功能：求n1、n2和n3中的最小浮点数。

例

```
SELECT LEAST(1.2,3.4,2.1);
```

查询结果：1.2

16．函数 LN

语法格式：

```
LN(n)
```

功能：返回 n 的自然对数。n 为数值类型，且大于 0。

例

```
SELECT ln(95) "Natural log of 95";
```

查询结果为：4.5538768916E+000

17．函数 LOG

语法格式：

```
LOG(m[,n])
```

功能：返回数值 n 以数值 m 为底的对数；若参数 m 省略，返回 n 的自然对数。m,n 为数值类型，m 大于 0 且不为 1。

例

```
SELECT LOG(10,100);
```

查询结果为：2.0000000000E+000

```
SELECT LOG(95);
```

查询结果为：4.5538768916E+000

18．函数 LOG10

语法格式：

```
LOG10(n)
```

功能：返回数值 n 以 10 为底的对数。n 为数值类型，且大于 0。

例

```
SELECT LOG10(100);
```

查询结果为：2.0000000000E+000

19．函数 MOD

语法格式：

```
MOD(m,n)
```

功能：返回 m 除以 n 的余数，当 n 为 0 时直接返回 m。m,n 为数值类型。

例

```
SELECT ROUND(NOWPRICE),mod(ROUND(NOWPRICE),10) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.1.2 所示。

表 9.1.2

ROUND(NOWPRICE)	mod(ROUND(NOWPRICE),10)
15	5
14	4
6	6
22	2
20	0
38	8
26	6
11	1
11	1
42	2

20．函数 PI

语法格式：

```
PI()
```

功能：返回常数 π。

例

```
SELECT PI();
```

查询结果为：3.1415926536E+000

21．函数 POWER

语法格式：

```
POWER(m,n)
```

功能：返回 m 的 n 次幂。m,n 为数值类型，如果 m 为负数的话，n 必须为一个整数。

例

```
SELECT POWER(3,2) "Raised";
```

查询结果为：9.0000000000 E +000

```
SELECT POWER(-3,3) "Raised";
```

查询结果为：-27.0000000000 E +001

22．函数 RADIANS()

语法格式：

```
RADIANS(n)
```

功能：返回角度 n 对应的弧度值，返回值类型与 n 的类型相同。

例

```
SELECT RADIANS(180.0);
```

查询结果为：3.1415926536 E+000

23．函数 RAND()

语法格式：

```
RAND([n])
```

功能：返回一个 0 到 1 之间的随机浮点数。n 为数值类型，为生成随机数的种子，当 n 省略时，系统自动生成随机数种子。

例

```
SELECT RAND();
```

查询结果为一个随机生成的小数

```
SELECT RAND(314);
```

查询结果为：3.2471694082E-002

24．函数 ROUND

语法格式：

```
ROUND(n [,m])
```

功能：返回四舍五入到小数点后面 m 位的 n 值。m 应为一个整数，缺省值为 0，m 为负整数则四舍五入到小数点的左边，m 为正整数则四舍五入到小数点的右边。若 m 为小数，系统将自动将其转换为整数。

例

```
SELECT NOWPRICE,ROUND(NOWPRICE) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.1.3 所示。

表 9.1.3

NOWPRICE	ROUND(NOWPRICE)
15.2000	15
14.3000	14
6.1000	6
21.7000	22
20.0000	20
37.7000	38
25.5000	26

11.4000	11
11.1000	11
42.0000	42

```
SELECT     ROUND(15.163,-1);
```

查询结果为：20.0

```
SELECT     ROUND(15.163);
```

查询结果为：15

25．函数 SIGN

语法格式：

```
SIGN(n)
```

功能：如果 n 为正数，SIGN(n)返回 1，如果 n 为负数，SIGN(n)返回-1，如果 n 为 0，SIGN(n)返回 0。

例

```
SELECT ROUND(NOWPRICE),SIGN(ROUND(NOWPRICE)-20) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.1.4 所示。

表 9.1.4

ROUND(NOWPRICE)	SIGN(ROUND(NOWPRICE)-20)
15	-1
14	-1
6	-1
22	1
20	0
38	1
26	1
11	-1
11	-1
42	1

26．函数 SIN

语法格式：

```
SIN(n)
```

功能：返回 n 的正弦值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT SIN(0);
```

查询结果为：0.0

27．函数 SINH

语法格式：

```
SINH(n)
```

功能：返回 n 的双曲正弦值。

例

```
SELECT SINH(1);
```

查询结果为：1.1752011936E+000

28．函数 SQRT

语法格式：

```
SQRT(n)
```

功能：返回 n 的平方根。n 为数值类型，且大于等于 0。

例

```
SELECT ROUND(NOWPRICE),SQRT (ROUND(NOWPRICE)) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.1.5 所示。

表 9.1.5

ROUND(NOWPRICE)	SQRT(ROUND(NOWPRICE))
15	3.872983346207417
14	3.7416573867739413
6	2.449489742783178
22	4.69041575982343
20	4.47213595499958
38	6.164414002968976
26	5.0990195135927845
11	3.3166247903554
11	3.3166247903554
42	6.48074069840786

29．函数 TAN

语法格式：

```
TAN(n)
```

功能：返回 n 的正切值。n 必须是数值类型，是用弧度表示的值。将角度乘以 π/180，可以转换为弧度值。

例

```
SELECT TAN(45*Pi()/180);
```

查询结果为：0.9999999999999999

30．函数 TANH

语法格式：

```
TANH(n)
```

功能：返回 n 的双曲正切值。

例

```
SELECT TANH(0);
```

查询结果为：0.0000000000E+000

31．函数 TO_NUMBER

语法格式：

```
TO_NUMBER (char [,fmt])
```

功能：将 CHAR、VARCHAR、VARCHAR2 等类型的字符串转换为 DECIMAL 类型的数值。char 为待转换的字符串，fmt 为目标格式串。

若指定了 fmt 格式则转换后的 char 应该遵循相应的数字格式，若没有指定则直接转换成 DECIMAL。fmt 格式串一定要包容实际字符串的数据的格式，否则报错。无格式转换中只支持小数点和正负号。合法的 fmt 格式串字符如下表：

表 9.1.6

元素	例子	说明
,(逗号)	9,999	指定位置处返回逗号 注意：1.逗号不能开头 2.不能在小数点右边
.(小数点)	99.99	指定位置处返回小数点
$	$9999	美元符号开头
0	0999 9990	以 0 开头，返回指定字符的数字 以 0 结尾，返回指定字符的数字
9	9999	返回指定字符的数字，如果不够正号以空格代替， 负号以-代替，0 开头也以空格代替。
D	99D99	返回小数点的指定位置，默认为’.’,格式串中最多能有一个 D
G	9G999	返回指定位置处的组分隔符，可有多个，但不能出现在小数点右边
S	S9999 9999S	负值前面返回一个-号 正值前面不返回任何值 负值后面返回一个-号 正值后面不返回任何值 只能在格式串首尾出现
X	XXXX xxxx	返回指定字符的十六进制值，如果不是整数则四舍五入到整数， 如果为负数则返回错误。
C	C9999	返回指定字符的数字
B	B9999	返回指定字符的数字

例 1 使用 9、G、D 来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('2,222.22', '9G999D99');
```

查询结果：2222.22

例 2 使用 9、,(逗号)、.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('2,222.22', '9,999.99');
```

查询结果：2222.22

例 3 使用$、9、,(逗号)、.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('$2,222.22','$9G999D99');
```

查询结果：2222.22

例 4 使用 S、9 和.(小数点)来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('-1212.12','S9999.99');
```

查询结果：-1212.12

例 5 使用 XXXX 来转换字符串'2,222.22'。

```
SELECT TO_NUMBER('1,234','XXXX');
```

查询结果：4660

例 6 无格式转换。

```
SELECT TO_NUMBER('-123.4');
```

查询结果：-1234

32. 函数 TRUNC

语法格式：

```
TRUNC(n [,m])
```

功能：将数值 n 的小数点后第 m 位以后的数全部截去。当数值参数 m 为负数时表示将数值 n 的小数点前的第 m 位截去。当数值参数 m 省略时，m 默认为 0。

特殊说明：当 m 为负数，其绝对值大于或等于 n 的整数位个数时，结果取 0；当 m 取正数，其值大于等于 n 的小数位个数时，结果取 n。

例

```
SELECT SQRT(NOWPRICE), TRUNC(SQRT (ROUND(NOWPRICE) ),1)
FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.1.7 所示。

表 9.1.7

SQRT (NOWPRICE)	TRUNC(SQRT (ROUND(NOWPRICE)),1)
3.8987177379235853	3.8
3.7815340802378077	3.7
2.4698178070456938	2.4
4.658325879540846	4.6
4.47213595499958	4.4
6.1400325732035	6.1
5.049752469181039	5.0
3.3763886032268267	3.3
3.331666249791536	3.3
6.48074069840786	6.4

```
SELECT     TRUNC(15.167,-1);
```

查询结果为：10.000

33. 函数 TRUNCATE

语法格式：

```
TRUNCATE(n [,m])
```

功能：等价于 TRUNC()。将数值 n 的小数点后第 m 位以后的数全部截去。当数值参数 m 为负数时表示将数值 n 的小数点前的第 m 位截去。当数值参数 m 省略时，m 默认为 0。

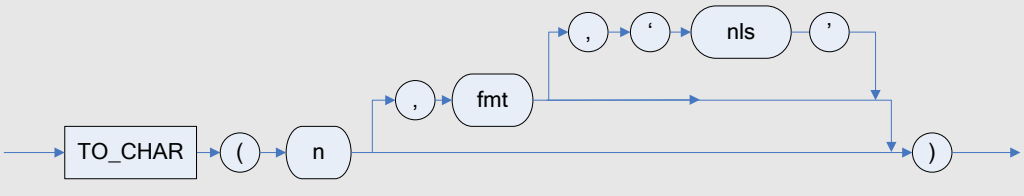
特殊说明：当 m 为负数，其绝对值大于或等于 n 的整数位个数时，结果取 0；当 m 取正数，其值大于等于 n 的小数位个数时，结果取 n。

34. 函数 TO_CHAR

语法格式：

```
TO_CHAR(n [, fmt [, 'nls' ] ])
```

图例



语句功能：

将数值类型的数据转化为 VARCHAR 类型输出。其中：n 为数值类型的数据；fmt 为目标格式串。DM 的缺省格式为数字的字符串本身。如 SELECT TO_CHAR(11.18)，查询结果为：11.18。

fmt 中包含的格式控制符主要可以分为三类，具体如下如下：

1) 主体标记；
2) 前缀标记；
3) 后缀标记。

其中主体标记包含的标记如表 9.1.8 所示。

表 9.1.8 主体标记

格式控制符	说明
逗号（,）	逗号只能出现在整数部分的任意位置，如 to_char(1234, '9,99,9')，结果为 1,23,4
点号（.）	作为小数点分隔符，不足的位数由后面的掩码决定
0	表示位数不足的时候用0填充，如to_char(1234, '09999.00')，结果为01234.00
D	作用同点号
G	作用同逗号
X	表示16进制
V	表示10的n次方
RN	转换为大写的罗马数字
rn	转换为小写的罗马数字

其中前缀标记包含的标记如表 9.1.9 所示。

表 9.1.9 前缀标记

格式控制符	说明
FM	去掉前置空格
$	美元符号。只能放在掩码最前面，且只能有一个
B	当整数部分的值为零时，返回空格
S	表示正负号， 如to_char(1234, 'S9999')结果为+1234， to_char(-1234,'S9999')结果为-1234
TM9	64个字符内返回原数值，超过则返回科学计数值
TME	返回科学计数值
C	当前货币名称缩写
L	当前货币符号

其中后缀标记包含的标记如表 9.1.10 所示。

表 9.1.10 后缀标记

格式控制符	说明
EEEE	科学计数符
MI	如'9999MI'，如果是负数，在尾部加上负号(-)；

	如果是正数，则尾部加上空格
PR	将负数放到尖括号<>中
C	当前货币名称缩写
L	当前货币符号
S	表示正负号

这些标记的组合规则主要包括以下几个：

1） 前缀之间的冲突；

2） 后缀与前缀之间的冲突；

3） 后缀之间的冲突。

其中，前缀之间的冲突如表 9.1.11 所示。

表 9.1.11 前缀之间的冲突

前缀	与指定前缀存在冲突的前缀
$	$, C, L
B	B
S	$, B, S, C, L
TM9	$, B, S
TME	$, B, S
FM	$, B, TM9, TME
C	C, L,$
L	C, L,$

注：前缀之间的冲突指上表中第二列的前缀不能放在第一列的前缀之前。

如当前缀为 S 时，前缀中不能还有$, B, S, C, L 标记，即$S，BS，SS，CS，LS 不能作为前缀。类似，对于前缀 L，则 CL，LL，$L 不能作为前缀。

后缀与前缀之间的冲突如表 9.1.12 所示。

表 9.1.12 后缀与前缀之间的冲突

后缀	与指定后缀存在冲突的前缀
EEEE	0
L	L, C, $
C	L, C, $
$	$, C, L, MI, PR
S	S
PR	S
MI	S

如当后缀为 C 时，前缀中不能还有 L, C, $等标记，如格式’L999C’等。

后缀之间的冲突如表 9.1.13 所示。

表 9.1.13 后缀之间的冲突

后缀	与指定后缀存在冲突的后缀
EEEE	S, EEEE, MI, PR
S	S, MI, PR
PR	S, MI, PR
MI	S, MI, PR
C	C, L, MI, PR, S, EEEE, $
L	C, L, MI, PR, S, EEEE, $
$	$, MI, PR, S, C, L

注：后缀之间的冲突指上表中第二列的后缀不能放在第一列的后缀之前。

如当后缀为 L 时，后缀中不能还有 C、L、MI、PR、S、EEEE、￥等标记，即后缀 CL，LL，MIL，PRL，SL，EEEEL 不能在格式字符串中出现。

nls 用来指定以下数字格式元素返回的字符：

1) 小数点字符；
2) 组分隔符；
3) 本地货币符号。
4) 国际货币符号。

这个参数可以有这种形式：

'NLS_NUMERIC_CHARACTERS = "dg"

NLS_CURRENCY = "text"

NLS_ISO_CURRENCY = territory '

其中： NLS_NUMERIC_CHARACTERS 参数指定字符 D 和 G 代表小数点字符和组分隔，必须用引号引起来。NLS_NUMERIC_CHARACTERS 串的长度只能是两个，并且这两个字符的不能相同；NLS_CURRENCY 指定的字符串用来代替本地货币符号，仅当 FMT 的前缀中有 L 时有效，不能超过 10 个字符的长度。NLS_ISO_CURRENCY 用来指定的字符串用来代替国际货币符号，仅当 FMT 的前缀中有 C 时有效，取值只能是表 9.1.14 中的值，得到的结果是缩写的内容。

表 9.1.14 NLS_ISO_CURRENCY 的值及缩写形式

NLS_TERRITORY	缩写
CHINA	CNY
TAIWAN	TWD
AMERICA	USD
UNITED KINGDOM	GBP
CANADA	CAD
FRANCE	EUR
GERMANY	EUR

ITALY	EUR
JAPAN	JPY
KOREA	KRW
BRAZIL	BRL
PORTUGAL	EUR

举例说明

```
SELECT TO_CHAR('01110' + 1);
```

查询结果为：1111

```
SELECT TO_CHAR(-10000,'L99G999D99MI') "Amount";
```

查询结果为：

Amount

￥10,000.00-

```
CREATE TABLE T_INT (C1 INT);
INSERT INTO T_INT VALUES(456),(0),(213);
SELECT TO_CHAR(C1, 'L999D99MI',
    'NLS_NUMERIC_CHARACTERS = ";:"
    NLS_CURRENCY = "AusDollars" ') FROM T_INT;
```

查询结果为：

AusDollars456;00

AusDollars;00

AusDollars213;00

```
 SELECT TO_CHAR(C1, 'C999D99MI',
    'NLS_NUMERIC_CHARACTERS = ";:"
    NLS_CURRENCY = "AusDollars"
    NLS_ISO_CURRENCY = "TAIWAN" ') FROM T_INT;
```

查询结果为：

TWD456;00

TWD;00

TWD213;00

9.2 字符串函数

字符串函数一般接受字符类型(包括 CHAR 和 VARCHAR)和数值类型的参数，返回值一般是字符类型或是数值类型。

1．函数 ASCII

语法格式：

```
ASCII(char)
```

功能：返回字符 char 对应的整数。

例

```
SELECT     ASCII('B') ,ASCII('中');
```

查询结果为：66 54992

2．函数 BIT_LENGTH

语法：BIT_LENGTH(char)

功能：返回字符串的位(bit)长度。

例

```
SELECT BIT_LENGTH('ab');
```

查询结果为：16

3．函数 CHAR

语法：CHAR(n)

功能：返回整数 n 对应的字符。

例

```
SELECT     CHAR(66),CHAR(67),CHAR(68) , CHAR(54992);
```

查询结果为：B C D 中

4．函数 CHAR_LENGTH / CHARACTER_LENGTH

语法：CHAR_LENGTH(char) 或 CHARACTER_LENGTH(char)

功能：返回字符串 char 的长度，以字符作为计算单位，一个汉字作为一个字符计算。字符串尾部的空格也计数。

例

```
SELECT    NAME,CHAR_LENGTH(TRIM(BOTH ' ' FROM NAME))
FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.2.1 所示。

表 9.2.1

NAME	CHAR_LENGTH(TRIM(BOTH ' ' FROM NAME))
红楼梦	3
水浒传	3
老人与海	4
射雕英雄传(全四册)	10
鲁迅文集(小说、散文、杂文)全两册	17
长征	2
数据结构(C 语言版)(附光盘)	15
工作中无小事	6
突破英文基础词汇	8
噼里啪啦丛书(全 7 册)	11

```
SELECT CHAR_LENGTH('我们');
```

查询结果为：2

5．函数 CHR

语法：CHR(n)

功能：返回整数 n 对应的字符。等价于 CHAR()。

6．函数 CONCAT

语法：CONCAT(char1,char2,char3…)

功能：返回多个字符串顺序联结成的一个字符串，该函数等价于连接符||。

例

```
SELECT PRODUCTID,NAME, PUBLISHER, CONCAT(PRODUCTID,NAME,PUBLISHER) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.2.2 所示：

表 9.2.2

PRODUCTID	NAME	PUBLISHER	CONCAT(PRODUCTID,NAME,PUBLISHER)
1	红楼梦	中华书局	1 红楼梦中华书局
2	水浒传	中华书局	2 水浒传中华书局
3	老人与海	上海出版社	3 老人与海上海出版社
4	射雕英雄传(全四册)	广州出版社	4 射雕英雄传(全四册)广州出版社
5	鲁迅文集(小说、散文、杂文)全两册		5 鲁迅文集(小说、散文、杂文)全两册
6	长征	人民文学出版社	6 长征人民出版社
7	数据结构(C 语言版)(附光盘)	清华大学出版社	7 数据结构(C 语言版)(附光盘)清华大学出版社
8	工作中无小事	机械工业出版社	8 工作中无小事机械工业出版社
9	突破英文基础词汇	外语教学与研究出版社	9 突破英文基础词汇外语教学与研究出版社
10	噼里啪啦丛书(全 7 册)	21 世纪出版社	10 噼里啪啦丛书(全 7 册)21 世纪出版社

7．函数 DIFFERENCE()

语法：DIFFERENCE(char1,char2)

功能：以整数返回两个字符串的 SOUNDEX 值之差。

例

```
SELECT DIFFERENCE('she', 'he');
```

查询结果为：3

8．函数 INITCAP

语法：INITCAP(char)

功能：返回句子字符串中，每一个单词的第一个字母改为大写，其他字母改为小写。单词用空格分隔，不是字母的字符不受影响。

例

```
SELECT INITCAP('hello world');
```

查询结果为：Hello World

9．函数 INS

语法： INS(char1,begin,n,char2)

功能：删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 串的 begin 所指位置。

例

```
SELECT INS ('abcdefg',1,3, 'kkk') ;
```

查询结果为：kkkdefg

10．函数 INSERT / INSSTR

语法： INSERT(char1,n1,n2,char2) / INSSTR(char1,n1,n2,char2)

功能：将字符串 char1 从 n1 的位置开始删除 n2 个字符，并将 char2 插入到 char1 中 n1 的位置。

例

```
SELECT  INSERT('That is a cake',2,3, 'his') ;
```

查询结果为：This is a cake

11．函数 INSTR

语法：INSTR(char1,char2[,n[,m]])

功能：返回 char1 中包含 char2 的特定位置。INSTR 从 char1 的左边开始搜索，开始位置是 n，如果 n 为负数，则搜索从 char1 的最右边开始，当搜索到 char2 的第 m 次出现时，返回所在位置。n 和 m 的缺省值都为 1，即返回 char1 中第一次出现 char2 的位置，这时与 POSITION 相类似。如果从 n 开始没有找到第 m 次出现的 char2，则返回 0。n 和 m 以字符作为计算单位，一个西文字符和一个汉字都作为一个字符计算。

此函数中 char1 可以是 CHAR、VARCHAR 和 TEXT 数据类型，char2 是 CHAR 或 VARCHAR 类型。n 和 m 是数值类型。

例

```
SELECT INSTR('CORPORATE FLOOR', 'OR', 3, 2) "Instring";
```

查询结果为：14

```
SELECT INSTR('我们的计算机', '计算机',1,1);
```

查询结果为：4

12．函数 INSTRB

语法：INSTRB(char1,char2[,n[,m]])

功能：返回 char1 中包含 char2 的特定位置。INSTRB 从 char1 的左边开始搜索，开始位置是 n，如果 n 为负数，则搜索从 char1 的最右边开始，当搜索到 char2 的第 m 次出现时，返回所在位置。n 和 m 的缺省值都为 1，即返回 char1 中第一次出现 char2 的位置，这时与 POSITION 相类似。如果从 n 开始没有找到第 m 次出现的 char2，则返回 0。以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

此函数 char1 可以是 CHAR,VARCHAR 和 TEXT 数据类型，char2 是 CHAR 或 VARCHAR 类型。n 和 m 是数值类型。

例

```
SELECT INSTRB('CORPORATE FLOOR', 'OR', 3, 2) "Instring";
```

查询结果为：14

```
SELECT INSTRB('我们的计算机', '计算机',1,1);
```

查询结果为：7

13．函数 LCASE

语法：LCASE(char)

功能：返回字符串中，所有字母改为小写，不是字母的字符不受影响。

例

```
SELECT LCASE('ABC');
```

查询结果为：abc

14．函数 LEFT / LEFTSTR

语法：LEFT(char,n) / LEFTSTR(char,n)

功能：返回字符串最左边的 n 个字符组成的字符串。

例

```
SELECT  NAME,LEFT(NAME,2)  FROM  PRODUCTION.PRODUCT;
```

查询结果如下表 9.2.3 所示。

表 9.2.3

NAME	LEFT(NAME,2)
红楼梦	红楼
水浒传	水浒
老人与海	老人
射雕英雄传(全四册)	射雕
鲁迅文集(小说、散文、杂文)全两册	鲁迅
长征	长征
数据结构(C 语言版)(附光盘)	数据
工作中无小事	工作
突破英文基础词汇	突破
噼里啪啦丛书(全 7 册)	噼里

```
SELECT  LEFT ('computer science',10);
```

查询结果为：computer s

15．函数 LEN

语法：LEN(char)

功能：返回给定字符串表达式的字符(而不是字节)个数，其中不包含尾随空格。

例

```
SELECT LEN ('hi,你好□□');
```

查询结果为：5

说明：□表示空格字符

16．函数 LENGTH

语法：LENGTH(char)

功能：返回给定字符串表达式的字符(而不是字节)个数，其中包含尾随空格。

例

```
SELECT LENGTH('hi,你好□□');
```

查询结果为：7

说明：□表示空格字符

17．函数 LENGTHB / OCTET_LENGTH

语法：LENGTHB(char)/ OCTET_LENGTH(char)

功能：返回字符串 char 的长度，以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

例

```
SELECT LENGTHB('大家好') "Length in bytes";
```

查询结果为：6

18．函数 LOCATE

语法：LOCATE(char1,char2[,n])

功能：返回字符串 char1 在 char2 中从位置 n 开始首次出现的位置，如果参数 n 省略或为负数，则从 char2 的最左边开始找。

例

```
SELECT LOCATE('man', 'The manager is a man', 10);
```

查询结果为：18

```
SELECT LOCATE('man', 'The manager is a man');
```

查询结果为：5

19．函数 LOWER

语法：LOWER(char)

功能：返回字符串中，所有字母改为小写，不是字母的字符不受影响。等价于 LCASE()。

20．函数 LPAD

语法：LPAD(char1,length[,char2])

功能：返回值为字符串 char1 左边增加 char2，总长度达到 length 的字符串，length 为正整数。如果未指定 char2，缺省值为空格。如果 length 的长度比 char1 大，则返回 char2 的前(length-length(char1))个字符+char1，总长度为 length。如果 length 比 char1 小，则返回 char1 的前 lengh 个字符。长度以字符作为计算单位，一个汉字作为一个字符计算。

注：若 length 为小于或等于零的整数，则返回 NULL。

例

```
SELECT LPAD(LPAD('FX',19,'Teacher'),22,'BIG') "LPAD example";
```

查询结果为：BIGTeacherTeacherTeaFX

```
SELECT LPAD('计算机',2, '我们的');
```

查询结果为：计算

```
SELECT LPAD('计算机',4, '我们的');
```

查询结果为：我计算机

21．函数 LTRIM

语法：LTRIM(char1[,set])

功能：删除字符串 char 左边起出现的 set 中的任何字符，当遇到不在 set 中的第一个字符时结果被返回。set 缺省为空格。

例

```
SELECT LTRIM('xyyxxxXxyLAST WORD', 'xy') "LTRIM example";
```

查询结果为：XxyLAST WORD

```
SELECT LTRIM('我们的计算机', '我们');
```

查询结果为：的计算机

22．函数 POSITION

语法：POSITION(char1 IN char2) / POSITION(char1, char2)

功能：返回在 char2 串中第一次出现的 char1 的位置，如果 char1 是一个零长度的字符串，POSITION 返回 1，如果 char2 中 char1 没有出现，则返回 0。以字节作为计算单位，一个汉字根据编码类型不同可能占据 2 个或多个字节。

例

```
SELECT  POSITION('数' IN '达梦数据库');
```

查询结果为：5

23．函数 REPEAT / REPEATSTR

语法：REPEAT(char,n) / REPEATSTR(char,n)

功能：返回将字符串重复 n 次形成的字符串。

例

```
SELECT REPEAT ('Hello ',3);
```

查询结果为：Hello Hello Hello

24．函数 REPLACE

语法：REPLACE(char, search [,replacement])

功能：REPLACE 的三个参数都是字符串类型。在字符串 char 中找到字符串 search，替换成 replacement。若 replacement 为空，则在 char 中删除所有 search。

例

```
SELECT  NAME,REPLACE(NAME, '地址', '地点') FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 9.2.4 所示。

表 9.2.4

NAME	REPLACE(NAME, '地址', '地点')
发货地址	发货地点
送货地址	送货地点
家庭地址	家庭地点
公司地址	公司地点

25．函数 REPLICATE(char,times)

语法：REPLICATE(char,times)

功能：把字符串 char 自己复制 times 份。

例

```
SELECT  REPLICATE('aaa',3);
```

查询结果为：aaaaaaaaa

26.函数 REVERSE

语法：reverse(char)

功能：将输入字符串的字符顺序反转后返回。

例：

```
SELECT REVERSE('abcd');
```

查询结果：dcba

27．函数 RIGHT / RIGHTSTR

语法：RIGHT(char,n) / RIGHTSTR(char,n)

功能：返回字符串最右边 n 个字符组成的字符串。

例

```
SELECT  NAME, RIGHT (NAME,2) FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 9.2.5 所示。

表 9.2.5

NAME	RIGHT (NAME,2
发货地址	地址
送货地址	地址
家庭地址	地址
公司地址	地址

```
SELECT RIGHTSTR('computer',3);
```

查询结果为：ter

28．函数 RPAD

```
语法：RPAD(char1,length[,char2])
```

功能：返回值为字符串 char1 右边增加 char2，总长度达到 length 的字符串，length 为正整数。如果未指定 char2，缺省值为空格。如果 length 的长度比 char1 大，则返回 char1+char2 的前(length-length(char1))个字符，总长度为 length。如果 length 比 char1 小，则返回 char1 的前 lengh 个字符。长度以字符作为计算单位，一个汉字作为一个字符计算。

注：若 length 为小于或等于零的整数，则返回 null。

例

```
SELECT RPAD('FUXIN',11, 'BigBig') "RPAD example";
```

查询结果为：FUXINBigBig

```
SELECT RPAD('计算机',2, '我们的');
```

查询结果为：计算

```
SELECT RPAD('计算机',4, '我们的');
```

查询结果为：计算机我

29．函数 RTRIM

```
语法：RTRIM(char1[,set])
```

功能：删除字符串 char1 右边起出现的 set 中的任何字符，当遇到不在 set 中的第一个字符时结果被返回。set 缺省为空格。

例

```
SELECT RTRIM('TURNERyXxxxyyyxy', 'xy') "RTRIM e.g.";
```

查询结果为：TURNERyX

```
SELECT RTRIM('我们的计算机','我计算机');
```

查询结果为：我们的

30．函数 SOUNDEX

```
语法：SOUNDEX(char)
```

功能：返回一个表示英文字符串发音的字符串，仅仅对英文字符串有效，对别的字符或数字返回 0000。

例

```
SELECT SOUNDEX('Hello');
```

查询结果为：H400

31．函数 SPACE

语法：SPACE(n)

功能：返回一个包含 n 个空格的字符串。

例

```
SELECT SPACE(5);
```

查询结果为：□□□□□

```
SELECT CONCAT(CONCAT('Hello',SPACE(3)), 'world');
```

查询结果为：Hello□□□world

说明：□表示空格字符

32．函数 STRPOSDEC

语法：STRPOSDEC(char)

功能：把字符串 char 中最后一个字符的值减一。

例

```
SELECT STRPOSDEC('hello');
```

查询结果为：helln

33．函数 STRPOSDEC

语法：STRPOSDEC(char,pos)

功能：把字符串 char 中指定位置 pos 上的字符的值减一。

例

```
SELECT STRPOSDEC('hello',3);
```

查询结果为：heklo

34．函数 STRPOSINC

语法：STRPOSINC(char)

功能：把字符串 char 中最后一个字符的值加一。

例

```
SELECT STRPOSINC ('hello');
```

查询结果为：hellp

35．函数 STRPOSINC

语法：STRPOSINC (char,pos)

功能：把字符串 char 中指定位置 pos 上的字符的值加一。

例

```
SELECT STRPOSINC ('hello',3);
```

查询结果为：hemlo

36．函数 STUFF

语法：STUFF(char1,begin,n,char2)

功能：删除在字符串 char1 中以 begin 参数所指位置开始的 n 个字符，再把 char2 插入到 char1 的 begin 所指位置。

例

```
SELECT STUFF('ABCDEFG',1,3, 'OOO');
```

查询结果为：OOODEFG

37．函数 SUBSTR

语法：SUBSTR(char[,m[,n]]) / SUBSTRING(char[from m [for n]])

功能：返回 char 中从字符位置 m 开始的 n 个字符。若 m 为 0，则把 m 就当作 1 对待。若 m 为正数，则返回的字符串是从左边到右边计算的；反之，返回的字符是从 char 的结尾向左边进行计算的。如果没有给出 n，则返回 char 中从字符位置 m 开始的后续子串。如果 n 小于 1，则返回 NULL。如果 m 和 n 都没有给出，返回 char。函数以字符作为计算单位，一个西文字符和一个汉字都作为一个字符计算。

例

```
SELECT NAME,SUBSTRING(NAME FROM 3 FOR 2) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.2.6 所示。

表 9.2.6

NAME	SUBSTRING(NAME FROM 3 FOR 2)
红楼梦	梦
水浒传	传
老人与海	与海
射雕英雄传(全四册)	英雄
鲁迅文集(小说、散文、杂文)全两册	文集
长征	
数据结构(C 语言版)(附光盘)	结构
工作中无小事	中无
突破英文基础词汇	英文
噼里啪啦丛书(全 7 册)	啪啦

```
SELECT SUBSTR('我们的计算机',3,4) "Subs";
```

查询结果为：的计算机

38．函数 SUBSTRB

语法：SUBSTRB(string,m[,n])

功能：返回 char 中从第 m 字节位置开始的 n 个字节长度的字符串。若 m 为 0，则 m 就当作 1 对待。若 m 为正数，则返回的字符串是从左边到右边计算的；若 m 为负数，返回的字符是从 char 的结尾向左边进行计算的。若 m 大于字符串的长度，则返回空串。如果没有 n，则缺省的长度为整个字符串的长度。如果 n 小于 1，则返回错误，提示参数非法。

这里假设字符串 string 的长度为 len，如果 n 的值很大，超过 len – m，则返回的子串的长度为 len – m。

如果开始位置 m 不是一个正常的字符的开始位置，那么返回的结果是 k 个空格（k 的值等于下一个有效字符的开始位置和 m 的差），空格后面是有效字符；如果字符串的 m+n-1 的位置不是一个有效的字符，那么就以空格填充。也就是不截断字符。

例

```
SELECT SUBSTRB('达梦数据库有限公司',4,15);
```

查询结果为：□数据库有限公司

说明：□表示空格字符，下同。

字符串前面是一个空格，这是因为字符串'达梦数据库有限公司'的第 4 个字节不是一个完整的字符的开始，因此用空格代替。

SELECT SUBSTRB('我们的计算机',3,4) "Subs", LENGTHB(SUBSTRB('我们的计算机',3,4));

查询结果为：们的 4

SELECT SUBSTRB('ABCDEFG',3,3) "Subs";

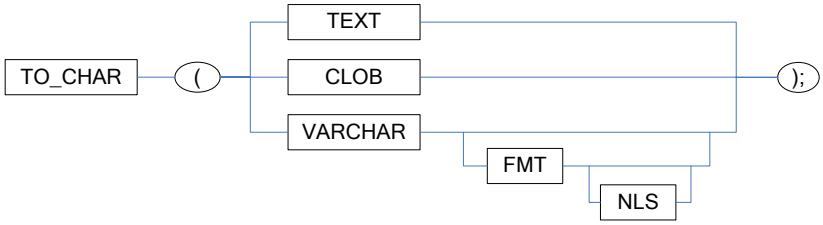
查询结果为：CDE

注意：函数 SUBSTRB 字节作为计算单位，一个字符在不同的编码方式下的字节长度是不同的。

39．函数 TO_CHAR

语法：TO_CHAR (character)

图例



功能：将 VARCHAR、CLOB、TEXT 类型的数据转化为 VARCHAR 类型输出。VARCHAR 类型的长度不能超过 8188 个字符，CLOB、TEXT 类型的长度不能超过 8187 个字符。

当参数类型为 VARCHAR 时，还可指定 FMT 与 NLS，FMT 和 NLS 的具体意义和限制可参见 9.1 节的 TO_CHAR 函数介绍。

例

```
select to_char('0110');
```

查询结果为：0110

```
create table t2(c1 varchar(4000));
insert into t2 values('达梦数据库有限公司成立于 2000 年，为国有控股的基础软件企业，专业从事数据库管理系统研发、销售和服务。其前身是华中科技大学数据库与多媒体研究所，是国内最早从事数据库管理系统研发的科研机构。达梦数据库为中国数据库标准委员会组长单位，得到了国家各级政府的强力支持。');
select to_char(c1) from t2;
```

查询结果为：达梦数据库有限公司成立于 2000 年，为国有控股的基础软件企业，专业从事数据库管理系统研发、销售和服务。其前身是华中科技大学数据库与多媒体研究所，是国内最早从事数据库管理系统研发的科研机构。达梦数据库为中国数据库标准委员会组长单位，得到了国家各级政府的强力支持。

```
select to_char('123','99,99','NLS_ISO_CURRENCY=CHINA');
```

查询结果为：1,23

40．函数 TRANSLATE

语法：TRANSLATE(char,from,to)

功能：TRANSLATE 是一个字符替换函数。char、from和to分别代表一字符串。对于char字符串，首先，查找char中是否含有from字符串，如果找到，则将其含有的from与to中的字符一一匹配，并用to中相应的字符替换，直至from中的字符全部替换完毕。to中的不足或多余的字符，均视为空值。

例

```
SELECT TRANSLATE('我们的计算机', '我们的', '大世界');
```

查询结果为：大世界计算机　　('我'将被'大'替代,'们'将被'世'替代,'的'将被'界'替代)

```
SELECT TRANSLATE('我们的计算机', '我们的', '世界');
```

查询结果为：世界计算机　　('我'将被'世'替代,'们'将被'界'替代,'的'对应的是空值，将被移走)

```
SELECT TRANSLATE('我们的计算机', '我们的', '大大世界');
```

查询结果为：大大世计算机　　('我'将被'大'替代,'们'将被'大'替代, '的'将被'世'替代，'界'对应的是空值，将被忽略)

例

```
SELECT NAME,TRANSLATE (NAME,'发货','送货') FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 9.2.7 所示。

表 9.2.7

NAME	TRANSLATE (NAME,'发货','送货')
发货地址	送货地址
送货地址	送货地址
家庭地址	家庭地址
公司地址	公司地址

41．函数 TRIM

```
语法：TRIM([LEADING|TRAILING|BOTH] [char1] FROM char2])
```

功能：TRIM 从 char2 的首端(LEADING)或末端(TRAILING)或两端(BOTH)删除 char1 字符，如果任何一个变量是 NULL，则返回 NULL。默认的修剪方向为 BOTH，默认的修剪字符为空格。

例

```
SELECT　NAME,TRIM(TRAILING '址' FROM NAME) FROM PERSON.ADDRESS_TYPE;
```

查询结果如下表 9.2.8 所示。

表 9.2.8

NAME	TRIM(TRAILING '址' FROM NAME)
发货地址	发货地
送货地址	送货地
家庭地址	家庭地
公司地址	公司地

```
SELECT　TRIM( '　 Hello World　 ');
```

查询结果为：Hello World

```
SELECT　TRIM(LEADING FROM '　 Hello World　 ');
```

查询结果为：Hello World□□□

说明：□表示空格字符，下同。

```
SELECT　TRIM(TRAILING FROM '　 Hello World　 ');
```

查询结果为：□□□Hello World

```
SELECT　TRIM(BOTH FROM '　 Hello World　 ');
```

查询结果为：Hello World

42．函数 UCASE

语法：UCASE(char)

功能：返回字符串中，所有字母改为大写，不是字母的字符不受影响。

例

```
SELECT    UCASE('hello world');
```

查询结果为：HELLO WORLD

43．函数 UPPER

语法：UPPER(char)

功能：返回字符串中，所有字母改为大写，不是字母的字符不受影响。等价于 UCASE()。

44．函数 REGEXP

REGEXP 函数是根据符合 POSIX 标准的正则表达式进行字符串匹配操作的系统函数，是字符串处理函数的一种扩展。达梦支持的匹配标准如下：

表 符合 POSIX 标准的正则表达式

语法	说明	示例
.	匹配任何除换行符之外的单个字符	d.m 匹配“dameng”
*	匹配前面的字符 0 次或多次	a*b 匹配“bat”中的“b”和“about”中的“ab”。
+	匹配前面的字符一次或多次	ac+ 匹配包含字母“a”和至少一个字母“c”的单词，如“race”和“ace”。
^	匹配行首	^car 仅当单词“car”显示为行中的第一组字符时匹配该单词
$	匹配行尾	end$ 仅当单词“end”显示为可能位于行尾的最后一组字符时匹配该单词
[]	字符集，匹配任何括号间的字符	be[n-t] 匹配“between”中的“bet”、“beneath”中的“ben”和“beside”中的“bes”，但不匹配“below”中的“bel”。
[^]	排除字符集。匹配任何不在括号间的字符	be[^n-t] 匹配“before”中的“bef”、“behind”中的“beh”和“below”中的“bel”，但是不匹配“beneath”中的“ben”。
(表达式)	在表达式加上括号或标签在替换命令中使用	(abc)+匹配“abcabcabc”
|	匹配 OR 符号 (|) 之前或之后的表达式。最常用在分组中。	(sponge|mud) bath 匹配“sponge bath”和“mud bath”。
\	按原义匹配反斜杠 (\) 之后的字符。这使您可以查找正则表达式表示法中使用的字符，如 { 和 ^。	\^ 搜索 ^ 字符
{min[,max]}	匹配以带括号的表达式标记的文本	zo{1} 匹配“Alonzo1”和“Gonzo1”中的“zo1”，但不匹配“zone”中的“zo”。
[[:alpha:]]	表示任意字母\w ([a-z]+) | ([A-Z]+)	
[[:digit:]]	表示任意数字\d ([0-9]+)	
[[:lower:]]	表示任意小写字母 ([a-z]+)	
[[:alnum:]]	表示任意字母和数字([a-z0-9]+)	
[[:space:]]	表示任意空格	
[[:upper:]]	表示任意大写字母([A-Z]+)	
[[:punct:]]	表示任意标点符号	
[[:xdigit:]]	表示任意 16 进制数([0-9a-fA-F]+)	
\w	表示一个词字符	
\W	表示一个非词字符	
\s	表示一个空格字符	
\S	表示一个非空格字符	

值得注意的是，对于 Perl 规则的正则表达式达梦暂不支持：[==], \d, \D, {n}?, \A, \Z, *?, +?, ??, {n}?, {n,}?, {n,m}?。

DM7 支持的 REGEXP 函数如下表：

表 REGEXP 函数

序号	函数名	功能简要说明
1	REGEXP_COUNT(str, pattern[, position [, match_param]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串的个数，并符合匹配参数 match_param
2	REGEXP_LIKE(str, pattern [, match_param])	根据 pattern 正则表达式，查找 str 字符串是否存在符合正则表达式的子串，并符合匹配参数 match_param
3	REGEXP_INSTR(str, pattern[, position[, occurrence [, return_opt [, match_param [, subexpr]]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，如果 return_opt 为 0，返回第 occurrence 次出现的位置，如果 return_opt 为大于 0，则返回该出现位置的下一个字符位置，并符合匹配参数。Subexpr 为匹配的子 pattern。
4	REGEXP_SUBSTR(str, pattern [,position [, occurrence [,match_param[, subexpr]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，返回第 occurrence 次出现的子串，并符合匹配参数 match_param。
5	REGEXP_REPLACE(str, pattern [, replace_str [, position [, occurrence [,match_param]]]])	根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串，并用 replace_str 进行替换第 occurrence 次出现的子串，并符合匹配参数 match_param。

参数说明：

str：待匹配的字符串，最大长度为 8188 字节；

pattern：符合 POSIX 标准的正则表达式，最大长度为 512 字节；

position：匹配的源字符串的开始位置，正整数，默认为 1；

occurrence：匹配次数，正整数，默认为 1；

match_param：正则表达式的匹配参数，默认大小写敏感，如下表所示：

表 匹配参数

值	说明
c	表示大小写敏感。例如：REGEXP_COUNT('AbCd', 'abcd', 1, 'c')，结果为 0。
i	表示大小写不敏感。例如：REGEXP_COUNT('AbCd', 'abcd', 1, 'i')，结果为 1。
m	将源字符串当成多行处理，默认当成一行。 例如：REGEXP_COUNT('ab'||CHR(10)||'ac', '^a.', 1, 'm')，结果为 2。
n	通配符（.）匹配换行符，默认不匹配。 例如：REGEXP_COUNT('a'||CHR(10)||'d', 'a.d', 1, 'n')，结果为 1。
x	忽略空格字符。例如：REGEXP_COUNT('abcd', 'a b c d', 1, 'x')，结果为 1。

return_opt：正整数，返回匹配子串的位置。值为 0：表示返回子串的开始位置；值大于 0：表示返回子串结束位置的下一个字符位置；

subexpr：正整数，取值范围为：0~9，表示匹配第 subexpr 个子 pattern 正则表达式，子 pattern 必须是括号的一部分。如果 subexpr=0，则表示匹配整个正则表达式；如果 subexpr > 0，则匹配对应的第 subexpr 个子 pattern；如果 subexpr 大于子 pattern 个数，则返回 0；如果 subexpr 为 NULL，则返回 NULL。

replace_str：用于替换的字符串，最大长度为 512 字节。

如下详细介绍各函数：

1. 函数 REGEXP_COUNT

语法格式：

```
REGEXP_COUNT(str, pattern[, position [, match_param]])
```

语句功能：

根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串的个数，并符合匹配参数 match_param。position 默认值为 1，position 为正整数，小于

0 则报错；如果 position 为空，则返回 NULL。pattern 必须符合正则表达式的规则，否则报错。match_param 不合法，则报错。

返回值：

如果 str、pattern 和 match_param 其中有一个为空串或 NULL，则返回 NULL。如果不匹配，返回 0；如果匹配，返回匹配的个数。

举例说明：

```
SELECT REGEXP_COUNT('AbCd', 'abcd', 1, 'i') FROM DUAL;
```

查询结果：1

```
SELECT REGEXP_COUNT('AbCd', 'abcd', 1, 'c') FROM DUAL;
```

查询结果：0

2. 函数 REGEXP_LIKE

语法格式：

```
REGEXP_LIKE(str, pattern [, match_param])
```

语句功能：

根据 pattern 正则表达式，查找 str 字符串是否存在符合正则表达式的子串，并符合匹配参数 match_param。

返回值：

如果匹配，则返回 TRUE；否则返回 FALSE。如果 str、pattern 和 match_param 中任一个为空串或 NULL，则返回 NULL；

举例说明：

```
Select 1 from dual where regexp_like('DM database V7', 'dm', 'c');
```

查询结果：无返回行

```
Select 1 from dual where regexp_like('DM database V7', 'dm', 'i');
```

查询结果：1

3. 函数 REGEXP_INSTR

语法格式：

```
REGEXP_INSTR(str, pattern[, position[,  occurrence [, return_opt [, match_param [, subexpr]]]]])
```

语句功能：

根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，如果 return_opt 为 0，返回第 occurrence 次出现的位置，如果 return_opt 为大于 0，则返回该出现位置的下一个字符位置，并符合匹配参数。Subexpr 为匹配的子 pattern。

返回值：

如果 str、pattern、position、occurrence、return_opt、match_param 和 subexpr 中任一个为 NULL，则返回 NULL。否则返回符合条件的子串位置，如果没有找到，则返回 0。

举例说明：

```
select regexp_instr('a 为了 aaac','aa') from dual;
```

查询结果：4

```
select regexp_instr('a 为了 aaac','aa',5) from dual;
```

查询结果：5

```
SELECT REGEXP_INSTR('1234567890', '(123)(4(56)(78))', 1, 1, 0, 'i', 2) "REGEXP_INSTR" FROM DUAL;
```

查询结果：4

4. REGEXP_SUBSTR

语法格式：

```
REGEXP_SUBSTR(str, pattern [,position [, occurrence [,match_param[, subexpr]]]])
```

语句功能：

根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合 subexpr 正则表达式的子串，返回第 occurrence 次出现的子串，并符合匹配参数 match_param。occurrence 默认为 1。如果 position 或 occurrence 的输入值不为正数，则报错。

返回值：

如果 str、pattern、position、occurrence、match_param 和 subexpr 中任一个为 NULL，则返回 NULL。如果找到符合正则表达式的子串，则返回匹配的子串；如果没有找到，则返回 NULL。

举例说明：

```
select regexp_substr('a 为 aa 了 aac','(a*)',2) from dual;
```

查询结果：空

```
select regexp_substr('a 为 aa 了 aac','(a+)',2) from dual;
```

查询结果：aa

```
SELECT REGEXP_SUBSTR('500 DM7 DATABASE, SHANG HAI, CN', ',[^,]+,', 5, 1, 'i', 0) "REGEXPR_SUBSTR"  FROM DUAL;
```

查询结果：, SHANG HAI,

5. REGEXP_REPLACE

语法格式：

```
REGEXP_REPLACE(str, pattern [, replace_str [, position [, occurrence [,match_param]]]])
```

语句功能：

根据 pattern 正则表达式，从 str 字符串的第 position 个字符开始查找符合正则表达式的子串，并用 replace_str 进行替换第 occurrence 次出现的子串，并符合匹配参数 match_param。occurrence 默认为 1。replace_str 默认为空串，在替换过程中，则相当于删除查找到的子串；position 默认值为 1，如果 position 的值不为正整数，则报错；

返回值：

返回替换后的 str。如果 str、pattern、position、occurrence 和 match_param 中任一个为 NULL，则返回 NULL；如果 str 中所有的字符都被空串替换，则返回 NULL，相当于删除所有的字符。

举例说明：

```
select regexp_replace('a 为了 aaac','aa','bb') from dual;
```

查询结果：a 为了 bbac

```
select regexp_replace('a 为了 ac','aa','bb') from dual;
```

查询结果：a 为了 ac

```
select regexp_replace('a 为 aa 了 aac','aa','bb') from dual;
```

查询结果：a 为 bb 了 bbc

```
SELECT REGEXP_REPLACE('500 DM7 DATABASE, SHANG HAI, CN', ',[^,]+,', ', WU HAN,', 5, 1,'i') "REGEXPR_REPLACE"  FROM DUAL;
```

查询结果：500 DM7 DATABASE, WU HAN, CN

45．函数 OVERLAY

```
语法：OVERLAY(char1 PLACING char2 FROM m [ FOR n ])
```

功能：用串 char2（称为“替换字符串”）覆盖源串 char1 的指定子串，该子串是通过在源串中的给定起始位置的数值（m）和长度的数值（n）而指明，来修改一个串自变量。当子串长度为 0 时，不会从源串中移去任何串；当不指定 n 时，默认 n 为 char2 的长度。函数

的返回串是在源串的给定起始位置插入替换字符串所得的结果。

例

```
SELECT NAME,OVERLAY(NAME PLACING '口' FROM 3 FOR 2) FROM PRODUCTION.PRODUCT;
```

查询结果如下表 9.2.6 所示。

表 9.2.10

NAME	OVERLAY(NAME PLACING '口' FROM 3 FOR 2)
红楼梦	红口梦
水浒传	水口传
老人与海	老口与海
射雕英雄传(全四册)	射口英雄传(全四册)
鲁迅文集(小说、散文、杂文)全两册	鲁口文集(小说、散文、杂文)全两册
长征	长口
数据结构(C 语言版)(附光盘)	数口结构(C 语言版)(附光盘)
工作中无小事	工口中无小事
突破英文基础词汇	突口英文基础词汇
噼里啪啦丛书(全 7 册)	噼口啪啦丛书(全 7 册)

```
SELECT OVERLAY('txxxxas' PLACING 'hom' FROM 2 FOR 4);
```

查询结果为：thomas

46．函数 TEXT_EQU

语法：TEXT_EQU(n1,n2)

功能：返回 n1，n2 的比较结果，完全相等，返回 1；否则返回 0。n1，n2 的类型为 CLOB、TEXT 或 LONGVARCHAR。如果 n1 或 n2 均为空串或 NULL，结果返回为 1；否则只有一个为空串或为 NULL，结果返回 0。不忽略结果空格和英文字母大小写。

例

```
select text_equal('a', 'b');
```

查询结果为：0

```
select text_equal('a','a');
```

查询结果为：1

47．函数 BLOB_EQU

语法：BLOB_EQU(n1,n2)

功能：返回 n1，n2 两个数的比较结果，完全相等，返回 1；否则返回 0。n1，n2 的类型为 BLOB、IMAGE 或 LONGVARBINARY。如果 n1 或 n2 均为空串或 NULL，结果返回为 1；否则只有一个为空串或为 NULL，结果返回 0。

例

```
select blob_equal(0xFFFEE, 0xEEEFF);
```

查询结果为：0

```
select blob_equal(0xFFFEE, 0xFFFEE);
```

查询结果为：1

48．函数 NLSSORT

语法：NLSSORT(str1 [,nls_sort=str2])

功能：返回对汉字排序的编码。当只有 str1 一个参数时，与 RAWTOHEX 类似，返回 16 进制字符串。str2 决定按哪种方式排序，可以选择 schinese_pinyin_m、schinese_stroke_m、schinese_radical_m。分别表示支持对汉字按拼音、笔画、部首排序。

例：

```
select nlssort('abc') from dual;
```

查询结果为：61626300

```
create table test(c1 varchar2(200));
insert into test values('啊');
insert into test values('不');
insert into test values('才');
insert into test values('的');
insert into test values('一');
insert into test values('二');
insert into test values('三');
insert into test values('四');
insert into test values('品');
insert into test values('磊');
select * from test order by nlssort(c1, 'nls_sort=schinese_pinyin_m');        --拼音
```

查询结果为：

行号	C1
1	啊
2	不
3	才
4	的
5	二
6	磊
7	品
8	三
9	四
10	一

```
select * from test order by nlssort(c1, 'nls_sort=schinese_stroke_m');        --笔画
```

查询结果为：

行号	C1
1	一
2	二
3	三
4	才
5	不
6	四
7	的
8	品
9	啊
10	磊

```
select * from test order by nlssort(c1, 'nls_sort=schinese_radical_m');        --部首
```

查询结果为：

```
行号          C1
---------- --
1             一
2             二
3             三
4             不
5             品
6             啊
7             四
8             才
9             的
10            磊
```

```
select c1,nlssort(c1),nlssort(c1, 'nls_sort=schinese_pinyin_m') from test1 order by nlssort(c1, 'nls_sort=schinese_pinyin_m');
```

分别返回 c1，返回将 c1 转化后的 16 进制字符串，返回用来为汉字排序的编码。

查询结果为：

```
C1      NLSSORT ( C1 )      NLSSORT ( C1, 'nls_sort=schinese_pinyin_m')
 啊     B0A100     3B2C
 不     B2BB00     4248
 才     B2C500     4291
 的     B5C400     4D8D
 二     B6FE00     531D
 磊     C0DA00     743E
 品     C6B700     8898
 三     C8FD00     932C
 四     CBC400     996A
 一     D2BB00     B310
```

9.3 日期时间函数

日期时间函数的参数至少有一个是日期时间类型(TIME，DATE，TIMESTAMP)，返回值一般为日期时间类型和数值类型。

1．函数 ADD_DAYS

```
语法：ADD_DAYS( date, n)
```

功能：返回日期 date 加上相应天数 n 后的日期值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回值为日期类型(DATE)。

例

```
SELECT ADD_DAYS( DATE '2000-01-12',1);
```

查询结果为：2000-01-13

2．函数 ADD_MONTHS

```
语法：ADD_MONTHS(date,n)
```

功能：返回日期 date 加上 n 个月的日期时间值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型固定为日期类型(DATE)。如果相加之后的结果日期中月份所包含的天数比 date 日期中的日分量要少，那么结果日期的该月最后一天被返回。

例

```
SELECT ADD_MONTHS(DATE '2000-01-31',1);
```

查询结果为：2000-02-29

```
SELECT ADD_MONTHS(TIMESTAMP '2000-01-31 20:00:00',1);
```

查询结果为：2000-02-29

3．函数　ADD_WEEKS

```
语法：ADD_WEEKS( date, n)
```

功能：返回日期 date 加上相应星期数 n 后的日期值。n 可以是任意整数，date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型固定为日期类型(DATE)。

例

```
SELECT ADD_WEEKS( DATE '2000-01-12',1);
```

查询结果为: 2000-01-19

4．函数 CURDATE

```
语法：CURDATE()
```

功能：返回当前日期值，结果类型为 DATE。

例

```
SELECT     CURDATE();
```

查询结果为：执行此查询当天日期，如 2003-02-27

5．函数 CURTIME

```
语法：CURTIME()
```

功能：返回当前时间值，结果类型为 TIME WITH TIME ZONE。

例

```
SELECT    CURTIME();
```

查询结果为：执行此查询的当前时间，如 14:53:54.859000 +8:00

6．函数 CURRENT_DATE

```
语法：CURRENT_DATE
```

功能：返回当前日期值，结果类型为 DATE，等价于 CURDATE()。

7．函数 CURRENT_TIME

```
语法：CURRENT_TIME(n)
```

功能：返回当前时间值，结果类型为 TIME WITH TIME ZONE，等价于 CURTIME()。n 的范围为[-2147483648, 2147483647]。

8．函数 CURRENT_TIMESTAMP

```
语法：CURRENT_TIMESTAMP(n)
```

功能：返回当前带会话时区的时间戳，结果类型为 TIMESTAMP WITH TIME ZONE。说明：n 暂时没有限制作用。

例

```
SELECT    CURRENT_TIMESTAMP();
```

查询结果为：执行此查询的当前日期时间，如 2011-12-27 13:03:56.000000 +8:00

9．函数 DATEADD

语法：DATEADD(datepart,n,date)

功能：向指定的日期 date 加上 n 个 datepart 指定的时间段，返回新的 timestamp 值。datepart 可以为 YEAR(缩写 YY 或 YYYY)、QUARTER(缩写 QQ 或 Q)、MONTH(缩写 MM 或 M)、DAYOFYEAR(缩写 DY 或 Y)、DAY(缩写 DD 或 D)、WEEK(缩写 WK 或 WW)、HOUR(缩写 HH)、MINUTE(缩写 MI 或 N)、SECOND(缩写 SS 或 S)和 MILLISECOND(缩写 MS)。

10．函数 DATEDIFF

语法：DATEDIFF(datepart,date1,date2)

功能：返回跨两个指定日期的日期和时间边界数。datepart 可以为 YEAR(缩写 YY 或 YYYY)、QUARTER(缩写 QQ 或 Q)、MONTH(缩写 MM 或 M)、DAYOFYEAR(缩写 DY 或 Y)、DAY(缩写 DD 或 D)、WEEK(缩写 WK 或 WW)、HOUR(缩写 HH)、MINUTE(缩写 MI 或 N)、SECOND(缩写 SS 或 S)和 MILLISECOND(缩写 MS)。

注：当结果超出整数值范围，DATEDIFF 产生错误。对于毫秒 MILLISECOND，最大数是 24 天 20 小时 31 分钟零 23.647 秒。对于秒，最大数是 68 年。若想提高可以表示的范围，可以使用 BIGDATEDIFF，其使用方法与 DATEDIFF 函数一致，只是可以表示更广范围的秒和毫秒。

例

```
SELECT DATEDIFF(QQ, '2003-06-01', DATE '2002-01-01');
```

查询结果为：-5

```
SELECT DATEDIFF(MONTH, '2001-06-01', DATE '2002-01-01');
```

查询结果为：7

```
SELECT DATEDIFF(WK, DATE '2003-02-07',DATE '2003-02-14');
```

查询结果为：1

```
SELECT DATEDIFF(MS,'2003-02-14 12:10:10.000','2003-02-14 12:09:09.300');
```

查询结果为：-60700

11．函数 DATEPART

语法：DATEPART(datepart,date)

功能：返回代表日期的指定部分的整数。datepart 可以为 YEAR(缩写 YY 或 YYYY)、QUARTER(缩写 QQ 或 Q)、MONTH(缩写 MM 或 M)、DAYOFYEAR(缩写 DY 或 Y)、DAY(缩写 DD 或 D)、WEEK(缩写 WK 或 WW)、WEEKDAY(缩写 DW)、HOUR(缩写 HH)、MINUTE(缩写 MI 或 N)、SECOND(缩写 SS 或 S)和 MILLISECOND(缩写 MS)。

例

```
SELECT DATEPART(SECOND, DATETIME '2000-02-02 13:33:40.00');
```

查询结果为：40

```
SELECT DATEPART(DY, '2000-02-02');
```

查询结果为：33

```
SELECT DATEPART(WEEKDAY, '2002-02-02');
```

查询结果为：7

说明：日期函数：date_part，其功能与 datepart 完全一样。但是写法有点不同：select

datepart(year,'2008-10-10');如果用 date_part，则要写成：select date_part('2008-10-10','year')，即：参数顺序颠倒，同时指定要获取的日期部分的参数要带引号。

12．函数 DAYNAME

语法：DAYNAME(date)

功能：返回日期的星期名称。

例

```
SELECT DAYNAME(DATE '2012-01-01');
```

查询结果为：Sunday

13．函数 DAYOFMONTH

语法：DAYOFMONTH(date)

功能：返回日期为所处月份中的第几天。

例

```
SELECT DAYOFMONTH('2003-01-03');
```

查询结果为：3

14．函数 DAYOFWEEK

语法：DAYOFWEEK(date)

功能：返回日期为所处星期中的第几天。

例

```
SELECT DAYOFWEEK('2003-01-01');
```

查询结果为：4

15．函数 DAYOFYEAR

语法：DAYOFYEAR(date)

功能：返回日期为所处年中的第几天。

例

```
SELECT DAYOFYEAR('2003-03-03');
```

查询结果为：62

16．DAYS_BETWEEN 函数

语法： DAYS_BETWEEN(dt1,dt2)

功能：返回两个日期之间相差的天数。

17．函数 EXTRACT

语法：EXTRACT(dtfield FROM date)

功能：EXTRACT 从日期时间类型或时间间隔类型的参数 date 中抽取 dtfield 对应的数值，并返回一个数字值。如果 date 是 NULL，则返回 NULL。Dtfiled 可以是 YEAR、MONTH、DAY、HOUR、MINUTE、SECOND。对于 SECOND 之外的任何域，函数返回整数，对于 SECOND 返回小数。

例：

```
SELECT EXTRACT(YEAR FROM DATE '2000-01-01');
```

查询结果为：2000

```
SELECT EXTRACT(DAY FROM DATE '2000-01-01');
```

查询结果为：1

```
SELECT EXTRACT(MINUTE FROM TIME '12:00:01.35');
```

查询结果为：0

```
SELECT EXTRACT(TIMEZONE_HOUR FROM TIME '12:00:01.35 +9:30');
```

查询结果为：9

```
SELECT EXTRACT(TIMEZONE_MINUTE FROM TIME '12:00:01.35 +9:30');
```

查询结果为：30

```
SELECT EXTRACT(SECOND FROM TIMESTAMP '2000-01-01 12:00:01.35');
```

查询结果为：1.3500000000E+000

```
SELECT EXTRACT(SECOND FROM INTERVAL '-05:01:22.01' HOUR TO SECOND);
```

查询结果为：-2.2010000000E+001

18．函数 GETDATE

语法：GETDATE()

功能：返回系统的当前时间戳。

例

```
SELECT GETDATE();
```

查询结果为：返回系统的当前日期时间，如 2011-12-05 11:31:10.359000

29．函数 GREATEST

语法：GREATEST(n1,n2,n3)

功能：求 n1、n2 和 n3 中的最大日期。

例

```
SELECT GREATEST(date'1999-01-01',date'1998-01-01',date'2000-01-01');
```

查询结果：2000-01-01

20．函数 HOUR

语法：HOUR(time)

功能：返回时间中的小时分量。

例

```
SELECT HOUR(TIME '20:10:16');
```

查询结果为：20

21．函数 LAST_DAY

语法：LAST_DAY(date)

功能：返回 date 所在月最后一天的日期,date 是日期类型(DATE)或时间戳类型(TIMESTAMP)，返回类型与 date 相同。

例

```
SELECT LAST_DAY(SYSDATE) "Days Left";
```

查询结果为：如：当前日期为 2003 年 2 月的某一天，则结果为 2003-02-28

```
SELECT LAST_DAY(TIMESTAMP '2000-01-11 12:00:00');
```

查询结果为：2000-01-31

22．函数 LEAST

语法：LEAST(n1,n2,n3)

功能：求n1、n2和n3中的最小日期。

例

```
SELECT LEAST(date'1999-01-01',date'1998-01-01',date'2000-01-01');
```

查询结果：1998-01-01

23．函数 MINUTE

语法：MINUTE(time)

功能：返回时间中的分钟分量。

例

```
SELECT MINUTE('20:10:16');
```

查询结果为：10

24．函数 MONTH

语法：MONTH(date)

功能：返回日期中的月份分量。

例

```
SELECT MONTH('2002-11-12');
```

查询结果为：11

25．函数 MONTHNAME

语法：MONTHNAME(date)

功能：返回日期中月份分量的名称。

例

```
SELECT MONTHNAME('2002-11-12');
```

查询结果为：November

26．函数 MONTHS_BETWEEN

语法：MONTHS_BETWEEN(date1,date2)

功能：返回 date1 和 date2 之间的月份值。如果 date1 比 date2 晚，返回正值，否则返回负值。如果 date1 和 date2 这两个日期为同一天，或者都是所在月的最后一天，则返回整数，否则返回值带有小数。date1 和 date2 是日期类型(DATE)或时间戳类型(TIMESTAMP)。

例

```
SELECT MONTHS_BETWEEN(DATE '1995-02-28', DATE '1995-01-31') "Months";
```

查询结果为：1.0

```
SELECT MONTHS_BETWEEN(TIMESTAMP '1995-03-28 12:00:00',
TIMESTAMP '1995-01-31 12:00:00') "Months";
```

查询结果为：1.903226

27．函数 NEXT_DAY

语法：NEXT_DAY(date,char)

功能：返回在日期 date 之后满足由 char 给出的条件的第一天。char 指定了一周中的某一个天(星期几)，返回值的时间分量与 date 相同，char 是大小写无关的。

Char 取值如表 9.3.1 所示。

表 9.3.1 星期描述说明

输入值	含义
SUN	星期日
SUNDAY	
MON	星期一
MONDAY	
TUES	星期二
TUESDAY	
WED	星期三
WEDNESDAY	
THURS	星期四
THURSDAY	
FRI	星期五
FRIDAY	
SAT	星期六
SATURDAY	

例

```
SELECT NEXT_DAY(DATE '2001-08-02', 'MONDAY');
```

查询结果为：2001-08-06

```
SELECT NEXT_DAY('2001-08-02 12:00:00', 'FRI');
```

查询结果为：2001-08-03

28．函数 NOW

语法：NOW()

功能：返回系统的当前时间戳。等价于 GETDATE()。

29．函数 QUARTER

语法：QUARTER(date)

功能：返回日期在所处年中的季度数。

例

```
SELECT QUARTER('2002-08-01');
```

查询结果为：3

30．函数 SECOND

语法：SECOND(time)

功能：返回时间中的秒分量。

例

```
SELECT SECOND('08:10:25.300');
```

查询结果为：25

31．函数 ROUND

语法：ROUND(date[,format])

功能：将日期时间 date 四舍五入到最接近格式参数 format 指定的形式。如果没有指定语法的话，到今天正午 12P.M.为止的时间舍取为今天的日期，之后的时间舍取为第二天 12A.M.。日期时间 12A.M.，为一天的初始时刻。参数 date 的类型可以是 DATE 或

TIMESTAMP，但应与 format 相匹配。函数的返回结果的类型与参数 date 相同。format 具体如表 9.3.2 所示。

表 9.3.2 日期时间说明

format 的格式	含义	date 数据类型
cc, scc	世纪，从 1950、2050 等年份的一月一号午夜凌晨起的日期，舍取至下个世纪的一月一号	DATE TIMESTAMP
syear, syyy, y, yy, yyy, yyyy, year	年，从七月一号午夜凌晨起的日期，舍取至下个年度的一月一号	DATE TIMESTAMP
Q	季度，从十六号午夜凌晨舍取到季度的第二个月，忽略月中的天数	DATE TIMESTAMP
month, mm, m, rm	月，从十六号午夜凌晨舍取	DATE TIMESTAMP
Ww	舍取为与本年第一天星期数相同的最近的那一天	DATE TIMESTAMP
W	舍取为本月第一天星期数相同的最近的一天	DATE TIMESTAMP
ddd, dd, j	从正午起，舍取为下一天，默认值	DATE TIMESTAMP
day, dy, d	星期三正午起，舍取为下个星期天	DATE TIMESTAMP
hh, hh12, hh24	在一个小时的 30 分 30 秒之后的时间舍取为下一小时	TIME TIMESTAMP
Mi	在一个分钟 30 秒之后的时间舍取为下一分	TIME TIMESTAMP

有关 ww 和 w 的计算进一步解释如下(下面的时间仅当 date 参数为时间戳时才有效)：

ww 产生与本年第一天星期数相同的最近的日期。因为每两个星期数相同日期之间相隔六天，这意味着舍取结果在给定日期之后三天以内。例如，如果本年第一天为星期二，若给定日期在星期五午夜 23:59:59 之前(包含星期五 23:59:59)，则舍取为本星期的星期二的日期；否则舍取为下星期的星期二的日期。

w 计算的方式类似，不是产生最近的星期一 00:00:00，而是产生与本月第一天相同的星期数的日期。

例

```
SELECT ROUND(DATE '1992-10-27', 'scc');
```

查询结果为：2001-01-01

```
SELECT ROUND(DATE '1992-10-27', 'YEAR') "FIRST OF THE YEAR";
```

查询结果为：1993-01-01

```
SELECT ROUND(DATE '1992-10-27', 'q');
```

查询结果为：1992-10-01

```
SELECT ROUND(DATE '1992-10-27', 'month');
```

查询结果为：1992-11-01

```
SELECT ROUND(TIMESTAMP '1992-10-27 11:00:00', 'ww');
```

查询结果为：1992-10-28 00:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 11:00:00', 'w');
```

查询结果为：1992-10-29 00:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:01', 'ddd');
```

查询结果为：1992-10-28 00:00:00.000000

```
SELECT ROUND(DATE '1992-10-27', 'day');
```

查询结果为：1992-10-25

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:31', 'hh');
```

查询结果为：1992-10-27 12:00:00.000000

```
SELECT ROUND(TIMESTAMP '1992-10-27 12:00:31', 'mi');
```

查询结果为：1992-10-27 12:01:00.000000

32．函数 TIMESTAMPADD

语法：TIMESTAMPADD(interval,n,timestamp)

功能：返回时间戳 timestamp 加上 n 个 interval 代表的时间间隔的结果。interval 可以为 SQL_TSI_FRAC_SECOND、SQL_TSI_SECOND、SQL_TSI_MINUTE、SQL_TSI_HOUR、SQL_TSI_DAY 、 SQL_TSI_WEEK 、 SQL_TSI_MONTH 、 SQL_TSI_QUARTER 和 SQL_TSI_YEAR。

例

```
SELECT TIMESTAMPADD(SQL_TSI_FRAC_SECOND, 5, '2003-02-10 08:12:20.300' );
```

查询结果为：2003-02-10 08:12:20.305000

```
SELECT TIMESTAMPADD(SQL_TSI_YEAR, 30, DATE '2002-01-01');
```

查询结果为：2032-01-01 00:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_QUARTER, 2, TIMESTAMP '2002-01-01 12:00:00');
```

查询结果为：2002-07-01 12:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_DAY, 40, '2002-12-01 12:00:00');
```

查询结果为：2003-01-10 12:00:00.000000

```
SELECT TIMESTAMPADD(SQL_TSI_WEEK, 1, '2002-01-30');
```

查询结果为：2002-02-06 00:00:00.000000

33．函数 TIMESTAMPDIFF

语法：TIMESTAMPDIFF(interval,timestamp1,timestamp2)

功能：返回一个表明 timestamp2 与 timestamp1 之间的 interval 类型的时间间隔的整数。interval 可以为 SQL_TSI_FRAC_SECOND、SQL_TSI_SECOND、SQL_TSI_MINUTE、SQL_TSI_HOUR 、 SQL_TSI_DAY 、 SQL_TSI_WEEK 、 SQL_TSI_MONTH 、 SQL_TSI_QUARTER 和 SQL_TSI_YEAR。

注：当结果超出整数值范围，TIMESTAMPDIFF 产生错误。对于秒级 SQL_TSI_SECOND，最大数是 68 年。

例

```
SELECT TIMESTAMPDIFF(SQL_TSI_FRAC_SECOND,
'2003-02-14 12:10:10.000', '2003-02-14 12:09:09.300');
```

查询结果为：-60700

```
SELECT TIMESTAMPDIFF(SQL_TSI_QUARTER, '2003-06-01', DATE '2002-01-01');
```

查询结果为：-5

```
SELECT TIMESTAMPDIFF(SQL_TSI_MONTH, '2001-06-01', DATE '2002-01-01');
```

查询结果为：7

```
SELECT TIMESTAMPDIFF(SQL_TSI_WEEK, DATE '2003-02-07',DATE '2003-02-14');
```

查询结果为：1

34．函数 SYSDATE

语法： SYSDATE()

功能： 获取系统当前时间。

例

```
SELECT  SYSDATE();
```

查询结果：当前系统时间

35．函数 TO_DATE

语法：TO_DATE (char [,fmt])

功能：将 CHAR 或者 VARCHAR 类型的值转换为 DATE 数据类型。其中 fmt 为指定输入串的日期语法，若省略 fmt，该函数使用缺省的日期语法。TO_DATE 中使用的有效的日期语法分量和 TO_CHAR 函数中的一致。

DM 缺省的日期语法为："YYYYMMDD HH:MI:SS"、"YYYY.MM.DD HH:MI:SS"、"YYYY-MM-DD HH:MI:SS"或是"YYYY/MM/DD HH:MI:SS"。

合法的 DATE 格式为年月日、月日年和日月年三种，各部分之间可以有间隔符(".","-","/")或者没有间隔符；合法的 TIME 格式为：时分和时分秒,间隔符为":"。

例

```
SELECT TO_DATE('2003-06-19 08:40:36','YYYY-MM-DD HH:MI:SS');
```

查询结果：2003-06-19 08:40:36.000000

```
SELECT TO_DATE('2003/06/19','YYYY/MM/DD');
```

查询结果：2003-06-19 00:00:00.000000

```
SELECT TO_DATE('20030619 08:40:36','YYYYMMDD HH24:MI:SS');
```

查询结果：2003-06-19 08:40:36.000000

说明：DM 系统中对于年数小于 50 的，返回 20XX，对于年数大于或等于 50 的，返回 19XX

例

```
SELECT TO_DATE('10-11-10', 'DD-MM-YY');
```

查询结果：2010-11-10 00:00:00.000000

```
SELECT TO_DATE('10-11-50', 'DD-MM-YY');
```

查询结果：2050-11-10 00:00:00.000000

36．函数 TRUNC

语法：TRUNC(date[,format])

功能：将日期时间 date 截断到最接近格式参数 format 指定的形式。若 format 缺省，则返回当天日期。语法与 ROUND 类似，但结果是直接截断，而不是四舍五入。参数及函数的返回类型与 ROUND 相同。参见 ROUND。

例

```
SELECT TRUNC(DATE '1992-10-27', 'scc');
```

查询结果为：1901-01-01

```
SELECT TRUNC(DATE '1992-10-27', 'YEAR') "FIRST OF THE YEAR";
```

查询结果为：1992-01-01

```
SELECT TRUNC(DATE '1992-10-27', 'q');
```

查询结果为：1992-10-01

```
SELECT TRUNC(DATE '1992-10-27', 'month');
```

查询结果为：1992-10-01

```
SELECT TRUNC(TIMESTAMP '1992-10-27 11:00:00', 'ww');
```

查询结果为：1992-10-21 00:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 11:00:00', 'w');
```

查询结果为：1992-10-22 00:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:01', 'ddd');
```

查询结果为：1992-10-27 00:00:00.000000

```
SELECT TRUNC(DATE '1992-10-27', 'day');
```

查询结果为：1992-10-25

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:31', 'hh');
```

查询结果为：1992-10-27 12:00:00.000000

```
SELECT TRUNC(TIMESTAMP '1992-10-27 12:00:31', 'mi');
```

查询结果为：1992-10-27 12:00:00.000000

37．函数 WEEK

语法：WEEK(date)

功能：返回指定日期属于所在年中的第几周。

例

```
SELECT WEEK(DATE '2003-02-10');
```

查询结果为：7

38．函数 WEEKDAY

语法：WEEKDAY(date)

功能：返回指定日期的星期值。如果是星期日则返回 0。

例

```
SELECT WEEKDAY(DATE '1998-10-26');
```

查询结果：1

39．函数 WEEKS_BETWEEN

语法：WEEKS_BETWEEN(date1,date2)

功能：返回两个日期之间相差周数。

例

```
SELECT WEEKS_BETWEEN(DATE '1998-2-28', DATE '1998-10-31');
```

查询结果：35

40．函数 YEAR

语法：YEAR(date)

功能：返回日期中的年分量。

例

```
SELECT YEAR(DATE '2001-05-12');
```

查询结果为：2001

41．函数 YEARS_BETWEEN

语法：YEARS_BETWEEN(date1,date2)

功能：返回两个日期之间相差年数。

例

```
SELECT    YEARS_BETWEEN(DATE '1998-2-28', DATE '1999-10-31');
```

查询结果为： 1

42. 函数 LOCALTIME

语法：LOCALTIME ()

功能：返回当前时间值，结果类型为 TIME。等价于 CURTIME()。

43. 函数 LOCALTIMESTAMP

语法：LOCALTIMESTAMP ()

功能：返回当前日期时间值，结果类型为 TIMESTAMP。

44. 函数 OVERLAPS

语法：OVERLAPS (date1,date2,date3,date4)

功能：返回两两时间段是否存在重叠，date1 为 datetime 类型、date2 可以为 datetime 类型也可以为 interval 类型，date3 为 datetime 类型，date4 可为 datetime 类型，也可以 interval 类型，判断(date1,date2),(date3,date4)有无重叠，结果类型为 BIT。其中 date2 与 date4 类型必须一致，如果 date2 为 interval year to month，date4 也必须是此类型。

例

```
select overlaps('2011-10-3','2011-10-9','2011-10-6','2011-10-13');
```

查询结果为：1

```
select overlaps('2011-10-3','2011-10-9','2011-10-10','2011-10-11');
```

查询结果为：0

```
select overlaps('2011-10-3',interval '09 23' day to hour,'2011-10-10',interval '09 23' day to hour);
```

查询结果为：1

```
select overlaps('2011-10-3',interval '01 23' day to hour,'2011-10-10',interval '09 23' day to hour);
```

查询结果为：0

```
select overlaps('2011-10-3',interval '1' year to month,'2012-10-10',interval '1-1' year to month);
```

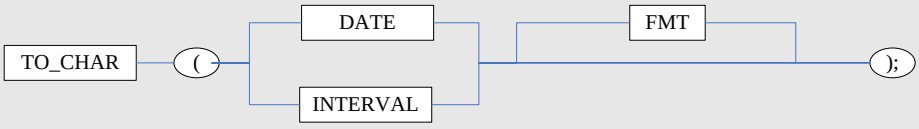
查询结果为：0

```
select overlaps('2011-10-3',interval '2' year to month,'2012-10-10',interval '1-1' year to month);
```

查询结果为：1

45. 函数 TO_CHAR

语法：TO_CHAR(DATE[,fmt])



功能：将日期数据类型 DATE 转换为一个在日期语法 fmt 中指定语法的 VARCHAR 类型字符串。若没有指定语法，日期 DATE 将按照缺省的语法转换为一个 VARCHAR 值。

有效的日期语法分量如下表：

表 9.3.3 有效日期语法分量

元素	说明
- / , . :	标点符号在结果中重新复制
D	周中的某一天，星期天算起
DD	月中的某一天
DDD	年中的某一天
HH HH12 HH24	天中的时(0－23) (均为 24 小时制)
MI	分(0－59)
MM	月(01－12)
SS	秒(0－59)
YYYY	4 位的年份
YY	年份的最后 2 位数字

DM 缺省的日期语法为：'YYYY-MM-DD HH:MI:SS'。

例

```
SELECT TO_CHAR(SYSDATE,'YYYYMMDD');
```

查询结果：20110321　/* SYSDATE 为系统当前时间*/

```
SELECT TO_CHAR(SYSDATE,'YYYY/MM/DD');
```

查询结果：2011/03/21

```
SELECT TO_CHAR(SYSDATE,'HH24:MI');
```

查询结果：16:56

```
SELECT TO_CHAR(SYSDATE,'YYYYMMDD HH24:MI:SS');
```

查询结果：20110321 16:56:19

```
SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD HH24:MI:SS');
```

查询结果：2011-03-21 16:56:28

```
SELECT TO_CHAR(INTERVAL '123-2' YEAR(3) TO MONTH) FROM DUAL;
```

查询结果：INTERVAL '123-2' YEAR(3) TO MONTH

46．函数 SYSTIMESTAMP

语法：SYSTIMESTAMP ()

功能：返回系统当前的时间戳，带数据库的时区信息。结果类型为 TIMESTAMP WITH TIME ZONE。

例

```
SELECT TIMESTAMP();
```

查询结果为：2012-10-10 11:06:12.171000 +08:00

9.4 空值判断函数

空值判断函数用于判断参数是否为 NULL，或根据参数返回 NULL。

1．函数 COALESCE

语法：COALESCE(n1,n2,…,nx)

功能：返回其参数中第一个非空的值，如果所有参数均为 NULL，则返回 NULL。如果参数为多媒体数据类型，如 TEXT 类型，则系统会将 TEXT 类型先转换为 VARCHAR 类型或 VARBINARY 类型，转换的最大长度为 8188，超过部分将被截断。

例

```
SELECT COALESCE(1,NULL);
```

查询结果：1

```
SELECT COALESCE(NULL,TIME '12:00:00',TIME '11:00:00');
```

查询结果：12:00:00

```
SELECT COALESCE(NULL,NULL,NULL,NULL);
```

查询结果：NULL

2．函数 IFNULL

语法：IFNULL(n1,n2)

功能：当表达式 n1 为非空时，返回 n1；若 n1 为空，则返回表达式 n2 的值。

例

```
SELECT IFNULL(1,3);
```

查询结果：1

```
SELECT IFNULL(NULL,3);
```

查询结果：3

```
SELECT IFNULL('',2);
```

查询结果：2

3．函数 ISNULL

语法：ISNULL(n1,n2)

功能：当表达式 n1 为非空时，返回 n1；若 n1 为空，则返回表达式 n2 的值。等价于 IFNULL()。

4．函数 NULLIF

语法：NULLIF(n1,n2)

功能：如果 n1=n2，返回 NULL，否则返回 n1。

例

```
SELECT NULLIF(1,2);
```

查询结果：1

```
SELECT NULLIF(1,1);
```

查询结果：NULL

5．函数 NVL

语法：NVL(n1,n2)

功能：等价于 COALESCE(n1,n2)

6．函数 NULL_EQU

语法：NULL_EQU(n1,n2)

功能：返回两个类型相同的值的比较，当 n1=n2 或 n1、n2 两个值中出现 null 时，返回 1。类型可以是 INT、BIT、BIGINT、FLOAT、DOUBLE、DEC、VARCHAR、DATE、TIME、TIME ZONE、DATETIME、DATETIME ZONE、INTERVAL 等。

例

```
select null_equ(1,1);
```

查询结果为：1

```
select null_equ(1,3);
```

查询结果为：0

```
select null_equ(1,null);
```

查询结果为：1

9.5 类型转换函数

1．函数 CAST

语法：CAST(value AS type)

功能：将参数 value 转换为 type 类型返回。类型之间转换的相容性如下表所示：表中，“允许”表示这种语法有效且不受限制，“一”表示语法无效，“受限”表示转换还受到具体参数值的影响。

数值类型为：精确数值类型和近似数值类型。

精确数值类型为：NUMERIC、DECIMAL、BYTE、INTEGER、SMALLINT。

近似数值类型为：FLOAT、REAL、DOUBLE PRECISION。

字符串为：变长字符串和固定字符串。

变长字符串为：VARCHAR、VARCHAR2。

固定字符串为：CHAR、CHARACTER。

字符串大对象为：CLOB、TEXT。

二进制为：BINARY、VARBIANRY。

二进制大对象为：BLOB、IMAGE。

日期为：DATE。时间为：TIME。时间戳为：TIMESTAMP。

时间时区为：TIME WITH TIME ZONE。

时间戳时区为：TIMESTAMP WITH TIME ZONE。

年月时间间隔为：INTERVAL YEAR TO MONTH、INTERVAL YEAR、INTERVAL MONTH。

日时时间间隔为：INTERVAL DAY、INTERVAL DAY TO HOUR、INTERVAL DAY TO MINUTE、INTERVAL DAY TO SECOND、INTERVAL HOUR、INTERVAL HOUR TO MINUTE、INTERVAL HOUR TO SECOND、INTERVAL MINUTE、INTERVAL MINUTE TO SECOND、INTERVAL SECOND。

表 9.5.1 CAST 类型转换相容矩阵

Value	**type 数据类型**											
数据类型	数值类型	字符串	字符串大对象	二进制	二进制大对象	日期	时间	时间戳	时间时区	时间戳时区	年月时间间隔	日时时间间隔
数值类型	受限	受限	一	允许	一	受限	受限	受限	一	一	受限	受限
字符串	允许	受限	允许	允许	允许	受限	受限	受限	受限	受限	允许	允许

字符串大对象	—	—	—	—	允许	—	—	—	—	—	—	—
二进制	允许	允许	—	允许	允许	—	—	—	—	—	—	—
二进制大对象	—	—	—	—	允许	—	—	—	—	—	—	—
日期	—	允许	—	—	—	允许	—	允许	—	允许	—	—
时间	—	允许	—	—	—	—	允许	允许	允许	允许	—	—
时间戳	—	允许	—	—	—	允许	允许	允许	允许	允许	—	—
时间时区	—	允许	—	—	—	—	允许	允许	允许	—	—	—
时间戳时区	—	允许	—	—	—	允许	允许	允许	—	允许	—	—
年月时间间隔	—	允许	—	—	—	—	—	—	—	—	允许	—
日时时间间隔	—	允许	—	—	—	—	—	—	—	—	—	允许

例

```
SELECT CAST(100.5678 AS NUMERIC(10,2));
```

查询结果：100.57

```
SELECT CAST(100.5678 AS VARCHAR(8));
```

查询结果：100.5678

```
SELECT CAST('100.5678' AS INTEGER);
```

查询结果：101

```
SELECT CAST(INTERVAL '01-01' YEAR TO MONTH AS char(50));
```

查询结果：INTERVAL '1-1' YEAR(9) TO MONTH

2．函数 CONVERT

```
语法：CONVERT(type,value)
```

功能：将参数 value 转换为 type 类型返回。其类型转换相容矩阵与函数 CAST()的相同。

例

```
SELECT CONVERT(VARCHAR(8),100.5678);
```

查询结果：100.5678

```
SELECT CONVERT(INTEGER, '100.5678');
```

查询结果：101

```
SELECT CONVERT(CHAR(50),INTERVAL '100-5' YEAR(3) TO MONTH);
```

查询结果：INTERVAL '100-5' YEAR(3) TO MONTH

3．函数 HEXTORAW

```
语法：HEXTORAW (string)
```

功能：将由 string 表示的二进制字符串转换为一个 binary 数值类型。

例

```
SELECT HEXTORAW ('abcdef');
```

查询结果为：0xABCDEF

```
SELECT HEXTORAW ('B4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE');
```

查询结果为：0xB4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE

4．函数 RAWTOHEX

语法：RAWTOHEX (binary)

功能：将 RAW 类数值 binary 转换为一个相应的十六进制表示的字符串。binary 中的每个字节都被转换为一个双字节的字符串。RAWTOHEX 和 HEXTORAW 是两个相反的函数。

例 1 SELECT RAWTOHEX('达梦数据库有限公司');

查询结果为：B4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE

例 2 SELECT RAWTOHEX('13');

查询结果为：3133

9.6 杂类函数

1．DECODE 函数

语法：DECODE(exp, search1, result1, … searchn, resultn[,default])

功能：查表译码，DECODE 函数将 exp 与 search1,search2, … searchn 相比较，如果等于 searchx，则返回 resultx，如果没有找到匹配项，则返回 default, 如果未定义 default，返回 NULL。

例

```
SELECT DECODE(1, 1, 'A', 2, 'B');
```

查询结果为：'A'

```
SELECT DECODE(3, 1, 'A', 2, 'B');
```

查询结果为：NULL

```
SELECT DECODE(3, 1, 'A', 2, 'B', 'C');
```

查询结果为：'C'

2．ISDATE 函数

语法：ISDATE(exp)

功能：判断给定表达式是否为有效的日期，是返回 1，否则返回 0。

例

```
SELECT ISDATE('2012-10-9');
```

查询结果为：1

```
SELECT ISDATE('2012-10-9 13:23:37');
```

查询结果为：1

```
SELECT ISDATE(100);
```

查询结果为：0

3．ISNUMERIC 函数

语法：ISNUMERIC(exp)

功能：判断给定表达式是否为有效的数值，是返回 1，否则返回 0。

例

```
SELECT ISNUMERIC(1.323E+100);
```

查询结果为：1

```
SELECT ISNUMERIC('2a');
```

查询结果为：0

第 10 章 一致性和并发性

数据一致性是指表示客观世界同一事物状态的数据，不管出现在何时何处都是一致的、正确的和完整的。数据库是一个共享资源，可为多个应用程序所共享，它们同时存取数据库中的数据，这就是数据库的并发操作。此时，如果不对并发操作进行控制，则会存取不正确的数据，或破坏数据库数据的一致性。

DM 利用事务和封锁机制提供数据并发存取和数据完整性。在一事务内由语句获取的全部封锁在事务期间被保持，防止其它并行事务的破坏性干扰。一个事务的 SQL 语句所做的修改在它提交后才可能为其它事务所见。

DM 自动维护数据库的一致性和完整性，并允许选择实施事务级读一致性，它保证同一事务内的可重复读，为此 DM 提供用户各种手动上锁语句和设置事务隔离级别语句。

本章介绍 DM 中和事务管理相关的 SQL 语句和手动上锁语句。在本章各例中，如不特别说明，各例的当前用户均为建表者 SYSDBA。

10.1 DM 事务相关语句

DM 中事务是一个逻辑工作单元，由一系列 SQL 语句组成。DM 把一个事务的所有 SQL 语句作为一个整体，即事务中的操作，要么全部执行，要么一个也不执行。

10.1.1 事务的开始

DM 没有提供显式定义事务开始的语句，第一个可执行的 SQL 语句(除登录语句外)隐含事务的开始。

10.1.2 事务的结束

用户可以使用显式的提交或回滚语句来结束一个事务，也可以隐式的结束一个事务。

1. 提交语句

语法格式

```
COMMIT [WORK];
```

参数

WORK　支持与标准 SQL 的兼容性，COMMIT 和 COMMIT WORK 等价。

功能

该语句使当前事务工作单元中的所有操作“永久化”，并结束该事务。

举例说明

例 1 插入数据到表 DEPARTMENT 并提交。

```
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('采购部门');
COMMIT WORK;
```

2．回滚语句

语法格式

```
ROLLBACK [WORK];
```

功能

该语句回滚(废除)当前事务工作单元中的所有操作，并结束该事务。

使用说明

建议用户退出时，用 COMMIT 或 ROLLBACK 命令来显式地结束应用程序。如果没有显式地提交事务，而应用程序又非正常终止，则最后一个未提交的工作单元被回滚。特别说明：CREATE TABLESPACE 和 ALTER DATABASE 两种 DDL 语句是不能回滚的。

举例说明

例 插入数据到表 DEPARTMENT 后回滚。

```
(1)往表 DEPARTMENT 中插入一个数据
INSERT INTO RESOURCES.DEPARTMENT(NAME) VALUES('销售部门');
(2)查询表 DEPARTMENT
SELECT * FROM RESOURCES.DEPARTMENT;
/*部门名为'销售部门'的记录可查询到*/
(3)回滚插入操作
ROLLBACK WORK;
(4)查询表 DEPARTMENT
SELECT * FROM RESOURCES.DEPARTMENT;
/*插入操作被回滚，表 DEPARTMENT 中不存在部门名为'销售部门'的记录*/
```

3．隐式的事务结束

事务开始后，当遇到下列情况时，该事务结束。该用户的下一个可执行的 SQL 语句将开始下一个新的事务。

1) 当一连接的属性设置为自动提交，每执行一条语句都会提交；
2) 在非自动提交事务中，当 DDL_AUTO_COMMIT 开关打开时，表示遇到任一 DDL 语句系统自动提交该 DDL 语句及之前的 DML 和 DDL 操作；当该开关关闭时，只有 CREATE TABLESPACE 和 ALTER DATABASE 两种 DDL 语句，物化视图刷新语句以及之前的 DML 和 DDL 操作会自动提交；
3) 事务所在的程序正常结束和用户退出；
4) 系统非正常终止时。

10.1.3 保存点相关语句

SAVEPOINT 语句用于在事务中设置保存点。保存点提供了一种灵活的回滚，事务在执行中可以回滚到某个保存点，在该保存点以前的操作有效，而以后的操作被回滚掉。一个事务中可以设置多个保存点。

1．设置保存点

语法格式

```
SAVEPOINT <保存点名>;
```

参数

<保存点名>　　指明保存点的名字。

使用说明

一个事务中可以设置多个保存点，但不能重名。

2．回滚到保存点

语法格式

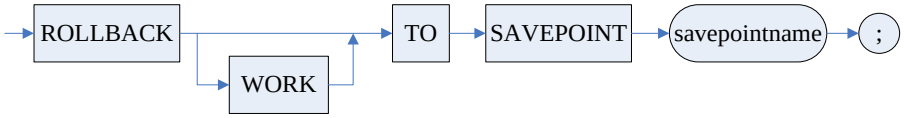
```
ROLLBACK [WORK] TO SAVEPOINT <保存点名>;
```

参数

(1)WORK　　支持与标准 SQL 的兼容性，ROLLBACK 和 ROLLBACK WORK 等价；

(2)<保存点名> 指明部分回滚时要回滚到的保存点的名字。

图例



使用说明

回滚到保存点后事务状态和设置保存点时事务的状态一致，在保存点以后对数据库的操作被回滚。

举例说明

例 插入数据到表 ADDRESS_TYPE 后设置保存点，然后再插入另一数据，回滚到保存点。

```
(1)往表 ADDRESS_TYPE 中插入一个数据
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('发货地址');
(2)查询表 ADDRESS_TYPE
SELECT * FROM PERSON.ADDRESS_TYPE;
/*地址类型名为'发货地址'的记录已经被插入到表中*/
(3)设置保存点
SAVEPOINT A;
(4)往表 ADDRESS_TYPE 中插入另一个数据
INSERT INTO PERSON.ADDRESS_TYPE(NAME) VALUES('家庭地址');
(5)回滚到保存点
ROLLBACK TO SAVEPOINT A;
(6)查询表 ADDRESS_TYPE
SELECT * FROM PERSON.ADDRESS_TYPE;
/*插入操作被回滚，ADDRESS_TYPE 中不存在地址类型名为'家庭地址'的记录*/
```

10.1.4 设置事务隔离级及读写特性

事务的隔离级描述了给定事务的行为对其它并发执行事务的暴露程度。通过选择两个隔离级中的一个，用户能增加对其它未提交事务的暴露程度，获得更高的并发度。DM 允许用户改变未启动的事务的隔离级和读写特性，即下列语句必须在事务开始时执行，否则无效。

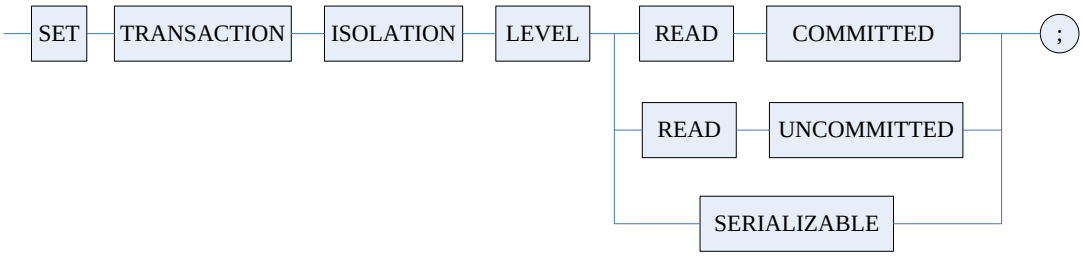
1．设置事务隔离级语句

事务的隔离级描述了给定事务的行为对其它并发执行事务的暴露程度。通过选择两个隔离级中的一个，用户能增加对其它未提交事务的暴露程度，获得更高的并发度。

语法格式

```
SET TRANSACTION ISOLATION LEVEL
[READ COMMITTED   |
READ UNCOMMITTED   |
SERIALIZABLE ];
```

图例



使用说明

1） 该语句设置事务的隔离级别：

----读提交(READ COMMITTED)：DM 默认级别，保证不读脏数据；

----读未提交(READ UNCOMMITTED)：可能读到脏数据；

----可串行化(SERIALIZABLE)：事务隔离的最高级别，事务之间完全隔离。

一般情况下，使用读提交隔离级别可以满足大多数应用，如果应用要求可重复读以保证基于查询结果的更新的正确性就必须使用可重复读或可串行读隔离级别。在访问只读表和视图的事务，以及某些执行 SELECT 语句的事务(只要其他事务的未提交数据对这些语句没有负面效果)时，可以使用读未提交隔离级。

2） 只能在事务未开始执行前设置隔离级，事务执行期间不能更改隔离级。

2．设置事务读属性的语句

语法格式

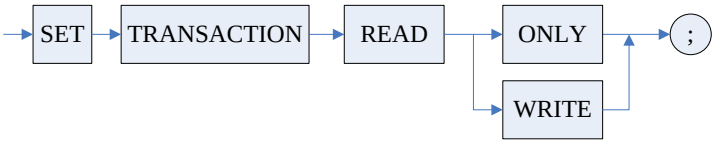
```
SET TRANSACTION READ ONLY | WRITE;
```

参数

(1)READ ONLY　　只读事务，该事务只能做查询操作，不能更新数据库；

(2)READ WRITE　　读写事务，该事务可以查询并更新数据库，是 DM 的默认设置。

图例



语句功能

该语句设置事务的读写属性。

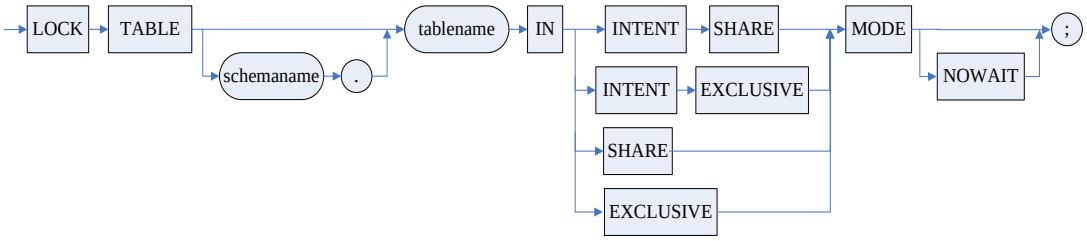
10.2 DM 手动上锁语句

DM 的隐式封锁足以保证数据的一致性，但用户可以根据自己的需要手工显式锁定表，允许或禁止在当前用户操作期间其它用户对此表的存取。DM 提供给用户四种表锁的封锁：意向共享锁(IS)、共享锁(S)、意向排它锁(IX)和排它锁(X)。

语法格式

```
LOCK TABLE [<模式名>.]<表名> IN <封锁方式> MODE [NOWAIT];
<封锁方式>::=
[INTENT SHARE ]|
[INTENT EXCLUSIVE] |
SHARE |
EXCLUSIVE
```

图例



使用说明

1. 意向共享表封锁：INTENT SHARE TABLE LOCKS (IS)

该封锁表明该事务封锁了表上的一些元组并试图修改它们（但是还未做修改，其它事务可读这些元组，但是不能修改这些元组）。意向共享表封锁是限制最少的锁，提供了表上最大的并发度。

1) 允许操作：

其他事务对该表的并发查询、插入、更新、删除或在该表上进行封锁，其他事务可以同时上意向共享锁(IS)、意向排他锁(IX)、共享锁(S)。

2) 禁止操作：

其它事务以排它锁方式(X)存取该表。

```
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

2. 意向排它表封锁：INETENSIVE EXCLUSIVE TABLE LOCKS (IX)

该锁表明该事务对表的元组进行一次或多次修改（其它事务不能访问这些元组）， 行排它表封锁较行共享表封锁稍严格。

1) 允许操作：

其它事务并行查询、插入、更新、删除或封锁该表上行，允许多个事务在同一表上获得意向排它(IX)和意向共享(IS)表锁。

2) 禁止操作：

其它事务对表进行共享(S)或者排它读写封锁(X)。

```
LOCK TABLE tablename IN SHARE MODE;
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

3. 共享表封锁：SHARE TABLE LOCKS (S)

该锁表明该事务访问表中所有元组，其他事务不能对该表做任何更新操作。

1) 允许操作：

其它事务在该表上作查询，但是不允许作修改，且允许多个事务在同一表上并发地持有共享表封锁(S)。

2) 禁止操作：

其它事务对表进行执行意向排他锁(IX)和排它锁方式(X)封锁。

```
LOCK TABLE tablename IN INTENT EXCLUSIVE MODE;
LOCK TABLE tablename IN EXCLUSIVE MODE;
```

4. 排它表封锁：EXCLUSIVE TABLE LOCKS (X)

该封锁是表封锁中最严格的方式，只允许持有封锁的事务可对该表进行修改。

1) 允许操作：

不允许任何操作。

2) 禁止操作：

其它事务对表执行任何 DML 语句，即不能插入、修改和删除该表中的行，封锁该表中的行或以任何方式封锁表。

用户上锁成功后锁将一直有效，直到当前事务结束时，该锁被系统自动解除；

5. 当使用 NOWAIT 时，若不能立即上锁成功则立刻返回报错信息，不再等待。

举例说明

例 当用户 SYSDBA 希望独占某表，他可以对该表显式地上排它锁。

```
LOCK TABLE PERSON.ADDRESS IN EXCLUSIVE MODE;
```

第 11 章 存储模块

DM 系统允许用户使用 DM PL/SQL 语言创建过程或函数，这些过程或函数像普通的过程或函数一样，有输入、输出参数和返回值，它们与表和视图等数据库对象一样被存储在数据库中，供用户随时调用。存储过程和存储函数在功能上相当于客户端的一段 SQL 批处理程序，但是在许多方面有着后者无法比拟的优点，它为用户提供了一种高效率的编程手段，成为现代数据库系统的重要特征。通常，我们将存储过程和存储函数统称为存储模块。

DM 的存储模块机制是一种技术，而不是一种独立的工具，它是和服务器紧密结合在一起的。可以认为这种技术是执行 DM PL/SQL 语言的一种机器，它可以接受任何有效的存储模块，按照语言本身所规定的语义执行，并将结果返回给客户。

DM 的存储模块机制具有如下优点：

1．提供更高的生产率：在设计应用时，围绕存储过程/函数设计应用，可以避免重复编码，提高生产率；在自顶向下设计应用时，不必关心实现的细节；编程方便。从 DM7 开始，PL/SQL 支持全部 C 语言语法，这样在对自定义的 PL/SQL 语法不熟悉的情况下也可以对数据库进行各种操作，这样可以使对数据库的操作更加灵活，也更加容易；

2．便于维护：用户的存储模块在数据库集中存放，用户可以随时查询、删除它们，而应用程序可以不作任何修改，或只做少量调整。存储模块能被其他的 PL/SQL 程序或 SQL 命令调用，任何客户/服务器工具都能访问 PL/SQL 程序，具有很好的可重用性；

3．提供更好的性能：存储模块在创建时被编译成伪码序列，在运行时不需要重新进行编译和优化处理，它具有更快的执行速度，可以同时被多个用户调用，并能够减少操作错误。使用存储模块可减少应用对 DM 的调用，降低系统资源浪费，显著提高性能，尤其是对在网络上与 DM 通讯的应用更显著；

4．安全性高：存储模块在执行时数据对用户是不可见的，提高了数据库的安全性。可以使用 DM 的数据工具管理存储在服务器中的存储模块的安全性。可以授权或撤销数据库其他用户访问存储模块的能力。

11.1 PL/SQL 数据类型和操作符

11.1.1 PL/SQL 数据类型

1. 数值数据类型

NUMERIC 类型

语法：NUMERIC[(精度 [, 标度])]

功能：NUMERIC 数据类型用于存储零、正负定点数。其中：精度是一个无符号整数，定义了总的数字数，精度范围是 1 至 38，标度定义了小数点右边的数字位数，定义时如省略精度，则默认是 16。如省略标度，则默认是 0。一个数的标度不应大于其精度。例如： NUMERIC(4,1)定义了小数点前面 3 位和小数点后面 1 位，共 4 位的数字，范围在 -999.9 到 999.9。所有 NUMERIC 数据类型，如果其值超过精度，达梦数据库返回一个出错信息，如果超过标度，则多余的位截断。

DEC 类型

语法：DEC[(精度 [, 标度])]

功能：与 DECIMAL 相同。

DECIMAL 类型

语法：DECIMAL[(精度 [, 标度])]

功能：与 NUMERIC 相似。

BIT 类型

语法：BIT

功能： BIT 类型用于存储整数数据 1、0 或 NULL，可以用来支持 ODBC 和 JDBC 的布尔数据类型。达梦数据库的 BIT 类型与 SQL SERVER2000 的 BIT 数据类型相似。

INTEGER 类型

语法：INTEGER

功能：用于存储有符号整数，精度为 10，标度为 0 。取值范围为：2147483648 (-2^31)～+2147483647(2^31-1) 。

INT 类型

语法：INT

功能：与 INTEGER 相同。

PLS_INTEGER 类型

语法：PLS_INTEGER

功能：与 INTEGER 相同。

BIGINT 类型

语法：BIGINT

功能：用于存储有符号整数，精度为 19，标度为 0。取值范围为：-9223372036854775808(-2^63)～9223372036854775807(2^63-1)。

TINYINT 类型

语法：TINYINT

功能：用于存储有符号整数，精度为 3，标度为 0 。取值范围为：-128 ～ +127。

BYTE 类型

语法：BYTE

功能：与 TINYINT 相似，精度为 3，标度为 0。

SMALLINT 类型

语法：SMALLINT

功能：用于存储有符号整数，精度为 5，标度为 0。

BINARY 类型

语法：BINARY[(长度)]

功能： BINARY 数据类型指定定长二进制数据。缺省长度为 1 个字节。 最大长度由数据库页面大小决定，具体算法与 1.4.1 节中介绍的相同。BINARY 常量以 0x 开始，后面跟着数据的十六进制表示。例如 0x2A3B4058。

VARBINARY 类型

语法： VARBINARY[(长度)]

功能： VARBINARY 数据类型指定变长二进制数据，用法类似 BINARY 数据类型，可以指定一个正整数作为数据长度。 缺省长度为 8188 个字节。最大长度由数据库页面大小决定，具体算法与 1.4.1 节中介绍的相同。

REAL 类型

语法：REAL

功能：REAL 是带二进制的浮点数，但它不能由用户指定使用的精度，系统指定其二进制精度为 24，

十进制精度为 7。取值范围-3.4E + 38 ～ 3.4E + 38。

FLOAT 类型

语法：FLOAT[(精度)]

功能：FLOAT 是带二进制精度的浮点数，精度最大不超过 53，如省略精度，则二进制精度为 53，十进制精度为 15。取值范围为-1.7E + 308 ～ 1.7E + 308。

DOUBLE 类型

语法：DOUBLE[(精度)]

功能：同 FLOAT 相似，精度最大不超过 53。

DOUBLE PRECISION 类型

语法：DOUBLE PRECISION

功能：该类型指明双精度浮点数，其二进制精度为 53，十进制精度为 15。取值范围-1.7E + 308 ～ 1.7E + 308。

2. 字符数据类型

CHAR 数据类型

语法： CHAR[(长度)]

定长字符串，最大长度由数据库页面大小决定，具体算法与 1.4.1 节中介绍的相同。长度不足时，自动填充空格

CHARACTER 类型

语法：CHARACTER[(长度)]

功能：与 CHAR 相同。

VARCHAR 数据类型

语法： VARCHAR[(长度)]

可变长字符串，最大长度由数据库页面大小决定，具体算法与 1.4.1 节中介绍的相同。

3. 多媒体数据类型

LOB(大对象，Large Object) 数据类型用于存储类似图像，声音这样的大型数据对象，LOB 数据对象可以是二进制数据也可以是字符数据，其最大长度不超过 2G。

在 PL/SQL 中操作 LOB 数据对象可分为以下 6 类：BLOB，CLOB，TEXT，IMAGE，LONGVARBINARY，LONGVARCHAR。

4. 日期数据类型

DATE 数据类型

语法： DATE

日期类型，包括年、月、日信息。数据格式：date ‘1999-10-01’

TIME 数据类型

语法： TIME

时间类型，包括时、分、秒信息。格式：time ’09:10:21’

TIMESTAMP 数据类型

语法： TIMESTAMP

时间戳型，包括年月日时分秒信息。例如：timestamp ‘1999-07-13 10:11:22’

TIME WITH TIME ZONE 类型

语法：TIME［(小数秒精度)］WITH TIME ZONE

描述一个带时区的 TIME 值，其定义是在 TIME 类型的后面加上时区信息。例如：TIME '09:10:21 +8:00'

TIMESTAMP WITH TIME ZONE 类型

语法：TIMESTAMP［(小数秒精度)］WITH TIME ZONE

描述一个带时区的 TIMESTAMP 值，其定义是在 TIMESTAMP 类型的后面加上时区信息。例如：TIMESTAMP '2002-12-12 09:10:21 +8:00'

5. BOOL 类型/BOOLEAN 类型

语法：BOOL

描述 BOOL 数据类型：TRUE 和 FALSE。达梦的 BOOL 类型和 INT 类型可以相互转化。如果变量或方法返回的类型是 BOOL 类型，则返回值为 0 或 1。TRUE 和非 0 值的返回值为 1，FALSE 和 0 值返回为 0。

BOOLEAN 与 BOOL 类型用法完全相同。

6. %TYPE 和%ROWTYPE

在许多情况下，存储模块变量可以被用来处理存储在数据库表中的数据。在这种情况下，变量应该拥有与表列相同的类型。例如表 T(ID INT, NAME VARCHAR(30))中有个字段 NAME 类型为 VARCHAR(30)。对应的在存储模块中，我们可以声明一个变量： DELCARE V_NAME VARCHAR(30)，但是如果 T 中的 NAME 字段定义发生了变化，比如变为 VARCHAR(100)。那么存储模块中的变量 V_NAME 也要做相应修改为 DELCARE V_NAME VARCHAR(100)。如果您有很多的变量以及存储过程代码，这种处理可能是十分耗时和容易出错的。

为了解决上述问题，DM 提供了%TYPE 类型。%TYPE 可以附加在表列或者另外一个变量上，并返回其类型。例如：DECLARE V_NAME T.NAME % TYPE 通过使用%TYPE，V_NAME 将拥有 T 表的 NAME 列所拥有的类型；如果表 T 的 NAME 列类型定义发生变化，V_NAME 的类型也随之自动发生变化，而不需要用户手动修改。如下面的例子：

```
CREATE TABLE T(ID INT, NAME VARCHAR(30));
INSERT INTO T VALUES(1, 'ZHANGSAN');

CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
    V1 T.NAME %TYPE;
    C1 CURSOR;
BEGIN
    OPEN C1 FOR SELECT NAME FROM T WHERE ID = 1;
    FETCH C1 INTO V1;
    PRINT V1;
    CLOSE C1;
END;
```

与%TYPE 类似，%ROWTYPE 将返回一个基于表定义的运算类型，它将一个记录声明为具有相同类型的数据库行。例如：DECLARE V_TREC T % ROWTYPE 将定义一个记录，该记录中的字段与表 T 中的列相对应。V_TREC 变量会拥有下面这样的结构：(ID INT, NAME VARCHAR(30))。与%TYPE 类似，如果表定义改变了，那么%ROWTYPE 定义的变量也会随之改变。如下面的例子：

```
CREATE OR REPLACE PROCEDURE PROC AS
DECLARE
    V1 T % ROWTYPE;
```

```
    C1 CURSOR;
BEGIN
    OPEN C1 FOR SELECT * FROM T WHERE ID = 1;
    FETCH C1 INTO V1;
    PRINT V1.ID; PRINT V1.NAME;
    CLOSE C1;
END;
```

7. 记录类型

记录类型类似于 C 语句的结构。记录类型提供了处理分立但又做为一个整体单元的相关变量的一种机制。例如：DECLARE V_ID INT; V_NAME VARCHAR(30); 这两个变量在逻辑上是相互关联的，因为它们对应 T 表中的相同字段。如果为这些变量声明一个记录类型，那么它们之间的关系就十分明显了。定义记录类型的语法如下：

```
TYPE  记录类型名  IS RECORD
(记录字段名 1  数据类型, [记录字段名 2   数据类型, ……]);
```

记录类型可以单独对记录中的字段赋值，也可以将一个记录直接赋值给另外一个记录。可以使用点标记引用一个记录中的字段（记录名.字段名）。如下面的例子：

```
DECLARE
    TYPE T_REC IS RECORD( ID INT, NAME VARCHAR(30));
    V_REC1 T_REC;
    V_REC2 T_REC;
BEGIN
    V_REC1.ID := 100;
    V_REC1.NAME:= '雷锋';
    V_REC2 := V_REC1;
    PRINT V_REC1.ID;PRINT V_REC1.NAME;
    PRINT V_REC2.ID;PRINT V_REC2.NAME;
END;
```

如果要进行记录赋值，那么这两个记录中的字段类型定义必须完全一致。如下面的例子所示：

```
DECLARE
    TYPE T_REC IS RECORD( ID INT, NAME VARCHAR(30));
    TYPE T_REC_NEW IS RECORD( ID INT, NAME VARCHAR(30));
    V_REC1 T_REC;
    V_REC2 T_REC_NEW;
BEGIN
    V_REC1.ID := 100; V_REC1.NAME:= '雷锋';
    V_REC2 := V_REC1;
    PRINT V_REC2.ID;
    PRINT V_REC2.NAME;
END;
```

8. 数组类型

数组数据类型包括静态数组类型和动态数组类型。静态数组是在声明时已经确定了数

组大小的数组，其长度是预先定义好的，在整个程序中，一旦给定大小后就无法改变。而动态数组则不然，它可以随程序需要而重新指定大小。动态数组的内存空间是从堆（HEAP）上分配（即动态分配）的，通过执行代码而为其分配存储空间。当程序执行到这些语句时，才为其分配，程序员自己负责释放内存。需要注意的是，DM 中数组下标的起始值为 1。理论上 DM 支持静态数组的每一个维度的最大长度为 65534，动态数组的每一个维度的最大长度为 2147483646，但是数组最大长度同时受系统内部堆栈（静态数组）和堆（动态数组）空间大小的限制，如果超出堆栈/堆的空间限制，系统会报错。

静态数组的语法格式为：

```
TYPE 数组名 IS ARRAY 数据类型 [常量表达式, 常量表达式, …];
```

静态数组使用例子如下：

```
DECLARE
    TYPE ARR IS ARRAY INT[3];          --TYPE 定义数组类型
    A ARR;                                  --用自己定义的数组类型申明数组
    TYPE ARR1 IS ARRAY INT[2,3];
    B ARR1;                                 --多维数组
BEGIN
    FOR I IN 1..3 LOOP   --TYPE 定义的数组
        A[I] := I * 10;
        PRINT A[I];
    END LOOP;
    FOR I IN 1..2 LOOP
      FOR J IN 1..3 LOOP
          B[I][J] := I * 10 + J;
          PRINT B[I][J];
      END LOOP;
    END LOOP;
END;
```

动态数组与静态数组的用法类似，区别只在于动态数组没有指定下标，需要动态分配空间。动态数组的语法格式为：

```
TYPE 数组名 IS ARRAY 数据类型 [,…];
```

多维动态数组分配空间的语法为：

```
数组名 := NEW 数据类型[常量表达式,…];
```

或者可以使用如下两种语法对多维数组的某一维度进行空间分配。其中，第二种方式是使用自定义类型来创建动态数据，前提是先定义好一个类型，该方式适用于含有精度或长度的数据类型。

```
数组名:= NEW 数据类型[常量表达式][];
数组名:= NEW 自定义类型[常量表达式][];
```

动态数组使用例子如下：

```
DECLARE
    TYPE ARR IS ARRAY INT[];
    A ARR;
    BEGIN
    A := NEW INT[3];          --动态分配空间
    FOR I IN 1..3 LOOP
```

```
        A[I] := I * 10;
        PRINT A[I];
        END LOOP;
        PRINT ARRAYLEN(A);    --函数 ARRAYLEN 用于求取数组的长度
    END;
```

自定义类型定义动态数组如下：

```
    DECLARE
        TYPE VAR IS VARCHAR(100) ;
        TYPE VARARR IS ARRAY VAR[];
        B VARARR;
    BEGIN
        B = NEW VAR[4];    --动态分配空间
        FOR I IN 1..3 LOOP
            B[I] := I * 11;
            PRINT B[I];
        END LOOP;
        PRINT ARRAYLEN(B);    --函数 ARRAYLEN 用于求取数组的长度
    END
```

多维动态数组：

```
    DECLARE
        TYPE ARR IS ARRAY INT[ , ];
        A ARR;
    BEGIN
        A := NEW INT[3][];            --动态分配第一维空间
        FOR I IN 1..3 LOOP
            A[I] := NEW INT[4];          --动态分配第二维空间
            FOR J IN 1 .. 4 LOOP
                A[I][J] := I * J;
                PRINT A[I][J];
            END LOOP;
        END LOOP;
        PRINT ARRAYLEN(A);    --函数 ARRAYLEN 用于求取数组的长度
    END;
```

也可以对多维动态数组一次分配空间，则上面的例子可写为：

```
    DECLARE
        TYPE ARR IS ARRAY INT[ , ];
        A ARR;
    BEGIN
        A := NEW INT[3,4];            --为二维动态数组一次性分配空间
        FOR I IN 1..3 LOOP
            FOR J IN 1 .. 4 LOOP
                A[I][J] := I * J;
                PRINT A[I][J];
            END LOOP;
```

```
        END LOOP;
        PRINT ARRAYLEN(A);   --函数 ARRAYLEN 用于求取数组的长度
    END;
```

DM 还支持索引表的数组，如下例所示。索引表的具体介绍参看下一部分索引表。

```
    DECLARE
        TYPE ARR IS TABLE OF INT INDEX BY INT;   --这种方式只能定义一维数组
        A ARR;
    BEGIN
            FOR I IN 1..3 LOOP
            A(I) := I * 10;
            PRINT A(I);
        END LOOP;
        PRINT A.COUNT;     --返回集合中元素的个数
    END;
```

9. 集合类型

1) VARRAY 类型

VARRAY（varying array）是一种具有可伸缩性的数组，它有一个最大容量。VARRAY 的下标是从 1 开始的有序数字，并提供多种方法操作数组中的项。

定义 VARRAY 的语法格式为：

```
    TYPE 数组名 IS VARRAY(常量表达式) OF 数据类型;
```

数据类型可以是基础数据类型，也可以是其它自定义类型或是对象、记录、其它 VARRAY 类型等，使得构造复杂的结构成为可能。

下面给出一个简单的 VARRAY 使用示例。

```
    DECLARE
        TYPE my_array_type IS VARRAY(10) OF INTEGER;
        v MY_ARRAY_TYPE;
        i, k INTEGER;

    BEGIN
        v=my_array_type(5,6,7,8);
        k=v.COUNT();
        PRINT 'V.COUNT()=' || k;

        FOR i IN 1..v.COUNT() LOOP
            PRINT 'V(' || i || ')=' ||v(i);
        END LOOP;
    END;
```

2) 索引表

索引表提供了一种快速，方便地管理一组相关数据的方法，是程序设计中的重要内容。通过索引表可以对大量性质相同的数据进行存储，排序，插入及删除等操作，从而可以有效地提高程序开发效率及改善程序的编写方式。

内存索引表是一组数据的集合，将数据按照一定规则组织起来，形成一个可操作的整体，

是对大量数据进行有效组织和管理的手段之一，通过函数可以对大量性质相同的数据进行存储，排序，插入及删除等操作，从而可以有效地提高程序开发效率及改善程序的编写方式。

内存索引表和数组类似，只是内存索引表使用起来更加方便，但是性能不如数组。数组在定义时需要用户显示指定数组的大小，当用户访问大小之外的数组元素时，系统会报错；内存索引表相当于一个一维数组，但却不需要用户指定大小，大小根据用户的操作自动增长。

语法格式为：

```
TYPE  内存索引表名  IS TABLE OF 数据类型  INDEX BY  数据类型;
```

第一个数据类型是索引表存放数据的类型，这个数据类型可以是基础数据类型，也可以是其它自定义类型或是对象、记录、静态数组，但是不能是动态数组；第二个则是索引表的的下标类型，目前仅支持 INTEGER/INT 和 VARCHAR 两种类型，分别代表整数下标和字符串下标。对于 VARCHAR 类型，长度不能超过 1024。

索引表的成员函数可以用来遍历索引表，或查看索引表的信息。使用例子如下：

```
DECLARE
TYPE ARR IS TABLE OF INT INDEX BY INT;
AR ARR;
C INT;
BEGIN
        AR(1) = 1;
C := AR.COUNT;
PRINT C;     -- 打印值为 1 表示里面有一个元素
END;
```

索引表的例子如下：

普通的索引表：

```
DECLARE
TYPE ARR IS TABLE OF VARCHAR(100) INDEX BY INT;
X ARR;
BEGIN
        X(1) := 'TEST1';
        X(2) := 'TEST2';
        X(3) := X(1) + X(2);
        PRINT X(3);
END;
```

索引表用来存游标记录：

```
DECLARE
TYPE ARR IS TABLE OF VARCHAR(200) INDEX BY INT;
X ARR;
I INT;
CURSOR C1;
BEGIN
    I := 1;
    OPEN C1 FOR SELECT NAME FROM SYSOBJECTS;
    LOOP
        IF C1%NOTFOUND THEN
            EXIT;
```

```
            END IF;
            FETCH C1 INTO X(I);              --遍历结果集，把每行的值都存放索引表中
            I := I + 1;
        END LOOP;
        I = X."FIRST"(); --遍历输出索引表中的记录
        LOOP
            IF I IS NULL THEN
                EXIT;
            END IF;
            PRINT X(I);
            I = X."NEXT"(I);
        END LOOP;
    END;
```

索引表管理记录：

```
    DECLARE
        TYPE RD IS RECORD(ID INT, NAME VARCHAR(128));
    TYPE ARR IS TABLE OF RD INDEX BY INT;
    X ARR;
    I INT;
    CURSOR C1;
    BEGIN
        I := 1;
        OPEN C1 FOR SELECT ID, NAME FROM SYSOBJECTS;
        LOOP
            IF C1%NOTFOUND THEN
                EXIT;
            END IF;
            FETCH C1 INTO X(I).ID, X(I).NAME;        --遍历结果集，把每行的值都存放索引表中
            I := I + 1;
        END LOOP;

        I = X."FIRST"(); --遍历输出索引表中的记录
        LOOP
            IF I IS NULL THEN
                EXIT;
            END IF;
            PRINT 'ID:' + CAST(X(I).ID AS VARCHAR(10)) + ', NAME:' + X(I).NAME;
            I = X."NEXT"(I);
        END LOOP;
    END;
```

多维索引表的遍历：

```
    DECLARE
    TYPE ARR IS TABLE OF VARCHAR(100) INDEX BY BINARY_INTEGER;
    TYPE ARR2 IS TABLE OF ARR INDEX BY VARCHAR(100);
```

```
X ARR2;
IND_I INT;
IND_J VARCHAR(10);
BEGIN
    FOR I IN 1 .. 100 LOOP
        FOR J IN 1 .. 50 LOOP
            X(I)(J) := CAST(I AS VARCHAR(100)) + '+' + CAST(J AS VARCHAR(10));
        END LOOP;
    END LOOP;
    --遍历多维数组
    IND_I := X."FIRST"();
    LOOP
        IF IND_I IS NULL THEN
            EXIT;
        END IF;
        IND_J := X(IND_I)."FIRST"();
        LOOP
            IF IND_J IS NULL THEN
                EXIT;
            END IF;
            PRINT X(IND_I)(IND_J);
            IND_J := X(IND_I)."NEXT"(IND_J);
        END LOOP;
        IND_I := X."NEXT"(IND_I);
    END LOOP;
END;
```

3) 嵌套表

嵌套表元素的下标从 1 开始，且元素个数没有限制。

嵌套表语法如下：

```
TYPE 嵌套表名 IS TABLE OF 元素数据类型;
```

元素数据类型用来指明嵌套表元素的数据类型，当元素数据类型为一个定义了某个表字段的对象类型时，嵌套表就是某些行的集合，实现了表的嵌套功能。

下面是一个嵌套表的使用示例。

```
CREATE TABLE myinfo(sno INT, name VARCHAR, age INT);

DECLARE
    TYPE info_t IS TABLE OF myinfo%ROWTYPE;
    info info_t;
BEGIN
SELECT sno, name, age BULK COLLECT INTO info FROM myinfo;
END;
```

集合类型支持的成员函数如下：

1） COUNT

参数：无

功能：返回集合中元素的个数。

2） DELETE([i])

参数：要删除元素的下标。

功能：i 参数省略时删除集合中所有元素，否则删除元素下标为 i 的元素，如果 i 为 null，则集合保持不变。

3） DELETE(i, j)

参数：i，要删除的第一个元素下标；j，要删除的最后一个元素下标；

功能：删除集合中下标从 i 到 j 的所有元素。

4） EXISTS(i)

参数：要判断的元素下标。

功能：如果集合元素 i 已经初始化，则返回 TRUE, 否则返回 FALSE。

5） FIRST

功能：返回集合中的第一个元素的下标号，对于 VARRAY 集合始终返回 1

6） LAST

功能：返回集合中最后一个元素的下标号，对于 VARRAY 返回值始终等于 COUNT。

7） NEXT(i)

参数：指定起始的元素下标。

功能：返回在元素 i 之后紧挨着它的元素的下标号，如果该元素是最后一个元素，则返回 null。

8） PRIOR(i)

参数：指定起始元素的下标。

功能：返回在元素 i 之前紧挨着它的元素的下标号，如果该元素是第一个元素，则返回 null。

9） LIMIT

参数：无

功能：返回 VARRY 集合的最大的元素个数。

说明：对内存索引表和嵌套表不适用。

10）EXTEND([n])

参数：扩展元素的个数。

功能：n 参数省略时扩展一个空元素，否则扩展 n 个空元素。

11）EXTEND(n, i)

参数：n，扩展元素的个数；i，要复制的元素小表。

功能：在集合末尾扩展 n 个与第 i 值相同的元素。

说明：对内存索引表不适用。

12）TRIM([n])

参数：删除元素的个数。

功能：n 参数省略时从集合末尾删除一个元素，否则从集合末尾开始删除 n 个元素。

说明：对内存索引表不适用。

10. 类类型

在 DM7 中，新增了两种 PL/SQL 的类型，分别是游标类类型及异常类类型，这两种类

型也就是在后面介绍到的游标操作及异常处理的实现方式。

11. 子类型

子类型相当于给类型定义一个别名，可以在任何 PL/SQL 块、子程序或包中定义自己的子类型，语法如下：

```
SUBTYPE subtype_name IS base_type[(精度,[刻度])] [NOT NULL];
```

一旦定义了子类型，就可以声明该类型的变量、常量等。

11.1.2 PL/SQL 操作符

与其他程序设计语言相同，PL/SQL 有一系列操作符。操作符分为下面几类：

1. 算术操作符；
2. 关系操作符；
3. 比较操作符；
4. 逻辑操作符。

算术操作符如表 11.1.1 所示

表 11.1.1 算术操作符表

操作符	对应操作
+	加
-	减
/	除
*	乘

关系操作符主要用于条件判断语句或用于 where 子串中，关系操作符检查条件和结果是否为 true 或 false，表 11.1.2 是 PL/SQL 中的关系操作符，表 11.1.3 显示的是比较操作符，表 11.1.4 显示的是逻辑操作符。

表 11.1.2 关系操作符表

操作符	对应操作
<	小于操作符
<=	小于或等于操作符
>	大于操作符
>=	大于或等于操作符
=	等于操作符
!=	不等于操作符
<>	不等于操作符
:=	赋值操作符

表 11.1.3 比较操作符

操作符	对应操作
IS NULL	如果操作数为 NULL 返回 TRUE
LIKE	比较字符串值
BETWEEN	验证值是否在范围之内

IN	验证操作数在设定的一系列值中

表 11.1.4 逻辑操作符

操作符	对应操作
AND	两个条件都必须满足
OR	只要满足两个条件中的一个
NOT	取反

11.2 存储模块的定义

本节中，我们将介绍如何使用 DM 的 PL/SQL 语言来定义一个存储模块。为使用户获得对存储模块的整体印象，本节中提供了一些例子，对于例子中涉及到的各种控制语句，其详细的使用方法，请参阅 11.5 节“存储模块的控制语句”。

定义一个存储模块的详细语法如下：

```
<存储模块定义语句> ::=
<存储过程定义语句> |
<存储函数定义语句>
<存储过程定义语句> ::= CREATE [OR REPLACE] <过程>
<过程>::= <过程申明> [AS | IS <模块体>]
<过程申明> ::= PROCEDURE <过程名定义> [WITH ENCRYPTION] [ (<参数列>)]
<过程名定义> ::=[ <模式名>.]<过程名>
<存储函数定义语句> ::= CREATE [OR REPLACE] <函数>
<函数> ::= <函数申明> [AS | IS <模块体>]
<函数申明>::= FUNCTION <函数名定义> [WITH ENCRYPTION] [FOR CALCULATE] [(<参数列>)] RETURN <返回数据类型> [PIPELINED]
<函数名定义>::=[ <模式名>.]<函数名>
<参数列> ::= <参数申明>[{, <参数申明>}…]
<参数申明> ::= <参数名> [<参数模式>] <参数类型> [ DEFAULT | ASSIGN | := <值表达式>]
<参数模式> ::= IN | OUT | [IN OUT] | [OUT IN]
<模块体> ::=
[<说明部分>]
BEGIN
<执行部分>
[<异常处理部分>]
END [过程名|函数名]
<说明部分> ::=[DECLARE]<说明定义>{<说明定义>}
<说明定义>::=<变量说明>|<异常变量说明>|<游标定义>|<子过程定义>|<子函数定义> ;
<变量说明>::=<变量名>{,<变量名>}[CONSTANT]<变量类型>[NOT NULL][DEFAULT|ASSIGN|:=<表达式>]
<变量类型>::=<PLSQL 类型>|<[模式名.]表名.列名%TYPE>|<[模式名.]表名%ROWTYPE>|<记录类型>
<记录类型>::= RECORD(<变量名> <PLSQL 类型>{,<变量名> <PLSQL 类型>})
<异常变量说明>::=<异常变量名>EXCEPTION[FOR<错误号>]
<游标定义>::=CURSOR <游标名> [FOR<不带 INTO 查询表达式>|<表连接>]
```

<子过程定义>::=PROCEDURE<过程名>[(<参数列>)]<IS|AS><模块体>

<子函数定义>::=FUNCTION<函数名>[(<参数列>)]RETURN<返回数据类型> [PIPELINED] <IS|AS><模块体>

<执行部分>::=<SQL 过程语句序列>{;< SQL 过程语句序列>}

< SQL 过程语句序列>::=[<标号说明>]<SQL 过程语句>;

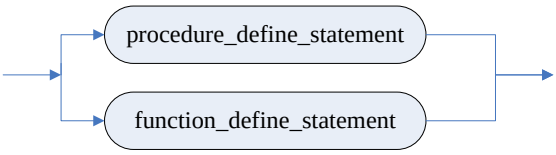
<标号说明>::=<<<标号名>>>

<SQL 过程语句>::=<SQL 语句>|<SQL 控制语句>

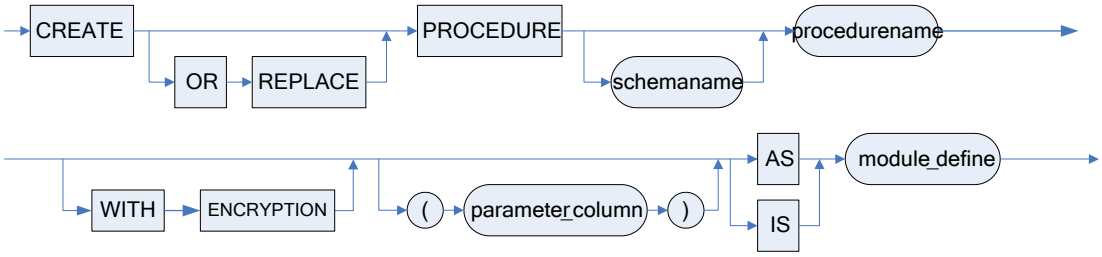
<异常处理部分>::=EXCEPTION<异常处理语句>{;<异常处理语句>}

<异常处理语句>::= WHEN <异常名> THEN < SQL 过程语句序列>

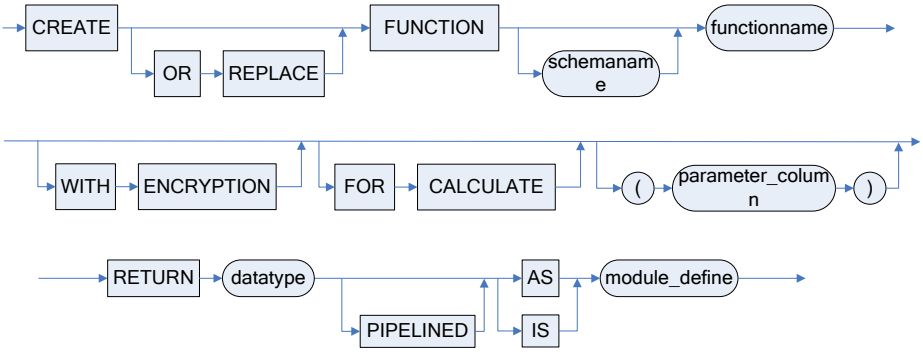
图例



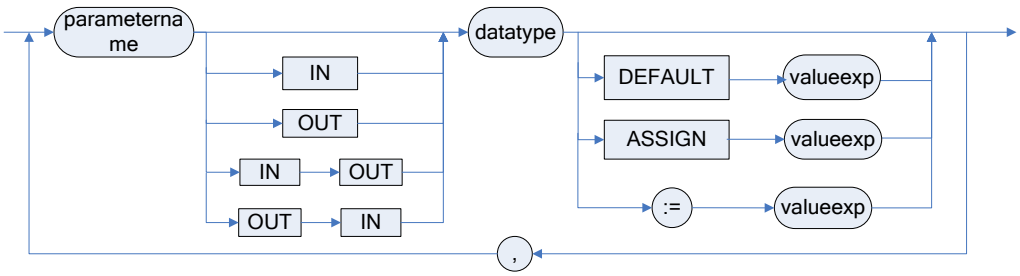
procedure_define_statement



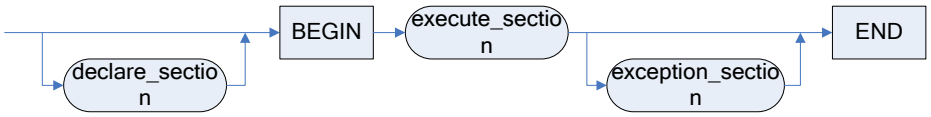
function_define_statement



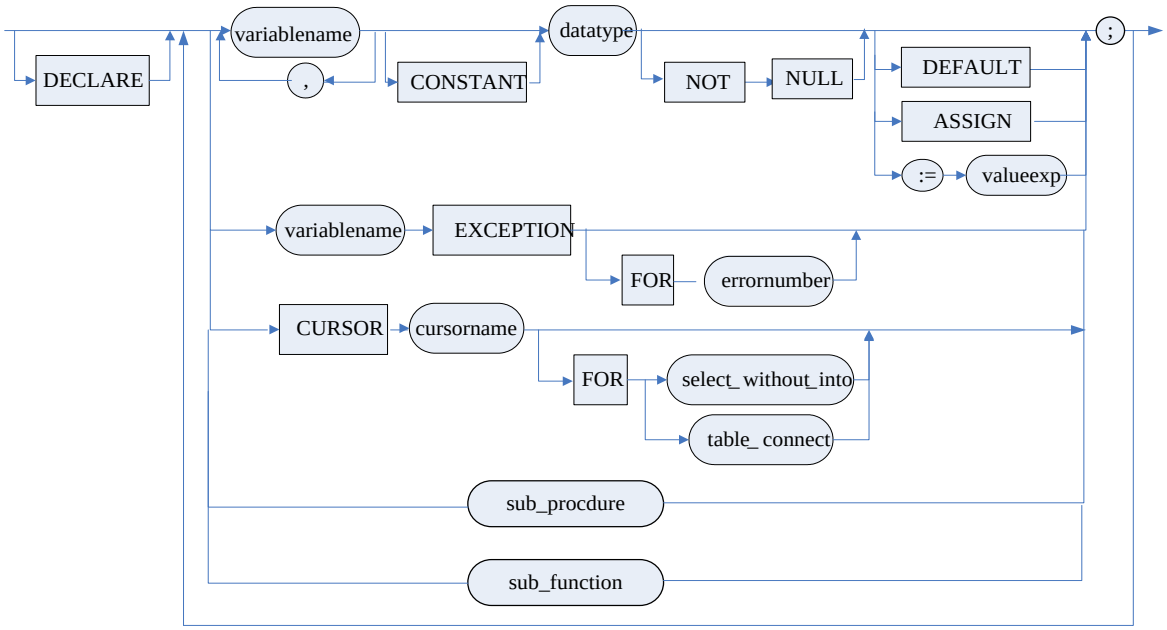
parameter_column



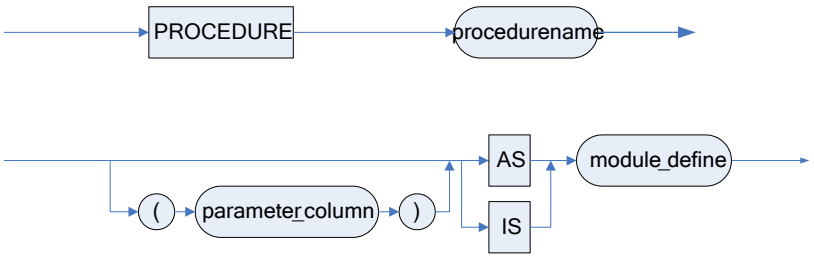
module_define



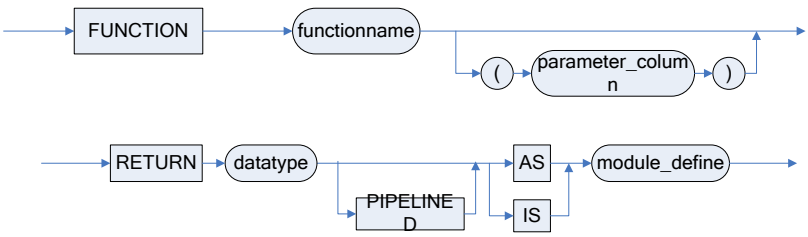
declare_section



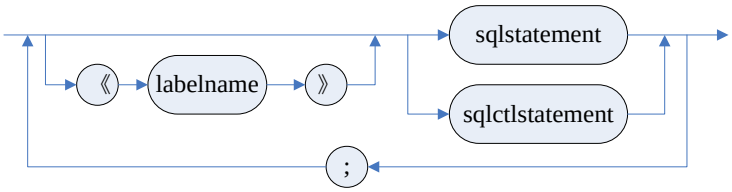
sub_procdure



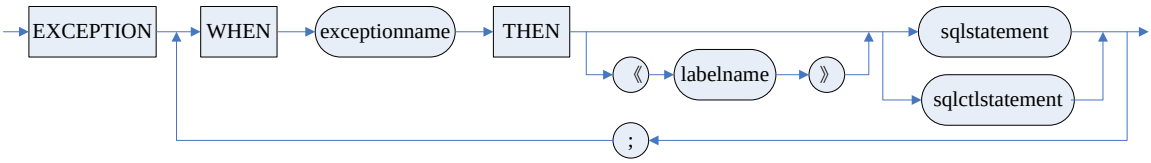
sub_function



execute_section



exception_section



1．存储过程

定义一个存储过程，应使用下面的语句：

语法格式

```
CREATE [OR REPLACE ] PROCEDURE <存储过程名定义> [WITH  ENCRYPTION]
[(<参数名> <参数模式> <参数类型> [<默认值表达式>]
 {,<参数名> <参数模式> <参数类型>  [<默认值表达式>] })]
AS | IS
[<说明部分>]
BEGIN
<执行部分>
[<异常处理部分>]
END;
<存储过程名定义> ::=[<模式名>.] <存储过程名>
```

参数

1) <存储过程名> 指明被创建的存储过程的名字；
2) <模式名> 指明被创建的存储过程所属模式的名字，缺省为当前模式名；
3) <参数名> 指明存储过程参数的名称；
4) <参数模式> 参数模式可设置为 IN、OUT 或 IN OUT（OUT IN），缺省为 IN 类型；
5) <参数类型> 指明存储过程参数的数据类型；
6) <说明部分> 由变量、游标和子程序等对象的申明构成，可缺省；
7) <执行部分> 由 SQL 语句和过程控制语句构成的执行代码；
8) <异常处理部分>各种异常的处理程序，存储过程执行异常时调用，可缺省。

使用说明

1) DM_SQL 过程语言支持的 SQL 语句包括：数据查询语句(SELECT)，数据操纵语句(INSERT、DELETE、UPDATE)，游标定义及操纵语句(DECLARE CURSOR、OPEN、FETCH、CLOSE)，事务控制语句(COMMIT、ROLLBACK)，动态 SQL 执行语句(EXECUTE IMMEDIATE)。SQL 语句必须以分号结尾，否则语法分析报错；

2) DM_SQL 过程控制语句包括：语句块、赋值语句、IF 语句、LOOP 语句、WHILE 语句、FOR 语句、EXIT 语句、调用语句、RETURN 语句、NULL 语句、GOTO 语句、RAISE 语句和打印语句等；

3) 存储过程中的 RETURN 语句不允许带返回值；

4) 在定义存储过程时使用赋值符号':='或关键字 DEFAULT，可以为 IN 参数指定一个

缺省值。拥有缺省参数的参数必须在参数列表的尾端；

权限

使用该语句的用户必须是 DBA 或该存储过程的拥有者且具有 CREATE PROCEDURE 数据库权限的用户。对象重建后，不能保证原有权限的合法性，所以全部去除。

举例说明

例 在模式 USER1 下创建一个简单的存储过程：

```
CREATE OR REPLACE PROCEDURE USER1.PROC_1(A IN OUT INT)    AS
B    INT;
BEGIN
A:=A+B;
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/
```

存储过程调用方法如下：

```
CALL USER1.PROC_1(1);
```

第 2 行是该存储过程的说明部分，这里申明了一个变量 B。注意在 DMPL/SQL 语言中说明变量时，变量的类型放在变量名称之后。第 3 行和第 4 行是该程序块运行时被执行的代码段，这里将 A 与 B 的和赋给参数 A。如果发生了异常，第 5 行开始的异常处理部分就对产生的异常情况进行处理，WHEN OTHERS 异常处理器处理所有不被其他异常处理器处理的异常，它必须是最后一个异常处理器。第 11.7 节详细介绍了异常的处理过程。说明部分和异常处理部分都是可选的。如果用户在模块中不需要任何局部变量或者不想处理发生的异常，则可以省略这两部分。

2．存储函数

存储函数和存储过程很相似，它们的区别在于：

1) 存储过程没有返回值，而存储函数有；
2) 存储过程中可以没有返回语句，而存储函数必须通过返回语句结束；
3) 存储过程的返回语句中不能带表达式，而存储函数必须带表达式；
4) 存储过程不能出现在一个表达式中，而存储函数只能出现在表达式中。

定义一个存储函数，应使用下面的语句：

语法格式

```
CREATE [OR REPLACE ] FUNCTION <存储函数名定义> [WITH  ENCRYPTION] [FOR CALCULATE][(<参数名> <参数模式> <参数类型> [<默认值表达式>]{,<参数名> <参数模式> <参数类型> [<默认值表达式>]})]
RETURN <返回数据类型>[PIPELINED]
AS | IS
[<说明部分>]
BEGIN
<执行部分>
[<异常处理部分>]
END;
<存储函数名定义> ::=[<模式名>.] <函数名>
```

参数

1) <存储函数名> 指明被创建的存储函数的名字；
2) <模式名> 指明被创建的存储函数所属模式的名字，缺省为当前模式名；
3) FOR CALCULATE 指定函数为计算函数，只做计算用；
4) <参数名> 指明存储函数参数的名称；
5) <参数模式> 参数模式可设置为 IN、OUT 或 IN OUT（OUT IN），缺省为 IN 类型；
6) <参数类型> 指明存储函数参数的数据类型；
7) <返回数据类型> 指明存储函数返回值的数据类型；
8) PIPELINED 指明函数为管道表函数。

使用说明

1) DM_SQL 过程语言支持的 SQL 语句包括：数据查询语句(带 INTO 的 SELECT 语句)，数据操纵语句(INSERT、DELETE、UPDATE)，游标定义及操纵语句(DECLARE CURSOR、OPEN、FETCH、CLOSE)，事务控制语句(COMMIT、ROLLBACK)， 动态 SQL 执行语句(EXECUTE IMMEDIATE)。SQL 语句必须以分号结尾，否则语法分析报错；

2) DM_SQL 过程控制语句包括：语句块、赋值语句、IF 语句、LOOP 语句、WHILE 语句、FOR 语句、EXIT 语句、调用语句、RETURN 语句、GOTO 语句、RAISE 语句和打印语句等；

3) 计算函数中不支持对表进行 insert、delete、update、select、上锁、设置自增列属性；对游标 declare、open、fetch、close；事务的 commit、rollback、savepoint、设置事务的隔离级别和读写属性；动态 sql 的执行 exec、创建 index、创建子过程。对于计算函数体内的函数调用必须是系统函数或者计算函数；

4) 在存储函数中必须使用 RETURN 语句向函数的调用环境返回一个值；

5) 存储函数可以用 CALL 语句调用，也可以出现在表达式中；

6) 如果指定函数为管道表函数，返回类型必须是集合类型（嵌套表或变长数组）。函数通过 PIPE ROW 语句将单个结果元组写入到结果集中。

权限

使用该语句的用户必须是 DBA 或该函数的拥有者且具有 CREATE PROCEDURE 数据库权限的用户。

举例说明

例 在模式 USER1 下创建一个简单的存储函数。

```
CREATE OR REPLACE FUNCTION USER1.FUN_1 (A INT, B INT) RETURN INT AS
        S   INT;
BEGIN
   S:=A+B;
   RETURN S;
EXCEPTION
   WHEN OTHERS THEN NULL;
END;
/
调用语句：
select USER1.FUN_1(1,2);
返回结果为 3
```

第 1 行说明了该函数的返回类型为 INT 类型。第 4 行将两个参数 A、B 的和赋给了变量 S，第 5 行的 RETURN 语句则将变量 S 的值作为函数的返回值返回。

例 创建一个计算函数 F1，并在表中使用。

```
CREATE OR REPLACE FUNCTION F1 FOR CALCULATE
RETURN INT
IS
BEGIN
RETURN 1;
```

```
END;
/
--在 T 表中使用
CREATE TABLE T(A INT,B INT DEFAULT F1());
或者 CREATE TABLE T(A INT,B INT DEFAULT F1);
```

3．参数

存储模块及模块中定义的子模块都可以带参数，用来给模块传送数据及向外界返回数据。在过程或函数中定义一个参数时，必须说明名称、参数模式和数据类型。三种可能的参数模式是，IN(缺省模式)、OUT 和 IN OUT，意义分别为：

1) IN：输入参数，用来将数据传送给模块；
2) OUT：输出参数，用来从模块返回数据到进行调用的模块；
3) IN OUT：既作为输入参数，也作为输出参数。

在存储模块中使用参数时要注意下面几点：

1) IN 参数能被赋值；
2) OUT 参数的初值始终为空，无论调用该模块时对应的实参值为多少；
3) 调用一个模块时，OUT 参数及 IN OUT 参数的实参必须是可赋值的对象。

请看下面的例子：

```
CREATE OR REPLACE PROCEDURE PROC_ARG(A IN INT, B INT) AS
BEGIN
A:=0;
B:=A+1;
END;
/
```

使用赋值符号‘:= ’或关键字 DEFAULT，可以为 IN 参数指定一个缺省值。如果调用时未指定参数值，系统将自动使用该参数的缺省值。例如：

```
CREATE PROCEDURE PROC_def_ARG(A varchar(10) default 'abc', B INT:=123) AS
BEGIN
PRINT A;
PRINT B;
END;
/
```

调用过程，不指定输入参数值：

```
CALL PROC_DEF_ARG;
```

系统使用缺省值作为参数值，打印结果为：

```
ABC
123
```

也可以只有第一个参数，省略后面的参数：

```
CALL PROC_DEF_ARG('我们');
```

系统对后面的参数使用缺省值，打印结果为：

```
我们
123
```

注：对于缺省值，要么都使用缺省值，要么必须指定第一个参数值。

4．变量

变量的申明应在说明部分。使用赋值符号’:=‘或关键字 DEFAULT、ASSIGN，可以为变量指定一个缺省值。其语法为：

语法格式

```
<变量名>{,<变量名>}[CONSTANT]<变量类型>[NOT NULL][DEFAULT|ASSIGN|:=<表达式>]
```

举例说明

例 下例中说明了几种类型的变量：

```
CREATE OR REPLACE PROCEDURE PROC_VAR1 AS
V1    VARCHAR(20);
```

```
V2    INT:=25;
V3    DATE DEFAULT DATE '2000-01-01';
BEGIN
   NULL;
END;
/
```

第 2 行说明了一个 VARCHAR 类型的变量。第 3 行说明了一个 INT 类型的变量，并将它的缺省值置为 25。第 4 行说明了一个 DATE 类型的变量，并将它的缺省值置为'2000-01-01'。

变量的作用域与 PASCAL 语言类似。它只在定义它的语句块(包括其下层的语句块)内可见，并且定义在下一层语句块中的变量可以屏蔽上一层的同名变量。当遇到一个变量名时，系统首先在当前语句块内查找变量的定义；如果没有找到，再向包含该语句块的上一层语句块中查找，如此直到最外层。

下例中，存储过程 PROC_VAR2 中定义了一个整型变量 A，而在其下层的语句块中又定义了一个同名的字符型变量 A。由于在该语句块中，字符型变量 A 屏蔽了整型变量 A，所以第一条打印语句打印的是字符型变量 A 的值，而第二条打印语句打印的则是整型变量 A 的值。例如：

```
CREATE OR REPLACE PROCEDURE USER1.PROC_VAR2 AS
A INTEGER :=5;
BEGIN
DECLARE
A VARCHAR(10);              /*  此处定义的变量 A 与上一层中的变量 A 同名  */
BEGIN
A:= 'ABCDEFG';
PRINT A;                /*  第一条打印语句  */
END;
PRINT A;                /*  第二条打印语句  */
END;
/
```

执行此过程：

```
CALL USER1.PROC_VAR2;
```

打印结果应为：

```
ABCDEFG
5
```

5．使用 OR REPLACE 选项

读者可能已经发现，在前面的例子中，我们在创建存储模块的时候，都使用了 OR REPLACE 选项。使用 OR REPLACE 选项的好处是，如果系统中已经有同名的存储模块，服务器会自动用新的代码覆盖原来的代码。如果不使用 OR REPLACE 选项，当创建的存储模块与系统中已有的存储模块同名时，服务器会报错。

11.3 存储模块的删除

当用户需要从数据库中删除一个存储模块时，可以使用存储模块删除语句。其语法如下：

```
<存储模块删除语句> ::=
<存储过程删除语句> |
<存储函数删除语句>
```

1．存储过程删除语句

从数据库中删除一个存储过程。

语法格式

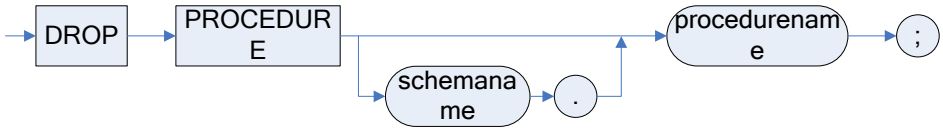
```
DROP PROCEDURE <存储过程名定义>;
<存储过程名定义> ::= [<模式名>.]<存储过程名>
```

参数

1) <模式名> 指明被删除的存储过程所属的模式，缺省为当前模式；

2) <存储过程名> 指明被删除的存储过程的名字。

图例



使用说明

如果被删除的存储过程不属于当前模式，必须在语句中指明过程的模式名。

权限

执行该操作的用户必须是该存储过程的拥有者，或者具有 DBA 权限。

2．存储函数删除语句

从数据库中删除一个存储函数。

语法格式

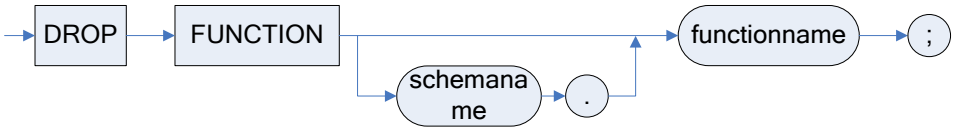
```
DROP FUNCTION <存储函数名定义>;
<存储函数名定义> ::= [<模式名>.]<存储函数名>
```

参数

1) <模式名> 指明被删除的存储函数所属的模式，缺省为当前模式；

2) <存储函数名> 指明被删除的存储函数的名字。

图例



使用说明

1) 如果被删除的存储函数不属于当前模式，必须在语句中指明函数的模式名；

2) 当模式名缺省时，默认为删除当前模式下的存储模块，否则，应指明存储模块所属的模式。除了 DBA 用户外，其他用户只能删除自己创建的存储模块。

权限

执行该操作的用户必须是该存储函数的拥有者，或者具有 DBA 权限。

举例说明

例 1 删除模式 USER1 下的存储过程 PROC_1。

```
DROP PROCEDURE USER1.PROC_1;
```

例 2 删除模式 USER1 下的存储函数 FUN_1，执行该语句的用户应为函数的创建者。

```
DROP FUNCTION USER1.FUN_1;
```

11.4 存储模块的重新编译

当用户需要调用存储模块时，可以先重新编译一下该存储模块，用来判断在当前情况下，存储模块是否可用。这是因为存储过程中可能用到一些表、索引等对象，但是这些对象已经被修改或者被删除，这就意味着存储过程（函数）可能已经失效了。

重新编译一个存储过程(函数)。

语法格式

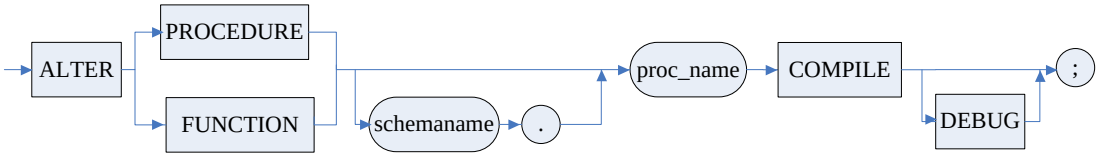
```
ALTER PROCEDURE|FUNCTION   <存储模块名定义> COMPILE [DEBUG];
<存储模块名定义> ::=[ <模式名>.]<模块名>
```

参数

1) <模式名> 指明要重新编译的存储模块所属的模式，缺省为当前模式；

2) <模块名> 指明要重新编译的存储模块的名字。

图例



使用说明

系统存储模块是不能进行重新编译的。

举例说明

例 重新编译存储过程 PRO_CALC。

```
ALTER PROCEDURE PRO_CALC COMPILE;
```

11.5 存储模块的控制语句

DMPL/SQL 语言对三种控制结构(分支结构、迭代结构和顺序结构)都提供了相应的语句支持。控制语句具体包括：

1. 语句块；
2. 赋值语句；
3. IF 语句；
4. 循环语句(包括 LOOP 语句、WHILE 语句、FOR 语句、REPEAT 语句)；
5. EXIT 语句；
6. 调用语句；
7. RETURN 语句；
8. NULL 语句；
9. GOTO 语句；
10. RAISE 语句；
11. 打印语句；
12. CASE 语句；
13. CONTINUE 语句；
14. REPEAT 语句。

以下详细介绍各种控制语句的语法及使用方法。

11.5.1 语句块

语句块是 DM PL/SQL 语言的基本程序单元。每个语句块由关键字 DECLARE、BEGIN、EXCEPTION 和 END 划分为声明部分、执行部分和异常处理部分。其中执行部分是必须的，说明和异常处理部分可以省略。用户可能已经发现，事实上整个存储模块就是一个语句块，只是其说明部分省略了 DECLARE 关键字而已。语句块可以嵌套，它可以出现在任何其他语句可以出现的位置。DM PL/SQL 块中的每一条语句都必须以分号结束，SQL 语句可以是

多行的，分号表示该语句的结束。一行中可以有多条 SQL 语句，他们之间以分号分隔。每一个 DM PL/SQL 块由 BEGIN 或 DECLARE 开始，以 END 结束。注释由--标示。

DM PL/SQL 语句块语法

```
[DECLARE <变量说明>{,<变量说明>};]
BEGIN
<执行部分>
[<异常处理部分>]
END
```

下面描述了 DM PL/SQL 块的不同部分：

声明部分(Declaration section)

声明部分包含了变量和常量的数据类型和初始值。这个部分由关键字 DECLARE 开始。如果不需要声明变量或常量，那么可以忽略这一部分；需要说明的是游标的声明也在这一部分。

变量声明

```
变量名 数据类型 [:=|DEFAULT expression]
```

声明一个变量需要给这个变量指定数据类型及名字，对于大多数据类型，都可以在定义的同时指定初始值。一个变量的名字一定要符合标识符的定义，在 DM 中标识符的定义规则与 C 语言相同。

对于要声明变量的数据类型，可以是基本的 SQL 数据类型，也可以是 PL/SQL 数据类型，比如一个游标、异常等。

常量声明

常量声明与变量声明基本相同，只有两点是不同的，在语法中需要用关键字 CONSTANT 来指定所声明的是常量，同时必须要给这个常量赋值。

不能修改常量的值，只能读取，不然会报错。

```
常量名 CONSTANT 数据类型:= DEFAULT expression;
ZERO_VALUE CONSTANT NUMBER:=0;
```

这个语句定了一个名叫 ZERO_VALUE，数据类型是 NUMBER，值为 0 的常量。

执行部分(Executable section)

执行部分是 DM PL/SQL 块中的指令部分，由关键字 BEGIN 开始，以关键字 EXCEPTION 结束，如果 EXCEPTION 不存在，那么将以关键字 END 结束。所有的可执行语句都放在这一部分，其他的 DM PL/SQL 块也可以放在这一部分。分号分隔每一条语句，使用赋值操作符:=或 SELECT INTO 或 FETCH INTO 给每个变量赋值，执行部分的错误将在异常处理部分解决，在执行部分中可以使用另一个 DM PL/SQL 程序块，这种程序块被称为嵌套块。

所有的 SQL 数据操作语句都可以用于执行部分，PL/SQL 块不能在屏幕上显示 SELECT 语句的输出。SELECT 语句必须包括一个 INTO 子串或者是游标的一部分，执行部分使用的变量和常量必须首先在声明部分声明，执行部分必须至少包括一条可执行语句，NULL 是一条合法的可执行语句，事务控制语句 COMMIT 和 ROLLBACK 可以在执行部分使用。数据定义语言(Data Definition Language)不能在执行部分中使用，DDL 语句与 EXECUTE IMMEDIATE 一起使用。

异常处理部分(Exception section)

这一部分是可选的，在这一部分中处理异常或错误，对异常处理的详细讨论我们在后面进行。

需要强调的一点是，一个语句块意味着一个作用域范围。也就是说，在一个语句块的说明部分定义的任何对象，其作用域就是该语句块。请看下面的例子，该例中有一个全局变量 X，同时子语句块中又定义了一个局部变量 X。

```
CREATE OR REPLACE PROCEDURE PROC_BLOCK AS
    X INT := 0;
    COUNTER INT := 0;
```

```
BEGIN
    FOR I IN 1 .. 4 LOOP
        X := X + 1000;
        COUNTER := COUNTER + 1;
        PRINT CAST(X AS CHAR(10)) || CAST(COUNTER AS CHAR(10)) || 'OUTER LOOP';
      /* 这里是一个嵌套的子语句块的开始 */
      DECLARE
      X INT := 0;   -- 局部变量 X，与全局变量 X 同名
      BEGIN
        FOR I IN 1 .. 4 LOOP
          X := X + 1;
          COUNTER := COUNTER + 1;
          PRINT CAST(X AS CHAR(10)) || CAST(COUNTER AS CHAR(10)) || 'INNER LOOP';
        END LOOP;
      END;
    /* 子语句块结束 */
    END LOOP;
END;
/
```

执行语句：

```
CALL PROC_BLOCK;
```

执行结果为：

```
1000        1           OUTER LOOP
1           2           INNER LOOP
2           3           INNER LOOP
3           4           INNER LOOP
4           5           INNER LOOP
2000        6           OUTER LOOP
1           7           INNER LOOP
2           8           INNER LOOP
3           9           INNER LOOP
4           10          INNER LOOP
3000        11          OUTER LOOP
1           12          INNER LOOP
2           13          INNER LOOP
3           14          INNER LOOP
4           15          INNER LOOP
4000        16          OUTER LOOP
1           17          INNER LOOP
2           18          INNER LOOP
3           19          INNER LOOP
4           20          INNER LOOP
```

由执行结果可以看出，两个 X 的作用域是完全不同的。

11.5.2 赋值语句

赋值语句的语法如下：

```
<赋值对象> := <值表达式>
或
SET <赋值对象> = <值表达式>
```

使用赋值语句可以给各种数据类型的对象赋值。被赋值的对象可以是变量，也可以是 OUT 参数或 IN OUT 参数。表达式的数据类型必须与赋值对象的数据类型兼容。

请看下面的例子：

```
CREATE OR REPLACE PROCEDURE PROC_ASSIGN AS
        V1      INT;
        V2      FLOAT;
        V3      NUMERIC;
        V4      VARCHAR(20);
        V5      DATE;
        V6      TIME;
```

```
        V7          TIMESTAMP;
        V8          INTERVAL YEAR TO MONTH;
BEGIN
        V1:=15;
        V2:=2.3456;
        V3:=V1+V2;
        V4:= '0123456789' || CAST(V1 AS VARCHAR(10));
        V5:= DATE '1998-06-12';
        V6:=TIME '08:09:20.12345678';
        V7:=TIMESTAMP '2000-08-14 11:12:23.45678901';
        V8:= INTERVAL '0022-11' YEAR TO MONTH;
 END;
```

11.5.3 条件语句

存储模块中的 IF 语句控制执行基于布尔条件的语句序列，以实现条件分支控制结构。其语法如下：

```
IF <条件表达式>
THEN <执行部分>;
[ELSEIF|ELSIF <条件表达式> THEN <执行部分>;{ELSEIF|ELSIF <条件表达式> THEN <执行部分>;}]
[ELSE <执行部分>;]
END IF
```

考虑到不同用户的编程习惯，ELSEIF 子句的起始关键字既可写作 ELSEIF，也可写作 ELSIF。

条件表达式中的因子可以是布尔类型的参数、变量，也可以是条件谓词。存储模块的控制语句中支持的条件谓词有：比较谓词、BETWEEN、IN、LIKE 和 IS NULL。以下就以 IF 语句为例，列举条件表达式里谓词的用法。

例 1 含 BETWEEN 谓词的条件表达式：

```
CREATE OR REPLACE PROCEDURE P_CONDITION1 (A INT) AS
BEGIN
   IF A BETWEEN -5 AND 5 THEN
     PRINT 'TRUE';
   ELSE
     PRINT 'FALSE';
   END IF;
END;
/
```

例 2 含 IN 谓词的条件表达式：

```
CREATE OR REPLACE PROCEDURE P_CONDITION2 (A INT) AS
BEGIN
   IF A IN (1,3,5,7,9) THEN
     PRINT 'TRUE';
   ELSE
     PRINT 'FALSE';
   END IF;
END;
/
```

例 3 含 LIKE 谓词的条件表达式：

```
CREATE OR REPLACE PROCEDURE P_CONDITION3(A VARCHAR(10)) AS
BEGIN
   IF A LIKE '%DM%' THEN
     PRINT 'TRUE';
   ELSE
     PRINT 'FALSE';
   END IF;
END;
/
```

例 4 含 IS NULL 谓词的条件表达式：

```
CREATE OR REPLACE PROCEDURE P_CONDITION4 (A INT) AS
BEGIN
   IF A IS NOT NULL THEN
     PRINT 'TRUE';
   ELSE
     PRINT 'FALSE';
   END IF;
END;
/
```

11.5.4 循环语句

存储模块支持四种基本类型的循环语句，即 LOOP 语句、WHILE 语句、FOR 语句和 REPEAT 语句。LOOP 语句循环重复执行一系列语句，直到 EXIT 语句终止循环为止；WHILE 语句循环检测一个条件表达式，当表达式的值为 TRUE 时就执行循环体的语句；FOR 语句对一系列的语句重复执行指定次数的循环；REPEAT 语句重复执行一系列语句直至达到条件表达式的限制要求。以下对这四种语句分别进行介绍。

1．LOOP 语句

LOOP 语句的语法如下：

```
LOOP
<执行部分>;
END LOOP
```

LOOP 语句实现对一语句系列的重复执行，是循环语句的最简单形式。它没有明显的终点，必须借助 EXIT 语句来跳出循环。以下是 LOOP 语句的用法举例：

```
CREATE OR REPLACE PROCEDURE P_LOOP(A IN OUT INT) AS
BEGIN
   LOOP
      IF A<=0 THEN
         EXIT;
END IF;
PRINT A;
A:=A-1;
END LOOP;
END;
/
```

第 3 行到第 9 行是一个 LOOP 循环，每一次循环都打印参数 A 的值，并将 A 的值减 1，直到 A 小于等于 0。

2．WHILE 语句

WHILE 语句的语法如下：

```
WHILE <条件表达式> LOOP
<执行部分>;
END LOOP
```

WHILE 循环语句在每次循环开始以前，先计算条件表达式，若该条件为 TRUE，语句序列被执行一次，然后控制重新回到循环顶部。若条件表达式的值为 FALSE，则结束循环。当然，也可以通过 EXIT 语句来终止循环。以下是 WHILE 语句的用法举例：

```
CREATE OR REPLACE PROCEDURE P_WHILE(A IN OUT INT) AS
BEGIN
WHILE A>0 LOOP
PRINT A;
A:=A-1;
END LOOP;
END;
/
```

这个例子的功能与上例相同，只是使用了 WHILE 循环结构。

3．FOR 语句

FOR 语句的语法如下：

```
FOR <循环计数器> IN [REVERSE] <下限表达式> .. <上限表达式> LOOP
<执行部分>;
END LOOP
```

循环计数器是一个标识符，它类似于一个变量，但是不能被赋值，且作用域限于 FOR 语句内部。下限表达式和上限表达式用来确定循环的范围，它们的类型必须和整型兼容。循环范围是在循环开始之前确定的，即使在循环过程中下限表达式或上限表达式的值发生了改变，也不会引起循环范围的变化。

执行 FOR 语句时，首先检查下限表达式的值是否小于上限表达式的值，如果下限数值大于上限数值，则不执行循环体。否则，将下限数值赋给循环计数器(语句中使用了 REVERSE 关键字时，则把上限数值赋给循环计数器)；然后执行循环体内的语句序列；执行完后，循环计数器值加 1(如果有 REVERSE 关键字，则减 1)；检查循环计数器的值，若仍在循环范围内，则重新继续执行循环体；如此循环，直到循环计数器的值超出循环范围。同样，也可以通过 EXIT 语句来终止循环。以下是 FOR 语句的用法举例：

```
CREATE OR REPLACE PROCEDURE P_FOR1 (A IN OUT INT) AS
BEGIN
FOR I IN REVERSE 1 .. A LOOP
PRINT I;
A:=I-1;
END LOOP;
END;
```

这个例子的功能也与上例相同，只是使用了 FOR 循环结构。

FOR 语句中的循环计数器可与当前语句块内的参数或变量同名，这时该同名的参数或变量在 FOR 语句的范围内将被屏蔽。例子如下：

```
CREATE OR REPLACE PROCEDURE P_FOR2 AS
V1 DATE:=DATE '2000-01-01';
BEGIN
FOR V1 IN 0 .. 5 LOOP
PRINT V1;
END LOOP;
PRINT V1;
END;
/
```

调用此过程：

```
CALL   P_FOR2;
```

打印结果为：

```
0
1
2
3
4
5
2000-01-01
```

此例中，循环计数器 V1 与变量 V1 同名。由调用结果可见，在 FOR 语句内，PRINT 语句将 V1 当做循环计数器。而 FOR 语句外的 PRINT 语句则将 V1 当作 DATE 类型的变量。请注意：在符号’..’和上/下限表达式间应用空格隔开，否则容易引起编译器的误解。

4．REPEAT 语句

REPEAT 语句的语法如下：

```
REPEAT
<执行部分>;
```

```
UNTIL <条件表达式>
```

以下是 REPEAT 语句的用法举例：

```
CREATE OR REPLACE PROCEDURE PROCEDURE_REPEAT1 AS
A INT;
BEGIN
A := 0;
REPEAT
A := A+1;
UNTIL A>10;
END;
/
```

11.5.5 EXIT 语句

EXIT 语句与循环语句一起使用，用于终止其所在循环语句的执行，将控制转移到该循环语句外的下一个语句继续执行。其语法如下：

```
<EXIT 语句> ::= EXIT [<标号名>] [WHEN <条件表达式>]
```

EXIT 语句必须出现在一个循环语句中，否则编译器将报错。

当 EXIT 后面的标号名省略时，该语句将终止直接包含它的那条循环语句；当 EXIT 后面带有标号名时，该语句用于终止标号名所标识的那条循环语句。需要注意的是，该标号名所标识的语句必须是循环语句，并且 EXIT 语句必须出现在此循环语句中。当 EXIT 语句位于多重循环中时，可以用该功能来终止其中的任何一重循环。

当 WHEN 子句省略时，EXIT 语句无条件的终止该循环语句；否则，先计算 WHEN 子句中的条件表达式，当条件表达式的值为真时，终止该循环语句。

前面我们已经见到了 EXIT 语句的用法，该例子还可以这样实现：

```
CREATE OR REPLACE PROCEDURE P_LOOP(A IN OUT INT) AS
BEGIN
LOOP
EXIT WHEN A<=0;
PRINT A;
A:=A-1;
END LOOP;
END;
```

下例说明了 EXIT 后带标号名的用法：

```
CREATE OR REPLACE PROCEDURE P_LOOP1 AS
a int;
b int;
begin
a := 0;
<<flag1>>
loop
for b in 0 .. 3 loop
exit flag1 when a > 3;
print a;
end loop;
a := a + 1;
end loop;
end;
```

11.5.6 调用语句

存储模块可以被其他存储模块或应用程序调用。同样，在存储模块中也可以调用其他存储模块。调用存储过程时，应给存储过程提供输入参数值，并获取存储过程的输出参数值。调用的形式为：

语法格式

```
[CALL] [<模式名>.]<存储过程名> [(<参数>{, <参数>})];
<参数> ::= <参数值>|<参数名=参数值>
```

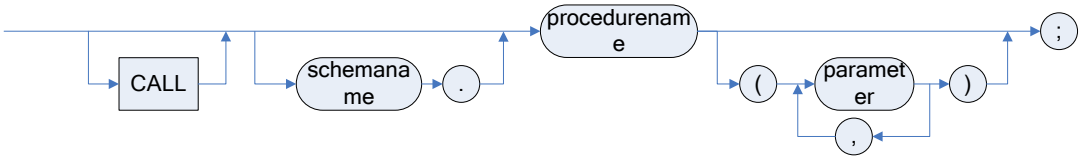
参数

1．<模式名>　　指明被调用存储过程所属的模式；

2．<存储过程名>　指明被调用存储过程的名字；

3．<参数>　　指明提供给存储过程的参数。

图例



使用说明

1．如果被调用的存储过程不属于当前模式，必须在语句中指明存储过程的模式名；

2．参数的个数和类型必须与被调用的存储过程一致；

3．存储过程的输入参数可以是嵌入式变量，也可以是值表达式；存储过程的输出参数必须是可赋值对象，如嵌入式变量；

4．在调用过程中，服务器将以存储过程创建者的模式和权限来执行过程中的语句；

5．一般情况下，用户调用过程时通过位置关系和形式参数相对应，这种方式被称为“位置标识法”。系统还支持一种过程调用方式：每个参数中包含形式参数和实际参数，这样的方法被称为“带名标识法”。对结果而言，这两种调用方式是等价的。

下表中对比了两种方式的差异：

表 11.5.1 位置标志法和带名标识法比较表

位置标志法	带名标识法
依赖于实际参数的名字定义	清楚地说明了实际参数和形式参数间的关系
形式参数地名字相对于实际参数而言是独立地	形式参数和实际参数所使用的次序是相互独立的
调用比较简洁，符合大多数第三代语言的使用习惯	需要更多的编码，调用时要需要指定形式参数和实际参数
使用参数缺省值时必须在参数列表的末尾	无论哪个参数拥有缺省值都不会对调用产生影响
维护的代价在于参数的定义次序发生了变化	维护的代价在于参数名的定义发生了变化

权限

执行该操作的用户必须拥有该存储过程的 EXECUTE 权限。存储过程的所有者和 DBA 用户隐式具有该过程的 EXECUTE 权限，该权限也可通过授权语句显式授予其他用户。

如果是过程调用，调用语句必须是一条单独的控制语句；如果是函数调用，则必须出现在一个表达式当中。

例 1 存储过程的调用

```
--以用户 SYSDBA 的身份创建存储过程 P1：
CREATE OR REPLACE PROCEDURE P1(A IN OUT INT) AS
V1 INT:=A;
BEGIN
A:=0;
FOR B IN 1 .. V1 LOOP
A:=A+B;
END LOOP;
END;
--在存储过程 P2 中调用存储过程 P1：
```

```
CREATE OR REPLACE PROCEDURE P2(A IN INT) AS
V1 INT :=A;
BEGIN
sysdba.P1(V1);
PRINT V1;
END;
```

存储过程的按名参数调用

```
--以用户 SYSDBA 的身份创建存储过程 P1：
CREATE OR REPLACE PROCEDURE P1(A INT, B IN OUT INT) AS
V1 INT:=A;
BEGIN
B:=0;
FOR C IN 1 .. V1 LOOP
B:=B+C;
END LOOP;
END;

--在存储过程 P2 中以按名方式调用过程 P1：
CREATE OR REPLACE PROCEDURE P2(A IN INT) AS
V1 INT :=A;
V2 INT;
BEGIN
sysdba.P1(B=V2, A=V1);
PRINT V2;
END;
```

例 2 存储函数的调用

创建存储函数 MAXNUM：

```
CREATE OR REPLACE FUNCTION MAXNUM (A INT, B INT) RETURN INT AS
BEGIN
IF (A >= B) THEN
RETURN A;
ELSE
RETURN B;
END IF;
END;
```

假设存在表 T_NUM(COL INT)，在存储过程 PROC_MAXNUM 中调用存储函数 MAXNUM：

```
CREATE OR REPLACE PROCEDURE PROC_MAXNUM(A INT, B INT) AS
BEGIN
INSERT INTO T_NUM VALUES(MAXNUM(A,B));
PRINT MAXNUM(A,B);
END;
```

11.5.7 RETURN 语句

RETURN 语句的语法如下：

```
RETURN [<返回值>];
```

其功能为：结束模块的执行，将控制返回给该模块调用者。如果是从函数返回，则同时将函数的返回值提供给调用环境。

除管道表函数外，函数的执行必须以 RETURN 语句结束。确切的说，函数中应至少有一个被执行的返回语句。由于程序中有分支结构，在编译时很难保证其每条路径都有返回语句，因此 DM 在函数执行时才对其进行检查，如果某个分支缺少 RETURN，DM 会自动为其隐式增加一条“RETURN NULL”语句。

管道表函数的返回值是特定且隐含的，函数体中可以省略 RETURN 语句，即使显式的 RETURN 语句，也不能在后标注返回对象。

例

```
CREATE OR REPLACE FUNCTION FUN1(A INT) RETURN VARCHAR(10) AS
BEGIN
IF (A<0) THEN
RETURN ' A<0';
ELSE
RETURN NULL;
END IF;
END;
```

此例中，当参数 A 大于或等于 0 时，函数 FUN1 没有以 RETURN 语句结束。因此，执行下面的语句：

```
SELECT FUN1(1);
```

返回 NULL。

11.5.8 NULL 语句

NULL 语句的语法为：

```
NULL
```

该语句不做任何事情，只是用于保证语法的正确性，或增加程序的可读性。

11.5.9 GOTO 语句

GOTO 语句无条件地跳转到一个标号所在的位置。标号的定义在一个语句块中必须是唯一的。其语法如下：

```
GOTO <标号名>
```

GOTO 语句将控制权交给带有标号的语句或语句块。为了保证 GOTO 语句的使用不会引起程序的混乱，我们对 GOTO 语句的使用有下列限制：

1. GOTO 语句不能跳入一个 IF 语句、循环语句或下层语句块中；
2. GOTO 语句不能从一个异常处理器跳回当前块，但是可以跳转到包含当前块的上层语句块。

下面是一些非法的 GOTO 语句的例子。

例 1

```
BEGIN
    ...
    GOTO update_row;            /* 错误，企图跳入一个 IF 语句 */
    ...
    IF valid THEN
        ...
        <<update_row>>
        UPDATE emp SET ...
    END IF;
END;
```

例 2

```
BEGIN
    ...
    IF valid THEN
        ...
        GOTO update_row;   /* 错误，企图从 IF 语句的一个子句跳入另一个子句 */
    ELSE
        ...
        <<update_row>>
        UPDATE emp SET ...
    END IF;
END;
```

例 3

```
BEGIN
    ...
    IF status = 'OBSOLETE' THEN
        GOTO delete_part;   /* 错误，企图跳入一个下层语句块 */
    END IF;
    ...
    BEGIN
        ...
        <<delete_part>>
        DELETE FROM parts WHERE ...
    END;
END;
```

11.5.10 RAISE 语句

语法：

```
RAISE <异常名>
```

RAISE 语句用于抛出异常。要查看详细的说明，请阅读第 11.7 节，存储模块的异常处理。

11.5.11 打印语句

打印语句用于从存储模块中向客户端输出一个字符串，方便用户调试存储模块代码。当存储模块的行为与预期的不一致时，可以在其中放进打印语句来观察各个阶段的运行情况。打印语句的语法为：

```
PRINT <表达式>
```

打印语句中的表达式可以是各种数据类型，系统自动将其转换为字符类型。前面的例子中有多处用到了打印语句，用户可自行参考，这里不再另外举例。

11.5.12 CASE 语句

CASE 语句是从一个序列条件中进行选择的，并且执行相应的语句块，主要有下面两种形式：

简单形式：将一个表达式与多个值进行比较。

语法：

```
CASE <条件表达式>
WHEN <条件 1> THEN <语句 1>
WHEN <条件 2> THEN <语句 2>
WHEN <条件 n> THEN <语句 n>
[ ELSE <语句> ]
END CASE;
```

其中每个条件可以是立即值，也可以是一个表达式，这种形式的 CASE 会选择第一个满足条件的对应的语句来执行，剩下的则不会计算，如果没有符合的条件，它会执行 ELSE 语句块中的语句，但是如果 ELSE 语句块不存在，则不会执行任何语句。

```
CREATE OR REPLACE PROCEDURE test AS
BEGIN
      CASE (1 + 1)
      WHEN 2 THEN PRINT 2;
```

```
        WHEN 3 THEN PRINT 3;
        WHEN 4 THEN PRINT 4;
        ELSE PRINT 5;
        END CASE;
END;
```

搜索形式：对多个条件进行计算，取第一个结果为真的条件。

语法：

```
CASE
WHEN <条件表达式> THEN <语句 1>
WHEN <条件表达式> THEN <语句 2>
WHEN <条件表达式> THEN <语句 n>
[ ELSE <语句> ]
END CASE;
```

从上面语法就可以看出，搜索模式的 CASE 语句执行的是第一个为真的条件，在第一个为真的条件后面的所有条件都不会再执行，如果所有的条件都不为真，则执行 ELSE 语句，如果 ELSE 不存在，则不执行任何语句。下面的例子就是搜索模式的：

```
CREATE OR REPLACE PROCEDURE test AS
BEGIN
        CASE
        WHEN 2=1 THEN PRINT 2;
        WHEN 3=2 THEN PRINT 3;
        WHEN 4=4 THEN PRINT 4;
        END CASE;
END;
```

CASE 语法有点类似 C 语言中的 SWITCH 语句，它的执行体可以被一个 WHEN 条件包含起来，和 IF 语句相似。一个 CASE 语句是由 END CASE 来结束的。

11.5.13 CONTINUE 语句

CONTINUE 语句的作用是无条件的退出当前循环迭代，并且将语句控制转移到这次循环的下一次迭代或者是一个指定标签的循环的开始位置并继续执行。

CONTINUE WHEN 语句的作用是当 WHEN 后面的条件满足时才将语句控制转移到这次循环的下一次迭代或者是一个指定标签的循环的开始位置并继续执行。当每次循环到达 EXIT WHEN 时，都会对 WHEN 的条件进行计算，如果条件为 FALSE，则 EXIT WHEN 对应的语句不会被执行，为了防止出现死循环，可以将 WHEN 条件设置为一个肯定可以为 TRUE 的表达式。

语法：

```
CONTINUE [[标签] WHEN <条件表达式>];
```

11.5.14 PIPE ROW 语句

PIPE ROW 语句只能在管道表函数中使用，其作用是将返回结果元组输入到管道函数的结果集中。

语法：

```
PIPE ROW ( 值表达式 )
```

11.6 存储模块的调用

存储模块的调用可通过语句 CALL 来完成，也可以通过 SELECT 语句来调用，还可以什么也不加直接通过名字及相应的参数执行即可，但这三种方式的执行是有一些差别的。

如果通过 CALL 来执行一个存储模块，那么它不会返回任何的结果集信息，也就是说即使这个存储模块是有结果集的，它也不会返回，它只负责执行，这就决定了 CALL 语句的运用场景，只有当存储模块只是对数据进行操作，不会返回信息的情况下用 CALL 才是适合的，如果有结果集还用 CALL 则不会看到任何结果集。

如果不通过任何语句来直接执行存储模块，结果和上面 CALL 是完全一样的，也不会返回任何信息，只会执行其中操作。

如果通过 SELECT 语句来调用存储模块可返回结果集，如果调用的过程函数有结果集则会有结果输出，如果没有则和上面 CALL 调用的结果是一样的。

```
CREATE OR REPLACE FUNCTION proc(A INT) RETURN int AS
DECLARE
S int;
lv int;
rv int;
BEGIN
    IF A = 1 THEN
       S = 1;
    ELSIF A = 2 THEN
       S = 1;
    ELSE
       rv = proc(A - 1);
       lv = proc(A - 2);
       S = lv + rv;
       print lv || '+' || rv || '=' || S;
    END IF;
    RETURN S;
END;
/
```

可以通过 CALL 来调用函数 PROC，语句为：call proc(10);这样就会打印结果，但这些结果不是它的结果集，而是程序中指定的打印操作。而如果用 select proc(10)来调用这个过程的话就会输出值 55，因为这个值是这个函数的返回值，所以就会输出它的返回值，这个相当于是结果集。

11.7 存储模块的异常处理

在存储模块的正常执行过程中，可能会出现未预料的事件，这就是异常(EXCEPTION)。异常事件会导致代码的不正确结束，因此用户需要在程序中对异常进行处理，以保证不可预知的错误不会终止 DM PL/SQL 模块。为此，DM 提供了异常处理机制，使用户可以在语句块的异常处理部分中处理异常。这里所说的异常，可以是系统预定义异常，也可以是用户自定义异常。

为方便用户编程，DM 系统提供了一些预定义的异常，这些异常与最常见的 DM 错误相对应。下表中列举了 DM 系统的预定义异常。

表 11.7.1 预定义异常表

异常名称	SQLCODE	说明
DUP_AVL_ON_INDEX	-6602	违反唯一性约束
INVALID_CURSOR	-4535	无效的游标操作

TOO_MANY_ROWS	-7046	SELECT INTO 中包含多行数据
ZERO_DIVIDE	-6103	除 0 错误
NO_DATA_FOUND	-7065	数据未找到

此外，还有一个特殊的异常名 OTHERS，它处理所有没有明确列出的异常。OTHERS 对应的异常处理语句必须放在其它异常处理语句之后。

11.7.1 异常变量的说明

如果用户需要处理非预定义的异常，可以自己定义一个异常。创建自定义异常的方法是：在程序块的说明部分定义一个异常变量，并将该异常变量与用户要处理的 DM 错误号绑定。DM7 支持两种自定义异常变量的方法。

1. 使用 EXCEPTION FOR 将异常变量与错误号绑定

语法如下：

```
<异常变量名> EXCEPTION [FOR <错误号> [, <错误描述>]]
```

其中，FOR 子句用来为异常变量绑定错误号(SQLCODE 值)及错误描述串。<错误号>必须是-20000 到-30000 间的负数值，<错误描述>则为字符串类型。如果未显式指定错误号，则系统在运行中在-10001 到-15000 区间中顺序为其绑定错误值。

2. 使用 EXCEPTION_INIT 将一个特定的错误号与程序中所声明的异常标示符关联起来

语法如下：

```
<异常变量名> EXCEPTION;
PRAGMA EXCEPTION_INIT(<异常变量名>, <错误号>);
```

EXCEPTION_INIT 将异常名与 DM 错误码结合起来，这样可以通过名字引用任意的内部异常，并且可以通过名字为异常编写一适当的异常处理。如果希望使用 RAISE 语句抛出一个用户自定义异常，则与异常关联的错误号必须是-20000 到-30000 之间的负数值。

异常变量类似于一般的变量，必须在块的说明部分说明，有同样的生存期和作用域。但是异常变量不能作参数传递，也不能被赋值。

需要注意的是，为异常变量绑定的错误号不一定是 DM 返回的系统错误，但是该错误号必须是一个负整数。自定义异常使得用户可以把违背事务规则的行为也作为异常来看待。

11.7.2 异常的抛出

在存储模块的执行中如果发生错误，系统将自动抛出一个异常。此外，我们还可以用 RAISE 语句自己抛出异常，例如，当操作合法，但是违背了事务规则时。一旦异常被抛出，执行就被传递给程序块的异常处理部分。RAISE 语句的语法如下：

```
RAISE <异常名>
```

其中，<异常名>可以是系统预定义异常，也可以是用户自定义异常。

11.7.3 异常处理器

程序块的异常处理部分可以处理一个或者多个异常。其语法如下：

```
<异常处理部分> ::= EXCEPTION  {<异常处理语句>;}
<异常处理语句> ::= WHEN <异常名>  THEN <执行部分>;
```

EXCEPTION 关键字表示异常处理部分的开始，异常处理器即异常处理语句中 WHEN 子句后面的部分。如果在语句块的执行中出现异常，执行就被传递给语句块的异常处理部分。

而如果在本语句块的异常处理部分没有相应的异常处理器对它进行处理，系统就会中止此语句块的执行，并将此异常传递到该语句块的上一层语句块或其调用者，这样一直到最外层。如果始终没有找到相应的异常处理器，则中止本次调用语句的处理，并向用户报告错误。

异常处理部分是可选的。但是如果出现 EXCEPTION 关键字时，必须至少有一个异常处理器。异常处理器可以按任意次序排列，只有 OTHERS 异常处理器必须在最后，它处理所有没有明确列出的异常。此外，同一个语句块内的异常处理器不允许处理重复的异常。

11.7.4 内置函数 SQLCODE 和 SQLERRM

内置函数 SQLCODE 和 SQLERRM 用在异常处理部分，分别用来返回错误代码和错误信息。SQLCODE 返回的是负数，对于 SQLERRM，如果找到对应系统错误码描述，则返回相应描述，否则：

1. 如果 ERROR_NUMBER 在-15000 至-19999 间，返回'User-Defined Exception';
2. 如果在-20000 至-30000 之间，返回'DM-<ERROR_NUMBER 绝对值>';
3. 如果大于 0 或小于-65535，返回'-<ERROR_NUMBER 绝对值>: non-DM exception';
4. 否则，返回'DM-<ERROR_NUMBER 绝对值>: Message <ERROR_NUMBER 绝对值> not found;'。

11.7.5 异常处理用法举例

下面的过程中，由于出现了除零错误，系统将抛出异常 ZERO_DIVIDE。在异常处理部分，我们定义了一个异常处理器来处理该异常。例子如下：

```
CREATE OR REPLACE PROCEDURE SYSEXCEPT1 AS
A INTEGER:=1;
BEGIN
A:=A/0;    /* 除零错误 */
EXCEPTION
     WHEN ZERO_DIVIDE THEN
          PRINT   TO_CHAR(SQLCODE) + SQLERRM;
END;
```

执行此过程：

```
CALL SYSEXCEPT1;
```

服务器返回打印信息：-6103 除 0 错误

下层语句块中出现的异常，如果在该语句块中没有对应的异常处理器，可以被上层语句块或其调用者的异常处理器处理。如下例中，一个下层语句块中抛出了异常 E1，但是没有处理该异常。那么该异常将由它的上层语句块中的异常处理器来处理：

```
CREATE OR REPLACE PROCEDURE EXCEPT1 AS
E1 EXCEPTION;
BEGIN
PRINT 'BEFORE SUB BLOCK';
/* 下层语句块开始 */
BEGIN
     RAISE E1;              /* 抛出异常 E1 */
END;
/* 下层语句块结束 */
PRINT 'AFTER SUB BLOCK ';
EXCEPTION
     WHEN E1 THEN
       PRINT 'EXCEPTION E1 RAISED FROM SUB BLOCK CATCHED';
END;
```

执行此过程：CALL EXCEPT1;

服务器返回打印信息：

```
BEFORE SUB BLOCK
EXCEPTION E1 RAISED FROM SUB BLOCK CATCHED
```

要注意异常变量的有效范围。在上例中，异常变量 E1 的有效范围是整个存储过程 EXCEPT1。如果改为：

```
CREATE OR REPLACE PROCEDURE EXCEPT2 AS
BEGIN
PRINT 'BEFORE SUB BLOCK ';
/* 下层语句块开始 */
DECLARE
E1 EXCEPTION;              /* 异常变量定义在下层语句块中 */
BEGIN
    RAISE E1;
END;
/* 下层语句块结束 */
PRINT 'AFTER SUB BLOCK ';
EXCEPTION
    WHEN E1 THEN              /* 错误：异常变量 E1 在此范围内没有定义 */
    PRINT 'EXCEPTION E1 RAISED FROM SUB BLOCK CATCHED';
END;
```

创建该过程时，编译器将报错：错误的对象名前缀。

有时候，在异常处理器的执行中又可能出现新的异常。这时，系统将新出现的异常作为当前需要处理的异常，并向上层传递。如下例：

```
CREATE OR REPLACE PROCEDURE EXCEPT3 AS
E1 EXCEPTION for -20000, 'EC_01';
E2 EXCEPTION for -30000, 'EC_02';
BEGIN
  BEGIN
    RAISE E1; /* 下层语句块抛出异常 E1 */
  EXCEPTION
    WHEN E1 THEN
      RAISE E2; /* 在异常 E1 的处理过程中，抛出异常 E2 */
  END;
  EXCEPTION
    WHEN E1 THEN /* 异常变量 E1 的异常处理器 */
      PRINT 'EXCEPTION E1 CATCHED';
    WHEN E2 THEN /* 异常变量 E2 的异常处理器 */
      PRINT 'EXCEPTION E2 CATCHED';
END;
```

执行此过程：

```
CALL EXCEPT3;
```

服务器将返回打印信息：

```
EXCEPTION E2 CATCHED
```

因为下层语句块在处理异常 E1 的过程中，又抛出了异常 E2。那么，E2 将作为当前需要处理的异常被传递到其上层语句块中。因此，在上层语句块中，E2(而不是 E1)的异常处理器将被执行。

11.8 存储模块的 SQL 语句

存储模块中支持的 SQL 语句包括：

1. 数据查询语句(SELECT)；
2. 数据操纵语句(INSERT、DELETE、UPDATE)；
3. 游标定义及操纵语句(DECLARE CURSOR、OPEN、FETCH、CLOSE)；
4. 事务控制语句(COMMIT、ROLLBACK、SAVEPOINT)；

5. 动态 SQL 执行语句(EXECUTE IMMEDIATE)。

SQL 语句中可以包含变量、参数和存储函数。DM 系统允许变量或参数与 SQL 语句中的表、视图及列同名，但是为了避免混淆及降低程序的可读性，建议用户尽量避免这种用法。

11.8.1 游标

存储模块中可以定义和操纵游标。一个游标指向的是一条 DML 语句或者是一个查询操作运行后的一个结果集区域，在用它的时候必须先创建方可对它进行操纵，在声明一个游标类型的变量之后，必须要将它与一个 SQL 语句进行关联，然后才可以对它的结果集进行操作，主要有两种方式：

1．打开游标，从取到的数据行上取出想要的数据，然后再显示的关闭游标；

2．在一个循环语句中可以对其结果集中的所有记录进行操纵。

对于一个显式游标，是不可以对它进行赋值，不可以在一个表达式中使用，也不可以将它作为过程/函数的参数传到函数内部的，如果想这样做，那么就要用“引用游标”。

因为这个游标是被显式声明的，它是一个游标类型的变量，所以对它引用的时候可以通过其名字来操作。所以有时候也被称作是“命名游标”。

1．游标的定义

游标定义必须出现在块的定义部分。它与变量处于同一个的命名空间，有着相同的作用域。但是游标不能作参数传递，也不能被赋值。游标的定义可以有两种选择，可以在声明的时候就指定其关联的 SQL 语句，然后在打开的时候这个游标就生效，还可以先只是声明一个游标类型的变量，然后再在打开的时候指定其关联的 SQL 语句同时游标生效。

```
CURSOR cursor_name [IS|FOR select clause|Join cluse]
```

cursor_name 后面的查询语句是可选的，也就是说可以先只声明一个游标而不关联 SQL 语句。

2．打开游标语句

语法格式

```
OPEN <游标变量名> FOR <不带 INTO 查询表达式>
```

或

```
OPEN <游标变量名> FOR <表达式> [USING <绑定参数> {,<绑定参数>}]
```

参数

<游标名> 指明被打开的游标的名称。

使用说明

1．打开游标必须是打开已定义过的游标，如果这个游标在声明时候已经指定了要关联的 SQL 语句，则在打开时不能再使用上面 FOR 语句后面的 SQL 再进行关联，否则会报错，而如果在声明时没有指定其关联的 SQL 语句，则在这里必须要指定相关语句才能打开游标；

2．该语句的执行将根据定义游标语句中的查询说明所指出的条件，从表中取出符合条件的行进入游标工作区，然后将游标置于第一行之前的位置。

使游标移动到指定的一行，若游标名后跟随有 INTO 子句，则将游标当前指示行的内容取出分别送入 INTO 后的各变量中。

打开游标时，系统会主要做以下几项工作：

1．将引用游标关联一个查询；

2．为这个查询分配数据库资源；

3．处理查询，得到结果集；

4．将游标定位到合适的位置。

在使用完游标之后，可以使用 CLOSE CURSOR_NAME 语句来关闭一个已经打开的游标，从而把其所占的资源还给系统。在关闭一个游标之后还可以再次打开，当打开一个游标时，这个游标所有的状态都被初始化，所以当游标没有被关闭就再次打开时，此时的游标属性数据都有可能会发生改变。

```
CREATE OR REPLACE PROCEDURE PROCA AS
DECLARE CURSOR csr is SELECT LOGINID FROM RESOURCES.EMPLOYEE WHERE TITLE='销售经理';
str    varchar(50);
BEGIN
    OPEN csr;
    IF csr%ISOPEN THEN
        PRINT 'CURSOR IS ALREADY OPENED';
        FETCH csr INTO str;
        PRINT csr%ROWCOUNT;--从游标上取数据后，csr%ROWCOUNT 属性加 1
        PRINT str;
    ELSE
        PRINT 'CURSOR IS NOT OPENED';
    END IF;
    OPEN csr;--再次打开，则会初始化各个属性
    IF csr%ISOPEN THEN
        PRINT 'CURSOR IS ALREADY OPENED';
        PRINT csr%ROWCOUNT;
    ELSE
        PRINT 'CURSOR IS NOT OPENED';
    END IF;
END;
/
```

调用过程：

```
SQL>CALL PROCA;
```

输出：

```
CURSOR IS ALREADY OPENED
1
L2
CURSOR IS ALREADY OPENED
0
```

从上面可以看出，在游标没有关闭就再次打开之后，其属性 ROWCOUNT 又变为 0 了。

当然在打开一个游标的时候，指定的 SQL 语句可以是带参数的，在 DM 中带参数的 SQL 语句是在参数位置用“?”来代替，同时要在打开语句中指定其参数，并且保证参数个数和“?”个数是一致的，同时参数的类型必须是依次都匹配的，不然都会报错。

```
CREATE OR REPLACE PROCEDURE proc AS
DECLARE str varchar;
str_sql varchar := 'SELECT LOGINID FROM RESOURCES.EMPLOYEE WHERE TITLE =? or TITLE =?';
cursor csr;
BEGIN
    open csr for str_sql USING '销售经理','总经理';
    LOOP
    FETCH csr into str;
    EXIT WHEN csr%NOTFOUND;
    print str;
    end loop;
CLOSE csr;
END;
/
```

调用过程：

```
SQL>call    proc;
```

输出结果：

```
 L1
L2
```

3．拨动游标语句

语法格式

```
FETCH [NEXT |PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n] <游标名 >
[INTO <主变量名 >{,<主变量名>} ];
```

参数

1．NEXT 游标下移一行；

2．PRIOR 游标前移一行；

3．FIRST 游标移动到第一行；

4．LAST 游标移动到最后一行；

5．ABSOLUTE 游标移动到第 n 行；

6．RELATIVE 游标移动到当前指示行后的第 n 行；

7．<游标名> 指明被拨动的游标的名称；

8．<主变量名> 指明存储数据的变量名。

使用说明

1．待拨动的游标必须是已打开过的游标。；

2．DM7 通过增加的 [NEXT|PRIOR|FIRST|LAST|ABSOLUTE n|RELATIVE n] 属性设置，可以方便地将游标拨动到结果集任意位置获得数据，游标拨动的默认属性为 NEXT，但这些增强属性只能在过程中使用；

3．当游标被打开后，若不指定游标的移动位置，第一次执行 FETCH 语句时，游标下移，指向工作区中的第一行，以后每执行一次 FETCH 语句，游标均顺序下移一行，使这一行成为当前行；

4．INTO 后的变量个数、类型必须与定义游标语句中、SELECT 后各值表达式的个数、类型一一对应；

5．在 Pro*C 中进行游标查询时，当返回值的数据类型与宿主变量的数据类型不一致时，达梦数据库系统将返回值转换成宿主变量的类型。这种转换只局限于数值转换。不论数据类型如何，如果返回给宿主变量的值是 NULL，那么相应指示符变量被置为-1。如果没有与之相应的指示符变量，那么 SQLCODE 被设置为-5000。数值型数据转换，包括 short、int、double、float 四种数值型数据的转换，若发生溢出错误，将给出警告信息；

6．如果当前游标已经指向查询的最后一条记录，在 Pro*C 中使用 FETCH 语句将会导致返回错误代码(SQLCODE=100)；若是在存储过程中使用，可通过游标属性 FOUND 和 NOTFOUND 的值进行判断。

4．关闭游标

游标在使用完后应及时关闭，以释放它所占用的内存空间，关闭游标的语法格式为：

```
CLOSE   游标名
```

当游标关闭后，不能再从游标中获取数据，否则返回“无效的游标操作.error code = -1800”。如果需要，可以再次打开游标。

5．游标属性

在 DM 中，游标有多个属性，表示当前游标所处的状态。每个游标都有 4 个属性，分别是 FOUND、NOTFOUND、ISOPEN 和 ROWCOUNT，用于描述其状态。具体说明如表 11.7.1 所示。

表 11.7.1　游标属性表

属性名	说　　明

FOUND	如果游标未打开，产生一个异常。否则，在第一次拨动游标之前，其值为 NULL。如果最后一次拨动游标时取到了数据，其值为真，否则为假。
NOTFOUND	如果游标未打开，产生一个异常。否则，在第一次拨动游标之前，其值为 NULL。如果最后一次拨动游标时取到了数据，其值为假，否则为真。
ISOPEN	游标打开时为真，否则为假。
ROWCOUNT	如果游标未打开，产生一个异常。如游标已打开，在第一次拨动游标之前，其值为 0，否则为最后一次拨动后已经取到的元组数。

以上的游标属性称为显式游标属性，其用法如下：

```
<游标名>%<游标属性>
```

下面是显式游标属性的用法举例。使用 OTHER.EMPSALARY 为例。

例 对于基表 EMPSALARY，输出表中的前 5 行数据。如果表中的数据不足 5 行，则输出表中的全部数据。

```
CREATE OR REPLACE PROCEDURE PROC_CURSOR AS
   CURSOR c1 FOR SELECT * FROM OTHER.EMPSALARY;
   my_ename    CHAR(10);
   my_empno    NUMERIC(4);
   my_sal      NUMERIC(7,2);
BEGIN
 OPEN c1;
 LOOP
    FETCH c1 INTO my_ename, my_empno, my_sal;
    EXIT WHEN c1%NOTFOUND;     /*  如表中数据不足 5 行，跳出循环  */
    PRINT my_ename || '    ' || my_empno || '    ' || my_sal;
    EXIT WHEN c1%ROWCOUNT=5; /*  已经输出了 5 行数据，跳出循环*/
 END LOOP;
 CLOSE c1;
END;
```

5．游标 FOR 循环

在大多数情况下，游标的应用步骤是：打开游标；开始循环；从游标中取出一行数据并进行处理；关闭循环；关闭游标，这一类应用称为游标用于循环。但还有一种循环与这种类型不同，这就是游标 FOR 循环。用于 FOR 循环的游标与一般游标的区别在于不需要显式的打开、关闭、取数据，测试数据的存在、定义存放数据的变量等。

比如下面使用游标 FOR 循环，将 PERSONID 加 1。

```
DECLARE
CURSOR STU_CURSOR IS
SELECT * FROM RESOURCES.EMPLOYEE FOR UPDATE;
BEGIN
FOR STU_ROW IN STU_CURSOR LOOP
UPDATE RESOURCES.EMPLOYEE SET PERSONID = PERSONID + 1 WHERE CURRENT OF STU_CURSOR;
END LOOP;
COMMIT;
END;
```

从上例中我们可以看到游标 FOR 循环确实很好的简化了游标的开发，我们不再需要 OPEN、FETCH 和 CLOSE 语句，也不需要用%FOUND 属性检测是否到最后一条记录，这一切 DM 服务器隐式的帮我们完成了。

11.8.2 引用游标

可以定义引用游标：

```
<游标变量名> CURSOR
```

引用游标有以下几个特点：

1. 不只局限于一个查询，可以为一个查询声明或者打开一个游标，然后对其结果集进行处理，之后又可以将这个引用游标为其它的查询打开；
2. 可以对引用游标进行赋值；
3. 可以像用一个变量一样在一个表达式中用这个引用游标；
4. 可以作为一个子程序的参数，也就是它可以像 C 语言中的变量一样充当参数；
5. 可以在PL/SQL的不同子程序中传递结果集。

引用游标的语法、用法与一般游标几乎是一样的，区别在于引用游标在声明的时候没有IS或FOR语句，且同一个游标变量再次打开游标的时候可以关联另一个查询语句，而原来的游标句柄会被自动关闭。还可以对引用游标进行赋值操作，语法如下：

```
DEST_CURSOR := SOURCE_CURSOR;
```

给一个引用游标变量赋值时，可以继承所有SOURCE_CURSOR的属性。如果SOURCE_CURSOR已经打开，则DEST_CURSOR也已经打开，也就是说DEST_CURSOR只是个变量，它相当于一个指针，指向的位置和SOURCE_CURSOR是完全一样的。

```
CREATE OR REPLACE PROCEDURE PROC AS
STR VARCHAR;
CURSOR C1 IS SELECT TITLE FROM RESOURCES.EMPLOYEE WHERE MANAGERID = 3;
CURSOR C2 IS SELECT TITLE FROM RESOURCES.EMPLOYEE WHERE MANAGERID = 7;
CSR CURSOR := C1;
BEGIN
OPEN C1;            --赋值后再打开
      IF CSR%ISOPEN THEN
            PRINT 'CURSOR IS ALREADY OPENED';
            FETCH CSR INTO STR;
            PRINT STR;
      ELSE
            PRINT 'CURSOR IS NOT OPENED';
      END IF;

      OPEN C2;     --打开后再赋值
      CSR := C2;
      IF CSR%ISOPEN THEN
            PRINT 'CURSOR IS ALREADY OPENED';
            FETCH CSR INTO STR;
            PRINT STR;
      ELSE
            PRINT 'CURSOR IS NOT OPENED';
      END IF;
      CLOSE C1;
      CLOSE C2;
END;
```

执行语句：

```
SQL>CALL PROC;
```

运行结果：

```
 CURSOR IS ALREADY OPENED
采购代表
CURSOR IS ALREADY OPENED
系统管理员
```

11.8.3 动态 SQL

动态 SQL 功能在许多应用中得到广泛使用。存储模块中也提供了动态执行 SQL 语句的功能。使用 EXECUTE IMMEDIATE 语句，可准备并立即执行一条动态 SQL 语句。

动态 SQL 执行语句 EXECUTE IMMEDIATE 的作用是执行一个指定的 SQL 语句，这个

SQL 语句可以带参数，也可以不带参数，如果带参数，则 EXECUTE IMMEDIATE 后面还需要加上 USING 来指定 SQL 语句中参数的值，而如果不带参数，EXECUTE IMMEDIATE 后面直接跟上要执行的 SQL 语句即可。所以其语法如下：

```
EXECUTE IMMEDIATE <SQL 动态语句文本> [USING <参数> {,<参数>}];
```

参数

<SQL 动态语句文本> 指明立即执行的动态语句文本。

功能

动态地准备和执行一条语句。

使用说明

1．该语句首先分析动态语句文本，检查是否有错误，如果有错误则不执行它，并在 SQLCODE 中返回错误码；如果没发现错误则执行它；

2．用该方法处理的 SQL 语句一定不是 SELECT 语句，而且不包含任何虚拟的输入宿主变量；

3．一般来说，应该使用一个字符串变量来表示一个动态 SQL 语句的文本，下列语句不能作动态 SQL 语句：CLOSE、DECLARE、FETCH、OPEN、WHENEVER；

4．如果动态 SQL 语句的文本中有多条 SQL 语句，那么只执行第一条语句；

5．对于仅执行一次的动态语句，用立即执行语句较合适。

```
CREATE OR REPLACE PROCEDURE proc AS
DECLARE
str_sql varchar := 'select * from sysobjects where name = ?';
BEGIN
      EXECUTE IMMEDIATE str_sql USING 'DUAL';
END;
/
```

如果动态 SQL 语句是查询语句，并且结果集只会是一条记录的话，可以通过 INTO 语句来操作查询到的结果集，把查询到的结果存储到某一个变量中。其语法如下：

```
EXECUTE IMMEDIATE <SQL 动态语句文本> [INTO variables][USING <参数> {,<参数>}];
```

如果这样使用的话一定要保证查询结果集只有一条记录或者无记录，如果记录数大于 1 的话就会报出“SELECT INTO 中包含多行数据.error code = -3401”的错误，这样的使用场景主要是通过查询语句来给一个变量赋值然后使用这个变量，前提是保证结果集只有一条记录。下面是使用 INTO 语句的例子：

```
CREATE OR REPLACE PROCEDURE proc AS
DECLARE
str varchar;
str_sql varchar := 'select TYPE$ from sysobjects where NAME =?';
csr cursor;
BEGIN
      EXECUTE IMMEDIATE str_sql into str USING 'DUAL';
      print str;
END;
/
```

调用过程：

```
SQL>call proc;
```

输出结果：

```
DSYNOM
```

11.8.4 动态 SQL 的参数绑定

在动态执行 SQL 语句的时候也可以对指定的 SQL 语句设置参数，指定参数有两种方式，一种是用“?”来表示；另一只是用“:variable”来表示。

用“?”来表示参数时，在传入时可以是任意的值，但参数个数一定要与“?”的个数相同，

同时数据类型一定要匹配（可以互相转换也可以），不然会报数据类型不匹配的错误。

用“:variable”来表示时，对于 variable 的名字会进行检测，如果参数列表中存在相同名字的 variable，则用这个名字指定的参数被设置的值在第一次绑定的时候确定，同时同名的其它参数都不能再被绑定参数，否则会报“参数个数不匹配.error code = -3205”的错误。

用“?”来表示参数的例子上一节已有使用。下面列举使用“:variable”来表示参数的例子。

```
CREATE OR REPLACE PROCEDURE proc AS--参数个数匹配
DECLARE
name varchar := 'damengren';
str int:= 6;
start int:= 1;
str_sql varchar := 'select substr(:x,:y,:y);';
BEGIN
EXECUTE IMMEDIATE str_sql USING name,start,str;
END;
/
```

调用过程：

```
SQL>call proc
```

输出结果：

```
dameng
```

11.8.5 返回查询结果集

在存储过程中如果执行了不带INTO子句的查询语句，系统将在调用结束时将该查询结果集返回给调用者。当出现多个查询语句时，只有第一个查询语句的结果集被返回，如果想知道其他结果集，在disql里输入 more 命令查看下一条。下面给出一个例子：

下面的例子需要拥有SYSDBA的权限。

```
CREATE OR REPLACE PROCEDURE p_sel_result(UserTag byte) AS
BEGIN
IF (UserTag = 1) THEN
SELECT NAME,ID FROM SYSOBJECTS WHERE ID = 3;
SELECT NAME,ID FROM SYSOBJECTS WHERE ID = 4;
ELSE
SELECT NAME,ID FROM SYSOBJECTS WHERE ID =9;
SELECT NAME,ID FROM SYSOBJECTS WHERE ID =10;
END IF;
END;
/
```

调用p_sel_result过程，仅返回第一条查询语句的结果：

```
SQL> call p_sel_result(1);
```

结果如下：

```
NAME          ID
SYSUSERS      3
```

再调用more命令：

```
SQL> more
```

结果如下：

```
NAME          ID
SYSCONS       4
```

11.8.6 自治事务

自治事务仅能在 PL/SQL 块中定义。定义自治事务的语法如下：

```
PRAGMA    AUTONOMOUS_TRANSACTION;
```

功能

定义一个自治事务。

使用说明

1. 作为自治事务的 PL/SQL 块可以是下面中的一种：
   1) 最顶层的（不是嵌套的）匿名 PL/SQL 块
   2) 函数和过程，或者在一个包里定义或者是一个独立的程序
   3) 对象类型的方法
   4) 数据库触发器
   5) 嵌套子过程
2. 自治事务的定义语句可以放在 PL/SQL 块声明单元的任何地方，但推荐放在数据结构声明之前。

例如，执行下列语句：

```
CREATE TABLE t1 (c1 INT);

INSERT INTO t1 VALUES(1);

DECLARE
PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO t1 VALUES(301);
    COMMIT;
END;
/

ROLLBACK;

SELECT * FROM t1;
```

可以看到，最后查询表 t1 中有记录 301，其所在自治事务提交，不受主事务回滚影响，而主事务插入的记录 1 被回滚。

11.9 客户端存储模块

DM 支持用户使用基于客户端的存储模块：客户端存储模块和普通存储模块功能一致，不过其处理方法为预编译阶段将其转化为虚过程的处理。虚过程不需要存储，创建后立即执行，当执行的语句释放时，虚过程对象也一同被释放。而普通存储模块编译后保存在服务器端，提高了性能，客户端存储模块只是从语法上和普通存储模块兼容，完成和存储模块一样的功能，是一种编程手段。

语法说明

客户端存储模块语法和普通存储模块类似，如下：

```
<过程语句脚本> ::= <模块体>
<模块体> ::=
[<说明部分>]
<执行部分>
[<异常处理部分>]
END
```

使用说明

1．与过程定义中不同，[<说明部分>]必须包含 DECLARE；

2．客户端存储模块中支持的 SQL 语句包括：

   1) 数据查询语句(SELECT)；
   2) 数据操纵语句(INSERT、DELETE、UPDATE)；
   3) 游标定义及操纵语句(DECLARE CURSOR、OPEN、FETCH、CLOSE)；

4) 事务控制语句(COMMIT、ROLLBACK)；

5) 动态 SQL 执行语句(EXECUTE IMMEDIATE)。

举例说明

例 比如表 OTHER.ACCOUNT，先清空该表的数据，现希望模仿自增列将该表数据初始化，执行模块语句。

```
DECLARE I INT:= 0;
BEGIN
DELETE FROM OTHER.ACCOUNT;
LOOP
IF I>10 THEN
EXIT;
      END IF;
INSERT INTO OTHER.ACCOUNT VALUES (I,1500);
I:=I+1;
END LOOP;
END;
```

查询表数据，OTHER.ACCOUNT 表数据应该如表 11.9.1 所示。

表 11.9.1

ACCOUNT_ID	BAL
0	1500
1	1500
2	1500
3	1500
4	1500
5	1500
6	1500
7	1500
8	1500
9	1500
10	1500

11.10 C 语法的 PL/SQL

11.10.1 概述

在我们印象中，PL/SQL 是很复杂的，因为它有自己的语法规则，用户通常都不会去记那么多复杂的语法，只是在用的时候查看手册，所以写一个可以执行并且正确的 PL/SQL 语句是比较困难的。

在 DM7 中，第一次实现了用 C 语言语法作为 PL/SQL 的一个可选语法，这就为那些了解 C 语言的人提供了很大的方便性，无需去查看手册就可以很自如的完成一个语句块，对 SQL 程序员而言，这个功能无疑是他们梦寐以求的。

11.10.2 举例说明

在 C 语法 PL/SQL 的应用方面，定义一个语句块不需要用 BEGIN 及 END 把语句包含

起来，而是直接用大括号括住即可。下面给出两个 C 语法的 PL/SQL 的例子来更具体地说明。

例 1

```
{
    string      str=' Hello World';
    int         count = 0;
    for(count = 0; count < 10; count++)
    {
        if(power(count, 2) % 2 == 0)
        print concat(cast(power(count, 2) as int), str);
    }
}
```

输出：

```
0 Hello World
4 Hello World
16 Hello World
36 Hello World
64 Hello World
```

例 2

```
{
    try
    {
        SELECT 1/0;
    }
    CATCH(EXCEPTION EX)
    {
        throw new exception(-20002,'TEST');
    }
}
```

输出：

```
[-20002]:TEST.
```

从上面两个例子可以看出，用 C 语法的 PL/SQL 时，程序可以变得非常简单易懂，可以很自由地调用一些系统内部函数（如上面例子中的 concat()、power()）、存储函数、过程等等。可以定义像 C#中的一些数据类型，如上面例子所示 STRING 类型，还可以定义 C 语言中的基本数据类型，如上面例子中的 int，另外还支持全部的 SQL 类型，达梦数据库内部定义的类类型包括 EXCEPTION 类、数组类型、游标类型等。

11.11 C 外部函数

11.11.1 概述

为了能够在创建和使用自定义 PL/SQL 时，使用其他语言实现的接口，DM7 提供了 C 外部函数。C 外部函数的调用都通过代理进程进行，这样即使外部函数在执行中出现了任何问题，都不会影响到服务器的正常执行。

C 外部函数是使用 C、C++语言编写，在数据库外编译并保存在.dll、.so 共享库文件中，被用户通过 PL/SQL 调用的函数。

当用户调用 C 外部函数时，服务器操作步骤如下：首先，确定调用的（外部函数使用的）共享库及函数；然后，通知代理进程工作。代理进程装载指定的共享库，并在函数执行后将结果返回给服务器。

11.11.2 生成动态库

用户必须严格按照如下格式书写代码。

C 外部函数格式

```
de_data  函数名(de_args *args)
{
      C 语言函数实现体;
}
```

参数

1. <de_data> 返回值类型。de_data 结构体类型如下：

```
struct de_data{
      int   null_flag;    /*参数是否为空，1 表示非空，0 表示空*/
      union          /*只能为 int、double 或 char 类型*/
      {
            int      v_int;
            double   v_double;
            char     v_str[];
      }data;
};
```

2. <de_args> 参数信类型。de_args 结构体类型如下：

```
struct de_args
{
      int            n_args;         /*参数个数*/
     de_data*         args;           /*参数列表*/
};
```

3. < C 语言函数实现体> C 语言函数对应的函数实现体。

使用说明

1. C 语言函数的参数可通过调用 DM7 提供的一系列 get 函数得到，同时可调用 set 函数重新设置这些参数的值；
2. 根据返回值类型，调用不同的 return 函数接口。
3. 必须根据参数类型、返回值类型，调用相同类型的 get、set 和 return 函数。当调用 de_get_str 和 de_get_str_with_len 得到字符串后，必须调用 de_str_free 释放空间。
4. DM7 提供的编写 C 外部函数动态库的接口如表 11.11.1 所示。

表 11.11.1 DM7 支持的编写 C 外部函数动态库的接口

函数类型	函数名	功能说明
get	int de_get_int(de_args *args, int arg_id);	第 arg_id 参数的数据类型为整型，从参数列表 args 中取出第 arg_id 参数的值，
	double de_get_double(de_args *args, int arg_id);	第 arg_id 参数的数据类型为 double 类型，从参数列表 args 中取出第 arg_id 参数的值。
	char* de_get_str(de_args *args, int arg_id);	第 arg_id 参数的数据类型为字符串类型，从参数列表 args 中取出第 arg_id 参数的值。
	char* de_get_str_with_len(de_args *args, int arg_id, int* len);	第 arg_id 参数的数据类型为字符串类型，从参数列表 args 中取出第 arg_id 参数的值以及字符串长度。
set	void de_set_int(de_args *args, int arg_id, int ret);	第 arg_id 参数的数据类型为整型，设置参数列表 args 的第 arg_id 参数的值为

		ret。
	void de_set_double(de_args *args, int arg_id, double ret);	第 arg_id 参数的数据类型为 double 类型，设置参数列表 args 的第 arg_id 参数的值为 ret。
	void de_set_str(de_args *args, int arg_id, char* ret);	第 arg_id 参数的数据类型为字符串类型，设置第 arg_id 参数的值为 ret。
	void de_set_str_with_len(de_args *args, int arg_id, char* ret, int len);	第 arg_id 参数的数据类型为字符串类型，将字符串 ret 的前 len 个字符赋值给参数列表 args 的第 arg_id 参数。
	void de_set_null(de_args *args, int arg_id);	设置参数列表 args 的第 arg_id 个参数为空。
return	de_data de_return_int(int ret);	返回值类型为整型。
	de_data de_return_double(double ret);	返回值类型为 double 型。
	de_data de_return_str(char* ret);	返回值为字符串类型。
	de_data de_return_str_with_len(char* ret, int len);	返回字符串 ret 的前 len 个字符。
	de_data de_return_null();	返回空值。
de_str_free	void de_str_free(char* str);	调用 de_get_str 函数后，需要调用此函数释放字符串空间。
de_is_null	int de_is_null(de_args *args, int arg_id);	判断参数列表 args 的第 arg_id 个参数是否为空。

注：参数个数 arg_id 的起始值为 0。

11.11.3 C 外部函数创建

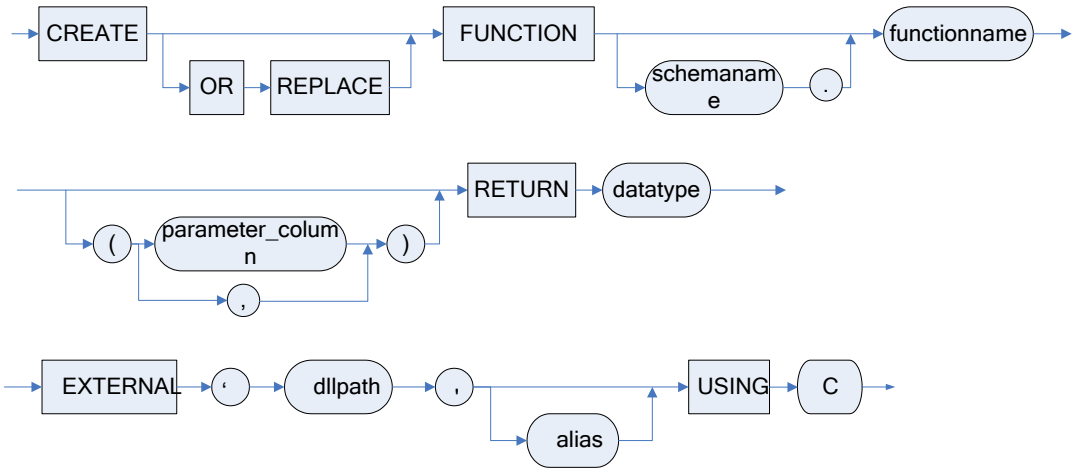
语法格式

```
CREATE OR REPLACE FUNCTION [模式名.]函数名[( 参数列表 )]
RETURN 返回值类型
EXTERNAL '<动态库路径>' [<引用的函数名>] USING C;
```

参数

1. <函数名> 指明被创建的 C 外部函数的名字；
2. <模式名> 指明被创建的 C 外部函数所属模式的名字，缺省为当前模式名；
3. <参数列表> 指明 C 外部函数参数信息，参数模式可设置为 IN、OUT 或 IN OUT（OUT IN），缺省为 IN 类型；
4. <动态库路径> 用户按照 DM7 规定的 C 语言函数格式编写的 DLL 文件生成的动态库所在的路径；
5. <引用函数名> 指明<函数名>在<动态库路径>中对应的函数名。

图例



语句功能

创建自定义 C 外部存储过程和函数。

使用说明

1．<引用函数名>如果为空，则默认与<函数名>相同；

2．<动态库路径>分为.dll 文件（windows）和.so 文件（linux）两种。

权限

使用该语句的用户必须是 DBA 或该存储过程的拥有者且具有 CREATE FUNCTION 数据库权限的用户。

11.11.2 举例说明

例 编写（C 语言）外部函数 C_CONCAT，用于将两个字符串连接。

首先，生成动态库。

第一步，创建新项目 newp，位于 d:\xx\tt 文件夹中。在 d:\xx\tt 文件夹中，直接拷入 dmde.lib 动态库和 de_pub.h 头文件。

第二步，在 newp 项目中，添加头文件。将已有 de_pub.h 头文件添加进来，同时，添加新的 tt.h 头文件。tt.h 文件内容如下：

```
#include "de_pub.h"
#include "string.h"
#include "stdlib.h"
```

第三步，在 newp 项目中，添加源文件。名为 tt.c。tt.c 内容如下：

```
#include "tt.h"

de_data C_CONCAT(de_args *args)
{
	de_data	de_ret;
	char* str1;
	char* str2;
	char* str3;
	int len1;
	int len2;

	str1 = de_get_str(args, 0);                    /*从参数列表中取第 0 个参数*/
	str2 = de_get_str_with_len(args, 1, &len2);    /*从参数列表中取第 1 个参数的值以及长度*/
	len1 = strlen(str1);
	str3 = malloc(len1 + len2);
	memcpy(str3, str1, len1);
	memcpy(str3 + len1, str2, len2);
```

```
    de_str_free(str1);      /*调用 get 函数得到字符串之后，需要调用此函数释放字符串空间*/
    de_str_free(str2);

    de_ret = de_return_str_with_len(str3, len1 + len2);      /*返回字符串*/
    free(str3);
    return de_ret;
}
```

第四步，在 newp 项目的源文件中，添加模块定义文件 tt.def，内容如下：

```
LIBRARY "tt.dll"
EXPORTS
    C_CONCAT
```

第五步，编译 tt 项目（设置项目输出文件路径：D:\xx\tt）。得到 tt.dll 文件。

第六步，编译 cexe 代理项目。

至此，外部函数的使用环境准备完毕。

其次，创建并使用外部函数。

第一步，启动数据库服务器（设置工作目录：D:\srcdm7\debug）。启动 disql。

第二步，在 disql 中，创建外部函数 MY_CONCAT3，语句如下：

```
CREATE OR REPLACE FUNCTION MY_CONCAT(A VARCHAR, B VARCHAR)
RETURN VARCHAR
EXTERNAL 'd:\xx\tt\tt.dll' C_CONCAT USING C;
```

第三步，调用 C 外部函数，语句如下：

```
select MY_CONCAT ('hello ', 'world!');
```

第四步，查看结果：

```
hello world!
```

第 12 章 包

DM7 支持 PL/SQL 包来扩展数据库功能，用户可以通过包来创建应用程序或者使用包来管理过程和函数。

12.1 创建包

包的创建包括包规范和包主体的创建。

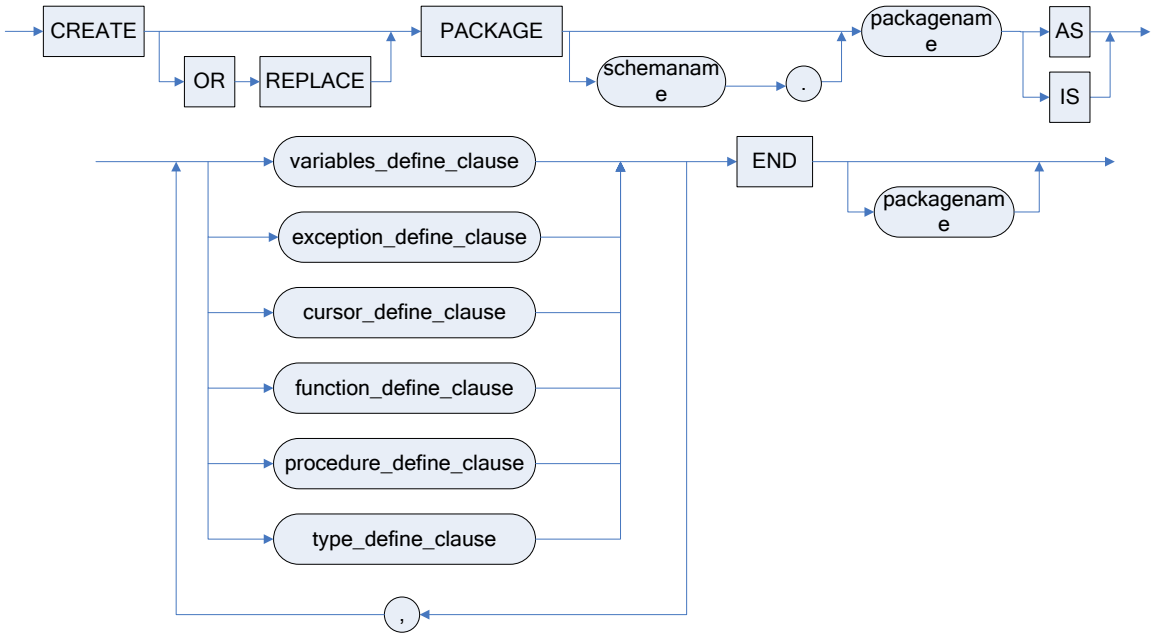
12.1.1 创建包规范

包规范中包含了有关包中的内容信息，但是它不包含任何过程的代码。定义一个包规范的详细语法如下。

语法格式

```
CREATE [OR REPLACE] PACKAGE [<模式名>.]<包名> AS|IS <包内声明列表>  END [包名]
<包内声明列表> ::= <包内声明>;{<包内声明>;}
<包内声明> ::= <变量列表定义>|<游标定义>|<异常定义>|<过程定义>|<函数定义>|<类型声名>
<变量列表定义>::= <变量定义>{<变量定义>}
<变量定义> ::= <变量名><变量类型>[DEFAULT|ASSIGN|:=<表达式>]
<变量类型>::=<PLSQL 类型>|< [模式名.]表名.列名%TYPE>|<[模式名.]表名%ROWTYPE>|<记录类型>
<记录类型>::= RECORD(<变量名> <PLSQL 类型>;{<变量名> <PLSQL 类型>;})
<游标定义> ::= CURSOR <游标名> [FOR <查询语句>]
<异常定义> ::= <异常名> EXCEPTION [FOR <异常码>]
<过程定义> ::= PROCEDURE <过程名> <参数列表>
<函数定义> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型> [PIPELINED]
<类型声名> ::= TYPE <类型名称> IS <数据类型>
```

图例



使用说明

1. 包部件可以以任意顺序出现，其中的对象必须在引用之前被声明；
2. 过程和函数的声明都是前向声明，包规范中不包括任何实现代码。

权限

使用该语句的用户必须是 DBA 或该包对象的拥有者且具有 CREATE PACKAGE 数据库权限的用户。

12.1.2 创建包主体

包主体中包含了在包规范中的前向子程序声明相应的代码。它的创建语法如下。

语法格式

```
CREATE [OR REPLACE] PACKAGE BODY [<模式名>.]<包名> AS|IS <包体部分> END [包名]
<包体部分> ::= <包体声明列表> [<初始化代码>]
<包体声明列表> ::=<包体声明>[,{<包体声明> }……]
<包体声明>::=<变量定义>|<游标定义>|<异常定义>|<过程定义>|<函数定义>|<类型声名> |<存储过程实现>|<函数实现>
<变量定义> ::= <变量名列表> <数据类型> [默认值定义]
<游标定义> ::= CURSOR <游标名> [FOR <查询语句>]
<异常定义> ::= <异常名> EXCEPTION [FOR <异常码>]
<过程定义> ::= PROCEDURE <过程名> <参数列表>
<函数定义> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型>
<类型声名> ::= TYPE <类型名称> IS <数据类型>
<存储过程实现> ::= PROCEDURE <过程名> <参数列表>  AS|IS  BEGIN <实现体> END [过程名];
<函数实现> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型>[PIPELINED] <AS|IS> BEGIN <实现体> END [函数名];
<初始化代码> ::= [[<说明部分>]BEGIN<执行部分>[<异常处理部分>]]
<说明部分> ::=[DECLARE]<说明定义>{<说明定义>}
<说明定义>::=<变量列表说明>|<异常变量说明>|<游标定义>|<子过程定义>|<子函数定义>;
```

<变量列表说明>::= <变量初始化>{<变量初始化>}

<记录类型>::= RECORD(<变量名> <PLSQL 类型>;{<变量名> <PLSQL 类型>;})

<异常变量说明>::=<异常变量名>EXCEPTION[FOR<错误号>]

<游标定义>::=cursor <游标名> [FOR<不带 INTO 查询表达式>|<表连接>]

<子过程定义>::=PROCEDURE<过程名>[(<参数列>)]<IS|AS><模块体>

<子函数定义>::=FUNCTION<函数名>[(<参数列>)]RETURN<返回数据类型><IS|AS><模块体>

<执行部分>::=<SQL 过程语句序列>{< SQL 过程语句序列>}

< SQL 过程语句序列>::=[<标号说明>]<SQL 过程语句>;

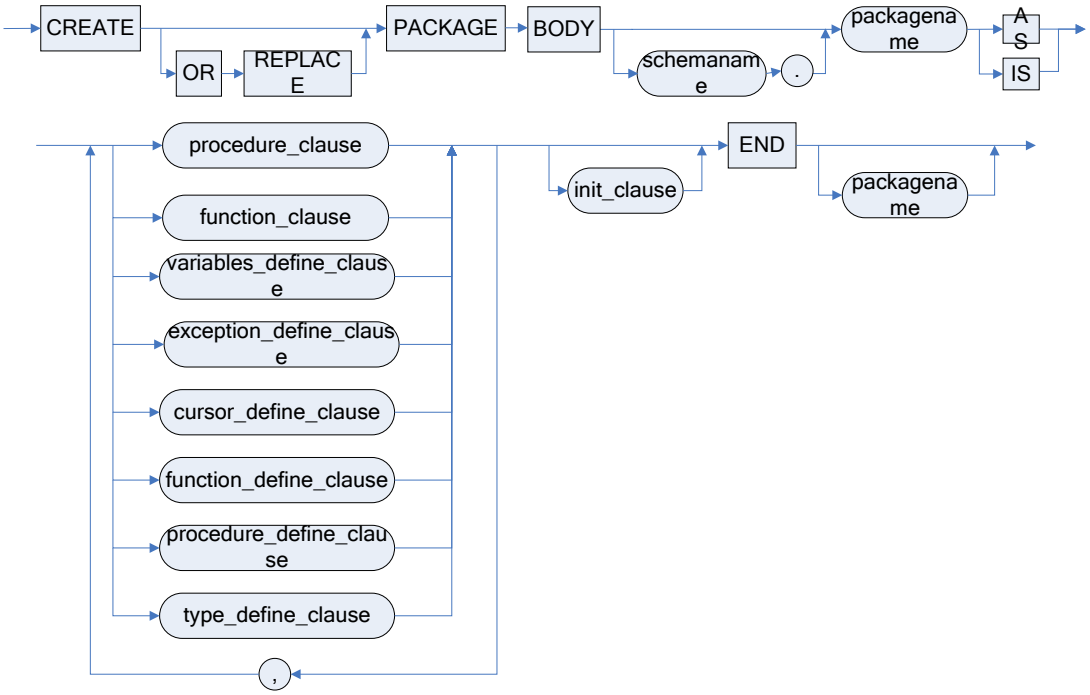
<标号说明>::=<<<标号名>>>

<SQL 过程语句>::=<SQL 语句>|<SQL 控制语句>

<异常处理部分>::=EXCEPTION<异常处理语句>{<异常处理语句>}

<异常处理语句>::= WHEN <异常名> THEN < SQL 过程语句序列>;

图例



使用说明

1. 包规范中定义的对象对于包主体而言都是可见的，不需要声明就可以直接引用。这些对象包括变量、游标、异常定义和类型定义；

2. 包主体中不能使用未在包规范中声明的对象；

3. 包主体中的过程、函数定义必须和包规范中的前向声明完全相同。包括过程的名字、参数定义列表的参数名和数据类型定义；

4. 包中可以有重名的过程和函数，只要它们的参数定义列表不相同。系统会根据用户的调用情况进行重载(OVERLOAD)；

5. 用户在第一次调用包内过程、函数时，系统会自动将包对象实例化。每个会话根据数据字典内的信息在本地复制包内变量的副本。如果用户定义了 PACKAGE 的初始化代码，还必须执行这些代码(类似于一个没有参数的构造函数执行)；

6. 对于一个会话，包头中声明的对象都是可见的，只要指定包名，用户就可以访问这些对象。可以将包头内的变量理解为一个 SESSION 内的全局变量；

7. 关于包内过程、函数的调用：DM 支持按位置调用和按名调用参数两种模式。除了

需要在过程、函数名前加入包名作为前缀，调用包内的过程、函数的方法和普通的过程、函数并无区别。

8. 包体内声明的变量、类型、方法以及实现的未在包头内声明的方法被称作本地变量、方法，本地变量、方法只能在包体内使用，用户无法直接使用。

9. 在包体声明列表中，本地变量必须在所有的方法实现之前进行声明；本地方法必须在使用之前进行声明或实现。

权限

使用该语句的用户必须是 DBA 或该包对象的拥有者且具有 CREATE PACKAGE 数据库权限的用户。

12.2 删除包

和创建方式类似，包对象的删除分为包规范的删除和包主体的删除。

12.2.1 删除包规范

从数据库中删除一个包对象。

语法格式

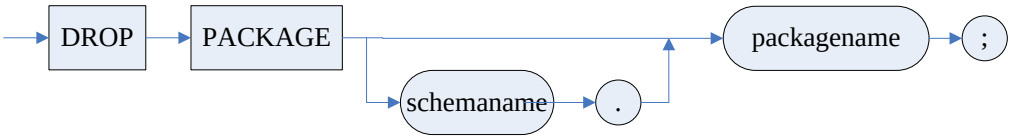
```
DROP PACKAGE [<模式名>.]<包名>;
```

参数

1．<模式名>　指明被删除的包所属的模式，缺省为当前模式；

2．<包名>　指明被删除的包的名字。

图例



使用说明

1．如果被删除的包不属于当前模式，必须在语句中指明模式名；

2．如果一个包规范被删除，那么对应的包主体被自动删除。

权限

执行该操作的用户必须是该包的拥有者，或者具有 DBA 权限。

12.2.2 删除包主体

从数据库中删除一个包的主体对象。

语法格式

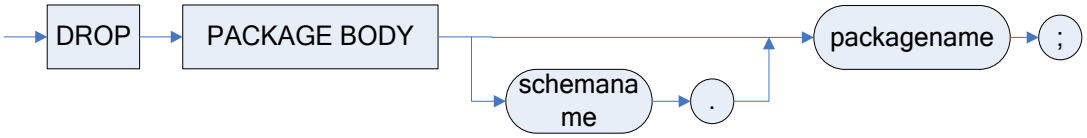
```
DROP PACKAGE BODY [<模式名>.]<包名>;
```

参数

1．<模式名>　指明被删除的包所属的模式，缺省为当前模式；

2．<包名>　指明被删除的包的名字。

图例



使用说明

如果被删除的包不属于当前模式，必须在语句中指明模式名。

权限

执行该操作的用户必须是该包的拥有者，或者具有 DBA 权限。

12.3 应用实例

以下是一个包规范的使用实例：

创建实例数据库，这些数据将在之后的例子中用到。

```
CREATE TABLE Person(Id INT IDENTITY, Name VARCHAR(100), City VARCHAR(100));
INSERT INTO Person(Name, City) VALUES('Tom','武汉');
INSERT INTO Person(Name, City) VALUES('Jack','北京');
INSERT INTO Person(Name, City) VALUES('Mary','上海');
```

表中数据如表 12.3.1 所示。

表 12.3.1

ID	NAME	CITY
1	TOM	武汉
2	JACK	北京
3	MARY	上海

创建包规范：

```
  CREATE OR REPLACE PACKAGE PersonPackage AS
        E_NoPerson EXCEPTION;
        PersonCount INT;
        Pcur CURSOR;
        PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100));
        PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100));
        PROCEDURE RemovePerson(Pid INT);
        FUNCTION GetPersonCount RETURN INT;
        PROCEDURE PersonList;
END PersonPackage;
```

这个包规范的部件中包括 1 个变量定义，1 个异常定义，1 个游标定义，4 个过程定义和 1 个函数定义。

以下是一个包主体的实例，它对应于前面的包规范定义，包括 4 个子过程和 1 个子函数的代码实现。在包主体的末尾，是这个包对象的初始化代码。当一个会话第一次引用包时，变量 PersonCount 被初始化为 Person 表中的记录数。

创建包主体：

```
CREATE OR REPLACE PACKAGE BODY PersonPackage AS

  PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100) )AS
```

```
    BEGIN
      INSERT INTO Person(Name, City) VALUES(Pname, Pcity);
      PersonCount = PersonCount + SQL%ROWCOUNT;
    END AddPerson;

  PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100)) AS
    BEGIN
      DELETE FROM Person WHERE NAME LIKE Pname AND City like Pcity;
      PersonCount = PersonCount - SQL%ROWCOUNT;
    END RemovePerson;

  PROCEDURE RemovePerson(Pid INT) AS
    BEGIN
      DELETE FROM Person WHERE Id = Pid;
      PersonCount = PersonCount - SQL%ROWCOUNT;
    END RemovePerson;

  FUNCTION GetPersonCount RETURN INT AS
  BEGIN
    RETURN PersonCount;
  END GetPersonCount;

  PROCEDURE PersonList AS
  DECLARE
    V_id INT;
    V_name VARCHAR(100);
    V_city VARCHAR(100);
  BEGIN
    IF PersonCount = 0 THEN
       RAISE E_NoPerson;
    END IF;
    OPEN Pcur FOR SELECT Id, Name, City FROM Person;
    LOOP
      FETCH Pcur INTO V_id,V_name,V_city;
      EXIT WHEN Pcur%NOTFOUND;
      PRINT ('No.' + (cast (V_id as varchar(100))) + '    ' + V_name + '来自' + V_city );
    END LOOP;
    CLOSE Pcur;
  END PersonList;

BEGIN
    SELECT COUNT(*) INTO PersonCount FROM Person;
END PersonPackage;
```

调用包中的 AddPerson 过程，往数据表中增加一条记录：

```
CALL    PersonPackage. AddPerson ('BLACK', '南京') ;
```

当前记录变化如表 12.3.2 所示。

表 12.3.2

ID	NAME	CITY
1	TOM	武汉
2	JACK	北京
3	MARY	上海
4	BLACK	南京

调用包中的 RemovePerson 过程，删除第二条记录：

```
CALL    PersonPackage. RemovePerson ('JACK', '北京') ;
或者
    CALL    PersonPackage. RemovePerson (2) ;
```

在此例中，以上两种写法可以得到相同的结果，系统对同名过程根据实际参数进行了重载。如果过程执行结果没有删除任何一条表中的记录，那么会抛出一个包内预定义的异常：E_NoPerson。

此时表中的数据如表 12.3.3 所示。

表 12.3.3

ID	NAME	CITY
1	TOM	武汉
3	MARY	上海
4	BLACK	南京

引用包中的变量。

```
SELECT PersonPackage. PersonCount;
或者
SELECT PersonPackage. GetPersonCount;
```

以上两句语句的作用是等价的。前一句是直接引用了包内变量，后一句是通过调用包内的子函数来得到想要的结果。

调用包中的过程 PersonList 查看表中的所有记录：

```
CALL    PersonPackage. PersonList;
```

可以得到以下输出：

```
NO.1    Tom 来自武汉
No.3    MARY 来自上海
No.4    BLACK 来自南京
```

第 13 章 类类型

DM7 通过类类型在 PL/SQL 中实现面向对象编程的支持。类将结构化的数据及对其进行操作的过程或函数封装在一起。允许用户根据现实世界的对象建模，而不必再将其抽象成关系数据。

DM7 的类的定义分为类头和类体两部分，类头完成类的声明；类体完成类的实现。

类中可以包括以下内容：

1. 类型定义

在类中可以定义游标、异常、记录类型、数组类型、以及内存索引表等数据类型，在类的声明及实现中可以使用这些数据类型；类的声明中不能声明游标和异常，但是实现中可以定义和使用。

2. 属性

类中的成员变量，数据类型可以是标准的数据类型，可以是在类中自定义的特殊数据类型。

3. 成员方法

类中的函数或过程，在类头中进行声明；其实现在类体中完成；

成员方法及后文的构造函数包含一个隐含参数，即自身对象，在方法实现中可以通过 this 来访问自身对象，如果不存在重名问题，也可以直接使用对象的属性和方法。

4. 构造函数

构造函数是类内定义及实现的一种特殊的函数，这类函数用于实例化类的对象，构造函数满足以下条件：

1) 函数名和类名相同；

2) 函数返回值类型为自身类。

构造函数存在以下的约束：

1) 系统为每个类提供两个默认的构造函数，分别为 0 参的构造函数和全参的构造函数；

2) 0 参构造函数的参数个数为 0，实例的对象内所有的属性初始化值为 NULL；

3) 全参构造函数的参数个数及类型和类内属性的个数及属性相同，按照属性的顺序依次读取参数的值并给属性赋值；

4) 用户可以自定义构造函数，一个类可以有多个构造函数，但每个构造函数的参数个数必须不同；

5) 如果用户自定义了 0 个参数、或参数个数同属性个数相同的构造函数，则会覆盖相应的默认构造函数。

下面从类的声明、类的实现、类的删除、类体的删除和类的使用几部分来详细介绍类类型的实现过程。

13.1 声明类

类的声明在类头中完成。类头定义通过 CREATE CLASS 语句来完成，其语法为：

语法格式

```
CREATE [OR REPLACE] CLASS [<模式名>.]<类名> AS|IS <类内声明列表>  END [类名]
```

<类内声明列表> ::= <类内声明>;{<类内声明>;}

<类内声明> ::= <变量定义>|<过程定义>|<函数定义>|<类型声名>

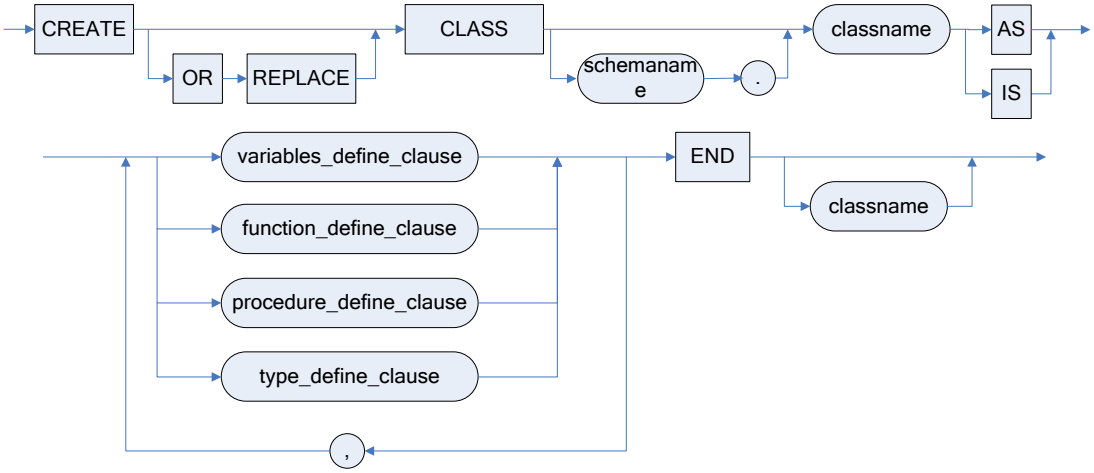
<变量定义> ::= <变量名列表> <数据类型> [默认值定义]

<过程定义> ::= PROCEDURE <过程名> <参数列表>

<函数定义> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型> [PIPELINED]

<类型声名> ::= TYPE <类型名称> IS <数据类型>

图例



使用说明

1. 类中元素可以以任意顺序出现，其中的对象必须在引用之前被声明；
2. 过程和函数的声明都是前向声明，类声明中不包括任何实现代码。

权限

使用该语句的用户必须是 DBA 或具有 CREATE CLASS 数据库权限的用户。

13.2 实现类

类的实现通过类体完成。类体的定义通过 CREATE CLASS BODY 语句来完成，其语法为：

语法格式

CREATE [OR REPLACE] CLASS BODY [<模式名>.]<类名> AS|IS <类体部分> END [类名]

<类体部分> ::= <存储过程/函数列表> [<初始化代码>]

<存储过程/函数列表> ::= <存储过程实现|函数实现>{,<存储过程实现|函数实现> }

<存储过程实现> ::= PROCEDURE <过程名> <参数列表> AS|IS　BEGIN <实现体> END [过程名]

<函数实现> ::= FUNCTION <函数名> <参数列表> RETURN <返回值数据类型> [PIPELINED] AS|IS BEGIN <实现体> END [函数名]

<初始化代码> ::= [[<说明部分>]BEGIN<执行部分>[<异常处理部分>]]

<说明部分> ::=[DECLARE]<说明定义>{<说明定义>}

<说明定义>::=<变量说明>|<异常变量说明>|<游标定义>|<子过程定义>|<子函数定义>

<变量说明>::=<变量名>{,<变量名>}<变量类型>[DEFAULT|ASSIGN|:=<表达式>];

<变量类型>::=<PLSQL 类型>|< [模式名.]表名.列名%TYPE>|<[模式名.]表名%ROWTYPE>|<记录类型>

<记录类型>::= RECORD(<变量名> <PLSQL 类型>{,<变量名> <PLSQL 类型>})

<异常变量说明>::=<异常变量名>EXCEPTION[FOR<错误号>]

<游标定义>::=CURSOR <游标名> [FOR<不带 INTO 查询表达式>|<表连接>]

<子过程定义>::=PROCEDURE<过程名>[(<参数列>)]<IS|AS><模块体>

<子函数定义>::=FUNCTION<函数名>[(<参数列>)]RETURN<返回数据类型> [PIPELINED]<IS|AS><模块体>

<执行部分>::=<SQL 过程语句序列>{;< SQL 过程语句序列>}

< SQL 过程语句序列>::=[<标号说明>]<SQL 过程语句>;

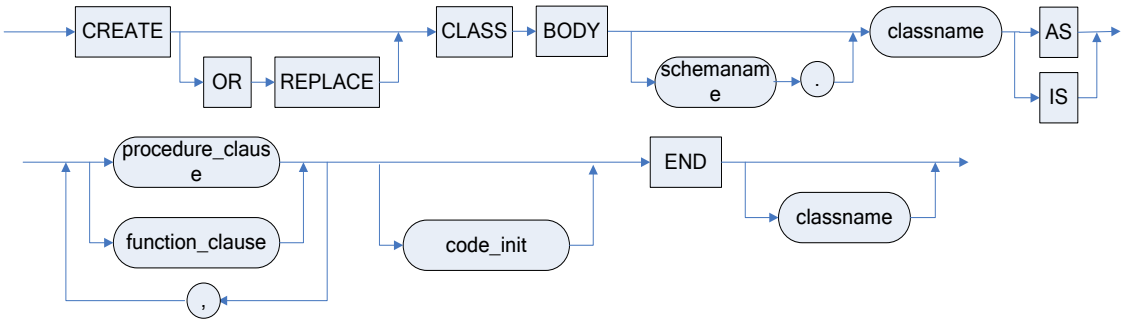
<标号说明>::=<<<标号名>>>

<SQL 过程语句>::=<SQL 语句>|<SQL 控制语句>

<异常处理部分>::=EXCEPTION<异常处理语句>{;<异常处理语句>};

<异常处理语句>::= WHEN <异常名> THEN < SQL 过程语句序列>

图例



使用说明

1. 类声明中定义的对象对于类体而言都是可见的，不需要声明就可以直接引用。这些对象包括变量、游标、异常定义和类型定义；

2. 类体中的过程、函数定义必须和类声明中的声明完全相同。包括过程的名字、参数定义列表的参数名和数据类型定义；

3. 类中可以有重名的成员方法，要求其参数定义列表各不相同。系统会根据用户的调用情况进行重载(OVERLOAD)；

权限

使用该语句的用户必须是 DBA 或该类对象的拥有者且具有 CREATE CLASS 数据库权限的用户。

完整的类头、类体的创建如下所示：

```
----类头创建
create class mycls
as
type rec_type is   record (c1 int, c2   int);      --类型声明
id                 int;                            --成员变量
r                  rec_type;                  --成员变量
function f1(a int, b int) return rec_type;  --成员函数
function mycls(id int , r_c1 int, r_c2 int) return mycls;
--用户自定义构造函数
end;
/
----类体创建
create or replace class body mycls
as
```

```
function f1(a int, b int) return rec_type
as
begin
        r.c1 = a;
        r.c2 = b;
        return r;
end;
function mycls(id int, r_c1 int, r_c2 int) return mycls
as
begin
        this.id = id;        --可以使用 this.来访问自身的成员
        r.c1 = r_c1;         --this 也可以省略
        r.c2 = r_c2;
        return this;         --使用 return this  返回本对象
end;
end;
/
```

13.3 删除类

类的删除分为两种方式：一是类头的删除，删除类头则会顺带将类体一起删除；另外一种是类体的删除，这种方式只能删除类体，类头依然存在。

13.3.1 删除类头

类的删除通过 DROP CLASS 完成，即类头的删除。删除类头的同时会一并删除类体。

语法格式

```
DROP CLASS [<模式名>.]<类名>;
```

使用说明

1．如果被删除的类不属于当前模式，必须在语句中指明模式名；

2．如果一个类的声明被删除，那么对应的类体被自动删除。

权限

执行该操作的用户必须是该类的拥有者，或者具有 DBA 权限。

13.3.2 删除类体

从数据库中删除一个类的实现主体对象。

语法格式

```
DROP CLASS BODY [<模式名>.]<类名>;
```

使用说明

如果被删除的类不属于当前模式，必须在语句中指明模式名。

权限

执行该操作的用户必须是该类的拥有者，或者具有 DBA 权限。

13.4 类的使用

类类型同普通的数据类型一样，可以作为表中列的数据类型，PLSQL 语句块中变量的数据类型或过程及函数参数的数据类型。

13.4.1 具体使用规则

1．作为表中列类型或其他类成员变量属性的类不能被修改或删除

类中定义的数据类型，其名称只在类的声明及实现中有效。如果类内的函数的参数或返回值是类内的数据类型，或是进行类内成员变量的复制，需要在 PLSQL 中定义一个结构与之相同的类型。

根据类使用方式的不同，对象可分为变量对象及列对象。变量对象指的是在 PLSQL 语句块中声明的类类型的变量；列对象指的是在表中类类型的列。变量对象可以修改其属性的值而列对象不能。

2．变量对象的实例化

类的实例化通过 NEW 表达式调用构造函数完成。

3．变量对象的引用

通过‘=’进行的类类型变量之间的赋值所进行的时对象的引用，并没有复制一个新的对象。

4．变量对象属性访问

可以通过如下方式进行属性的访问。

```
<对象名>.<属性名>
```

5．变量对象成员方法调用

成员方法的调用通过以下方式调用：

```
<对象名>.<成员方法名>(<参数>{,<参数>})
```

如果函数内修改了对象内属性的值，则该修改生效。

6．列对象的插入

列对象的创建是通过 INSERT 语句向表中插入数据完成，插入语句中的值是变量对象，插入后存储在表中的数据即为列对象。

7．列对象的复制

存储在表中的对象不允许对对象中成员变量的修改，通过 into 查询或’=’进行的列到变量的赋值所进行的是对象的赋值，生成了一个与列对象数据一样的副本，在该副本上进行的修改不会影响表中列对象的值。

8．列对象的属性访问

通过如下方式进行属性的访问：

```
<列名>.<属性名>
```

9．列对象的方法调用

```
<列名>.<成员方法名>(<参数>{,<参数>})
```

列对象方法调用过程中对类型内属性的修改，都是在列对象的副本上进行的，不会影响列对象的值。

10．对象表的更新

表中存储的列对象虽然不能进行修改，但是可以通过 update 语句直接更新某行数据，更新所进行的并不是修改对象内属性的值而是直接替换了对象。

13.4.2 应用实例

类类型的变量对象、列对象二种使用示例如下：

1. 变量对象的应用实例如下：

```
declare
        type ex_rec_t is    record (a int, b    int);    --使用一个同结构的类型代替类定义的类型
        rec     ex_rec_t;
        o1      mycls;
        o2       mycls;
begin
        o1 = new mycls(1,2,3);
        o2 = o1;                    --对象引用
        rec = o2.r;                 --变量对象的成员变量访问
        print rec.a; print rec.b;
        rec = o1.f1(4,5);  --成员函数调用
        print rec.a; print rec.b;
        print    o1.id;       --成员变量访问
end;
```

2. 列对象的应用实例如下：

表的创建。

```
Create table tt1(c1 int, c2 mycls);
```

列对象的创建--插入数据。

```
Insert into tt1 values(1, mycls(1,2,3));
```

列对象的复制及访问。

```
Declare
        o mycls;
        id int;
begin
       select top 1 c2 into o from tt1;              --列对象的复制
       select top 1 c2.id into id from tt1;          --列对象成员的访问
end;
```

第 14 章 触发器

DM 是一个具有主动特征的数据库管理系统，其主动特征包括约束机制和触发器机制。通过触发器机制，用户可以定义、删除和修改触发器。DM 自动管理和运行这些触发器，从而体现系统的主动性，方便用户使用。

触发器(TRIGGER)定义为当某些与数据库有关的事件发生时，数据库应该采取的操作。这些事件包括全局对象、数据库下某个模式、模式下某个基表上的 INSERT、DELETE 和 UPDATE 操作。触发器与存储模块类似，都是在服务器上保存并执行的一段 DMPL/SQL 语句。不同的是：存储模块必须被显式地调用执行，而触发器是在相关的事件发生时由服务器自动地隐式地激发。触发器是激发它们的语句的一个组成部分，即直到一个语句激发的所有触发器执行完成之后该语句才结束，而其中任何一个触发器执行的失败都将导致该语句的失败，触发器所做的任何工作都属于激发该触发器的语句。

触发器为用户提供了一种自己扩展数据库功能的方法。关于触发器应用的例子有：

1. 利用触发器实现表约束机制(如：PRIMARY KEY、FOREIGN KEY、CHECK 等)无法实现的复杂的引用完整性；
2. 利用触发器实现复杂的事务规则(如：想确保薪水增加量不超过 25%)；
3. 利用触发器维护复杂的缺省值(如：条件缺省)；
4. 利用触发器实现复杂的审计功能；
5. 利用触发器防止非法的操作。

触发器是应用程序分割技术的一个基本组成部分，它将事务规则从应用程序的代码中移到数据库中，从而可确保加强这些事务规则并提高它们的性能。

在本章各例中，如不特别说明，各例均使用实例库 BOOKSHOP，用户均为建表者 SYSDBA。

14.1 触发器的定义

触发器分为表触发器和事件触发器。表触发器是对表里数据操作引发的数据库的触发，事件触发器是对数据库对象操作引起的数据库的触发。另外，时间触发器是一种特殊的事件触发器。

1. 表触发器

用户可使用触发器定义语句(CREATE TRIGGER)在一张基表上创建触发器。下面是触发器定义语句的语法：

语法格式

```
CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名> [WITH  ENCRYPTION]
<触发限制描述> [REFERENCING OLD [ROW] [AS] <引用变量名> | NEW [ROW] [AS] <引用变量名>
| OLD [ROW] [AS] <引用变量名> NEW [ROW] [AS] <引用变量名>]
[FOR EACH {ROW | STATEMENT}][WHEN <条件表达式>]<触发器体>
<触发限制描述>::=<触发限制描述 1> | <触发限制描述 2>
<触发限制描述 1>::= {BEFORE | AFTER <触发事件> {OR <触发事件>} ON <触发表名>
<触发限制描述 2>::= INSTEAD OF <触发事件> {OR <触发事件>} ON <触发视图名>
<触发表名>::=[<模式名>.]<基表名>
```

参数

1．<触发器名>　　指明被创建的触发器的名称；

2．BEFORE　　指明触发器在执行触发语句之前激发；

3．AFTER　　指明触发器在执行触发语句之后激发；

4．INSTEAD OF　　指明触发器执行时替换原始操作；

5．<触发事件>　　指明激发触发器的事件，可以是 INSERT、DELETE 或 UPDATE，其中 UPDATE 事件可通过 UPDATE OF <触发列清单>的形式来指定所修改的列；

6．<基表名>　　指明被创建触发器的基表的名称；

7．WITH　ENCRYPTION 选项，指定是否对触发器定义进行加密；

8．REFERENCING 子句　指明相关名称可以在元组级触发器的触发器体和 WHEN 子句中利用相关名称来访问当前行的新值或旧值，缺省的相关名称为 OLD 和 NEW；

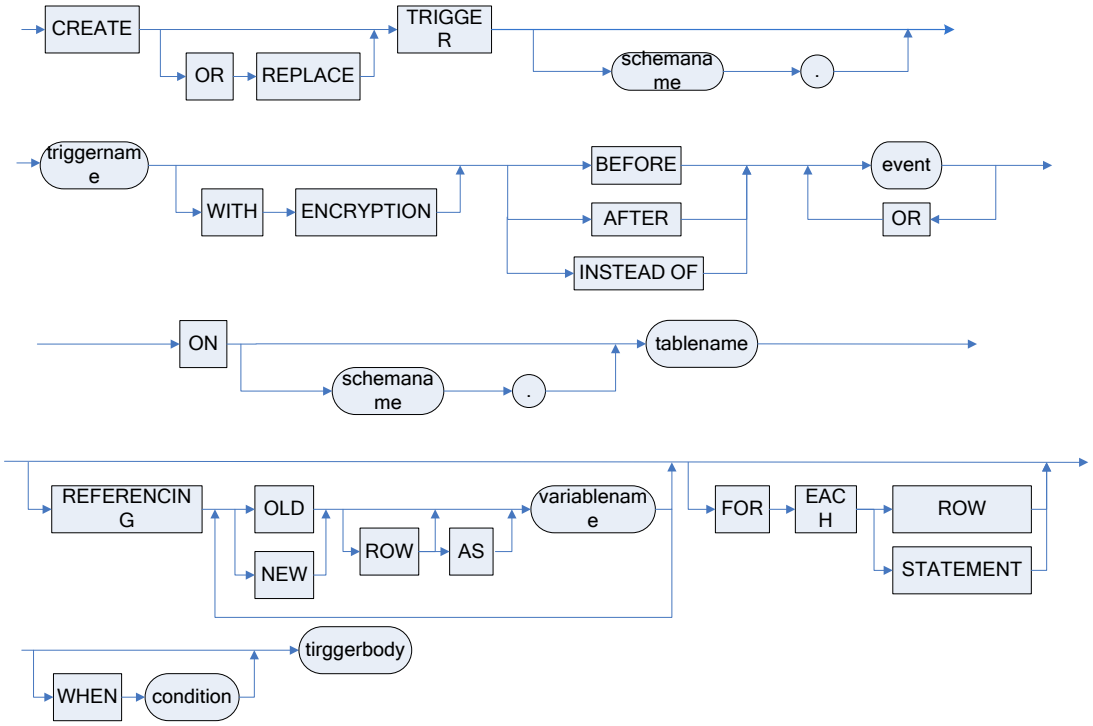
9．<引用变量名>　　标识符，指明行的新值或旧值的相关名称；

10．FOR EACH 子句　指明触发器为元组级或语句级触发器。FOR EACH ROW 表示为元组级触发器，它受被触发命令影响、且 WHEN 子句的表达式计算为真的每条记录激发一次。FOR EACH STATEMENT 为语句级触发器，它对每个触发命令执行一次。FOR EACH 子句缺省则为语句级触发器；

11．WHEN 子句　　只允许为元组级触发器指定 WHEN 子句，它包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；

12．<触发器体>　　触发器被触发时执行的 SQL 过程语句块。

图例



功能

创建触发器，并使其处于允许状态。

使用说明

1．<触发器名>是触发器的名称，它不能与模式内的其他模式级对象同名；

2．可以使用 OR REPLACE 选项来替换一个触发器，但是要注意被替换的触发器的触发表不能改变。如果要在同一模式内不同的表上重新创建一个同名的触发器，则必须先删除

该触发器，然后再创建；

3．<触发事件子句>说明激发触发器的事件；<触发器体>是触发器的执行代码；<引用子句>用来引用正处于修改状态下的行中的数据。如果指定了<触发条件>子句，则首先对该条件表达式求值，<触发器体>只有在该条件为真值时才运行。<触发器体>是一个 DMPL/SQL 语句块，它与存储模块定义语句中<模块体>的语法基本相同；

4．在一张基表上允许创建的表触发器的个数没有限制，一共允许有 12 种类型。它们分别是：BEFORE INSERT 行级、BEFORE INSERT 语句级、AFTER INSERT 行级、AFTER INSERT 语句级、BEFORE UPDATE 行级、BEFORE UPDATE 语句级、AFTER UPDATE 行级、AFTER UPDATE 语句级、BEFORE DELETE 行级、BEFORE DELETE 语句级、AFTER DELETE 行级和 AFTER DELETE 语句级；

5．触发器是在 DML 语句运行时激发的。执行 DML 语句的算法步骤如下：

1) 如果有语句级前触发器的话，先运行该触发器；
2) 对于受语句影响每一行：
   a) 如果有行级前触发器的话，运行该触发器；
   b) 执行该语句本身；
   c) 如果有行级后触发器的话，运行该触发器。
3) 如果有语句级后触发器的话，运行该触发器。

6．INSTEAD OF 触发器仅允许建立在视图上，并且只支持行级触发。

2. 事件触发器

用户可使用触发器定义语句(CREATE TRIGGER)在数据库全局对象上创建触发器。下面是触发器定义语句的语法：

语法格式

```
CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名> [WITH  ENCRYPTION]
BEFORE| AFTER <触发事件子句> ON <触发对象名>[WHEN <条件表达式>]<触发器体>
<触发事件子句>:=<DDL 事件子句>| <系统事件子句>
<DDL 事件子句>:=<DDL 事件>{OR <DDL 事件>}
<DDL 事件>:=<CREATE> | <ALTER> | <DROP>|<GRANT> | <REVOKE>|<TRUNCATE>
<系统事件子句>:=<系统事件>{OR <系统事件>}
<系统事件>:= <LOGIN> | <LOGOUT> | <SERERR>|<BACKUP DATABASE>
| <RESTORE DATABASE>|<AUDIT>|<NOAUDIT>|<TIMER>
<触发对象名>:=[<模式名>.]{SCHEMA|DATABASE>}
```

参数

1．<模式名>　指明被创建的触发器的所在的模式名称或触发事件发生的对象所在的模式名，缺省为当前模式；

2．<触发器名> 指明被创建的触发器的名称；

3．BEFORE　指明触发器在执行触发语句之前激发；

4．AFTER　指明触发器在执行触发语句之后激发；

5．<DDL 触发事件子句> 指明激发触发器的 DDL 事件，可以是 CREATE、ALTER、DROP、GRANT、REVOKE、TRUNCATE 等；

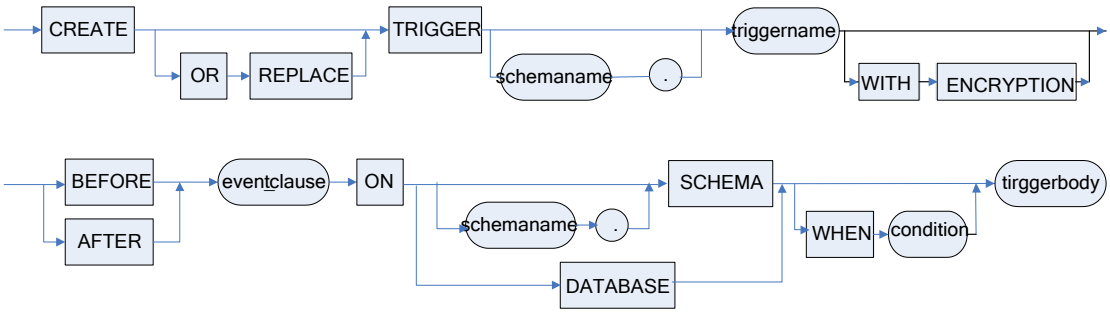
6．<系统事件子句> LOGIN、LOGOUT、SERERR、BACKUP DATABASE、RESTORE DATABASE、AUDIT、NOAUDIT、TIMER；

7．WITH　ENCRYPTION 选项，指定是否对触发器定义进行加密；

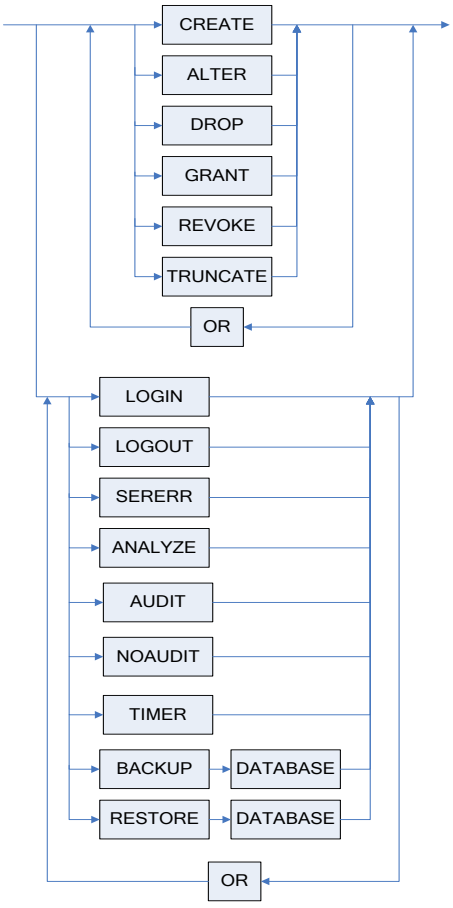
8．WHEN 子句　只允许为元组级触发器指定 WHEN 子句，它包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；

9．<触发器体>　触发器被触发时执行的 SQL 过程语句块。

图例



event_clause



使用说明

1．<触发器名>是触发器的名称，它不能与模式内的其他模式级对象同名；

2．可以使用 OR REPLACE 选项来替换一个触发器，但是要注意被替换的触发器的触发对象名不能改变。如果要在模式中不同的对象上重新创建一个同名的触发器，则必须先删除该触发器，然后再创建；

3．<触发事件子句>说明激发触发器的事件，DDL 事件以及系统事件。DDL 事件包括数据库和模式上的 DDL 操作；系统事件包括数据库上的除 DDL 操作以外系统事件；以上事件可以有多个，用 OR 列出。DDLEVENT 即只要执行了任意 DDL 语句时就激发触发器；

SYSEVENT 即只要执行了任意系统事件语句时就激发触发器。触发事件按照兼容性可以分为以下几个集合：

{CREATE, ALTER, DROP, TRUNCATE }、{ GRANT, REVOKE }、{ LOGIN, LOGOUT }、{ SERERR }、{ BACKUP DATABASE, RESTORE DATABASE }、{AUDIT, NOAUDIT}、{TIMER}。

只有同一个集合中的事件，才能在创建语句中并列出现。

4. <触发对象名>是触发事件发生的对象，DATABASE 和<模式名>只对 DDL 事件有效，<模式名>可以缺省；

5. 在一个数据库或模式上创建的事件触发器个数没有限制，可以有以下类型：CREATE、ALTER、DROP、GRANT、REVOKE、TRUNCATE、LOGIN、LOGOUT、SERERR、BACKUP DATABASE，且仅表示指该类操作，不涉及到具体数据库对象如 CREATE/ALTER/DROP TABLE，只要能引起任何数据字典表中的数据对象变化，都可以激发相应触发器，触发时间分为 BEFORE 和 AFTER；所有 DDL 事件触发器都可以设置 BEFORE 或 AFTER 的触发时机，但系统事件中 LOGOUT 仅能设置为 BEFORE，而其它则只能设置为 AFTER。模式级触发器不能是 LOGIN、LOGOUT、SERERR、BACKUP DATABASE 和 RESTORE DATABASE 事件触发器。

6．通过系统存储过程 SP_ENABLE_EVT_TRIGGER 和 SP_ENABLE_ALL_EVT_TRIGGER 可以禁用/启用指定的事件触发器或所有的事件触发器。

7．事件操作说明如下：

对于事件触发器，所有的事件信息都通过伪变量 :EVENTINFO 来取得。

下面对每种事件可以获得的信息进行详细说明：

1) CREATE：添加新的数据库对象(包括用户、基表、视图等)到数据字典时触发；

对象类型描述：:eventinfo.objecttype

指明事件对象的类型，类型为 VARCHAR(128)，对于不同的类型其值如下：

用户： ‘USER’

表： ‘TABLE’

视图： ‘VIEW’

索引： ‘INDEX’

过程： ‘PROCEDURE’

函数： ‘FUNCTION’

角色： ‘ROLE’

模式： ‘SCHEMA’

序列： ‘SEQUENCE’

触发器：‘TRIGGER’

同义词：‘SYNONYM’

包：‘PACKAGE’

类：‘CLASS’

类型：‘TYPE’

包体：‘PACKAGEBODY’

类体：‘CLASSBODY’

类型体：‘TYPEBODY’

对象名称： :eventinfo.objectname

指明事件对象的名称，类型为 VARCHAR(128)

所属模式： :eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

所属数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

操作用户名： :eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(128)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

2) ALTER：只要ALTER修改了数据字典中的数据对象(包括用户、基表、视图等)，就激活触发器；

对象类型描述： :eventinfo.objecttype

指明事件对象的类型，类型为 CHAR(1)，对于不同的类型其值如下：

用户： ‘USER’

表： ‘TABLE’

视图： ‘VIEW’

索引： ‘INDEX’

过程： ‘PROCEDURE ’

函数： ‘FUNCTION’

触发器：’TRIGGER’

对象名称： :eventinfo.objectname

指明事件对象的名称，类型为 VARCHAR(128)

所属模式： :eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

所属数据库： :eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

操作用户名： :eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(128)

事件发生时间： :eventinfo.optime

指明事件发生的时间，类型为 DATETIME

3) DROP：从数据字典删除数据库对象(包括用户、登录、基表、视图等)时触发；

对象类型描述： :eventinfo.objecttype

指明事件对象的类型，类型为 VARCHAR(128)，对于不同的类型其值如下：

用户： ‘USER’

表： ‘TABLE’

视图： ‘VIEW’

索引： ‘INDEX’

过程： ‘PROCEDURE’

函数： ‘FUNCTION’

角色： ‘ROLE’

模式： ‘SCHEMA’

序列： ‘SEQUENCE’

触发器：’TRIGGER’

同义词：‘SYNONYM’

包：‘PACKAGE’

类：‘CLASS’

类型：‘TYPE’

对象名称：　　:eventinfo.objectname

指明事件对象的名称，类型为 VARCHAR(128)

所属模式：　　:eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

所属数据库：　:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(128)，针对不同类型的对象有可能为空。

操作用户名：　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(128)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

4) GRANT：执行GRANT命令时触发；

权限类型描述：:eventinfo.granttype，对于不同的类型其值如下：

对象权限：'OBJECT_PRIV'

系统权限：'SYSTEM_PRIV'

角色权限：'ROLE_PRIV'

指明授予权限的类型，类型为 varchar(256)

授予权限对象的用户名：:eventinfo.grantee

指明授予权限的对象用户，类型为 varchar(256)

所属数据库：　:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

操作用户名：　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

5) REVOKE：执行REVOKE命令时触发；

权限类型描述：:eventinfo.granttype，对于不同的类型其值如下：

对象权限：'OBJECT_PRIV'

系统权限：'SYSTEM_PRIV'

角色权限：'ROLE_PRIV'

指明回收权限的类型，类型为 varchar(256)

授予权限对象的用户名：:eventinfo.grantee

指明回收权限的对象用户，类型为 varchar(256)

操作用户名：　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(256)

所属数据库：　:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

6) TRUNCATE：执行TRUNCATE命令时触发；

对象名称：　　:eventinfo.objectname，对于不同的类型其值如下：

表：　'T'

指明事件对象的名称，类型为 VARCHAR(256)

所属模式：　　:eventinfo.schemaname

指明事件对象所属的模式名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

所属数据库：　　:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

操作用户名：　　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

7) LOGIN：登录时触发；

登录名：　　:eventinfo.loginname

指明登录时的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

8) LOGOUT：退出时触发；

登录名：　　:eventinfo.loginname

指明退出时的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

9) BACKUP DATABASE:备份数据库时触发；

备份的数据库：:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

备份名：　　:eventinfo.backuname

指明的备份名，类型为 VARCHAR(256)

操作用户名：　　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

10) RESTORE DATABASE:还原数据库时触发；

还原的数据库：　:eventinfo.databasename

指明事件对象所属的数据库名，类型为 VARCHAR(256)，针对不同类型的对象有可能为空。

还原的备份名：:eventinfo.backuname

指明的备份名，类型为 VARCHAR(256)

操作用户名：　　:eventinfo.opuser

指明事件对象所属的用户名，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

11) SERERR：只要服务器记录了错误消息就触发；

错误号：　　:eventinfo. ERRCODE

指明错误的错误号，类型为 INT

错误信息：　　:eventinfo.errmsg

指明错误的错误信息，类型为 VARCHAR(256)

事件发生时间：:eventinfo.optime

指明事件发生的时间，类型为 DATETIME

12) AUDIT：进行审计时触发（用于收集，处理审计信息）；

13) NOAUDIT：不审计时触发；

14）TIMER：定时触发。见下文时间触发器。

7．<触发器体>是触发器的执行代码，是一个 DMPL/SQL 语句块，语句块与存储模块定义语句中<模块体>的语法基本相同。有关详细语法，可参考第 10 章的相关部分。<引用子句>用来引用正处于修改状态下表中行的数据。如果指定了<触发条件>子句，则首先对该条件表达式求值，<触发器体>只有在该条件为真值时才运行；

8．创建模式触发器时，触发对象名直接用 SCHEMA；

9．创建的触发器可以分为以下几类：

1) 在自己拥有的模式中创建自己模式的对象上的触发器或创建自己模式上的触发器；
2) 在任意模式中创建任意模式的对象上的触发器或创建其他用户模式上(. SCHEMA)的触发器，即支持跨模式的触发器，表现为<触发器名>和<触发对象名>的<模式名>不同；
3) 创建数据库上(DATABASE)的触发器。

10．触发器的创建者必须拥有 CREATE TRIGGER 数据库权限并具有触发器定义种引用对象的权限；

11．DDL 触发事件的用户必须拥有对模式或数据库上相应对象的 DDL 权限；系统触发事件的用户必须有 DBA 权限；

12．如果触发器执行 SQL 语句或调用过程或函数，那么触发器的拥有者必须拥有执行这些操作所必需的权限。这些权限必须直接授予触发器拥有者，而不是通过角色授予，这与存储模块或函数的限制一致；

13．如果触发器同时也是触发事件对象，则该触发器不会被激发，例如：当删除触发器本身被删除时不会触发 DROP 触发器。

权限

用户必须是基表的拥有者，或者具有 DBA 权限。

需要强调的是，由于触发器是激发它们的语句的一个组成部分，为保证语句的原子性，在<触发器体>以及<触发器体>调用的存储模块中不允许使用可能导致事务提交或回滚的 SQL 语句。具体地说，在触发器中允许的 SQL 语句有： SELECT、INSERT、DELETE、UPDATE、DECLARE CURSOR、OPEN、FETCH、CLOSE 语句、COMMIT 和 ROLLBACK 等。

每张基表上的可创建的触发器的个数没有限制，但是触发器的个数越多，处理 DML 语句所需的时间就越长，这是显而易见的。注意，不存在触发器的执行权限，因为用户不能主动“调用”某个触发器，是否激发一个触发器是由系统来决定的。

事件触发器（DDL 触发事件）使用示例如下：

```
SQL>CREATE TABLE T01_TRI_10000(OBJECTTYPE VARCHAR(500),OBJECTNAME VARCHAR(500), SCHEMANAME VARCHAR(500),DATABASENAME VARCHAR(500),OPUSER VARCHAR(500), OPTIME VARCHAR(500));

SQL>CREATE TABLE T02_TRI_10000 (C1 INT,C2 VARCHAR(10));
SQL>INSERT INTO T02_TRI_10000 VALUES (1,'ABCD');
SQL>CREATE TRIGGER TRI01_TRI_10000 BEFORE CREATE ON DATABASE BEGIN INSERT INTO T01_TRI_10000
VALUES(:EVENTINFO.OBJECTTYPE,:EVENTINFO.OBJECTNAME,:EVENTINFO.SCHEMANAME,:EVENTINFO.DATABASENAME,:EVENTINFO.OPUSER, :EVENTINFO.OPTIME); END;
```

```
SQL>CREATE USER L01_TRI_10000 IDENTIFIED BY L01_TRI_10000;
SQL>CREATE TABLE T03_TRI_10000(C1 INT);
SQL>CREATE VIEW V01_TRI_10000 AS SELECT * FROM T01_TRI_10000;
SQL>CREATE INDEX I01_TRI_10000 ON T01_TRI_10000(OBJECTTYPE);
SQL>CREATE OR REPLACE PROCEDURE P01_TRI_10000 AS BEGIN SELECT * FROM T02_TRI_10000; END;

SQL>CREATE FUNCTION F01_TRI_10000 RETURN VARCHAR(30) AS A1 VARCHAR(30); BEGIN SELECT C2 INTO A1 FROM T02_TRI_10000 WHERE C1=1; PRINT A1; RETURN A1; END;
SQL>CREATE ROLE R01_TRI_10000;
SQL>CREATE SEQUENCE S01_TRI_10000 INCREMENT BY 10;
SQL>CREATE TRIGGER TRI02_TRI_10000 AFTER CREATE ON DATABASE BEGIN PRINT 'SUCCESS';END;

SQL>SELECT OBJECTTYPE, OBJECTNAME, SCHEMANAME, DATABASENAME, OPUSER FROM T01_TRI_10000;
```

查询结果如下：

```
行号        OBJECTTYPE          OBJECTNAME SCHEMANAME DATABASENAME OPUSER
----------  ------------------  ---------- ---------- ------------ ----------
1           USER                L01_TRI_10000 NULL       DAMENG       SYSDBA
2           TABLE               T03_TRI_10000 SYSDBA     DAMENG       SYSDBA
3           VIEW                V01_TRI_10000 SYSDBA     DAMENG       SYSDBA
4           INDEX               I01_TRI_10000 SYSDBA     DAMENG       SYSDBA
5           PROCEDURE           P01_TRI_10000 SYSDBA     DAMENG       SYSDBA
6           FUNCTION            F01_TRI_10000 SYSDBA     DAMENG       SYSDBA
7           ROLE                R01_TRI_10000 NULL       DAMENG       SYSDBA
8           SEQUENCE            S01_TRI_10000 SYSDBA     DAMENG       SYSDBA
9           TRIGGER             TRI02_TRI_10000 SYSDBA   DAMENG       SYSDBA
```

事件触发器（系统触发事件）使用示例如下：

```
create or replace trigger test_trigger after LOGIN on database begin print'SUCCESS'; end;
----只要一登录，服务器就会打印出 SUCCESS
```

3. 时间触发器

从 DM7 开始支持时间触发器，时间触发器属于一种特殊的事件触发器，它使得用户可以定义一些有规律性执行的、定点执行的任务，比如在晚上服务器负荷轻的时候通过时间触发器做一些更新统计信息的操作、自动备份操作等等，因此时间触发器是非常有用的。

语法格式

```
CREATE [OR REPLACE] TRIGGER [<模式名>.]<触发器名>[WITH  ENCRYPTION]
AFTER TIMER ON DATABASE
<{FOR ONCE AT DATETIME [时间表达式]}|{{<month_rate>|<week_rate>|<day_rate>}{once_in_day|times_in_day}{during_date}}>
[WHEN <条件表达式>]
<触发器体>
<month_rate>:= {FOR EACH <整型变量> MONTH {day_in_month}}| FOR EACH <整型变量> MONTH { day_in_month_week}}
```

```
<day_in_month>:= DAY <整型变量>
<day_in_month_week>:= {DAY <整型变量> OF WEEK<整型变量>}|{DAY <整型变量> OF WEEK LAST}
<week_rate>: =FOR EACH <整型变量> WEEK {day_of_week_list}
< day_of_week_list >:= {<整型变量>}|{, <整型变量>}
<day_rate>: =FOR EACH <整型变量> DAY
< once_in_day >:= AT TIME <时间表达式>
< times_in_day >: ={ duaring_time } FOR EACH <整型变量> MINUTE
<duaring_time>: ={NULL}|{FROM TIME <时间表达式>}|{FROM TIME <时间表达式> TO TIME <时间表达式>}
<duaring_date>: ={NULL}|{FROM DATETIME <日期时间表达式>}|{FROM DATETIME <日期时间表达式> TO DATETIME <日期时间表达式>}
```

参数

1．<模式名>　指明被创建的触发器的所在的模式名称或触发事件发生的对象所在的模式名，缺省为当前模式；

2．<触发器名> 指明被创建的触发器的名称；

3．WHEN 子句　包含一个布尔表达式，当表达式的值为 TRUE 时，执行触发器；否则，跳过该触发器；

4．<触发器体>　触发器被触发时执行的 SQL 过程语句块。

时间触发器的最低时间频率精确到分钟级，定义很灵活，完全可以实现数据库中的代理功能，只要通过定义一个相应的时间触发器即可。在触发器体中定义要做的工作，可以定义操作的包括执行一段 SQL 语句、执行数据库备份、执行重组 B 树、执行更新统计信息、执行数据迁移（DTS）。

下面的简单例子在屏幕上每隔一分钟输出一行“HELLO WORLD”。

```
CREATE OR REPLACE TRIGGER timer2
AFTER TIMER on database
for each 1 day for each 1 minute
BEGIN
    print 'HELLO WORLD';
END;
/
```

关闭时间触发器和普通触发器是一样的，这里不再叙述。

14.1.1 触发器类型

一个触发器的类型由该触发器的触发事件、级别及其执行的时间共同决定。事件触发器上文已介绍的很详尽，此处不再累述。本节详细介绍表触发器的分类。

1. 表触发器分类

1) INSERT、DELETE 和 UPDATE 触发器

激发触发器的触发事件可以是三种数据操作命令，即 INSERT、DELETE 和 UPDATE 操作。在触发器定义语句中用关键字 INSERT、DELETE 和 UPDATE 指明构成一个触发器事件的数据操作的类型，其中 UPDATE 触发器可能依赖于所修改的列，在定义中可通过 UPDATE OF <触发列清单>的形式来指定所修改的列，<触发列清单>指定的字段数不能超过

128 个。

在 PERSON.PERSON 上建立触发器。如下例所示：

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_UPD
AFTER UPDATE OF NAME,PHONE ON PERSON.PERSON
BEGIN
  PRINT 'UPDATE OPERATION ON COLUMNS NAME OR PHONE OF PERSON';
END;

SET SCHEMA SYSDBA;
```

当对表 PERSON 进行更新操作，并且更新的列中包括 NAME 或 PHONE 时，此例中定义的触发器 TRG_UPD 将被激发。

如果一个触发器的触发事件为 INSERT，则该触发器被称为 INSERT 触发器，同样也可以这样来定义 DELETE 触发器和 UPDATE 触发器。一个触发器的触发事件也可以是多个数据操作命令的组合，这时这个触发器可由多种数据操作命令激发。如下例所示：

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_INS_DEL
AFTER INSERT OR DELETE ON PERSON.PERSON
BEGIN
  PRINT 'INSERT OR DELETE OPERATION ON PERSON';
END;

SET SCHEMA SYSDBA;
```

此例中的触发器 TRG_INS_DEL 既是 INSERT 触发器又是 DELETE 触发器，对基表 T1 的 INSERT 和 DELETE 操作都会激发该触发器。

2) 元组级触发器和语句级触发器

根据触发器的级别可将触发器分为元组级触发器(也称行级触发器)和语句级触发器。

元组级触发器为触发命令所影响的每条记录激发一次。假如一个 DELETE 命令从表中删除了 1000 行记录，那么这个表上的元组级 DELETE 触发器将被执行 1000 次。元组级触发器常用于数据审计、完整性检查等应用中。元组级触发器是在触发器定义语句中通过 FOR EACH ROW 子句创建的。对于元组级触发器，可以用一个 WHEN 子句来限制针对当前记录是否执行该触发器。WHEN 子句包含一条布尔表达式，当它的值为 TRUE 时，执行触发器；否则，跳过该触发器。

语句级触发器对每个触发命令执行一次。例如，对于一条将 500 行记录插入表 TABLE_1 中的 INSERT 语句，这个表上的语句级 INSERT 触发器只执行一次。语句级触发器一般用于对表上执行的操作类型引入附加的安全措施。语句级触发器是在触发器定义语句中通过 FOR EACH STATEMENT 子句创建的，该子句可缺省。

以下分别是元组级触发器和语句级触发器的例子。

```
SET SCHEMA PERSON;

CREATE OR REPLACE TRIGGER TRG_DEL_ROW
```

```
BEFORE DELETE ON PERSON.PERSON
FOR EACH ROW                    -- 元组级：此子句一定不能省略
BEGIN
  PRINT 'DELETE' || :OLD.NAME|| ' ON PERSON';
END;

CREATE OR REPLACE TRIGGER TRG_INS_ST
AFTER INSERT ON PERSON.PERSON
FOR EACH STATEMENT      -- 语句级：此子句可省略
BEGIN
  PRINT 'AFTER INSERT ON PERSON';
END;

SET SCHEMA SYSDBA;
```

3) BEFORE 和 AFTER 触发器

触发器服务于触发事件，通过在触发器定义语句中指定 BEFORE 或 AFTER 关键字可以选择在触发事件之前还是之后运行触发器。显然，BEFORE 触发器在触发事件之前执行，而 AFTER 触发器在触发事件之后执行。

在元组级触发器中可以引用当前修改的记录在修改前后的值，修改前的值称为旧值，修改后的值称为新值。对于插入操作不存在旧值，而对于删除操作则不存在新值。

对于新、旧值的访问请求常常决定一个触发器是 BEFORE 类型还是 AFTER 类型。如果需要通过触发器对插入的行设置列值，那么为了能设置新值，需要使用一个 BEFORE 触发器，因为在 AFTER 触发器中不允许用户设置已插入的值。在审计应用中则经常使用 AFTER 触发器，因为元组修改成功后才有必要运行触发器，而成功地完成修改意味着成功地通过了该表的引用完整性约束。

以下是 BEFORE 触发器和 AFTER 触发器的举例。

BEFORE 触发器示例

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER TRG_INS_BEFORE
BEFORE INSERT ON OTHER.READER
FOR EACH ROW
BEGIN
  :NEW.READER_ID:=:NEW.READER_ID+1;
END;

SET SCHEMA SYSDBA;
```

该触发器在插入一条记录前，将记录中 COL1 列的值加 1。

```
CREATE TABLE T_TEMP(C1 INT,C2 CHAR(20));
```

新建表 T_TEMP。

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER TRG_INS_AFTER
AFTER INSERT ON OTHER.READER
```

```
FOR EACH ROW
BEGIN
   INSERT INTO SYSDBA.T_TEMP VALUES(:NEW.READER_ID, 'INSERT ON READER');
END;


SET SCHEMA SYSDBA;
```

该触发器在插入一条记录后，将插入的值以及操作类型记录到用于审计的表 T_TEMP 中。

2. 触发器分类总结

综合上文所述，在一张基表上所允许的可能的合法触发器类型共有 12 种，如表 14.1.1 所示：

表 14.1.1 表触发器类型

名　　称	功　　能
BEFORE INSERT	在一个 INSERT 处理前激发一次
BEFORE INSERT FOR EACH ROW	每条新记录插入前激发
AFTER INSERT	在一个 INSERT 处理后激发一次
AFTER INSERT FOR EACH ROW	每条新记录插入后激发
BEFORE DELETE	在一个 DELETE 处理前激发一次
BEFORE DELETE FOR EACH ROW	每条记录被删除前激发
AFTER DELETE	在一个 DELETE 处理后激发一次
AFTER DELETE FOR EACH ROW	每条记录被删除后激发
BEFORE UPDATE	在一个 UPDATE 处理前激发一次
BEFORE UPDATE FOR EACH ROW	每条记录被修改前激发
AFTER UPDATE	在一个 UPDATE 处理后激发一次
AFTER UPDATE FOR EACH ROW	每条记录被修改后激发

14.1.2 触发器激发顺序

下面是执行 DML 语句的算法步骤：

1．如果有语句级前触发器的话，先运行该触发器；

2．对于受语句影响每一行：

1) 如果有行级前触发器的话，运行该触发器；

2) 执行该语句本身；

3) 如果有行级后触发器的话，运行该触发器。

3．如果有语句级后触发器的话，运行该触发器。

为了说明上面的算法，假设我们用 OTHER.READER 表为例，并在其上创建了所有四种 UPDATE 触发器，即之前、之后、行级前和行级后。其代码如下：

```
SET SCHEMA OTHER;


CREATE OR REPLACE TRIGGER Reader_Before_St
BEFORE UPDATE ON OTHER.READER
BEGIN
```

```
    PRINT 'BEFORE UPDATE TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER Reader_After_St
AFTER UPDATE ON OTHER.READER
BEGIN
    PRINT 'AFTER UPDATE TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER    Reader_Before_Row
BEFORE UPDATE ON OTHER.READER
FOR EACH ROW
BEGIN
    PRINT 'BEFORE UPDATE EACH ROW TRIGGER FIRED';
END;

CREATE OR REPLACE TRIGGER    Reader_After_Row
AFTER UPDATE ON OTHER.READER
FOR EACH ROW
BEGIN
    PRINT 'AFTER UPDATE EACH ROW TRIGGER FIRED';
END;

SET SCHEMA SYSDBA;
```

现在，执行更新语句：

```
UPDATE OTHER.READER SET AGE=AGE+1;
```

该语句对三行有影响。语句级前触发器和语句级后触发器将各自运行一次，而行级前触发器和行级后触发器则各运行三次。因此，服务器返回的打印消息应为：

```
BEFORE UPDATE TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
BEFORE UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE EACH ROW TRIGGER FIRED
AFTER UPDATE TRIGGER FIRED
```

同类触发器的激发顺序没有明确的定义。如果顺序非常重要的话，应该把所有的操作组合在一个触发器中。

14.1.3 新、旧行值的引用

前面曾经提到，在元组级触发器内部，可以访问正在处理中的记录的数据，这种访问是通过两个引用变量:OLD 和:NEW 实现的。:OLD 表示记录被处理前的值，:NEW 表示记录被处理后的值，标识符前面的冒号说明它们是宿主变量意义上的连接变量，而不是一般的

DMPL/SQL 变量。我们还可以通过引用子句为这两个行值重新命名。

引用变量与其它变量不在同一个命名空间，所以变量可以与引用变量同名。在触发器体中使用引用变量时，必须采用下列形式：

```
:引用变量名.列名
```

其中，列名必须是触发表中存在的列，否则编译器将报错。

下表总结了标识符:OLD 和:NEW 的含义。

表 14.1.2 标识符:OLD 和:NEW 的含义

触发语句	标识符:OLD	标识符:NEW
INSERT	无定义，所有字段都为 NULL	该语句结束时将插入的值
UPDATE	更新前行的旧值	该语句结束时将更新的值
DELETE	行删除前的旧值	无定义，所有字段都为 NULL

:OLD 引用变量只能读取，不能赋值(因为设置这个值是没有任何意义的)；而:NEW 引用变量则既可读取，又可赋值(当然必须在 BEFORE 类型的触发器中，因为数据操作完成后再设置这个值也是没有意义的)。通过修改:NEW 引用变量的值，我们可以影响插入或修改的数据。

:NEW 行中使用的字段数不能超过 255 个，:NEW 行与:OLD 行中使用的不同字段总数不能超过 255 个。

注意：对于 INSERT 操作，引用变量:OLD 无意义；而对于 DELETE 操作，引用变量:NEW 无意义。如果在 INSERT 触发器体中引用:OLD，或者在 DELETE 触发器体中引用:NEW，不会产生编译错误。但是在执行时，对于 INSERT 操作，:OLD 引用变量的值为空值；对于 DELETE 操作，:NEW 引用变量的值为空值，且不允许被赋值。

例 1 下例中触发器 GenerateValue 使用了:OLD 引用变量。该触发器是一个 UPDATE 前触发器，其目的是做更新操作时不论是否更新表中某列的值，改列的值保持原值不变。这里使用 PRODUCTION.PRODUCT 表为例。

```
SET SCHEMA PRODUCTION;

CREATE OR REPLACE TRIGGER GenerateValue
BEFORE UPDATE ON PRODUCTION.PRODUCT
FOR EACH ROW
BEGIN
:new.NOWPRICE:=:old.NOWPRICE;
END;

SET SCHEMA SYSDBA;
```

当执行一个 UPDATE 语句时，无论用户是否给定 NOWPRICE 字段的值，触发器都将自动保持原来的 NOWPRICE 值不变。例如，用户查询 PRODUCTID=1 的一行数据的 NOWPRICE 值。

```
SELECT NOWPRICE FROM PRODUCTION.PRODUCT WHERE PRODUCTID=1;
```

可得到结果为 NOWPRICE=15.2000；

用户可以执行如下所示的 UPDATE 语句。

```
UPDATE PRODUCTION.PRODUCT SET NOWPRICE =1.0000 WHERE PRODUCTID=1;
```

再次执行查询语句。

```
SELECT NOWPRICE FROM PRODUCTION.PRODUCT WHERE PRODUCTID=1;
```

可发现结果仍然为 NOWPRICE=15.2000，而非修改的 1.0000。

例 2 下例中触发器 GenerateValue 使用了:NEW 引用变量。该触发器是一个 INSERT 前触发器，其目的是自动生成一些字段值。这里使用 PRODUCTION.PRODUCT 表为例。

```
SET SCHEMA PRODUCTION;

CREATE OR REPLACE TRIGGER GenerateValue
BEFORE INSERT ON PRODUCTION.PRODUCT
FOR EACH ROW
BEGIN
:new.NOWPRICE:=:new.ORIGINALPRICE*:new.DISCOUNT;
END;

SET SCHEMA SYSDBA;
```

触发器 GenerateValue 实际上是修改引用变量:NEW 的值，这就是:NEW 引用变量的用途之一。当执行一个 INSERT 语句时，无论用户是否给定 NOWPRICE 字段的值，触发器都将自动利用 ORIGINALPRICE 字段和 DISCOUNT 字段来计算 NOWPRICE。例如，用户可以执行如下所示的 INSERT 语句。

```
INSERT INTO PRODUCTION.PRODUCT
(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,
SATETYSTOCKLEVEL,ORIGINALPRICE,DISCOUNT,SELLSTARTTIME)
VALUES('老人与海','海明威','上海出版社','2006-8-1','9787532740088',1,'10','100','7.5','2008-1-10');
```

即使为 NOWPRICE 字段指定了值，该值也会被忽略，因为触发器将改变该值。

```
INSERT INTO PRODUCTION.PRODUCT
(NAME,AUTHOR,PUBLISHER,PUBLISHTIME,PRODUCTNO,PRODUCT_SUBCATEGORYID,
SATETYSTOCKLEVEL,ORIGINALPRICE,DISCOUNT,NOWPRICE,SELLSTARTTIME)
VALUES('老人与海','海明威','上海出版社','2006-8-1','9787532740089',1,'10','100','7.5','88','2008-1-10');
```

新插入元组的 NOWPRICE 字段将被触发器修改为 750.0000，而不是语句中的 88。

14.1.4 触发器谓词

如前面介绍的，触发事件可以是多个数据操作的组合，即一个触发器可能既是 INSERT 触发器，又是 DELETE 或 UPDATE 触发器。当一个触发器可以为多个 DML 语句触发时，在这种触发器体内部可以使用三个谓词：INSERTING、DELETING 和 UPDATING 来确定当前执行的是何种操作。这三个谓词的含义如下表所示。

表 14.1.3 触发器谓词

谓 词	状 态
INSERTING	当触发语句为 INSERT 时为真，否则为假
DELETING	当触发语句为 DELETE 时为真，否则为假
UPDATING[(<列名>)]	未指定列名时，当触发语句为 UPDATE 时为真，否则为假；指定某一列名时，当触发语句为对该列的 UPDATE 时为真，否则为假

虽然在其他 DMPL/SQL 语句块中也可以使用这三个谓词，但这时它们的值都为假。

下例中的触发器 LogChanges 使用这三个谓词来记录表 OTHER.READER 发生的所有变化。除了记录这些信息外，它还记录对表进行变更的用户名。该触发器的记录存放在表

OTHER.READERAUDIT 中。

触发器 LogChanges 的创建语句如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER LogChanges
AFTER INSERT OR DELETE OR UPDATE ON OTHER.READER
FOR EACH ROW
DECLARE
v_ChangeType CHAR(1);
BEGIN
/* ‘I’表示 INSERT 操作，’D’表示 DELETE 操作，’U’表示 UPDATE 操作 */
IF INSERTING THEN
v_ChangeType := 'I';
ELSIF UPDATING THEN
v_ChangeType := 'U';
ELSE
v_ChangeType := 'D';
END IF;
/* 记录对 Reader 做的所有修改到表 ReaderAudit 中，包括修改人和修改时间 */
INSERT INTO OTHER.READERAUDIT
VALUES
(v_ChangeType, USER, SYSDATE,
:old.reader_id, :old.name, :old.age, :old.gender, :old.major,
:new.reader_id, :new.name, :new.age, :new.gender, :new.major);
END;

SET SCHEMA SYSDBA;
```

14.1.5 设计触发器的原则

在应用中使用触发器功能时，应遵循以下设计原则，以确保程序的正确和高效：

1．如果希望保证一个操作能引起一系列相关动作的执行，请使用触发器；

2．不要用触发器来重复实现 DM 中已有的功能。例如，如果用约束机制能完成希望的完整性检查，就不要使用触发器；

3．避免递归触发。所谓递归触发，就是触发器体内的语句又会激发该触发器，导致语句的执行无法终止。例如，在表 T1 上创建 BEFORE UPDATE 触发器，而该触发器中又有对表 T1 的 UPDATE 语句；

4．合理地控制触发器的大小和数目。要知道，一旦触发器被创建，任何用户在任何时间执行的相应操作都会导致触发器的执行，这将是一笔不小的开销。

14.2 触发器的删除

当用户需要从数据库中删除一个触发器时，可以使用触发器删除语句。其语法如下：

语法格式

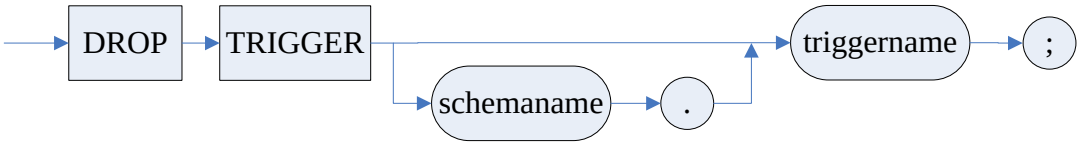
```
DROP TRIGGER [<模式名>.]<触发器名>;
```

参数

1．<模式名>　　指明被删除触发器所属的模式；

2．<触发器名>　　指明被删除的触发器的名字。

图例



使用说明

1．当触发器的触发表被删除时，表上的触发器将被自动地删除；

2．除了 DBA 用户外，其他用户必须是该触发器所属基表的拥有者才能删除触发器。

权限

执行该操作的用户必须是该触发器所属基表的拥有者，或者具有 DBA 权限。

举例说明

例 1 删除触发器 TRG1。

```
DROP TRIGGER TRG1;
```

例 2 删除模式 SYSDBA 下的触发器 TRG2。

```
DROP TRIGGER SYSDBA.TRG2;
```

14.3 禁止和允许触发器

每个触发器创建成功后都自动处于允许状态(ENABLE)，只要基表被修改，触发器就会被激发。但是在某些情况下，例如：

1．触发器体内引用的某个对象暂时不可用；

2．载入大量数据时，希望屏蔽触发器以提高执行速度；

3．重新载入数据。

用户可能希望触发器暂时不被触发，但是又不想删除这个触发器。这时，可将其设置为禁止状态(DISABLE)。

当触发器处于允许状态时，只要执行相应的 DML 语句，且触发条件计算为真，触发器体的代码就会被执行；当触发器处于禁止状态时，则在任何情况下触发器都不会被激发。根据不同的应用需要，用户可以使用触发器修改语句将触发器的状态设置为允许或禁止状态。其语法如下：

语法格式

```
ALTER TRIGGER [<模式名>.]<触发器名> DISABLE | ENABLE;
```

参数

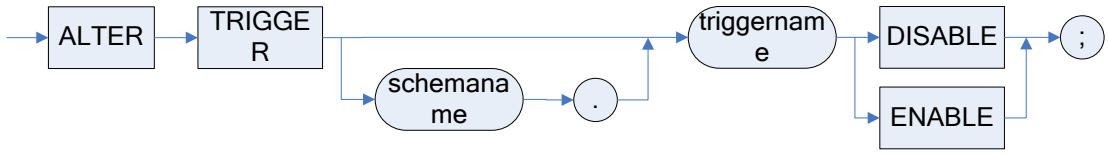
1．<模式名>　　指明被修改的触发器所属的模式；

2．<触发器名>　　指明被修改的触发器的名字；

3．DISABLE　　指明将触发器设置为禁止状态。当触发器处于禁止状态时，在任何情况下触发器都不会被激发；

4．ENABLE　　指明将触发器设置为允许状态。当触发器处于允许状态时，只要执行相应的 DML 语句，且触发条件计算为真，触发器就会被激发。

图例



14.4 触发器的重编

重新对触发器进行编译，如果重新编译失败，则将触发器置为禁止状态。

重编功能主要用于检验触发器的正确性。

语法格式

```
ALTER TRIGGER [<模式名>.]<触发器名> COMPILE [DEBUG];
```

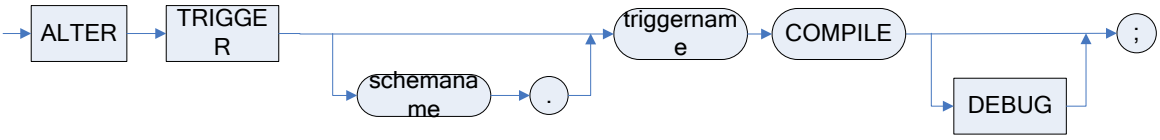
参数

1．<模式名>　　指明被修改的触发器所属的模式；

2．<触发器名>　　指明被修改的触发器的名字；

3．[DEBUG]　　可忽略。

图例



权限

执行该操作的用户必须是触发器的创建者，或者具有 DBA 权限。

举例说明

例　重新编译触发器

```
ALTER TRIGGER OTHER.TRG_AI_ACCOUNT COMPILE;
```

14.5 触发器应用举例

正如我们在本章所介绍的，触发器是 DM 系统提供的重要机制。我们可以使用该机制来加强比正常的审计机制、完整性约束机制、安全机制等所能提供的功能更复杂的事务规则。为帮助用户更好地使用该机制，我们提供了一些触发器应用的例子供用户参考。

14.5.1 使用触发器实现审计功能

尽管 DM 系统本身已经提供了审计机制，但是在许多情况下我们还是可以利用触发器完成条件更加复杂的审计。与内置的审计机制相比，采用触发器实现的审计有如下优点：

1．使用触发器可针对更复杂的条件进行审计；

2．使用触发器不仅可以记录操作语句本身的信息，还可以记录被该语句修改的数据的具体值；

3．内置的审计机制将所有审计信息集中存放，而触发器实现的审计可针对不同的操作对象分别存放审计信息，便于分析。

虽然如此，触发器并不能取代内置的审计机制。因为内置审计机制的某些功能触发器是无法做到的。例如：

1. 内置审计机制可审计的类型更多。触发器只能审计表上的 DML 操作，而内置审计机制可以针对各种操作、对象和用户进行审计；
2. 触发器只能审计成功的操作，而内置审计机制能审计失败的操作；
3. 内置审计机制使用起来更简单，并且其正确性更有保障。

用于审计的触发器通常都是 AFTER 类型。关于审计的实例，请参考第 14.1.4 节的例子，其中的触发器 LogChanges 就是一个典型的审计触发器。

14.5.2 使用触发器维护数据完整性

触发器与完整性约束机制都可以用于维护数据的完整性，但是二者之间存在着显著的区别。一般情况下，如果使用完整性约束机制可以完成约束检查，我们不建议用户使用触发器。这是因为：

1．完整性约束机制能保证表上所有数据符合约束，即使是约束创建前存在的数据也必须如此；而触发器只保证其创建后的数据满足约束，但之前存在数据的完整性则得不到保证；

2．完整性约束机制使用起来更简单，并且其正确性更有保障。

触发器通常用来实现完整性约束机制无法完成的约束检查和维护，例如：

1．引用完整性维护

删除被引用表中的数据时，级联删除引用表中引用该数据的记录；更新被引用表中的数据时，更新引用表中引用该数据的记录的相应字段。下例中，表 OTHER.DEPTTAB 为被引用表，其主关键字为 Deptno；表 OTHER.EMPTAB 为引用表。其结构如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Dept_del_upd_cascade
AFTER DELETE OR UPDATE ON OTHER.DEPTTAB FOR EACH ROW
BEGIN
    IF DELETING THEN
        DELETE FROM OTHER.EMPTAB
            WHERE Deptno = :old.Deptno;
    ELSE
        UPDATE OTHER.EMPTAB SET Deptno = :new.Deptno
            WHERE Deptno = :old.Deptno;
    END IF;
END;

SET SCHEMA SYSDBA;
```

2．CHECK 规则检查

增加新员工或者调整员工工资时，保证其工资不超过规定的范围，并且涨幅不超过 25%。该例中，表 OTHER.EMPTAB 记录员工信息；表 OTHER.SALGRADE 记录各个工种的工资范围，其结构如下：

```
SET SCHEMA OTHER;
```

```
CREATE OR REPLACE TRIGGER Salary_check
BEFORE INSERT OR UPDATE ON OTHER.EMPTAB
FOR EACH ROW
DECLARE
    Minsal        FLOAT;
    Maxsal        FLOAT;
    Salary_out_of_range      EXCEPTION FOR -20002;
BEGIN
/* 取该员工所属工种的工资范围 */
SELECT Losal, Hisal INTO Minsal, Maxsal FROM OTHER.SALGRADE
WHERE Job_classification = :new.Job;
/* 如果工资超出工资范围，报告异常 */
IF (:new.Sal < Minsal OR :new.Sal > Maxsal) THEN
        RAISE Salary_out_of_range;
END IF;
/* 如果工资涨幅超出 25%，报告异常 */
IF UPDATING AND (:new.Sal - :old.Sal) / :old.Sal > 0.25 THEN
        RAISE Salary_out_of_range;
END IF;
END;

SET SCHEMA SYSDBA;
```

14.5.3 使用触发器保障数据安全性

在复杂的条件下，可以使用触发器来保障数据的安全性。同样，要注意不要用触发器来实现 DM 安全机制已提供的功能。使用触发器进行安全控制时，应使用语句级 BEFORE 类型的触发器，其优点如下：

1．在执行触发事件之前进行安全检查，可以避免系统在触发语句不能通过安全检查的情况下做不必要的工作；

2．使用语句级触发器，安全检查只需要对每个触发语句进行一次，而不必对语句影响的每一行都执行一次。

下面这个例子显示如何用触发器禁止在非工作时间内修改表 OTHER.EMPTAB 中的工资(Sal)栏。非工作时间包括周末、公司规定的节假日以及下班后的时间。为此，我们用表 OTHER.C OMPANYHOLIDAYS 来记录公司规定的节假日。其结构如下：

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Emp_permit_changes
BEFORE INSERT OR DELETE OR UPDATE
ON OTHER.EMPTAB
DECLARE
    Dummy              INTEGER;
    Invalid_Operate_time     EXCEPTION FOR -20002;
```

```
BEGIN
    /* 检查是否周末 */
    IF (DAYNAME(Sysdate) = 'Saturday' OR
        DAYNAME(Sysdate) = 'Sunday') THEN
        RAISE Invalid_Operate_time;
    END IF;
    /* 检查是否节假日 */
    SELECT COUNT(*) INTO Dummy FROM OTHER.COMPANYHOLIDAYS
        WHERE Holiday= Current_date;
    IF dummy > 0 THEN
        RAISE Invalid_Operate_time;
    END IF;
    /* 检查是否上班时间 */
    IF (EXTRACT(HOUR FROM Current_time) < 8 OR
         EXTRACT(HOUR FROM Current_time) >= 18) THEN
         RAISE Invalid_Operate_time;
    END IF;
END;

SET SCHEMA SYSDBA;
```

14.5.4 使用触发器生成字段默认值

触发器还经常用来自动生成某些字段的值，这些字段的值有时依赖于本记录中的其他字段的值，有时是为了避免用户直接对这些字段进行修改。这类触发器应该是元组级 BEFORE INSERT 或 UPDATE 触发器。因为：

1．必须在 INSERT 或 UPDATE 操作执行之前生成字段的值；

2．必须为每条元组自动生成一次字段的值。

```
SET SCHEMA OTHER;

CREATE OR REPLACE TRIGGER Emp_auto_value
BEFORE INSERT
ON OTHER.EMPTAB
FOR EACH ROW
BEGIN
   :new.Sal = 999.99;
END;

SET SCHEMA SYSDBA;
```

第 15 章 DM 安全管理

数据库的安全性是指保护数据库以防止不合法的使用所造成的数据泄露、更改或破坏。安全性问题不是数据库系统所独有的，计算机系统都有这个问题，只是在数据库系统中大量数据集中存放，而且为许多用户直接共享，是宝贵的信息资源，从而使安全性问题更为突出。

数据库安全性保护措施是否有效是衡量数据库系统的重要性能指标之一。

在数据库存储这一级可采用密码技术，当物理存储设备失窃后，它起到保密作用。在数据库系统这一级中提供两种控制：用户标识和鉴定，数据存取控制。

数据库安全可分为二类：系统安全性和数据安全性。

系统安全性是指在系统级控制数据库的存取和使用的机制，包含：

1. 有效的用户名/口令的组合；
2. 一个用户是否授权可连接数据库；
3. 用户对象可用的磁盘空间的数量；
4. 用户的资源限制；
5. 数据库审计是否是有效的；
6. 用户可执行哪些系统操作。

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。

为了确保 DM 服务器的安全性，DM 在安全管理方面采用了三权分立的安全管理体制，即把系统管理员分为数据库管理员 DBA，数据库安全管理员 SSO，数据库审计员 AUDITOR 三类。把数据库的普通用户、审计用户与标记用户的权限及相关处理完全分开，包括系统预设系统登录、用户、系统角色、用户角色及权限分配和控制。它们三者之间互相联系、互相制约，共同完成数据库的管理工作。

在 DM 数据库系统中，安全机制做下列工作：

1. 防止非授权的数据库存取；
2. 防止非授权用户对模式对象的存取；
3. 控制磁盘使用；
4. 控制系统资源使用；
5. 审计用户动作。

用户要存取一个对象必须有相应的权限授给该用户。已授权的用户可任意地将它授权给其它用户，由于这个原因，这种安全性类型叫做任意型。

DM 利用下列机制管理数据库安全性：

1. 数据库用户和模式；
2. 权限；
3. 角色；
4. 存储设置和空间份额；
5. 资源限制；
6. 审计。

本章介绍的主要内容包括：角色与权限、自主存取控制、强制访问控制(仅在安全版本中提供)、审计管理以及加密引擎。这些内容属于高级系统管理范畴，是系统管理员应该掌握的内容。安全管理对信息敏感部门尤为重要，应用系统设计者和系统管理员必须熟练掌握

系统的安全特性，以便建立高安全性数据库应用系统。

虽然本章所讨论的内容是针对 DM 的，但其原理和方法与其它系统大同小异，因而，对于了解和掌握其它数据库管理系统的安全管理也是有帮助的。如不特别说明，各例均使用实例库 BOOKSHOP。

15.1 创建角色语句

在 DM 系统中，可以对用户直接授权，也可以通过角色来授权。在实际的权限分配方案中，通常是用角色。先为数据库定义一系列的角色，然后将权限分配给这些角色。基于这些角色的用户间接地获得权限。创建一个新的角色所用的语法如下。

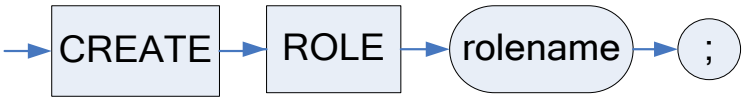
语法格式

```
CREATE ROLE  <角色名>;
```

参数

1．<角色名>　　指明要创建的角色的名称；

图例



语句功能

创建一般用户角色。

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．角色名不允许是关键字。

举例说明

例 1 创建角色 BOOKSHOP_ROLE1，赋予 ADDRESS 表的 SELECT 权限。

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

例 2 创建角色 ROLE。

```
CREATE ROLE ROLE;
```

说明：这一句话从语法上讲是正确的。但是不提倡创建这样的角色。

15.2 删除角色语句

删除一个角色。

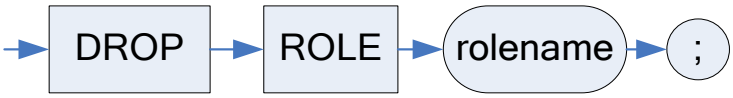
语法格式

```
DROP ROLE  <角色名>;
```

参数

1．<角色名>　　指明要删除的角色的名称。

图例



语句功能

从数据库中删除一个角色。

使用说明

删除者必须具有 DROP ROLE 数据库权限。

举例说明

例 假定存在角色 BOOKSHOP_ROLE1、用户 BOOKSHOP_USER1，现将角色权限授予 BOOKSHOP_USER1，然后回收角色权限，并删除角色。

```
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_USER1;
REVOKE BOOKSHOP_ROLE1 FROM BOOKSHOP_USER1;
DROP ROLE BOOKSHOP_ROLE1;
/*删除将会成功*/
```

15.3 授权语句(数据库权限)

将指定的数据库权限授予用户/角色。

语法格式

```
GRANT <特权> TO <用户或角色>{,<用户或角色>} [WITH    ADMIN OPTION];
<特权> ::= <动作>{,<动作>};
<动作> ::= <ALTER DATABASE>|
<BACKUP DATABASE>|
<CREATE TABLESPACE>|
<ALTER TABLESPACE>|
<DROP TABLESPACE>|
<CREATE ROLE>|
<DROP ROLE>|
<CREATE SCHEMA>|
<CREATE TABLE>|
<CREATE VIEW>|
<CREATE PROCEDURE>|
<CREATE SEQUENCE>|
<CREATE TRIGGER>|
<CREATE INDEX>|
<CREATE CONTEXT INDEX>|
<CREATE PACKAGE>|
<CREATE LINK>|
<CREATE TYPE>|
<CREATE SYNONYM>|
<CREATE PUBLIC SYNONYM>|
<DROP PUBLIC SYNONYM>|
<CREATE MATERIALIZED VIEW>|
<CREATE DOMAIN>
```

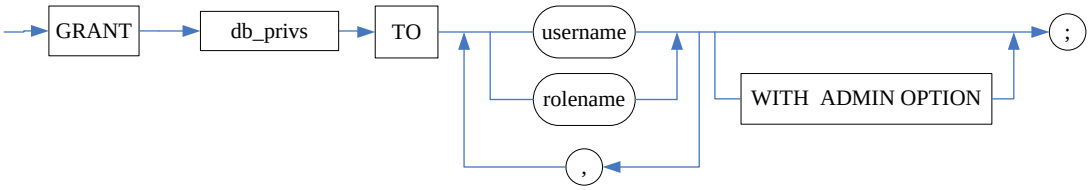
<用户或角色>::= <用户名> | <角色名>

参数

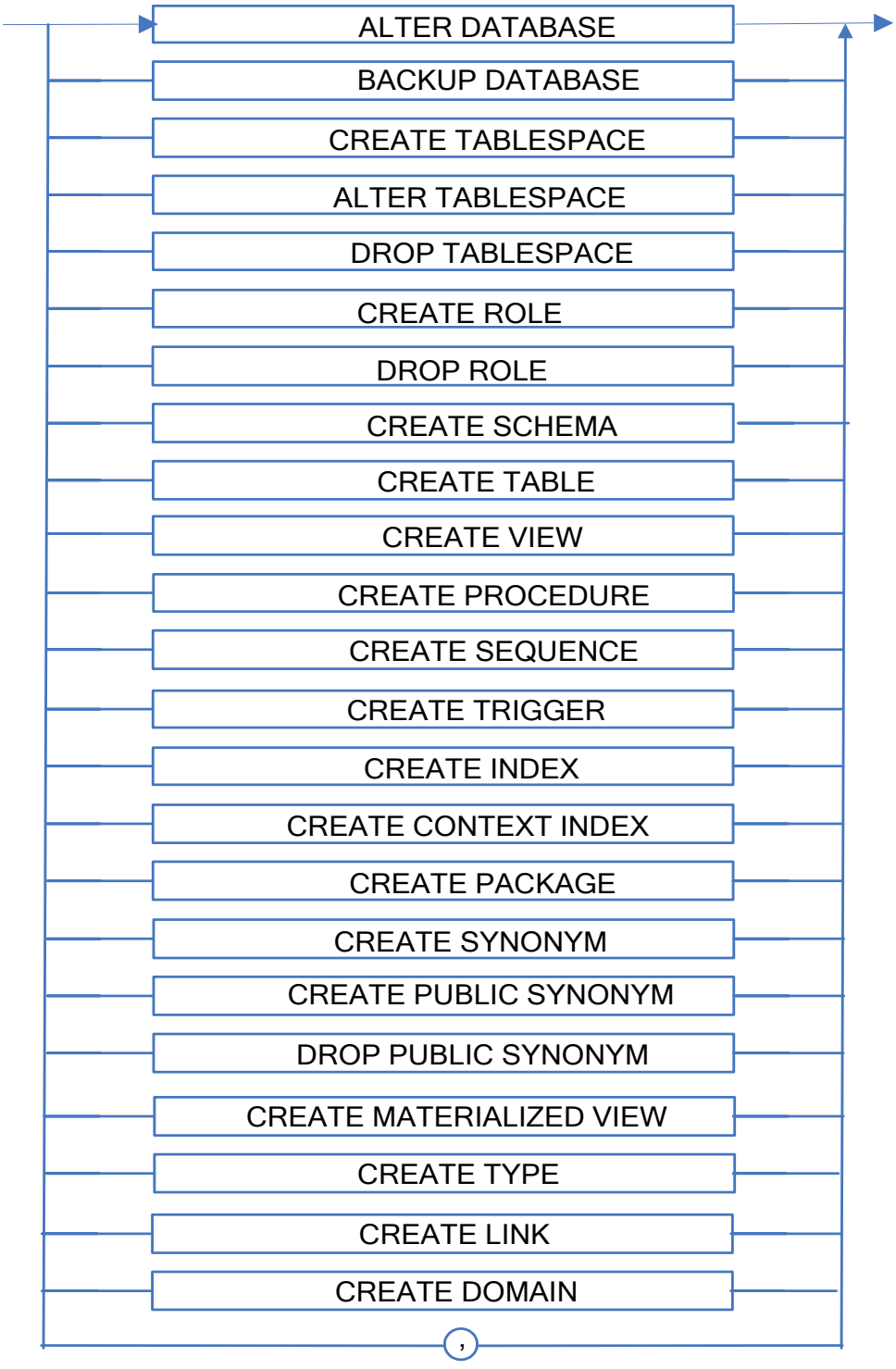
1．<用户名>　　指明被授予权限的用户的名称；

2．<角色名>　　指明被授予权限的角色的名称。

图例



db_privs：



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类型一致（譬如都是审计用户或都不是）；

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；

4. 有些数据库权限比较特殊，故不支持转授，这类权限集中在系统预定义角色中，只能通过转授这些角色进行授权。这类权限包括：

预设数据库权限	是否支持转授

CREATE USER	否
ALTER USER	否
DROP USER	否
CREATE DUPLICATE	否
ALTER DUPLICATE	否
DROP DUPLICATE	否
ADMIN ANY ROLE	否
ADMIN ANY DATABASE PRIVILEGE	否
GRANT ANY OBJECT PRIVILEGE	否
CREATE ANY SCHEMA	否
DROP ANY SCHEMA	否
CREATE ANY TABLE	否
ALTER ANY TABLE	否
DROP ANY TABLE	否
INSERT TABLE	否
INSERT ANY TABLE	否
UPDATE TABLE	否
UPDATE ANY TABLE	否
DELETE TABLE	否
DELETE ANY TABLE	否
SELECT TABLE	否
SELECT ANY TABLE	否
REFERENCES TABLE	否
REFERENCES ANY TABLE	否
GRANT TABLE	否
GRANT ANY TABLE	否
CREATE ANY VIEW	否
ALTER ANY VIEW	否
DROP ANY VIEW	否
INSERT VIEW	否
INSERT ANY VIEW	否
UPDATE VIEW	否
UPDATE ANY VIEW	否
DELETE VIEW	否
DELETE ANY VIEW	否
SELECT VIEW	否
SELECT ANY VIEW	否
GRANT VIEW	否
GRANT ANY VIEW	否
CREATE ANY PROCEDURE	否
DROP ANY PROCEDURE	否
EXECUTE PROCEDURE	否
EXECUTE ANY PROCEDURE	否

GRANT PROCEDURE	否
GRANT ANY PROCEDURE	否
CREATE ANY SEQUENCE	否
DROP ANY SEQUENCE	否
SELECT SEQUENCE	否
SELECT ANY SEQUENCE	否
GRANT SEQUENCE	否
GRANT ANY SEQUENCE	否
CREATE ANY TRIGGER	否
DROP ANY TRIGGER	否
CREATE ANY INDEX	否
ALTER ANY INDEX	否
DROP ANY INDEX	否
CREATE ANY CONTEXT INDEX	否
ALTER ANY CONTEXT INDEX	否
DROP ANY CONTEXT INDEX	否
CREATE ANY PACKAGE	否
DROP ANY PACKAGE	否
EXECUTE PACKAGE	否
EXECUTE ANY PACKAGE	否
GRANT PACKAGE	否
GRANT ANY PACKAGE	否
CREATE ANY LINK	否
DROP ANY LINK	否
CREATE ANY SYNONYM	否
DROP ANY SYNONYM	否
CREATE ANY TYPE	否
DROP ANY TYPE	否
SELECT ANY DICTIONARY	否
ADMIN REPLAY	否
ADMIN BUFFER	否
ALTER ANY TRIGGER	否
CREATE ANY MATERIALIZED VIEW	否
DROP ANY MATERIALIZED VIEW	否
ALTER ANY MATERIALIZED VIEW	否
SELECT MATERIALIZED VIEW	否
SELECT ANY MATERIALIZED VIEW	否
CREATE ANY DOMAIN	否
DROP ANY DOMAIN	否
GRANT ANY DOMAIN	否
GRANT DOMAIN	否
USAGE ANY DOMAIN	否
USAGE DOMAIN	否

举例说明

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

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

15.4 授权语句(对象权限)

将指定对象的权限授予用户。

语法格式

```
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|
USAGE
<列清单>::= <列名> {,<列名>}
<对象类型>::= TABLE | VIEW | PROCEDURE | SEQUENCE|DOMAIN
<对象> ::= [<模式名>.]<对象名>
<对象名> ::= <表名> | <视图名> | <存储过程/函数名> |<序列名>|<域名>
<用户或角色>::= <用户名> | <角色名>
```

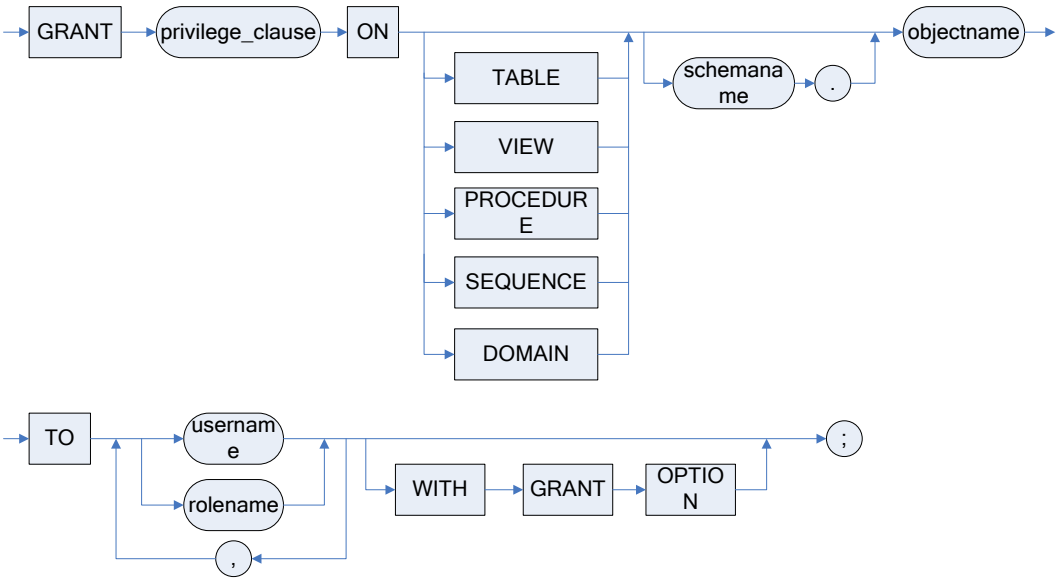
参数

1．<模式名>　　指明对象所属的模式名称；

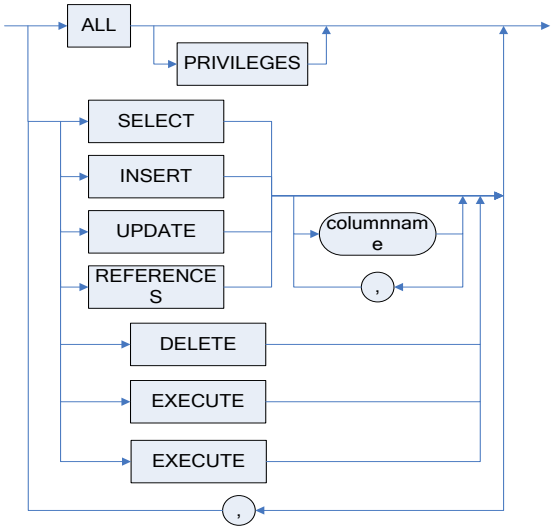
2．<用户名>　　指明被授予权限的用户的名称；

3．<角色名>　　指明被授予权限的角色的名称。

图例



privilege_clause



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；

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．对象重建后，不能保证原有权限的合法性。

举例说明

例 1 表的创建者 SYSDBA 把所创建的 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 BOOKSHOP_T1 TO BOOKSHOP_USER1;
```

例 2 数据库管理员 SYSDBA 把用户 BOOKSHOP_USER1 创建的存储过程 BOOKSHOP_USER1_PROC1 的执行权 EXECUTE 授给用户 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 SYSDBA 是表 BOOKSHOP_T1 的创建者，用户 BOOKSHOP_USER1、BOOKSHOP_USER2、BOOKSHOP_USER3 存在，且都是 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 BOOKSHOP_T1 TO BOOKSHOP_USER2 WITH GRANT OPTION;
(3)以 BOOKSHOP_USER2 身份登录，并执行语句：
GRANT SELECT ON BOOKSHOP_T1 TO BOOKSHOP_USER3 WITH GRANT OPTION;
```

```
/*正确*/
```

例 4 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 BOOKSHOP_T1 TO BOOKSHOP_USER2;
/*正确*/
(3)以 BOOKSHOP_USER2 身份登录，并执行语句：
GRANT SELECT ON BOOKSHOP_T1 TO BOOKSHOP_USER3;
/*错误，用户 BOOKSHOP_USER2 没有 SELECT 权限的转授权*/
```

例 5 假定用户 SYSDBA 创建了一个名为 V_PRODUCT 的视图，要求对于该视图，既允许 BOOKSHOP_USER1 能够进行查询、插入、删除和更新操作。用户 BOOKSHOP_USER2 和 BOOKSHOP_USER3 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 V_PRODUCT AS
SELECT *
FROM PRODUCTION.PRODUCT
WHERE NOWPRICE>=20 AND ORIGINALPRICE<40 WITH CHECK OPTION;
GRANT SELECT, INSERT, DELETE, UPDATE
ON V_PRODUCT
TO BOOKSHOP_USER1
WITH GRANT OPTION;
(2)以用户 BOOKSHOP_USER1 的身份登录：
GRANT SELECT, INSERT, DELETE, UPDATE ON V_PRODUCT TO BOOKSHOP_USER2, BOOKSHOP_USER3;
```

例 6 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 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 PRODUCT
TO BOOKSHOP_USER2;
```

15.5 授权语句(角色权限)

一般用户角色包含权限或其它角色。用户或其它角色可以继承该角色的权限。

语法格式

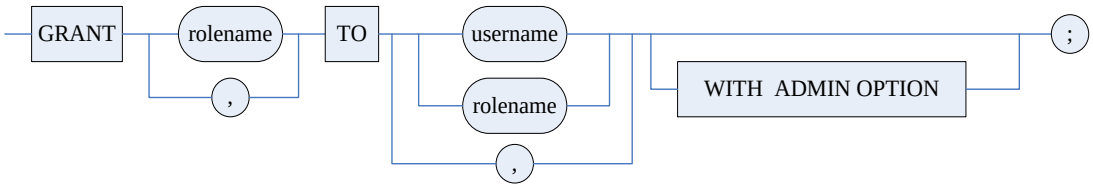
```
GRANT <角色名>{, <角色名>} TO <用户或角色>{,<用户或角色>} [WITH    ADMIN OPTION];
<用户名或角色名> ::= <用户名> | <角色名>
```

参数

1．<角色名>　　指明系统中已经创建的一个角色的名称；

2．<角色名或用户名>　指明要继承指定角色权限的其它角色或用户。

图例



使用说明

1．角色的授予者必须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。

举例说明

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

```
(1)让用户 BOOKSHOP_USER1 继承角色 BOOKSHOP_ROLE1 的权限：
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_USER1;
(2)让角色 BOOKSHOP_ROLE2 继承角色 BOOKSHOP_ROLE1 的权限：
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_ROLE2;
```

15.6 回收权限语句(数据库权限)

回收用户或者角色的数据库权限。

语法格式

```
REVOKE [ADMIN OPTION FOR]<特权> FROM <用户或角色>{,<用户或角色>} ;
<特权> ::= <动作>{,<动作>}
<动作> ::= <ALTER DATABASE>|
<BACKUP DATABASE>|
<CREATE TABLESPACE>|
<ALTER TABLESPACE>|
<DROP TABLESPACE>|
<CREATE ROLE>|
<DROP ROLE>|
<CREATE SCHEMA>|
<CREATE TABLE>|
<CREATE VIEW>|
```

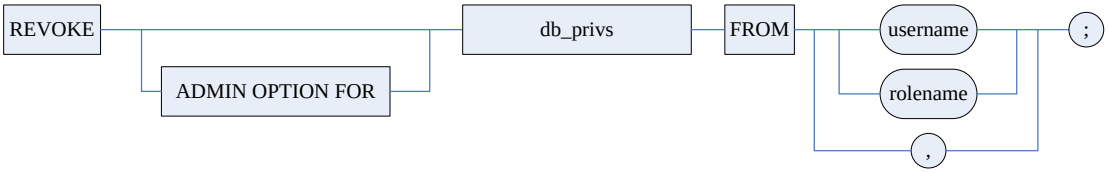
<CREATE PROCEDURE>|
<CREATE SEQUENCE>|
<CREATE TRIGGER>|
<CREATE INDEX>|
<CREATE CONTEXT INDEX>|
<CREATE PACKAGE>|
<CREATE SYNONYM>|
<CREATE PUBLIC SYNONYM>|
<DROP PUBLIC SYNONYM>|
<AUDIT DATABASE>|
<LABEL DATABASE>
<用户或角色>::= <用户名> | <角色名>

参数

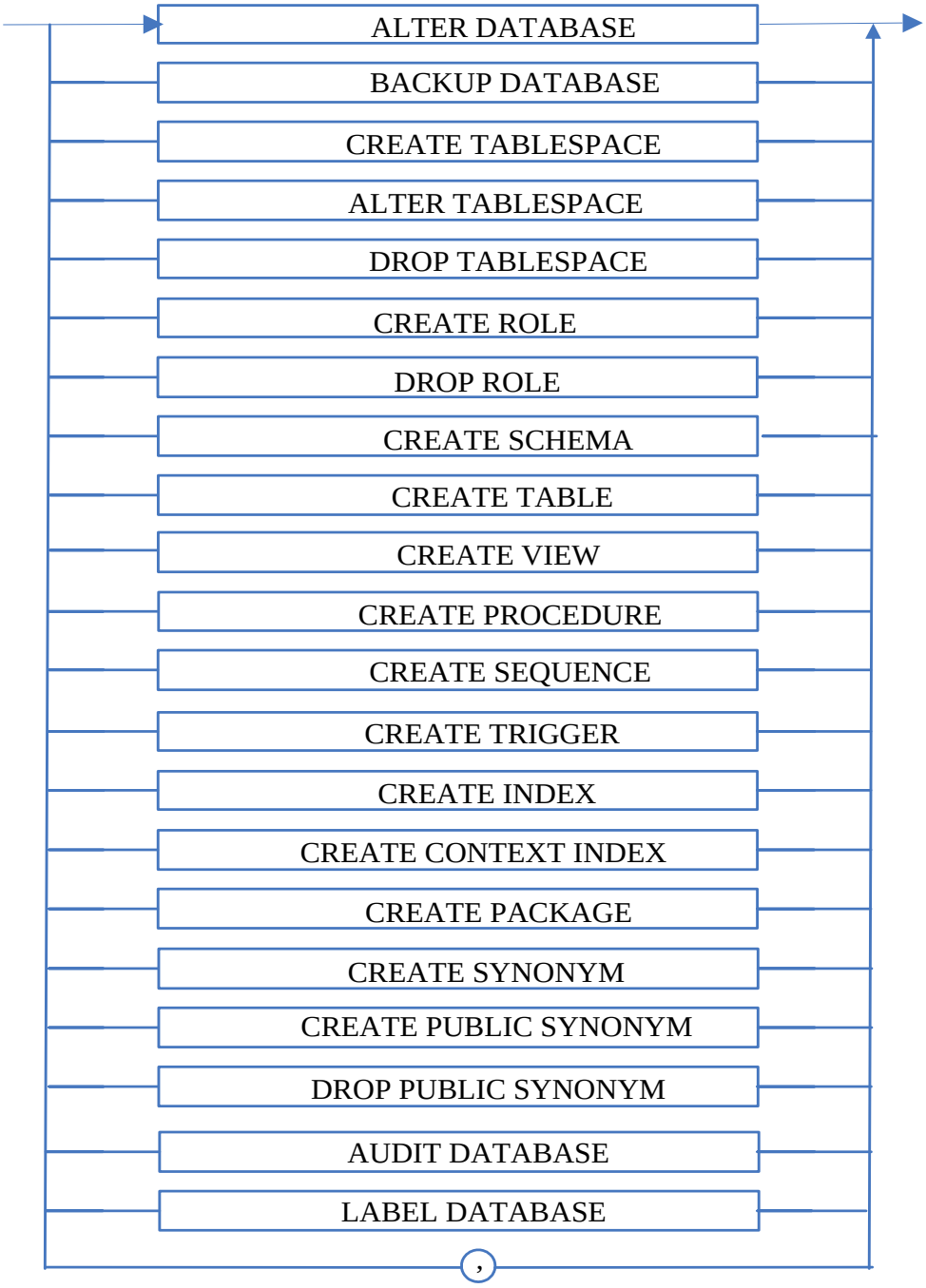
1．<用户名>　指明被回收权限的用户的名称；

2．<角色名>　指明被回收权限的角色的名称。

图例



db_privs：



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。

举例说明

例 假设系统管理员 SYSDBA 把建表、建视图的权限授给了用户 BOOKSHOP_USER1。现在，系统管理员 SYSDBA 把用户 BOOKSHOP_USER1 的建表权限收回。

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

例 假设系统管理员 SYSDBA 把建表的权限授给了用户 BOOKSHOP_USER1，并且允许转授。现在 SYSDBA 不让用户 BOOKSHOP_USER1 转授 CREATE TABLE 权限。

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

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

15.7 回收权限语句(对象权限)

回收用户对指定实体的权限。

语法格式

```
REVOKE [GRANT OPTION FOR] <特权> ON [<对象类型>]<数据库对象> 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|
USAGE
<对象类型>::= TABLE | VIEW | PROCEDURE | SEQUENCE| DOMAIN
<对象名> ::= <表名> | <视图名> | <存储过程/函数名> |<序列名>|<域名>
<用户或角色>::= <用户名> | <角色名>
<回收选项> ::= RESTRICT | CASCADE
```

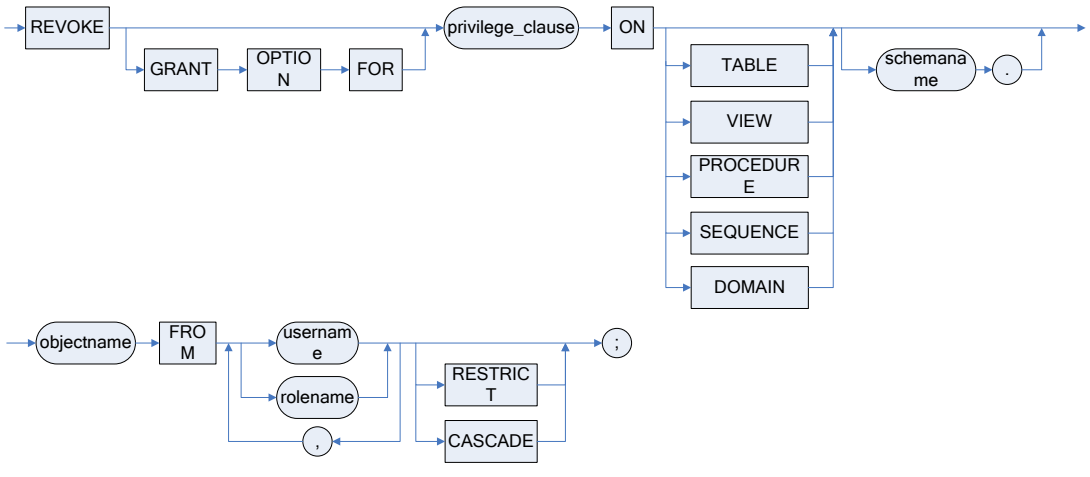
参数

1．<模式名>　指明被回收权限的用户或角色所属模式名称；

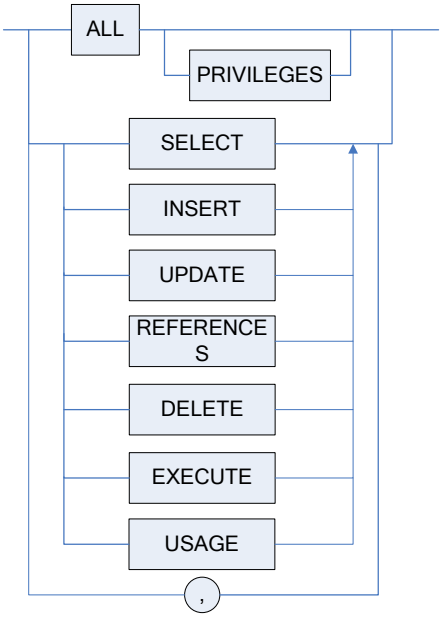
2．<用户名>　指明被回收权限的用户的名称；

3．<角色名>　指明被回收权限的角色的名称。

图例



privilege_clause



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：

1) 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；

2) 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；只能回收授予时没有带有 WITH GRANT OPTION 选项的权限；

3) 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，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。

举例说明

例 1 视图 V_PRODUCT 的创建者 SYSDBA 从用户 BOOKSHOP_USER1 处回收其对视图 V_PRODUCT 的所有权限。

```
REVOKE ALL ON V_PRODUCT FROM BOOKSHOP_USER1;
```

例 2 SYSDBA 从用户 BOOKSHOP_USER1 处回收其对存储过程 BOOKSHOP_FUNC1 的 EXECUTE 权限，该存储过程的创建者为用户 SYSDBA。

```
REVOKE EXECUTE ON PROCEDURE BOOKSHOP_FUNC1 FROM BOOKSHOP_USER1;
```

例 3 假定系统中存在非 DBA 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:
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:
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:
GRANT INSERT(ID) ON BOOKSHOP_USER1.T1 TO BOOKSHOP_USER4 WITH GRANT OPTION;
BOOKSHOP_USER4:
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以前授予的 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;
```

15.8 回收权限语句(角色权限)

用来回收用户或其它角色从指定角色继承过来的权限。

语法格式

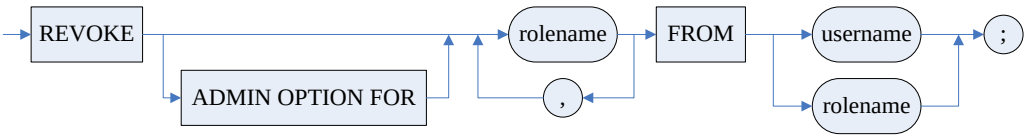
```
REVOKE  [ADMIN OPTION FOR]<角色名>{,<角色名>} FROM  <角色名或用户名>;
<角色名或用户名> ::= <用户名> | <角色名>
```

参数

1．<角色名>　　指明系统中已经创建的一个角色的名称；

2．<角色名或用户名>　指明要回收其指定角色权限的其它角色或用户。

图例



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 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。

举例说明

例 假定数据库中存在角色 BOOKSHOP_ROLE1，角色 BOOKSHOP_ROLE2，用户 BOOKSHOP_USER1。

```
(1)让用户 BOOKSHOP_USER1 继承角色 BOOKSHOP_ROLE1 的权限：
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_USER1;
(2)回收用户 BOOKSHOP_USER1 从角色 BOOKSHOP_ROLE1 继承过来的权限：
REVOKE BOOKSHOP_ROLE1 FROM BOOKSHOP_USER1;
(3)让角色 BOOKSHOP_ROLE2 继承角色 BOOKSHOP_ROLE1 的权限：
GRANT BOOKSHOP_ROLE1 TO BOOKSHOP_ROLE2;
(4)回收角色 BOOKSHOP_ROLE2 从角色 BOOKSHOP_ROLE1 继承过来的权限：
REVOKE BOOKSHOP_ROLE1 FROM BOOKSHOP_ROLE2;
```

15.9 策略与标记管理

为了实现强制访问控制，安全管理员需要在每个数据库中定义多种安全策略，每个安全策略中可以定义多种安全等级、范围和组，用来表示现实生活中的不同安全特征。将这些安全策略应用于表和用户上，就给用户、表和元组都指定了安全标记。数据库就是通过比较用户、表、以及元组之间的安全标记来确定用户是否可以操作表或元组的。

15.9.1 策略

15.9.1.1 创建策略

创建一个新的策略。

函数说明

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

参数

POLICY_NAME　新创建的策略名称。

使用说明

该语句只能由 SYSSSO 管理员执行。

举例说明

例 创建策略 P_01。

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

15.9.1.2 修改策略

修改一个策略的策略名。

函数说明

```
VOID
MAC_ALTER_POLICY(
```

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

参数

1．POLICY_NAME　待修改的策略名称；

2．NEW_NAME　将修改成的策略名称。

使用说明

该语句只能由 SYSSSO 管理员执行。

举例说明

例 将策略 P_01 改为 P_02。

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

15.9.1.3 删除策略

删除一个策略。

函数说明

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

参数

1．POLICY_NAME　待删除的策略名称；

2．DROP_COLUMN

1：删除应用此策略的所有表对应的标记列；

0：不删除应用此策略的所有表对应的标记列；

若 DROP_COLUMN 为 NULL，则按照默认为 0 进行处理。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在。

举例说明

例 删除策略 P_02。

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

15.9.2 等级

15.9.2.1 创建等级

根据策略名，给相应的策略添加等级。

函数说明

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

```
);
```

参数

1．POLICY_NAME　要添加等级的策略名；

2．LEVEL_NUM 介于 0-9999 之间的整数；

3．LEVEL_NAME　新创建的等级名称。

使用说明

1．LEVEL_NAME 不能包含“:”和“,”；

2．该语句只能由 SYSSSO 管理员执行；

3．该策略必须存在；

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。

举例说明

例 创建策略 P_03，并给策略 P_03 添加等级 L_01。

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

15.9.2.2 更新等级

更新一个 LEVEL 的名称。

函数说明

```
VOID
MAC_ALTER_LEVEL(
    POLICY_NAME         VARCHAR(128),
    LEVEL_NAME          VARCHAR(128),
    NEW_NAME            VARCHAR(128)
);
```

参数

1．POLICY_NAME　要更新的等级名所在的策略名称；

2．LEVEL_NAME　待修改的等级名称；

3．NEW_NAME　待修改成的等级名称。

使用说明

1．NEW_LEVEL 不能包含“:”和“,”；

2．该语句只能由 SYSSSO 管理员执行；

3．该等级必须存在。

举例说明

例 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');
```

15.9.2.3 删除等级

删除相应策略中的等级。

函数说明

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

```
    LEVEL_NAME          VARCHAR(128)
);
```

参数

1．POLICY_NAME　待删除的等级名所在的策略名称；

2．LEVEL_NAME　待删除的等级名称。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该等级必须存在；

3．删除一个等级，如果这个等级被某个标记使用，则拒绝删除。

举例说明

例 删除策略 P_03 中的等级 L_02。

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

15.9.3 范围

15.9.3.1 创建范围

根据策略名，给相应的策略添加范围。

函数说明

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

参数

1．POLICY_NAME　要添加范围的策略名；

2．COMPART_NUM　介于 0-9999 之间的整数；

3．COMPART_NAME 新创建的范围名称。

使用说明

1．COMPART_NAME 不能包含“:”和“,”；

2．该语句只能由 SYSSSO 管理员执行；

3．该策略必须存在；

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。

举例说明

例 给策略 P_03 添加范围 C_01。

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

15.9.3.2 更新范围

更新一个范围的名称。

函数说明

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

参数

1．POLICY_NAME　要更新的范围名所在的策略名称；

2．COMPART_NAME 待修改的范围名称；

3．NEW_NAME　　待修改成的范围名称。

使用说明

1．NEW_NAME 不能包含“:”和“,”；

1．该语句只能由 SYSSSO 管理员执行；

2．该范围必须存在。

举例说明

例 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');
```

15.9.3.3 删除范围

删除相应策略中的范围。

函数说明

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

参数

1．POLICY_NAME　待删除的范围名所在的策略名称；

2．COMPART_NAME 待删除的范围名称。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该范围必须存在；

3．删除一个范围，如果这个范围被某个标记使用，则拒绝删除。

举例说明

例 删除策略 P_03 中的范围 C_02。

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

15.9.4 组

15.9.4.1 创建组

根据策略名，给相应的策略添加组。

函数说明

```
VOID
```

```
MAC_CREATE_GROUP(
    POLICY_NAME         VARCHAR(128),
    GROUP_NUM           INT,
    GROUP_NAME          VARCHAR(128),
    PARENT_NAME         VARCHAR(128)
);
```

参数

1．POLICY_NAME　要添加组的策略名；

2．GROUP_NUM 介于 0-9999 之间的整数；

3．GROUP_NAME　新创建的组名称；

4．PARENT_NAME　新创建的组的父组名称。

使用说明

1．GROUP_NAME 不能包含“:”和“,”；

2．该语句只能由 SYSSSO 管理员执行；

3．该策略必须存在；

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 P 父组为 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。

举例说明

例 给策略 P_03 添加组 G_01，并创建根组；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');
```

15.9.4.2 更新组

更新一个组的名称。

函数说明

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

参数

1．POLICY_NAME　要更新的组名所在的策略名称；

2．GROUP_NAME　　待修改的组名称；

3．NEW_NAME　　待修改成的组名称。

使用说明

1．NEW_NAME 不能包含“:”和“,”；

1．该语句只能由 SYSSSO 管理员执行；

2．该组必须存在。

举例说明

例 将策略 P_03 中的组 G_03，更名为 G_04。

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

15.9.4.3 更新父组

更新一个组的父组。

函数说明

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

参数

1．POLICY_NAME　要更新的组名所在的策略名称；

2．GROUP_NAME　待修改的组名称；

3．PARENT_NAME　待修改成的父组名称。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该组必须存在；

3．更新一个组的父节点，父节点不能是自身，同时不能是其子节点。

举例说明

例 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');
```

15.9.4.4 删除组

删除相应策略中的组。

函数说明

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

参数

1．POLICY_NAME　待删除的组名所在的策略名称；

2．GROUP_NAME　待删除的组名称。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该组必须存在；

3．删除一个组，不能有孩子存在，否则删除失败。

举例说明

例 删除策略 P_03 中的组 G_04。

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

15.9.5 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>}];
```

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，即用户在做插入时，若不指定列清单，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。

标记列与表上的其他列的处理基本一样，可以建索引，设置列约束、改列名等，但不能被用户显式删除，除非在表上取消该策略。

15.9.5.1 创建标记

创建一个新的标记。

函数说明

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

参数

1．POLICY_NAME　要添加标记的策略名；

2．LABEL_TAG　标记值，有效范围为 0~999999999；

3．LABEL_VALUE　标记串。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在；

3．创建标记所用到的等级、范围或组必须存在；

4．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。

举例说明

例 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');
```

15.9.5.2 更新标记

根据标记的标记值更新标记，根据标记串更新标记。

函数说明

```
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

1．POLICY_NAME　要更新标记的策略名；

2．LABEL_TAG　标记值；

3．LABEL_VALUE　待更新的标记串；

4．NEW_LABEL_VALUE　待更新为的标记串。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在；

3．更新标记串，可以根据标记值或标记串进行更新，故提供了重载函数；

4．新的标记串必须是不存在的标记串，如果存在则报错；

5．更新标记的用处主要在于可以不用更新表中的标记列，而直接更新标记串，来改变原始数据的安全级别；

6．如果新的标记串为 NULL，则保持原始值；

7．如果更新的标记同时应用在用户上，那么用户的标记合法性就会遭到破坏。

举例说明

例 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: ');
```

15.9.5.3 删除标记

删除一个标记。

函数说明

```
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

1．POLICY_NAME　要删除标记的策略名；

2．LABEL_TAG　待删除的标记值；

3．LABEL_VALUE　待删除的的标记串。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在；

3．支持两种方式删除标记(函数重载) ，以标记值删除标记和以原始标记串删除相应的标记；

4．待删除的标记值必须是已存在的标记值，否则报错；

5．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。

举例说明

例 删除标记。

```
MAC_CREATE_LABEL('P_04',12, 'L_01:: ');
MAC_DROP_LABEL('P_04',11);
MAC_CREATE_LABEL('P_04',13, 'L_01::G_01');
MAC_DROP_LABEL('P_04', 'L_01::G_01');
```

15.9.5.4 隐式创建标记

创建一个隐式标记。

函数说明

```
VOID
SF_MAC_LABEL_FROM_CHAR(
    POLICY_NAME     VARCHAR(128),
    LABEL_VALUE     VARCHAR(4000)
);
```

参数

1．POLICY_NAME　要添加标记的策略名；

2．LABEL_VALUE　标记串。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在；

3．获取一个标记字符串格式的内部形式，返回一个 TAG 值，若此标记值已经存在，则直接返回其 TAG，若不存在，则新建一个标记，TAG 由系统自动生成。

举例说明

例 隐式创建标记，标记值不存在的情况。

```
SELECT SF_MAC_LABEL_FROM_CHAR ('P_04', 'L_01::G_02');
```

例 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');
SELECT SF_MAC_LABEL_FROM_CHAR ('P_04', 'L_01:C_01:G_02');
```

15.9.6 表标记

15.9.6.1 表应用策略

将策略应用于指定的表，使该表处于一定的等级和范围内。

函数说明

```
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

1．POLICY_NAME　应用于指定表的策略名称；

2．SCHEMANAME　表所属的模式名称；

3．TABLENAME　策略所应用的表名称；

4．COLNAME　用于存放该策略数据的列名称；

5．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；

6．OPTION　1 代表隐藏标记列，0 代表不隐藏标记列。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在；

3．<列名>用来指定向该表中新增一列，用来存放此策略的数据。每新增一个策略都要新增一列，注意列名不可重名。该列不能被用户直接删除；

4．关键字 OPTION 用来指定该列是否隐藏。若修改表策略时指定新增的这一列隐藏，那么对表进行 INSERT 数据时，如果未指定具体的列，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，此列也是隐藏不予显示；但也允许通过指明具体的列来查询该列。若修改表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；

5．修改表策略时，若未指定新增的这一列是否 HIDE，默认为 NOT HIDE；

6．应用于表的等级、范围和组必须是在该策略中定义的；

7．当一个新的策略被应用于表上时，SYSSSO 管理员所给的初始标记用于初始化表中已有的数据关于该策略的标记；

8．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；

9．当从表上删除一个策略时，可选择是否保留其对应的标记列数据。

举例说明

例 1 给 PRODUCTION 模式中 PRODUCT 表，应用标记列 LABEL_COL，不隐藏标记列。

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

15.9.6.2 取消表策略

清除指定表上的标记列。

函数说明

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

参数

1．POLICY_NAME　应用于指定表的策略名称；

2．SCHEMANAME　表所属的模式名称；

3．TABLENAME 策略所应用的表名称；

4．DROP_COLUMN　1 代表删除标记列，0 代表不删除标记列，默认为 0。

使用说明

1．该语句只能由 SYSSSO 管理员执行；

2．该策略必须存在。

举例说明

例 1 取消应用在 PRODUCTION 模式中 PRODUCT 表上的标记列，删除标记列。

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

15.9.7 用户标记

15.9.7.1 设置用户等级

将策略应用于指定的用户，给用户设定等级。

函数说明

```
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

1．POLICY_NAME　应用于用户的策略名称；

2．USER_NAME 策略所应用的用户名称；

3．MAX_LEVEL 应用于用户的最大等级；

4．MIN_LEVEL　应用于用户的最小等级；

5．DEF_LEVEL　　应用于用户的默认等级；

6．ROW_LEVEL 应用于用户的行等级。

使用说明

用户标记来源于策略。包含以下几个组成部分。

表 15.9.1 用户等级属性表

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

1．max_level：　不能为空；

2．min_level：　不指定时，设置为策略的最小级别；

3．def_level：　不指定时，设置为 max_level；

4．row_level：　不指定时，设置为 def_level；

5．合法的 level 规则如下：

max_level >= min_level

max_level >= def_level >= min_level

def_level >= row_level >= min_level

举例说明

例 1 设置 BOOKSHOP_USER 用户的标记。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');
```

15.9.7.2 设置用户范围

将策略应用于指定的用户，给用户指定范围。

函数说明

```
VOID
MAC_USER_SET_COMPARTMENTS(
    POLICY_NAME     VARCHAR(128),
    USER_NAME       VARCHAR(128),
    READ_COMP       VARCHAR(128),
    WRITE_COMP      VARCHAR(128),
    DEF_COMP        VARCHAR(128),
    ROW_COMP        VARCHAR(128)
);
```

参数

1．POLICY_NAME　应用于用户的策略名称；

2．USER_NAME 策略所应用的用户名称；

3．READ_COMP 应用于用户的可读范围；

4．WRITE_COMP　应用于用户的可写范围；

5．DEF_COMP　应用于用户的默认范围；

6．ROW_COMP 应用于用户的行范围。

使用说明

对于范围，其属性列表如下。

表 15.9.2 用户范围属性表

属性名称
READ
WRITE
DEFAULT
ROW

1．READ：　　所有的范围都具有这个属性；
2．DEFAULT：　会话默认使用的；
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．合法的范围规则：

WRITE 属于 READ
DEFAULT 属于 READ
ROW 属于 DEFAULT 和 WRITE 的交集

举例说明

例 设置 BOOKSHOP_USER 用户的标记。范围 C_01,C_02,C_03 已存在。

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

15.9.7.3 设置用户组

将策略应用于指定的用户，给用户指定组。

函数说明

```
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

1．POLICY_NAME　应用于用户的策略名称；
2．USER_NAME 策略所应用的用户名称；
3．READ_GROUP　应用于用户的可读组；
4．WRITE_GROUP　应用于用户的可写组；
5．DEF_GROUP 应用于用户的默认组；
6．ROW_GROUP 应用于用户的行组。

使用说明

对于组，其属性列表如下。

表 15.9.3 用户组属性表

属性名称
READ
WRITE

DEFAULT
ROW

1．READ：　　所有的组都具有这个属性；

2．DEFAULT：　会话默认使用的；

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．合法的组规则：

WRITE 属于 READ

DEFAULT 属于 READ

ROW 属于 DEFAULT 和 WRITE 的交集

举例说明

例 设置 BOOKSHOP_USER 用户的标记。组 G_01,G_02,G_03 已存在。

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

15.9.7.4 设置用户特权

设置用户特权。

函数说明

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

参数

1．POLICY_NAME　策略名；

2．USER_NAME 用户名；

3．PRIVS　　特权类型。

使用说明

1. PRIVS 取值可以是 READ, FULL, WRITEUP, WRITEDOWN, WRITEACCROSS，这几种的一种或几种的组合；

2. 访问特权分为两种：

1) 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；

2) FULL：可以读写任何数据，不受策略影响；

3. 行标记特权，一旦一个行的标记设定后，就需要行标记特权才能改变其标记。对表中记录进行 UPDATE 显示更新 LABEL 时，需要行标记特权，行标记特权有如下三种：

1) WRITEUP：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；

2) WRITEDOWN：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；

3) WRITEACROSS：用户可以利用该特权修改一个行的范围和等级，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。

举例说明

例 设置 BOOKSHOP_USER 用户的特权。

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

15.9.8 会话标记

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。

会话标记的设置是通过对外存储过程接口进行实现的。

15.9.8.1 设置会话默认标记

设置会话的默认标记。

函数说明

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

参数

1．POLICY_NAME　策略名；

2．LABELVALUE标记值。

使用说明

1． 设置会话默认标记：设置会话的默认标记，其值必须在合法的范围内，等级必须在 MIN 和 MAX 之间，范围和组必须是 READ 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。

举例说明

例 设置 BOOKSHOP_USER 用户，会话默认标记。

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

15.9.8.2 设置会话行标记

设置会话的行标记。

函数说明

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

参数

1．POLICY_NAME　策略名；

2．LABELVALUE标记值。

使用说明

设置会话的行标记，保证其值是合法值即可。

举例说明

例 设置当前会话行标记。

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

15.9.8.3 清除会话标记

清除会话上相应策略的标记。

函数说明

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

参数

POLICY_NAME　　策略名。

使用说明

清除当前会话所有的标记，仅用户的标记可用。

举例说明

例 清除当前的会话标记。

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

15.9.8.4 保存会话标记

将会话上指定策略刷入相应的数据字典。

函数说明

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

参数

POLICY_NAME　　策略名。

使用说明

用会话的标记刷入对应用户的数据字典，清除相应的用户对象，清除会话的 LABEL，因为此时的会话标记和用户标记是一致的，只需使用用户的标记即可。

举例说明

例 将当前会话上指定策略刷入相应的数据字典。

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

15.9.9 扩展客体标记

扩展客体标记支持对数据库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。

对于任何一个对象的操作，用户必须具有本层对象以及上层对象的标记，才能操作此对象，如查询一个表的一列，则用户的读标记必须支配查询列、表、以及模式的标记，才能查询此列。

对对象的不同的 DML 操作，会匹配不同的用户标记，如对对象的 SELECT、EXECUTE 以及 REFERENCE，会检查用户的读标记。对对象的 UPDATE、INSERT、DELETE，会检查用户的写标记。

对对象的 DDL 操作，也会根据用户的写标记进行相应的扩展标记检查，检查范围为改

对象的标记，以及该对象上层的标记。如删除一个表，则会检查用户的写标记是否支配表、模式的标记。

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。

15.9.9.1 对客体应用标记

函数说明

```
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

1. POLICY_NAME：策略名称；
2. OBJ_TYPE：必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”；
3. SCH_NAME：模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词；
4. 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；
5. COL_NAME：应用标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效；
6. LABEL：具体的标记值。

使用说明

从整体上对客体进行打标记，控制用户对客体的操作。

举例说明

例 对模式 SYSDBA 下的表 T1 应用策略 P1 的标记（L1:C1:G1）。

```
MAC_APPLY_OBJ_POLICY('P1', 'TABLE','SYSDBA', 'T1', NULL, 'L1:C1:G1');
```

15.9.9.2 修改客体标记

函数说明

```
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

1. POLICY_NAME：策略名称；
2. OBJ_TYPE：必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、

“COLUMN”、“SYNONYM”；

3. SCH_NAME：模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词；
4. 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；
5. COL_NAME：应用标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效；
6. LABEL：具体的标记值。

使用说明

修改客体的标记，达到控制访问的目的。

举例说明

例 修改 SYSDBA 下表 T1 的标记，改为标记（L1:C1,C2:G1）。

```
MAC_ALTER_OBJ_POLICY('P1', 'TABLE','SYSDBA', 'T1', NULL, 'L1:C1,C2:G1');
```

15.9.9.3 删除客体标记

函数说明

```
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

1. POLICY_NAME：策略名称；
2. OBJ_TYPE：必须是下面几个选项之一，“SCHEMA”、“TABLE”、“VIEW”、“INDEX”、“PROCEDURE”、“FUNCTION”、“PACKAGE”、“SEQUENCE”、“TRIGGER”、“COLUMN”、“SYNONYM”；
3. SCH_NAME：模式名，为 NULL 时代表对库级的对象应用策略，如公用同义词；
4. 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；
5. COL_NAME：应用标记的列名，只有在 OBJ_TYPE='COLUMN'时，此列才有效。

使用说明

删除对客体的标记。

举例说明

例 删除 SYSDBA 下表 T1 的扩展标记。

```
MAC_DROP_OBJ_POLICY('P1','TABLE','SYSDBA','T1',NULL);
```

15.10 审计设置语句

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开关，DM 则对审计设置语句指定的审计对象进行审计，否则不记录审计记录。此开关的设置是通过 DM 控制台工具 Console 中数据库配置中的 ENABLE_AUDIT 参数进行配置，设置为 1 则打开普通审计开关，设置为 2 则打开普通审计和实时审计，设置为 0 则关闭审计开关，此配置项默认值为 0。参数值修改后，需服务器重新启动才生效。

数据库审计员指定被审计对象的活动称为审计设置。DM 提供审计设置语句来实现这种

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 15.10.1 所示。

表 15.10.1 审计级别

审计级别	说明
系统级	系统的启动与关闭，此级别的审计记录在任何情况下都会强制产生，无法也无需由用户进行设置。
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。

审计设置存放于系统表中，进行一次审计设置在系统表 SYSAUDIT 中增加设置的记录，取消审计则删除系统表 SYSAUDIT 中相应的记录。

所有的审计记录都存放在审计日志中，审计日志命名方式为 AUDIT_自动生成的全局唯一的串_日期时间.log，服务器每次重启生成一个日志文件。

DM 的审计分为两大类，语句级审计和对象级审计。语句级审计的动作是全局的，不对应具体的数据库对象。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。

审计选项如表 15.10.2 和表 15.10.3 所示。

表 15.10.2 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	创建／修改／删除视图操作

	DROP VIEW	
INDEX	CREATE INDEX DROP INDEX	创建 / 删除索引操作
PROCEDURE	CREATE PROCEDURE ALTER PROCEDURE DROP PROCEDURE	创建 / 修改 / 删除存储模块操作
TRIGGER	CREATE TRIGGER ALTER TRIGGER DROP TRIGGER	创建 / 修改 / 删除触发器操作
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE	创建 / 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 DROP POLICY ALTER POLICY	创建 / 修改 / 删除策略
MAC LEVEL	CREATE LEVEL	创建 / 修改 / 删除等级

	ALTER LEVEL DROP LEVEL	
MAC COMPARTMENT	CREATE COMPARTMENT DROP COMPARTMENT ALTER COMPARTMENT	创建／修改／删除范围
MAC GROUP	CREATE GROUP DROP GROUP ALTER GROUP ALTER GROUP PARENT	创建／修改／删除组，更新父组
MAC LABEL	CREATE LABEL DROP LABEL ALTER LABEL	创建／修改／删除标记
MAC USER	USER SET LEVELS USER SET COMPARTMENTS USER SET GROUPS USER SET PRIVS	设置用户等级／范围／组／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	CHECKPIONT	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

表 15.10.3　对象级审计选项

审计选项(SYSAUDITRECORDS 表中 operation 字段对应的内容)	TABLE	VIEW	PROCEDURE FUNCTION
INSERT(insert)	√	√	
UPDATE(update)	√	√	
DELETE(delete)	√	√	
SELECT(select)	√	√	
EXECUTE(execute)			√
MERGE INTO	√	√	
EXECUTE TRIGGER			
LOCK TABLE	√		
ALL(所有对象级审计选项)	√	√	√

审计语法格式

参照附录 3 中审计函数

SP_AUDIT_STMT 和 SP_AUDIT_OBJECT

审计功能使用说明：

1．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；

2．无论数据库审计功能是否被启用，系统级的审计记录都会产生；

3．语句级审计不针对特定的对象，只针对用户；

4．对象级审计针对指定的用户与指定的对象进行审计；

5．在设置审计时，审计选项不区分包含关系，都可以设置；

6．在设置审计时，审计时机不区分包含关系，都可以进行设置；

7．所有审计员执行的 SELECT、INSERT、DELETE、UPDATE 操作不审计；

8．如果用户执行的一条语句与设置的若干审计项都匹配，则只有一个审计项会在审计日志中生成一条审计记录。

举例说明

例 1 通过语句级审计对所有用户登录时失败操作进行审计。

```
SP_AUDIT_STMT('CONNECT', 'NULL', 'FAIL');
```

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');
```

例 4 对 SYSDBA 对表 USER1.TB1 进行的添加和修改的成功操作进行审计。

```
SP_AUDIT_OBJECT('INSERT', 'SYSDBA', 'USER1', 'TB1', 'SUCCESSFUL');
SP_AUDIT_OBJECT('UPDATE', 'SYSDBA', 'USER1', 'TB1', 'SUCCESSFUL');
```

15.11 审计取消语句

DM 支持对审计数据进行分析，以便从海量的审计数据中快速提取出审计员关心的，可以说明安全问题的数据。

DM 的审计分析处理采用服务器端、客户端相结合的方式：在服务器端存放审计规则，具体分析流程置于客户端。

语法格式

```
参照附录 3 中审计函数
SP_NOAUDIT_STMT 和 SP_NOAUDIT_OBJECT
```

使用说明

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

举例说明

例 1 取消对所有用户登录时失败操作进行审计。

```
SP_NOAUDIT_STMT('CONNECT', 'NULL', 'FAIL');
```

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进行审计。

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

例 4 取消对 SYSDBA 对表 USER1.TB1 进行的添加和修改的成功操作进行审计。

```
SP_NOAUDIT_OBJECT('INSERT', 'SYSDBA', 'USER1', 'TB1', 'SUCCESSFUL');
SP_NOAUDIT_OBJECT('UPDATE', 'SYSDBA', 'USER1', 'TB1', 'SUCCESSFUL');
```

15.12 创建审计实时侵害检测规则

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侵害检测规则，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。

语法格式

```
参照附录 3 中审计函数
SP_CREATE_AUDIT_RULE
```

实时侵害检测规则使用说明：

必须将审计开关设置为 ENABLE_AUDIT ＝2，才能使用审计实时侵害检测规则。

举例说明

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);
```

15.13 删除审计实时侵害检测规则

当数据库审计员认为对某些审计实时侵害检测规则不需要时，可用 DM 提供的取消规

则语句将规则清除掉。同样，取消审计设置也是立即生效的，不需系统重新启动。

语法格式

参照附录 3 中审计函数

```
SP_DROP_AUDIT_RULE
```

使用说明

必须将审计开关设置为 ENABLE_AUDIT ＝2，才能使用。

举例说明

例 删除审计侵害检测规则 DANGEROUS_SESSION。

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

15.14 加密引擎

DM 系统中内置了常用的 DES，AES，RC4 等加密算法供用户使用，以此来保护数据的安全性。然而在有些特殊的环境下，这些加密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加密算法，或强度更高的加密算法。DM 的加密引擎功能则可以满足这样的需求。

加密引擎没有提供相应的 SQL 语法，而是需要用户编程实现 DM 预定义的加密接口。详细使用方法见《DM 程序员手册》的第 10 章《DM 加密引擎接口编程指南》。

第 16 章 DM 备份还原

DM 支持通过 SQL 语句对数据库执行联机备份还原操作。本章仅介绍这些操作的语法。关于备份还原概念与特点等内容，请参考《DM 系统管理员手册》中的相关章节。

16.1 备份数据库

语法格式

BACKUP DATABASE [FULL|BTREE|INCREMENT WITH BACKUPDIR '<备份目录>'{,'<备份目录>'}] TO <备份名> [BAKFILE '<备份路径>'] [BACKUPINFO '<备份描述>'] [MAXSIZE <限制大小>] [IDENTIFIED BY <密钥>[WITH ENCRYPTION<TYPE>]][COMPRESSED] [PARALLEL '<并行映射文件路径>'];

参数

1．FULL|INCREMENT|BTREE　备份类型，FULL 表示完全备份，INCREMENT 表示增量备份，BTREE 表示 B 树备份；

2．<备份目录>　基础备份所在的目录，最大长度为 256 个字节；

3．<备份名>　备份的名称，在 DMDBMS 中以此标识不同的备份，最大长度为 128 个字节；

4．<备份路径>　备份文件存放的完整路径，最大长度为 256 个字节；

5．<备份描述>　备份的描述信息；

6．<限制大小>　最大备份文件大小，最小值为 16M，最大值 2G；

7．<密钥>　备份加密通过使用 IDENTIFIED　BY 来指定密码；

8．WITH ENCRPYTION <TYPE>　用来指定加密类型，0 表示不加密，1 表示简单加密，2 表示复杂加密；

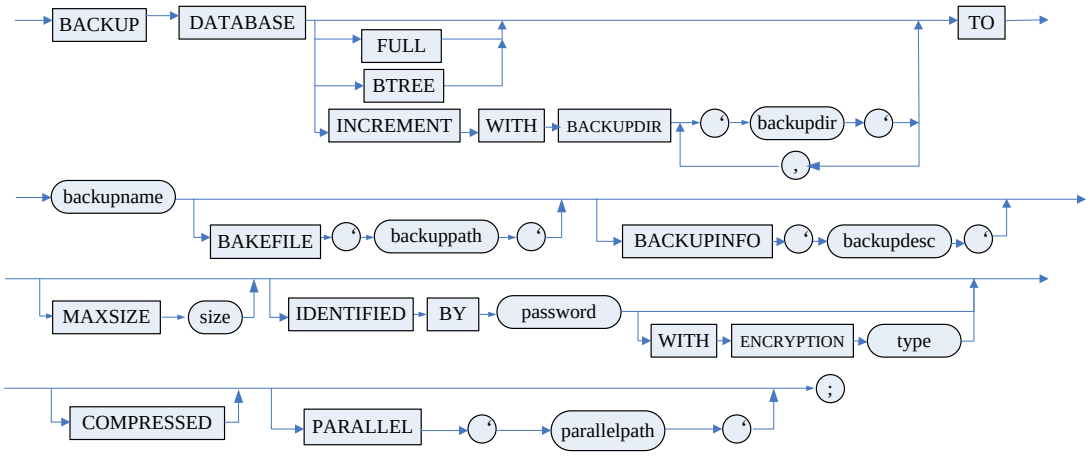
9．COMPRESSED　用来指定是否压缩。如果使用，则表示压缩，否则表示不压缩；

10．PARALLEL　用来指定并行映射文件的完整路径。

11．<并行映射文件路径>　并行映射文件的完整路径，最大长度为 256 个字节。

备份成功后会在备份默认目录下生成备份文件，若没有指定备份文件名，则系统自动生成以“数据库名_日期.bak”格式的备份文件，通过该备份文件可以进行数据库还原。

图例



语句功能

对数据库进行联机备份，生成备份文件。

使用说明

1．备份路径为可选，如果不设置备份文件名，则系统会在默认备份目录中生成备份文件，备份文件名生成规则是数据库名加时间；

2．当备份数据超过限制大小时，会生成新的备份文件，新的备份文件名是初始文件名后加文件编号；

3. 系统处于归档模式下时，才允许进行数据库备份；

4. 并行备份时，并行映射文件的扩展名是 parallel，如果不写扩展名，则系统默认是 parallel 类型的文件。

5. 关于映射文件格式请参考《DM 系统管理员手册》中的相关章节。

举例说明

例 1 假设现在启动的是数据库 BOOKSHOP 的实例，对数据库进行完全备份，备份名为 BOOKSHOP_BAK1。

```
BACKUP  DATABASE  FULL TO BOOKSHOP_BAK1;
```

例 2 假设现在启动的是数据库 BOOKSHOP 的实例，对数据库进行完全备份，备份名为 BOOKSHOP_BAK2，并对备份数据压缩。

```
BACKUP DATABASE FULL TO BOOKSHOP_BAK2 COMPRESSED;
```

例 3 假设现在启动的是数据库 BOOKSHOP 的实例，对数据库进行 B 树备份，备份名为 BOOKSHOP_BAK3，对备份数据加密，加密类型为简单加密，加密密码为 ABCDEF。

```
BACKUP DATABASE BTREE TO BOOKSHOP_BAK3 IDENTIFIED BY ABCDEF WITH ENCRYPTION 1;
```

例 4 假设现在启动的是数据库 BOOKSHOP 的实例，对数据库进行并行备份，备份名为 BOOKSHOP_BAK4，映射文件为 c:\BOOKSHOP_BAK4.parallel。

```
BACKUP DATABASE TO BOOKSHOP_BAK4 PARALLEL ‘c:\ BOOKSHOP_BAK4.parallel’;
```

16.2 备份表空间

语法格式

```
BACKUP TABLESPACE <表空间名>  [FULL| INCREMENT WITH BACKUPDIR ‘<备份目录>’{,‘<备份目录>’}] TO <备份名> [BAKFILE ‘<备份路径>’] [BACKUPINFO ‘<备份描述>’] [MAXSIZE <限制大小>] [IDENTIFIED BY <密钥>[WITH ENCRYPTION<TYPE>]] [COMPRESSED];
```

参数

1．<表空间名>　　需要备份的表空间的名称，只能备份用户表空间；

2．FULL|INCREMENT　备份类型，FULL 表示完全备份，INCREMENT 表示增量备份；

3. <备份目录>　　基础备份所在的目录；

4．<备份名>　　备份的名称，在 DMDBMS 中以此标识不同的备份；

5．<备份路径>　　备份文件存放的完整路径；

6．<备份描述>　　备份的描述信息；

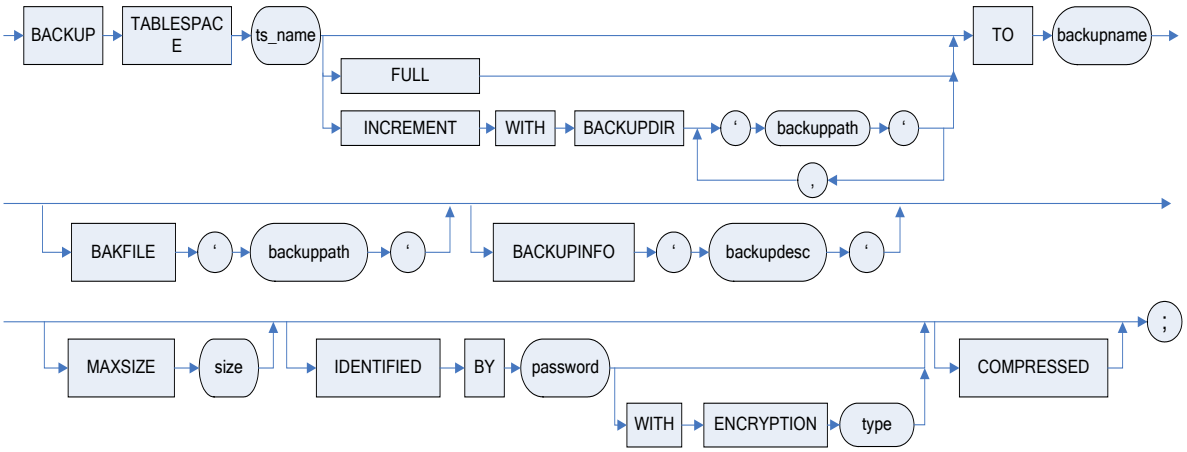
7．<限制大小>　　最大备份文件大小，最小值为 16M，最大值 2G；

8．<密钥>　　备份加密通过使用 IDENTIFIED　BY 来指定密码；

9．<TYPE>　　用来指定加密类型，0 表示不加密，1 表示简单加密，2 表示复杂加密；

10．COMPRESSED　　用来指定是否压缩。如果使用，则表示压缩，否则表示不压缩。

图例



语句功能

联机备份表空间，生成备份文件。

使用说明

1．备份路径为可选，如果不设置备份文件名，则系统会在 INI 参数 BAK_PATH 指定的默认备份目录中生成备份文件，备份文件名生成规则是数据库名加时间；

2．当备份数据超过限制大小时，会生成新的备份文件，新的备份文件名是初始文件名后加文件编号；

3. 系统处于归档模式下时，才允许进行表空间备份。

举例说明

例 1 对数据库增加表空间 FG，对其进行完全备份，备份名为 FG_BAK1。

```
CREATE TABLESPACE FG DATAFILE 'C:\WINNT\SYSTEM32\DAMENG\FG.DBF' SIZE 64;
BACKUP TABLESPACE FG FULL TO FG_BAK1 BAKFILE 'C:\FULL.BAK';
```

例 2 对例 1 中创建的表空间 FG 进行增量备份，备份名为 FG_BAK1_INCR，基础备份所在目录为 C 盘根目录。

```
BACKUP TABLESPACE FG INCREMENT WITH BACKUPDIR 'C:\' TO FG_BAK1_INCR BAKFILE 'C:\INCR_BAK.BAK';
```

16.3 还原表空间

语法格式

```
RESTORE TABLESPACE <表空间名> FROM '<备份路径>' | BACKUP <备份文件名> [ INDENTIFIED BY <密码>] [ WITH BACKUPDIR '<备份目录>'{,'<备份目录>'}][WITH ARCHIVEDIR '归档目录'{,'归档目录'}][DBFILE <文件 id> TO '<文件路径>' {，DBFILE <文件 id> TO '<文件路径>'}];
```

参数

1．<表空间名>　　需要还原的表空间名称；

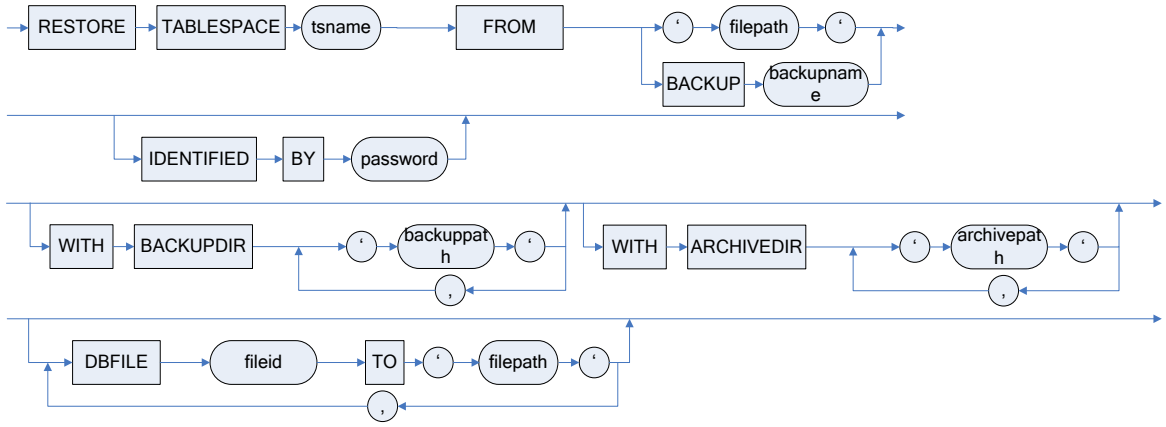
2．<备份路径>　　备份文件存放的完整路径；

3．<备份文件名>　备份的名称，在 DMDBMS 中以此标识不同的备份；

4. <密码>　　加密备份表空间时，用户设置的密码；

5. <备份目录>　　搜集备份文件的目录；
6. <归档目录>　　还原时，搜集归档文件的目录；
7. <文件 ID>　　需要重新设置文件路径文件对应的文件号；
8. <文件路径>　　设置还原后，数据文件的路径。

图例



语句功能

使用备份文件链接还原表空间。

使用说明

1．文件路径应该是完整的备份文件存储路径（包括文件名本身）；

2．还原前应该先将还原的表空间脱机，只能还原用户表空间；

3．还原既可以从指定的备份文件进行，也可以指定备份名进行还原，这取决于在语句中指定文件路径还是指定备份名；

4. 当使用备份名进行还原时，如果 INI 参数 BAK_PATH 指定的目录下有多个同名备份，则使用找到的第一个指定名字的备份进行还原；

5．该功能在配置本地归档后才起作用。

举例说明

例 1 对表空间 FG 使用完全备份 FG_BAK1 进行还原。

```
ALTER TABLESPACE FG OFFLINE;
RESTORE TABLESPACE　FG　FROM　'C:\FULL.BAK';
ALTER TABLESPACE FG ONLINE;
```

例 2 对表空间 FG 使用增量备份 FG_BAK1_INCR 进行还原。

```
ALTER TABLESPACE FG OFFLINE;
RESTORE TABLESPACE　FG　FROM　'C:\INCR_BAK.BAK';
ALTER TABLESPACE FG ONLINE;
```

16.4 备份用户表

语法格式

```
BACKUP TABLE <表名>　TO <备份名> [BAKFILE '<备份路径>'] [BACKUPINFO '<备份描述>'] [MAXSIZE <限制大小>] [IDENTIFIED BY <密钥>[WITH ENCRYPTION <TYPE>]] [COMPRESSED];
```

参数

1．<表名>　　需要备份的用户表的名称；

2．<备份名>　　备份的名称，在 DMDBMS 中以此标识不同的备份；

3．<备份路径>　　备份文件存放的完整路径；

4．<备份描述>　　备份的描述信息；

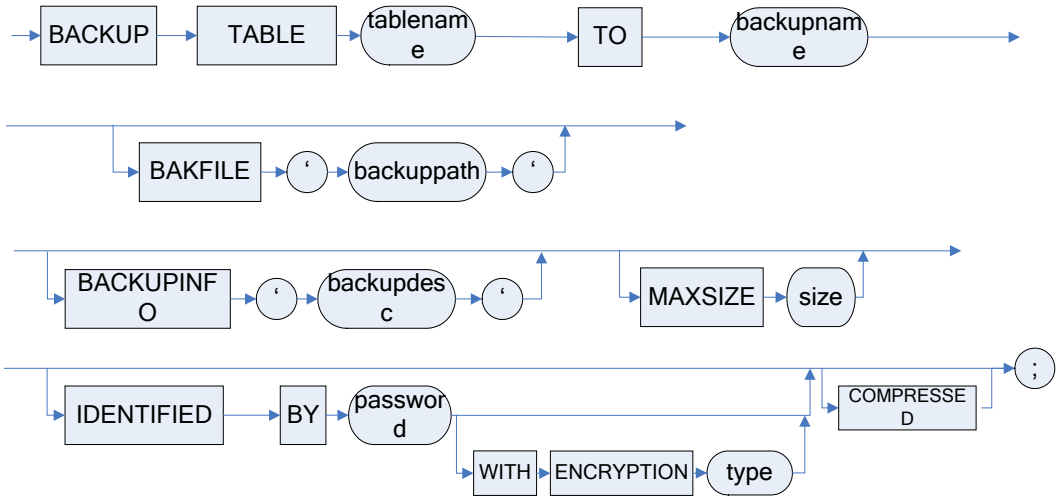
5．<限制大小>　　最大备份文件大小，最小值为 16M，最大值 2G；

6．<密钥>　　备份加密通过使用 IDENTIFIED　BY 来指定密码；

7．WITH ENCRPYTION <TYPE>　用来指定加密类型，0 表示不加密，1 表示简单加密，2 表示复杂加密；

8．COMPRESSED　用来指定是否压缩。如果使用，则表示压缩，否则表示不压缩。

图例



语句功能：

备份用户表。

使用说明

1．备份路径为可选，如果不设置备份文件名，则系统会在默认备份目录中生成备份文件，备份文件名生成规则是数据库名加时间；

2．当备份数据超过限制大小时，会生成新的备份文件，新的备份文件名是初始文件名后加文件编号；

3. 不支持临时表的备份；

4. 不支持垂直分区表子表的备份；

5. 不支持系统表的备份；

6. 对水平分区表子表进行备份还原时，用户必须保证表版本信息的一致性。

举例说明

例 对 BOOKSHOP 数据库 PERSON 模式下的 ADDRESS 表进行备份，备份名为 ADDR_BAK1。

```
BACKUP TABLE PERSON.ADDRESS TO ADDR_BAK1;
```

16.5 还原用户表

语法格式

```
RESTORE TABLE　FROM　'<文件路径>' | BACKUP <备份名> [IDENTIFIED BY <密钥>]
```

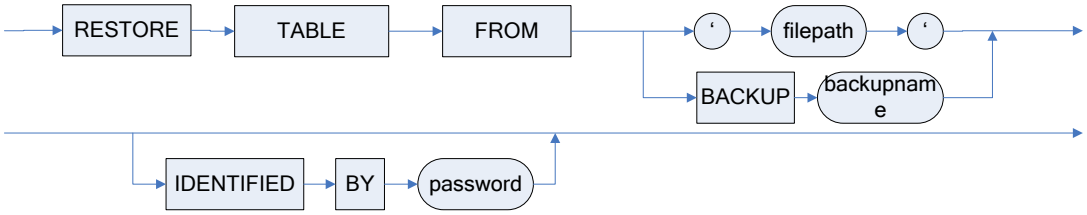
参数

1．<文件路径>　　表备份的备份文件路径；

2. <备份名>　　备份的名称，在 DMDBMS 中以此标识不同的备份；

3. <密钥>　　执行加密备份时，用户设置的密钥，用该密钥恢复。

图例



语句功能：

还原用户表。

使用说明

1. 文件路径应该是完整的备份文件存储路径（包括文件名本身）；

2. 当两表之间存在引用约束时，如在还原时其中一个表或两个表都不存在（备份后删除某表，或还原到一个新的表空间等），需要重新执行建表语句时，引用约束将丢失。

3. 当备份完成后，删除模式，执行还原时，首先创建表所在的模式，然后再还原表对象。如果此时删除的是用户而不是模式，执行还原时，同样，首先创建表所在的模式，然后再还原表对象，而用户不会再创建。

4. 如果备份的是用户自定义的表空间中表对象，执行还原时，表对象会还原到MAIN表空间，而非用户自定义的表空间。

举例说明

例 利用备份 ADDR_BAK1 还原用户表 ADDRESSS。

```
RESTORE TABLE FROM BACKUP ADDR_BAK1;
```

第 17 章 同义词

同义词(Synonym)让用户能够为数据库的一个模式下的对象提供别名。同义词通过掩盖一个对象真实的名字和拥有者，并且对远程分布式的数据库对象给予了位置透明特性以此来提供了一定的安全性。同时使用同义词可以简化复杂的 SQL 语句。同义词可以替换模式下的表、视图、序列、函数、存储过程等对象。

17.1 创建同义词

语法格式

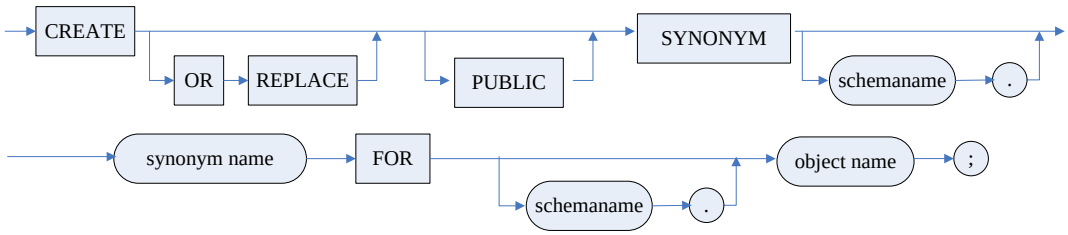
```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [<模式名>.]<同义词名>  FOR [<模式名>.]<对象名>
```

参数

1．<同义词名>　　指被定义的同义词的名字；

2．<对象名>　　指示同义词替换的对象。

图例



语句功能

创建一个同义词。

使用说明

1．同义词分为全局同义词(PUBLIC SYNONYM)和非全局同义词。用户在自己的模式下创建同义词，必须有 CREATE SYNONYM 权限。用户要创建全局同义词(PUBLIC SYNONYM)，必须有 CREATE PUBLIC SYNONYM 权限；

2．全局同义词创建时不能指定同义词的模式名限定词，它能够被所有用户使用，使用时不需要加任何模式限定名。非全局同义词被非同义词所属模式拥有者引用需要在前面加上模式名；公有同义词和私有同义词，可以具有相同的名字；

3．用户使用 SQL 语句对某个对象进行操作，那么解析一个对象的顺序，首先是查看模式内是否存在该对象，然后再查看模式内的同义词(非全局同义词)，最后才是全局同义词。例如，用户 OE 和 SH 在他们的模式下都有一个表叫 customer，SYSDBA 为 OE 模式下的 customer 表创建了一个全局同义词 customer_syn，SYSDBA 为 SH 模式下的 customer 表创建了一个私有同义词 customer_syn，如果用户 SH 查询：SELECT COUNT(*) FROM customer_syn，则此时返回的结果为 SH.CUSTOMER 下的行数，而如果需要访问 OE 模式下的 CUSTOMER 表，则必须在前面加模式名(此时全局同义词 CUSTOMER 失效)：SELECT COUNT(*) FROM OE.customer_syn；

4．如果创建时候没有指定 REPLACE 语法要素，则不允许创建同名同类型同义词(不同类型则可以，这里的不同类型指的是 PUBLIC 和非 PUBLIC)；

5．同义词创建时，并不会检查他所指代的同义词对象是否存在，用户使用该同义词时候，如果不存在指代对象或者对该指代对象不拥有权限，则会报错。

举例说明

例 1 用户 A 对 A 模式下的表 T1 创建同义词。

在 A 模式下建立表 T1。

```
CREATE TABLE "T1" ("ID" INTEGER, "NAME" VARCHAR(50), PRIMARY KEY("ID"));
INSERT INTO "A"."T1" ("ID", "NAME") VALUES (1, '张三');
INSERT INTO "A"."T1" ("ID", "NAME") VALUES (2, '李四');
```

对 A 模式下的表 T1 创建同义词。

```
CREATE SYNONYM S1 FOR A.T1;
```

如果用户 A 想查询 T1 表的行数，可以通过如下语句来获得结果：

```
SELECT COUNT(*) FROM A.S1;
```

17.2 删除同义词

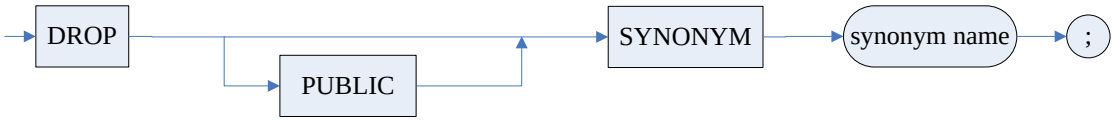
语法格式

```
DROP [PUBLIC] SYNONYM <同义词名>
```

参数

<同义词名> 指被定义的同义词的名字。

图例



语句功能

删除一个同义词。

使用说明

1． 必须是 DBA 或者拥有此同义词的用户才能删除改同义词；

2． 如果要删除公有同义词，则必须要指定 PUBLIC，因为在达梦数据库中，公有同义词和私有同义词可以同名，所以如果不指定，则删除的是当前模式下的同名同义词；

3． 删除公有同义词，需要指定 PUBLIC，而删除私有同义词，则不能指定，否则报错；如果删除当前模式下的同义词，可以不指定模式名，如果删除其它模式下的同义词，需要指定相应的模式名，否则报错。

举例说明：

例 1 删除模式 A 下的同义词 S1。

```
DROP SYNONYM A.S1.
```

例 2 删除公有同义词 S2。

```
DROP PUBLIC SYNONYM S2.
```

第 18 章 外部链接

外部链接对象（LINK）是 DM 中的一种特殊的数据库实体对象，它记录了远程数据库的连接和路径信息，用于建立与远程数据的联系。通过多台数据库主机间的相互通讯，用户可以透明地操作远程数据库的数据，使应用程序看起来只有一个大型数据库。用户远程数据库中的数据请求，都被自动转换为网络请求，并在相应结点上实现相应的操作。用户可以建立一个数据库链接，以说明一个对象在远程数据库中的访问路径。这个链接可以是公用的（数据库中所有用户使用），也可以是私有的（只能被某个用户使用）。

DM7 的外部链接仅支持 DM 数据库，不支持异构数据库；支持对外部链接表的查询，不支持增删改操作。

18.1 创建外部链接

创建一个外部链接。

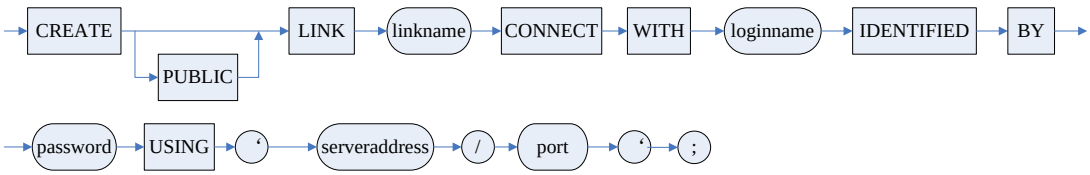
语法格式

```
CREATE [PUBLIC] LINK <外部链接名> CONNECT WITH <登录名> IDENTIFIED BY
<登录口令> USING  '<连接串> ';
<连接串> ::=<外部链接串>
<外部链接串>::=服务器地址/端口
```

参数

1．<外部链接名> 数据库链接的名称；
2．<登录名> 登录名称；
3．<登录口令> 登录口令；
4．PUBLIC 此链接对象是否能够被创建者之外的用户引用。

图例



语句功能

创建一个外部链接。

使用说明

1．连接串的形式为<IP 地址>/<端口号>；
2．目前只支持对 DM 数据库创建 LINK；
3．端口号为 DM 外部链接服务器的 MAL 配置中的端口号。
4．只支持普通用户，不支持 SSL 和 Kerberos 认证。
5．支持创建 LINK 链接自身。
6．支持在 CREATE SCHEMA 中 CREATE LINK，但是不支持 CREATE PUBLIC LINK。
7．只有 DBA 和具有 CREATE LINK 权限的用户可以创建外部链接。
8．外部链接是在 MAL 的基础上进行实现的，必须首先配置 MAL，才能使用 LINK。

举例说明

创建一个连接到 IP 地址为 192.168.0.31，端口号为 5369 的 MAL 站点的外部链接，登录到此站点使用的用户名为 USER1，密码为 AAAAAA。

```
CREATE LINK LINK1 CONNECT WITH USER1 IDENTIFIED BY AAAAAA USING '192.168.0.31/5369';
```

18.2 删除外部链接

删除一个外部链接。

语法格式

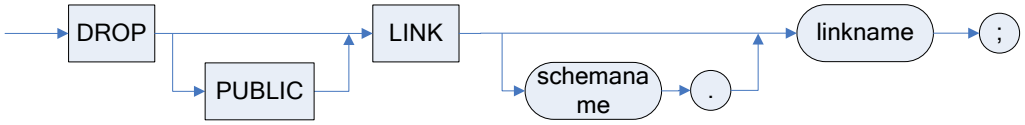
```
DROP [PUBLIC] LINK [<模式名>.]<外部链接名>;
```

参数

1．<模式名> 指明被操作的外部链接属于哪个模式，缺省为当前模式；

2．<外部链接名> 指明被操作的外部链接的名称。

图例



语句功能

删除一个外部链接。

使用说明

只有链接对象的创建者和 DBA 拥有该对象的删除权限。

举例说明

删除外部链接 LINK1。

```
DROP LINK LINK1;
```

18.3 使用外部链接

使用外部链接进行查询的语法格式与普通格式基本一致，唯一的区别在于指定外部链接表时需要使用如下格式作为表或视图的引用：

```
<TABLENAME | VIEWNAME > LINK 链接名
```

关键字 LINK 可以用符号‘@’代替；

指明模式下的外部链接，查询之前需要 SET SCHEMA 为创建外部链接时指定的模式。

举例说明

使用外部链接查询 LINK1 上的远程表 SYSOBJECTS。

```
SELECT * FROM SYSOBJECTS LINK LINK1;
```

本语句等同于：

```
SELECT * FROM SYSOBJECTS@LINK1;
```

第 19 章 闪回查询

当系统 INI 参数 ENABLE_FLASHBACK 置为 1 时，闪回功能开启，可以进行闪回查询。

19.1 闪回查询

闪回查询的语法，是在数据查询语句（参考第 4 章）的基础上，为 FROM 子句增加了闪回查询子句。

语法格式

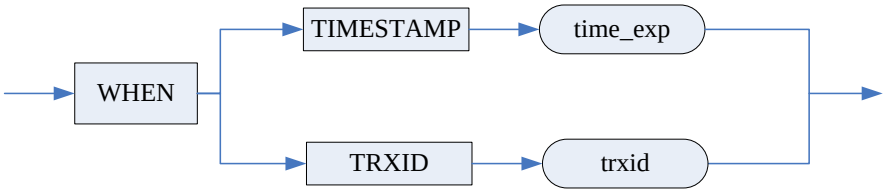
```
SELECT <选择列表>
FROM [<模式名>.]<基表名> [<闪回查询子句>]<相关名>
[<WHERE 子句>]
[<CONNECT BY 子句>]
[<GROUP BY 子句>]
[<HAVING 子句>]
[ORDER BY 子句];

<闪回查询子句>::=WHEN <TIMESTAMP time_exp> | <TRXID trxid>
```

参数

1．time_exp　一个日期表达式，一般用字符串方式表示

2．　trxid　指定事务 ID 号

图例



语句功能

用户通过闪回查询子句，可以得到指定表过去某时刻的结果集。指定条件可以为时刻，或事务号。

使用说明

1．闪回查询只支持普通表（包括加密表与压缩表）、临时表和 list 表，不支持水平分区表、垂直分区表、列存储表、外部表与视图；

2．闪回查询中 trxid 的值，一般需要由闪回版本查询（见下节）的伪列来确定。实际使用中多采用指定时刻的方式。

举例说明

例 1 闪回查询特定时刻的 PERSON_TYPE 表。

查询 PERSON_TYPE 表。

```
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如表 19.1.1 所示。

表 19.1.1

PERSON_TYEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

在 2012-01-01 12:22:49 时刻插入数据，并提交。

```
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('防损员');
INSERT INTO PERSON.PERSON_TYPE(NAME) VALUES('保洁员');
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如下表 19.1.2 所示。

表 19.1.2

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员
6	保洁员

使用闪回查询取得 2012-01-01 12:22:45 时刻的数据。此时刻在插入数据的操作之前，可见此时的结果集不应该有 2012-01-01 12:22:49 时刻插入的数据。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TIMESTAMP '2012-01-01 12:22:45';
```

结果集如表 19.1.3 所示。

表 19.1.3

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表

在 2012-01-01 12:23:29 时刻删除数据，并提交。

```
DELETE FROM PERSON.PERSON_TYPE WHERE ID > 5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如表 19.1.4 所示。

表 19.1.4

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员

使用闪回查询得到删除前的数据。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TIMESTAMP '2012-01-01 12:23:00';
```

结果集如表 19.1.5 所示。

表 19.1.5

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	防损员
6	保洁员

例 2 闪回查询指定 TRXID 的 PERSON_TYPE 表。

要获得 TRXID 信息，需要通过闪回版本查询的伪列 VERSIONS_ENDTRXID。（详细内容见下节“闪回版本查询”）

在 2012-01-01 12:24:05 时刻修改数据，并提交。

```
UPDATE PERSON.PERSON_TYPE SET NAME='保安员' WHERE PERSON_TYPEID=5;
COMMIT;
UPDATE PERSON.PERSON_TYPE SET NAME='收银员' WHERE PERSON_TYPEID=5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如表 19.1.6 所示。

表 19.1.6

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	收银员

进行闪回版本查询，确定 TRXID。

```
SELECT VERSIONS_ENDTRXID, NAME FROM PERSON.PERSON_TYPE VERSIONS BETWEEN TIMESTAMP '2012-01-01 12:24:00' AND SYSDATE;
```

得到结果集如表 19.1.7 所示。

表 19.1.7

VERSION_ENDTRXID	NAME
NULL	采购经理
NULL	采购代表
NULL	销售经理
NULL	销售代表
323	收银员
322	保安员
NULL	防损员

根据 TRXID 确定版本。

```
SELECT * FROM PERSON.PERSON_TYPE WHEN TRXID 322;
```

结果集如表 19.1.8 所示。

表 19.1.8

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	保安员

19.2 闪回版本查询

语法格式

```
SELECT <选择列表>
FROM [<模式名>.]<基表名> [<闪回版本查询子句>]<相关名>
[<WHERE 子句>]
[<CONNECT BY 子句>]
[<GROUP BY 子句>]
[<HAVING 子句>]
[ORDER BY 子句];

<闪回版本查询子句>::=VERSIONS BETWEEN <TIMESTAMP time_exp1 AND time_exp2> | <TRXID trxid1 AND trxid2>
```

参数

1．time_exp 日期表达式，一般用字符串方式表示。time_exp1 表示起始时间，time_exp2 表示结束时间

2. trxid 指定事务 ID 号，整数表示。trxid1 表示起始 trxid，trxid2 表示结束 trxid

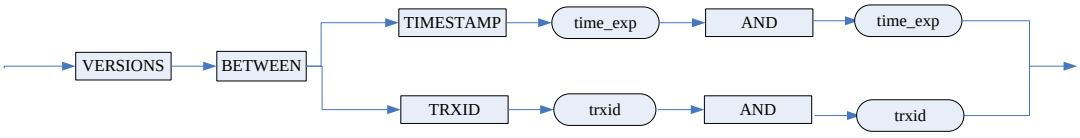
使用说明

1. 闪回版本查询支持普通表（包括加密表与压缩表）、临时表和 list 表，不支持水平分区表、垂直分区表、列存储表、外部表与视图；
2. 支持如表 19.2.1 所示伪列，作为闪回版本查询的辅助信息。

表 19.2.1 闪回版本查询支持的伪列

伪列	说明
VERSIONS_START{TRXID|TIME}	起始 TRXID 或时间戳
VERSIONS_END{TRXID|TIME}	提交 TRXID 或时间戳。如果该值为 NULL，表示行版本仍然是当前版本
VERSIONS_OPERATION	在行上的操作（I=Insert,D=Delete,U=Update）

图例



语句功能

用户通过闪回版本查询子句，可以得到指定表过去某个时间段内，事务导致记录变化的全部记录。指定条件可以为时刻，或事务号。

举例说明

例 1 闪回版本查询指定时间段内，PERSON_TYPE 表的记录变化。

在 2012-01-01 12:24:05 时刻修改数据，并提交。

```
UPDATE PERSON.PERSON_TYPE SET NAME='保安员' WHERE PERSON_TYPEID=5;
COMMIT;
UPDATE PERSON.PERSON_TYPE SET NAME='收银员' WHERE PERSON_TYPEID=5;
COMMIT;
SELECT * FROM PERSON.PERSON_TYPE;
```

结果集如表 19.2.2 所示。

表 19.2.2

PERSON_TYPEID	NAME
1	采购经理
2	采购代表
3	销售经理
4	销售代表
5	收银员

进行闪回版本查询，获得指定时间段内变化的记录。

```
SELECT VERSIONS_ENDTRXID, NAME FROM PERSON.PERSON_TYPE VERSIONS BETWEEN TIMESTAMP '2012-01-01 12:24:00' AND SYSDATE;
```

得到结果集如表 19.2.3 所示。

表 19.2.3

VERSION_ENDTRXID	NAME
NULL	采购经理
NULL	采购代表
NULL	销售经理
NULL	销售代表
323	收银员
322	保安员
NULL	防损员

19.3 闪回事务查询

闪回事务查询提供系统视图 V$FLASHBACK_TRX_INFO 供用户查看在事务级对数据库所做的更改。根据视图信息，可以确定如何还原指定事务或指定时间段内的修改。

使用说明

系统视图名为 V$FLASHBACK_TRX_INFO，定义如表 19.3.1 所示。

表 19.3.1

列名	数据类型	说明
START_TRXID	BIGINT	事务中第一个 DML 的 TRXID
START_TIMESTAMP	TIMESTAMP	事务中第一个 DML 的时间戳
COMMIT_TRXID	BIGINT	提交事务的 TRXID
COMMIT_TIMESTAMP	TIMESTAMP	提交事务时的时间戳
LOGON_USER	VARCHAR(128)	拥有事务的用户

UNDO_CHANGE#	BIGINT	记录修改顺序序号
OPERATION	VARCHAR(32)	DML 操作类型。 D：删除；U：修改；I：插入；N：更新插入（专门针对 CLUSTER PRIMARY KEY 的插入）； C：事务提交；P：预提交记录；O：default
TABLE_NAME	VARCHAR(256)	DML 修改的表
TABLE_OWNER	VARCHAR(128)	DML 修改表的拥有者
ROW_ID	VARBINARY(32)	DML 修改行的 ROWID
UNDO_SQL	VARCHAR(4000)	撤销 DML 操作的 SQL 语句

举例说明

例 1 查询指定时间之后的事务信息，可为闪回查询操作提供参考。

```
SELECT * FROM V$FLASHBACK_TRX_INFO WHERE COMMIT_TIMESTAMP > '2012-01-01 12:00:00';
```

第 20 章 系统包

达梦数据库还提供了达梦特有的 DBMS_DBG、DBMS_GEO 系统包和兼容 ORACLE 数据库的 DBMS_ALERT、DBMS_OUTPUT、UTL_FILE 和 UTL_MAIL 等系统包功能。用户可以通过调用系统过程 SP_INIT_DBG_SYS(create_flag) 创建 DBMS_DBG 包；SP_INIT_GEO_SYS(create_flag)创建 DBMS_GEO 包； SP_INIT_JOB_SYS(create_flag)创建 DBMS_JOB 包；SP_CREATE_SYSTEM_PACKAGES(create_flag)一次性创建其余的所有包，系统过程的具体介绍参考附录 3。

以下详细介绍这各种包的相关内容。

20.1 DBMS_DBG 包

达梦数据库为用户提供了 PL/SQL 调试工具 DMDBG。DMDBG 可调试在 DISQL 中直接执行的非 DDL 语句或语句块，以便程序开发人员定位 PL/SQL 中存在的错误。

用户只要提前调用系统过程 SP_INIT_DBG_SYS(1)，创建好调试所需要的包，就可以使用 DMDBG 来调用 PL/SQL 中的过程或函数。

```
SP_INIT_DBG_SYS(1);
```

20.2 DBMS_GEO 包

DBMS_GEO 系统包实现了 SFA 标准（《OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option》）中规定的 SQL 预定义 schema，基于 SQL UDT（自定义数据类型）的空间数据类型和空间数据类型的初始化，以及针对空间数据类型的几何体计算函数。

20.2.1 数据类型

达梦的空间数据类型是以类的方式进行体现的，具体有：

ST_Geometry：最基本的几何体 所有其他几何体的基类。

ST_Point：点几何体。

ST_Curve，ST_LineString：线几何体。ST_Curve 是抽象类，ST_LineString 是 ST_Curve 可实例化的子类。

ST_Surface，ST_Polygon：面几何体。ST_Surface 是抽象类，ST_Polygon 是 ST_Surface 可实例化的子类。

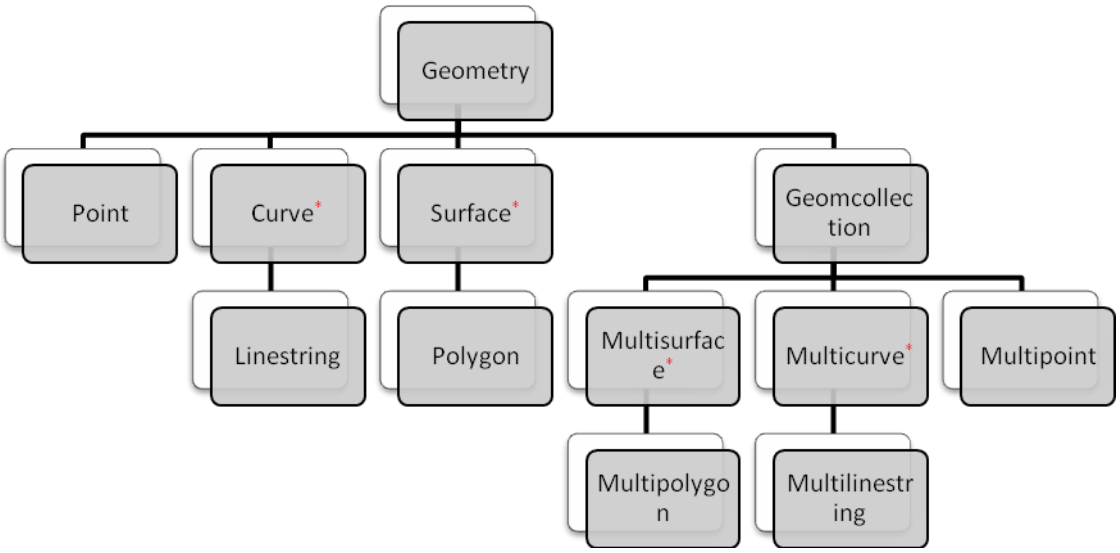
ST_GeomCollection：几何体集合。

ST_Multipoint：点集合。

ST_Multicurve，ST_Multilinestring：线集合。ST_Multicurve 是抽象类，ST_Multilinestring 是 ST_Multicurve 可实例化的子类。

ST_Multisurface，ST_Multipolygon：多边形集合。ST_Multisurface 是抽象类，ST_Multipolygon 是 ST_Multisurface 可实例化的子类。

相关类型的继承关系如下图所示（带*号的表示抽象父类）：



如下详细介绍各数据类型：

1. ST_Geometry

基础几何体类定义：

```
CREATE OR REPLACE CLASS ST_GEOMETRY AS
SRID                INT;
GEO_DIM             INT;
GEO_TYPE            VARCHAR(20);
GEO_ISEMPTY         INT;
GEO_ISSIMPLE        INT;
GEO_WKT             CLOB;
END;
```

类成员说明：

srid：空间参考坐标系信息；

geo_dim：维度信息；

geo_type：集合体类型信息；

geo_isempty：是否为空；

geo_issimple：是否为简单集合体；

geo_wkt：集合体的 wkt 信息。

2. ST_Point

点类定义：

```
CREATE OR REPLACE CLASS ST_POINT AS
GEO                 ST_GEOMETRY;
POINT_X         DOUBLE;
POINT_Y         DOUBLE;
END;
```

类成员说明：

geo：父类；

point_x：X 坐标；

point_y：Y 坐标。

3. ST_Curve

抽象线类定义：

```
CREATE OR REPLACE CLASS ST_CURVE AS
GEO                 ST_GEOMETRY;
START_POINT         ST_POINT;
END_POINT           ST_POINT;
IS_RING         INT;
LENGTH              DOUBLE;
IS_CLOSED           INT;
```

```
END;
```

类成员说明：

geo：父类；

start_point：起点；

end_point：终点；

is_ring：是否为环；

length：长度；

is_closed：是否闭合。

4. ST_Linestring

线类定义：

```
CREATE OR REPLACE CLASS ST_LINESTRING AS
CURVE          ST_CURVE;
NUM_POINTS     INT;
END;
```

类成员说明：

curve：父类；

num_points：点数。

5. ST_Surface

基础面类定义：

```
CREATE OR REPLACE CLASS ST_SURFACE AS
GEO                ST_GEOMETRY;
CENTROID           ST_POINT;
POINT_ON           ST_POINT;
AREA               DOUBLE;
END;
```

类成员说明：

geo：父类；

centroid：质心点；

point_on：面上一点；

area：面积。

6. ST_Polygon

面类定义：

```
CREATE OR REPLACE CLASS ST_POLYGON AS
SURFACE        ST_SURFACE;
EXT_RING       ST_LINESTRING;
NUM_INTER_RING INT;
END;
```

类成员说明：

surface：父类；

ext_ring：外环；

num_inter_ring：内环数。

7. ST_GeomCollection

基础几何体集合类定义：

```
CREATE OR REPLACE CLASS ST_GEOMCOLLECTION AS
GEO            ST_GEOMETRY;
NUM_GEOS       INT;
END;
```

类成员说明：

geo：父类；

num_geos：几何体个数。

8. ST_Multipoint

多点类定义：

```
CREATE OR REPLACE CLASS ST_MULTIPOINT AS
GEOS           ST_GEOMCOLLECTION;
END;
```

类成员说明：

geos：父类。

9. ST_MultiCurve

抽象多线类定义：

```
CREATE OR REPLACE CLASS ST_MULTICURVE AS
GEOS            ST_GEOMCOLLECTION;
IS_CLOSED       INT;
LENGTH          DOUBLE;
END;
```

类成员说明：

geos：父类；

is_closed：是否闭合；

length：长度。

10. ST_Multilinestring

多线类定义：

```
CREATE OR REPLACE CLASS ST_MULTILINESTRING AS
MCURVE    ST_MULTICURVE;
END;
```

类成员说明：

mcurve：父类。

11. ST_MultiSurface

抽象多面类定义：

```
CREATE OR REPLACE CLASS ST_MULTISURFACE AS
GEOS            ST_GEOMCOLLECTION;
CENTROID        ST_POINT;
POINT_ON        ST_POINT;
AREA            DOUBLE;
END;
```

类成员说明：

geos：父类；

centroid：质心点；

point_on：面上一点；

area：面积。

12. l) ST_Multipolygon

多面类定义：

```
CREATE OR REPLACE CLASS ST_MULTIPOLYGON AS
MSURFACE        ST_MULTISURFACE;                --父类
END;
```

类成员说明：

msurface：父类。

20.2.2 相关方法

用户在使用 dmgeo 包之前，需要提前调用系统过程 SP_INIT_GEO_SYS(1)，创建好调试所需要的包，就可以使用 dmgeo 来调用 PL/SQL 中的过程或函数了。

```
SP_INIT_GEO_SYS(1);
```

20.2.2.1 几何体构造函数

达梦实现的集合体函数以包中方法的形式提供给用户，包名为 dmgeo，用户可以通过包脚本自行创建，使用如下函数时，必须加上包名 dmgeo。

1. ST_GeoFromText

定义：

```
FUNCTION ST_GEOFROMTEXT(
WKT CLOB,
SRID INT
)RETURN ST_GEOMETRY;
```

功能说明：

根据 wkt 信息和 srid 信息构造空间数据基础类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

2. ST_PointFromText

定义：

```
FUNCTION ST_POINTFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_POINT;
```

功能说明：

根据 wkt 信息和 srid 信息构造点类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

3. ST_LineFromText

定义：

```
FUNCTION ST_LINEFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_LINESTRING;
```

功能说明：

根据 wkt 信息和 srid 信息构造线类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

4. ST_PolyFromText

定义：

```
FUNCTION ST_POLYFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_POLYGON;
```

功能说明：

根据 wkt 信息和 srid 信息构造面类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

5. ST_MPointFromText

定义：

```
FUNCTION ST_MPOINTFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_MULTIPOINT;
```

功能说明：

根据 wkt 信息和 srid 信息构造多点类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

6. ST_MLineFromText

定义：

```
FUNCTION ST_MLINEFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_MULTILINESTRING;
```

功能说明：

根据 wkt 信息和 srid 信息构造多线类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

7. ST_MPolyFromText

定义：

```
FUNCTION ST_MPOLYFROMTEXT (
WKT CLOB,
SRID INT
)RETURN ST_MULTIPOLYGON;
```

功能说明：

根据 wkt 信息和 srid 信息构造多面类。

参数说明：

wkt：几何对象文本描述信息；

srid：空间参考坐标系信息。

8. ST_PointFromWKB

定义：

```
FUNCTION ST_POINTFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_POINT;
```

功能说明：

根据 wkb 信息和 srid 信息构造点类。

参数说明：

wkb：几何对象序列化描述信息；

srid：空间参考坐标系信息。

9. ST_LineFromWKB

定义：

```
FUNCTION ST_LINEFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_LINESTRING;
```

功能说明：

根据 wkb 信息和 srid 信息构造线类。

参数说明：

wkb：几何对象序列化描述信息；

srid：空间参考坐标系信息。

10. ST_PolyFromWKB

定义：

```
FUNCTION ST_POLYFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_POLYGON;
```

功能说明：

根据 wkb 信息和 srid 信息构造面类。

参数说明：

wkb：几何对象序列化描述信息；

srid：空间参考坐标系信息。

11. ST_MPointFromWKB

定义：

```
FUNCTION ST_MPOINTFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_MULTIPOINT;
```

功能说明：

根据 wkb 信息和 srid 信息构造多点类。

参数说明：

wkb：几何对象序列化描述信息；
srid：空间参考坐标系信息。
12. ST_MLineFromWKB
定义：

```
FUNCTION ST_MLINEFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_MULTILINESTRING;
```

功能说明：
根据 wkb 信息和 srid 信息构造多线类。
参数说明：
wkb：几何对象序列化描述信息；
srid：空间参考坐标系信息。
13. ST_MPolyFromWKB
定义：

```
FUNCTION ST_MPOLYFROMWKB (
WKT CLOB,
SRID INT
)RETURN ST_MULTIPOLYGON;
```

功能说明：
根据 wkb 信息和 srid 信息构造多面类。
参数说明：
wkb：几何对象序列化描述信息。；
srid：空间参考坐标系信息

20.2.2.2 几何信息获取函数

1. ST_Dimension
定义：

```
FUNCTION ST_DIMENSION (
GEO ST_GEOMETRY
)RETURN INT;
```

功能说明：
获取几何体对象的维度。
参数说明：
geo：几何体对象。
返回值：

几何体对象的维度
2. ST_GeometryType
定义：

```
FUNCTION ST_GEOMETRYTYPE (
GEO ST_GEOMETRY
)RETURN VARCHAR(20);
```

功能说明：
获取几何体对象的类型。
参数说明：
geo：几何体对象；
返回值：

几何体对象的类型。
3. ST_AsText
定义：

```
FUNCTION ST_ASTEXT (
GEO ST_GEOMETRY
)RETURN CLOB;
```

功能说明：

获取几何体对象的 wkt 格式文本描述信息。

参数说明：

geo：几何体对象；

返回值：

几何体对象的文本表示形式。

4. ST_AsBinary

定义：

```
FUNCTION ST_ASBINARY (
GEO ST_GEOMETRY
)RETURN CLOB;
```

功能说明：

获取几何体对象的 wkb 格式序列化描述信息。

参数说明：

geo：几何体对象。

返回值：

几何体对象的序列化表示形式。

5. ST_SRID

定义：

```
FUNCTION ST_SRID (
GEO ST_GEOMETRY
)RETURN INT;
```

功能说明：

获取几何体对象的空间参考坐标系信息。

参数说明：

geo：几何体对象。

返回值：

几何体对象的空间参考坐标系。

6. ST_IsEmpty

定义：

```
FUNCTION ST_ISEMPTY (
GEO ST_GEOMETRY
)RETURN INT;
```

功能说明：

获取几何体对象是否为空。

参数说明：

geo：几何体对象。

返回值：

0：非空；1：空。

7. ST_IsSimple

定义：

```
FUNCTION ST_ISSIMPLE (
GEO ST_GEOMETRY
)RETURN INT;
```

功能说明：

获取几何体对象是否为空。

参数说明：

geo：几何体对象。

返回值：

0：复杂几何体； 1：简单几何体。

8. ST_Boundary

定义：

```
FUNCTION ST_BOUNDARY (
GEO ST_GEOMETRY
)RETURN ST_LINESTRING;
```

功能说明：
获取几何体对象的边界。
参数说明：
geo：几何体对象。
返回值：
几何体对象的边界线。
9. ST_Envelope
定义：
```
FUNCTION ST_ENVELOPE (
GEO ST_GEOMETRY
)RETURN ST_POLYGON;
```
功能说明：
获取几何体对象的矩形范围。
参数说明：
geo：几何体对象。
返回值：
几何体对象的矩形范围。
10. ST_X
定义：
```
FUNCTION ST_X (
POINT ST_POINT
)RETURN INT;
```
功能说明：
获取点对象的 X 坐标。
参数说明：
point：点对象。
返回值：
点对象的 X 坐标值。
11. ST_Y
定义：
```
FUNCTION ST_Y (
POINT ST_POINT
)RETURN INT;
```
功能说明：
获取点对象的 Y 坐标。
参数说明：
point：点对象。
返回值：
点对象的 Y 坐标值。
12. ST_StartPoint
定义：
```
FUNCTION ST_STARTPOINT (
CURVE ST_CURVE
)RETURN ST_POINT;
```
功能说明：
获取线对象的起始点。
参数说明：
curve：线对象。
返回值：
线对象的起始点。
13. ST_EndPoint
定义：
```
FUNCTION ST_ENDPOINT (
CURVE ST_CURVE
)RETURN ST_POINT;
```

功能说明：
获取线对象的终点。
参数说明：
curve：线对象。
返回值：
线对象的终点。
14. ST_IsRing
定义：
```
FUNCTION ST_ISRING (
CURVE ST_CURVE
)RETURN INT;
```
功能说明：
获取线对象是否是环。
参数说明：
curve：线对象。
返回值：
0：不是环；1：是环。
15. ST_IsClosed
定义：
```
FUNCTION ST_ISCLOSED (
CURVE ST_CURVE
)RETURN INT;
```
功能说明：
获取线对象是否闭合。
参数说明：
curve：线对象。
返回值：
0：不闭合；1：闭合。
16. ST_Length
定义：
```
FUNCTION ST_LENGTH (
CURVE ST_CURVE
)RETURN INT;
```
功能说明：
获取线对象的长度。
参数说明：
curve：线对象。
返回值：
线对象长度
17. ST_NumPoints
定义：
```
FUNCTION ST_NUMPOINTS (
LINE ST_LINESTRING
)RETURN INT;
```
功能说明：
获取线对象的节点数。
参数说明：
line：线对象。
返回值：
线对象节点数。
18. ST_PointN
定义：
```
FUNCTION ST_POINTN (
LINE ST_LINESTRING,
N INT
```

```
)RETURN ST_POINT;
```

功能说明：

获取线对象的第 n 个节点。

参数说明：

line：线对象。

返回值：

线对象的第 n 个节点。

19. ST_Centroid

定义：

```
FUNCTION ST_CENTROID (
SURFACE ST_SURFACE
)RETURN ST_POINT;
```

功能说明：

获取面对象的中心点。

参数说明：

surface：面对象。

返回值：

面对象的中心点。

20. ST_PointOnSurface

定义：

```
FUNCTION ST_POINTONSURFACE (
SURFACE ST_SURFACE
)RETURN ST_POINT;
```

功能说明：

获取面对象上的一点。

参数说明：

surface：面对象。

返回值：

面对象上的一点。

21. ST_Area

定义：

```
FUNCTION ST_AREA (
SURFACE ST_SURFACE
)RETURN DOUBLE;
```

功能说明：

获取面对象的面积。

参数说明：

surface：面对象。

返回值：

面对象的面积。

22. ST_ExteriorRing

定义：

```
FUNCTION ST_EXTERIORRING (
POLYGON ST_POLYGON
)RETURN ST_LINESTRING;
```

功能说明：

获取面对象的外环。

参数说明：

polygon：面对象。

返回值：

面对象的外环。

23. ST_NumInteriorRing

定义：

```
FUNCTION ST_NUMINTERIORRING (
POLYGON ST_POLYGON
```

```
)RETURN INT;
```

功能说明：

获取面对象的内环数。

参数说明：

polygon：面对象。

返回值：

面对象的内环数。

24. ST_NumInteriorRing

定义：

```
FUNCTION ST_NUMINTERIORRING (
POLYGON ST_POLYGON
)RETURN INT;
```

功能说明：

获取面对象的内环数。

参数说明：

polygon：面对象。

返回值：

面对象的内环数。

25. ST_InteriorRingN

定义：

```
FUNCTION ST_INTERIORRINGN (
POLYGON        ST_POLYGON,
N              INT
)RETURN ST_LINESTRING;
```

功能说明：

获取面对象的第 n 个内环。

参数说明：

polygon：面对象。

n：第几个内环。

返回值：

面对象的第 n 个内环。

26. ST_NumGeometries

定义：

```
FUNCTION ST_NUMGEOMETRIES (
GC   ST_GEOMCOLLECTION
)RETURN INT
```

功能说明：

获取几何对象个数。

参数说明：

gc：几何体集合类。

返回值：

几何体对象个数

27. ST_GeometryN

定义：

```
FUNCTION ST_GEOMETRYN (
GC   ST_GEOMCOLLECTION，
N    INT
)RETURN ST_GEOMETRY
```

功能说明：

获取第 n 个几何对象。

参数说明：

gc：几何体集合类。

n：第几个几何对象。

返回值：

第 n 个几何对象。

28. ST_IsClosed

定义：

```
FUNCTION ST_ISCLOSED (
MCURVE ST_MULTICURVE
)RETURN INT;
```

功能说明：

获取多线象是否闭合。

参数说明：

murve：多线对象。

返回值：

0：不闭合；1：闭合。

29. ST_Length

定义：

```
FUNCTION ST_LENGTH (
MCURVE ST_MULTICURVE
)RETURN DOUBLE;
```

功能说明：

获取多线对象的长度。

参数说明：

murve：多线对象。

返回值：

多线对象长度。

30. ST_Centroid

定义：

```
FUNCTION ST_CENTROID (
MSURFACE ST_MULTISURFACE
)RETURN ST_POINT;
```

功能说明：

获取多面对象的中心点。

参数说明：

msurface：多面对象。

返回值：

多面对象的中心点。

31. ST_PointOnSurface

定义：

```
FUNCTION ST_POINTONSURFACE (
MSURFACE ST_MULTISURFACE
)RETURN ST_POINT;
```

功能说明：

获取多面对象上的一点。

参数说明：

msurface：多面对象。

返回值：

多面对象上的一点。

32. ST_Area

定义：

```
FUNCTION ST_AREA (
MSURFACE ST_MULTISURFACE
)RETURN DOUBLE;
```

功能说明：

获取多面对象的面积。

参数说明：

msurface：多面对象。

返回值：

多面对象的面积。

33. ST_Area

定义：

```
FUNCTION ST_AREA (
MSURFACE ST_MULTISURFACE
)RETURN DOUBLE;
```

功能说明：

获取多面对象的面积。

参数说明：

msurface：多面对象。

返回值：

多面对象的面积。

20.2.2.3 空间关系判断函数

1. ST_Equals

定义：

```
FUNCTION ST_EQUALS (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是否相同。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：两个几何对象不相同；1：两个几何对象相同。

2. ST_Disjoint

定义：

```
FUNCTION ST_DISJOINT (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是否不相交。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：两个几何对象相交；1：两个几何对象不相交。

3. ST_Intersects

定义：

```
FUNCTION ST_INTERSECTS (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是否相交。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：两个几何对象不相交；1：两个几何对象相交。

4. ST_Touches

定义：

```
FUNCTION ST_TOUCHES (
```

```
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是否接触，即存在边界点相同，内节点不相交。

参数说明：

g1：几何对象；

g2：几何对象；

返回值：

0：两个几何对象不接触；1：两个几何对象接触。

5. ST_Crosses

定义：

```
FUNCTION ST_CROSSES (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是否交叉，存在相同的点，但不是完全一致。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：两个几何对象不交叉；1：两个几何对象交叉。

ST_Within

定义：

```
FUNCTION ST_WITHIN (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断对象是否完全包含，对象 g1 是否完全在 g2 的内部

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：g1 不完全在 g2 内部；1：g1 完全在 g2 内部。

6. ST_Contains

定义：

```
FUNCTION ST_CONTAINS (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断对象是否包含，g2 不存在点在 g1 的外边界，且至少存在一个 g2 的内节点在 g1 内部。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：g1 不包含 g2；1：g1 包含 g2。

7. ST_Overlaps

定义：

```
FUNCTION ST_OVERLAPS (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

功能说明：

判断两个几何对象是存在重叠，但并不被对方完全包含。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

0：两个几何对象不存在重叠；1：两个几何对象存在重叠。

8. ST_Relate

定义：

```
FUNCTION ST_RELATE (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY,
PATTERN CHAR(9)
) RETURN INT;
```

功能说明：

判断两个几何对象是否满足 DE-9IM 字符串关系。

参数说明：

g1：几何对象；

g2：几何对象；

pattern：DE-9IM 字符串。

返回值：

0：g1,g2 不满足 pattern 指定的关系；1：g1,g2 满足 pattern 指定的关系。

20.2.2.4 几何运算函数

1. ST_Distance

定义：

```
FUNCTION ST_DISTANCE (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN DOUBLE;
```

功能说明：

获取几何对象间的最短距离。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

几何对象间最短距离。

2. ST_Intersection

定义：

```
FUNCTION ST_INTERSECTION (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN ST_GEOMETRY;
```

功能说明：

获取几何对象的交集。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

几何对象的交集；

3. ST_Difference

定义：

```
FUNCTION ST_DIFFERENCE (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
```

```
) RETURN ST_GEOMETRY;
```

功能说明：

获取几何对象的差集。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

g1 与 g2 的差集。

4. ST_Union

定义：

```
FUNCTION ST_UNION (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN ST_GEOMETRY;
```

功能说明：

获取几何对象的并集。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

g1 与 g2 的并集。

5. ST_SymDifference

定义：

```
FUNCTION ST_SYMDIFFERENCE (
G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN ST_GEOMETRY;
```

功能说明：

获取几何对象的差异集。

参数说明：

g1：几何对象；

g2：几何对象。

返回值：

g1 与 g2 的差异集。

6. ST_Buffer

定义：

```
FUNCTION ST_BUFFER (
G1    ST_GEOMETRY,
D     DOUBLE
) RETURN ST_GEOMETRY;
```

功能说明：

获取代替几何对象 g1 的几何对象，其到 g1 的距离小于等于 d。

参数说明：

g1：几何对象；

d：距离。

返回值：

几何对象 g1 以 d 为半径的外包几何对象。

7. ST_ConvexHull

定义：

```
FUNCTION ST_CONVEXHULL (
G1    ST_GEOMETRY,
D     DOUBLE
) RETURN ST_GEOMETRY;
```

功能说明：

获取几何对象凸壳。

参数说明：

g1：几何对象。

返回值：

几何对象 g1 以 d 为半径的外包几何对象。

20.3 DBMS_JOB 包

为了兼容 ORACLE 定时任务的创建，按指定的时间或间隔执行用户定义的作业。达梦提供了 DBMS_JOB 包以及 DBA_JOBS 视图来实现跟 ORACLE 类似的功能。

20.3.1 相关方法

1. BROKEN 过程

定义：

```
PROCEDURE    Broken(
job    in     integer，
 broken      in     boolean,
next_date   in     datetime :=SYSDATE
)
```

语句功能：

过程更新一个已提交的工作的状态，典型地是用来把一个已过期工作标记为未过期工作；或者把一个未过期的工作设置为何时过期。

参数说明：

job：工作号，唯一标识一个特定工作；

broken：指示此工作是否将标记为过期，TRUE 或 FALSE。TRUE 表明过期，而 FALSE 表明未过期；

next_date：指示在什么时候此工作将再次运行。此参数缺省值为当前日期和时间。

2. CHANGE 过程

定义：

```
PROCEDURE    Change (
job    in     integer,
what in     varchar2,
next_date   in     datetime,
interval     in     varchar2
)
```

语句功能：

用来改变指定工作的设置。

参数说明：

job：工作号，唯一标识一个特定工作；

what：是由此工作运行的一块 PL/SQL 代码块；

next_date：指示何时此工作将被执行；

interval：指示一个工作重执行的频度。

注意：由于 CHANGE 是保留字，调用该过程时需要在过程名上加双引号。

3. INTERVAL 过程

定义：

```
PROCEDURE    Interval (
job    in     integer,
interval     in     varchar2
)
```

语句功能：

用来显式地设置重执行一个工作之间的时间间隔数。

参数说明：

job：工作号，唯一标识一个特定工作；

interval：指示一个工作重执行的频度，该频度是一个日期表达式字符串，以当天 0 点 0 分为起点和该日期表达式字符串计算出来的日期时间之差，以分钟为最小间隔单位。例如’trunc(sysdate) + 1/1440’，则表示间隔 1 分钟，’trunc(sysdate) + 1/24’则表示间隔 1 小时，’trunc(sysdate) + 1’则表示间隔 1 天，最多允许间隔 100 天。达梦的 INTERVAL 参数不支持以周，月，季等单位方式指定间隔。

注意：由于 INTERVAL 是保留字，调用该过程时需要在过程名上加双引号。

4. ISUBMIT 过程

定义：

```
PROCEDURE   ISubmit (
job   in    integer,
what in     varchar2,
next_date   in    datetime,
interval    in    varchar2,
no_parse    in    boolean:=FALSE
)
```

语句功能：

用来用特定的工作号提交一个工作。这个过程与 Submit()过程的唯一区别在于此 job 参数作为 IN 型参数传递且包括一个由开发者提供的工作号。如果提供的工作号已被使用，将产生一个错误。

5. NEXT_DATE 过程

定义：

```
PROCEDURE   Next_Date(
job   in    integer,
next_date   in    datetime
)
```

语句功能：

用来显式地设定一个工作的执行时间。

参数说明：

job：标识一个已存在的工作；

next_date：指示了此工作应被执行的日期与时间。

6. REMOVE 过程

定义：

```
PROCEDURE   Remove(
job   in    integer
)
```

语句功能：

用来删除一个已计划运行的工作。

参数说明：

job：唯一地标识一个工作。这个参数的值是由为此工作调用 Submit()过程返回的 job 参数的值。

已正在运行的工作不能由调用过程序删除。

7. RUN 过程

定义：

```
PROCEDURE   Run(
job   in    integer
)
```

语句功能：

用来立即执行一个指定的工作。

参数说明：

job：标识将被立即执行的工作。

8. SUBMIT 过程

定义：

```
PROCEDURE   Submit (
```

```
job out      integer,
what in      varchar2,
next_date    in     date,
interval     in     varchar2,
no_parse     in     boolean:=FALSE
)
```

语句功能：

使用 Submit()过程，工作被正常地计划好。

参数说明：

job：是由 Submit()过程返回的工作编号。这个值用来唯一标识一个工作；

what：是将被执行的 PL/SQL 代码块；

next_date：指识何时将运行这个工作；

interval：何时这个工作将被重执行；

no_parse：指示此工作在提交时或执行时是否应进行语法分析,指示此 PL/SQL 代码在它第一次执行时应进行语法分析，而 FALSE 指示本 PL/SQL 代码应立即进行语法分析。

9. WHAT 过程

定义：

```
PROCEDURE    What (
job    in     integer,
what in out varchar2
)
```

语句功能：

应许在工作执行时重新设置此正在运行的命令。

参数说明：

job：标识一个存在的工作。

what：指示将被执行的新的 PL/SQL 代码，如果传入的 what 参数值为空串，那将返回该任务原来的 PL/SQL 代码。

10. DBA_JOBS 视图

该视图拥有 JOB（工作号）、LOG_USER（执行工作的用户）、WHAT（任务的 PL/SQL 代码）和 INTERVAL（间隔日期时间串）四个列。

20.3.2 举例说明

用户在使用 DMBS_JOB 包之前，需要提前调用系统过程 SP_INIT_JOB_SYS(1)，创建好调试所需要的包，就可以使用 DMBS_JOB 来调用 PL/SQL 中的过程或函数了。

```
SP_INIT_JOB_SYS(1);
```

创建测试表

```
create    table    a(a    date);
```

创建一个自定义过程

```
create    or    replace    procedure    test    as
begin
insert    into    a    values(sysdate);
end;
/
```

创建 JOB

```
begin
dbms_job.isubmit(1,'test;',sysdate,'trunc(sysdate)+1/1440');--每天 1440 分钟，即一分钟运行 test 过程一次
end;
/
```

运行 JOB

```
begin
dbms_job.run(1);
end;
/
```

PL/SQL　过程已成功完成

```
select to_char(a,'yyyy/mm/dd hh24:mi:ss') 时间 from a;
```

结果如下：

```
时间
----------------------------
2001/01/07 23:51:21
2001/01/07 23:52:22
2001/01/07 23:53:24
```

删除 JOB

```
begin
 dbms_job.remove(1);
 end;
 /
```

PL/SQL 过程已成功完成。

20.4 DBMS_ALERT 包

DBMS_ALERT 系统包是为了在 DM 上兼容 oracle 的 DBMS_ALERT 系统包。用于生成并传递数据库预警信息，当发生特定数据库事件时能够将预警信息传递给应用程序。

20.4.1 相关方法

1．DBMS_ALERT.REGISTER

定义：

```
DBMS_ALERT.REGISTER (
NAME IN VARCHAR(30)
)
```

功能说明：

为当前会话注册一个预警事件，本操作会提交当前事务。

参数说明：

NAME：输入参数，预警事件名。

2．DBMS_ALERT.REMOVE

定义：

```
DBMS_ALERT. REMOVE (
NAME IN VARCHAR(30)
)
```

功能说明：

删除当前会话的一个预警事件，如果该预警事件正在 DBMS_ALERT.WAITONE 或者 DBMS_ALERT.WAITANY 等待，则被阻塞，直到 DBMS_ALERT.WAITONE 或者 DBMS_ALERT.WAITANY 结束。本操作会提交当前事务。

参数说明：

NAME：输入参数，预警事件名。

3．DBMS_ALERT.REMOVEALL

定义：

DBMS_ALERT. REMOVEALL ()

功能说明：

删除当前会话下所有的预警事件，如果该预警事件正在 DBMS_ALERT.WAITONE 或者 DBMS_ALERT.WAITANY 等待，则被阻塞，直到 DBMS_ALERT.WAITONE 或者 DBMS_ALERT.WAITANY 结束. 本操作会提交当前事务。

4．DBMS_ALERT. SIGNAL

定义：

```
DBMS_ALERT. SIGNAL (
NAME IN VARCHAR(30),
MESSAGE IN VARCHAR(1800)
```

```
)
```

功能说明：

给所有名为 name 的预警事件发送消息，当前事务提交时生效。

参数说明：

NAME：输入参数，预警事件名；

MESSAGE：输入参数，待发送的消息。

5．DBMS_ALERT. SET_DEFAULTS

定义：

```
DBMS_ALERT. SET_DEFAULTS (
SENSITIVITY    IN     INT
)
```

功能说明：

设置轮询事件，在 DM 中无用，只为兼容 oracle。

参数说明：

SENSITIVITY：输入参数，轮询时间间隔，单位秒。

6．DBMS_ALERT. WAITONE

定义：

```
DBMS_ALERT. WAITONE (
NAME      IN       VARCHAR(30),
MESSAGE         OUT VARCHAR(1800),
STATUS    OUT INT,
TIMEOUT         IN     INT DEFAULT   8640000
)
```

功能说明：

预警事件上等待预警信号，获得 DBMS_ALERT. SIGNAL 发送的消息内容。本操作会提交当前事务。

参数说明：

NAME：输入参数，预警事件名；

MESSAGE：输出参数，获得 DBMS_ALERT. SIGNAL 发送的消息；

STATUS：输出参数，表示 WAITONE 是否成功；

TIMEOUT：输入参数，WAITONE 的等待超时时间，单位为秒，默认值为 8640000 秒。

7．DBMS_ALERT. WAITANY

定义：

```
DBMS_ALERT. WAITANY (
NAME      OUT    VARCHAR(30),
MESSAGE         OUT VARCHAR(1800),
STATUS    OUT INT,
TIMEOUT         IN     INT DEFAULT   8640000
)
```

功能说明：

等待当前会话任意一个预警信号，获得 DBMS_ALERT. SIGNAL 发送的消息内容。本操作会提交当前事务。

20.4.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 为当前会话注册一个名为 A1 的预警事件，给预警事件上等待预警信号，获得 DBMS_ALERT. SIGNAL 发送的消息内容，最后删除当前会话的一个预警事件。

```
CREATE OR REPLACE PROCEDURE WAIT1 IS
msg    VARCHAR(1800);
stat INTEGER;
BEGIN
```

```
dbms_alert.register('A1');
dbms_alert.waitone('A1', msg, stat, 999999);
print 'Msg: ' || msg || ' Stat: ' || stat;
dbms_alert.remove('A1');
END WAIT1;
/
EXEC WAIT1;
```

例 2 给名为 A1 的预警事件发送消息，当前事务提交时生效。

```
CREATE OR REPLACE PROCEDURE SET_ALERT1
is
BEGIN
dbms_alert.signal('A1', 'this is a sig');
COMMIT;
END SET_ALERT1;
/
EXEC SET_ALERT1;
```

20.5 DBMS_OUTPUT 包

DBMS_OUTPUT 系统包是为了在 DM 上兼容 oracle 的 DBMS_OUTPUT 系统包。提供将文本行写入内存、供以后提取和显示的功能。为用户从 oracle 移植应用提供方便，功能上与 oracle 基本一致。

20.5.1 相关方法

1．DBMS_OUTPUT. DISABLE

定义：

```
DBMS_OUTPUT. DISABLE
```

功能说明：

禁用 DBMS_OUTPUT 包。

举例说明：

```
DBMS_OUTPUT. DISABLE(); --禁用 DBMS_OUTPUT 包
```

2．DBMS_OUTPUT. ENABLE

定义：

```
DBMS_OUTPUT. ENABLE (
buffer_size IN INT DEFAULT 20000
)
```

功能说明：

启用 DBMS_OUTPUT 包。

参数说明：

buffer_size：参数被忽略，只为兼容 oracle 语法。

举例说明：

```
DBMS_OUTPUT. ENABLE(); --启用 DBMS_OUTPUT 包
```

3．DBMS_OUTPUT . PUT_LINE

定义：

```
DBMS_OUTPUT . PUT_LINE(
STR IN    VARCHAR
)
```

功能说明：

输出字符串到客户端。

参数说明：

STR：VARCHAR。

20.5.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 输出字符串到客户端

```
DBMS_OUTPUT . PUT_LINE('ABCDEFG'); --输出字符串到客户端
```

20.6 DBMS_LOGMNR 包

使用 DBMS_LOGMNR 包实现的日志分析工具，是通过分析逻辑日志记载的事务变化，最终确定误操作的时间，跟踪用户事务操作，跟踪并还原用户执行的 DML 操作。

20.6.1 相关方法

1. BUILD 过程

定义：

```
PROCEDURE   LOGMNR_BUILD_DICT(
DICTIONARY_FILEPATH   VARCHAR(256)
);
```

功能说明：

LOGMNR_BUILD_DICT 过程导出字典文件，用于日志分析工具使用脱机字典分析逻辑日志文件。

参数说明：

Dictionary_Filepath：指定导出的字典文件路径名。

备注：字典文件如果已存在，会报错，可以删除字典文件后，重新执行该过程。

2. ADD_LOGFILE 过程

定义：

```
PROCEDURE   LOGMNR_ADD_LLOGFILE(
LOGFILENAME       VARCHAR(256)
);
```

功能说明：

LOGMNR_ADD_LLOGFILE 过程添加逻辑日志文件，用于日志分析。

参数说明：

LogFileName：指定添加的逻辑日志文件路径名。

备注：添加的逻辑日志文件必须是以.log 为扩展名的。否则，会报错。

3. CLEAR_LOGFILE 过程

定义：

```
PROCEDURE   LOGMNR_CLEAR_LLOGFILE;
```

功能说明：

LOGMNR_CLEAR_LLOGFILE 过程用来清空已添加的逻辑日志文件。

4. LOGMNR_GEN 过程

定义：

```
PROCEDURE   LOGMNR_GEN(
    RESULTFILENAME          VARCHAR(256)       DEFAULT '',
    DICTFILENAME        VARCHAR(256)       DEFAULT '',
    OPTIONS             CHAR                    DEFAULT 0,
    STARTLCN            BIGINT                  DEFAULT 0,
    ENDLCN              BIGINT                  DEFAULT 0,
    STARTTIME           TIMESTAMP          DEFAULT '1900-01-01 00:00:00',
```

```
    ENDTIME                 TIMESTAMP                   DEFAULT '1900-01-01 00:00:00'
);
```

功能说明：

LOGMNR_GEN 过程用来对添加的逻辑日志文件进行分析，输出分析结果文件。

参数说明：

ResultFileName：指定输出的分析结果文件路径名；

DictFileName：指定脱机分析时使用的字典文件路径名；

Options：指定日志分析使用的方式，1(脱机)/0(联机)；

StartLcn：指定日志分析的起始 LCN，默认值为 0；

EndLcn：指定日志分析的终止 LCN，默认值为 0；

StartTime：指定日志分析的起始时间值，默认值为'1900-01-01 00:00:00'；

EndTime：指定日志分析的终止时间值，默认值为'1900-01-01 00:00:00'。

备注：

当同时指定 LCN 和 TIME 时，分析结果以 LCN 作为筛选条件。结果文件必须存在，否则报错。

脱机分析：

```
CALL DBMS_LOGMNR.LOGMNR_GEN ('E:\RESULT.TXT','E:\DATA.DICT', 0,1,100,'2012-02-20 10:23:23','2012-02-20 16:00:00');
```

联机分析：

```
CALL DBMS_LOGMNR.LOGMNR_GEN('E:\RESULT.TXT','E:\DATA.DICT', 1,1,100,'2012-02-20 10:23:23','2012-02-20 16:00:00' );
```

也可以只指定参数名，其他使用默认值的方式：

```
CALL DBMS_LOGMNR.LOGMNR_GEN ('E:\RESULT.TXT', 0); --脱机
或者
CALL DBMS_LOGMNR.LOGMNR_GEN ('E:\RESULT.TXT', 1); --联机
```

5. LOGMNR_CREATE_EXTERNAL 过程

定义：

```
PROCEDURE LOGMNR_CREATE_EXTERNAL(
CTRLFILENAME    VARCHAR(256),
EXTERNALNAME    VARCHAR(128)
);
```

功能说明：

LOGMNR_CREATE_EXTERNAL 过程用来创建外部表，显示分析结果内容。

参数说明：

CtrlFileName：指定调用的外部表使用的控制文件路径名；

ExternalName：指定外部表的表名。

备注：如果指定的外部表名已存在，则会报错，可删除该表之后，重新执行该过程。

20.6.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 展示日志分析工具使用流程。

第1步 配置 dm.ini 和 dmllog.ini

使系统产生逻辑日志，这是进行日志分析的前置条件。

a)dm.ini

```
LLOG_INI                    =   2               #LLOG.INI
```

备注：对所有表生成逻辑日志。

b)dmllog.ini

```
[LLOG_INFO]
```

```
#ID LLOG_PATH LOCAL_ARCH_SPACE_LIMIT(G) REMOTE_ARCH_FLAG
REMOTE_ARCH_INSTNAME REMOTE_ARCH_TYPE REMOTE_ARCH_TIMER
-1 D:\LLOG 0

[LLOG_TAB_MAP]
#LLOG_ID SCH_ID TAB_ID
-1 -1 -1
```

备注：-1，区别于复制 ID；D:\LLOG 表示逻辑日志的生成目录；

逻辑日志文件最大空间限制为 32M，超过 32M，自动生成新的日志文件。

第2步 导出离线数据字典

这一步骤不是必需的，只有当用户利用离线数据字典分析逻辑日志时，才需要执行这一步。利用 dbms_logmnr 包中的 build 存储过程实现。

比如：导出离线字典到 e:\data.dict

```
CALL DBMS_LOGMNR.LOGMNR_BUILD_DICT('E:\DATA.DICT');
```

指定导出的字典文件名后缀不加限制。

第3步 添加/删除日志文件列表

利用 dbms_logmnr 包中的 logmnr_add_llogfile/logmnr_clear_llogfile 添加或删除日志文件。

比如：添加日志文件。

```
CALL DBMS_LOGMNR.LOGMNR_ADD_LLOGFILE
('D:\LLOG\LLOG_ARCH_FILE_20120220144758.LOG');
```

上述操作，添加了日志文件 llog_arch_file_20120220144758.log。

删除日志文件，会删除所有日志文件：

```
CALL DBMS_LOGMNR.LOGMNR_CLEAR_LLOGFILE ( );
```

第4步 分析日志文件

如果没有执行步骤 3)，执行步骤 4)会报缺少逻辑日志错误。

利用 dbms_logmnr 包中的 logmnr_gen 执行分析日志。结果分析文件必须存在，工具会先清空该文件，然后重新写入分析结果。分析结果文件 d:\kkk.result 需提前自行建好。

具体调用如下：

```
CALL DBMS_LOGMNR. LOGMNR_GEN('D:\KKK.RESULT','D:\ABC.DICT', 1,1,100, '2012-02-20
10:23:23', '2012-02-20 16:00:00'); --脱机
或者
CALL DBMS_LOGMNR. LOGMNR_GEN('D:\KKK.RESULT','D:\ABC.DICT', 0,1,100, '2012-02-20
10:23:23', '2012-02-20 16:00:00'); --联机
```

也可以通过只指定分析结果文件名，其他参数使用默认值的方式来分析日志文件：

```
CALL DBMS_LOGMNR. LOGMNR_GEN('D:\KKK.RESULT','D:\ABC.DICT',1); --脱机
或者
CALL DBMS_LOGMNR. LOGMNR_GEN('D:\KKK.RESULT','D:\ABC.DICT',0); --联机
```

第5步 将分析结果导入外部表

用户根据外部表导入流程，构造外部表的控制文件，其中控制文件的路径填写分析结果文件路径。调用 dbms_logmnr 包中的 logmnr_create_external 创建外部表。

比如：将分析结果文件 d:\kkk.result 结果导入外部表 LOG_RESULT 中。

首先执行的控制文件（E:\CTRL.TXT）如下：

```
LOAD DATA
INFILE 'D:\ KKK.RESULT'
INTO TABLE EXT
FIELDS '|'
```

其次，调用 logmnr_create_external 创建外部表。

```
CALL DBMS_LOGMNR.LOGMNR_CREATE_EXTERNAL('E:\CTRL.TXT', 'LOG_RESULT');
```

如果创建的外部表已存在，则创建失败，删除该表后，重新执行即可。

第6步 查询分析结果

通过查询外部表，获取分析结果。如下为分析结果对应外部表字段说明：

表 外部表字段说明

名称	说明	字段类型
LCN	LCN 号	BIGINT
TIMESTAMP	时间	TIMESTAMP
OPERATOR	操作用户	VARCHAR(128)
TABLE_NAME	操作表名	VARCHAR(128)
OPERATION	操作类型	VARCHAR(100)
SQL	执行语句信息	VARCHAR(4000)

如下为查询分析结果的 sql 语句：

```
CREATE TABLE TEST(C1 INT, C2 CHAR);
INSERT INTO TEST VALUES(1, 'A');
COMMIT;
UPDATE TEST SET C2 = 'B' WHERE C1 = 1;
DELETE FROM TEST WHERE C1 = 1;
COMMIT;
CALL
DBMS_LOGMNR.LOGMNR_ADD_LLOGFILE('D:\LLOG\LLOG_ARCH_FILE_20120220144758.LOG');
CALL DBMS_LOGMNR.LOGMNR_GEN('D:\KKK.RESULT','D:\ABC.DICT', 1,1, 100, '2012-02-20
10:23:23', '2012-02-20 16:00:00');
CALL DBMS_LOGMNR.LOGMNR_CREATE_EXTERNAL('E:\CTRL.TXT', 'EXTERNAL_TABLE');
SELECT * FROM EXTERNAL_TABLE WHERE LCN >= 50360;
```

如下为上述操作结果的日志分析结果：

```
LCN TIMESTAMP OPERATOR TABLE_NAME OPERATION SQL
1 50360 2011-11-29 12:30:27.000000 SYSDBA SYSOBJECTS INSERT INSERT INTO
"SYS"."SYSOBJECTS"("NAME", "ID", "SCHID", "TYPE$", "SUBTYPE$", "PID", "VERSION", "CRTDATE",
"INFO1", "INFO2", "INFO3", "INFO4", "INFO5", "INFO6", "INFO7", "INFO8",
"VALID")VALUES('TEST',7926,150994945,'SCHOBJ','UTAB',0,0,'2011-11-29
12:30:27',2097152,0,65536,0,NULL,NULL,NULL,NULL,'Y');
2 50361 2011-11-29 12:30:27.000000 SYSDBA SYSCOLUMNS INSERT INSERT INTO
"SYS"."SYSCOLUMNS"("NAME", "ID", "COLID", "TYPE$", "LENGTH$", "SCALE", "NULLABLE$",
"DEFVAL", "INFO1", "INFO2")VALUES('C1',7926,0,'INT',4,0,'Y',NULL,0,0);
3 50362 2011-11-29 12:30:27.000000 SYSDBA SYSCOLUMNS INSERT INSERT INTO
"SYS"."SYSCOLUMNS"("NAME", "ID", "COLID", "TYPE$", "LENGTH$", "SCALE", "NULLABLE$",
"DEFVAL", "INFO1", "INFO2")VALUES('C2',7926,1,'CHAR',1,0,'Y',NULL,0,0);
4 50363 2011-11-29 12:30:27.000000 SYSDBA SYSOBJECTS INSERT INSERT INTO
"SYS"."SYSOBJECTS"("NAME", "ID", "SCHID", "TYPE$", "SUBTYPE$", "PID", "VERSION", "CRTDATE",
"INFO1", "INFO2", "INFO3", "INFO4", "INFO5", "INFO6", "INFO7", "INFO8",
"VALID")VALUES('INDEX33562265',33562265,150994945,'TABOBJ','INDEX',7926,0,'2011-11-29
12:30:27',NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,'Y');
5 50364 2011-11-29 12:30:27.000000 SYSDBA SYSINDEXES INSERT INSERT INTO
"SYS"."SYSINDEXES"("ID", "ISUNIQUE", "GROUPID", "ROOTFILE", "ROOTPAGE", "TYPE$", "XTYPE",
"FLAG", "KEYNUM", "KEYINFO", "INIT_EXTENTS", "BATCH_ALLOC",
"MIN_EXTENTS")VALUES(33562265,'N',4,0,219952,'BT',0,1,0,0X,1,1,1);
6 50365 2011-11-29 12:30:27.000000 SYSDBA NULL COMMIT COMMIT;
7 50367 2011-11-29 12:31:34.000000 SYSDBA TEST INSERT INSERT INTO "SYSDBA"."TEST"("C1",
"C2")VALUES(1,'A');
8 50368 2011-11-29 12:31:36.000000 SYSDBA NULL COMMIT COMMIT;
9 50370 2011-11-29 12:31:46.000000 SYSDBA TEST UPDATE UPDATE "SYSDBA"."TEST" SET "C2" =
'B' WHERE ROWID = 1;
10 50371 2011-11-29 12:31:54.000000 SYSDBA TEST DELETE DELETE FROM "SYSDBA"."TEST"
WHERE ROWID = 1;
11 50372 2011-11-29 12:32:16.000000 SYSDBA NULL COMMIT COMMIT;
```

20.7 DBMS_ADVANCED_REWRITE 包

在不改变应用程序的前提下，在服务器端将查询语句替换成其他的查询语句执行，此时就需要查询重写（QUERY REWRITE）。DM7 使用 DBMS_ADVANCED_REWRITE 包实现该功能，不支持安全策略。DM7 支持对原始语句中的某些特定词的替换，以及整个语句的替换，不支持递归和变换替换。

20.7.1 相关方法

1. DECLARE_REWRITE_EQUIVALENCE

定义：

```
PROCEDURE DECLARE_REWRITE_EQUIVALENCE (
    NAME                VARCHAR(128),
    SOURCE_STMT         VARCHAR(8188),
    DESTINATION_STMT    VARCHAR(8188),
    VALIDATE            BOOLEAN,
    REWRITE_MODE        VARCHAR(16)
);
```

功能说明：

声明一个等价重写规则。

参数说明：

NAME：重写规则的唯一标识，在会话级唯一；

SOURCE_STMT：原始查询语句；

DESTINATION_STMT：目标语句；

VALIDATE：是否校验原始语句和目标语句等价；

REWRITE_MODE：包含 4 种：DISABLED，不允许重写；TEXT_MATCH，文本匹配重写；GENERAL，变换重写；RECURSIVE，递归重写。后面两种暂不支持，可以声明成功，但是不起作用。

使用说明：

(1) 如果一个 SOURCE_STMT 对应多个 DESTINATION_STMT，即用户创建很多相同的重写规则，只是名字不同，在重写时只有第一个才有效。也就是说只执行第一个重写规则，后面的所有重写规则无效。

(2) VALIDATE 置为“TRUE”时，要求 SOURCE_STMT 和 DESTINATION_STMT 等价，例如：“SELECT C1 FROM X1 WHERE C1 != 1;”与“SELECT C1 FROM X1 WHERE C1 > 1 OR C1 < 1;”,否则报错。暂不支持为“TRUE”的情况。

(3) 不允许 SOURCE_STMT 和 DESTINATION_STMT 完全相同，否则报错：“源语句与目标语句相同”。

(4) SOURCE_STMT 和 DESTINATION_STMT 语句必须为查询语句，否则报错：“源语句与目标语句不兼容”。

(5) SOURCE_STMT 和 DESTINATION_STMT 语句必须经过语法和语义正确解析，否则报错。如果两个语句中都出错，则报 SOURCE_STMT 的错，即首先解析 SOURCE_STMT，如果出错直接报错，不再解析后面的 DESTINATION_STMT。

(6) REWRITE_MODE 为“TEXT_MATCH”时，只允许替换 SOURCE_STMT 中完全相同的语句，不区分大小写。

(7) SOURCE_STMT 和 DESTINATION_STMT 语句不允许包含“{}”字符。

(8) 扩展功能说明：如果 SOURCE_STMT 为空，DESTINATION_STMT 格式为“TABLE/[TABLE]”，则可以将当前会话语句中的“TABLE”换成“[TABLE]”，“TABLE”为正则表达式。DM7 支持符合 POSIX 标准的正则表达式。如下表所示：

表 正则表达式语法

语法	说明	示例
.	匹配任何除换行符之外的单个字符	DA. 匹配“DAMENG”
*	匹配前面的字符 0 次或多次	A*B 匹配“BAT”中的“B”和“ABOUT”中的“AB”
+	匹配前面的字符一次或多次	AC+ 匹配包含字母“A”和至少一个字母“C”的单词，如“RACE”和“ACE”
^	匹配行首	^CAR 仅当单词“CAR”显示为行中的第一组字符时匹配该单词
$	匹配行尾	END$ 仅当单词“END”显示为可能位于行尾的最后一组字符时匹配该单词
[]	字符集，匹配任何括号间的字符	BE[N-T] 匹配“BETWEEN”中的“BET”、

		“BENEATH” 中 的 “BEN” 和 “BESIDE” 中 的 “BES”，但不匹配“BELOW”中的“BEL”
[^]	排除字符集。匹配任何不在括号间的字符	BE[^N-T] 匹 配 “BEFORE” 中 的 “BEF” 、 “BEHIND” 中 的 “BEH” 和 “BELOW” 中 的 “BEL”，但是不匹配“BENEATH”中的“BEN”
(表达式)	在表达式加上括号或标签在替换命令中使用	(ABC)+匹配“ABCABCABC”
|	匹配 OR 符号 (|) 之前或之后的表达式。最常用在分组中	(SPONGE|MUD) BATH 匹 配 “SPONGE BATH”和“MUD BATH”
\	按原义匹配反斜杠 (\) 之后的字符。这使您可以查找正则表达式表示法中使用的字符，如 { 和 ^	\^ 搜索 ^ 字符
{}	匹配以带括号的表达式标记的文本	ZO{1} 匹 配 “ALONZO1” 和 “GONZO1” 中 的 “ZO1”，但不匹配“ZONE”中的“ZO”
[[:alpha:]]	表示任意字母\w ([a-z]+) | ([A-Z]+)	
[[:digit:]]	表示任意数字\d ([0-9]+)	
[[:lower:]]	表示任意小写字母 ([a-z]+)	
[[:alnum:]]	表示任意字母和数字([a-z0-9]+)	
[[:space:]]	表示任意空格	
[[:upper:]]	表示任意大写字母([A-Z]+)	
[[:punct:]]	表示任意标点符号	
[[:xdigit:]]	表示任意 16 进制数([0-9a-fA-F]+)	

2. ALTER_REWRITE_EQUIVALENCE

定义：

```
PROCEDURE ALTER_REWRITE_EQUIVALENCE (
    NAME                CHAR(128),
    REWRITE_MODE        VARCHAR(16)
);
```

功能说明：

修改重写模式。

参数说明：

NAME: 重写规则的唯一标识；

REWRITE_MODE: 包含 4 种，同上。

3. VALIDATE_REWRITE_EQUIVALENCE

定义：

```
VALIDATE_REWRITE_EQUIVALENCE (NAME    CHAR(128));
```

功能说明：

校验重写规则是否等效，暂不支持。提供该函数，但是不进行校验。

参数说明：

NAME: 重写规则的唯一标识；

4. DROP_REWRITE_EQUIVALENCE。

定义：

```
PROCEDURE DROP_REWRITE_EQUIVALENCE (NAME       CHAR(128));
```

功能说明：

删除重写规则。

参数说明：

NAME: 重写规则的唯一标识。

20.7.2 字典信息

所有的重写规则信息都保存到系统表 SYS_REWRITE_EQUIVALENCES 中，该系统表的结构如下表所示。所有的用户创建的语句重写信息都保存到该表中。

表 系统表 SYS_REWRITE_EQUIVALENCES 结构

序号	列名	类型	是否允许为空	说明

1	OWNER	VARCHAR(128)	N	重写规则的拥有者
2	NAME	VARCHAR(128)	N	重写规则名
3	SOURCE_STMT	VARCHAR(3200)	Y	原始语句
4	DESTINATION_STMT	VARCHAR(3200)	Y	目标语句
5	REWRITE_MODE	VARCHAR(16)	Y	重写模式
6	RESV1	INTEGER	Y	保留字段
7	RESV2	VARCHAR(128)	Y	保留字段

主键（OWNER，NAME）。

通过查询语句可以获得重写规则的信息：

```
SELECT * FROM SYS_REWRITE_EQUIVALENCES;
```

20.7.3 使用说明

1. 授权

用户必须拥有 EXECUTE ON DBMS_ADVANCED_REWRITE 权限才可以执行语句重写。

语法格式：

```
GRANT EXECUTE ON DBMS_ADVANCED_REWRITE TO USER;
```

2. 设置会话标记

每个会话对应一个 QUERY_REWRITE_INTEGRITY 标记，该标记只在会话期间起作用，不保存到字典信息中，如果会话结束，则该标记也无效。会话重建后，该标记也需要重新赋值才有效。

QUERY_REWRITE_INTEGRITY 标记用于表示该会话是否可以执行查询语句重写，默认值为“ENFORCED”，另外两个值为：“TRUSTED”和“STALE_TOLERATED”。默认不允许语句重写；DM7 未对后两个参数进行严格区分，一般设置后两个参数都允许查询重写。

语法格式：

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
```

参数说明：

TRUSTED：表示在保证物化视图数据是正确时才使用该视图替换；

STALE_TOLERATED：表示能容忍错误的数据，即可以使用不正确的物化视图执行重写。

3. 重写规则的作用域

重写规则只允许在当前会话中执行，禁止跨模式使用其他用户指定的重写规则。SYSDBA 也不可以使用其他用户的重写规则替换。

4. 回滚

执行方法 DECLARE_REWRITE_EQUIVALENCE 后，将自动提交，不能执行回滚。在该方法之前的所有操作也自动提交。

5. 自动重写

自动重写，指的是用户在执行任何查询语句时，可以自动调用重写规则执行重写，而不需要设置会话标记来控制是否执行查询重写。SYSDBA 可以设置某些用户自动重写标记，设置函数为：SP_USER_SET_AUTO_REWRITE_FLAG。该标记和会话标记中有一个有效，则允许查询重写。

语法格式：

```
SP_USER_SET_AUTO_REWRITE_FLAG ('USER_NAME', VALUE) ;
```

参数说明：

USER_NAME：用户名；

VALUE：0 表示不允许自动重写，1 表示允许自动重写。

6. 查询重写时机

查询重写的时机主要发生在如下几个地方：

(1) 一般的用户查询，例如 SELECT COUNT(*) FROM T1;

(2) 非动态的查询执行时，例如存储过程中重写：SELECT COUNT(*) FROM T1 WHERE

C1 > 1;但是以下语句不可以重写：I INT := 1; SELECT COUNT(*) FROM T1 WHERE C1 > I;

(3) 视图中的查询重写，不支持递归替换。例如：CREATE VIEW V1 AS SELECT COUNT(*) FROM T1;如果“SELECT COUNT(*) FROM T1;”存在重写规则，也可以重写。

7. 可以关闭重用执行计划功能

使用时，可以在 DM.INI 中将重用执行计划标记（USE_PLN_POOL）置为 0。否则，先执行过查询语句后，再设置查询规则，然后对比两次的查询结果会相同，即未被重写。如果不关闭，只能等到设置过查询规则之后，才能执行查询。

20.7.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 REWRITE 包的应用。

```
CREATE USER USER1 IDENTIFIED BY USER123;                              --GRANT DBA TO USER1;
GRANT CREATE TABLE TO USER1;
GRANT EXECUTE ON SYS.DBMS_ADVANCED_REWRITE TO USER1;
DROP TABLE X1;
CREATE TABLE X1(C1 INT, C2 CHAR(20));
INSERT INTO X1 VALUES(12, 'TEST12');
INSERT INTO X1 VALUES(13, 'TEST13');
INSERT INTO X1 VALUES(14, 'TEST14');
INSERT INTO X1 VALUES(15, 'TEST15');
CREATE TABLE X2(D1 INT, D2 CHAR(30));
COMMIT;
BEGIN
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE(
'TEST_REWRITE',
'SELECT COUNT(*) FROM X1',                                    --原始查询语句
'SELECT COUNT(*) FROM X2',                                    --目标语句
FALSE,
'TEXT_MATCH'
);
END;
/
SELECT COUNT(*) FROM X1;                                       --执行结果 4

ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;     --默认为'ENFORCED'
SELECT COUNT(*) FROM X1;                                       --执行结果 0

DBMS_ADVANCED_REWRITE.DROP_REWRITE_EQUIVALENCE ('TEST_REWRITE');
--DROP REWRITE

BEGIN
    DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE(
     'TEST_REWRITE',
     NULL,                                                     --为空
     'FRO./FROM',                                              --使用正则表达式
     FALSE,
     'TEXT_MATCH'
     );
     END;
SELECT COUNT(*) FROK X1;                                       --执行结果 4
```

20.8 DBMS_BINARY 包

DBMS_BINARY 系统包用于读写二进制流，实现从一个二进制流指定位置开始对基本数据类型的读写，包括 CHAR、VARCHAR、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE 数据类型。

20.8.1 相关方法

DBMS_BINARY 包所支持的 8 个过程和 8 个函数，分别用来在二进制流中存取数据。这些过程和函数通过调用相应系统内部函数来实现存取数据的过程。如下详细介绍各过程和函数：

1. BINARY_GET_CHAR

定义：

```
FUNCTION BINARY_GET_CHAR(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 CHAR 类型数据。

返回值:

CHAR 类型数据。

2. BINARY_GET_VARCHAR

定义：

```
FUNCTION BINARY_GET_VARCHAR(VB VARBINARY, OFFSET INT, LENGTH INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个长度为 LENGTH 的 VARCHAR 类型数据。

返回值:

VARCHAR 类型数据

3. BINARY_GET_TINYINT

定义：

```
FUNCTION BINARY_GET_TINYINT(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 TINYINT 型数据。

返回值:

TINYINT 类型数据。

4. BINARY_GET_SMALLINT

定义：

```
FUNCTION BINARY_GET_SMALLINT(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 SMALLINT 型数据。

返回值:

SMALLINT 类型数据。

5. BINARY_GET_INT

定义：

```
FUNCTION BINARY_GET_INT(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 INT 型数据。

返回值:

INT 类型数据。

6. BINARY_GET_BIGINT

定义：

```
FUNCTION BINARY_GET_BIGINT(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 BIGINT 型数据。

返回值:

BIGINT 类型数据。

7. BINARY_GET_FLOAT

定义：

```
FUNCTION BINARY_GET_FLOAT(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 FLOAT 型数据。

返回值:

FLOAT 类型数据。

8. BINARY_GET_DOUBLE

定义：

```
FUNCTION BINARY_GET_DOUBLE(VB VARBINARY, OFFSET INT)
```

功能说明：

返回从二进制流 VB 中偏移 OFFSET 开始的一个 DOUBLE 型数据。

返回值:

DOUBLE 类型数据。

9. BINARY_SET_CHAR

定义：

```
PROCEDURE BINARY_SET_CHAR(VB IN OUT VARBINARY, OFFSET INT, VALUE CHAR)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 CHAR 型数据。

10. BINARY_SET_VARCHAR

定义：

```
PROCEDURE BINARY_SET_VARCHAR(VB IN OUT VARBINARY, OFFSET INT, VALUE VARCHAR)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入 VARCHAR 型数据。

11. BINARY_SET_TINYINT

定义：

```
PROCEDURE BINARY_SET_TINYINT(VB IN OUT VARBINARY, OFFSET INT, VALUE TINYINT)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 TINYINT 型数据。

12. BINARY_SET_SMALLINT

定义：

```
PROCEDURE BINARY_SET_SMALLINT(VB IN OUT VARBINARY, OFFSET INT, VALUE SMALLINT)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 SMALLINT 型数据。

13. BINARY_SET_INT

定义：

```
PROCEDURE BINARY_SET_INT(VB IN OUT VARBINARY, OFFSET INT, VALUE INT)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 INT 型数据。

14. BINARY_SET_BIGINT

定义：

```
PROCEDURE BINARY_SET_BIGINT(VB IN OUT VARBINARY, OFFSET INT, VALUE BIGINT)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 BIGINT 型数据。

15. BINARY_SET_FLOAT

定义：

```
PROCEDURE BINARY_SET_FLOAT(VB IN OUT VARBINARY, OFFSET INT, VALUE FLOAT)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 FLOAT 型数据。

16. BINARY_SET_DOUBLE

定义：

```
PROCEDURE BINARY_SET_DOUBLE(VB IN OUT VARBINARY, OFFSET INT, VALUE DOUBLE)
```

功能说明：

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 DOUBLE 型数据。

参数说明：

因为 DBMS_BINARY 包所支持的 8 个过程和 8 个函数参数意义相同，如下统一说明：

VB：用户输入的二进制流；

OFFSET：向二进制流中写入数据或者从二进制流取出数据时的偏移量；

LENGTH：表示从二进制流指定偏移位置开始取多少个字符；

VALUE：待写入的数据。

20.8.2 错误处理

1. **空值处理**

如果输入参数中存在空值，则结果返回空值。

例如：

```
BINARY_GET_CHAR (VB VARBINARY, OFFSET INT)
SELECT DBMS_BINARY.BINARY_GET_CHAR (‘DFEGRRWEGHE’, NULL);
```

结果：NULL

说明：偏移 OFFSET 输入空值，在函数内部处理的时候返回 NULL。

2. **非法参数**

如果偏移量为负数或者偏移量加上所取数据类型的长度大于二进制流的长度，报非法参数错误 EC_RN_INVALID_ARG_DATA，错误码：-6803。

```
BINARY_GET_CHAR (VB VARBINARY, OFFSET INT)
SELECT DBMS_BINARY.BINARY_GET_CHAR (‘DFEGRRWEGHE’, 11);
```

结果：输入参数非法。

说明：输入的二进制流的长度为 11，从 0 开始取值，偏移量为 11 的时候已经越界，提示非法参数错误。

20.8.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 使用 BINARY_SET_TINYINT 过程写入一个 TINYINT 类型数据，并使用 BINARY_GET_TINYINT 函数输出该 TINYINT 类型数据。

```
DECLARE
BIN VARBINARY(50);
VAL TINYINT;
BEGIN
BIN = 'ABCDEF87';
VAL = 127;
DBMS_BINARY.BINARY_SET_TINYINT(BIN,0,VAL);
VAL = DBMS_BINARY.BINARY_GET_TINYINT(BIN,0);
PRINT VAL;
END;
```

结果：

```
127
```

例 2 使用 BINARY_SET_VARCHAR 过程写入一个 VARCHAR 类型数据，并使用 BINARY_GET_VARCHAR 函数输出该 VARCHAR 类型数据。

```
DECLARE
```

```
BIN VARBINARY(50);
VAL VARCHAR;
BEGIN
BIN = 'ABCDEF87';
DBMS_BINARY.BINARY_SET_VARCHAR(BIN,0,'AAAA');
VAL = DBMS_BINARY.BINARY_GET_VARCHAR(BIN,0,4);
PRINT VAL;
END;
```

结果：

```
AAAA
```

20.9 DBMS_PAGE 包

20.9.1 索引页

供获取索引页信息的包 DBMS_SPACE 包的索引页部分。

20.9.1.1 数据类型

DBMS_PAGE 包的索引页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
DPH_T 类型	用于承载页类型及数据页头的所有信息包括：记录数、用户记录数、堆栈底部偏移、堆栈顶部偏移、分支号、碎片空间起始偏移、最小记录偏移、最大记录偏移、外部记录信息、已用空间
DPIH_T 类型	用于承载索引数据头的所有信息，包括：在 B 树中的层次、索引号、B 树中叶节点所在的段的组号、B 树中叶节点所在的段文件号、B 树中叶节点所在的段的页号、B 树中叶节点所在的段的偏移、B 树中内节点所在段的组号、B 树中内节点所在段的文件号、B 树中内节点所在段的页号、B 树中内节点所在段的偏移
DPICH_T 类型	用于承载索引控制头的所有信息，包括：下一个 ROWID、索引的已用空间、B 树中所有记录数、目前 IDENTITY 值
DPS_T 类型	用于承载页空间使用状况的信息，包括：堆底偏移、堆顶偏移、已用字节数、空闲字节数、使用率、堆顶剩余空间、碎片空间大小、碎片空间占总剩余比率、最大可插入记录长度
REC_T 类型	用于承载记录相关信息，包括：记录序号、记录偏移、记录长度、是否被删除、ROWID 或聚簇索引 ID、回滚地址-文件号、回滚地址-页号、回滚地址-偏移
FREE_LIST_T 类型	用于承载单个碎片空间的相关信息，包括：碎片偏移、碎片长度
CTL_BIT_ARR_T 类型	用于承载控制位的 TINYINT 类型的数组
DPH_ARR_T 类型	DPH_T 类型的数组
DPIH_ARR_T 类型	DPIH_T 类型的数组
DPICH_ARR_T 类型	DPICH_T 类型的数组
DPS_ARR_T 类型	DPS_T 类型的数组
REC_ARR_T 类型	REC_T 类型的数组
FREE_LIST_ARR_T 类型	FREE_LIST_T 类型的数组

20.9.1.2 相关方法

DBMS_PAGE 包中的索引页部分所包含的过程和函数详细介绍如下：

1. DATA_PAGE_HEAD_LOAD/ DATA_PAGE_HEAD_GET

定义：

```
PROCEDURE DATA_PAGE_HEAD_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
INFO OUT DPH_T
);
```

定义：

```
FUNCTION DATA_PAGE_HEAD_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN DPH_T;
```

功能说明：

获取数据页头信息。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

2. DATA_PAGE_IND_HEAD_LOAD/ DATA_PAGE_IND_HEAD_GET

定义：

```
PROCEDURE DATA_PAGE_IND_HEAD_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
INFO OUT DPIH_T
);
```

定义：

```
FUNCTION DATA_PAGE_IND_HEAD_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN DPIH_T;
```

功能说明：

获取索引数据页头信息。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

3. DATA_PAGE_IND_CTL_HEAD_LOAD/ DATA_PAGE_IND_CTL_HEAD_GET

定义：

```
PROCEDURE DATA_PAGE_IND_CTL_HEAD_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
INFO OUT DPICH_T
);
```

定义：

```
FUNCTION DATA_PAGE_IND_CTL_HEAD_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN DPICH_T;
```

功能说明：

获取索引控制页头信息。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

4. DATA_PAGE_SPACE_LOAD/ DATA_PAGE_SPACE_GET

定义：

```
PROCEDURE DATA_PAGE_SPACE_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
INFO OUT DPS_T
);
```

定义：

```
FUNCTION DATA_PAGE_SPACE_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN DPS_T;
```

功能说明：

获取页空间使用状况信息。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

5. DATA_PAGE_LOAD_REC_BY_SLOT_NO/ DATA_PAGE_GET_REC_BY_SLOT_NO

定义：

```
PROCEDURE DATA_PAGE_LOAD_REC_BY_SLOT_NO(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
SLOT_NO IN SMALLINT,
REC OUT REC_T
);
```

定义：

```
FUNCTION DATA_PAGE_GET_REC_BY_SLOT_NO(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
SLOT_NO IN SMALLINT)
RETURN REC_T;
```

功能说明：

获取指定序号的记录。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号；

SLOT_NO：记录序号。

6. DATA_PAGE_CTL_BIT_BY_SLOT_NO_LOAD/
   DATA_PAGE_CTL_BIT_BY_SLOT_NO_GET

定义：

```
PROCEDURE DATA_PAGE_CTL_BIT_BY_SLOT_NO_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
```

```
SLOT_NO IN SMALLINT,
COL_NUM IN SMALLINT,
CTL_BIT OUT CTL_BIT_ARR_T
);
```

定义：

```
FUNCTION DATA_PAGE_CTL_BIT_BY_SLOT_NO_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
SLOT_NO IN SMALLINT,
COL_NUM IN SMALLINT)
RETURN CTL_BIT_ARR_T;
```

功能说明：

获取指定序号的记录的控制位。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号；

SLOT_NO：记录序号；

COL_NUM：要取得控制位的个数。

7. DATA_PAGE_LOAD_FREE_LIST/ DATA_PAGE_GET_FREE_LIST

定义：

```
PROCEDURE DATA_PAGE_LOAD_FREE_LIST(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
NO IN SMALLINT,
FREE_LIST OUT FREE_LIST_ARR_T
);
```

定义：

```
FUNCTION DATA_PAGE_GET_FREE_LIST(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
NO IN SMALLINT)
RETURN FREE_LIST_ARR_T;
```

功能说明：

获取所有碎片空间大小及偏移。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号；

NO：要取到第几个空闲页。

8. DATA_PAGE_TID_LOAD/ DATA_PAGE_TID_GET

定义：

```
PROCEDURE DATA_PAGE_TID_LOAD(
TS_ID IN SMALLINT，
FILE_ID IN SMALLINT，
PAGE_NO IN INT，
TID OUT BIGINT);
```

定义：

```
FUNCTION DATA_PAGE_TID_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN INT;
```

功能说明：

输入表空间号、文件号及页号，获得表 ID。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

9. DATA_PAGE_TNAME_LOAD/ DATA_PAGE_TNAME_GET

定义：

```
PROCEDURE DATA_PAGE_TNAME_LOAD(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT,
TNAME OUT VARCHAR(35)
);
```

定义：

```
FUNCTION DATA_PAGE_TNAME_GET(
TS_ID IN SMALLINT,
FILE_ID IN SMALLINT,
PAGE_NO IN INT)
RETURN VARCHAR;
```

功能说明：

输入表空间号、文件号及页号，输出表名。过程和函数功能相同。

参数说明：

TS_ID：表空间 ID；

FILE_ID：文件号；

PAGE_NO：页号。

20.9.1.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取数据页头信息的，利用包内类型 DPH_ARR_T 以及过程 DATA_PAGE_HEAD_LOAD，这里假设 0 号表空间的 0 号文件 32 号页是一个数据页：

```
DECLARE
INFO DBMS_PAGE.DPH_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.DPH_T[1];
DBMS_PAGE.DATA_PAGE_HEAD_LOAD(0,0,32,INFO[1]);
SELECT * FROM ARRAY INFO;
END;
```

输出结果(具体数值取决于实际状况)：

```
N_SLOT   N_REC     HEAP_LOW HEAP_HIGH BRANCH_NO FREE_LIST_OFF  REC_MIN_OFF
REC_MAX_OFF    EXTERNAL   USED_BYTE
4     2      155  0      0      98     82     90     <NULL>   152
```

例 2 获取页内所有用户记录的信息,利用包内类型 DPH_T、REC_T、REC_ARR_T 以及过程 DATA_PAGE_HEAD_LOAD、DATA_PAGE_REC_BY_SLOT_NO_LOAD,这里假设 4 号表空间的 0 号文件 32 号页是一个数据页，并存储了两行记录：

```
DECLARE
HEAD DBMS_PAGE.DPH_T;
INFO DBMS_PAGE.REC_ARR_T;
BEGIN
DBMS_PAGE.DATA_PAGE_HEAD_LOAD(4,0,32,HEAD);
INFO = NEW DBMS_PAGE.REC_T[HEAD.N_REC];
FOR I IN 1..HEAD.N_REC LOOP
DBMS_PAGE.DATA_PAGE_REC_BY_SLOT_NO_LOAD(4,0,32,I,INFO[I]);
END LOOP;
SELECT * FROM ARRAY INFO;
```

```
END;
```

输出结果(具体数值取决于实际状况)：

```
SLOT_NO OFFSET   LEN IS_DEL TRX_ID CLU_ROWID ROLL_ADDR_FILE ROLL_ADDR_PAGE
ROLL_ADDR_OFF
1       117      19  0      2705   33559296   0             2              0
2       136      19  0      2706    50336512  0             3              0
```

20.9.2INODE 页

INODE 页用于存储 INODE 控制块信息，INODE 控制块用于存储段的属性信息，每个 INODE 控制块对应一个段。INODE 页结构包括通用页头、INODE 页链接、以及 INODE 控制块信息。INODE 控制块结构包括段号、半满簇中使用页的个数、空闲簇链表、半满簇链表、满簇链表。

20.9.2.1 数据类型

DBMS_PAGE 包的 INODE 页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
INODE_PH_T 类型	用于承载 INODE 页头的所有信息，包括：所属表空间 ID、当前页文件 ID、当前页页号、前一个页的文件 ID、前一个页页号、后一个页文件 ID、后一个页页号、页类型、校验和、LSN
INODE_PAGE_LINK_T 类型	用于承载 INODE 页的链接信息，包括：前一个 INODE 页的文件 ID、页号与偏移量，后一个 INODE 页的文件 ID、页号与偏移量
INODE_T 类型	用于承载 INODE 项的所有信息，包括：段 ID、半满簇中使用页数、空闲簇个数、第一个空闲簇描述项的文件 ID、第一个空闲簇描述项的页号、到下一个空闲簇描述项的偏移量、最后一个空闲簇描述项的文件 ID、最后一个空闲簇描述项的页号、到下一个空闲簇描述项的偏移量、半满簇个数、第一个半满簇描述项的文件 ID、第一个半满簇描述项的页号、到下一个半满簇描述项的偏移量、最后一个半满簇描述项的文件 ID、最后一个半满簇描述项的页号、到下一个半满簇描述项的偏移量、满簇个数、第一个满簇描述项的文件 ID、第一个满簇描述项的页号、到下一个满簇描述项的偏移量、最后一个满簇描述项的文件 ID、最后一个满簇描述项的页号、到下一个满簇描述项的偏移量
INODE_PH_ARR_T 类型	INODE_PH_T 类型的数组
INODE_ARR_T 类型	INODE_T 类型的数组

20.9.2.2 相关方法

DBMS_PAGE 包中的 INODE 页部分所包含的过程和函数详细介绍如下：

1. DBMS_PAGE. IPAGE_HEAD_LOAD/ IPAGE_HEAD_GET

定义：

```
PROCEDURE IPAGE_HEAD_LOAD(
PAGE IN VARBINARY,
```

```
INFO OUT INODE_PH_T
);
```

定义：

```
FUNCTION IPAGE_HEAD_GET(
PAGE IN VARBINARY
)RETURN INODE_PH_T;
```

功能说明：

获得 INODE 页中页头信息。过程和函数功能相同。

参数说明：

PAGE：页内容。

2. DBMS_PAGE. IPAGE_NTH_INODE_LOAD/ IPAGE_NTH_INODE _GET

定义：

```
PROCEDURE IPAGE_NTH_INODE_LOAD(
PAGE IN VARBINARY,
N IN SMALLINT,
INFO OUT INODE_T
);
```

定义：

```
FUNCTION IPAGE_NTH_INODE_GET(
PAGE IN VARBINARY,
N IN SMALLINT
)RETURN INODE_T;
```

功能说明：

获得 INODE 页中第 N 个 INODE 项的信息。过程和函数功能相同。

参数说明：

PAGE：页内容；

N：第几个 INODE 项。

3. DBMS_PAGE. IPAGE_ALL_INODES_LOAD/ IPAGE_ALL_INODES _GET

定义：

```
PROCEDURE IPAGE_ALL_INODES_LOAD(
PAGE IN VARBINARY,
INFO OUT INODE_ARR_T
);
```

定义：

```
FUNCTION IPAGE_ALL_INODES_GET(
PAGE IN VARBINARY
)RETURN INODE_ARR_T;
```

功能说明：

获得 INODE 页中所有 INODE 项的信息。过程和函数功能相同。

参数说明：

PAGE：页内容。

20.9.2.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取某 INODE 页的页头信息

```
DECLARE
PAGE8 VARBINARY;
INFO DBMS_PAGE.INODE_PH_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.INODE_PH_T[1];
SF_PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_HEAD_LOAD(PAGE8,INFO[1]);
```

```
SELECT * FROM ARRAY INFO;
END;
```

结果如下 (具体数值取决于实际状况)：

```
TSID SELF_FILE_ID  SELF_PAGE_NO       PREV_FILE_ID PREV_PAGE_NO       NEXT_FILE_ID
NEXT_PAGE_NO      PAGE_TYPE     CHECKSUM      LSN
0      0      8      -1     -1     0      9      17     0      15270
```

例 2 获取某 INODE 项的信息

```
DECLARE
PAGE8 VARBINARY;
INFO DBMS_PAGE.INODE_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.INODE_T[10];
SF_PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_NTH_INODE_LOAD(PAGE8,1,INFO[1]);
SELECT * FROM ARRAY INFO;
END;
```

执行结果，显示 INFO 中的内容是一个 INODE _T 记录类型的查询结果， INODE_T 中的信息是获取的某 INODE 页中的某个 INODE 项的信息(具体数值取决于实际状况)

例 3 获取某 INODE 页中所有 INODE 项信息

```
DECLARE
PAGE8 VARBINARY;
INFO DBMS_PAGE.INODE_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.INODE_T[150];
SF_PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_ALL_INODES_LOAD(PAGE8,INFO);
SELECT * FROM ARRAY INFO;
END;
```

执行结果，显示 INFO 中的内容也就是一个 INODE_ARR_T 数组的查询结果，INODE_ARR_T 数组中的信息是获取的某 INODE 页中所有 INODE 项的信息(具体数值取决于实际状况)

20.9.3 描述页

描述页是用来显示簇信息的页。主要包括簇的上一个链接簇描述项的文件 ID、上一个链接簇描述项的页号、上一个链接簇描述项的偏移、下一个链接簇描述项的文件 ID、下一个链接簇描述项的页号、下一个簇链接描述项的偏移、簇使用情况、本簇所属段 ID 和页使用情况等信息。

20.9.3.1 数据类型

DBMS_PAGE 包的描述页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
DESC_ITEM_T 类型	用于记录描述项信息，包括：上一个链接簇描述项的文件 ID、上一个链接簇描述项的页号、上一个链接簇描述项的偏移、下一个链接簇描述项的文件 ID、下一个链接簇描述项的页号、下一个簇链接描述项的偏移、簇使用情况、本簇所属段 ID 和页使用情况
DESC_ITEM_ARR_T 类型	DESC_ITEM_T 类型数组

20.9.3.2 相关方法

DBMS_PAGE 包中的描述页部分所包含的过程和函数详细介绍如下：

1. SPAGE_NTH_DESC_LOAD/SPAGE_NTH_DESC_GET

定义：

```
PROCEDURE SPAGE_NTH_DESC_LOAD(
PAGE IN VARBINARY,
NTH IN SMALLINT,
DESC_ITEM OUT DESC_ITEM_T
);
```

定义：

```
FUNCTION SPAGE_NTH_DESC_GET(
PAGE    VARBINARY,
NTH    SMALLINT
)
RETURN DESC_ITEM_T;
```

功能说明：

获得描述页信息。

参数说明：

PAGE：调用接口获取，要判定是否是描述页；

NTH：用户输入的第 NTH 个描述页。

2. SPAGE_ALL_DESC_LOAD/SPAGE_ALL_DESC_GET

定义：

```
PROCEDURE SPAGE_ALL_DESC_LOAD(
PAGE IN VARBINARY,
DESC_ITEM OUT DESC_ITEM_ARR_T
);
```

调用做好的 LOAD，获取页信息，输出所有描述页信息。

定义：

```
FUNCTION    XDESC_ALL_DESC_GET(
PAGE    VARBINARY
)
RETURN DESC_ITEM_ARR_T;
```

功能说明：

获得所有描述页信息，一次最多可以查看 256 个描述页信息。

参数说明：

PAGE：调用接口获取，要判定是否是描述页；

20.9.3.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取第 1 个描述页信息

```
DECLARE
    PAGE     VARBINARY;
    I        SMALLINT;
    DESC_ITEM_ARR DBMS_PAGE.DESC_ITEM_ARR_T;
BEGIN
    SF_PAGE_LOAD(0, 0, 0, PAGE);
    I = 0;
    DESC_ITEM_ARR = NEW DBMS_PAGE.DESC_ITEM_T[1];
    SF_XDESC_NTH_DESC(
        PAGE,
        I,
        DESC_ITEM_ARR[1].PREV_FILE_ID,
        DESC_ITEM_ARR[1].PREV_PAGE_NO,
        DESC_ITEM_ARR[1].PREV_OFFSET,
        DESC_ITEM_ARR[1].NEXT_FILE_ID,
        DESC_ITEM_ARR[1].NEXT_PAGE_NO,
        DESC_ITEM_ARR[1].NEXT_OFFSET,
```

```
            DESC_ITEM_ARR[1].STATE,
            DESC_ITEM_ARR[1].SEGID,
            DESC_ITEM_ARR[1].DES_BITMAP);
        SELECT * FROM ARRAY DESC_ITEM_ARR;
    END;
```

将打印如下信息(具体数值取决于实际状况)：

```
    PREV_FILE_ID PREV_PAGE_NO PREV_OFFSET NEXT_FILE_ID NEXT_PAGE_NO NEXT_OFFSET
STATE SEGID DES_BITMAP
    -1   -1   -1   0   2048   148   1202   0   0000000000000111
```

例 2 获取 0 号描述页的第 0 到第 5 个描述项

```
    DECLARE
        PAGE            VARBINARY;
        I               SMALLINT;
        DESC_ITEM_ARR   DBMS_PAGE.DESC_ITEM_ARR_T;
    BEGIN
        SF_PAGE_LOAD(0, 0, 0, PAGE);
        DESC_ITEM_ARR = NEW DBMS_PAGE.DESC_ITEM_T[6];
        FOR I IN 1..6 LOOP
            SF_XDESC_NTH_DESC(
                PAGE,
                I - 1,
                DESC_ITEM_ARR[I].PREV_FILE_ID,
                DESC_ITEM_ARR[I].PREV_PAGE_NO,
                DESC_ITEM_ARR[I].PREV_OFFSET,
                DESC_ITEM_ARR[I].NEXT_FILE_ID,
                DESC_ITEM_ARR[I].NEXT_PAGE_NO,
                DESC_ITEM_ARR[I].NEXT_OFFSET,
                DESC_ITEM_ARR[I].STATE,
                DESC_ITEM_ARR[I].SEGID,
                DESC_ITEM_ARR[I].DES_BITMAP);
        END LOOP;
        SELECT * FROM ARRAY DESC_ITEM_ARR;
    END;
```

将打印如下信息(具体数值取决于实际状况)：

```
    PREV_FILE_ID PREV_PAGE_NO PREV_OFFSET NEXT_FILE_ID NEXT_PAGE_NO NEXT_OFFSET
STATE SEGID DES_BITMAP
    -1   -1   -1   0   2048   148   1202   0   0000000000000111
    -1   -1   -1   -1   -1   -1   1202   1   0011111111111111
    -1   -1   -1   -1   -1   -1   1202   3   0111111111111111
    0   2048   340   -1   -1   -1   1202   4   0010100000000001
    -1   -1   -1   -1   -1   -1   1202   5   0011111111111111
    -1   -1   -1   -1   -1   -1   1202   7   0011111111111111
```

例 3 获取 0 号页所有的描述项

```
    DECLARE
        PAGE            VARBINARY;
        DESC_ITEM_ARR   DBMS_PAGE.DESC_ITEM_ARR_T;
    BEGIN
        SF_PAGE_LOAD(0, 0, 0, PAGE);
        DBMS_PAGE.SPAGE_ALL_DESC_LOAD(PAGE, DESC_ITEM_ARR);
        SELECT * FROM    ARRAY DESC_ITEM_ARR;
    END;
```

将打印 0 号页所有的描述项的信息，共 256 条，记录格式同上图。

20.9.4 控制页

让用户更直观的查看数据库表空间的控制页的信息。

20.9.4.1 数据类型

DBMS_PAGE 包的控制页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释

PH_T 类型	页头记录类型，包括：表空间 ID、本文件 ID、本文件的 PAGE NO、前一个文件的 FILE ID、前一个文件的 PAGE NO、下一个文件的 FILE ID、下一个文件的 PAGE NO、文件类型、CHECKSUM 检查点、LSN 该页实际刷盘的位置信息。当表空间 ID，文件 ID 均为 0，那 PAGE NO 从 0-6 均是控制页，分别为 0 号控制页，1 号 ID 页，2 号 SEQ 页，3 号提交事物的 LAST_LCN 页，4 号系统信息页，5 号复制关系已处理的 LCN 页，6 号保留页。PAGE NO 是从 0 开始，若 PAGE NO 为-1 代表该页为空。若 FILE ID 为-1 则代表该项所代表的 FILE 为空
P0_TSH_T 类型	0 号控制页的表空间头记录类型，包括：LSN、空闲的 INODE 页的数量、INODE FREE 链表中第一个文件号、INODE FREE 链表中第一个页号、INODE FREE 链表中第一个页偏移、INODE FREE 链表中最后一个文件号、INODE FREE 链表中最后一个页号、INODE FREE 链表中最后一个页偏移、全满的 INODE 页的数量、INODE FULL 链表中第一个文件号、INODE FULL 链表中第一个页号、INODE FULL 链表中第一个页偏移、INODE FULL 链表中最后一个文件号、INODE FULL 链表中最后一个页号、INODE FULL 链表中最后一个页偏移、空闲簇的数量、空闲簇链表中第一个文件号、空闲簇链表中第一个页号、空闲簇链表中第一个页偏移、空闲簇链表中最后一个文件号、空闲簇链表中最后一个页号、空闲簇链表中最后一个页偏移、半满簇的数量、半满簇链表中第一个页 ID、半满簇链表中第一个 PAGE NO、半满簇链表中第一个页的偏移、半满簇链表中最后一个页 ID、半满簇链表中最后一个 PAGE NO、半满簇链表中最后一个页的偏移、SYSINDEX 的根页的位置、SYSOBJ 的根页、SYSOBJ 页的位置、一个簇的文件头的位置、LOG 日志文件头的位置
P0_FH_T 类型	0 号控制页的文件头记录类型，包括：一个文件中页的数量、空闲页的页号
P1_INFO_T 类型	1 号控制页的描述项记录类型，包括：下一个段 ID、下一个表 ID、下一个视图 ID、下一个用户 ID、下一个索引 ID、下一个存储过程 ID、下一个函数 ID、下一个角色 ID、下一个文件 ID、下一个触发器 ID、下一个约束 ID、下一个模式 ID、下一个序列 ID、下一个 POLICY ID、下一个 DBLINK ID、下一个事务 ID、下一个跟踪 ID、下一个事件 ID、下一个 OPERATOR ID、下一个警告 ID、下一个作业 ID、下一个作业步骤 ID、下一个作业调度 ID、下一个异步复制 ID、下一个包 ID、下一个作业步骤 ID、下一个密码引擎 ID、下一个 OBJECTTYPE ID、下一个 SYNONY ID、下一个 BLOB ID、下一个 MAL ID、下一个 TAG ID、下一个 GROUP ID、下一个 INST ID、下一个 REP ID、下一个未被使用的组 ID、下一个 NEW TRX ID
SEQ_ITEM_T 类型	2 号控制页的 SEQ 记录类型，包括：该页的使用情况、SEQ 的值
CPAGE_SEQ_HEAD_T 类型	2 号控制页的页头信息类型，包括：申请下一页所在的表空间 ID、申请下一页所在的文件 ID、申请下一页的 PAGENO、页中已使用的 SEQ 项的数量
CPAGE_SEQ_INFO_T 类型	每个页中使用多少个 SEQ 项的数量类型，包括：该页所在的表空间 ID、该页所在的文件 ID、该页的页号、该页中已使用的 SEQ 项的数量
P3_HEAD_T 类型	3 号控制页页头记录类型，包括：LAST_LCN 页的表空间 ID、LAST_LCN 页的文件 ID、LAST_LCN 页的文件号、LAST LCN 的数量
P3_ITEM_T 类型	3 号控制页的描述项记录类型，包括：该页的使用情况、前一个 LCN 的 ID、前一个 TRX 的 ID、前一个 LCN
P4_DES_T 类型	4 号控制页的描述项记录类型，包括：版本号、大小写敏感、编码方式、空串是否等同于 NULL、SITE MAGIC 站点魔数、PRIKEY 解密服务主密钥的 RSA 私钥文件路径、SVR KEY 服务器主密钥、SYSTEM DB KEY 系统数据库的 KEY、数据库模式、LOG REP TYPE 复制实例类型日志、COORDINATOR ADDR 协调器地址、COORDINATOR PORT 协调器端口、时区

P5_HEAD_T 类型	5 号控制页页头记录类型，包括：LOG FILE1 的长度以 KB 为单位、LOG FILE2 的长度以 KB 为单位、LOG WRITE FILE 的 SEQ、LOG READ FILE 的 SEQ、LOG READ FILE 的 OFFSET、复制关系已处理的 LAST LCN 的数量
P5_ITEM_T 类型	5 号控制页的描述项记录类型，包括：该项是否已被使用、REP ID2、上一个 LCN 项
PH_ARR_T 类型	PH_T 类型的数组
P0_FH_ARR_T 类型	P0_FH_T 类型的数组
P1_INFO_ARR_T 类型	P1_INFO_T 类型的数组
CPAGE_SEQ_HEAD_ARR_T 类型	CPAGE_SEQ_HEAD_T 类型的数组
CPAGE_SEQ_INFO_ARR_T 类型	CPAGE_SEQ_INFO_T 类型的数组
SEQ_ITEM_ARR_T 类型	SEQ_ITEM_T 类型的数组
P3_HEAD_ARR_T 类型	P3_HEAD_T 类型的数组
LAST_LCN_PAGE_HEAD_ARR_T 类型	P3_HEAD_T 类型的数组
P3_ITEM_ARR_T 类型	P3_ITEM_T 类型的数组
P4_DES_ARR_T 类型	P4_DES_T 类型的数组
P5_HEAD_ARR_T 类型	P5_HEAD_T 类型的数组
P5_ITEM_ARR_T 类型	P5_ITEM_T 类型的数组

20.9.4.2 相关方法

DBMS_PAGE 包中的控制页部分所包含的过程和函数详细介绍如下：

1. PAGE_HEAD_INFO

定义：

```
PROCEDURE PAGE_HEAD_LOAD (
PAGE IN VARBINARY,
PH_INFO OUT PH_T
);
```

定义：

```
FUCNTION PAGE_HEAD_GET (
PAGE IN VARBINARY
) RETURN PH_T;
```

功能说明：

获得通用页头信息。过程和函数功能相同。

参数说明：

PAGE：页内容。

2. CTL_PAGE0_TS_HEAD_LOAD/ CTL_PAGE0_TS_HEAD_GET

定义：

```
PROCEDURE CTL_PAGE0_TS_HEAD_LOAD (
GHEAD OUT P0_TSH_T
);
```

定义：

```
FUNCTION     CTL_PAGE0_TS_HEAD_GET
RETURN P0_TSH_T;
```

功能说明：

获得 0 号页表空间头的信息。

3. CTL_PAGE0_FH_LOAD/ CTL_PAGE0_FH_GET

定义：

```
PROCEDURE CTL_PAGE0_FH_LOAD (
FHEAD OUT P0_FH_T
);
```

定义：

```
FUNCTION     CTL_PAGE0_FH_GET
RETURN P0_FH_T;
```

功能说明：

获得 0 号页文件头的信息。

4. CTL_PAGE1_DES_LOAD/ CTL_PAGE1_DES_GET

定义：

```
PROCEDURE    CTL_PAGE1_DES_LOAD (
P1DES OUT P1_INFO_T
);
```

定义：

```
FUNCTION     CTL_PAGE1_DES_GET
RETURN P1_INFO_T;
```

功能说明：

获得 1 号控制页的描述信息。

5. CPAGE_SEQ_HEAD_LOAD/ CPAGE_SEQ_HEAD_GET

定义：

```
PROCEDURE CPAGE_SEQ_HEAD_LOAD (
      PAGE2 IN    VARBINARY,
      P2DES OUT CPAGE_SEQ_HEAD_T
);
```

定义：

```
FUNCTION CPAGE_SEQ_HEAD_GET (
PAGE2 IN VARBINARY
) RETURN CPAGE_SEQ_HEAD_T;
```

功能说明：

获得 SEQ 页的 SEQ 页头信息。

参数说明：

PAGE2：SEQ 页内容。

6. SEQ_ITEM_ALL_LOAD/ SEQ_ITEM_ALL_GET

定义：

```
PROCEDURE SEQ_ITEM_ALL_LOAD (
PAGE2 IN    VARBINARY,
P2DES OUT SEQ_ITEM_ARR_T
);
```

定义：

```
FUNCTION SEQ_ITEM_ALL_GET (
PAGE2 IN VARBINARY
) RETURN SEQ_ITEM_ARR_T;
```

功能说明：

获得该页的所有 SEQ 项的信息。

参数说明：

PAGE2：SEQ 页内容。

7. CTL_PAGE_SEQ_COUNT_N_LOAD/ CTL_PAGE_SEQ_COUNT_N_GET

定义：

```
PROCEDURE CTL_PAGE_SEQ_COUNT_N_LOAD (
OFFSET_N      IN      SMALLINT DEFAULT 0,
N                  IN      SMALLINT DEFAULT 100,
SEQCOUNT       OUT CPAGE_SEQ_INFO_ARR_T
);
```

定义：

```
FUNCTION CTL_PAGE_SEQ_COUNT_N_GET (
OFFSET_N       IN SMALLINT DEFAULT 0,
N                  IN SMALLINT DEFAULT 100
) RETURN CPAGE_SEQ_INFO_ARR_T;
```

功能说明：

获得系统中从 OFFSET_N 开始往后 N 页的 SEQ 页信息并返回（最多 100 页）。

参数说明：

OFFSET_N：页偏移量；

N：从 OFFSET_N 开始往后 N 页。

8. CPAGE3_HEAD_GET/ CPAGE3_HEAD_LOAD

定义：

```

PROCEDURE CPAGE3_HEAD_LOAD(
P3_HEAD OUT P3_HEAD_T
);
```

定义：

```
FUNCTION    CPAGE3_HEAD_GET
RETURN P3_HEAD_T;
```

功能说明：

获得 3 号控制页的页头格式信息并返回

9. LAST_LCN_PAGE_HEAD_LOAD /LAST_LCN_PAGE_HEAD_GET

定义：

```
FUNCTION LAST_LCN_PAGE_HEAD_GET (
OFFSET INT DEFAULT 0,
NUM         SMALLINT DEFAULT 100
) RETURN LAST_LCN_PAGE_HEAD_ARR_T;
```

定义：

```
PROCEDURE LAST_LCN_PAGE_HEAD_LOAD (
OFFSET                  INT DEFAULT 0,
NUM                      INT DEFAULT 100,
P_HEAD_ARR OUT LAST_LCN_PAGE_HEAD_ARR_T
);
```

功能说明：

获得系统中从 OFFSET_N 开始往后 N 页的 SEQ 页信息（最多 100 页）。

参数说明：

OFFSET：页偏移量；

NUM：从 OFFSET_N 开始往后 N 页。

10. LAST_LCN_PAGE_ITEM_LOAD /LAST_LCN_PAGE_ITEM_GET

定义：

```
FUNCTION LAST_LCN_PAGE_ITEM_GET (
TS_ID SMALLINT,
FILE_ID SMALLINT,
PAGE_NO INT
) RETURN P3_ITEM_ARR_T;
```

定义：

```
PROCEDURE LAST_LCN_PAGE_ITEM_LOAD (
TS_ID SMALLINT,
FILE_ID SMALLINT,
PAGE_NO INT,
P3_ITEM_ARR OUT P3_ITEM_ARR_T
);
```

功能说明：

获得相应页的 ITEM 信息。

参数说明：

TS_ID：表空间号；

FILE_ID：文件号；

PAGE_NO：页号。

11. CPAGE3_LCN_MAX_CNT_GET

定义：

```
FUNCTION CPAGE3_LCN_MAX_CNT_GET
RETURN INT;
```

功能说明：

返回一个 LAST LCN 页能存的最大 LAST LCN ITEM 项的个数

12. CPAGE4_DES_LOAD/ CPAGE4_DES_GET

定义：

```
PROCEDURE CPAGE4_DES_LOAD(
P4_INFO OUT P4_DES_T
);
```

定义：

```
FUNCTION    CPAGE4_DES_GET
RETURN P4_DES_T;
```

功能说明：

获得 4 号控制页的描述信息。

13. CPAGE5_HEAD_LOAD/ CPAGE5_HEAD_GET

定义：

```
PROCEDURE CPAGE5_HEAD_LOAD(
P5_HEAD OUT P5_HEAD_T
```

```
);
```

定义：

```
FUNCTION        CPAGE5_HEAD_GET
RETURN P5_HEAD_T;
```

功能说明：

获得 5 号控制页的页头格式信息。

14. CPAGE5_ITEM_LOAD/ CPAGE5_ITEM_GET

定义：

```
PROCEDURE CPAGE5_ITEM_LOAD (
P5DES OUT P5_ITEM_ARR_T
);
```

定义：

```
FUNCTION        CPAGE5_ITEM_GET
RETURN P5_ITEM_ARR_T;
```

功能说明：

获得 5 号控制页的描述信息。

20.9.4.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 取得通用页头信息 0-7

```
DECLARE
PAGE0           VARBINARY;
PAGE1           VARBINARY;
PHINFO          DBMS_PAGE.PH_T;
PHINFO_ARR   DBMS_PAGE.PH_ARR_T;
BEGIN

PHINFO_ARR = NEW DBMS_PAGE.PH_T[8];
FOR I IN 1..8 LOOP
      DBMS_PAGE.PAGE_LOAD(0,0,I-1,PAGE0);
      --DBMS_PAGE.PAGE_HEAD_LOAD(PAGE0,PHINFO);
      PHINFO_ARR[I] = DBMS_PAGE.PAGE_HEAD_GET(PAGE0);
END LOOP;
SELECT * FROM ARRAY PHINFO_ARR;
END;
/
```

结果如下：

```
TS_ID SELF_FILE_ID SELF_PAGE_NO PREV_FILE_ID PREV_PAGE_NO NEXT_FILE_ID NEXT_PAGE_NO PAGE_TYPE CHECKSUM LSN
0 0 0 -1 -1 -1 -1 19 0 13971
0 0 1 -1 -1 -1 -1 -1 0 14681
0 0 2 -1 -1 -1 -1 -1 0 1
0 0 3 -1 -1 -1 -1 99 0 1
0 0 4 -1 -1 -1 -1 100 0 1
0 0 5 -1 -1 -1 -1 101 0 1
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

20.10 DBMS_METADATA 包

GET_DDL 函数用于获取数据库对象表、视图、索引、全文索引、存储过程、函数、包、序列、同义词、约束、触发器的 DDL 语句。

20.10.1 相关方法

1. GET_DDL

定义：

```
FUNCTION GET_DDL(
OBJECT_TYPE IN VARCHAR(30),
NAME            IN VARCHAR(128),
SCHNAME         IN VARCHAR(128) DEFAULT NULL
) RETURN CLOB
```

功能说明：

获取指定对象元数据中的 DDL 语句。

参数说明：

OBJECT_TYPE：对象类型。包括表、视图、索引、全文索引、存储过程、函数、包、等，详情请见 OPEN 参数说明。其中，OBJECT_TYPE 只能为大写；

NAME：对象名称，区分大小写；

SCHEMA：模式，默认是当前用户模式；

返回值：以 DDL 返回对象元数据中的 DDL 语句。

错误处理：

INVALID_ARGVAL：如果输入参数中存在空值或非法值。

OBJECT_NOT_FOUND：如果指定的对象在数据库中不存在。

2. ”OPEN”

定义：

```
FUNCTION OPEN (
    OBJECT_TYPE    IN VARCHAR2
) RETURN NUMBER;
```

功能说明：

打开对象类型的句柄

参数说明：

OBJECT_TYPE：可返回的对象类型名称，合法的类型名称见下表。

表 OPEN 函数可返回的对象类型名称

类型名称	含义	属性	说明
AUDIT	审计	D	
AUDIT_OBJ	审计对象	DG	
CLASS	用户自定义类型	SN	默认返回类头类体
CLASS_HEAD	类型名	SN	无
CLASS_BODY	类型体	SN	无
COL_STATISTICS	列统计	D	
COMMENT	注释	D	
CONSTRAINT	约束	SND	不包括聚集主键和非空约束
CONTEXT	全文索引	N	无
DATABASE_EXPORT	数据库下的所有对象	H	库级导出
DB_LINK	数据库链接	SN	因此类对象具有所有者,因此将其视为模式级对象。 对于公有连接，它们的所有者是 PUBLIC；对于私有链接，它们的

			创建者就是它们的所有者
FUNCTION	存储函数	SN	无
INDEX	索引	SND	不包括系统内部定义的索引
INDEX _STATISTICS	索引统计	D	
JOB	任务	N	无
OBJECT_GRANT	对象权限	DG	无
PACKAGE	包	SN	默认返回包头包体
PKG_SPEC	包头	SN	无
PKG_BODY	包体	SN	无
POLICY	策略	D	无
PROCEDURE	存储过程	SN	无
ROLE	角色	N	无
ROLE_GRANT	角色权限	G	无
SCHEMA _EXPORT	模式下的所有对象	H	模式级导出
SEQUENCE	序列	SN	无
SYNONYM	同义词	见说明	私有同义词为模式对象，公有同义词为命名对象
SYSTEM_GRANT	系统权限	G	无
TABLE	表	SN	无
TABLE_STATISTICS	表统计信息	D	无
TABLE_EXPORT	表及与其相关的元数据	H	表级导出
TABLESPACE	表空间	N	无
TRIGGER	触发器	SND	无
USER	用户	N	无
VIEW	视图	SN	无

注：属性列中，S 表示模式对象，N 表示命名的对象，D 表示依赖对象，G 表示被授权的对象，H 表示包含不同类型的对象。

返回值：

不透明的句柄。该句柄将用于 SET_FILTER, SET_PARSE_ITEM, FETCH_XXX, AND CLOSE。

错误处理：

INVALID_ARGVAL：参数为 NULL，或者参数值非法。

3. ”CLOSE”

定义：

```
PROCEDURE CLOSE (
    HANDLE    IN NUMBER
);
```

功能说明：关闭 OPEN 打开的句柄，并清理相关环境。

参数说明：

HANDLE：OPEN 函数返回的句柄。

错误处理：

INVALID_ARGVAL：参数为 NULL，或者参数值非法。

4. FETCH_DDL()

定义：

```
FUNCTION FETCH_DDL (
```

```
    HANDLE    IN NUMBER
)RETURN KU$_DDLS;
```

功能说明：

该函数返回符合 OPEN，SET_FILTER，SET_PARSE_ITEM 设定条件的对象元数据。

参数说明：

HANDLE：OPEN 函数返回的句柄。

返回值：

对象的元数据或者当所有对象都已返回后返回 NULL。

使用说明：

从表 KU$_DDLS 中返回 DDL 语句，表 KU$_DDLS 的每一行在 DDLTEXT 列包含一条 DDL 语句；如果使用解析的话，解析出的对象属性将从 PARSEDITEM 列返回。

错误处理：

INVALID_ARGVAL：参数为 NULL,或者参数值非法。

5. FETCH_CLOB()

```
FUNCTION    FETCH_CLOB(
      HANDLE IN INT
)RETURN CLOB;
```

函数功能：

该函数返回符合 OPEN，SET_FILTER 设定条件的对象元数据。

使用说明：

HANDLE：OPEN 函数返回的句柄。

返回值：

对象的元数据或者当所有对象都已返回后返回 NULL。

使用说明：

将对象以 CLOB 返回。

异常：

INVALID_ARGVAL：参数为 NULL，或者参数值非法。

6. GET_GRANTED_DDL()

定义：

```
FUNCTION GET_GRANTED_DDL (
      OBJECT_TYPE        IN VARCHAR2,
      GRANTEE            IN VARCHAR2,
      OBJECT_COUNT       IN NUMBER      DEFAULT 10000
)RETURN CLOB;
```

功能说明：

该函数用于返回对象的授权语句。

参数说明：

OBJECT_TYPE：对象类型。参数要与 OPEN 的对象类型参数相同，不能为 HETEROGENEOUS 对象类型。

GRANTEE：被赋予权限的对象。

OBJECT_COUNT：返回对象个数的最大值。

返回值：

以 DDL 返回对象的元数据

异常:

INVALID_ARGVAL：参数值为 NULL，或者参数值非法。

OBJECT_NOT_FOUND： 指定的对象数据库中不存在。

7. GET_DEPENDENT_DDL()

定义：

```
FUNCTION GET_DEPENDENT_DDL (
OBJECT_TYPE              IN VARCHAR2,
BASE_OBJECT_NAME       IN VARCHAR2,
BASE_OBJECT_SCHEMA    IN VARCHAR2 DEFAULT NULL,
```

```
    OBJECT_COUNT            IN NUMBER     DEFAULT 10000
    )RETURN CLOB;
```

功能说明：

该函数用于返回依赖对象的 DDL 语句。

参数说明：

OBJECT_TYPE: 对象类型。参数要与 OPEN 的对象类型参数相同，不能为 HETEROGENEOUS 对象类型。

BASE_OBJECT_NAME：基对象名称。内部使用 BASE_OBJECT_NAME 过滤器过滤。

BASE_OBJECT_SCHEMA：基对象模式。内部使用 BASE_OBJECT_SCHEMA 过滤器过滤。

OBJECT_COUNT：返回对象个数的最大值。

返回值：

以 DDL 返回对象的元数据。

异常:

INVALID_ARGVAL：参数值为 NULL，或者参数值非法。

OBJECT_NOT_FOUND： 指定的对象数据库中不存在。

8. SET_FILTER()

定义：

```
PROCEDURE SET_FILTER (
    HANDLE                  IN  NUMBER,
    NAME                    IN  VARCHAR2,
    VALUE                   IN  VARCHAR2
);
```

功能说明：该函数用于过滤待返回的对象。

参数说明：

HANDLE：句柄，由 DBMS_METADATA.OPEN 输出。

NAME：过滤器的名称，详见下表。

VALUE：过滤器的值，与 NAME 相对应，详见下表。

表 SET_FILTER 过程中的过滤器名称

对象类型	名称	数据类型	含义
可命名对象	NAME	文本	通过对象名过滤
可命名对象	NAME_EXPR	文本表达式	通过对象名的 where 表达式过滤，满足条件的返回
可命名对象	EXCLUDE_NAME_EXPR	文本表达式	通过对象名的 where 表达式过滤，不满足条件的返回
模式对象	SCHEMA	文本	通过模式名过滤，如果是公有同义词则指定其模式为 PUBLIC
模式对象	SCHEMA_EXPR	文本表达式	通过对象名的 where 表达式过滤，默认为当前模式
表、索引、表级导出	TS	文本	通过表空间名过滤
表、索引、表级导出	TABLESPACE_EXPR	文本表达式	通过表空间名的 where 表达式过滤
依赖对象	BASE_OBJECT_NAME	文本	通过基对象名过滤。
依赖对象	BASE_OBJECT_SCHEMA	文本	通过基对象模式过滤。如果使用 BASE_OBJECT_NAME 过滤，默认是当前模式。
依赖对象	BASE_OBJECT_NAME_EXPR	文本表达式	通过基对象名的 where 表达式过滤，满足条件的返回。不适用于模式和库级触发器。

依赖对象	EXCLUDE_BASE_OBJECT_NAME_EXPR	文本表达式	通过基对象名的 where 表达式过滤，不满足条件的返回。不适用于模式和库级触发器。
依赖对象	BASE_OBJECT_SCHEMA_EXPR	文本表达式	通过基对象模式的 where 表达式过滤。
依赖对象	BASE_OBJECT_TYPE	文本	通过基对象的对象类型过滤。
依赖对象	BASE_OBJECT_TYPE_EXPR	文本表达式	通过基对象的对象类型表达式过滤。
依赖对象	BASE_OBJECT_TS	文本	通过基对象的表空间过滤。
依赖对象	BASE_OBJECT_TS_EXPR	文本表达式	通过基对象的表空间 where 表达式过滤。
被授权对象	GRANTEE	文本	被授权的用户或角色。AUDIT_obj 只能过滤跟 grantee 有关的 其他 grant 过滤器不适用。
被授权对象	ROLE_NAME	文本	通过权限或角色的名称过滤
被授权对象	ROLE_NAME_EXPR	文本表达式	通过权限或角色的 where 表达式名称过滤
被授权对象	GRANTEE_EXPR	文本表达式	通过被授权者名称的 where 表达式过滤，符合条件的返回
被授权对象	EXCLUDE_GRANTEE_EXPR	文本表达式	通过被授权者名称的 where 表达式过滤，不符合条件的返回
对象权限(OBJECT_GRANT)	GRANTOR	文本	通过授权者过滤，对于系统用户（sysdba sysauditor syssso）赋予的权限 在权限表里面 ID 记录的都是-1，所以设定了一个 grantor 叫"$SUPER",当 filter 中设定 grantor 为 $SUPER 时，系统用户赋予的权限将全部返回 而不区分具体是哪个系统用户
所有对象	CUSTOM_FILTER	文本	用户自定义过滤，要求填写完整的 where 子句，如“NAME = ‘T1’”
模式级导出	SCHEMA	文本	通过模式名过滤
模式级导出	SCHEMA_EXPR	文本表达式	通过模式名 where 表达式过滤，有以下两种情况：1.获取模式对象；2.获取依赖于此模式的对象。默认情况下，将选中当前用户的对象。
表级导出	SCHEMA	文本	通过模式名过滤
表级导出	SCHEMA_EXPR	文本表达式	通过模式名 where 表达式过滤，有以下两种情况：1.获取模式下的表；2.获取依赖于表的对象。默认情况下，将选中当前用户的对象。

表级导出	NAME	文本	通过表名过滤出表及依赖表的对象
表级导出	NAME_EXPR	文本表达式	通过表名的 where 表达式过滤出表及依赖表的对象

注：约束的 BASE_OBJECT_TYPE 是 UTAB，注释的 BASE_OBJECT_TYPE 为 TABLE 或者 VIEW。索引、OBJECT_GRANT 的 BASE_OBJECT_TYPE 为 UTAB，TABLE_STATISTICS，INDEX_STATISTICS，COL_STATISTICS 的 BASE_OBJECT_TYPE 分别对应为 T、I、C。

错误处理：

INVALID_ARGVAL： 参数为 NULL，或者参数值非法。

INVALID_OPERATION：非法操作。在调用 FETCH_XXX 之后，不允许再调用 SET_FILTER 。

INCONSISTENT_ARGS：参数不一致，包括如下状况：

- 过滤器名字与 DBMS_METADATA.OPEN 中指定的类型不匹配。
- 过滤器的名字与 OBJECT_TYPE_PATH 不匹配。
- OBJECT_TYPE_PATH 与 DBMS_METADATA.OPEN 中指定的类型不匹配。
- 输入的过滤器的 VALUE 与期待的数据类型不匹配。

9. SET_PARSE_ITEM()

定义：

```
FUNCTION SET_PARSE_ITEM (
    HANDLE          IN   NUMBER,
    NAME            IN   VARCHAR2,
    OBJECT_TYPE     IN   VARCHAR2 DEFAULT NULL
);
```

功能说明：

该过程用于解析对象的属性。

参数说明：

HANDLE：OPEN 函数返回的句柄。

NAME：被解析对象属性的名称，具体的可被解析的对象属性名称见下表。

OBJECT_TYPE：应用于不同对象的集合，不设置时解析所有的对象。

表 SET_PARSE_ITEM 可解析的对象属性

对象类型	属性名称	含义
所有对象	VERB	如果调用 FETCH_DDL， 表 ku$_ddls 每一行 ddltext 列中的动词将被返回. 如果 ddlText 是 SQL DDL 语句, SQL 中的动词 (如 CREATE, GRANT, AUDIT) 被返回 。 如果 ddlText 是过程 (如 DBMS_METADATA.FETCH_DDL()) 那么 package.procedure-name 被返回
所有对象	OBJECT_TYPE	如果调用 FETCH_DDL， ddlText 是 SQL DDL 语句且包含动词 CREATE 或者 ALTER，DDL 语句中的对象类型将被返回 (如 TABLE，PKG_BODY 等等)。 否则“表 OPEN 函数可返回的对象类型名称”中的对象类型被返回
模式对象	SCHEMA	返回对象的模式，如果不是模式对象则没有返回值
命名的对象	NAME	返回对象的名称，如果不是命名的对象则没有返回值
表、表数据、索引	TABLESPACE	返回对象表空间名称，如果是分区表则返回默认的表空间。 对于 TABLE_DATA 对象, 总是返回行所在的表空间
触发器	ENABLE	返回触发器是禁用还是启用状态
依赖对象	BASE_OBJECT_NAME	返回基对象名称，如果不是依赖对象则没有返回值
依赖对象	BASE_OBJECT_SCHEMA	返回基对象所属的模式，如果不是依赖对象则没有返回值
依赖对象	BASE_OBJECT_TYPE	返回基对象所属类型，如果不是依赖对象则没有返回值

注：对象类型是可解析的对象属性所适用的范围；BASE_OBJECT_NAME，BASE_OBJECT_SCHEMA，BASE_OBJECT_TYPE 这三个属性只解析约束、索引、全文索引、表级/视图级触发器。

使用说明：

FETCH_XXX 函数可以返回对象的 DDL 语句，使用 SET_PARSE_ITEM 则可以返回对象的各个属性，可以多次调用 SET_PARSE_ITEM 来解析和返回多个解析项，返回的解析项存于表 KU$_PARSED_ITEMS 中。

错误处理：

INVALID_ARGVAL：参数为 NULL,或者参数值非法。

INVALID_OPERATION： 非法操作。SET_PARSE_ITEM 只能用于 FETCH_XXX 函数之前。第一次调用 FETCH_XXX 后，不允许再调用 SET_PARSE_ITEM。

10. GET_QUERY()

定义：

```
FUNCTION GET_QUERY (
    HANDLE    IN NUMBER)
 RETURN VARCHAR2;
```

功能说明：该函数用于获取查询文本。

参数说明：

HANDLE: OPEN 函数返回的句柄。

返回值：

将被用于 FENTH_XXX 函数的查询文本。

错误处理：

INVALID_ARGVAL：参数值为 NULL，或非法。

11. SET_COUNT()

定义：

```
PROCEDURE SET_COUNT (
    HANDLE                IN NUMBER,
    VALUE                 IN NUMBER);
```

功能说明：

该过程用于指定一次 FETCH_XXX 函数调用所返回对象个数的最大值。

参数说明：

HANDLE: OPEN 函数返回的句柄；

VALUE：设定的最大值。

错误处理：

INVALID_ARGVAL：参数值为 NULL 或非法；

INVALID_OPERATION：非法操作。SET_COUNT 只能用于第一次调用 FETCH_XXX 函数之前，第一次调用 FETCH_XXX 后，不允许再调用 SET_COUNT。

20.10.2 错误处理

错误、异常情况释义：

INVALID_ARGVAL：非法的参数数据；

OBJECT_NOT_FOUND：未找到对象；

INVALID_OPERATION：无效的过程/函数调用顺序；

INCONSISTENT_ARGS：对象类型与过滤器不匹配。

20.10.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取表和存储过程的 DDL 语句。

```
--建表
CREATE TABLE "SYSDBA"."T1"
(
"C1" CHAR(10) NOT NULL,
"C2" CHAR(10),
CONSTRAINT "PK" PRIMARY KEY("C1")) STORAGE(ON "MAIN", CLUSTERBTR);
--获取对象的 DDL 语句
SELECT DBMS_METADATA.GET_DDL('TABLE','T1','SYSDBA');
```

执行结果：

```
CREATE TABLE "SYSDBA"."T1"
(
"C1" CHAR(10) NOT NULL,
"C2" CHAR(10),
CONSTRAINT "PK" PRIMARY KEY("C1")) STORAGE(ON "MAIN", CLUSTERBTR);
```

例 2 使用 DBMS_METADATA 程序化接口获取元数据

```
CREATE OR REPLACE FUNCTION GET_TABLE_MD RETURN CLOB IS
-- 定义局部变量
H NUMBER; --HANDLE RETURNED BY OPEN
TH NUMBER; -- HANDLE RETURNED BY ADD_TRANSFORM
DOC CLOB;
BEGIN
-- 指定对象类型
H := DBMS_METADATA."OPEN" ('TABLE');

-- 使用过滤器返回特定的对象
DBMS_METADATA.SET_FILTER(H,'SCHEMA','HR');
DBMS_METADATA.SET_FILTER(H,'NAME','TIMECARDS');

 -- 获取对象
DOC := DBMS_METADATA.FETCH_CLOB(H);

 -- 释放资源
DBMS_METADATA."CLOSE" (H);
RETURN DOC;
END;
/
SELECT GET_TABLE_MD;
```

查询结果如下：

```
CREATE TABLE "HR"."TIMECARDS"
  (       "EMPLOYEE_ID" NUMBER(6,0),
          "WEEK" NUMBER(2,0),
                  "JOB_ID" VARCHAR2(10),
              "HOURS_WORKED" NUMBER(4,2),
                FOREIGN KEY ("EMPLOYEE_ID")
                  REFERENCES "HR"."EMPLOYEES" ("EMPLOYEE_ID") ENABLE
     ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOCOMPRESS LOGGING
    STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
    PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT)
    TABLESPACE "EXAMPLE"
```

例 3 使用 DBMS_METADATA 浏览接口获取元数据

```
SELECT DBMS_METADATA.GET_DDL('TABLE','TIMECARDS','HR');
```

查询结果同例 2。

例 4 获取多个对象

```
DROP TABLE MY_METADATA;
CREATE TABLE MY_METADATA (MD CLOB);
CREATE OR REPLACE PROCEDURE GET_TABLES_MD IS
--  定义局部变量
H           NUMBER;               -- HANDLE RETURNED BY 'OPEN'
TH          NUMBER;               -- HANDLE RETURNED BY 'ADD_TRANSFORM'
DOC         CLOB;                 -- METADATA IS RETURNED IN A CLOB
```

```
BEGIN
 -- 指定对象类型
 H := DBMS_METADATA."OPEN" ('TABLE');

 -- 使用过滤器指定模式
 DBMS_METADATA.SET_FILTER(H,'SCHEMA','SYSDBA') ;
 -- 获取对象
 LOOP
    DOC := DBMS_METADATA.FETCH_CLOB(H);
  -- 当没有对象可获取时, FETCH_CLOB 返回 NULL
    EXIT WHEN DOC IS NULL;
    -- 将元数据存入表中
    INSERT INTO MY_METADATA(MD) VALUES (DOC);
    COMMIT;
 END LOOP;
 -- 释放资源
 DBMS_METADATA."CLOSE" (H);
END;
/
CALL GET_TABLES_MD;
SELECT * FROM MY_METADATA;
```

例 5 使用 PARSE ITEMS 解析特定的元数据属性

```
DROP TABLE MY_METADATA;
CREATE TABLE MY_METADATA (
   OBJECT_TYPE     VARCHAR2(30),
   NAME            VARCHAR2(30),
   MD              CLOB);
CREATE OR REPLACE PROCEDURE GET_TABLES_AND_INDEXES IS
-- 定义局部变量
H1          NUMBER;              -- 打开表返回的句柄
DOC         DBMS_METADATA.KU$_DDLS;          -- KU$_DDLS，返回的元数据
DDL         CLOB;                -- 对象的 DDL 语句
PI          DBMS_METADATA.KU$_PARSED_ITEMS;    --包含在 KU$_DD 中的对象返回的解析项
OBJNAME VARCHAR2(30);    -- 解析对象的名称
BEGIN
-- 指定对象类型: TABLE.
 H1 := DBMS_METADATA."OPEN" ('TABLE');

 -- 将表明作为解析项返回
 DBMS_METADATA.SET_PARSE_ITEM(H1,'NAME');

 -- 设置循环: 获取 TABLE 对象
 LOOP
    DOC := DBMS_METADATA.FETCH_DDL(H1);

--当没有对象可获取时，FETCH_CLOB 返回 NULL
    EXIT WHEN DOC IS NULL;

-- 循环 KU$_DDLS 表中的行
    FOR I IN DOC.FIRST..DOC.LAST LOOP
       DDL := DOC(I).DDLTEXT;
       PI := DOC(I).PARSEDITEM;
       -- 循环返回的解析项
       IF PI IS NOT NULL AND PI.COUNT > 0 THEN
          FOR J IN PI.FIRST..PI.LAST LOOP
             IF PI(J).ITEM='NAME' THEN
                OBJNAME := PI(J).VALUE;
             END IF;
          END LOOP;
       END IF;
       -- 将该对象的信息插入表 MY_METADATA 中
       INSERT INTO MY_METADATA(OBJECT_TYPE, NAME, MD)
          VALUES ('TABLE',OBJNAME,DDL);
       COMMIT;
```

```
    END LOOP;
  END LOOP;
  DBMS_METADATA."CLOSE" (H1);
END;
/
CALL GET_TABLES_AND_INDEXES;
SELECT * FROM MY_METADATA;
```

20.11 DBMS_SPACE 包

为了展示所有物理对象（文件、页）和逻辑对象（表空间、簇、段）的存储空间信息。通过 DBMS_SPACE 包来获取表空间、文件、页、簇、段的内容。

20.11.1 数据类型

DBMS_SPACE 包中涉及到的变量和记录类型。如下统一说明：

包内变量和记录类型	解释
TS_T	表空间记录类型，用于记录表空间的信息，包括：表空间 ID、表空间名、表空间类型：1 DB 类型，2 临时文件组、表空间状态、表空间的最大空间、表空间的总大小（页）、包含文件的个数
TS_ARR_T	表空间记录类型数组
TS_ALL_ARR_T	表空间 ID 数组
FILE_T	文件记录类型，用于记录文件的信息，包括：文件路径、文件创建时间 、文件读写状态 1 读，2 写 文件修改的时间、修改的事务 ID 、文件的总大小（M）、文件的空闲大小（M）、数据文件中连续空白页的起始页号、读页个数、写页个数、页大小（K）、读请求个数 、写请求个数 、文件可扩展标记 、文件最大大小（M）、文件每次扩展大小（M）、文件包含的总描述页的数目
FILE_ARR_T	文件记录类型数组
FILE_ALL_ARR_T	文件的 ID 数组
SEG_T	段记录类型，用于记录段的信息，包括：表空间 ID、段 INODE 项的文件 ID、段 INODE 项的页号、段 INODE 项的页偏移、全满簇的个数、半满簇的个数、空闲簇的个数
SEG_ARR_T	段记录类型数组
SEG_ID_ARR_T	段 ID 数组
EXTENT_T	簇记录类型，用于记录簇的信息，包括：表空间号、文件 ID、段 ID、簇状态、簇的起始页号、簇的终止页号、簇的页标记位图、簇描述项所在页号、下一个簇描述项的文件号 ID、下一个簇描述项的页号、下一个簇描述项的的页偏移、上一个簇描述项的文件 ID、上一个簇描述项的页号、下一个簇描述项的页偏移
EXTENT_ARR_T	簇记录类型数组
EXTENT_SIZE	簇的大小（页为单位）
PAGE_ADDR_T	页地址记录类型，用于记录页地址的信息，包括：表空间 ID，文件 ID，页号
PAGE_ADDR_ARR_T	页地址记录类型数组

PAGE_N	描述页能描述的页数

20.11.2 相关方法

DBMS_SPACE 包中包含的过程和函数如下详细介绍：

1. TS_LOAD/ TS_GET

定义：

```
PROCEDURE TS_LOAD(
TSID            IN      SMALLINT,
TS_ARR OUT TS_ARR_T
)
```

定义：

```
FUNCTION    TS_GET (
    TSID     IN      SMALLINT
)RETURN TS_ARR_T;
```

功能说明：

根据输入的表空间 ID，获得表空间信息。过程和函数功能相同。

参数说明：

TSID：表空间 ID；

TS_ARR_T：表空间记录类型数组。

2. TS_N_LOAD / TS_N_GET

定义：

```
PROCEDURE TS_N_LOAD(
NUMBER OUT INT
);
```

定义：

```
FUNCTION TS_N_GET RETURN INT;
```

功能说明：

获得数据库中表空间的个数。过程和函数功能相同。

参数说明：

NUMBER：表空间的个数。

3. TS_ALL_LOAD / TS_ALL_GET

定义：

```
PROCEDURE TS_ALL_LOAD(
TS_ALL_ARR OUT TS_ALL_ARR_T,
OFFSET    IN INT DEFAULT 1,
NUM IN INT DEFAULT 100
);
```

定义：

```
FUNCTION TS_ALL_GET(
OFFSET    IN    INT DEFAULT 0,
NUM           IN      INT DEFAULT 100
) RETURN TS_ARR_T;
```

功能说明：

获得所有表空间的 ID。过程和函数功能相同。

参数说明：

OFFSET：数组起始位置。

NUM：数组长度

TS_ALL_ARR：表空间 ID

4. FILE_LOAD/ FILE_GET

定义：

```
PROCEDURE FILE_LOAD(
TS_ID    IN SMALLINT,
FILE_ID IN SMALLINT,
FILE_ARR OUT FILE_ARR_T
);
```

定义：

```
FUNCTION FILE_GET(
TS_ID    IN SMALLINT,
FILE_ID IN SMALLINT
) RETURN FILE_ARR_T;
```

功能说明：

根据输入的表空间 ID、文件 ID，获得文件信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；

FILE_ID ： 文件号；

FILL_ARR_T：文件记录类型数组。

5. FILE_ALL_LOAD/ FILE_ALL_GET

定义：

```
PROCEDURE FILE_ALL_LOAD(
TS_ID IN SMALLINT,
FILE_ALL_ARR OUT FILE_ALL_ARR_T,
OFFSET    IN INT DEFAULT 1,
NUM IN INT DEFAULT 100
);
```

定义：

```
FUNCTION FILE_ALL_GET(
TS_ID IN SMALLINT,
OFFSET    IN INT DEFAULT 1,
NUM IN INT DEFAULT 100
)RETURN FILE_ALL_ARR_T;
```

功能说明：获得所有文件的 ID。过程和函数功能相同。

参数说明：

TSID： 表空间 ID。

FILE_ALL_ARR：文件的 ID

OFFSET：数组起始位置；

NUM：数组的长度。

6. SEG_LOAD_BY_ID/SEG_GET_BY_ID

定义：

```
PROCEDURE SEG_LOAD_BY_ID(
TS_ID IN SMALLINT,
SEGID IN INT,
SEG_INFO OUT SEG_T
);
```

定义：

```
FUNCTION SEG_GET_BY_ID(
TS_ID IN SMALLINT,
SEGID IN INT
)RETURN SEG_T;
```

功能说明：

根据输入的表空间 ID、文件号、段号、段头获得段信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；
FILE_ID ： 文件号；
SEGID ： 段号；
SEG_INFO：段的信息；
SEG_T：段记录类型。

7. SEG_LOAD_BY_HEADER/ SEG_GET_BY_HEADER

定义：

```
PROCEDURE SEG_LOAD_BY_HEADER(
TS_ID IN SMALLINT,
HEADER_FILE_ID IN SMALLINT,
HEADER_PAGE_NO IN INT,
HEADER_OFFSET IN SMALLINT,
SEG_INFO    OUT SEG_T
);
```

定义：

```
FUNCTION SEG_GET_BY_HEADER(
TS_ID IN SMALLINT,
HEADER_FILE_ID IN SMALLINT,
HEADER_PAGE_NO IN INT,
HEADER_OFFSET IN SMALLINT
)RETURN SEG_T;
```

功能说明：

根据输入的表空间 ID、段头的文件号、页号、页内偏移，获得段信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；
FILE_ID ：文件号；
HEADER_FILE_ID：段头文件号；
HEADER_PAGE_NO：段头页号 ；
HEADER_OFFSET：段头偏移量；
SEG_INFO：段的信息。

8. SEG_ALL_LOAD/ SEG_ALL_GET

定义：

```
PROCEDURE SEG_ALL_LOAD(
TS_ID IN    SMALLINT,
SEG_ID_ARR OUT SEG_ID_ARR_T,
OFFSET IN INT DEFAULT 1,
NUM IN INT DEFAULT 100);
```

定义：

```
FUNCTION SEG_ALL_GET(
TS_ID          IN    SMALLINT,
OFFSET         IN    INT DEFAULT 1,
NUM                  IN    INT DEFAULT 100
)RETURN SEG_ID_ARR_T;
```

功能说明：

根据表空间 ID、数组相对起始位置、数组长度，获得数据库中表空间中所有段的 ID 信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；
OFFSET ：数组起始位置；
NUM：数组长度；
SEG_ID_ARR_T：段 ID 信息。

9. SEG_EXTENT_LST_ LOAD/ SEG_EXTENT_LST_GET

定义：

```
PROCEDURE SEG_EXTENT_LST_LOAD(
TS_ID    IN    SMALLINT,
SEGID IN INT,
EXTENT_ARR OUT EXTENT_ARR_T,
LINKTYPE IN    VARCHAR(5) DEFAULT 'FULL',
DIRECT IN    VARCHAR(10) DEFAULT 'FORWARD',
OFFSET IN    INT DEFAULT 1,
NUM IN    INT DEFAULT 100);
```

定义：

```
FUNCTION SEG_EXTENT_LST_GET(
TSID              IN    SMALLINT,
SEGID             IN    SMALLINT,
LINKTYPE          IN     CHAR(4),
DIRECT            IN     CHAR(1) DEFAULT 'F',
OFFSET           IN     INT DEFAULT 0,
NUM               IN     INT DEFAULT 100,
) RETURN EXTENT_ARR_T;
```

功能说明：

根据输入的表空间 ID、段号、链表类型、链表方向、链表相对起始位置、指定链表长度，获得 FULL 或 FREE 或 FRAG 链表的信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；

SEGID：段 ID；

LINKTYPE：链表类型

DIRECT：链表方向

OFFSET：链表相对起始位置；

NUM：指定链表长度。

EXTENT_ARR_T：簇信息结构体链表

10. EXTENT_SIZE_LOAD/ EXTENT_SIZE_GET

定义：

```
PROCEDURE EXTENT_SIZE_LOAD(
EXTENT_SIZE OUT INT
);
```

定义：

```
FUNCTION    EXTENT_SIZE_GET    RETURN INT;
```

功能说明：

获得簇的大小（页为单位）。过程和函数功能相同。

11. EXTENT_XDESC_PAGE_NO_LOAD/ EXTENT_XDESC_PAGE_NO_GET

定义：

```
PROCEDURE EXTENT_XDESC_PAGE_NO_LOAD(
TS_ID SMALLINT,
FILE_ID SMALLINT,
XPAGE_ADDR_ARR OUT PAGE_ADDR_ARR_T,
OFFSET INT DEFAULT 1,
NUM    INT DEFAULT 100    );
```

定义：

```
FUNCTION EXTENT_XDESC_PAGE_NO_GET(
TSID     IN     SMALLINT,
FILE_ID                          IN     SMALLINT,
OFFSET          IN     INT DEFAULT 0,
NUM                                    IN     INT DEFAULT 100,
)RETURN PAGE_ADDR_ARR_T;
```

功能说明：

根据输入的表空间 ID、文件 ID，获得所有簇描述页的信息。过程和函数功能相同。

参数说明：

TSID： 表空间 ID；

FILE_ID：文件 ID；

OFFSET：数组的起始位置；

NUM：指定输出的长度；

PAGE_ ADDR _ARR_T：页地址记录类型数组。

12. PAGE_N_LOAD/ PAGE_N_GET

定义：

```
PROCEDURE PAGE_N_LOAD(
```

```
PAGE_N OUT INT
);
```

定义：

```
FUNCTION PAGE_N_GET RETURN INT;
```

功能说明：

获得一个描述页能描述的页数。过程和函数功能相同。

参数说明：

PAGE_N：描述页能描述的页数。

13. IPAGE_NO_ALL_LOAD/ IPAGE_NO_ALL_GET

定义：

```
PROCEDURE IPAGE_NO_ALL_LOAD(
TS_ID    IN    SMALLINT,
PAGE_ADDR_ARR OUT PAGE_ADDR_ARR_T ,
LINKTYPE    IN VARCHAR(5) DEFAULT 'FULL',
DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
OFFSET IN INT DEFAULT 1,
NUM IN INT DEFAULT 100 );
```

定义：

```
FUNCTION IPAGE_NO_ALL_GET(
TS_ID    IN    SMALLINT,
LINKTYPE    IN  VARCHAR(5) DEFAULT 'FULL',
DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
OFFSET IN INT DEFAULT 1,
NUM IN INT DEFAULT 100
)RETURN PAGE_ADDR_ARR_T;
```

功能说明：

输入表空间 ID、INODE 页链表类型 FULL 或 FREE 或 FRAG、链表方向、链表相对起始位置、指定链表长度，输出所有 INODE 页的页号。过程和函数功能相同。

参数说明：

TSID：表空间 ID；

PAGE_ADDR_ARR_T：页地址记录类型数组；

LINKTYPE：INODE 页链表类型，FULL、FREE 或 FRAG；

DIRECT：链表的方向；

OFFSET：链表的相对起始位置；

NUM：指定链表长度。

20.11.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取表空间信息（表空间 ID、表空间名、表空间类型：1 DB 类型，2 临时文件组、表空间状态、表空间的最大空间、表空间的总大小（页）、包含文件的个数）。

```
DECLARE
   TS_INFO     DBMS_SPACE.TS_ARR_T;
BEGIN
    TS_INFO = DBMS_SPACE.TS_GET(4);
    IF TS_INFO IS NOT NULL
    THEN
         SELECT * FROM ARRAY TS_INFO;
    ELSE PRINT 'TABLESPACE IS NULL';
    END IF;
END;
```

结果：

```
行号     ID     NAME     CACHE TYPE     STATUS     MAX_SIZE    TOTAL_SIZE            FILE_NUM
---------- ---------- ---- ----- ---------- ---------- ------------------- ------------------- ----------------------------------------
1        4        MAIN          1                0               0               16384                    1
```

例 2 获得数据库中表空间的个数

```
DECLARE
TS_NUM INT;
BEGIN
TS_NUM = DBMS_SPACE.TS_N_GET;
PRINT 'TS_NUM:';
PRINT TS_NUM;
END;
/
```

结果：

```
TS_NUM:
4
```

例 3 获得文件信息（文件路径、文件创建时间 、文件读写状态 1 读，2 写 文件修改的时间、修改的事务 ID 、文件的总大小（M）、文件的空闲大小（M）、数据文件中连续空白页的起始页号、读页个数、写页个数、页大小（K）、读请求个数 、写请求个数 、文件可扩展标记 、文件最大大小（M）、文件每次扩展大小（M）、文件包含的总描述页的数目）

```
DECLARE
   FILE_INFO         DBMS_SPACE.FILE_ARR_T;
BEGIN
     FILE_INFO = DBMS_SPACE.FILE_GET(4,0);
     IF FILE_INFO IS NOT NULL
     THEN
          SELECT * FROM ARRAY FILE_INFO;
     ELSE PRINT 'FILE IS NULL';
     END IF;
END;
/
```

结果：

```
行 号      PATH      CREATE_TIME      MODIFY_TIME      TOTAL_SIZE      FREE_SIZE      FREE_PAGE_NO
PAGE_SIZE    AUTOEXTEND MAX_SIZE      NEXT_SIZE
      ---------      ------------------------------      --------------------------      --------------------------      --------------------
--------------------
1    D:\DATABASE0718\DAMENG\MAIN.DBF    2012-07-19  09:47:55.000000    2012-08-07  09:43:33.000000
16384    16348    368      8192            1               0               0
```

20.12 DBMS_SQL 包

支持在 PL/SQL 中使用动态 SQL 语句。

20.12.1 相关方法

DBMS_SQL 包中所用到的结构定义如下：

```
TYPE INT_INT IS TABLE OF INT INDEX BY INTEGER;
TYPE INT_VARCHAR IS TABLE OF VARCHAR INDEX BY INTEGER;
TYPE INT_DEC IS TABLE OF DEC INDEX BY INTEGER;
TYPE INT_DT IS TABLE OF DATETIME WITH TIME ZONE INDEX BY INTEGER;
TYPE INT_TS IS TABLE OF TIME WITH TIME ZONE INDEX BY INTEGER;
```

DBMS_SQL 包中包含的过程和函数如下详细介绍：

1. IS_OPEN

定义：

```
FUNCTION IS_OPEN(
CURID INT
) RETURN INT;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，判断该 XCSR 是否被打开。

2. OPEN_CURSOR

定义：

```
FUNCTION OPEN_CURSOR RETURN INT;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组的一个没有初始化的项并对其初始化，返回下标。

3. CLOSE_CURSOR

定义：

```
PROCEDURE CLOSE_CURSOR(
CURID INT
);
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，关闭掉。

4. BIND_VARIABLE

定义：

```
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR INT) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR SMALLINT) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR TINYINT) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR DOUBLE) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR CHAR) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR VARCHAR) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR DATE) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR DATETIME) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR DATETIME WITH TIME ZONE) ;
PROCEDURE BIND_VARIABLE(CURID INT, VARNAME VARCHAR, VAR TIME) ;
PROCEDURE BIND_VARIABLE(
CURID INT,
VARNAME VARCHAR,
VAR TIME WITH TIME ZONE
) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，将其中待绑定名为 VARNAME 的位与变量 VAR 绑定。

5. CURSOR_PARSE

定义：

```
PROCEDURE CURSOR_PARSE(
CURID INT,
SQLSTR VARCHAR,
VERSION INT
) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，对 XCSR 的 SQLSTR 进行简单的语法分析，得到语句类型‘S’（SELECT）或者‘L’（其他 DML 语句），并设置 XCSR 的 SQL_TYPE 位，参数 VERSION 为兼容 ORACLE 语法。

6. BIND_ARRAY

定义：

```
PROCEDURE BIND_ARRAY(CURID INT, VARNAME VARCHAR, VAR INT_INT) ;
PROCEDURE BIND_ARRAY(CURID INT, VARNAME VARCHAR, VAR INT_VARCHAR) ;
PROCEDURE BIND_ARRAY(CURID INT, VARNAME VARCHAR, VAR INT_DEC) ;
PROCEDURE BIND_ARRAY(CURID INT, VARNAME VARCHAR, VAR INT_DT) ;
PROCEDURE BIND_ARRAY(CURID INT, VARNAME VARCHAR, VAR INT_TS) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，将其中待绑定名为 VARNAME 的位与索引表 VAR 绑定。

7. FETCH_ROWS/ EXEC_CURSOR/ EXECUTE_AND_FETCH

定义：

```
FUNCTION FETCH_ROWS(
CURID INT
) RETURN INT;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，并执行其中语句，若语句为 SELECT 类型，将会对 XCURSOR 结构中的 XCUR 游标进行 FETCH 操作，并返回该次 FETCH 的行数。FETCH 的行数由 DEFINE_COLUMN 以及 DEFINE_ARR 决定，FETCH 的结果将会被放到 XCURSOR 的 FETCH_RES_VARCHAR 索引表中；如果语句不是 SELECT 类型，则报错。

```
FUNCTION EXEC_CURSOR(
CURID INT
) RETURN INT;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，并执行其中语句。若语句为 SELECT 类型，将会打开 XCURSOR 结构中的 XCUR 游标，返回值无意义。

定义：

```
FUNCTION DBMS_SQL.EXECUTE_AND_FETCH(
CURID INT
) ;
```

功能说明：

功能相当于先调用 EXECUTE，然后再执行 FETCH 函数。返回 FETCH 的行数。

8. DEFINE_ARRAY

定义：

```
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_INT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_VARCHAR) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_DT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_TS) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_DEC) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，指定 FETCH_ROWS 每次 FETCH 的位置 SEQNO，绑定到 COL 索引表类型的变量。

参数说明：

CURID：DBMS_SQL 包中 XCURSOR 数组的下标；

SEQNO：被定义的列在 FETCH 数据的相对位置，FETCH 数据的第一列位置为 1；

COL：被定义的列的类型。

定义：

```
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_INT,LEAST_FETCH
INT,START_OP INT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_VARCHAR,LEAST_FETCH
INT,START_OP INT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_DT,LEAST_FETCH
INT,START_OP INT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_TS,LEAST_FETCH
INT,START_OP INT) ;
PROCEDURE DEFINE_ARRAY(CURID INT,SEQNO INT,COL INT_DEC,LEAST_FETCH
INT,START_OP INT) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，指定 FETCH_ROWS

每次 FETCH 的位置 SEQNO，绑定到 COL 类型的变量。LEAST_FETCH 为 FETCH 的行数，START_OPINT 为绑定数据导 COL 索引表的起始位置。

9. DEFINE_COLUMN

定义：

```
PROCEDURE DEFINE_COLUMN(CURID INT,SEQNO INT,COL INT_INT) ;
PROCEDURE DEFINE_COLUMN(CURID INT,SEQNO INT,COL INT_VARCHAR) ;
PROCEDURE DEFINE_COLUMN(CURID INT,SEQNO INT,COL INT_DT) ;
PROCEDURE DEFINE_COLUMN(CURID INT,SEQNO INT,COL INT_TS) ;
PROCEDURE DEFINE_COLUMN(CURID INT,SEQNO INT,COL INT_DEC) ;
```

功能说明：

定义从游标中获取的一列数据。

参数说明：

CURID： DBMS_SQL 包中 XCURSOR 数组的下标；

SEQNO：被定义的列在 FETCH 数据的相对位置，FETCH 数据的第一列位置为 1；

COL：被定义的列的值，该值的类型即为要定义列的类型。

10. COLUMN_VALUE

定义：

```
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT_INT) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT_VARCHAR) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT_DEC) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT_DT) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT_TS) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，将 FETCH 得到的结果的第 SEQNO 列数据送入到索引表 VAR 中。

定义：

```
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT INT) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT BIGINT) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT BINARY) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT VARBINARY) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT DOUBLE) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT REAL) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT VARCHAR) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT DEC) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT DATETIME) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT DATETIME WITH TIME ZONE) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT CLOB) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT TIME) ;
PROCEDURE COLUMN_VALUE(CURID INT,SEQNO INT,VAR OUT TIME WITH TIME ZONE) ;
```

功能说明：

找到 DBMS_SQL 包中 XCURSOR 数组中下标为 CURID 的游标，将 FETCH 得到的结果的第 SEQNO 列数据送入到变量 VAR 中。

20.12.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 通过过程 COPY 调用 DBMS_SQL 中的过程和函数，来实现两表之间的复制，要求表的列类型为第一列 INT，第二列 VARCHAR。

```
--创建 COPY 过程
```

```
CREATE OR REPLACE PROCEDURE COPY(SOURCE IN VARCHAR,DES IN VARCHAR) AS
SCID INT;
DCID INT;
VAR_NUMBER INT;
VAR_NAME      VARCHAR;
BEGIN
SCID = DBMS_SQL.OPEN_CURSOR();
DBMS_SQL.CURSOR_PARSE(SCID, 'SELECT * FROM '||SOURCE, 1);
DBMS_SQL.EXEC_CURSOR(SCID);
DCID = DBMS_SQL.OPEN_CURSOR();
DBMS_SQL.CURSOR_PARSE(DCID,'INSERT INTO '||DES||' VALUES(:A,:B)',1);

WHILE DBMS_SQL.FETCH_ROWS(SCID) > 0 LOOP
DBMS_SQL.COLUMN_VALUE(SCID,1,VAR_NUMBER);
DBMS_SQL.COLUMN_VALUE(SCID,2,VAR_NAME);
PRINT('VAR_NUMBER');
PRINT(VAR_NUMBER);
PRINT('VAR_NAME');
PRINT(VAR_NAME);

DBMS_SQL.BIND_VARIABLE(DCID,'A',VAR_NUMBER);
PRINT(DBMS_SQL.XCURSOR[DCID].NAME[1]);
PRINT(DBMS_SQL.XCURSOR[DCID].NAME[2]);
DBMS_SQL.BIND_VARIABLE(DCID,'B',VAR_NAME);


DBMS_SQL.EXEC_CURSOR(DCID);
END LOOP;
END;
/
--创建表 A，表 B
DROP TABLE A;
DROP TABLE B;

CREATE TABLE A(ID INT,ID1 VARCHAR);
INSERT INTO A VALUES(1,'AAA');
INSERT INTO A VALUES(2,'BAA');
INSERT INTO A VALUES(3,'CAA');
INSERT INTO A VALUES(4,'DAA');
INSERT INTO A VALUES(5,'EAA');
INSERT INTO A VALUES(6,'FAA');
INSERT INTO A VALUES(7,'GAA');

CREATE TABLE B(ID INT,ID1 VARCHAR);

--调用过程 COPY，将表 A 数据复制到表 B 中
CALL COPY('SYSDBA.A','SYSDBA.B');
--查询表 B，看是否与表 A 相同
SELECT * FROM B;
```

结果如下:

```
ID    ID1
1    AAA
2    BAA
3    CAA
4    DAA
5    EAA
6    FAA
7    GAA
```

20.13 DBMS_TRANSACTION 包

DBMS_TRANSACTION 包提供获得当前活动事务号的功能。

20.13.1 相关方法

定义：

```
FUNCTION LOCAL_TRANSACTION_ID (
      CREATE_TRANSACTION BOOLEAN := FALSE
) RETURN VARCHAR2;
```

功能说明：

返回当前活动事务号。

参数说明：

CREATE_TRANSACTION：表示如果当前没有活动事务，是否创建一个新的事务。TRUE 创建；FALSE 不创建，返回空。默认为 FALSE。

20.13.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 返回当前事务号，若当前没有活动事务，则返回一个新的事务的事务号

```
SELECT DBMS_TRANSACTION.LOCAL_TRANSACTION_ID(TRUE);
```

结果如下：

```
1871
```

20.14 DBMS_RANDOM 包

为提高 ORACLE 向 DM 移植的兼容性。实现随机产生 INT 类型、NUMBER 类型数，随机字符串，以及符合正态分布的随机数。支持 ORACLE 的 DBMS_RANDOM 系统包。

20.14.1 相关方法

1. INITIALIZE

定义：

```
PROCEDURE INITIALIZE(
VAL IN INT
);
```

功能说明：

初始化随机种子。接收 INT 类型参数，将 RAND()种子设为输入的参数。

2. SEED

定义：

```
PROCEDURE SEED(
VAL IN INT
);
```

功能说明：

重置随机种子。接收 INT 类型参数，将 RAND()种子设为输入的参数。

定义：

```
PROCEDURE SEED(
VAL IN VARCHAR2
);
```

功能说明：

接收 VARCHAR 类型参数，通过转换成 INT 类型再将 RAND()种子设为输入的参数转换好的值。

3. TERMINATE

定义：

```
PROCEDURE TERMINATE;
```

功能说明：

ORACLE 中不开放的功能，在此只做语法支持，无意义。

4. RANDOM_NORMAL/ RANDOM/ RANDOM_STRING

定义：

```
FUNCTION RANDOM_NORMAL      RETURN NUMBER;
```

功能说明：

产生符合正态分布的随机数。

定义：

```
FUNCTION    RANDOM    RETURN INTEGER;
```

功能说明：

产生 INT 类型随机数。

定义：

```
FUNCTION RANDOM_STRING (
OPT IN CHAR,
LEN IN NUMBER
)RETURN VARCHAR2;
```

功能说明：

生成随机字符串。

参数说明：

OPT：CHAR 类型，表示字符串模式，模式解释如下：

'U', 'U'：只产生随机的大写字母字符串；

'L', 'L'：只产生随机的小写字母字符串；

'A', 'A'：产生大小写混合的字母字符串；

'X', 'X'：返回大写字母和数字随机字符串；

'P', 'P'：返回随机可打印字符字符串。

LEN：表示生成字符串的长度，最大为 8188。

定义：

```
FUNCTION VALUE (
LOW IN NUMBER DEFAULT 0,
HIGH IN NUMBER DEFAULT 1
)RETURN NUMBER;
```

功能说明：

生成大于 LOW，小于 HIGH 的 NUMBER 行随机数。

参数说明：

LOW、HIGH都为NUMBER类型，LOW小于HIGH时，产生大于等于LOW小于等于HIGH的随机数；当LOW大于HIGH时，大于HIGH小于等于LOW的随机数。

20.14.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 初始化随机种子。

```
CALL DBMS_RANDOM.INITIALIZE(15);
```

例 2 返回大于等于 10 小于 100 的 NUMBER 类型随机数。

```
SELECT    DBMS_RANDOM.VALUE(10,100);
```

结果如下：

```
10.112613
```

20.15 DBMS_STATS 包

优化统计信息描述了数据库中的对象细节。查询优化使用这些信息选择最合适的执行计划。使用 DBMS_STATS 包来收集统计、删除信息，将收集的统计信息记录在数据字典中。

20.15.1 数据类型

DBMS_STATS 包中涉及到类型。如下统一说明。

OBJECTELEM 类型是 DBMS_STATS 专有类型。用户不能引用和改变该记录的内容。OBJECTELEM 记录类型定义如下：

```
TYPE OBJECTELEM IS RECORD (
        OWNNAME         VARCHAR(128) ,
        OBJTYPE         VARCHAR(6) ,
        OBJNAME         VARCHAR(128) ,
        PARTNAME        VARCHAR(128) ,
        SUBPARTNAME VARCHAR(128)
 ) ;
```

参数说明：

OWNNAME：模式名；

OBJTYPE：对象类型，TABLE 或 INDEX；

OBJNAME：对象（表或索引）名称，区分大小写；

PARTNAME：分区名称，区分大小写；

SUBPARTNAME：子分区名称，区分大小写；

OBJECTTAB 为 OBJECTELEM 类型的索引表：

```
TYPE OBJECTTAB IS TABLE OF OBJECTELEM INDEX BY INT;
```

20.15.2 相关方法

DBMS_STATS 包中包含的过程和函数如下详细介绍：

1. COLUMN_STATS_SHOW

定义：

```
PROCEDURE COLUMN_STATS_SHOW (
      OWNNAME       IN    VARCHAR(128),
      TABNAME       IN    VARCHAR(128),
      COLNAME       IN    VARCHAR(128)
);
```

功能说明：

根据模式名，表名和列名获得该列的统计信息。返回两个结果集：一个是列的统计信息；另一个是直方图的统计信息。

表 列统计信息表 列统计信息

名称	解释
NUM_DISTINCT	不同列值的个数
LOW_VALUE	列最小值
HIGH_VALUE	列最大值

NUM_NULLS	空值的个数
NUM_BUCKETS	直方图桶的个数
SAMPLE_SIZE	样本容量
HISTOGRAM	直方图的类型

表 直方图的统计信息

名称	解释
OWNER	模式名
TABLE_NAME	表名
COLUMN_NAME	列名
HISTOGRAM	直方图类型
ENDPOINT_VALUE	样本值
ENDPOINT_HEIGHT	对于频率直方图，样本值的个数；对于等高直方图，小于样本值大于前一个样本值的个数。
ENDPOINT_KEYGHT	对于频率直方图无效；对于等高直方图，样本值的个数。
ENDPOINT_DISTINCT	对于频率直方图无效；对于等高直方图，小于样本值大于前一个样本值之间不同样本的个数。

参数说明：

OWNNAME：模式名，区分大小写；

TABNAME：表名，区分大小写；

COLNAME：列名，区分大小写。

2. TABLE_STATS_SHOW

定义：

```
PROCEDURE TABLE_STATS_SHOW (
    OWNNAME      IN   VARCHAR(128),
    TABNAME      IN   VARCHAR(128)
);
```

功能说明：

根据模式名，表名获得该表的统计信息。

表 列统计信息

名称	解释
NUM_ROWS	表的总行数
LEAF_BLOCKS	总的页数
LEAF_USED_BLOCKS	已经使用的页数

参数说明：

OWNNAME：模式名，区分大小写；

TABNAME：表名，区分大小写。

3. INDEX_STATS_SHOW

定义：

```
PROCEDURE INDEX_STATS_SHOW (
    OWNNAME      IN   VARCHAR(128),
    INDEXNAME    IN    VARCHAR(128)
);
```

功能说明：

根据模式名，索引名获得该索引的统计信息。返回两个结果集：一个是列的统计信息；

另一个是直方图的统计信息。

表 列统计信息

名称	解释
BLEVEL	B_Tree 的层次
LEAF_BLOCKS	页数
DISTINCT_KEYS	不同样本的个数
CLUSTERING_FACTOR	聚集因子，表示索引的数据分布与物理分布之间的关系
NUM_ROWS	行数
SAMPLE_SIZE	样本容量

表 直方图的统计信息

名称	解释
OWNER	模式名
NAME	索引名
COLUMN_NAME	列名
HISTOGRAM	直方图类型
ENDPOINT_VALUE	样本值
ENDPOINT_HEIGHT	对于频率直方图，样本值的个数；对于等高直方图，小于样本值大于前一个样本值的个数。
ENDPOINT_KEYGHT	对于频率直方图无效；对于等高直方图，样本值的个数。
ENDPOINT_DISTINCT	对于频率直方图无效；对于等高直方图，小于样本值大于前一个样本值之间不同样本的个数。

参数说明：

OWNNAME：模式名，区分大小写；

INDEXNAME：索引名，区分大小写。

4. GATHER_TABLE_STATS

定义：

```
PROCEDURE GATHER_TABLE_STATS (
    OWNNAME             VARCHAR(128),
    TABNAME             VARCHAR(128),
    PARTNAME            VARCHAR(128)    DEFAULT NULL,
    ESTIMATE_PERCENT                DOUBLE                                          DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    BLOCK_SAMPLE        BOOLEAN     DEFAULT FALSE,
    METHOD_OPT          VARCHAR     DEFAULT GET_PREFS('METHOD_OPT'),
    DEGREE              INT         DEFAULT TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY         VARCHAR     DEFAULT GET_PREFS('GRANULARITY'),
    CASCADE             BOOLEAN     DEFAULT TO_CASCADE_TYPE(GET_PREFS('CASCADE')),
    STATTAB             VARCHAR     DEFAULT NULL,
    STATID              VARCHAR     DEFAULT NULL,
    STATOWN             VARCHAR     DEFAULT NULL,
    NO_INVALIDATE               BOOLEAN         DEFAULT     TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN     DEFAULT FALSE
);
```

功能说明：

根据设定的参数，收集表的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

TABNAME：表名，区分大小写；

PARTNAME：分区表名，默认为 NULL，区分大小写；

ESTIMATE_PERCENT：收集的百分比，范围为 0.000001~100，默认系统自定；

BLOCK_SAMPLE：保留参数，是否使用随机块代替随机行，默认为 TRUE；

METHOD_OPT：控制列的统计信息集合和直方图的创建的格式，默认为 FOR ALL COULMNS SIZE AUTO。其中 BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、BOOLEAN 类型不能被收集。格式选项如下：

1) FOR ALL [INDEXED | HIDDEN] COLUMNS [<size_clause>]
   <size_clause> := SIZE {INTEGER | REPEAT | AUTO | SKEWONLY}
2) FOR COLUMNS [<size clause>] <column_name>| [<size_clause>] >{,<column_name | [<size_clause>]>}
   各参数解释如下：
   - ●INDEXED | HIDDEN　表示只统计索引或者隐藏的列，不加表示都统计
   - ●INTEGER　直方图的桶数，范围 1~254
   - ●REPEAT　只统计已经有直方图的列
   - ●AUTO　根据数据分布和工作量自动决定统计直方图的列
   - ●SKEWONLY　根据数据分布决定统计直方图的列

DEGREE： 保留参数，收集的并行度，默认为 1；

GRANULARITY： 保留参数，收集的粒度，默认为 ALL；GRANULARITY 可选参数如下：AUTO | DEFAULT | ALL | PARTITION | SUBPARTITION | GLOBAL | GLOBAL AND PARTITION。各参数解释如下：

- ●ALL　收集全部的统计信息（SUBPARTITION，PARTITION 和 GLOBAL）
- ●AUTO　默认值，根据分区的类型来决定如何收集
- ●DEFAULT　和 AUTO 功能相同
- ●GLOBAL　收集 GLOBAL 表的统计信息
- ●GLOBAL AND PARTITION　收集 GLOBAL 和 PARTITION 表的统计信息
- ●PARTITION　收集 PARTITION 表的统计信息
- ●SUBPARTITION　收集 SUBPARTITION 表的统计信息

CASCADE：是否收集索引信息，TRUE 或 FALSE。默认为 TRUE；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE。

5. GATHER_INDEX_STATS

定义：

```
PROCEDURE GATHER_INDEX_STATS (
    OWNNAME             VARCHAR(128),
    INDNAME             VARCHAR(128),
    PARTNAME            VARCHAR(128) DEFAULT NULL,
    ESTIMATE_PERCENT            DOUBLE                                  DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    STATTAB             VARCHAR DEFAULT NULL,
    STATID              VARCHAR DEFAULT NULL,
    STATOWN             VARCHAR DEFAULT NULL,
```

```
    DEGREE              INT                 DEFAULT
TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY         VARCHAR DEFAULT GET_PREFS('GRANULARITY'),
    NO_INVALIDATE                                   BOOLEAN                 DEFAULT
TO_NO_INVALIDATE_TYPE(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN DEFAULT FALSE
);
```

功能说明：

根据设定的参数，收集索引的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

INDNAME：索引名，区分大小写；

PARTNAME：分区表名，默认为 NULL，区分大小写；

ESTIMATE_PERCENT：收集的百分比，范围为 0.000001~100，默认系统自定；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

DEGREE：保留参数，收集的并行度，默认为 1；

GRANULARITY：保留参数，收集的粒度，默认为 ALL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE。

6. GATHER_SCHEMA_STATS

定义：

```
PROCEDURE GATHER_SCHEMA_STATS (
    OWNNAME             VARCHAR(128),
    ESTIMATE_PERCENT                DOUBLE                                  DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    BLOCK_SAMPLE        BOOLEAN    DEFAULT FALSE,
    METHOD_OPT          VARCHAR DEFAULT GET_PREFS('METHOD_OPT'),
    DEGREE              INT     DEFAULT TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY         VARCHAR DEFAULT GET_PREFS('GRANULARITY'),
    CASCADE             BOOLEAN    DEFAULT TO_CASCADE_TYPE(GET_PREFS('CASCADE')),
    STATTAB             VARCHAR DEFAULT NULL,
    STATID              VARCHAR DEFAULT NULL,
    OPTIONS             VARCHAR DEFAULT 'GATHER',
    OBJLIST      OUT    OBJECTTAB DEFAULT NULL,
    STATOWN             VARCHAR DEFAULT NULL,
    NO_INVALIDATE               BOOLEAN         DEFAULT     TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN DEFAULT FALSE,
    OBJ_FILTER_LIST    OBJECTTAB DEFAULT NULL
);
```

功能说明：

收集模式下对象的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

ESTIMATE_PERCENT：收集的百分比，范围为 0.000001~100，默认系统自定；

BLOCK_SAMPLE：保留参数，是否使用随机块代替随机行，默认为 TRUE；

METHOD_OPT：控制列的统计信息集合和直方图的创建；默认为 FOR ALL COULMNS SIZE AUTO；只支持其中一种格式：

FOR ALL [INDEXED | HIDDEN] COLUMNS [<size_clause>]

<size_clause> := SIZE {INTEGER | REPEAT | AUTO | SKEWONLY}

DEGREE：保留参数，收集的并行度，默认为 1；

GRANULARITY：保留参数，收集的粒度，默认为 ALL；

CASCADE：是否收集索引信息，TRUE 或 FALSE。默认为 TRUE；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

OPTIONS：控制收集的列，默认为 NULL；选项如下：GATHER| GATHER AUTO| GATHER STALE| GATHER EMPTY| LIST AUTO| LIST STALE| LIST EMPTY。各选项解释如下：

- ●GATHER：收集模式下所有对象的统计信息。
- ●GATHER AUTO：自动收集需要的统计信息。系统隐含的决定哪些对象需要新的统计信息，以及怎样收集这些统计信息。此时，只有 OWNNAME，STATTAB，STATID，OBJLIST AND STATOWN有效，返回收集统计信息的对象。
- ●GATHER STALE：对旧的对象收集统计信息。 返回找到的旧的对象。
- ●GATHER EMPTY：收集没有统计信息对象的统计信息。返回这些对象。
- ●LIST AUTO：返回GATHER AUTO方式处理的对象。
- ●LIST STALE：返回旧的对象信息。
- ●LIST EMPTY：返回没有统计信息的对象。

OBJLIST：返回 OPTION 选项对应的链表，默认为 NULL；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE；

OBJ_FILTER_LIST：存放过滤条件的模式名、表名和子表名，默认为 NULL。

7. DELETE_TABLE_STATS

定义：

```
PROCEDURE DELETE_TABLE_STATS (
    OWNNAME              VARCHAR(128),
    TABNAME              VARCHAR(128),
    PARTNAME             VARCHAR(128) DEFAULT NULL,
    STATTAB              VARCHAR DEFAULT NULL,
    STATID               VARCHAR DEFAULT NULL,
    CASCADE_PARTS        BOOLEAN    DEFAULT TRUE,
    CASCADE_COLUMNS      BOOLEAN    DEFAULT TRUE,
    CASCADE_INDEXES      BOOLEAN    DEFAULT TRUE,
    STATOWN              VARCHAR DEFAULT NULL,
    NO_INVALIDATE        BOOLEAN    DEFAULT TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE                BOOLEAN DEFAULT FALSE
);
```

功能说明：

根据设定参数，删除与表相关对象的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

TABNAME：表名，区分大小写；

PARTNAME：分区表名，默认为 NULL，区分大小写；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

CASCADE_PARTS：是否级联删除分区表信息，默认为 TRUE；

CASCADE_COLUMNS：是否级联删除表中列的信息，TRUE 或 FALSE。默认为 TRUE；

CASCADE_INDEXES：是否级联删除表的索引信息，TRUE 或 FALSE。默认为 TRUE；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE。

8. DELETE_SCHEMA_STATS

定义：

```
PROCEDURE DELETE_SCHEMA_STATS (
    OWNNAME             VARCHAR(128),
    STATTAB             VARCHAR DEFAULT NULL,
    STATID              VARCHAR DEFAULT NULL,
    STATOWN             VARCHAR DEFAULT NULL,
    NO_INVALIDATE       BOOLEAN DEFAULT TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN DEFAULT FALSE
);
```

功能说明：

根据设定参数，删除模式下对象的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE。

9. DELETE_INDEX_STATS

定义：

```
PROCEDURE DELETE_INDEX_STATS (
    OWNNAME             VARCHAR(128),
    INDNAME             VARCHAR(128),
    PARTNAME            VARCHAR(128) DEFAULT NULL,
    STATTAB             VARCHAR DEFAULT NULL,
    STATID              VARCHAR DEFAULT NULL,
    CASCADE_PARTS       BOOLEAN  DEFAULT TRUE,
    STATOWN             VARCHAR DEFAULT NULL,
    NO_INVALIDATE       BOOLEAN  DEFAULT TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN  DEFAULT FALSE
);
```

功能说明：

根据设定参数，删除索引的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

INDNAME：索引名，区分大小写；

PARTNAME：分区表名，默认为 NULL，区分大小写；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

CASCADE_PARTS：是否级联删除分区表信息，TRUE 或 FALSE。默认为 TRUE；

STATOWN：保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE。

10. DELETE_COLUMN_STATS

定义：

```
PROCEDURE DELETE_COLUMN_STATS (
    OWNNAME             VARCHAR(128),
    TABNAME             VARCHAR(128),
    COLNAME             VARCHAR(128),
    PARTNAME            VARCHAR(128) DEFAULT NULL,
```

```
    STATTAB             VARCHAR DEFAULT NULL,
    STATID              VARCHAR DEFAULT NULL,
    CASCADE_PARTS   BOOLEAN   DEFAULT TRUE,
    STATOWN             VARCHAR DEFAULT NULL,
    NO_INVALIDATE                                   BOOLEAN                                 DEFAULT
TO_NO_INVALIDATE_TYPE(GET_PREFS('NO_INVALIDATE')),
    FORCE               BOOLEAN   DEFAULT FALSE,
    COL_STAT_TYPE   VARCHAR DEFAULT 'ALL'
);
```

功能说明：

根据设定参数，删除列的统计信息。

参数说明：

OWNNAME：模式名，区分大小写；

TABNAME：表名，区分大小写；

COLNAME：列名，区分大小写；

PARTNAME：分区表名，默认为 NULL，区分大小写；

STATTAB：保留参数，统计信息存放的表，默认为 NULL；

STATID：保留参数，统计信息的 ID，默认为 NULL；

CASCADE_PARTS：是否级联删除分区表信息，TRUE 或 FALSE。默认为 TRUE；

STATOWN： 保留参数，统计信息的模式，默认为 NULL；

NO_INVALIDATE：保留参数，是否让依赖游标失效，默认为 TRUE；

FORCE：保留参数，是否强制收集统计信息，默认为 FALSE；

COL_STAT_TYPE：保留参数，是否只删除直方图信息，默认为 ALL。

11. UPDATE_ALL_STATS

定义：

```
PROCEDURE UPDATE_ALL_STATS ();
```

功能说明：

更新已有的统计信息。

20.15.3 约束

以下对象不支持统计信息：

1. 外部表、临时表、REMOTE 表、动态视图表、记录类型数组所用的临时表；
2. 所在表空间为 OFFLINE 的对象；
3. 位图索引，位图连接索引、虚索引、无效的索引、全文索引；
4. BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、BOOLEAN 等列类型。

20.15.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 收集模式 PERSON 下表 ADDRESS 的统计信息，并打印收集的表信息。

```
BEGIN
    DECLARE
      OBJTAB DBMS_STATS.OBJECTTAB;
      OBJ_FILTER_LIST DBMS_STATS.OBJECTTAB;
```

```
BEGIN
        OBJ_FILTER_LIST(0).OWNNAME = 'PERSON';
        OBJ_FILTER_LIST(0).OBJTYPE = 'TABLE';
        OBJ_FILTER_LIST(0).OBJNAME = 'ADDRESS';
    DBMS_STATS.GATHER_SCHEMA_STATS(
        'PERSON',
         1.0,
         FALSE,
         'FOR ALL COLUMNS SIZE AUTO',
         1,
         'AUTO',
         TRUE,
         NULL,
         NULL,
         'GATHER',
         OBJTAB,
         NULL,
         TRUE,
         TRUE,
         OBJ_FILTER_LIST
    );
    PRINT OBJTAB.COUNT;
        FOR I IN 0..OBJTAB.COUNT-1 LOOP
            PRINT OBJTAB(I).OWNNAME;
            PRINT OBJTAB(I).OBJTYPE;
            PRINT OBJTAB(I).OBJNAME;
            PRINT OBJTAB(I).PARTNAME;
            PRINT OBJTAB(I).SUBPARTNAME;
            PRINT '-------- ';
         END LOOP;
END;
END;
```

打印结果

```
1
PERSON
TABLE
ADDRESS
NULL
NULL
```

例 2 获得 PERSON 模式下表 ADDRESS 中列 ADDRESSID 的统计信息。

```
DBMS_STATS.COLUMN_STATS_SHOW('PERSON','ADDRESS','ADDRESSID');
```

返回结果集 1：

```
NUM_DISTINCT LOW_VALUE HIGH_VALUE NUM_NULLS NUM_BUCKETS SAMPLE_SIZE HISTOGRAM
16 1 16 0 16 16 FREQUENCY
```

结果集 2

```
OWNER TABLE_NAME COLUMN_NAME HISTOGRAM ENDPOINT_VALUE ENDPOINT_HEIGHT ENDPOINT_KEYHEIGHT ENDPOINT_DISTINCT
PERSON ADDRESS ADDRESSID FREQUENCY 1 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 2 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 3 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 4 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 5 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 6 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 7 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 8 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 9 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 10 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 11 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 12 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 13 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 14 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 15 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 16 1 NULL NULL
```

例 3 删除 PERSON 模式下表 ADDRESS 的统计信息。

```
BEGIN
   DBMS_STATS.DELETE_TABLE_STATS ('PERSON', 'ADDRESS');
END;
```

20.16 UTL_FILE 包

UTL_FILE 包为 PL/SQL 程序提供读和写操作系统数据文件的功能。它提供一套严格的使用标准操作系统文件 I/O 方式：OPEN、 PUT、 GET 和 CLOSE 操作。

当用户想读取或写一个数据文件的时候，可以使用 FOPEN 来返回的文件句柄。这个文件句柄将用于随后在文件上的所有操作。例如，过程 PUT_LINE 写 TEXT 字符串和行终止符到一个已打开的文件句柄，以及使用 GET_LINE 来读取指定文件句柄的一行到提供的缓存。

20.16.1 数据类型

UTL_FILE 定义了一种 FILE_TYPE 记录类型。FILE_TYPE 类型是 UTL_FILE 专有类型。用户不能引用和改变该记录的内容。FILE_TYPE 记录类型定义如下：

定义：

```
TYPE FILE_TYPE IS RECORD (
    ID              INTEGER,
    DATATYPE        INTEGER,
    BYTE_MODE       BOOLEAN);
```

参数说明：

ID：需要处理的外部文件句柄；

DATATYPE：文件的类型：1 表示 CHAR 、2 表示 NCHAR、3 表示 BINARY；

BYTE_MODE：文件打开后的类型：TRUE 表示二进制模式；FALSE 表示文本模式。

20.16.2 相关方法

UTL_FILE 包中的 18 个过程和函数详细介绍如下：

如下详细介绍各过程和函数：

1. FCLOSE

定义：

```
PROCEDURE FCLOSE(
      FILE        IN OUT FILE_TYPE
);
```

功能说明：

关闭一个已打开文件。

参数说明：

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

2. FCLOSE_ALL

定义：

```
PROCEDURE FCLOSE_ALL;
```

功能说明：

关闭在这个会话里打开的所有文件。用作紧急情况下清理程序，比如，当 PL/SQL 存在异常时。FCLOSE_ALL 不会改变用户打开的文件句柄的状态。即，在调用 FCLOSE_ALL 后用 IS_OPEN 去测试一个文件句柄，返回值仍为 TRUE，即使该文件已经关闭了。在 FCLOSE_ALL 之前打开的所有文件都无法进一步读或写。

3. FCOPY

定义：

```
PROCEDURE FCOPY(
    SRC_LOCATION    IN VARCHAR(128),
    SRC_FILENAME    IN VARCHAR(128),
    DEST_LOCATION IN VARCHAR(128),
    DEST_FILENAME IN VARCHAR(128),
    START_LINE      IN INTEGER DEFAULT 1,
    END_LINE        IN INTEGER DEFAULT NULL
);
```

功能说明：

把一个文本文件中指定的连续的行数据拷贝到另外一个文件中。源文件需要以可读模式打开。

参数说明：

SRC_LOCATION：源文件目录。

SRC_FILENAME：被拷贝的源文件名称。

DEST_LOCATION：新创建的目标文件的目录。

DEST_FILENAME：源文件的新名称。

START_LINE：拷贝的起始行，默认为文件的第 1 行。

END_LINE：拷贝的终止行，默认为 NULL，即为文件的最后一行。

4. FFLUSH

定义：

```
PROCEDURE FFLUSH(
     FILE    IN FILE_TYPE
) ;
```

功能说明：

以物理写入的方式将待定的数据写入到文件句柄所指定的文件中。通常地，待写入文件的数据是存放在缓冲区中的。FFLUSH 程序强制将缓冲区中的数据写入到文件中。数据必须以换行符结束。

参数说明：

FILE：由 FOPEN 或 FOPEN_NCHAR 调用，返回的文件句柄

5. FGETATTR

定义：

```
PROCEDURE FGETATTR(
    LOCATION      IN    VARCHAR(128),
    FILENAME      IN    VARCHAR(128),
    FEXISTS       OUT BOOLEAN,
    FILE_LENGTH OUT NUMBER,
    BLOCKSIZE     OUT NUMBER) ;
```

功能说明：

读取并返回磁盘文件的属性。

参数说明：

LOCATION：源文件路径；

FILENAME：被检查的文件；

FEXISTS：文件是否存在，存在为 TRUE，不存在为 FALSE；

FILE_LENGTH：文件的长度（字节），若文件不存在，则长度为 NULL；

BLOCKSIZE：文件系统中数据块的大小（字节），若文件不存在，则长度为 NULL。

6. FGETPOS

定义：

```
FUNCTION    FGETPOS(
    FILE        IN FILE_TYPE
)RETURN INTEGER;
```

功能说明：

指定文件中当前相对位置偏移量（字节）。

参数说明：

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

返回值：FGETPOS 以字节的方式返回一个打开文件相对位置偏移量。如果文件未打开则发生异常。如果当前位置在文件的最开始，则返回 0。

注意事项：

如果文件以 BYTE 的模式打开，那么会发生 INVALID OPERATION。

7. FOPEN

定义：

```
FUNCTION FOPEN(
    LOCATION        IN VARCHAR(128),
    FILENAME        IN VARCHAR(128),
    OPEN_MODE       IN VARCHAR(128),
    MAX_LINESIZE IN INTEGER DEFAULT 1024
)RETURN FILE_TYPE;
```

功能说明：

打开指定文件并返回一个文件句柄。用户指定文件每行最大的字符数，并且同时可以打开文件最多 50 个。

参数说明：

LOCATION：源文件路径。

FILENAME：文件名称，包括文件类型，但不包含文件路径。如果文件名称中包含路径，则 FOPEN 忽略此处的路径。在 UNIX 系统中，文件名不能包含转义符：“/”。

OPEN_MODE：文件打开模式。包括：R 只读模式；W 写模式；A 附加模式；RB 只读打开一个二进制文件，只允许读；WB 只写打开或建立一个二进制文件，只允许写数据；AB 追加打开一个二进制文件，并在文件末尾写数据。当以“A”或“AB”的方式打开文件时，若该文件不存在，则以“写”的方式创建该文件。

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

MAX_LINESIZE：文件每行最大的字符数，包括换行符。最小为 1，最大为 32767。

注意事项：

文件的路径和文件名必须加上引号，以便于和相近的路径名区分开来。

异常：

INVALID_PATH：文件路径无效；

INVALID_MODE：参数 OPEN_MODE 字符串无效；

INVALID_OPERATION：文件无法按照请求打开；

INVALID_MAXLINESIZE：指定的最大行字符数值太大或太小。

8. FREMOVE

定义：

```
PROCEDURE FREMOVE(
    LOCATION IN VARCHAR(128),
    FILENAME IN VARCHAR(128));
```

功能说明：

在有足够权限的情况下，从系统中删除一个文件。

参数说明：

LOCATION：文件路径；

FILENAME：被删除文件名称。

注意事项：

FREMOVE 存储过程不会在删除文件之前验证权限，但是操作系统会验证文件和路径的正确性。没有权限，或者文件和路径不正确，则返回删除失败信息。

9. FRENAME

定义：

```
PROCEDURE FRENAME(
    SRC_LOCATION    IN VARCHAR(128),
    SRC_FILENAME    IN VARCHAR(128),
    DEST_LOCATION IN VARCHAR(128),
    DEST_FILENAME IN VARCHAR(128),
    OVERWRITE       IN BOOLEAN DEFAULT FALSE
);
```

功能说明：

修改一个文件的名称。

参数说明：

SRC_LOCATION：要改名文件的存放目录；

SRC_FILENAME：要改名的源文件名称；

DEST_LOCATION：改名后文件存放的路径；

DEST_FILENAME：修改后的新名称；

OVERWRITE：设置是否能够重写。如果设置为“TRUE”，则在 SRC_LOCATION 目录中覆盖任何名为 DEST_FILENAME 的文件。若设置为“FALSE”，就会产生异常。缺省为 FLASE。

10. FSEEK

定义：

```
PROCEDURE FSEEK(
    FILE    IN FILE_TYPE,
    ABSOLUTE_OFFSET IN INTEGER DEFAULT NULL,
    RELATIVE_OFFSET IN INTEGER DEFAULT NULL
);
```

功能说明：

根据指定的字节数，调整文件指针前进或后退。

参数说明：

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

ABSOLUTE_OFFSET：要寻找的字节的绝对路径，默认为 NULL。

RELATIVE_OFFSET：指针前进或后退的字节数。用正整数表示向前查找，负整数表示向后查找。

注意事项：

使用 FSEEK，可以读取先前的行而不用关闭文件再重新找开。但是你必须知道要跳过的字符数。

该存储过程根据 RELATIVE_OFFSET 参数指定的字节数进行查找。

如果在指定字节数之前就已经到达了文件的开头，那么文件指针指定在文件开头。如果在指定字节数之前就已经到达了文件的尾部，那么 INVALID_OFFSET 错误就会发生。

如果文件是以 BYTE 的模式打开的，那么 INVALID OPERATION 异常就会发生。

11. GET_ LINE

定义：

```
PROCEDURE    GET_LINE(
    FILE            IN    FILE_TYPE,
    BUFFER          OUT VARCHAR,
    LEN             IN    INTEGER DEFAULT NULL
) ;
```

功能说明：

读取指定文件的一行到提供的缓存。读取的文本不包括该行的终结符，或者直到文件的末尾，或者到指定的长度。并且不能超过 FOPEN 时指定的 MAX_LINESIZE。

参数说明：

FILE：通过 FOPEN 调用，返回的活动文件句柄。

BUFFER：接收数据文件中行读的数据缓冲区。

LEN：从文件中读取的字节数。默认情况为 NULL，NULL 表示读取的字节数为 MAX_LINESIZE。

注意事项：

如果该行不匹配缓冲区，则发生 READ_ERROR 异常。如果到文件结束了仍没有文本读取，发生 NO_DATA_FOUND 异常。如果 FILE 是以 BYTE 模式打开的，则产生 INVALID_OPERATION 异常。

由于行的结束符不被读取，读取空行则返回空字符串。

缓冲区最大大小为 32767 字节，除非 FOPEN 指定了更小的值。如果未指定，默认值为 1024。

12. GET_ RAW

定义：

```
FUNCTION GET_RAW(
    FILE        IN    FILE_TYPE,
```

```
    BUFFER      OUT BLOB,
    LEN         IN   INTEGER DEFAULT NULL
)RETURN INTEGER;
```

功能说明：

从文件中读取原始字符串值，并通过读取的字节数调整文件指针的头部。GET_RAW 函数忽略掉行结束符，并返回通过 GET_RAW 的 LEN 参数请求的实际字节数。

参数说明：

FILE： 通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

BUFFER： 接收从文件中读取的数据行的缓冲区。

LEN： 从文件中读取的字节数。默认情况为 NULL，NULL 表示读取的字节数为 MAX_LINESIZE。

注意事项：

如果子程序读取起过文件末尾，那么将产生 DATA_NO_FOUND 的异常。程序在执行循环时应该允许捕获这个异常。

13. IS_ OPEN

定义：

```
FUNCTION   IS_OPEN(
    FILE       IN FILE_TYPE
)RETURN BOOLEAN;
```

功能说明：

判断文件句柄指定的文件是否已打开，如果已打开则返回 TRUE，否则 FALSE。不保证用户在尝试使用该文件句柄时没有操作系统的权限。

参数说明：

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

14. NEW_ LINE

定义：

```
PROCEDURE NEW_LINE(
    FILE        IN FILE_TYPE,
    LINES       IN INTEGER DEFAULT 1
);
```

功能说明：

在当前位置输出一个或多个行终止符。

参数说明：

FILE：通过 FOPEN 或 FOPEN_NCHAR 调用，返回的活动文件句柄。

LINES：写入文件的行终结符数量。

异常：

INVALID_FILEHANDLE

INVALID_OPERATION

WRITE_ERROR

15. FPUT

定义：

```
PROCEDURE FPUT(
    FILE        IN FILE_TYPE,
    BUFFER      IN VARCHAR
);
```

功能说明：

把缓冲区中的文本字符写入到数据文件。文件必须是以写模式打开的。UTL_FILE.PUT 输出数据时不会附加行终止符。使用 NEW_LINE 来添加行结束符或者使用 PUT_LINE 来写一行带有结束符的数据。

参数说明：

FILE：由 FOPEN_NCHAR 返回的文件句柄。

BUFFER：包含要写入文件的数据缓存。

注意事项：

如果 FOPEN 没有指定一个更小的值，BUFFER 参数的最大值将是 32767。如果未指定，FOPEN 默认的行最大值为 1024。连续调用 PUT 的总和在没有缓冲区刷页的情况下不能超

过 32767。

异常：

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

16. PUT_ LINE

定义：

```
PROCEDURE PUT_LINE(
    FILE          IN FILE_TYPE,
    BUFFER        IN VARCHAR,
    AUTOFLUSH IN BOOLEAN DEFAULT FALSE
);
```

功能说明：

把缓冲区中的文本字符串写入数据文件，同时输出一个与系统有关的行终止符。

参数说明：

FILE：由 FOPEN 返回的文件句柄。
BUFFER：包含要写入文件的数据缓存。
AUTOFLUSH：数据写完后，自动清理缓冲区

注意事项：

缓区大小最大为 32767。
如果文件有 BYTE 模式打开，则产生 INVALID_OPERATION 异常。

异常：

INVALID_FILEHANDLE
INVALID_OPERATION
WRITE_ERROR

17. PUT_ RAW

定义：

```
PROCEDURE PUT_RAW(
    FILE          IN FILE_TYPE,
    BUFFER        IN BLOB,
    AUTOFLUSH IN BOOLEAN DEFAULT FALSE
);
```

功能说明：

接收输入的原始数据并将其写入缓存。

参数说明：

FILE：由 FOPEN 返回的文件句柄；
BUFFER：包含要写入文件的数据缓存；
AUTOFLUSH：数据写完后，自动清理缓冲区。

注意事项：

通过设置第三个参数可以使缓冲区自动冲刷。
BUFFER 最大值为 32767。

18. PUTF

定义：

```
PROCEDURE PUTF(
    FILE         IN FILE_TYPE,
    FORMAT       IN VARCHAR,
    ARG1         IN VARCHAR DEFAULT NULL,
    ARG2         IN VARCHAR DEFAULT NULL,
    ARG3         IN VARCHAR DEFAULT NULL,
    ARG4         IN VARCHAR DEFAULT NULL,
    ARG5         IN VARCHAR DEFAULT NULL
);
```

功能说明：

以一个模版样式输出至多 5 个字符串，类似 C 中的 PRINTF（）。

参数说明：

FILE：由 FOPEN 返回的文件句柄。

FORMAT：决定格式的格式串。格式串可使用以下样式：1）%S 在格式串中可以使用最多 5 个%S，与后面的 5 个参数一一对应。%S 会被后面的参数依次填充，如果没有足够的参数，%S 会被忽视，不被写入文件；2）\N 换行符。在格式串中没有个数限制。

ARGN：可选的 5 个参数，最多 5 个。

注意事项：

如果文件以 BYTE 模式打开，返回 INVALID_OPERATION 错误。

%S，\N 与 C 语言中的类似。

20.16.3 错误处理

错误、异常情况释义：

1、EC_OPEN_FILE_FAILED：打开文件失败
2、EC_RN_INVALID_MODE_NAME：无效的文件模式名
3、EC_WRITE_FILE_FAILED：写入文件失败
4、EC_FILE_CLOSE_FAILED：文件关闭失败
5、EC_FILE_REMOVE_FAILED：文件删除失败
6、EC_FILE_NAME_TOO_LONG：文件名过长
7、EC_FILE_PATH_TOO_LONG：文件路径名过长
8、EC_CANNOT_FIND_DATA：无法获取数据
9、EC_INVALID_OPERATION：无效的操作
10、EC_INVALID_OFFSET：无效的偏移
11、EC_RN_INVALID_STRINGFORMAT：无效的字符串格式
12、EC_RN_TOO_MANY_FILES：打开文件数目过多
13、EC_FILE_EXIST：文件已存在

20.16.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

在成功调用下面 3 个例子中的过程和函数之前，先要建好文件 C:\UTL_FILE_TEMP\U12345.TMP。文件的内容为：

```
THIS IS LINE 1.
THIS IS LINE 2.
THIS IS LINE 3.
THIS IS LINE 4.
THIS IS LINE 5.
```

例 1 使用 GET_LINE 过程读文件

```
DECLARE
   V1 VARCHAR2(8186);
   F1 UTL_FILE.FILE_TYPE;
BEGIN

   -- 本例子中 MAX_LINESIZE 小于 GET_LINE 的长度要求，所以返回的字节数小于等于 256。
   F1 := UTL_FILE.FOPEN('C:\UTL_FILE_TEMP','U12345.TMP','R',256);
   UTL_FILE.GET_LINE(F1,V1,32767);
   PRINT 'GET_LINE:'||V1;
   UTL_FILE.FCLOSE(F1);

   -- FOPEN 的 MAX_LINESIZE 是 NULL ， NULL 即为默认的 1024， 所以返回的字节数小于等于 1024。
   F1 := UTL_FILE.FOPEN('C:\UTL_FILE_TEMP','U12345.TMP','R');
```

```
    UTL_FILE.GET_LINE(F1,V1,32767);
    PRINT 'GET_LINE:'||V1;
    UTL_FILE.FCLOSE(F1);

    -- GET_LINE 未指定字节数，所以默认为 NULL1024，NULL 即为 1024。
    F1 := UTL_FILE.FOPEN('C:\UTL_FILE_TEMP','U12345.TMP','R');
    UTL_FILE.GET_LINE(F1,V1);
    PRINT 'GET_LINE:'||V1;
    UTL_FILE.FCLOSE(F1);
END;
```

每个过程都是输出第一行，这个例子的结果如下所示:

```
GET LINE: THIS IS LINE 1.
GET LINE: THIS IS LINE 1.
GET LINE: THIS IS LINE 1.
```

例 2 使用 PUTF 过程向文件尾部添加内容。

```
DECLARE
    HANDLE UTL_FILE.FILE_TYPE;
    MY_WORLD    VARCHAR2(4) := 'ZORK';
BEGIN
    HANDLE := UTL_FILE.FOPEN('C:\UTL_FILE_TEMP', 'U12345.TMP', 'A');
    UTL_FILE.PUTF(HANDLE, '\NHELLO, WORLD!\\NI COME FROM %S WITH %S.\\N',
MY_WORLD,'GREETINGS FOR ALL EARTHLINGS');
    UTL_FILE.FFLUSH(HANDLE);
    UTL_FILE.FCLOSE(HANDLE);
END;
/
```

该例子在 U12345.TMP 文件的末尾添加了如下内容：

```
HELLO, WORLD!
I COME FROM ZORK WITH GREETINGS FOR ALL EARTHLINGS.
```

例 3 使用 GET_RAW 过程从 UTL_FILE_TEMP 路径下的 U12345.TMP 文件中读取原始信息。首先，将 U12345.TMP 文件中的内容写为：HELLO WORLD!

```
CREATE OR REPLACE PROCEDURE GETRAW(N IN VARCHAR2) IS
   H        UTL_FILE.FILE_TYPE;
   BUF      BLOB;
   AMNT    CONSTANT BINARY_INTEGER := 32767;
BEGIN
   H := UTL_FILE.FOPEN('C:\UTL_FILE_TEMP', N, 'R', 32767);
   UTL_FILE.GET_RAW(H, BUF, AMNT);
   PRINT 'THIS IS THE RAW DATA:';
   SELECT BUF;
   UTL_FILE.FCLOSE (H);
END;
/
```

过程建好后，运行如下语句：

```
BEGIN
GETRAW('U12345.TMP');
END;
/
```

输出结果为：

```
THIS IS THE RAW DATA:
行号     BUF
-----    -------
1        54686973206973206C696E6520312E0A
```

20.17 UTL_INADDR 包

为了在 PL/SQL 中，提供网络地址转换的支持，开发 UTL_INADDR 包，提供一组 API，实现主机的 IP 地址和主机名之间的转换。

20.17.1 相关方法

1. GET_HOST_ADDRESS

定义：

```
FUNCTION GET_HOST_ADDRESS (
HOST IN VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

功能说明：

依据给定的主机名返回其 IP 地址。返回给定 HOST 主机的 IP 地址（以 CHAR 的形式展现）。

参数说明：

HOST：主机名，VARCHAR2 类型。默认值为 NULL，用本机的主机名代替，表示获取本机的 IP 地址；

2. GET_HOST_NAME

定义：

```
FUNCTION GET_HOST_NAME(
IP IN VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

功能说明：

依据给定的 IP 地址返回主机名；

参数说明：

IP： IP 地址，VARCHAR2 类型。默认值为 NULL，用本机的 IP 地址代替，表示获取本机的主机名。

20.17.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 获得 192.168.0.30 机器的名称，以及名称为 TEST 机器的 IP 地址。

```
SELECT UTL_INADDR.GET_HOST_NAME('192.168.0.30');
SELECT UTL_INADDR.GET_HOST_ADDRESS('TEST');
```

20.18 UTL_TCP 包

许多的应用程序都是基于 TCP/IP 协议的，UTL_TCP 包的功能是提供一个可以与外部的 TCP/IP 服务器通讯，并能够进行基本的收发数据的功能。

20.18.1 相关方法

1. AVAILABLE

定义：

```
FUNCTION AVAILABLE(
C IN OUT CONNECTION,
TIMEOUT IN INT DEFAULT 0
)RETURN INT;
```

功能说明：

当前能够读取的连接字符数(最大不超过 96 字节)。

参数说明：

C：一个 OPEN_CONNECTION 打开的、有效的 CONNECTION；

TIMEOUT：如果没有数据，则在制定的 TIMEOUT 时间返回。

2. CLOSE_ALL_CONNECTIONS

定义：

```
UTL_TCP.CLOSE_ALL_CONNECTIONS;
```

功能说明：

关闭所有的连接。

3. CLOSE_CONNECTION

定义：

```
UTL_TCP.CLOSE_CONNECTION (
C IN OUT    CONNECTION
);
```

功能说明：

关闭指定的连接。

4. GET_RAW

定义：

```
FUNCTION GET_RAW    (C IN OUT CONNECTION, LEN IN INT DEFAULT 1, PEEK IN BOOLEAN DEFAULT FALSE)     RETURN VARBINARY;
```

功能说明：

读取指定长度（二进制的数据）的数据。

参数说明：

C：一个 OPEN_CONNECTION 打开的、有效的 CONNECTION；

LEN：要收到的数据的长度，默认值 1；

PEEK：该选项为 TRUE，则指只是查看队列的数据，不会从队列中删除，下次还能够读到；为 FALSE，则会删除队列中的数据。默认参数，FALSE。

5. OPEN_CONNECTION

定义：

```
FUNCTION OPEN_CONNECTION(REMOTE_HOST IN VARCHAR2, REMOTE_PORT IN INT, LOCAL_HOST IN VARCHAR2 DEFAULT NULL, LOCAL_PORT IN INT DEFAULT NULL, TX_TIMEOUT IN INT DEFAULT NULL) RETURN CONNECTION;
```

功能说明：

打开一个连接到外部服务器的连接。

参数说明：

REMOTE_HOST：远程主机的名字，或者 IP 地址；

REMOTE_PORT：远程主机的端口；

LOCAL_HOST：本地主机的名字（一般不用设置）；

LOCAL_PORT：本地主机的端口（一般不用设置）；

TX_TIMEOUT：尝试链接到远程主机的等待时间。

6. READ_RAW

定义：

```
FUNCTION READ_RAW (
C IN OUT CONNECTION,
DATA OUT VARBINARY,
LEN IN INT DEFAULT 1,
PEEK IN BOOLEAN DEFAULT FALSE
) RETURN INT;
```

功能说明：

以二进制形式读取一行数据。

参数说明：

C：一个 OPEN_CONNECTION 打开的、有效的 CONNECTION；

DATA：用于存放读到的数据；

LEN：要收到的数据的长度，默认值 1；

PEEK：该选项为 TRUE，则指只是查看队列的数据，不会从队列中删除，下次还能够读到；为 FALSE，则会删除队列中的数据。默认参数，FALSE。

7. WRITE_RAW

定义：

```
FUNCTION WRITE_RAW(
C IN OUT CONNECTION,
DATA IN VARBINARY,
LEN IN INT DEFAULT NULL
) RETURN INT;
```

功能说明：

发送一行二进制数据。

参数说明：

C：一个 OPEN_CONNECTION 打开的、有效的 CONNECTION；

DATA：存放要发送的数据；

LEN：要发送数据的长度。

20.18.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 UTL_TCP 包示例。

```
DECLARE
        HOST VARCHAR(128);
        PORT INT;
        RET         INT;
        LEN         INT;
        DATA VARBINARY(512);
        SHOW      VARBINARY(128);
        C UTL_TCP.CONNECTION;
BEGIN
        HOST := '192.168.0.212';
        PORT := 25;
        C := UTL_TCP.OPEN_CONNECTION(HOST,PORT);
        RET := UTL_TCP.WRITE_RAW(C, RAWTOHEX('EHLO WXH@DAMENG.SHANGHAI'),50);
        PRINT RET;
        LEN := 48;
        RET := UTL_TCP.AVAILABLE(C,0);
        PRINT RET;
        SHOW := UTL_TCP.GET_RAW(C,LEN);
        PRINT SHOW;       -- ' 220 192.168.0.212 ESMTP '的二进制流
        RET := UTL_TCP.READ_RAW(C,DATA,LEN);
        UTL_TCP.CLOSE_CONNECTION(C);
END;
```

结果如下：

```
24
25
323230203139322E3136382E302E3231322045534D54500D0A
```

20.19 UTL_MAIL 包

为了在 PL/SQL 中，提供 EMAIL 发送的支持，开发 UTL_MAIL 包，提供一组 API，实现发送简单的邮件和包含附件的邮件。

20.19.1 相关方法

1. INIT

定义：

```
PROCEDURE INIT(
        SMTP_SERVER VARCHAR2(128),
        SMTP_PORT     INT DEFAULT 25,
        SENDER          VARCHAR2(128),
        PASSWORD      VARCHAR2(128)
) ;
```

功能说明：

包需要的配置信息。

参数说明：

SMTP_SERVER：SMTP 服务器的地址，如 SMTP.163.COM；

SMTP_PORT：SMTP 的端口号，一般都是 25；

SENDER：设置发送的邮箱，也就是要认证的邮箱；

PASSWORD：设置发送邮箱的密码。

2. SEND

定义：

```
PROCEDURE SEND (
        SENDER IN VARCHAR2,
        RECIPIENTS IN VARCHAR2,
        CC IN VARCHAR2 DEFAULT NULL,
        BCC IN VARCHAR2 DEFAULT NULL,
        SUBJECT IN VARCHAR2 DEFAULT NULL,
        MESSAGE IN VARCHAR2 ,
        MIME_TYPE IN VARCHAR2 DEFAULT 'TEXT/PLAIN; CHARSET=US-ASCII',
        PRIORITY IN INT DEFAULT NULL
);
```

功能说明：

发送 MESSAGE 的信息。

参数说明：

SENDER：发送的邮箱地址；

RECIPIENTS：接收邮件的地址；

CC：抄送的邮件的地址；

BCC：秘密抄送的邮件的地址；

SUBJECT：邮件的主题；

MESSAGE：邮件的内容，正文部分；

MIME_TYPE：邮件的内容格式，DEFAULT IS 'TEXT/PLAIN; CHARSET=US-ASCII'；

PRIORITY：优先级（不用设置）。

3. SEND_ATTACH_RAW

定义：

```
PROCEDURE SEND_ATTACH_RAW (
        SENDER IN VARCHAR2,
        RECIPIENTS IN VARCHAR2,
        CC IN VARCHAR2 DEFAULT NULL,
        BCC IN VARCHAR2 DEFAULT NULL,
        SUBJECT IN VARCHAR2 DEFAULT NULL,
        MESSAGE IN VARCHAR2 DEFAULT NULL,
        MIME_TYPE IN VARCHAR2 DEFAULT 'TEXT/PLAIN; CHARSET=US-ASCII',
        PRIORITY IN INT DEFAULT NULL,
        ATTACHMENT IN BLOB,
        ATT_INLINE IN BOOLEAN DEFAULT TRUE,
        ATT_MIME_TYPE IN VARCHAR2 DEFAULT 'APPLICATION/OCTET-STREAM',
        ATT_FILENAME IN VARCHAR2 DEFAULT NULL
);
```

功能说明：

发送带附件的邮件。

参数说明：

SENDER：发送的邮箱地址；

RECIPIENTS：接收邮件的地址；

CC：抄送的邮件的地址；
BCC：秘密抄送的邮件的地址；
SUBJECT：邮件的主题；
MESSAGE：邮件的内容，正文部分；
MIME_TYPE：邮件的内容格式：文本、HTML 等。默认为'TEXT/PLAIN;CHARSET=US-ASCII';
PRIORITY：优先级（不用设置）；
ATTACHMENT：附件；
ATT_INLINE：附件在文件中显示是否使用下划线；
ATT_MIME_TYPE：邮件附件的类型，如文本；
ATT_FILENAME：附件的文件名。

20.19.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 在使用该包前，需要先初始化该包，调用 INIT 函数，设置使用 SMTP 服务器，端口（一般默认 25，设置用于认证的邮箱，认证的密码。设置完后，就可以使用包的函数：SEND 和 SEND_ATTACH_RAW。

```
/*初始化包的函数*/
UTL_MAIL.INIT(
        'SMTP.163.COM',             //SMTP 服务器
        25,                         //端口
        'UTL_MAIL@163.COM',   //认证的邮箱
        'DM_UTL_MAIL'               //认证邮箱的密码
);
/*发送不带附件的邮件*/
UTL_MAIL.SEND(
        'UTL_MAIL@163.COM',                 //发送的邮箱地址
        'RECIPIENT@DAMENG.COM',         //接受的邮箱地址
        '',
        '',
        'TEST',                             //主题
        'THIS IS A MAIL FOR TEST .',        //正文
        'TEXT/PLAIN; CHARSET=US-ASCII',//类型
        3
);
/*  发送带附件的邮件  */
UTL_MAIL.SEND_ATTACH_RAW(
        'UTL_MAIL@163.COM',
        'RECIPIENT@DAMENG.COM',
        '',
        '',
        'TEST',
        'THIS IS A MAIL FOR TEST .',
        'TEXT/PLAIN; CHARSET=US-ASCII',
        3,
        HEXTORAW('12345678905645318131694341964316513983208431264654654654654615 64'),  // 附件二进制的内容
        FALSE,
        'APPLICATION/OCTET-STREAM',         //附件的类型
        'ATTACH.ZIP'                        //附件的名字
);
```

20.20 DBMS_OBFUSCATION_TOOLKIT 包

为了保护敏感数据，DM7 提供一个数据加密包 DBMS_OBFUSCATION_TOOLKIT。利用这个包，用户可以对数据进行 DES、DES3 加密，或者对数据进行 MD5 散列。

20.20.1 相关方法

DBMS_OBFUSCATION_TOOLKIT 包中包含的过程和函数如下详细介绍：

1. DES3DECRYPT

定义：

```
PROCEDURE DES3DECRYPT(
  INPUT                IN      VARBINARY,
  KEY                  IN      VARBINARY,
  DECRYPTED_DATA       OUT     VARBINARY,
  WHICH                IN       INT              DEFAULT    0,
  IV                   IN      VARBINARY         DEFAULT NULL);
```

功能说明：

对 VARBINARY 数据进行 DES3 解密。

返回值：

解密后的 VARBINARY 数据。

定义：

```
PROCEDURE DES3DECRYPT(
  INPUT_STRING          IN       VARCHAR2,
  KEY_STRING            IN       VARCHAR2,
  DECRYPTED_STRING      OUT      VARCHAR2,
  WHICH                 IN       INT              DEFAULT    0,
  IV_STRING             IN       VARCHAR2         DEFAULT NULL);
```

功能说明：

对 VARCHAR2 数据进行 DES3 解密。

返回值：

解密后的 VARCHAR2 类型数据。

定义：

```
FUNCTION DES3DECRYPT(
  INPUT             IN    VARBINARY,
  KEY               IN    VARBINARY,
  WHICH             IN    INT DEFAULT 0,
  IV                IN    VARBINARY DEFAULT NULL)
RETURN VARBINARY;
```

功能说明：

对 VARBINARY 数据进行 DES3 解密。

返回值：

解密后的 VARBINARY 数据。

定义：

```
FUNCTION DES3DECRYPT(
  INPUT_STRING    IN    VARCHAR2,
  KEY_STRING      IN    VARCHAR2,
  WHICH           IN     INT             DEFAULT 0,
  IV_STRING       IN    VARCHAR2        DEFAULT NULL)
```

```
RETURN VARCHAR2;
```

功能说明：

对 VARCHAR2 数据进行 DES3 解密。

返回值：

解密后的 VARCHAR2 数据。

参数说明：

INPUT_STRING：输入参数，需要解密的数据；

KEY_STRING：输入参数，解密数据使用的密钥；

WHICH：输入参数，解密模式，可选值 0、1。0 表示双密钥 DES3 解密，1 表示三密钥 DES3 解密；

IV_STRING：输入参数，解密数据使用的初始化向量。

2. DES3ENCRYPT

定义：

```
PROCEDURE DES3ENCRYPT(
 INPUT                IN      VARBINARY,
 KEY                  IN      VARBINARY,
 ENCRYPTED_DATA       OUT     VARBINARY,
 WHICH                IN      INT             DEFAULT 0,
 IV                   IN      VARBINARY       DEFAULT NULL);
```

功能说明：

对 VARBINARY 数据进行 DES3 加密。

参数说明：

INPUT：输入参数，需要加密的数据；

KEY：输入参数，加密数据使用的密钥；

ENCRYPTED_DATA：输出参数，被加密后的数据；

WHICH：输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。

IV：输入参数，加密数据使用的初始化向量。

定义：

```
PROCEDURE DES3ENCRYPT(
 INPUT_STRING         IN      VARCHAR2,
 KEY_STRING           IN      VARCHAR2,
 ENCRYPTED_STRING     OUT     VARCHAR2,
 WHICH                IN      INT             DEFAULT     0,
 IV_STRING            IN      VARCHAR2        DEFAULT     NULL);
```

功能说明：

对 VARCHAR2 数据进行 DES3 加密。

参数说明：

INPUT_STRING：输入参数，需要加密的数据；

KEY_STRING：输入参数，加密数据使用的密钥；

ENCRYPTED_STRING：输出参数，被加密后的数据；

WHICH：输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。

IV_STRING：输入参数，加密数据使用的初始化向量。

定义：

```
FUNCTION DES3ENCRYPT(
 INPUT          IN    VARBINARY,
```

```
  KEY              IN   VARBINARY,
  WHICH            IN   INT              DEFAULT 0,
  IV               IN   VARBINARY   DEFAULT NULL)
RETURN VARBINARY;
```

功能说明：

对 VARBINARY 数据进行 DES3 加密。

参数说明：

INPUT：输入参数，需要加密的数据；

KEY：输入参数，加密数据使用的密钥；

WHICH：输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密；

IV：输入参数，加密数据使用的初始化向量。

返回值：加密后的 VARBINARY 数据。

定义：

```
FUNCTION DES3ENCRYPT(
  INPUT_STRING   IN   VARCHAR2,
  KEY_STRING     IN   VARCHAR2,
  WHICH          IN   INT            DEFAULT   0,
  IV_STRING      IN   VARCHAR2       DEFAULT   NULL)
RETURN VARCHAR2;
```

功能说明：

对 VARCHAR2 数据进行 DES3 加密。

参数说明：

INPUT_STRING：输入参数，需要加密的数据；

KEY_STRING：输入参数，加密数据使用的密钥；

WHICH：输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密；

IV_STRING：输入参数，加密数据使用的初始化向量。

返回值：加密后的 VARCHAR2 数据。

3. DES3GETKEY

定义：

```
PROCEDURE DES3GETKEY(
  WHICH          IN    INT            DEFAULT   0,
  SEED           IN    VARBINARY,
  KEY            OUT   VARBINARY);
```

功能说明：

生成 VARBINARY 类型数据的 DES3 加密密钥。

参数说明：

WHICH：输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3；

SEED：输入参数，获取 DES3 密钥使用的种子；

KEY：输出参数，VARBINARY 类型的 DES3 加密密钥。

定义：

```
PROCEDURE DES3GETKEY(
  WHICH          IN    INT            DEFAULT 0,
  SEED_STRING    IN    VARCHAR2,
  KEY            OUT   VARCHAR2);
```

功能说明：

生成 VARCHAR2 类型的 DES3 加密密钥。

参数说明：

WHICH：输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3；

SEED_STRING：输入参数，获取 DES3 密钥使用的种子；

KEY：输出参数，VARCHAR2 类型的 DES3 加密密钥。

定义：

```
FUNCTION DES3GETKEY(
 WHICH   IN    INT         DEFAULT   0,
 SEED    IN    VARBINARY)
RETURN VARBINARY;
```

功能说明：

生成 VARBINARY 类型的 DES3 加密密钥。

参数说明：

WHICH：输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3；

SEED：输入参数，获取 DES3 密钥使用的种子。

返回值：VARBINARY 类型的 DES3 加密密钥。

定义：

```
FUNCTION DES3GETKEY(
 WHICH             IN   INT          DEFAULT 0,
 SEED_STRING   IN   VARCHAR2)
RETURN VARCHAR2;
```

功能说明：

生成 VARCHAR2 类型的 DES3 加密密钥。

参数说明：

WHICH：输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3；

SEED_STRING：输入参数，获取 DES3 密钥使用的种子。

返回值：VARCHAR2 类型的 DES3 加密密钥。

4. DESDECRYPT

定义：

```
PROCEDURE DESDECRYPT(
 INPUT                IN      VARBINARY,
 KEY                  IN      VARBINARY,
 DECRYPTED_DATA       OUT     VARBINARY);
```

功能说明：

对 VARBINARY 数据进行 DES 解密。

参数说明：

INPUT：输入参数，需要解密的数据；

KEY：输入参数，解密数据使用的密钥；

DECRYPTED_DATA：输出参数，被解密后的数据。

定义：

```
PROCEDURE DESDECRYPT(
 INPUT_STRING         IN      VARCHAR2,
 KEY_STRING           IN      VARCHAR2,
```

```
 DECRYPTED_STRING      OUT     VARCHAR2);
```

功能说明：

对 VARCHAR2 数据进行 DES 解密。

参数说明：

INPUT_STRING：输入参数，需要解密的数据；

KEY_STRING：输入参数，解密数据使用的密钥；

DECRYPTED_STRING：输出参数，被解密后的数据。

定义：

```
FUNCTION DESDECRYPT(
 INPUT              IN    VARBINARY,
 KEY                IN    VARBINARY)
RETURN VARBINARY;
```

功能说明：

对 VARBINARY 数据进行 DES 解密。

参数说明：

INPUT：输入参数，需要解密的数据；

KEY：输入参数，解密数据使用的密钥。

返回值：解密后的 VARBINARY 数据。

定义：

```
FUNCTION DESDECRYPT(
 INPUT_STRING    IN    VARCHAR2,
 KEY_STRING      IN    VARCHAR2)
RETURN VARCHAR2;
```

功能说明：

对 VARCHAR2 数据进行 DES 解密。

参数说明：

INPUT_STRING：输入参数，需要解密的数据；

KEY_STRING：输入参数，解密数据使用的密钥。

返回值：解密后的 VARCHAR2 数据。

5. DESENCRYPT

定义：

```
PROCEDURE DESENCRYPT(
 INPUT                IN        VARBINARY,
 KEY                  IN        VARBINARY,
 ENCRYPTED_DATA       OUT       VARBINARY);
```

功能说明：

对 VARBINARY 数据进行 DES 加密。

参数说明：

INPUT：输入参数，需要加密的数据；

KEY：输入参数，加密数据使用的密钥；

ENCRYPTED_DATA：输出参数，被加密后的数据。

定义：

```
PROCEDURE DESENCRYPT(
 INPUT_STRING         IN        VARCHAR2,
 KEY_STRING           IN        VARCHAR2,
 ENCRYPTED_STRING     OUT       VARCHAR2);
```

功能说明：

对 VARCHAR2 数据进行 DES 加密。

参数说明：

INPUT_STRING：输入参数，需要加密的数据；

KEY_STRING：输入参数，加密数据使用的密钥；

ENCRYPTED_STRING：输出参数，被加密后的数据。

定义：

```
FUNCTION DESENCRYPT(
  INPUT             IN VARBINARY,
  KEY               IN VARBINARY)
RETURN VARBINARY;
```

功能说明：

对 VARBINARY 数据进行 DES 加密。

参数说明：

INPUT：输入参数，需要加密的数据；

KEY：输入参数，加密数据使用的密钥。

返回值：加密后的 VARBINARY 数据。

定义：

```
FUNCTION DESENCRYPT(
  INPUT_STRING    IN    VARCHAR2,
  KEY_STRING      IN    VARCHAR2)
RETURN VARCHAR2;
```

功能说明：

对 VARCHAR2 数据进行 DES 加密。

参数说明：

INPUT_STRING：输入参数，需要加密的数据；

KEY_STRING：输入参数，加密数据使用的密钥。

返回值：加密后的 VARCHAR2 数据。

6. DESGETKEY

定义：

```
PROCEDURE DESGETKEY(
SEED              IN      VARBINARY,
  KEY               OUT    VARBINARY);
```

功能说明：

生成 VARBINARY 类型的 DES 加密密钥。

参数说明：

SEED：输入参数，获取 DES 密钥使用的种子；

KEY：输出参数，VARBINARY 类型的 DES 加密密钥。

定义：

```
PROCEDURE DESGETKEY(
  SEED_STRING   IN      VARCHAR2,
  KEY               OUT    VARCHAR2);
```

功能说明：

生成 VARCHAR2 类型的 DES 加密密钥。

参数说明：

SEED_STRING：输入参数，获取 DES 密钥使用的种子；

KEY：输出参数，VARCHAR2 类型的 DES 加密密钥。

定义：

```
FUNCTION DESGETKEY(
  SEED       IN      VARBINARY)
RETURN VARBINARY;
```

功能说明：

生成 VARBINARY 类型的 DES 加密密钥。

参数说明：

SEED：输入参数，获取 DES 密钥使用的种子；

返回值：VARBINARY 类型的 DES 加密密钥。

定义：

```
FUNCTION DESGETKEY(
  SEED_STRING    IN    VARCHAR2)
RETURN VARCHAR2;
```

功能说明：

生成 VARCHAR2 类型的 DES 加密密钥。

参数说明：

SEED_STRING：输入参数，获取 DES 密钥使用的种子；

返回值：VARCHAR2 类型的 DES 加密密钥。

7. MD5

定义：

```
PROCEDURE MD5(
  INPUT                    IN       VARBINARY,
  CHECKSUM                 OUT    VARBINARY);
```

功能说明：

生成 VARBINARY 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

参数说明：

INPUT：输入参数，需要进行散列的数据；

CHECKSUM：输出参数，经 MD5 散列后的数据。

定义：

```
PROCEDURE MD5(
  INPUT_STRING           IN      VARCHAR2,
  CHECKSUM_STRING    OUT    VARCHAR2);
```

功能说明：

生成 VARCHAR2 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

参数说明：

INPUT_STRING：输入参数，需要进行散列的数据；

CHECKSUM_STRING：输出参数，经 MD5 散列后的数据。

定义：

```
FUNCTION MD5(
  INPUT                    IN       VARBINARY)
  RETURN VARBINARY;
```

功能说明：

生成 VARBINARY 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

参数说明：

INPUT：输入参数，需要进行散列的数据；

返回值：MD5 散列后的 VARBINARY 数据。

定义：

```
FUNCTION MD5(
  INPUT_STRING          IN     VARCHAR2)
RETURN VARCHAR2;
```

功能说明：

生成 VARCHAR2 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

参数说明：

INPUT_STRING：输入参数，需要进行散列的数据；

返回值：MD5 散列后的 VARCHAR2 数据。

20.20.2 使用说明

1. 被加解密的数据不能为空。
2. DM7 的 DES3 加解密目前只支持双密钥，即 WHICH 默认为 0 的情况。
3. IV 加解密数据使用的初始化向量目前不使用，默认为 NULL，此时采用系统默认的初始化向量。
4. DES3GETKEY 和 DESGETKEY 生成加密密钥的种子 SEED 目前不使用，因为 DM7 用于生成密钥的环境已经配置好，只需要根据密钥的大小生成指定长度的密钥即可。
5. DES 和 DES3 加解密的块大小为 8BYTES，因此用于加解密的数据必须是 8 的整数倍。
6. DES 加解密使用的密钥长度为 8，多余 8 后的字符被忽略。
7. 双密钥 DES3 加解密使用的密钥长度为 16，多余 16 后的字符被忽略。
8. 三密钥 DES3 加解密使用的密钥长度为 24，多余 24 后的字符被忽略。
9. 十六进制在 DM7 对应的是 VARBINARY 数据类型，在 ORACLE 中对应的是 RAW 类型。
10. MD5 主要用途是对数据进行散列，用于校验。不同的机器生成的散列值不同。

20.20.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程 SP_CREATE_SYSTEM_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
SET SERVEROUTPUT ON;   --dbms_output.put_line 需要设置这条语句，才能打印出消息
```

例 1　分别使用 DES3ENCRYPT 加密算法、DES3DECRYPT 解密算法对 VARBINARY 类型数据进行加密解密。

```
DECLARE
RAW_INPUT                    VARBINARY(128)    :=    '74696765727469676572746967657274';
--'TIGERTIGERTIGERT'
RAW_KEY                      VARBINARY(128)    :=    '73636F747473636F747473636F747473';
--'SCOTTSCOTTSCOTTS'
ENCRYPTED_RAW                VARBINARY(2048);
DECRYPTED_RAW                VARBINARY(2048);

BEGIN
DBMS_OUTPUT.PUT_LINE('> ========== BEGIN TEST RAW DATA ==========');
DBMS_OUTPUT.PUT_LINE('> RAW INPUT                                    : ' ||
                RAW_INPUT);
```

```
        DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT(RAW_INPUT,
                RAW_KEY, ENCRYPTED_RAW );
        DBMS_OUTPUT.PUT_LINE('> ENCRYPTED HEX VALUE                    : ' ||
                ENCRYPTED_RAW);
        DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(ENCRYPTED_RAW,
                RAW_KEY, DECRYPTED_RAW);
        DBMS_OUTPUT.PUT_LINE('> DECRYPTED RAW OUTPUT                   : ' ||
                        DECRYPTED_RAW);
        DBMS_OUTPUT.PUT_LINE('>    ');
        IF RAW_INPUT =
                        DECRYPTED_RAW THEN
            DBMS_OUTPUT.PUT_LINE('>    RAW    DES    ENCYPTION    AND    DECRYPTION
SUCCESSFUL');
        END IF;

    END;
/
```

结果：

```
> ========= BEGIN TEST RAW DATA =========
> RAW INPUT                              : 74696765727469676572746967657274
> ENCRYPTED HEX VALUE                    : F77BB98AF8E7F59E329C31027CAF6C98
> DECRYPTED RAW OUTPUT                   : 74696765727469676572746967657274
>
> RAW DES ENCYPTION AND DECRYPTION SUCCESSFUL
```

例 2　分别使用 DESENCRYPT 加密算法、DESDECRYPT 解密算法对 VARCHAR2 类型的数据进行加密、解密。

```
DECLARE
INPUT_STRING            VARCHAR2(8)    := 'DM123456';
KEY_STRING              VARCHAR2(8)    := 'PASSWORD';

ENCRYPTED_STRING        VARCHAR2(2048);
DECRYPTED_STRING        VARCHAR2(2048);

BEGIN
        DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT(
                        INPUT_STRING, KEY_STRING, ENCRYPTED_STRING );
        DBMS_OUTPUT.PUT_LINE('> ENCRYPTED STRING                    : ' ||
                        ENCRYPTED_STRING);
        DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT(
                        ENCRYPTED_STRING,
                        KEY_STRING,
                        DECRYPTED_STRING);
        DBMS_OUTPUT.PUT_LINE('> DECRYPTED OUTPUT                    : ' ||
                        DECRYPTED_STRING);
        DBMS_OUTPUT.PUT_LINE('>    ');
        IF INPUT_STRING =
                        DECRYPTED_STRING THEN
            DBMS_OUTPUT.PUT_LINE('> DES ENCRYPTION AND DECRYPTION SUCCESSFUL');

        END IF;
END;
/
```

结果：

```
> ENCRYPTED STRING                    : 眨 0 婀苓
> DECRYPTED OUTPUT                    : DM123456
>
> DES ENCRYPTION AND DECRYPTION SUCCESSFUL
```

例 3　使用 MD5 算法对 VARBINARY 类型数据散列，查看其散列值。

```
DECLARE
    RETVAL VARBINARY(100);
    INPUT_STRING VARBINARY(100) := '74696765727469676572746967657274';
    BEGIN
    DBMS_OUTPUT.PUT_LINE('> ========= BEGIN TEST VARBINARY DATA =========');
```

```
        DBMS_OUTPUT.PUT_LINE('> INPUT STRING                    : '
                             || INPUT_STRING);
        RETVAL := DBMS_OBFUSCATION_TOOLKIT.MD5(INPUT_STRING);
        DBMS_OUTPUT.PUT_LINE( '> MD5 STRING OUTPUT              : ' ||
                    RETVAL);
        END;
/
```

结果：

```
> ========= BEGIN TEST VARBINARY DATA =========
> INPUT STRING                    : 7469676572746967657274696765727469676572746967657274
> MD5 STRING OUTPUT               : 831F2AA79221A0F8F330E43A76B53323
```

例 4　使用 DES3GETKEY，生成 VARCHAR2 类型的 DES3 加密密钥，输出加密密钥。

```
DECLARE
    RETVAL VARCHAR2(100);
    INPUT_STRING                    VARCHAR2(100)                   :=
'012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789';
    BEGIN
        DBMS_OUTPUT.PUT_LINE('> ========= BEGIN TEST STRING DATA =========');

      DBMS_OUTPUT.PUT_LINE('> INPUT STRING                  : '
                           || INPUT_STRING);
       DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY(INPUT_STRING, RETVAL);
       DBMS_OUTPUT.PUT_LINE('> DECRYPTED STRING OUTPUT      : ' ||
                    RETVAL);
    END;
/
```

结果：

```
> ========= BEGIN TEST STRING DATA =========
> INPUT STRING                    : 012345678901234567890123456789012345678901234567890123456789012
345678901234567890123456789012345678901234567890123456789
> DECRYPTED STRING OUTPUT         : 4OXDW4FKLE8386JQ    --每次生成密码均不一样
```

附录 1 关键字和保留字

A

ABORT、*ABSOLUTE、*ABSTRACT、ACROSS、ACTION、*ADD、*AUDIT、*ADMIN、AFTER、*ALL、ALLOW_DATETIME、ALLOW_IP、*ALTER、ANALYZE、*AND、*ANY、ARCHIVEDIR、ARCHIVELOG、ARCHIVESTYLE、ARRAY、*ARRAYLEN、*AS、*ASC、*ASSIGN、AT、ATTACH、*AUTHORIZATION、AUTO、AUTOEXTEND、AVG

B

BACKUP、BACKUPDIR、BACKUPINFO、BAKFILE、*BASE、BEFORE、BEGIN、*BETWEEN、*BIGDATEDIFF、*BIGINT、BINARY、BIT、BITMAP、BLOB、BLOCK、*BOOL、BOOLEAN、*BOTH、*BOUNDARY、BRANCH、*BREAK、*BSTRING、BTREE、*BY、*BYTE

C

CACHE、*CALL、CASCADE、CASCADED、*CASE、*CAST、CATALOG、*CATCH、CHAIN、*CHAR、CHARACTER、*CHECK、CIPHER、*CLASS、CLOB、*CLOSE、*CLUSTER、CLUSTERBTR、*COLUMN、*COMMENT、*COMMIT、COMMITTED、*COMMITWORK、COMPILE、COMPRESS、COMPRESSED、*CONNECT、CONNECT_BY_IS_CYCLE、CONNECT_BY_IS_LEAF、*CONNECT_BY_ROOT、CONNECT_IDLE_TIME、*CONST、CONSTANT、*CONSER_OP、*CONSTRAINT、*CONTAINS、CONTEXT、*CONTINUE、*CONVERT、COUNT、*CPU_REF_CALL、*CPU_REF_SESSION、*CREATE、*CROSS、*CRYPTO、CTLFILE、*CUBE、*CURRENT、*CURSOR、CYCLE

D

DANGLING、*DATABASE、DATAFILE、DATE、*DATEADD、*DATEDIFF、*DATEPART、DATETIME、DAY、DBFILE、DEBUG、DEC、*DECIMAL、*DECLARE、*DECODE、*DEFAULT、DEFERRABLE、*DELETE、DELETING、DEREF、*DESC、DETACH、DISABLE、DISCONNECT、*DISKSPACE、*DISTINCT、*DISTRIBUTED、*DO、*DOUBLE、DOWN、*DROP

E

EACH、*ELSE、*ELSEIF、ENABLE、ENCRYPT、ENCRYPTION、*END、EQU、ERROR、ESCAPE、EVENTINFO、*EXCEPT、*EXCEPTION、EXCHANGE、EXCLUSIVE、*EXECUTE、*EXISTS、*EXIT、*EXPLAIN、*EXTERN、EXTERNAL、EXTERNALLY、*EXTRACT

F

FAILED_LOGIN_ATTEMPS、FALSE、*FETCH、FILEGROUP、FILLFACTOR、*FINALLY、*FIRST、*FLOAT、*FOR、FORCE、*FOREIGN、FREQUENCE、*FROM、*FULL、*FUNCTION、FOLLOWING、

G

*GET、GLOBAL、*GOTO、*GRANT、*GROUP *GROUPING

H

HASH、*HAVING、HEXTORAW、HOUR

I

IDENTIFIED、*IDENTITY、IDENTITY_INSERT、*IF、IMAGE、*IMMEDIATE、IN、INCREASE、INCREMENT、*INDEX、INITIAL、INITIALLY、*INNER、INNERID、*INSERT、INSERTING、 INSTEAD、 *INT、 INTEGER、 INTENT、 *INTERNAL、 *INTERSECT、*INTERVAL、*INTO、*IS、ISOLATION

J

*JAVA、*JOIN

K

KEY

L

LABEL、*LAST、*LEAD、*LEFT、LESS、*LEVEL、LEXER、*LIKE、*LIMIT、*LINK、LIST、LOB、LOCAL、LOCK、LOG、LOGFILE、*LOGIN、LOGOUT、LONG、LONGVARBINARY、LONGVARCHAR、*LOOP、*LP_OP、*LT_BINTEGER、*LT_BIGINTEGER、*LT_BITSTRING、*LT_DECIMAL、*LT_GLOBAL_VAR、*LT_IDENTIFIER、*LT_INTEGER、*LP_REAL、*LT_STRING

M

MANUAL、MAP、MATCH、MATCHED、MAX、MAXSIZE、MAXVALUE、*MEMBER、*MEN_SPACE、MERGE、MIN、MINEXTENTS、*MINUS、MINUTE、MINVALUE、MODE、MODIFY、MONEY、MONTH、MOUNT

N

*NATURAL、*NEW、*NEXT、NO、NOARCHIVELOG、NOAUDIT、NOBRANCH、NOCACHE、*NOCYCLE、NOMAXVALUE、NOMINVALUE、NONE、NOORDER、NORMAL、*NOSALT、*NOT、NOT_ALLOW_DATETIME、NOT_ALLOW_IP、NOWAIT、*NULL、、NUMBER、NUMERIC、

O

*OBJECT、*OF、OFF、OFFLINE、OFFSET、OLD、*ON、ONCE、ONLINE、ONLY、*OP_SHIFT_LERT、*OP_SHIFT_RIGHT、 *OPEN、OPTION、*OR、*ORDER、*OUT、OUTER、*OVER 、OVERLAPS、*OVERRIDE

P

*PACKAGE、*PACKAGE_BODY、PAGE、PARTIAL、*PARTITION、PARTITIONS、PASSWORD_GRACE_TIME、PASSWORD_LIFE_TIME、PASSWORD_LOCK_TIME、PASSWORD_POLICY、PASSWORD_REUSE_MAX、PASSWORD_REUSE_TIME、*PENDANT、*PERCENT、PRECEDING、PRECISION、PRESERVE、*PRIMARY、*PRINT、*PRIOR、*PRIVATE、*PRIVILEGES、*PROCEDURE、*PROTECTED、*PT_FOUND、

*PT_ISOPEN、*PT_NOFOUND、*PT_ROWCOUNT、*PT_ROWTYPE、*PT_TYPE、*PUBLIC、*PUT

R

*RAISE、*RANGE、RAWTOHEX、READ、READ_PER_CALL、READ_PER_SESSION、*READONLY、REAL、REBUILD、*RECORD、*REF、*REFERENCES、*REFERENCING、RELATED、*RELATIVE、RENAME、*REPEAT、REPEATABLE、REPLACE、*REPLICATE、RESIZE、RESTORE、RESTRICT、*RETURN、RETURNING、*REVERSE、*REVOKE、*RIGHT、ROLE、*ROLLBACK、ROLLFILE、*ROLLUP、ROOT、*ROW、ROWCOUNT、ROWID、ROWNUM、*ROWS、RULE

S

SALT、*SAVEPOINT、*SBYTE、*SCHEMA、SCOPE、*SEALED、*SECTION、SECOND、*SELECT、*SELSTAR、SEQUENCE、SERERR、SERIALIZABLE、SERVER、SESSION_PER_USER、*SET、*SETS、SHARE、*SHORT、SHUTDOWN、SIBLINGS、*SIZE、*SIZEOF、SMALLINT、SNAPSHOT、*SOME、SOUND、SPLIT、SQL、STANDBY、*START_WITH、STARTUP、STATEMENT、*STATIC、STAT、*STDDEV、STORAGE、STORE、*STRING、*STRUCT、STYLE、SUBSTRING、SUCCESSFUL、SUM、SUSPEND、*SWITCH、SYNC、*SYNONYM、SYS_CONNECT_BY_PATH

T

*TABLE、*TABLESPACE、TEMPORARY、TEXT、THAN、THEN、*THROW、TIES、TIME、TIMER、TIMES、TIMESTAMP、*TIMESTAMPADD、*TIMESTAMPDIFF、TINYINT、*TO、*TOO_MANY_ROWS、*TOP、*TRAIL、TRANSACTION、TRANSACTIONAL、*TRIGGER、TRIGGERS、*TRIM、TRUE、*TRUNCATE、TRUNCSIZE、*TRY、TYPE、*TYPE_BODY、*TYPEOF

U

*UINT、*ULONG、UNBOUNDED、UNCOMMITTED、UNDER、*UNION、*UNIQUE、UNLIMITED、*UNSAFE、*UNTIL、UP、*UPDATE、UPDATING、*USER、*USHORT、*USING

V

VALUE、*VALUES、VARBINARY、VARCHAR、VARCHAR2、VARIANCE、VARYING、*VERIFY、VERTICAL、*VIEW、*VIRTUAL、*VOID、*VOLATILE、VSIZE

W

WEEK、*WHEN、*WHENEVER、*WHERE、*WHILE、*WITH、WORK、WRAPPED、WRITE

Y

YEAR

Z

ZONE

注：以上均为关键字，带*号的为保留字。

1、非保留关键字(可用于任何标识符)

KW_ACROSS|KW_ABORT|KW_ROWID|KW_ACTION|KW_AFTER|KW_ALLOW_DATETIME |KW_ALLOW_IP |KW_ANALYZE |KW_ARCHIVEDIR |KW_ARCHIVELOG |KW_ARCHIVESTYLE |KW_AT |KW_ATTACH |KW_AUTO |KW_AVG |KW_BACKUP |KW_BACKUPDIR |KW_BACKUPINFO |KW_BAKFILE |KW_BEFORE |KW_BIGINT |KW_BINARY |KW_BIT |KW_BITMAP |KW_BLOB |KW_BLOCK |KW_BOOLEAN |KW_CACHE |KW_CASCADE |KW_CASCADED |KW_CATALOG |KW_CHAIN |KW_CHARACTER |KW_CIPHER |KW_CLOB |KW_COMMITED |KW_COMPILE |KW_COMPRESS|KW_COMPRESSED |KW_CONNECT_BY_IS_CYCLE |KW_CONNECT_BY_IS_LEAF |KW_CONNECT_IDLE_TIME|KW_CONTEXT|KW_COUNT|KW_CTLFILE|KW_CYCLE |KW_DANG LING|KW_DATAFILE |KW_DATE|KW_DATETIME|KW_DBFILE|KW_DEBUG|KW_DEC |KW_ DEFERRABLE |KW_DELETING|KW_DEREF |KW_DETACH |KW_DISABLE |KW_DIS CONNECT |KW_ DOWN |KW_EACH |KW_ENABLE |KW_ENCRYPT |KW_ENCRYPTION |KW_ESCAPE |KW_EVENT INFO |KW_EXCLUSIVE|KW_EXTERNAL|KW_EXTERNALLY |KW_FAILED_LOGIN_ATTEMPS |KW_FILEGROUP |KW_FILLFACTOR |KW_FORCE |KW_ FREQUENCE |KW_GLOBAL |KW_HASH |KW_HEXTORAW |KW_IDENTIFIED|KW_ IDENTITY_INSERT |KW_IMAGE |KW_INCREASE |KW_INCREMENT |KW_INITIAL |KW_INITIALLY |KW_INNERID |KW_INSERTING |KW_INSTEAD |KW_INTEGER |KW_INTENT |KW_ISOLATION|KW_KEY |KW_LABEL |KW_LOCAL|KW_LOCK |KW_LOGFILE |KW_LOGOUT|KW_LONG|KW_LONGVARBINARY|KW_LONGVARCHAR |KW_MANUAL|KW_
MAP|KW_MATCH|KW_MATCHED|KW_MAX|KW_MAXSIZE|KW_MAXVALUE |KW_MERGE |KW_MIN|KW_MINEXTENTS |KW_MINVALUE |KW_MODE |KW_MODIFY |KW_MONEY |KW_MOUNT |KW_NO|KW_NOARCHIVELOG |KW_NOAUDIT |KW_NOCACHE |KW_ NOMAXVALUE|KW_NOMINVALUE|KW_NOORDER
|KW_NOT_ALLOW_DATETIME |KW_NOT_
ALLOW_IP|KW_NOWAIT|KW_NUMBER|KW_NUMERIC |KW_OFF |KW_OFFLINE
|KW_OFFSET |KW_ONLINE|KW_ONLY |KW_OPTION |KW_OUTER |KW_OVERLAPS
|KW_CONSTANT
|KW_PAGE|KW_PARTIAL|KW_PASSWORD_GRACE_TIME|KW_PASSWORD_LIFE_TIME
|KW_CPU_PER_CALL |KW_CPU_PER_SESSION |KW_MEM_SPACE
|KW_READ_PER_CALL
|KW_READ_PER_SESSION|KW_PASSWORD_LOCK_TIME|KW_PASSWORD_POLICY
|KW_PASSWORD_REUSE_MAX|KW_PASSWORD_REUSE_TIME|KW_PRECISION
|KW_PRESERVE |KW_RAWTOHEX |KW_READ|KW_REAL|KW_REBUILD
|KW_RELATED|KW_ RENAME |KW_REPEATABLE
|KW_REPLACE|KW_RESTORE|KW_RESTRICT|KW_RETURNING|KW
_ROLE|KW_ROLLFILE
|KW_ROOT|KW_ROWCOUNT|KW_RULE|KW_SALT|KW_SCOPE|KW_
SERERR|KW_SERIALIZABLE |KW_SESSION_PER_USER|KW_SHARE |KW_SHUTDOWN
|KW_SIBLINGS|KW_SMALLINT

|KW_SNAPSHOT|KW_SOUND|KW_SPLIT|KW_SQL|KW_STARTUP
|KW_STATEMENT|KW_STORAGE|KW_STYLE
|KW_SUBSTRING|KW_SUCCESSFUL|KW_SUM |KW_SUSPEND |KW_SYNC
|KW_SYS_CONNECT_BY_PATH |KW_TEMPORARY|KW_TEXT|KW_ THEN |KW_TIES
|KW_TIME |KW_TIMES |KW_TIMESTAMP |KW_TINYINT |KW_TRANSACTION
|KW_TRIGGERS
|KW_TRUNCSIZE|KW_UNCOMMITED|KW_UNDER|KW_UNLIMITED|KW_UP
|KW_UPDATING
|KW_VALUE|KW_VARBINARY|KW_VARCHAR|KW_VARCHAR2|KW_STDDEV
|KW_VARIANCE |KW_VARYING|KW_VSIZE|KW_WORK|KW_WRITE |KW_SEQUENCE
|KW_SERVER|KW_TIMER |KW_WEEK |KW_ONCE|KW_ZONE |KW_VERTICAL|KW_LOG
|KW_NONE|KW_LOB|KW_ERROR |KW_NORMAL |KW_STANDBY
|KW_TRANSACTIONAL |KW_ARRAY|KW_STORE|KW_BRANCH|KW_NOBRANCH|KW_
READONLY|KW_STAT |KW_UN BOUNDED|KW_PRECEDING |KW_FOLLOWING
|KW_AUTOEXTEND |KW_WRAPPED|KW_BTREE

2、别名保留字(不能用于别名)

KW_YEAR | KW_MONTH | KW_DAY | KW_HOUR | KW_MINUTE | KW_SECOND | KW_LEFT | KW_RIGHT | KW_EXCEPT | KW_MINUS | KW_INTERSECT | KW_CROSS | KW_FULL | KW_INNER | KW_JOIN | KW_NATURAL | KW_WHERE

3、模式名保留字(不能用于模式名)

KW_AUTHORIZATION

附录 2 SQL 语法描述说明

< >　　表示一个语法对象。

::=　　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。

|　　或者符，或者符限定的语法选项在实际的语句中只能出现一个。

{ }　　大括号指明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。

[ ]　　中括号指明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，关键字可以为大写也可以为小写。

SQL 语法图的说明

SQL 语法图是用来帮助用户正确地理解和使用 DM SQL 语法的图形。阅读语法图时，请按照从上到下，从左到右的方式，依箭头所指方向进行阅读。

SQL 命令、语法关键字等终结符以全大写方式在长方形框内显示，使用时直接输入这些内容；语法参数或语法子句等非终结符的名称以全小写方式在圆角框内显示；各类标点符号显示在圆圈之中。注：如果小写参数中不带下划线_，则表示是由用户输入的参数，带下划线则表示是还需要进一步解释的子句或语法对象，如果在前面已解释过，则未重复列出。

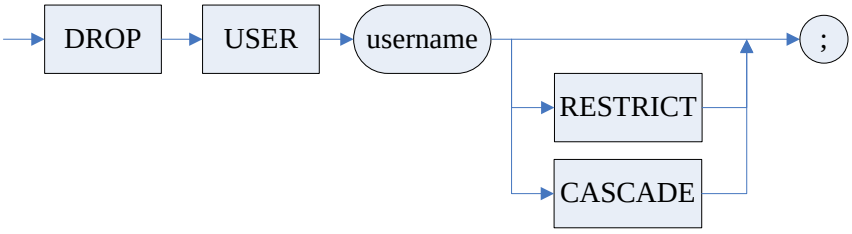
1.必须关键字和参数

必须关键字和参数出现在语法参考图的主干路径上，也就是说，出现在当前阅读的水平线上。例如：

用户删除语句

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

用语法图表示为



这里 DROP、USER、username 和;都是语句必须的。

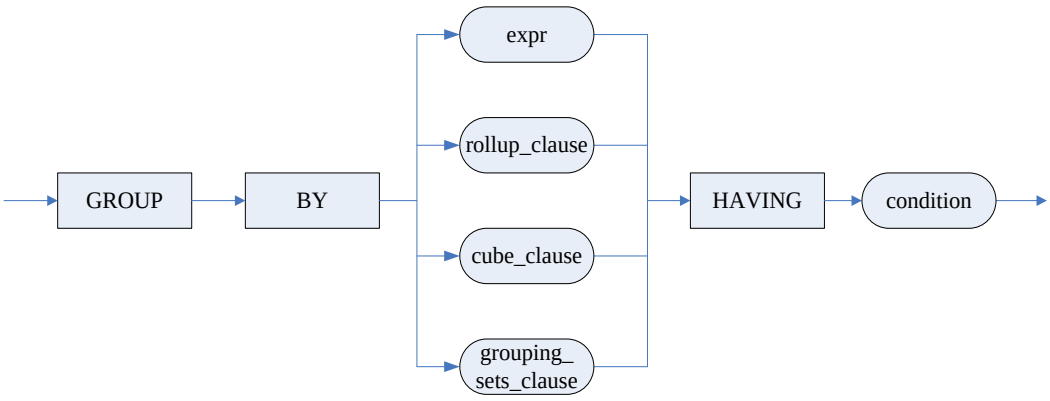
如果多个关键字或参数并行地出现在从主路径延伸出的多条分支路径中，则他们之中有一个是必须的。也就是说，必须选择他们其中的一个，但不需要一定是主路径上的那个。

例如：

```
<GROUP BY 子句> ::= GROUP BY <分组项> | <ROLLUP 项> | <CUBE 项> | <GROUPING SETS 项>
```

用语法图表示

GROUP BY 子句



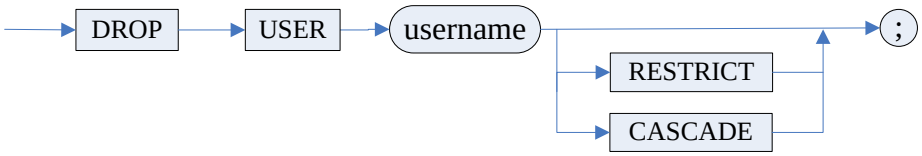
从语法图可以看出，GROUP BY 子句有四种形式，可以任选一种。如果存在 GROUP BY 子句，则必须选择其中一种。

2.可选关键字和参数

如果关键字或参数并行地出现在主路径下方，而主路径是一条直线，则这些关键字和参数是可选的。例如：

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

用语法图表示为:

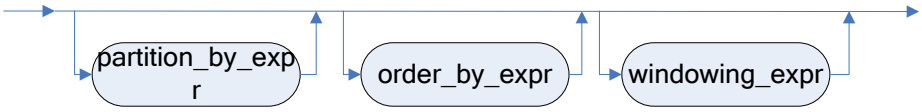


从语法图可以看出，DROP、USER、username 是必须的。而 RESTRICT 和 CASCADE 都是可选的，也可以选择直接通过主路径，而不需走这些并行路径。这些并行路径能且只能选择其中一种。又如：

分析函数的分析子句语法：

```
<分析子句> ::= [<PARTITION BY 项>] [<ORDER BY 项> ] [<窗口子句>]
```

用语法图表示为



其中，partition_by_expr、order_by_expr 和 windowing_expr 参数都是可选的，但如果出现，则出现的顺序不能颠倒。

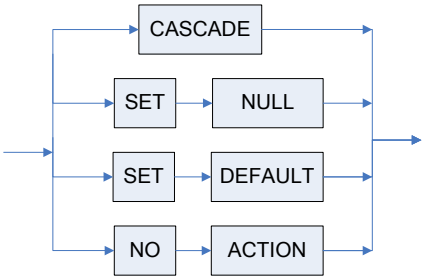
3.多条路径

如果一张语法参考图有一条以上的路径，可以从任意一条路径进行阅读。如果可以选择多个关键字、操作符、参数或者语法子句，这些选项将被并行地列出。例如：

```
<引用动作> ::= [CASCADE] | [SET NULL] | [SET DEFAULT] | [NO ACTION]
```

用语法图表示为

ref_action



从语法图可以看出，引用动作可以选择这四种的任一种。

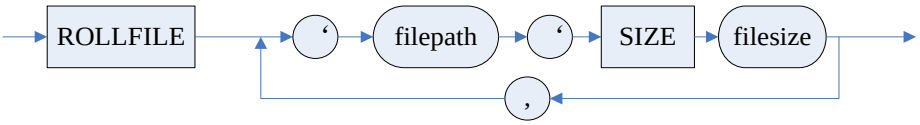
4.循环语法

循环语法表示可以按照需要，使用循环内的语法一次或者多次。例如：

```
<回滚文件子句> ::= ROLLFILE <文件说明子句>,{<文件说明子句>}
<文件说明子句> ::= <文件路径> SIZE <文件大小>
```

用语法图表示为

roll_file_clause



从语法图可以看出，通过逗号隔开，可以重复语法对象 ‘filepath’ SIZE filesize 多个。

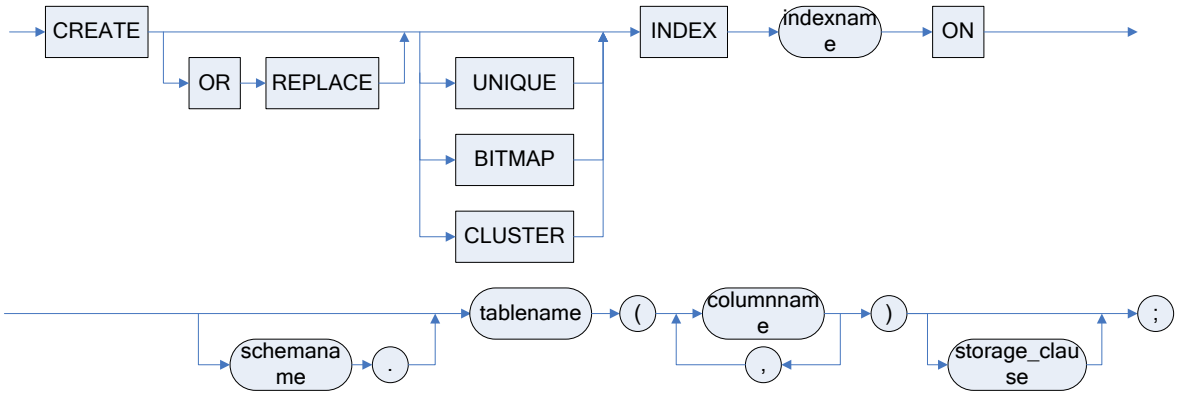
5.多行语法图

由于有些 SQL 语句的语法十分复杂，生成的语法参考图无法完整地显示在一行之内，于是他们将被分行显示。阅读此类图形时，请从上至下，从左至右地进行。例如：

索引定义语句

```
CREATE [OR REPLACE] [UNIQUE | BITMAP | CLUSTER] INDEX <索引名>
ON [<模式名>.]<表名>(<列名>{,<列名>}) [<STORAGE 子句>];
```

用语法图表示为



从语法图可以看出，索引定义的语法被分成了两行显示。

附录 3 系统存储过程和函数

如下为达梦数据库所用到的系统存储过程和函数：

1）系统信息管理

1. SF_GET_PARA_VALUE

定义：

```
int
SF_GET_PARA_VALUE (
  scope              int,
  ini_param_name     varchar(8188)
)
```

功能说明：

返回 dm.ini 文件中的参数值

参数说明：

scope： 取值为 1、2 。 1 表示从 dm.ini 文件中读取；2 表示从内存中读取；ini_param_name：dm.ini 文件中的参数名

返回值：

当前 INI 文件中对应的参数值

举例说明：

获得 dm.ini 文件中 BUFFER 参数值

```
SELECT SF_GET_PARA_VALUE (1, 'BUFFER');
```

2. SP_SET_PARA_VALUE

定义：

```
void
SP_SET_PARA_VALUE (
  scope              int,
  ini_param_name     varchar(8188)
  value              bigint
)
```

功能说明：

设置 dm.ini 文件中的参数值

参数说明：

scope：取值为 1，2 。1 表示 dm.ini 文件和内存参数都已修改，不需要重启服务器；2 表示只修改 dm.ini 文件，服务器重启后生效。

ini_param_name：dm.ini 文件中的参数名。

value：设置的值。

返回值：

无

举例说明：

将 dm.ini 文件中 MULTI_PAGE_GET_NUM 参数值设置为 32

```
SP_SET_PARA_VALUE (1,'MULTI_PAGE_GET_NUM',32);
```

3. SF_GET_PARA_DOUBLE_VALUE

定义：

```
double
SF_GET_PARA_DOUBLE_VALUE (
  scope              int,
  ini_param_name     varchar(256)
)
```

功能说明：

返回dm.ini文件中浮点型的参数值

参数说明：

scope：取值为1，2 。 1表示从dm.ini文件中读取； 2表示从内存中读取；

ini_param_name：dm.ini文件中的参数名。

返回值：

当前INI文件中对应的参数值

举例说明：

获得dm.ini文件中SEL_RATE_EQU参数值

```
SELECT SF_GET_PARA_DOUBLE_VALUE (1, 'SEL_RATE_EQU');
```

4. SP_SET_PARA_DOUBLE_VALUE

定义：

```
void
SP_SET_PARA_DOUBLE_VALUE (
    scope              int,
    ini_param_name     varchar(256),
    value              double
)
```

功能说明：

设置dm.ini文件中浮点型的参数值

参数说明：

scope: 取值为 1，2。1 表示 dm.ini 文件和内存参数都已修改，不需要重启服务器；2 表示只可修改 dm.ini 文件,服务器重启后生效。

ini_param_name：dm.ini文件中的参数名。

value：设置的值。

返回值：

无

举例说明：

将dm.ini文件中SEL_RATE_EQU参数值设置为0.3

```
SP_SET_PARA_DOUBLE_VALUE(1, 'SEL_RATE_EQU', 0.3);
```

5. SF_GET_PARA_STRING_VALUE

定义：

```
char*
SF_GET_PARA_STRING_VALUE (
    scope              int,
    ini_param_name     varchar(256)
)
```

功能说明：

返回 dm.ini 文件中字符串类型的参数值

参数说明：

scope： 取值为1，2 。 1表示从dm.ini文件中读取；2表示从内存中读取；ini_param_name：dm.ini文件中的参数名

返回值：

当前 INI 文件中对应的参数值

举例说明：

获得dm.ini文件中TEMP_PATH参数值

```
SELECT SF_GET_PARA_STRING_VALUE (1, 'TEMP_PATH');
```

6. SP_SET_PARA_STRING_VALUE

定义：

```
void
SP_SET_PARA_ STRING_VALUE (
  scope              int,
  ini_param_name     varchar(256)
  value              varchar(8187)
)
```

功能说明：

设置 dm.ini 文件中的字符串型参数值

参数说明：

scope：取值为 1，2 。1 表示 dm.ini 文件和内存参数都已修改，不需要重启服务器；2 表示只修改 dm.ini 文件，服务器重启后生效。

ini_param_name：dm.ini 文件中的参数名。

value：设置的字符串的值。

返回值：

无

举例说明：

将 dm.ini 文件中 SQL_TRACE_MAS 参数值设置为 1

```
SP_SET_PARA_STRING_VALUE(1, 'SQL_TRACE_MASK','1');
```

7. SP_SET_SESSION_PARA_VALUE

定义：

```
int
SP_SET_SESSION_PARA_VALUE (
      paraname     varchar(8187),
      value         bigint
)
```

功能说明：

设置会话级 INI 参数的值

参数说明：

paraname： 会话级 INI 参数的参数名。

value：要设置的新值

返回值：

是否成功

举例说明：

设置会话级 ini 参数 BDTA_SIZE 的值为 2000：

```
SP_SET_SESSION_PARA_VALUE ( 'BDTA_SIZE',  2000);
```

8. SP_RESET_SESSION_PARA_VALUE

定义：

```
int
SP_RESET_SESSION_PARA_VALUE (
    paraname    varchar(8187)    )
```

功能说明：

重置会话级 INI 参数的值，使得参数的值和系统级一致。

参数说明：

paraname：会话级 INI 参数的参数名。

返回值：

是否成功

举例说明：

获取会话级 INI 参数 BDTA_SIZE 的值：

```
SP_RESET_SESSION_PARA_VALUE ( 'BDTA_SIZE');
```

9. SF_GET_SESSION_PARA_VALUE

定义：

```
int
SF_GET_SESSION_PARA_VALUE (
    paraname    varchar(8187)    )
```

功能说明：

获得会话级 INI 参数的值。

参数说明：

paraname：会话级 INI 参数的参数名。

返回值：

会话级 INI 参数的值

举例说明：

获取会话级 INI 参数 BDTA_SIZE 的值：

```
SELECT SF_GET_SESSION_PARA_VALUE ( 'BDTA_SIZE');
```

10. SF_GET_CASE_SENSITIVE_FLAG

定义：

```
INT
SF_GET_CASE_SENSITIVE_FLAG()
```

功能说明：

返回大小写敏感信息

参数说明：

无

返回值：

1：敏感

0：不敏感

举例说明：

获得大小写敏感信息

```
SELECT SF_GET_CASE_SENSITIVE_FLAG();
```

11. SF_GET_EXTENT_SIZE

定义：

INT

SF_GET_EXTENT_SIZE()

功能说明：

返回簇大小

参数说明：

无

返回值：

系统建库时指定的簇大小

举例说明：

获得系统建库时指定的簇大小

```
SELECT SF_GET_EXTENT_SIZE ();
```

12. SF_GET_PAGE_SIZE

定义：

INT

SF_GET_PAGE_SIZE()

功能说明：

返回页大小

参数说明：

无

返回值：

系统建库时指定的页大小

举例说明：

获得系统建库时指定的页大小

```
SELECT SF_GET_PAGE_SIZE ();
```

补充说明：

获得系统建库时指定的页大小也可使用：

```
SELECT PAGE();
```

13. SF_GET_UNICODE_FLAG

定义：

INT

SF_GET_UNICODE_FLAG()

功能说明：

返回 UNICODE 标志

参数说明：

无

返回值：

系统建库时指定的 UNICODE 标志，0 表示指定为非 UNICODE 编码，1 表示指定为 UNICODE 编码。

举例说明：

获得系统建库时指定的 UNICODE 标志

```
SELECT SF_GET_UNICODE_FLAG ();
```

14. SF_GET_SGUID

定义：

INT

SF_GET_SGUID ()

功能说明：

返回数据库唯一标志 sguid

参数说明：

无

返回值：

返回数据库唯一标志 sguid

举例说明：

获取数据库唯一标志 sguid

```
SELECT SF_GET_SGUID();
```

15. SF_SET_SQL_LOG

定义：

```
INT
SF_SET_SQL_LOG (
    svrlog    int,
    svrmsk    varchar(1000)
)
```

功能说明：

设置服务器日志相关 INI 参数

参数说明：

svrlog：INI 参数 SVR_LOG 的设置值

svrmsk：INI 参数 SQL_TRACE_MASK 的设置值

返回值：

是否成功

举例说明：

设置服务器日志相关 INI 参数

```
SELECT   SF_SET_SQL_LOG(1, '3:5:7');
```

16. UID

定义：

```
INT
UID ()
```

功能说明：

返回当前用户 ID

参数说明：

无

返回值：

返回当前用户 ID

举例说明：

返回当前用户 ID

```
SELECT UID();
```

17. USER

定义：

```
CHAR*
USER ()
```

功能说明：

返回当前用户名

参数说明：

无

返回值：

返回当前用户名

举例说明：

返回当前用户名

```
SELECT USER();
```

18. CUR_DATABASE

定义：

CHAR*
CUR_DATABASE ()

功能说明：

返回数据库名

参数说明：

无

返回值：

返回数据库名

举例说明：

获取数据库名

```
SELECT CUR_DATABASE();
```

19. VSIZE

定义：

INT
VSIZE(n)

功能说明：

返回 n 的核心内部表示的字节数。如果 n 为 NULL，则返回 NULL。

参数说明：

n：待求字节数的参数，可以为任意数据类型

返回值：

n 占用的字节数

举例说明：

```
SELECT VSIZE(256);
```

查询结果：4 /*整数类型*/

```
SELECT VSIZE('数据库');
```

查询结果：6 /*中文字符*/

20. SP_RECLAIM_TS_FREE_EXTENTS

定义：

```
SP_RECLAIM_TS_FREE_EXTENTS (
    tsname          varchar(128)
)
```

功能说明：

重组表空间空闲簇

参数说明：

tsname：表空间名

返回值：

无

举例说明：

重组表空间空闲簇

```
CALL SP_RECLAIM_TS_FREE_EXTENTS('SYSTEM');
```

21. SP_CLEAR_PLAN_CACHE

定义：

SP_CLEAR_PLAN_CACHE()

功能说明：

清空执行缓存信息。

参数说明：

无

返回值：

无

举例说明：

清空执行缓存信息

```
CALL SP_CLEAR_PLAN_CACHE();
```

22. SF_CHECK_USER_TABLE_PRIV

定义：

```
int
SF_CHECK_USER_TABLE_PRIV(
   schema_name    varchar(128),
   table_name     varchar(128),
   user_name      varchar(128),
   priv_code      int
)
```

功能说明：

返回用户对表是否具有某种权限

参数说明：

schema_name： 模式名；

table_name：表名；

user_name：用户名；

priv_code：权限代码， 0=SELECT, 1=INSERT, 2=DELETE, 3=UPDATE，4=REFERENCE

返回值：

0：用户不具备相应权限；1：用户具备相应权限

举例说明：

获得用户 SYSDBA 对表 SYS.SYSOBJECTS 的查询权限

```
SELECT SF_CHECK_USER_TABLE_PRIV ('SYS', 'SYSOBJECTS', 'SYSDBA', 0);
```

23. SF_CHECK_USER_TABLE_COL_PRIV

定义：

```
int
SF_CHECK_USER_TABLE_COL_PRIV(
   schema_name    varchar(128),
   table_name     varchar(128),
```

```
    col_name        varchar(128),
    user_name       varchar(128),
    priv_code       int
)
```

功能说明：

返回用户对表中某列是否具有某种权限

参数说明：

schema_name：模式名；

table_name：表名；

col_name：列名；

user_name：用户名；

priv_code：权限代码，0=SELECT，1=INSERT，2=DELETE，3=UPDATE，4=REFERENCE

返回值：

0：用户不具备相应权限；1：用户具备相应权限

举例说明：

获得用户 SYSDBA 对表 SYS.SYSOBJECTS 的 ID 列的查询权限

```
SELECT SF_CHECK_USER_TABLE_COL_PRIV ('SYS', 'SYSOBJECTS', 'ID' , 'SYSDBA', 0);
```

24. SF_RESET_TEMP_TS

定义：

```
int
SF_RESET_TEMP_TS ()
```

功能说明：

在仅有一个活动会话时回收临时表空间中文件所占用的磁盘空间

参数说明：

无

返回值：

实际回收的磁盘空间大小（以M为单位）或者是没有回收原因的错误码

举例说明：

向服务器请求回收临时表空间文件所占用的磁盘空间。

```
SELECT SF_RESET_TEMP_TS();
```

25. CUR_TICK_TIME

定义：

```
varchar
CUR_TICK_TIME ()
```

功能说明：

获取系统当前时钟记数

参数说明：

无

返回值：

时钟记数的字符串

举例说明：

获取系统当前时钟记数。

```
SELECT CUR_TICK_TIME();
```

26. SP_SET_LONG_TIME

定义：

SP_SET_LONG_TIME (
int long_exec_time)

功能说明：

设置 V$LOG_EXEC_SQLS_TIME 动态视图中监控 SQL 语句的最短执行时间，以毫秒为单位，有效范围 50~3600000。仅 INI 参数 ENABLE_MONITOR 值大于 0 时设置有效。

参数说明：

无

返回值：

无

举例说明：

监控执行时间超过 5 秒的 SQL 语句。

```
CALL SP_SET_LONG_TIME(5000);
```

27. SF_GET_LONG_TIME

定义：

int
SF_GET_LONG_TIME ()

功能说明：

返回 V$LOG_EXEC_SQLS_TIME 动态视图中监控的最短执行时间，以毫秒为单位。

参数说明：

无

返回值：

V$LOG_EXEC_SQLS_TIME所监控的最短执行时间。

举例说明：

查看 V$LONG_EXEC_SQLS_TIME 监控的最短执行时间。

```
SELECT SF_GET_LONG_TIME();
```

2）备份恢复管理

保证系统处于归档模式，对示例库 BOOKSHOP 执行联机备份，后面的每个函数的示例都是基于该备份，执行的 SQL 如下：

```
BACKUP DATABASE TO BOOKSHOP_BAK bakfile 'D:\BOOKSHOP_BAK.bak' backupinfo 'BOOKSHOP BAK INFO TEST.' maxsize 1024 identified by BOOKSHOP_BAK_PWD COMPRESSED;
```

1. SF_BAK_GET_BAK_DIR

定义：

CHAR*
SF_BAK_GET_BAK_DIR()

功能说明：

取得系统备份路径，可通过 INI 参数 BAK_PATH 指定

返回值：

系统备份路径

举例说明：

```
SELECT SF_BAK_GET_BAK_DIR();
```

查询结果：

返回安装时或 INI 参数 BAK_PATH 指定的备份路径

2. SF_BAK_GET_NAME

定义：

```
CHAR*
SF_BAK_GET_NAME(
    path    varchar(256)
)
```

功能说明：

取得一个备份的备份名

参数说明：

path：备份的完整路径

返回值：

备份名

举例说明：

```
SELECT SF_BAK_GET_NAME('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
BOOKSHOP_BAK
```

3. SF_BAK_GET_DESC

定义：

```
CHAR*
SF_BAK_GET_DESC(
    path        varchar(256)
)
```

功能说明：

取得一个备份的描述信息

参数说明：

path：备份的完整路径

返回值：

备份描述信息

举例说明：

```
SELECT SF_BAK_GET_DESC('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
BOOKSHOP BAK INFO TEST.
```

4. SF_BAK_GET_TYPE

定义：

```
INT
SF_BAK_GET_TYPE(
    path        varchar(256)
)
```

功能说明：

取得一个备份类型：完全备份，增量备份，B 树备份

参数说明：

path：备份的完整路径

返回值：

0：完全备份；1：增量备份；2：B 树备份。

举例说明：

```
SELECT SF_BAK_GET_TYPE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
0
```

5. SF_ BAK_GET_LEVEL

定义：

```
INT
SF_BAK_GET_LEVEL(
    path        varchar(256)
)
```

功能说明：

取得一个备份级别

参数说明：

path：备份的完整路径

返回值：

0：联机备份；1：脱机备份。

举例说明：

```
SELECT SF_BAK_GET_LEVEL('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
0
```

6. SF_BAK_GET_BASE_NAME

定义：

```
CHAR*
SF_BAK_GET_BASE_NAME(
    path     varchar(256)
)
```

功能说明：

取得一个备份的基础备份名

参数说明：

path：备份的完整路径

返回值：

基础备份名，如果没有基备份则返回空串

举例说明：

```
SELECT SF_BAK_GET_BASE_NAME('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
结果为空串
```

7. SF_BAK_GET_TIME

定义：

```
CHAR*
SF_BAK_GET_TIME(
    path     varchar(256)
)
```

功能说明：

取得一个备份的备份时间

参数说明：

path：备份的完整路径

返回值：

备份时间，与备份执行时间相关

举例说明：

```
SELECT SF_BAK_GET_TIME('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
2011-09-21 09:10:40
```

8. SF_BAK_GET_PAGE_SIZE

定义：

```
INT
SF_BAK_GET_PAGE_SIZE(
    path    varchar(256)
)
```

功能说明：

取得一个备份库的数据文件页大小

参数说明：

path：备份的完整路径

返回值：

备份库的数据文件页大小，与建库参数相关

举例说明：

```
SELECT SF_BAK_GET_PAGE_SIZE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
8192
```

9. SF_BAK_GET_LOG_PAGE_SIZE

定义：

```
INT
SF_BAK_GET_LOG_PAGE_SIZE(
    path    varchar(256)
)
```

功能说明：

取得一个备份库的日志文件页大小

参数说明：

path：备份的完整路径

返回值：

备份库的日志文件页大小

举例说明：

```
SELECT SF_BAK_GET_LOG_PAGE_SIZE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
512
```

10. SF_BAK_GET_EXTENT_SIZE

定义：

```
INT
SF_BAK_GET_EXTENT_SIZE(
    path    varchar(256)
```

)

功能说明：

取得一个备份库的簇大小

参数说明：

path：备份的完整路径

返回值：

备份库的簇大小，与建库参数相关

举例说明：

```
SELECT SF_BAK_GET_EXTENT_SIZE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
16
```

11. SF_BAK_GET_CASE_SENSITIVE

定义：

```
INT
SF_BAK_GET_CASE_SENSITIVE (
    path    varchar(256)
)
```

功能说明：

取得一个备份库大小写敏感信息

参数说明：

path：备份的完整路径

返回值：

备份库大小写敏感信息，与建库参数相关。0：大小写不敏感，1：大小写敏感。

举例说明：

```
SELECT SF_BAK_GET_CASE_SENSITIVE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
1
```

12. SF_BAK_GET_UNICODE_FLAG

定义：

```
INT
SF_BAK_GET_UNICODE_FLAG (
    path    varchar(256)
)
```

功能说明：

取得一个备份库 UNICODE 编码信息

参数说明：

path：备份的完整路径

返回值：

备份库 UNICODE 编码信息，与建库参数相关

0：非 UNICODE 编码，1：UNICODE 编码

举例说明：

```
SELECT SF_BAK_GET_UNICODE_FLAG('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
0
```

13. SF_BAK_GET_DB_VERSION

定义：

```
INT
SF_BAK_GET_DB_VERSION(
    path    varchar(256)
)
```

功能说明：

取得一个备份库的数据文件版本信息

参数说明：

path：备份的完整路径

返回值：

备份库的数据文件版本信息，与 DM 服务器版本相关

举例说明：

```
SELECT SF_BAK_GET_DB_VERSION('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
458753
```

14. SF_BAK_GET_GLOBAL_VERSION

定义：

```
CHAR*
SF_BAK_GET_GLOBAL_VERSION(
    path    varchar(256)
)
```

功能说明：

取得一个备份库的服务器版本信息

参数说明：

path：备份的完整路径

返回值：

备份库的服务器版本信息，与 DM 服务器版本相关

举例说明：

```
SELECT SF_BAK_GET_GLOBAL_VERSION('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
V7.0.1.28-Build(2011.09.14)
```

15. SF_BAK_GET_ENABLE_POLICY

定义：

```
INT
SF_BAK_GET_ENABLE_POLICY(
    path    varchar(256)
)
```

功能说明：

获得备份库中是否允许策略信息

参数说明：

path：备份的完整路径

返回值：

备份库中是否允许策略信息，与 DM 服务器是否是安全版本相关。0：不允许；1：允许。

举例说明：

```
SELECT SF_BAK_GET_ENABLE_POLICY('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
0
```

16. SF_BAK_GET_ARCH_FLAG

定义：

```
INT
SF_BAK_GET_ARCH_FLAG(
    path    varchar(256)
)
```

功能说明：

获得备份库是否归档信息

参数说明：

path：备份的完整路径

返回值：

备份库中是否归档信息。0：不归档，1：归档

举例说明：

```
SELECT SF_BAK_GET_ARCH_FLAG('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
1
```

17. SF_BAK_GET_ENCRYPT_TYPE

定义：

```
INT
SF_BAK_GET_ENCRYPT_TYPE (
    path    varchar(256)
)
```

功能说明：

获得备份库加密类型

参数说明：

path：备份的完整路径

返回值：

备份库的加密类型，与建库参数相关。0：不加密；1：简单加密；2：复杂加密

举例说明：

```
SELECT SF_BAK_GET_ENCRYPT_TYPE ('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
0
```

18. SF_BAK_GET_COMPRESSED

定义：

```
INT
SF_BAK_GET_COMPRESSED(
    path    varchar(256)
)
```

功能说明：

获得备份库是否压缩信息

参数说明：

path：备份的完整路径

返回值：

备份库是否压缩信息。0：不压缩；1：压缩

举例说明：

```
SELECT SF_BAK_GET_COMPRESSED('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
1
```

19. SF_BAK_GET_RANGE

定义：

```
INT
SF_BAK_GET_RANGE(
    path    varchar(256)
)
```

功能说明：

获得备份所属的备份类别，包括数据库级，表空间级和表级

参数说明：

path：备份的完整路径

返回值：

备份类别。1：库级备份；2：表空间级备份；3：表级备份

举例说明：

```
SELECT SF_BAK_GET_RANGE('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
1
```

20. SF_BAK_GET_OBJ_NAME

定义：

```
CHAR*
SF_BAK_GET_OBJ_NAME(
    path    varchar(256)
)
```

功能说明：

获得备份对象的名字

参数说明：

path：备份的完整路径

返回值：

备份对象的名字，库级备份返回空，表空间备份返回表空间名，表级备份返回表名。

举例说明：

```
SELECT SF_BAK_GET_OBJ_NAME('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
NULL
```

21. SF_BAK_GET_SCHEMA_NAME

定义：

```
CHAR*
SF_BAK_GET_SCHEMA_NAME(
    path    varchar(256)
)
```

功能说明：

获得备份表对象所属的模式名，仅对表备份有效

参数说明：

path：备份的完整路径

返回值：

备份表对象所属的模式名

举例说明：

```
SELECT SF_BAK_GET_SCHEMA_NAME('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
NULL
```

22. SF_BAK_LST_INIT

定义：

```
INT
SF_BAK_LST_INIT(VOID)
```

功能说明：

初始化备份链表

返回值：

1：成功； 0：失败。

举例说明：

```
SELECT SF_BAK_LST_INIT();
```

查询结果：

```
1
```

23. SF_BAK_LST_DEINIT

定义：

```
INT
SF_BAK_LST_DEINIT(VOID)
```

功能说明：

销毁备份链表

返回值：

1：成功；0：失败

备注：

调用此接口，需要先调用 SF_BAK_LST_INIT 初始化链表，在完成备份链表操作后应调用此接口销毁备份链表。

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_DEINIT();
```

查询结果：

```
1
```

24. SF_BAK_LST_SET_N_PATH

定义：

```
INT
SF_BAK_LST_SET_N_PATH(
    int    npath
)
```

功能说明：

设置构造备份链表的搜索路径数目

参数说明：

npath，设置路径数，npath 取值范围 1~16

返回值：

1：成功，0：失败

备注：

调用此接口，需要先调用 SF_BAK_LST_INIT 初始化链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
```

查询结果：

```
1
```

25. SF_BAK_LST_SET_PATH

定义：

```
INT
SF_BAK_LST_SET_PATH(
    int nth,
    char* path
)
```

功能说明：

设置第 nth 条路径

参数说明：

nth，第 nth 条路径(从 0 开始)

path，设置路径

返回值：

1：成功；0：失败

备注：

调用此接口，需要先调用 SF_BAK_LST_INIT 初始化链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
```

查询结果：

```
1
```

26. SF_BAK_LST_COLLECT_ALL

定义：

```
INT
SF_BAK_LST_COLLECT_ALL()
```

功能说明：

构造备份链表

返回值：

1：成功；0：失败

备注：

调用此接口需先调用 SF_BAK_LST_INIT 初始化链表，并可以选择调用 SF_BAK_LST_SET_N_PATH 与 SF_BAK_LST_SET_PATH 设置备份搜索路径。当不需要备份链表时，调用 SF_BAK_LST_DEINIT 销毁链表。

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
```

查询结果：

```
1
```

27. SF_BAK_LST_COLLECT

定义：

```
INT
SF_BAK_LST_COLLECT(
char*      objname，
char*      schema_name,
int        range
)
```

功能说明：

搜集指定对象名的备份链表

参数说明：

objname，对象名，可以是数据库名，表空间名，表名

shema_name, 模式名，对表名有效

range，　　级别：1 表示数据库级；2 表示表空间级；3 表示表级

返回值：

1：成功；0：失败

备注：

调用此接口需先调用 SF_BAK_LST_INIT 初始化链表，并可以选择调用 SF_BAK_LST_SET_N_PATH 与 SF_BAK_LST_SET_PATH 设置备份搜索路径。当不需要备份链表时，调用 SF_BAK_LST_DEINIT 销毁链表。

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT('BOOKSHOP', '', 1);
```

查询结果：

```
1
```

28. SF_GET_BAK_LST_NUM

定义：

```
INT
SF_ GET_BAK_LST_NUM(
      dbname     varchar(128)
)
```

功能说明：

获得备份链表中指定数据库名的备份数目

参数说明：

dbname：数据库名

返回值：

备份链表中指定数据库名的备份数目

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_GET_BAK_LST_NUM('BOOKSHOP');
```

查询结果：

```
1
```

29. SF_GET_BAK_BY_ID

定义：

```
CHAR*
SF_GET_BAK_BY_ID(
    id          int,
    dbname      varchar(128)
)
```

功能说明：

获得备份链表中指定数据库名的第 ID 个备份。

参数说明：

id：同名备份在备份链表中的相对位置编号,id 从 1 开始编号
dbname：数据库名

返回值：

指定备份的路径

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'d:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_GET_BAK_BY_ID(1, 'DAMENG');
```

查询结果：

```
D:\v_bak.bak
```

30. SF_GET_BAK_BY_NAME

定义：

```
CHAR*
SF_GET_BAK_BY_ NAME (
    NAME      VARCHAR(128),
    OBJNAME VARCHAR(128),
    SCHNAME VARCHAR(128),
    RANGE   INT
)
```

功能说明：

通过备份名称获得备份路径。

参数说明：

name：备份名

objname：表空间名、表名或数据库名

schname：模式名，对表名有效

range：级别。1：库级；2：表空间级；3：表级

返回值：

备份的路径

备注：

调用此接口需先调用 SF_BAK_LST_INIT()初始化备份链表。

举例说明：

```
SF_BAK_LST_INIT();
SELECT SF_GET_BAK_BY_NAME ('DB01_PRO_01190_BAK01',",",1);
```

31. SF_GET_BAK_OBJ_LST_NUM

定义：

```
INT
SF_BAK_GET_BAK_OBJ_LIST_NUM(
    objname     varchar(128),
    schname     varchar(128),
    int         range
)
```

功能说明：

获得备份链表中指定对象名的备份数目

参数说明：

objname：表空间名或表名

schname: 模式名，对表名有效

range：级别：2：表空间级；3：表级

返回值：

备份链表中指定对象名的备份数目

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_GET_BAK_OBJ_LST_NUM('T1', 'SYSDBA', 3);
```

查询结果：

```
1
```

32. SF_DB_HAS_BACKUP

定义：

```
INT
SF_DB_HAS_BACKUP(
char*       dbname
)
```

功能说明：

判断指定数据库是否有备份信息

参数说明：

dbname, 数据库名

返回值：

0：表示没有；1：表示有

备注：

调用此接口需先调用 SF_BAK_LST_INIT 初始化链表，并可以选择调用 SF_BAK_LST_SET_N_PATH 与 SF_BAK_LST_SET_PATH 设置备份搜索路径。当不需要备份链表时，调用 SF_BAK_LST_DEINIT 销毁链表。

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_DB_HAS_BACKUP('BOOKSHOP');
```

查询结果：

```
1
```

33. SF_BAK_LST_GET_FIRST

定义：

INT
SF_BAK_LST_GET_FIRST()

功能说明：

将备份链表中当前备份定位至第一个备份

返回值：

0：失败；1：成功

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_FIRST();
```

查询结果：

```
1
```

34. SF_BAK_LST_GET_NEXT

定义：

INT
SF_BAK_LST_GET_NEXT()

功能说明：

获取备份链表中的下一个备份

返回值：

0：失败；1：成功

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_NEXT();
```

查询结果：

```
1
```

35. SF_BAK_GET_CUR_PATH

定义：

CHAR*
SF_BAK_LST_GET_CUR_PATH()

功能说明：

获取备份链表中当前备份的路径信息

返回值：

当前备份的路径信息

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_FIRST();
SELECT SF_BAK_GET_CUR_PATH();
```

查询结果：

```
D:\BOOKSHOP_BAK.bak
```

36. SF_BAK_GET_CUR_NAME

定义：

CHAR*
SF_BAK_LST_GET_CUR_NAME()

功能说明：

获取备份链表中当前备份的备份名

返回值：

当前备份的备份名

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
```

```
SELECT SF_BAK_LST_GET_FIRST();
SELECT SF_BAK_GET_CUR_NAME();
```

查询结果：

```
BOOKSHOP_BAK
```

37. SF_BAK_GET_CUR_DB_NAME

定义：

CHAR*
SF_BAK_LST_GET_CUR_DB_NAME()

功能说明：

获取备份链表中当前备份的数据库名

返回值：

当前备份的数据库名

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_FIRST();
SELECT SF_BAK_GET_CUR_DB_NAME();
```

查询结果：

```
bookshop
```

38. SF_BAK_GET_FILE_NUM

定义：

INT
SF_BAK_GET_FILE_NUM()

功能说明：

获取备份链表中当前备份的文件数目

返回值：

当前备份的文件数目

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_FIRST();
SELECT SF_BAK_GET_FILE_NUM();
```

查询结果：

```
1
```

39. SF_BAK_GET_FILE

定义：

CHAR*
SF_BAK_GET_FILE(
    int nth
)

功能说明：

获取当前备份的第 nth 个文件路径

参数说明：

nth, 第 nth 个文件(nth 从 1 开始)

返回值：

当前备份第 nth 个文件的路径信息

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL
或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_BAK_LST_GET_FIRST();
SELECT SF_BAK_GET_FILE(1);
```

查询结果：

```
D:\BOOKSHOP_BAK.bak
```

40. SF_BAK_GET_DB_FILE_TYPE

定义：

CHAR*
SF_BAK_GET_DB_FILE_TYPE(
    int nth,
    char* backuppath
)

功能说明：

获取指定备份第 nth 个数据文件的文件类型

参数说明：

nth, 第 nth 个数据文件
backuppath, 备份文件所在路径

返回值：

文件类型

举例说明：

```
SELECT SF_BAK_GET_DB_FILE_TYPE(1,'D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
DATAFILE
```

41. SF_BAK_GET_DB_FILE_NUM

定义：

INT
SF_BAK_GET_DB_FILE_NUM(
    char* backuppath

)

功能说明：

获取指定备份库包含的数据文件数目

参数说明：

backuppath，备份文件所在路径

返回值：

备份库包含的数据文件数目

举例说明：

```
SELECT SF_BAK_GET_DB_FILE_NUM('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
4
```

42. SF_BAK_GET_DB_FILE_NAME

定义：

```
CHAR*
SF_BAK_GET_DB_FILE_NAME(
    int         nth,
    char*       backuppath
)
```

功能说明：

获取指定备份第 nth 个数据文件文件名

参数说明：

nth，第 nth 个数据文件

backuppath，备份文件所在路径

返回值：

文件名

举例说明：

```
SELECT SF_BAK_GET_DB_FILE_NAME(1,'D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
d:\dm7data\bookshop\SYSTEM.DBF
```

43. SF_BAK_FILE_CHECK

定义：

```
INT
SF_BAK_FILE_CHECK(
    char*       backuppath
)
```

功能说明：

对备份文件进行校验

参数说明：

backuppath，文件所在路径

返回值：

0：表示校验失败；1 表示校验成功

举例说明：

```
SELECT SF_BAK_FILE_CHECK('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
1
```

44. SF_DEL_BAK

定义：

```
INT
SF_DEL_BAK (
    char*     objname,
    char*     schema_name,
    char*     bakname,
    int       range
)
```

功能说明：

删除指定对象名和备份名的备份

参数说明：

objname，对象名，可以是数据库名，表空间名，表名

schema_name，模式名，对表名有效

bakname，备份名

range，级别，1：数据库级，2：表空间级，3：表级

返回值：

0：删除失败，1：删除成功

备注：

调用此接口需先调用 SF_BAK_LST_COLLECT_ALL 或 SF_BAK_LST_COLLECT 构造备份链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_N_PATH(1);
SELECT SF_BAK_LST_SET_PATH(0,'D:\');
SELECT SF_BAK_LST_COLLECT_ALL();
SELECT SF_DEL_BAK('BOOKSHOP', '', 'BOOKSHOP_BAK', 1);
```

查询结果：

```
1
```

45. SF_BAK_LST_SET_PARALLEL_DIR

定义：

```
INT
SF_BAK_LST_SET_PARALLEL_DIR (
    schar*     parallel_dir
)
```

功能说明：

设置搜集并行备份映射文件的目录

参数说明：

parallel_dir：指定并行备份映射文件存储目录

备注：

调用此接口，需要先调用 SF_BAK_LST_INIT 初始化链表

举例说明：

```
SELECT SF_BAK_LST_INIT();
SELECT SF_BAK_LST_SET_PARALLEL_DIR ('D:\ ');
```

查询结果：

```
1
```

46. SP_GET_DB_BAK_INFO

定义：

SP_GET_DB_BAK_INFO(
char* dbname,
char* backup_dir,
char* parallel_dir
)

功能说明：

获取指定目录下，指定数据库名备份的备份信息

参数说明：

dbname： 数据库名,
backup_dir：搜集备份的目录
parallel_dir：指定并行备份映射文件存储目录

返回信息：

返回备份信息的结果集

备注：

结果集格式信息如下

字段名	字段类型	字段含义
ID	INT	记录号
NAME	VARCHAR(128)	备份名
DESC	VARCHAR(256)	备份描述信息
TYPE	INT	备份类型：完全备份，增量备份，B 树备份
RANGE	INT	备份级别：库级，表空间级，表级
SCHEMA_NAME	VARCHAR(128)	对表备份有效
OBJ_NAME	VARCHAR(128)	表名或表空间名
BASE	VARCHAR(128)	基备份名
TIME	VARCHAR(128)	备份时间
PATH	VARCHAR(256)	备份路径
PAGE_SIZE	INT	备份库页大小
EXTENT_SIZE	INT	备份库簇大小
CASE_SENSITIVE	INT	备份库大小写敏感信息
LOG_PAGE_SIZE	INT	备份库日志页大小
UNICODE_FLAG	INT	备份库是否采用 UNICODE 编码
DB_VERSION	INT	备份库版本信息
GLOBAL_VERSION	VARCHAR(128)	备份数据库服务器版本信息
ENABLE_POLICY	INT	备份库是否允许策略
ARCH_FLAG	INT	备份库归档标志
ENCRYPT_TYPE	INT	备份库加密类型
IS_COMPRESSED	INT	备份库是否压缩存储

举例说明：

```
SP_GET_DB_BAK_INFO('BOOKSHOP','D:\','D:\');
```

查询结果：

```
1
```

47. SP_GET_TS_BAK_INFO

定义：

SP_GET_TS_BAK_INFO(

char* tsname,
char* backup_dir
)

功能说明：

获取指定目录下，指定表空间名所有备份的备份信息

参数说明：

tsname：表空间名，若设置为‘’，则搜集目录下所有的表空间备份

backup_dir：搜集备份的目录

返回信息：

返回备份信息的结果集

举例说明：

```
SP_GET_TS_BAK_INFO('BOOKSHOP','D:\');
```

查询结果：

```
--没有表空间级备份，返回 0 行
```

48. SP_GET_TABLE_BAK_INFO

定义：

SP_GET_TABLE_BAK_INFO(
char* tabname,
char* schname,
char* backup_dir
)

功能说明：

获取指定目录下，指定表名所有备份的备份信息

参数说明：

tabname：表名，若设置为‘’，则搜集目录下所有表备份

schname：模式名

backup_dir：搜集备份的目录

返回信息：

返回备份信息的结果集

举例说明：

```
SP_GET_TABLE_BAK_INFO('BOOKSHOP', ' ','D:\');
```

查询结果：

```
--没有表级备份，返回 0 行
```

49. SP_GET_ALL_BAK_INFO

定义：

SP_GET_ALL_BAK_INFO(
schar* backup_dir,
schar* parallel_dir
)

功能说明：

获取指定目录下所有备份的备份信息

参数说明：

backup_dir：搜集备份的目录

parallel_dir：指定并行备份映射文件存储目录

返回信息：

返回备份信息的结果集

备注：

结果集形式与 SP_GET_DB_BAK_INFO 相同

举例说明：

```
SP_GET_ALL_BAK_INFO('D:\','D:\');
```

查询结果：

```
ID                  NAME                DESC                TYPE                RANGE
SCHEMA_NAME         OBJ_NAME            BASE                TIME
PATH                PAGE_SIZE           EXTENT_SIZE
CASE_SENSITIVE
                    LOG_PAGE_SIZE       UNICODE_FLAG        DB_VERSION
GLOBAL_VERSION      ENABLE_POLICY       ARCH_FLAG
ENCRYPT_
TYPE                IS_COMPRESSED

1           1                   BOOKSHOP_BAK        BOOKSHOP BAK INFO TEST. 0
1                   NULL        NULL                2011-09-21 13:20:34
                    D: \BOOKSHOP_BAK.bak    8192            16              1
512                 0               458753          V7.0.1.28-Build(2011.09.14)
0                   0               0               1
1 rows got
```

50. SP_GET_BAK_INFO

定义：

SP_GET_BAK_INFO(

char*　　backuppath

)

功能说明：

获取备份路径指定的备份信息

参数说明：

backuppath：　备份路径

返回信息：

返回备份路径指定的备份信息的结果集

备注：

结果集形式与 SP_GET_DB_BAK_INFO 相同

举例说明：

```
SP_GET_BAK_INFO('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
ID                  NAME                DESC                TYPE                RANGE
SCHEMA_NAME         OBJ_NAME            BASE                TIME
PATH                PAGE_SIZE           EXTENT_SIZE
CASE_SENSITIVE
                    LOG_PAGE_SIZE       UNICODE_FLAG        DB_VERSION
GLOBAL_VERSION      ENABLE_POLICY       ARCH_FLAG
ENCRYPT_
TYPE                IS_COMPRESSED

1           1                   BOOKSHOP_BAK        BOOKSHOP BAK INFO TEST. 0
1                   NULL        NULL                2011-09-21 13:20:34
```

```
                D:\BOOKSHOP_BAK.bak     8192            16              1
        512             0               458753          V7.0.1.28-Build(2011.09.14)
        0               0               0               1
        1 rows got
```

51. SP_BAK_GET_DB_FILE_LIST

定义：

SP_BAK_GET_DB_FILE_LIST(

char* pathname

)

功能说明：

获取指定备份所在库的文件列表

参数说明：

pathname：指定备份路径

返回信息：

返回指定备份所在库的文件的结果集

备注：

结果集格式信息如下

字段名	字段类型	字段含义
ID	INT	记录号
TYPE	VARCHAR(128)	文件类型
PATHNAME	VARCHAR(256)	文件路径名

举例说明：

```
SP_BAK_GET_DB_FILE_LIST('D:\BOOKSHOP_BAK.bak');
```

查询结果：

```
ID      TYPE            PATHNAME
1       DATAFILED:\soft_dm7\install\data\DAMENG\SYSTEM.DBF
2       DATAFILED:\soft_dm7\install\data\DAMENG\ROLL.DBF
3       DATAFILED:\soft_dm7\install\data\DAMENG\MAIN.DBF
4       DATAFILED:\soft_dm7\install\data\DAMENG\BOOKSHOP.DBF
5       LOGFILE
```

3）定时器管理

本小节中的定时器管理相关系统存储过程都必须在系统处于 MOUNT 状态时执行。

1. SP_ADD_TIMER

定义：

```
SP_ADD_TIMER(
        TIMER_NAME                  VARCHAR(128),
        TYPE                        INT,
        FREQ_MONTH_WEEK_INTERVAL    INT,
        FREQ_SUB_INTERVAL           INT,
        FREQ_MINUTE_INTERVAL        INT,
        START_TIME                  TIME,
        END_TIME                    TIME,
        DURING_START_DATE           DATETIME,
        DURING_END_DATE             DATETIME,
```

NO_END_DATE_FLAG BOOL
DESCRIBE VARCHAR(128),
IS_VALID BOOL
)

功能说明：

创建一个定时器。

参数说明：

timer_name：定时器名称

type：调度类型，取值如下：

1：执行一次
2：按日执行
3：按周执行
4：按月执行的第几天
5：按月执行的第一周
6：按月执行的第二周
7：按月执行的第三周
8：按月执行的第四周
9：按月执行的最后一周

freq_month_week_interval：间隔的月/周/天（调度类型决定）数

freq_sub_interval：第几天/星期几/一个星期中的某些天

freq_minute_interval：间隔的分钟数

start_time：开始时间

end_time：结束时间

during_start_date：有效日期时间段的开始日期时间

during_end_date：有效日期时间段的结束日期时间

no_end_date_flag：是否有结束日期（0：有结束日期；1：没有结束日期）

describe：描述

is_valid: 定时器是否有效

返回值：

无

说明：

1. type = 1 时，freq_sub_interval，freq_month_week_interval，freq_minute_interval，end_time，during_end_date 无效。只有 start_time，during_start_date 有意义。
2. type = 2 时，freq_month_week_interval 有效，表示相隔几天，取值范围为 1-100，freq_sub_interval 无效，freq_minute_interval <=24* 60 有效。
   a) 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效。
   b) 当 freq_minute_interval > 0 且，表示当天可执行多次。
   c) 当 freq_minute_interval >= 24* 60 时，非法。
3. type = 3 时，意思是每隔多少周开始工作（从开始日期算起）。计算方法为：(当前日期和开始日期的天数之差/7)% freq_month_week_interval =0 且当前是星期 freq_sub_interval，其中 freq_sub_interval 的八位如下图所示：

8	7	6	5	4	3	2	1

1-7 位分别代表星期天，星期一、星期二…、星期六，第 8 位无意义。这几位为 1 表示满足条件，为 0 表示不满足条件。

   a) 1<=freq_month_week_interval<=100，代表每隔多少周；
   b) 1<=freq_sub_interval<=127 代表星期中的某些天

c) freq_minute_interval 代表分钟数。
   i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效
   ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。
   iii. 当 freq_minute_interval >= 24* 60 时，非法。

4. type = 4 时，每 freq_month_week_interval 个月的第 freq_sub_interval 日开始工作。其中，是否满足 freq_month_week_interval 个月的判断条件是：(月份的差 + 日期的差/15) % freq_month_week_interval = 0 且当前是当月的 freq_sub_interval 日，表示满足条件；否则不满足。
   a) 1<=freq_sub_interval<31,代表第几日；
   b) 1<=freq_month_week_interval<100，代表每隔多少个月；
   c) freq_minute_interval 代表分钟数。
      i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效
      ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。
      iii. 当 freq_minute_interval >= 24* 60 时，非法。
5. type = 5，6，7，8，9 时，每 freq_month_week_interval 个月的第 type-4 周的周 freq_sub_interval 开始工作。其中，是否满足 freq_month_week_interval 个月的判断条件是：(月份的差 + 日期的差/15) % freq_month_week_interval = 0 且当前是当月的第 type-4 周的周 freq_sub_interval，表示满足条件；否则不满足。
   a) 1<=freq_sub_interval<=7，代表星期星期天到星期六（星期天是一个星期的第一天）；
   b) freq_month_week_interval<100，代表每隔多少个月；
   c) freq_minute_interval 代表分钟数。
      i. 当 freq_minute_interval = 0 时，当天只执行一次。end_time 无效
      ii. 当 freq_minute_interval > 0 且，表示当天可执行多次。
      iii. 当 freq_minute_interval >= 24* 60 时，非法。
6. 如果 no_end_date_flag = TRUE：表示永远不结束，一直存在下去。
7. 如果 is_valid= TRUE：表示定时器创建时就有效。
8. 总结 freq_sub_interval，freq_month_week_interval 和 freq_minute_interval 的取值范围如下表所示。

type	1		2		3		4		5,6,7,8,9	
	max	min	max	min	max	min	max	min	max	min
freq_sub_interval	0	0	0	0	127	1	31	1	7	1
freq_month_week_interval	0	0	100	1	100	1	100	1	100	1
freq_minute_interval	0	0	1440	0	1440	0	1440	0	1440	0

举例说明：

创建一个定时器，每天 02：00 进行调度，开始日期：2010-02-01，结束日期：2010-05-01，间隔天数 1 天，每隔一分钟循环执行。

```
CALL SP_ADD_TIMER('TIMER1', 2, 1, 0, 1, '02:00:00', '20:00:00', '2011-02-01 14:30:34', '2011-09-01 ', 0, '每天凌晨两点进行调度', 1);
```

2．SP_DROP_TIMER

定义：

```
SP_DROP_TIMER (
    timer_name varchar(128)
)
```

功能说明：

删除一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

删除定时器 TIMER1

```
CALL SP_DROP_TIMER('TIMER1');
```

3. SP_OPEN_TIMER

定义：

```
SP_OPEN_TIMER (
    timer_name varchar(128)
)
```

功能说明：

打开一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

打开定时器 TIMER1

```
SP_OPEN_TIMER('TIMER1');
```

4. SP_CLOSE_TIMER

定义：

```
SP_CLOSE_TIMER (
    timer_name varchar(128)
)
```

功能说明：

关闭一个定时器。

参数说明：

timer_name：定时器名

返回值：

无

举例说明：

关闭定时器 TIMER1

```
SP_CLOSE_TIMER('TIMER1');
```

4）作业调度管理

本小节只给出作业调度管理相关系统存储过程的语法说明，作业调度的具体使用及这些存储过程的相互关系请参考《DM 系统管理员手册》相关章节。

1. SP_INIT_JOB_SYS

定义：

```
SP_INIT_JOB_SYS (
  createflag     int
```

)

功能说明：

初始化 JOB 系统表

参数说明：

createflag：0，删除作业相关系统表；1，创建作业相关系统表

返回值：

无

举例说明：

创建作业相关系统表

```
SP_INIT_JOB_SYS(1);
```

删除作业相关系统表

```
SP_INIT_JOB_SYS(0);
```

2. SP_CREATE_OPERATOR

定义：

```
SP_CREATE_OPERATOR (
   opr_name          varchar(128),
   enabled           int,
   emailaddr         varchar(128),
   netsend_ip        varchar(128)
)
```

功能说明：

创建一个操作员

参数说明：

opr_name：操作员名字，必须是有效的标识符，同时不能是 DM 关键字。操作员不能同名，创建前会检测这个名字是否已经存在，如果存在则报错

enabled：是否启用这个操作员。参数类型为布尔类型

emailaddr：修改操作员的 EMAIL 地址

netsend_ip：修改操作员的 IP 地址

返回值：

无

举例说明：

创建一个操作员

```
SP_CREATE_OPERATOR('A', 1, 'xxx@dameng.com', '192.168.0.123');
```

3. SP_ALTER_OPERATOR

定义：

```
SP_ALTER_OPERATOR (
   opr_name          varchar(128),
   enabled           int,
   emailaddr         varchar(128),
   netsend_ip        varchar(128)
)
```

功能说明：

修改一个操作员

参数说明：

opr_name：操作员名字，必须是有效的标识符，同时不能是 DM 关键字。操作员不能同名，创建前会检测这个名字是否已经存在，如果存在则报错

enabled：是否启用这个操作员。参数类型为布尔类型

emailaddr：这个操作员的 EMAIL 地址

netsend_ip：这个操作员的 IP 地址（主要用于网络发送）

返回值：

无

举例说明：

修改一个操作员 A 的 IP 为 192.168.0.111

```
SP_ALTER_OPERATOR('A', 1, 'xxx@dameng.com', '192.168.0.111');
```

4. SP_DROP_OPERATOR

定义：

```
SP_DROP_OPERATOR (
  opr_name                varchar(128)
)
```

功能说明：

删除一个操作员

参数说明：

opr_name：操作员名字

返回值：

无

举例说明：

删除操作员 A。

```
SP_DROP_OPERATOR('A');
```

5. SP_CREATE_JOB

定义：

```
SP_CREATE_JOB (
  job_name                varchar(128),
  enabled                 int,
  enable_email            int,
  email_optr_name         varchar(128),
  email_type              int,
  enabled_netsend         int,
  netsend_optr_name       varchar(128),
  netsend_type            int,
  describe                varchar(8187)
)
```

功能说明：

创建一个作业

参数说明：

job_name：作业名字，必须是有效的标识符，同时不能是 DM 关键字。作业名不能重复，创建前会检测这个名字是否已经存在，如果存在则报错

enabled：是否有效，或者启用，参数值为布尔值

enable_email：表示是否通过邮件功能将作业的执行结果通知给操作员，参数值为布尔值

email_optr_name：如果要通过邮件通知，则这里需要指定操作员，当然在创建的时候需要判断这个操作员是否存在，不存在则报错

email_type：是邮件发送类型，包括 0，当作业成功时进行通知；1，当作业失

败时进行通知；2，当作业完成时进行通知

enable_netsend：和上面邮件的意义相对应，类型为布尔值

netsend_optr_name：和上面邮件的意义相对应

netsend_type：和上面邮件的意义相对应

describe：对这个作业的描述，创建时需要对它的长度检测，不能超过 500 字节，否则报错

返回值：

无

举例说明：

创建一个作业

```
SP_CREATE_JOB('TEST', 1, 1, 'WZF', 2, 1, 'WZF', 2, '每一个测试作业');
```

6. SP_ALTER_JOB

定义：

```
SP_ALTER_JOB (
  job_name              varchar(128),
  enabled               int,
  enable_email          int,
  email_optr_name       varchar(128),
  email_type            int,
  enabled_netsend       int,
  netsend_optr_name     varchar(128),
  netsend_type          int,
  describe              varchar(8187)
)
```

功能说明：

修改一个作业

参数说明：

参数和 SP_CREATE_JOB 的相同

返回值：

无

举例说明：

修改一个作业的描述

```
SP_ALTER_JOB('TEST', 0, 1, 'DBA', 2, 1, 'DBA', 2, '修改一个作业');
```

7. SP_DROP_JOB

定义：

```
SP_DROP_JOB (
  job_name              varchar(128)
)
```

功能说明：

删除一个作业

参数说明：

job_name：所要删除的作业名字

返回值：

无

举例说明：

删除作业 TEST

```
SP_DROP_JOB('TEST');
```

8. SP_ADD_JOB_STEP

定义：

```
SP_ADD_JOB_STEP (
   job_name              varchar(128),
   step_name             varchar(128),
   type                  int,
   command               varchar(8187),
   succ_action           int,
   fail_action           int,
   retry_attempts        int,
   retry_interval        int,
   output_file_path      varchar(256),
   append_flag           int
   )
```

功能说明：

在指定作业中加入一个步骤

参数说明：

job_name：要增加步骤的作业名，在加入前需要检测这个作业是否存在

step_name：步骤名，必须是有效的标识符，同时不能是 DM 关键字。步骤名不能重复，创建前会检测这个名字是否已经存在，如果存在则报错

type：表示的是这个步骤的类型，类型主要包括下面几种：0，要执行的 SQL 语句或者语句块；1，数据库备份操作；2，重组索引、表等；3，更新数据库中的统计信息；4，数据迁移；5，执行指定的应用程序

command：这个步骤需要做的工作。主要就是 SQL 语句或者函数等命令

succ_action：操作包括：0，接着执行下一个步骤；1，报告执行成功；3，返回第一个步骤重新开始执行

fail_action：操作包括：0，接着执行下一个步骤； 2，报告执行失败；3，返回第一个步骤重新开始执行

retry_attempts：表示的是这个步骤在失败后重试几次

retry_interval：表示的是这个步骤在失败后重试是隔时间，最长 10 秒钟

output_file_path：一些步骤的执行可能会有输出，那么这个参数定义的就是其输出路径。创建时需要对这个路径进行合法法性检测

append_flag：这个参数的作用就是在输出到上面的路径文件时是否是从文件后面写入，参数类型为布尔类型

返回值：

无

举例说明：

在作业 TEST 中加入一个步骤 STEP1；

```
SP_ADD_JOB_STEP('TEST', 'STEP1', 0, 'insert into myinfo values(1000, "STEP1 Hello World");', 0, 0, 1, 1, NULL, 0);
```

9. SP_DROP_JOB_STEP

定义：

```
SP_DROP_JOB_STEP (
   job_name              varchar(128),
```

```
    step_name                varchar(128)
)
```

功能说明：

删除作业的一个步骤

参数说明：

job_name： 作业名

step_name：所要删除的步骤名

返回值：

无

举例说明：

删除步骤 STEP1

```
SP_DROP_JOB_STEP ('TEST', 'STEP1');
```

10. SP_ADD_JOB_SCHEDULE

定义：

```
SP_ADD_JOB_SCHEDULE (
  job_name               varchar(128),
  schedule_name          varchar(128),
  enable                 int,
  type                   int,
  freq_interval          int,
  freq_sub_interval      int,
  freq_minute_interval   int,
  starttime              varchar(128),
  endtime                varchar(128),
  during_start_date      varchar(128),
  during_end_date        varchar(128),
  describe               varchar(500)
  )
```

功能说明：

在指定的作业上建立一个对作业中所有步骤的调度。

参数说明：

job_name：作业名，这个调度所属的作业

schedule_name：要创建的调度的名字，必须是有效的标识符，同时不能是 DM 关键字

enable：功能和上面一样，也是指明这个调度是否启用

type：调度类型，包括 0，只执行一次；1，按天计算执行频率；2，按周计算执行频率；3，按月计划执行频率，同时指定在一个月中的第几天；4，按月计算，指定在一个月第一周；5，按月，指定一个月第二周；6，按月，指定一个月第三周；7，按月，指定一个月第四周；8，按月，指定一个月最后一周

freq_interval、freq_sub_interval：这两表示的意义与上面的类型有关，不同类型具有不同意义。比如对于类型 1，它的值就是指的是每多少天执行一次。对于类型 2，指定是每 freq_interval 个周的周几（可以连接多天，一周只有七天，所以是 1 到 7 中的任意值，FREQ_SUB_INTERVAL 的值从低位到高位，分别表示周一周二……周日是否已经指定）执行一次。类型 3 表示的是每 FREQ_INTERVAL 个月的第 freq_sub_interval（一个月中的任意一天，从 1 到 31）天；类型 4 表示的是每 freq_interval 个月的第 1 周的第 freq_sub_interval（从 0-8）天；类型 5、6、7、8 都

与上面的相似，不再叙述

freq_minute_intervaL：表示的是分钟的信息，在上面已经定义的具体的一天内，每隔 freq_minute_interval（0- 1440）分钟执行一次

starttime、endtime：表示的是上面定义的时间的有效时间范围，上面定义的要不是执行的循环方式，要不就只执行一次，但这个参数指定的是一个有效范围，只有在这个范围内才会被执行

during_start_date、during_end_date：上面表示的是一个时间范围，也就是某一天的一个时间段，但这两个表示的是日期范围

describe：对这个调度的描述信息，长度需要检测

返回值：

无

举例说明：

在作业 TEST 上创建一个调度 SCHEDULE1；

```
SP_ADD_JOB_SCHEDULE('TEST', 'SCHEDULE1', 1, 1, 1, 0, 1, CURTIME, '23:59:59', CURDATE, NULL, '一个测试调度');
```

11. SP_DROP_JOB_SCHEDULE

定义：

```
SP_DROP_JOB_SCHEDULE (
  job_name          varchar(128),
  schedule_name     varchar(128)
  )
```

功能说明：

删除一个作业下面的调度。

参数说明：

job_name：作业名，要删除的调度所属的作业

schedule_name：要删除的调度的名字

返回值：

无

举例说明：

删除调度 SCHEDULE1；

```
SP_DROP_JOB_SCHEDULE('TEST', 'SCHEDULE1');
```

12. SP_CREATE_ALERT

定义：

```
SP_CREATE_ALERT (
  name          varchar(128),
  enabled       int,
  type          int,
  errtype       int,
  errcode       int,
  delaytime     int,
  describe      varchar(8187)
)
```

功能说明：

创建一个警告，警告的作用是对一些指定的事件进行捕获并用预定义的方式通知操作员。

参数说明：

name：警告名，必须是有效的标识符，同时不能是 DM 关键字。警告名不能重复，创建前会检测这个名字是否已经存在，如果存在则报错

enabled：是否启用，类型为布尔

type：警告的类型：0，当发生错误时报；1，某些事件激发时报告

errtype：错误的类型，当警告类型为 0 时，主要有下面几种：1，普通错误；2，启动错误；3，系统错误；4，数据库配置错误；5，分析错误；6，权限错误；7 运行时错误；8，备份恢复错误；9，作业管理错误；数据复制错误；11，其它错误；12，指定错误码。当警告类型为 1 时，主要有下面几种：1，DDL 事件错误；2，授权或者回收权时错误；3，连接错误；4，分析错误；5，备份错误

errcode：这个参数指的是在警告的类型为 0 情况下的一个错误码，必须是小于零的

delaytime：这个参数指的是警告可以推迟的时间

describe：对这个警告的描述。长度不可以超过 500 字节

返回值：

无

举例说明：

创建一个警告

```
SP_CREATE_ALERT('ALERT1', 1, 0, 1, -600, 1, '错误码的测试');
```

13. SP_ALTER_ALERT

定义：

```
SP_ ALTER _ALERT (
   name              varchar(128),
   enabled           int,
   type              int,
   errtype           int,
   errcode           int,
   delaytime         int,
   describe          varchar(8187)
)
```

功能说明：

修改一个警告的信息

参数说明：

name：这个警告的名字，创建前需要检测同名的警告是否已经存在

enabled：是否启用，类型为布尔

type：警告的类型：0，当发生错误时报；1，某些事件激发时报告

errtype：错误的类型，当警告类型为 0 时，主要有下面几种：1，普通错误；2，启动错误；3，系统错误；4，数据库配置错误；5，分析错误；6，权限错误；7 运行时错误；8，备份恢复错误；9，作业管理错误；数据复制错误；11，其它错误；12，指定错误码。当警告类型为 1 时，主要有下面几种：1，DDL 事件错误；2，授权或者回收权时错误；3，连接错误；4，分析错误；5，备份错误

errcode：这个参数指的是在警告的类型为 0 情况下的一个错误码，必须是小于零的

delaytime：这个参数指的是警告可以推迟的时间

describe：对这个警告的描述。长度不可以超过 500 字节

返回值：

无

举例说明：

修改一个警告

```
SP_ALTER_ALERT('ALERT1', 1, 0, 1, -10000, 1, '错误码的测试');
```

14. SP_DROP_ALERT

定义：

```
SP_DROP _ALERT (
  name              varchar(128),
)
```

功能说明：

删除一个警告

参数说明：

name：这个警告的名字，创建前需要检测同名的警告是否已经存在

返回值：

无

举例说明：

删除警告 ALERT1

```
SP_DROP_ALERT('ALERT1');
```

15. SP_ALERT_ADD_OPERATOR

定义：

```
SP_ALERT_ADD_OPERATOR (
  alertname          varchar(128),
  opr_name           varchar(128),
  enablemail         int,
  enablenetsent      int
)
```

功能说明：

将一个警告与一个操作员的关系关联起来

参数说明：

alertname：指定的警告名，创建时需要检测这个名字是否存在

opr_name：操作员名字，同样需要检测其名字是否存在

enablemail：指定是否开启通过邮件通知操作员，值只能是 0 和 1

enablenetsend：指定是否开启通过网络发送通知操作员，只能是 0 和 1

返回值：

无

举例说明：

删除警告 ALERT1

```
SP_ALERT_ADD_OPERATOR ('ALERT1','A',1,1);
```

16. SP_ALERT_DROP_OPERATOR

定义：

```
SP_ALERT_DROP_OPERATOR (
  alertname          varchar(128),
  opr_name           varchar(128)
)
```

功能说明：

删除一个警告与一个操作员的关联关系

参数说明：

alertname：指定的警告名

opr_name：操作员名字，同样需要检测其名字是否存在

返回值：

无

举例说明：

删除警告 ALERT1 与操作员的关系

```
SP_ALERT_DROP_OPERATOR ('ALERT1','A');
```

17. SP_JOB_CONFIG_START

定义：

```
SP_JOB_CONFIG_START (
  job_name              varchar(128)
)
```

功能说明：

标记作业配置开始

参数说明：

job_name：作业名

返回值：

无

举例说明：

标记 TEST 作业配置开始

```
SP_JOB_CONFIG_START ('TEST');
```

18. SP_JOB_CONFIG_COMMIT

定义：

```
SP_JOB_CONFIG_COMMIT (
  job_name              varchar(128)
)
```

功能说明：

完成作业的所有配置并且生效

参数说明：

job_name：作业名

返回值：

无

举例说明：

完成作业 TEST 的所有配置

```
SP_JOB_CONFIG_COMMIT ('TEST');
```

19. SP_JOB_CLEAR_HISTORIES

定义：

```
SP_JOB_CLEAR_HISTORIES (
  job_name              varchar(128)
)
```

功能说明：

清除所有作业日志记录。

参数说明：

job_name：作业名

返回值：

无

举例说明：

清除作业 TEST 的所有日志记录

```
SP_JOB_CLEAR_HISTORIES ('TEST');
```

20. SP_ALERT_DROP_HISTORIES

定义：

```
SP_ALERT_DROP_HISTORIES (
  alertname                varchar(128)
)
```

功能说明：

清除所有的警告日志记录。

参数说明：

alertname，警告名

返回值：

无

举例说明：

清除所有的 ALERT1 警告日志记录

```
SP_ALERT_DROP_HISTORIES ('ALERT1');
```

21. SP_ADD_MAIL_INFO

定义：

```
SP_ADD_MAIL_INFO (
  login_name               varchar(128),
  login_pwd                varchar(128),
  smtp_server              varchar(128),
  email                    varchar(128),
  user_name                varchar(128),
  user_pwd                 varchar(128)
)
```

功能说明：

增加一个 DMMONITOR 操作员的信息。

参数说明：

login_name：这个操作员的登录名，必选参数，不能为空

login_pwd：这个操作员登录时的密码，必选参数，不能为空

smtp_server：登录时指定的邮件 SMTP 服务器，必选参数，不能为空

email：邮件地址，必选参数，不能为空

user_name：邮件用户名，必选参数，不能为空

user_pwd：邮箱登录密码，必选参数，不能为空

返回值：

无

举例说明：

增加一个 DMMONITOR 操作员的信息

```
SP_ADD_MAIL_INFO('SYSDBA','SYSDBA','SMTP.QQ.COM','xxx@qq.com','xxx','****');
```

22. SP_ALTER_MAIL_INFO

参数说明：

定义：

```
SP_ALTER_MAIL_INFO (
   login_name              varchar(128),
   login_pwd               varchar(128),
   smtp_server             varchar(128),
   email                   varchar(128),
   user_name               varchar(128),
   user_pwd                varchar(128)
)
```

功能说明：

修改一个已存在的 DMMONITOR 操作员的信息。

参数说明：

login_name：这个操作员的登录名

login_pwd：这个操作员登录时的密码

smtp_server：登录时指定的邮件 SMTP 服务器

email：邮件地址

user_name：邮件用户名

user_pwd：邮箱登录密码

返回值：

无

举例说明：

修改 DMMONITOR 操作员的信息。

```
SP_ALTER_MAIL_INFO('SYSDBA','SYSDBA','SMTP.QQ.COM','xxx@qq.com','xxx','abcd');
```

23. SP_DROP_MAIL_INFO

定义：

```
SP_DROP_MAIL_INFO ( login_name              varchar(128) )
```

功能说明：

删除一个 DMMONITOR 操作员的信息

参数说明：

login_name：这个操作员的登录名

返回值：

无

举例说明：

删除操作员 SYSDBA 的信息。

```
SP_DROP_MAIL_INFO ('SYSDBA');
```

5）数据复制管理

本小节的存储过程都与 DM 的数据复制功能相关，关于数据复制的概念和相关环境配置与操作可以参考《DM7 系统管理员手册》相关章节。

1. SP_INIT_REP_SYS

定义：

```
SP_INIT_REP_SYS(
CREATE_FLAG          INT
);
```

功能说明：

创建或删除数据复制所需的系统表

参数说明：

CREATE_FLAG：为 1 表示创建复制所需系统表；为 0 表示删除这些系统表

返回值：

无

举例说明：

创建复制所需的系统表

```
SP_INIT_REP_SYS(1);
```

2. SP_RPS_ADD_GROUP

定义：

```
SP_RPS_ADD_GROUP(
GROUP_NAME    VARCHAR(128),
GROUP_DESC          VARCHAR(1000)
);
```

功能说明：

创建复制组

参数说明：

GROUP_NAME：创建的复制组名称

GROUP_DESC：复制组描述

返回值：

无

备注：

指示 RPS 创建一个新的复制组。如果已存在同名复制组则报错。

举例说明：

创建复制组 REPB2C

```
SP_RPS_CREATE_GROUP('REPB2C','主从同步复制');
```

3. SP_RPS_DROP_GROUP

定义：

```
SP_RPS_DROP_GROUP(
GROUP_NAME    VARCHAR(128)
);
```

功能说明：

删除复制组

参数说明：

GROUP_NAME：复制组名称

返回值：

无

举例说明：

删除复制组 REPB2C

```
SP_RPS_DROP_GROUP ('REPB2C');
```

4. SP_RPS_ADD_REPLICATION

定义：

```
SP_RPS_ADD_REPLICATION(
GRP_NAME        VARCHAR(128),
REP_NAME        VARCHAR(128),
REP_DESC            VARCHAR(1000),
MINSTANCE       VARCHAR(128),
SINSTANCE       VARCHAR(128),
REP_TIMER       VARCHAR(128),
ARCH_PATH       VARCHAR(256)
);
```

功能说明：

创建复制关系

参数说明：

GROUP_NAME：复制组名称

GRP_NAME：复制组名,

REP_NAME：复制名，必须在 RPS 上唯一

REP_DESC：复制描述

MINSTANCE：主节点实例名，必须在 RPS 的 MAL 中已配置

SINSTANCE：从节点实例名，必须在 RPS 的 MAL 中已配置

REP_TIMER：复制定时器名。借助定时器，可以设置复制数据的同步时机。如果是同步复制则为 NULL

ARCH_PATH：主服务器上逻辑日志的完整归档路径。

返回值：

无

举例说明：

创建复制关系

```
SP_RPS_ADD_REPLICATION (‘REPB2C’, ‘REPB2C’, ‘B 到 C 的同步复制’, ‘B’, ‘C’, NULL, ‘{ DEFARCHPATH}\REPB2C’);
```

5. SP_RPS_DROP_REPLICATION

定义：

```
SP_RPS_DROP_REPLICATION (
REP_NAME        VARCHAR(128)
);
```

功能说明：

删除复制关系

参数说明：

REP_NAME：复制名称

返回值：

无

举例说明：

删除复制关系

```
SP_RPS_DROP_REPLICATION (' B 到 C 的同步复制');
```

6. SP_RPS_SET_ROUTE_FAULT_TIMEOUT

定义：

```
SP_RPS_SET_ROUTE_FAULT_TIMEOUT (
REP_NAME        VARCHAR(128),
TIMEOUTS            INT
);
```

功能说明：

设置复制路径故障超时

参数说明：

REP_NAME：复制关系名。

TIMEOUTS：故障超时值，以秒为单位。0 为立即超时；-1 表示无超时限制

返回值：

无

备注：

该接口用于设置复制路径故障处理策略。设置后，RPS 如检测到复制路径产生故障，且故障持续超过设定的超时值后，则需要取消故障的复制关系。

举例说明：

设置复制路径故障超时

```
SP_ RPS_SET_ROUTE_FAULT_TIMEOUT ('B 到 C 的同步复制',10);
```

7. SP_RPS_SET_INST_FAULT_TIMEOUT

定义：

```
SP_RPS_SET_INST_FAULT_TIMEOUT (
INST_NAME       VARCHAR(128),
TIMEOUTS            INT
);
```

功能说明：

设置复制节点故障超时

参数说明：

INST_NAME：复制节点实例名

TIMEOUTS：故障超时值，以秒为单位。0 为立即超时；-1 表示无超时限制

返回值：

无

举例说明：

设置复制节点故障超时

```
SP_ RPS_SET_INST_FAULT_TIMEOUT ('B 到 C 的同步复制',10);
```

8. SP_RPS_ADD_TIMER

定义：

```
SP_RPS_ADD_TIMER(
TIMER_NAME          VARCHAR(128),
TIMER_DESC          VARCHAR(1000),
TYPE$                   INT,
FERQ_INTERVAL           INT,
FREQ_SUB_INTERVAL       INT,
FREQ_MINUTE_INTERVAL    INT,
START_TIME      TIME,
END_TIME            TIME,
DURING_START_DATE   DATETIME,
DURING_END_DATE     DATETIME,
NO_END_DATA_FLAG        INT
);
```

功能说明：

设置复制关系的定时器

参数说明：

TIMER_NAME：定时器名

TIMER_DESC：定时器描述

TYPE$：定时器类型，取值如下：

1：执行一次

2：每日执行

3：每周执行

4：按月执行的第几天

5：按月执行的第一周

6：按月执行的第二周

7：按月执行的第三周

8：按月执行的第四周

9：按月执行的最后一周

FERQ_INTERVAL：间隔的月/周（调度类型决定）数

FREQ_SUB_INTERVAL： 间隔天数

FREQ_MINUTE_INTERVAL：间隔的分钟数

START_TIME：开始时间

END_TIME：结束时间

DURING_START_DATE：有效日期时间段的开始日期时间

DURING_END_DATE：有效日期时间段结束日期时间

NO_END_DATA_FLAG：结束日期是否无效标识

返回值：

无

举例说明：

设置复制关系的定时器

```
SP_RPS_ADD_TIMER ('TIMER1','按天计算', 1, 1, 0, 1, CURTIME, '23:59:59', NOW, NULL, 1);
```

9. SP_RPS_REP_RESET_TIMER

定义：

```
SP_RPS_REP_RESET_TIMER(
REP_NAME        VARCHAR(128)，
TIMER_NAME          VARCHAR(128)
);
```

功能说明：

重新设置复制关系的定时器

参数说明：

REP_NAME：复制名

TIMER_NAME：新的定时器名

返回值：

无

举例说明：

重新设置复制关系的定时器

```
SP_ RPS_REP_RESET_TIMER ('B 到 C 的同步复制','TIMER1');
```

10. SP_RPS_ADD_TAB_MAP

定义：

```
SP_RPS_ADD_TAB_MAP(
REP_NAME        VARCHAR(128),
MTAB_SCHEMA  VARCHAR(128),
MTAB_NAME      VARCHAR(128),
STAB_SCHEMA   VARCHAR(128),
STAB_NAME      VARCHAR(128),
READ_ONLY_MODE        INT
);
```

功能说明：

添加表级复制映射

参数说明：

REP_NAME：复制关系名

MTAB_SCHEMA：主表模式名

MTAB_NAME：主表名

STAB_SCHEMA：从表模式名

STAB_NAME：从表名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加复制映射

```
SP_RPS_ADD_TAB_MAP(‘REPB2C’, ‘USER1’, ‘T1’, ‘USER2’, ‘T2’, 0);
```

11. SP_RPS_DROP_TAB_MAP

定义：

```
SP_RPS_DROP_TAB_MAP(
REP_NAME        VARCHAR(128),
MTAB_SCHEMA  VARCHAR(128),
MTAB_NAME      VARCHAR(128),
STAB_SCHEMA   VARCHAR(128),
STAB_NAME      VARCHAR(128),
);
```

功能说明：

删除表级复制映射

参数说明：

REP_NAME：复制关系名

MTAB_SCHEMA：主表模式名

MTAB_NAME：主表名

STAB_SCHEMA：从表模式名

STAB_NAME：从表名

返回值：

无

举例说明：

删除表级复制映射

```
SP_RPS_DROP_TAB_MAP('REPB2C', 'USER1', 'T1', 'USER2', 'T2');
```

12. SP_RPS_ADD_SCH_MAP

定义：

```
SP_RPS_ADD_SCH_MAP(
REP_NAME        VARCHAR(128),
MSCH            VARCHAR(128),
SSCH            VARCHAR(128),
READ_ONLY_MODE      INT
);
```

功能说明：

添加模式级复制映射

参数说明：

REP_NAME：复制关系名

MSCH：主模式名

SSCH： 从表模式名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加复制映射

```
SP_RPS_ADD_SCH_MAP('REPB2C', 'USER1', 'USER2', 0);
```

13. SP_RPS_DROP_SCH_MAP

定义：

```
SP_RPS_DROP_SCH_MAP(
REP_NAME        VARCHAR(128),
MSCH            VARCHAR(128),
SSCH            VARCHAR(128)
);
```

功能说明：

删除模式级复制映射

参数说明：

REP_NAME：复制关系名

MSCH：主模式名

SSCH：从模式名

返回值：

无

举例说明：

删除模式级复制映射

```
SP_RPS_DROP_SCH_MAP('REPB2C', 'USER1', 'USER2');
```

14. SP_RPS_ADD_DB_MAP

定义：

```
SP_RPS_ADD_DB_MAP(
REP_NAME        VARCHAR(128),
READ_ONLY_MODE        INT
);
```

功能说明：

添加库级复制映射

参数说明：

REP_NAME：复制关系名

READ_ONLY_MODE：只读复制模式，1 表示只读模式，从表只接受复制更新，0 表示非只读模式

返回值：

无

举例说明：

添加库级复制映射

```
SP_RPS_ADD_DB_MAP('REPB2C', 0);
```

15. SP_RPS_DROP_DB_MAP

定义：

```
SP_RPS_DROP_DB_MAP(
REP_NAME        VARCHAR(128)
);
```

功能说明：

删除库级复制映射

参数说明：

REP_NAME：复制关系名

返回值：

无

举例说明：

删除库级复制映射

```
SP_RPS_DROP_DB_MAP('REPB2C');
```

16. SP_RPS_SET_BEGIN

定义：

```
SP_RPS_SET_BEGIN(
GRP_NAME        VARCHAR(128),
);
```

功能说明：

开始复制设置

参数说明：

GRP_NAME：复制组名,

返回值：

无

备注：

开始对指定复制组进行属性设置。创建/删除复制关系与创建/删除复制映射等接口都必须在此接口调用后执行，否则会报错“错误的复制设置序列”。同一会话中也不能同时开始多个复制设置。

举例说明：

复制组 REPB2C 开始复制

```
SP_RPS_SET_BEGIN(‘REPB2C’);
```

17. SP_RPS_SET_APPLY

定义：

SP_RPS_SET_APPLY ();

功能说明：

提交复制设置，保存并提交本次设置的所有操作。如果需要继续设置，则必须重新调用 SP_RPS_SET_BEGIN。

参数说明：

无

返回值：

无

举例说明：

提交复制设置

```
SP_RPS_SET_APPLY ();
```

18. SP_RPS_SET_CANCEL

定义：

SP_RPS_SET_CANCEL ();

功能说明：

放弃复制设置，放弃本次设置的所有操作。如果需要重新设置，则必须再次调用 SP_RPS_SET_BEGIN。

参数说明：

无

返回值：

无

举例说明：

放弃复制设置

```
SP_RPS_SET_CANCEL();
```

6）模式对象相关信息管理

1. TABLEDEF

定义：

CHAR *

TABLEDEF (
schname varchar(128),
tablename varchar(128)
)

功能说明：

获得表的定义

参数说明：

schname：模式名

tablename：表名

返回值：

表的定义

举例说明：

```
SELECT TABLEDEF('PRODUCTION','PRODUCT');
```

2. VIEWDEF

定义：

CHAR *
VIEWDEF (
schname varchar(128),
viewname varchar(128)
)

功能说明：

获得视图的定义

参数说明：

schname：模式名

viewname：视图名

返回值：

视图的定义

举例说明：

```
SELECT VIEWDEF('PURCHASING','VENDOR_EXCELLENT');
```

3. SF_VIEW_EXPIRED

定义：

INI*
SF_VIEW_EXPIRED(
SCHNAME varchar(128),
VIEWNAME varchar(128)
)

功能说明：

检查当前系统表中视图列定义是否有效。

返回值：

返回 0 或 1。0 表示有效，1 表示无效。

举例说明：

```
CREATE TABLE T_01_VIEW_DEFINE_00(C1 INT,C2 INT);
CREATE VIEW TEST_T_01_VIEW_DEFINE_00 AS SELECT* FROM
T_01_VIEW_DEFINE_00;
SELECT SF_VIEW_EXPIRED('SYSDBA','TEST_T_01_VIEW_DEFINE_00');
--查询结果为 0
```

```
ALTER TABLE T_01_VIEW_DEFINE_00 DROP COLUMN C1 CASCADE ;
SELECT SF_VIEW_EXPIRED('SYSDBA','TEST_T_01_VIEW_DEFINE_00');
--查询结果为 1
```

4. CHECKDEF

定义：

```
CHAR *
CHECKDEF (
consid     int,
preflag    int
)
```

功能说明：

获得 check 约束的定义

参数说明：

consid：check 约束 id 号

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

check 约束的定义

举例说明：

```
CREATE TABLE TEST_CHECKDEF(C1 INT CHECK(C1>10));
通过查询系统表
SELECT A.name, A.ID FROM SYSOBJECTS A, SYSOBJECTS B WHERE B.NAME='TEST_CHECKDEF' AND A.PID=B.ID AND A.SUBTYPE$='CONS';
--得到约束 ID 为 134217770
SELECT  CHECKDEF(134217770,1);
```

5. CONSDEF

定义：

```
CHAR *
CONSDEF (
indexid int,
preflag    int
)
```

功能说明：

获取 unique 约束的定义

参数说明：

indexid：索引号数字字符串

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

unique 约束的定义

举例说明：

```
CREATE TABLE TEST_CONSDEF(C1 INT PRIMARY KEY,C2 INT, CONSTRAINT CONS1 UNIQUE(C2));
--通过查询系统表
SELECT C.INDEXID FROM SYSOBJECTS O,SYSCONS C WHERE O.NAME='CONS1' AND O.ID=C.ID;
--系统生成 C2 上的 INDEX 为 33555481
SELECT CONSDEF(33555481,1);
```

6. INDEXDEF

定义：

```
CHAR *
INDEXDEF (
indexid    int,
preflag    int
)
```

功能说明：

获取 index 的创建定义

参数说明：

indexid：索引 ID

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

索引的创建定义

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
--查询系统表得到索引 ID
select name, id from sysobjects where name='PRODUCT_IND' and subtype$='INDEX';
SELECT indexdef(33555530,1);
```

7. SP_REORGANIZE_INDEX

定义：

```
SP_REORGANIZE_INDEX (
    schname       varchar(128),
    indexname    varchar(128)
)
```

功能说明：

对指定表进行空间整理

参数说明：

schname：模式名

indexname：索引名

返回值：

无

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
CALL SP_REORGANIZE_INDEX('PRODUCTION','PRODUCT_IND');
```

8. SP_REBUILD_INDEX

定义：

```
SP_REBUILD_INDEX (
schname       varchar(128),
indexed        int
)
```

功能说明：

重建索引。约束：1. 分区子表、临时表和系统表上建的索引不支持重建；2. 虚索引和聚集索引不支持重建。

参数说明：

schname：模式名

indexid：索引 ID

返回值：

无

举例说明：

```
CREATE INDEX PRODUCT_IND ON PRODUCTION.PRODUCT(PRODUCTID);
--查询系统表得到索引 ID
SP_REBUILD_INDEX('SYSDBA ', 33555530);
```

9. CONTEXT_INDEX_DEF

定义：

```
CHAR *
CONTEXT_INDEX_DEF (
indexed   int,
preflag    int
)
```

功能说明：

获取 context_index 的创建定义

参数说明：

indexid：索引 ID

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

索引的创建定义

举例说明：

```
create context index product_cind on production.product(name) LEXER DEFAULT_LEXER;
--查询系统表得到全文索引 ID
select name, id from sysobjects where name='PRODUCT_CINd';
select context_index_def(33555531, 1);
```

10. SYNONYMDEF

定义：

```
CHAR *
SYNONYMDEF (
username varchar(128),
synname    varchar(128),
type         int,
preflag    int
)
```

功能说明：

获取同义词的创建定义

参数说明：

username：用户名

synname：同义词名

type：同义词类型 0，public 1, user

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

同义词的创建定义

举例说明：

```
SELECT SYNONYMDEF('SYSDBA', 'SYSOBJECTS',0,1);
```

11. SEQDEF

定义：

```
CHAR *
SEQDEF (
seqid       int,
preflag     int
)
```

功能说明：

获取序列的创建定义

参数说明：

seqid：序列 id 号

preflag：对象前缀个数，1 表示导出模式名，0 表示只导出对象名

返回值：

序列的创建定义

举例说明：

```
CREATE SEQUENCE SEQ1;
SELECT SEQDEF(167772160, 1);
```

12. IDENT_CURRENT

定义：

```
int
IDENT_CURRENT (
    fulltablename varchar(8187)
)
```

功能说明：

获取自增列当前值

参数说明：

fulltablename：表全名；格式为“模式名.表名”

返回值：

自增列当前值

举例说明：

```
SELECT IDENT_current('PRODUCTION.PRODUCT');
```

13. IDENT_SEED

定义：

```
INT
IDENT_SEED (
    fulltablename varchar(8187)
)
```

功能说明：

获取自增列种子

参数说明：

fulltablename：表全名；格式为“模式名.表名”

返回值：

自增列种子

举例说明：

```
SELECT IDENT_SEED('PRODUCTION.PRODUCT');
```

14. IDENT_INCR

定义：

```
INT
IDENT_INCR (
    fulltablename varchar(8187)
)
```

功能说明：

获取自增列 increment

参数说明：

fulltablename：表全名；格式为“模式名.表名”

返回值：

自增列 increment

举例说明：

```
SELECT IDENT_INCR('PRODUCTION.PRODUCT');
```

15. SF_COL_IS_CHECK_KEY

定义：

```
INT
SF_COL_IS_CHECK_KEY (
    key_num     int,
    key_info    varchar(8187),
    col_id      int
)
```

功能说明：

判断一个列是否为CHECK约束列

参数说明：

key_num：约束列总数；

key_info：约束列信息；

col_id：列id

返回值：

返回1：表示该列是check约束列，否则返回0。

举例说明：

```
CREATE TABLE TC (C1 INT, C2 DOUBLE, C3 DATE, C4 VARCHAR, CHECK(C1 < 100 AND C4 IS NOT NULL));
SELECT TBLS.NAME, COLS.NAME, COLS.COLID, COLS.TYPE$, COLS.LENGTH$, COLS.SCALE, COLS.NULLABLE$, COLS.DEFVAL
FROM (SELECT ID, NAME FROM SYS.SYSOBJECTS WHERE NAME = 'TC' AND TYPE$ = 'SCHOBJ' AND SUBTYPE$ = 'UTAB' AND SCHID = (SELECT ID FROM SYS.SYSOBJECTS WHERE NAME = 'SYSDBA' AND TYPE$ = 'SCH')) AS TBLS,
(SELECT ID, PID, INFO1, INFO6 FROM SYS.SYSOBJECTS WHERE TYPE$ = 'TABOBJ' AND SUBTYPE$ = 'CONS') AS CONS_OBJ,SYS.SYSCOLUMNS AS COLS,SYS.SYSCONS AS CONS
WHERE TBLS.ID = CONS_OBJ.PID AND TBLS.ID = COLS.ID AND
SF_COL_IS_CHECK_KEY(CONS_OBJ.INFO1, CONS_OBJ.INFO6, COLS.COLID) = 1 AND
CONS.TABLEID = TBLS.ID AND CONS.TYPE$ = 'C';
```

16. SF_REPAIR_HFS_TABLE

定义：

```
SF_REPAIR_HFS_TABLE (
    SCHNAME VARCHAR(128),
    TABNAME VARCHAR(128)
)
```

功能说明：

HFS表日志属性为LOG NONE时，如果系统出现故障，导致该表数据不一致，则通过该函数修复表数据，保证数据的一致性。

参数说明：

SCHNAME：模式名；

TABNAME：表名；

返回值：

成功返回0，否则报错。

举例说明：

```
CREATE HUGE TABLE T_DM(C1 INT, C2 VARCHAR(20)) LOG NONE;
INSERT INTO T_DM VALUES(99, 'DM7');
COMMIT;
UPDATE T_DM SET C1 = 100;--系统故障
SF_REPAIR_HFS_TABLE('SYSDBA','T_DM');
```

17. SP_ENABLE_EVT_TRIGGER

定义：

```
SP_ENABLE_EVT_TRIGGER (
    SCHNAME VARCHAR(128),
    TRINAME  VARCHAR(128),
ENABLE BOOL
)
```

功能说明：

禁用/启用指定的事件触发器。

参数说明：

SCHNAME：模式名；

TABNAME：触发器名；

ENABLE：　1 表示启用，0 表示禁用。

返回值：

成功返回0，否则报错。

举例说明：

```
SP_ENABLE_EVT_TRIGGER('SYSDBA', 'TRI_1', 1);
SP_ENABLE_EVT_TRIGGER('SYSDBA', 'TRI_1', 1);
```

18. SP_ENABLE_ALL_EVT_TRIGGER

定义：

```
SP_ENABLE_ALL_EVT_TRIGGER (
    ENABLE    BOOL
)
```

功能说明：

禁用/启用数据库上的所有事件触发器

参数说明：

ENABLE：1 表示启用，0 表示禁用。

返回值：

成功返回0，否则报错。

举例说明：

```
SP_ENABLE_ALL_EVT_TRIGGER(1);
SP_ENABLE_ALL_EVT_TRIGGER(0);
```

7）数据守护管理

本小节的存储过程都与 DM 的数据守护功能相关，关于数据守护的概念和相关环境配置与操作可以参考《DM7 系统管理员手册》相关章节。

1. SP_INIT_DW_SYS

定义：

```
void
SP_INIT_DW_SYS(
    create_flag     int
)
```

功能说明：

初始化或清除数据守护观察器的相关系统表

参数说明：

create_flag：为 1 时初始化观察器的相关系统表；为 0 时删除这些系统表

返回值：

无

举例说明：

```
SP_INIT_DW_SYS(1);
```

2. SP_DW_ADD_GROUP

定义：

```
void
SP_DW_ADD_GROUP (
    group_name    varchar(128),
    group_desc    varchar(128)
)
```

功能说明：

增加观察器组

参数说明：

group_name：观察器组名

group_desc：观察器描述

返回值：

无

举例说明：

```
SP_DW_ADD_GROUP('TEST','for TEST');
```

3. SP_DW_DROP_GROUP

定义：

```
void
SP_DW_DROP_GROUP (
    group_name varchar(128)
```

)

功能说明：

删除观察器组

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_DROP_GROUP('TEST');
```

4. SP_DW_ADD_PRIMARY

定义：

```
void
SP_DW_ADD_PRIMARY (
    group_name      varchar(128),
    instance_name   varchar(128)
)
```

功能说明：

增加主机

参数说明：

group_name：观察器组名

instance_name：实例名

返回值：

无

举例说明：

```
SP_DW_ADD_PRIMARY('TEST', 'INST2');
```

5. SP_DW_ADD_STANDBY

定义：

```
void
SP_DW_ADD_STANDBY (
    group_name varchar(128),
    instance_name   varchar(128),
    arch_type       varchar(128),
    arch_timer      varchar(128)
)
```

功能说明：

增加备机

参数说明：

group_name：观察器组名

instance_name：实例名

arch_type：归档类型。取值为 REALTIME/ASYNC/SYNC

arch_timer：归档间隔

返回值：

无

举例说明：

```
SP_DW_ADD_STANDBY('TEST', 'INST3','REALTIME', '');
```

6. SP_DW_DROP_STANDBY

定义：

```
void
SP_DW_DROP_STANDBY (
    group_name varchar(128),
    instance_name    varchar(128)
)
```

功能说明：

删除备机

参数说明：

group_name：观察器组名

instance_name：实例名

返回值：

无

举例说明：

```
SP_DW_DROP_STANDBY('TEST', 'INST3');
```

7. SP_DW_SET_NASFO_INTERVAL

定义：

```
void
SP_DW_SET_NASFO_INTERVAL (
    group_name varchar(128),
    interval        int
)
```

功能说明：

设置同步备机、异步备机检测间隔

参数说明：

group_name：观察器组名

interval：时间间隔，单位 s。取值范围 2~60s，默认值 10 秒

返回值：

无

举例说明：

```
SP_DW_SET_NASFO_INTERVAL('TEST', 30);
```

8. SP_DW_SET_ASFO_INTERVAL

定义：

```
void
SP_DW_SET_ASFO_INTERVAL (
    group_name varchar(128),
    interval        int
)
```

功能说明：

设置实时备机检测间隔

参数说明：

group_name：观察器组名

interval：时间间隔，单位 s。取值范围 2~60s，默认值 10 秒

返回值：

无

举例说明：

```
SP_DW_SET_ASFO_INTERVAL('TEST', 30);
```

9. SP_DW_APPLY_CONFIG

定义：

```
void
SP_DW_APPLY_CONFIG (
    group_name varchar(128)
)
```

功能说明：

应用设置到主备机

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_APPLY_CONFIG('TEST');
```

10. SP_DW_STARTUP

定义：

```
void
SP_DW_STARTUP (
    group_name varchar(128)
)
```

功能说明：

启动数据守护系统

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_STARTUP ('TEST');
```

11. SP_DW_STOP

定义：

```
void
SP_DW_STOP (
    group_name varchar(128)
)
```

功能说明：

停止观察器检测，和 SP_DW_SHUTDOWN 的区别是本过程会 MOUNT 主备

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_STOP('TEST');
```

12. SP_DW_SHUTDOWN

定义：

```
void
SP_DW_SHUTDOWN (
    group_name varchar(128),
)
```

功能说明：

关闭数据守护系统

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_SHUTDOWN ('TEST');
```

13. SP_DW_ADD_STANDBY_DYNAMIC

定义：

```
void
SP_DW_ADD_STANDBY_DYNAMIC (
    group_name varchar(128),
    instance_name    varchar(128),
    arch_type        varchar(128),
    arch_timer       varchar(128)
)
```

功能说明：

动态增加备机

参数说明：

group_name：观察器组名

instance_name：实例名

arch_type：归档类型。取值为 REALTIME/ASYNC/SYNC

arch_timer：归档间隔

返回值：

无

举例说明：

```
SP_DW_ADD_STANDBY_DYNAMIC (TEST', 'INST4','SYNC', '');
```

14. SP_DW_DROP_STANDBY_DYNAMIC

定义：

```
void
SP_DW_DROP_STANDBY_DYNAMIC (
    group_name varchar(128),
    instance_name    varchar(128),
)
```

功能说明：

动态删除备机

参数说明：

group_name：观察器组名

instance_name：实例名

返回值：

无

举例说明：

```
SP_DW_DROP_STANDBY_DYNAMIC ('TEST', 'INS4');
```

15. SP_DW_SWITCHOVER

定义：

```
void
SP_DW_SWITCHOVER (
    group_name varchar(128)
)
```

功能说明：

手动切换主备机

参数说明：

group_name：观察器组名

返回值：

无

举例说明：

```
SP_DW_SWITCHOVER('TEST');
```

16. SP _SET_OGUID

定义：

```
void
SP _SET_OGUID (
    oguid       int
)
```

功能说明：

设置主备机监控组的 ID 号

参数说明：

oguid：oguid

返回值：

无

举例说明：

```
SP _SET_OGUID (451245)
```

17. SF_DW_GET_GROUP_STATUS

定义：

```
INT
SF_DW_GET_GROUP_STATUS (
    group_name varchar(128)
)
```

功能说明：

获取观察器组的监视状态

参数说明：

group_name：观察器组名

返回值：

0：已停止

1：已启动

其他：错误号(<0)

举例说明：

```
SELECT SF_DW_GET_GROUP_STATUS ('TEST');
```

18. SP_DW_CLEAR_HISTORY

定义：

```
VOID
SF_DW_ CLEAR_HISTORY ()
```

功能说明：

清空 DW.SYSDWHISTORY 历史表数据

参数说明：

无

返回值：

无

示例：

```
SELECT SP_DW_CLEAR_HISTORY();
```

19. SP_DW_CLEAR_GROUP_HISTORY

定义：

```
VOID
SP_DW_ CLEAR_GROUP_HISTORY(
    group_name    varchar(128),
)
```

功能说明：

清空 DW.SYSDWHISTORY 历史表中指定组名的数据

参数说明：

group_name：观察器组名

返回值：

无

示例：

```
SELECT SP_DW_CLEAR_GROUP_HISTORY('TEST');     --TEST 为观察组名
```

8）日志与检查点管理

1. CHECKPOINT

定义：

```
INT
CHECKPOINT (
  rate     int
)
```

功能说明：

设置检查点

参数说明：

rate：刷脏页百分比

返回值：

检查点是否成功，0 表示成功，非 0 表示失败

举例说明：

设置刷脏页百分比为 30%的检查点

```
SELECT CHECKPOINT(30);
```

9）事件跟踪与审计

本小节的系统存储过程都必须在审计开关打开（系统 INI 参数 ENABLE_AUDIT=1 或者 2）的情况下，由 AUDITOR 用户进行操作。

1．SP_AUDIT_OBJECT

定义：

```
VOID
SP_AUDIT_OBJECT (
type            varchar(30),
username        varchar (128),
schname  varchar (128),
tvname          varchar (128),
whenever        varchar (20)
)
```

功能说明：

对象级审计选项设置审计

参数说明：

type：审计类型；对象级审计选项

username：用户名

schname：模式名，为空时‘null’

tvname：表视图存储过程名，不能为空

whenever：审计时机；取值范围：ALL/SUCCESSFUL/FAIL

返回值：

无

举例说明：

审计视图的查询操作

```
CREATE TABLE T1_AUDIT(C1 INT);
CREATE VIEW V1_AUDIT AS SELECT * FROM T1_AUDIT;
SP_AUDIT_OBJECT('SELECT', 'NULL', 'SYSDBA', 'V1_AUDIT', 'ALL');
```

审计存储过程的执行操作

```
CREATE PROCEDURE P1_AUDIT AS
BEGIN
SELECT * FROM SYSDBA.T1_AUDIT;
END;

SP_AUDIT_OBJECT('EXECUTE', 'SYSDBA', 'SYSDBA', 'P1_AUDIT', 'ALL');
```

2．SP_AUDIT_STMT

定义：

```
VOID
SP_AUDIT_STMT(
type            varchar(30),
username        varchar (128),
whenever        varchar (20)
```

)

功能说明：

语句级审计选项设置审计

参数说明：

type：审计类型；语句级审计选项

username：用户名

whenever：审计时机；取值范围：ALL/SUCCESSFUL/FAIL

返回值：

无

举例说明：

审计用户登录

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

审计表的创建修改和删除

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

3. SP_NOAUDIT_OBJECT

定义：

```
VOID
SP_NOAUDIT_OBJECT (
type            varchar(30),
username        varchar (128),
schname  varchar (128),
tvname          varchar (128),
whenever        varchar (20)
)
```

功能说明：

对象级审计选项取消审计

参数说明：

type：审计类型；对象级审计选项

username：用户名

schname：模式名，为空时‘null’

tvname：表视图存储过程名，不能为空

whenever：审计时机；取值范围：ALL/SUCCESSFUL/FAIL

返回值：

无

举例说明：

取消审计视图的查询操作

```
SP_NOAUDIT_OBJECT('SELECT', 'NULL', 'SYSDBA', 'V1_AUDIT', 'ALL');
```

取消审计存储过程的执行操作

```
SP_NOAUDIT_OBJECT('EXCUTE', 'SYSDBA', 'SYSDBA', 'P1_AUDIT', 'ALL');
```

4. SP_NOAUDIT_STMT

定义：

```
VOID
SP_NOAUDIT_STMT(
type            varchar(30),
username        varchar (128),
whenever        varchar (20)
```

)

功能说明：

语句级审计选项取消审计

参数说明：

type：审计类型；语句级审计选项

username：用户名

whenever：审计时机；取值范围：ALL/SUCCESSFUL/FAIL

返回值：

无

举例说明：

取消用户登录的审计

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

取消表的创建修改和删除的审计

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

取消表数据的修改和删除的审计

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

5. SP_CREATE_AUDIT_RULE

定义：

```
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
);
```

功能说明：

创建审计实时侵害检测规则

参数说明：

rulename：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
DELECT
UPDATE
EXECUTE
EXECUTE TRIGGER
MERGE INTO
LOCK TABLE
UNLOCK USER
ALTER DATABASE
SAVEPOINT
COMMIT
ROLLBACK
CONNECT
DISCONNECT
EXPLAIN
STARTUP
SHUTDOWN

username：用户名，没有指定时为‘null’表示所有用户
schname：模式名，没有指定时为‘null’
objname：对象名，没有指定时为‘null’
whenever：审计时机；取值范围：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
```

inserval：时间，单位分钟

times：次数

举例说明：

创建审计实时侵害检测规则 RULE1

```
SP_CREATE_AUDIT_RULE ('RULE1','CREATE TABLE', 'NULL', 'NULL', 'NULL', 'ALL', "'192.168.0.1","127.0.0.1"','MON "8:00:00" TO FRI "9:00:00","2011-6-10" "8:00:00" TO "2011-6-10" "9:00:00"',0, 0);
```

6．SP_DROP_AUDIT_RULE

定义：

```
VOID
SP_DROP_AUDIT _RULE(
  rulename VARCHAR(128)
);
```

功能说明：

删除审计实时侵害检测规则

参数说明：

rulename：审计实时侵害检测规则名

举例说明：

删除审计实时侵害检测规则 RULE1

```
SP_DROP_AUDIT_RULE ('RULE1');
```

7．SP_DROP_AUDIT_FILE

定义：

```
VOID
SP_DROP_AUDIT_FILE(
  TIME_STR   VARCHAR(128),
  TYPE       INT
);
```

功能说明：

删除审计记录文件，当前正在使用的审计文件不会被删除。

参数说明：

TIME_STR：时间字符串

TYPE：0，表示删除普通审计文件。1 表示删除实时审计文件

举例说明：

删除 2011-12-6 16:30:00 以前的普通审计文件

```
SP_DROP_AUDIT_FILE('2011-12-6 16:30:00',0);
```

10）数据库重演

1．SP_START_CAPTURE

定义：

```
SP_START_CAPTURE(
    path      varchar(256),
    duration  int
)
```

功能说明：

手工设置负载捕获开始

参数说明：

path：捕获文件保存的绝对路径

duration：捕获持续的时间，如果设置为-1 秒，表示需要手动停止捕获，或者磁盘空间满了自动停止。

返回值：

无

2．SP_STOP_CAPTURE

定义：

SP_STOP_CAPTURE()

功能说明：

手工停止数据库重演捕获

参数说明：

无

返回值：

无

举例说明：

首先对数据库进行备份，然后调用 SP_START_CAPTURE，创建一张表，插入 3 条数据，最后调用 SP_STOP_CAPTURE。利用重演工具 dbreplay 进行数据库重演。

```
CALL SP_START_CAPTURE('D:\dmdbms\data', -1);

CREATE TABLE "SYSDBA"."TEST4REPLAY"
(
 C1 INT,
 C2 VARCHAR(64)
);

INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(1, 'A');
INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(2, 'B');
INSERT INTO "SYSDBA"."TEST4REPLAY" VALUES(3, 'C');
COMMIT;

CALL SP_STOP_CAPTURE();
--然后利用 dbreplay 工具进行数据库重演：
dreplay SERVER=localhost:5236 FILE='D:\dmdbms\data'
```

11）统计信息

以下对象不支持统计信息：1. 外部表、临时表、REMOTE 表、动态视图表、记录类型数组所用的临时表；2. 所在表空间为 OFFLINE 的对象；3. 位图索引，位图连接索引、虚索引、无效的索引、全文索引；4. BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、BOOLEAN 等列类型。

1. SP_TAB_INDEX_STAT_INIT

定义：

SP_TAB_INDEX_STAT_INIT (
schname varchar(128),
tablename varchar(128)
)

功能说明：

对表上所有的索引生成统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

对 SYSOBJECTS 表上所有的索引生成统计信息

```
CALL SP_TAB_INDEX_STAT_INIT ('SYS', 'SYSOBJECTS');
```

2. SP_DB_STAT_INIT

定义：

SP_DB_STAT_INIT (
)

功能说明：

对库上所有模式下的所有用户表上的所有索引生成统计信息

举例说明：

对库上所有模式下的所有用户表上的所有索引生成统计信息

```
CALL SP_DB_STAT_INIT ();
```

3. SP_INDEX_STAT_INIT

定义：

SP_INDEX_STAT_INIT (
schname varchar(128),
indexname varchar(128)
)

功能说明：

对指定的索引生成统计信息

参数说明：

schname：模式名

indexname：索引名

举例说明：

对指定的索引 IND 生成统计信息

```
CALL SP_INDEX_STAT_INIT ('SYSDBA', 'IND');
```

4. SP_COL_STAT_INIT

定义：

SP_COL_STAT_INIT (
schname varchar(128),
tablename varchar(128),
colname varchar(128)
)

功能说明：

对指定的列生成统计信息，不支持大字段列

参数说明：

schname：模式名
tablename：表名
colname：列名

举例说明：

对表 SYSOBJECTS 的 ID 列生成统计信息

```
CALL SP_COL_STAT_INIT ('SYS', 'SYSOBJECTS', 'ID');
```

5. SP_TAB_COL_STAT_INIT

定义：

```
SP_TAB_COL_STAT_INIT (
    schname       varchar(128),
    tablename     varchar(128)
)
```

功能说明：

对某个表上所有的列生成统计信息

参数说明：

schname：模式名
tablename：表名

举例说明：

对'SYSOBJECTS'表上所有的列生成统计信息

```
CALL SP_TAB_COL_STAT_INIT ('SYS', 'SYSOBJECTS');
```

6. SP_TAB_STAT_INIT

定义：

```
SP_TAB_STAT_INIT (
    schname       varchar(128),
    tablename     varchar(128)
)
```

功能说明：

对某张表生成统计信息

参数说明：

schname：模式名
tablename：表名

举例说明：

对表 SYSOBECTS 生成统计信息

```
CALL SP_TAB_STAT_INIT ('SYS', 'SYSOBJECTS');
```

7. SP_SQL_STAT_INIT

定义：

```
SP_SQL_STAT_INIT (
    sql       varchar(8187)
)
```

功能说明：

对某个 SQL 查询语句中涉及的所有表和过滤条件中的列(不包括大字段、ROWID)生成统计信息。

可能返回的错误提示：

1） 语法分析出错，sql 语句语法错误

2） 对象不支持统计信息，统计的表或者列不存在，或者不允许被统计

参数说明：

sql：sql 语句

举例说明：

对'SELECT * FROM SYSOBJECTS'语句涉及的所有表生成统计信息

```
CALL SP_SQL_STAT_INIT ('SELECT * FROM SYSOBJECTS');
```

8. SP_INDEX_STAT_DEINIT

定义：

```
SP_INDEX_STAT_DEINIT (
    schname       varchar(128),
    indexname     varchar(128)
)
```

功能说明：

清空指定索引的统计信息

参数说明：

schname：模式名

indexname：索引名

举例说明：

清空索引 IND 的统计信息

```
CALL SP_INDEX_STAT_DEINIT ('SYSDBA', 'IND');
```

9. SP_COL_STAT_DEINIT

定义：

```
SP_COL_STAT_DEINIT (
    schname       varchar(128),
    indexname     varchar(128),
    colname       varchar(128)
)
```

功能说明：

删除指定列的统计信息

参数说明：

schname：模式名

indexname：索引名

colname：列名

举例说明：

删除 SYSOBJECTS 的 ID 列的统计信息

```
CALL SP_COL_STAT_DEINIT ('SYS', 'SYSOBJECTS', 'ID');
```

10. SP_TAB_COL_STAT_DEINIT

定义：

```
SP_TAB_COL_STAT_DEINIT (
    schname       varchar(128),
    tablename     varchar(128)
)
```

功能说明：

删除表上所有列的统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

删除 SYSOBJECTS 表上所有列的统计信息

```
CALL SP_TAB_COL_STAT_DEINIT ('SYS', 'SYSOBJECTS');
```

11. SP_TAB_STAT_DEINIT

定义：

```
SP_TAB_STAT_DEINIT (
    schname      varchar(128),
    tablename    varchar(128)
)
```

功能说明：

删除某张表的统计信息

参数说明：

schname：模式名

tablename：表名

举例说明：

删除表 SYSOBECTS 的统计信息

```
CALL SP_TAB_STAT_DEINIT ('SYS', 'SYSOBJECTS');
```

12. ET

定义：

```
ET(
ID_IN INT
);
```

功能说明：

统计执行 ID 为 ID_IN 的所有操作符的执行时间。需设置 ini 参数 ENABLE_MONITOR=3

参数说明：

ID_IN：SQL 语句的执行 ID

举例说明：

```
select count (*)from sysobjects;      可以得到 Execute id is 10.
et(10);
```

得到结果：

OP	TIME(MS)	PERCENT	RANK	SEQ
PRJT2	2	2.27%	3	2
FAGR2	13	14.77%	2	3
NSET2	73	82.95%	1	1

12）资源监测

1. SP_CHECK_IDLE_MEM

定义：

```
SP_CHECK_IDLE_MEM (
)
```

功能说明：

对可用内存空间进行检测，并在低于阈值（对应 INI 参数 IDLE_MEM_THRESHOLD）

的情况下打印报警记录到日志，同时报内存不足的异常。

举例说明：

监测当前系统的内存空间是否低于阀值

```
CALL  SP_CHECK_IDLE_MEM ();
```

2. SP_CHECK_IDLE_DISK

定义：

SP_CHECK_IDLE_DISK (
path varchar(256)
)

功能说明：

对指定位置的磁盘空间进行检测，并在低于阈值（对应 INI 参数 IDLE_DISK_THRESHOLD）的情况下打印报警记录到日志，同时报磁盘空间不足的异常。

参数说明：

Path：监测的路径

举例说明：

监测 d:\data 路径下的磁盘空间是否低于阀值

```
CALL  SP_CHECK_IDLE_DISK ('d:\data');
```

3. SF_GET_CMD_RESPONSE_TIME

定义：

SF_GET_CMD_RESPONSE_TIME()

功能说明：

查看 DM 服务器对用户命令的平均响应时间

返回值：

命令的平均响应时间，单位秒

举例说明：

在 dm.ini 中 ENABLE_MONITOR=2 的前提下执行

```
SELECT SYS.SF_GET_CMD_RESPONSE_TIME();
```

4. SF_GET_TRX_RESPONSE_TIME

定义：

SF_GET_TRX_RESPONSE_TIME()

功能说明：

查看事务的平均响应时间

返回值：

事务的平均响应时间，单位秒

举例说明：

在 dm.ini 中 ENABLE_MONITOR=1 或=2 的前提下执行

```
SELECT SYS.SF_GET_TRX_RESPONSE_TIME();
```

5. SF_GET_DATABASE_TIME_PER_SEC

定义：

SF_GET_DATABASE_TIME_PER_SEC()

功能说明：

查看数据库中用户态时间占总处理时间的比值

返回值：

用户态时间占总处理时间的比值，该比值越大表明用于 IO、事务等待等耗费的时间越少

举例说明：

在 dm.ini 中 ENABLE_MONITOR=2 的前提下执行

```
SELECT SF_GET_DATABASE_TIME_PER_SEC();
```

6. TABLE_USED_SPACE

定义：

```
INT *
TABLE_USED_SPACE (
    schname    varchar(256);
    tabname    varchar(256)
)
```

功能说明：

获取指定表所占用的空间大小

参数说明：

schname：模式名

tabname：表名

返回值：

表所占用的页数

举例说明：

查看 SYSOJBECTS 的所占用的空间大小

```
SELECT    TABLE_USED_SPACE ('SYS','SYSOBJECTS');
```

查询结果：

```
32
```

7. USER_USED_SPACE

定义：

```
INT *
USER_USED_SPACE (
    username    varchar(256)
)
```

功能说明：

获取指定用户所占用的空间大小

参数说明：

username：用户名

返回值：

用户所占用的页数

举例说明：

查看 SYSDBA 的所占用的空间大小

```
SELECT    USER_USED_SPACE ('SYSDBA');
```

查询结果：

```
64
```

8. TS_USED_SPACE

定义：

```
INT *
TS_USED_SPACE (
```

```
    tsname    varchar(256)
)
```

功能说明：

获取指定表空间所占用的空间大小

参数说明：

tsname：表空间名

返回值：

表空间占用的页数

举例说明：

查看 MAIN 表空间的所占用的空间大小

```
SELECT    TS_USED_SPACE ('MAIN');
```

查询结果：

```
16384
```

9. DB_USED_SPACE

定义：

```
INT *
DB_USED_SPACE ()
```

功能说明：

获取整个数据库占用的空间大小

返回值：

整个数据库占用的页数

举例说明：

查看数据库所占用的空间大小

```
SELECT    DB_USED_SPACE ();
```

查询结果：

```
19712
```

10. INDEX_USED_SPACE

定义：

```
INT *
INDEX _USED_SPACE (
    indexid    int
)
```

功能说明：

获取指定索引所占用的空间大小

参数说明：

indexid：索引 ID

返回值：

索引占用的页数

举例说明：

查看索引号为 33554540 的索引所占用的空间大小

```
SELECT    INDEX_USED_SPACE (33554540);
```

查询结果：

```
32
```

11. INDEX_USED_PAGES

定义：

INT *
INDEX _USED_PAGES (
indexid int
)

功能说明：

获取指定索引已使用的空间大小

参数说明：

indexid：索引 ID

返回值：

索引已使用的页数

举例说明：

查看索引号为 33554540 的索引已使用的空间大小

```
SELECT INDEX_USED_PAGES (33554540);
```

查询结果：

```
14
```

12. TABLE_USED_PAGES

定义：

INT *
TABLE_USED_PAGES (
schname varchar(256);
tabname varchar(256)
)

功能说明：

获取指定表已使用的空间大小

参数说明：

schname：模式名
tabname：表名

返回值：

表已使用的页数

举例说明：

查看 SYSOJBECTS 已使用的空间大小

```
SELECT TABLE_USED_PAGES('SYS','SYSOBJECTS');
```

查询结果：

```
14
```

13）类型别名

DM 支持用户对各种基础数据类型定义类型别名，定义的别名可以在 SQL 语句和 PL/SQL 中使用。

1. SP_INIT_DTYPE_SYS

定义：

SP_INIT_DTYPE_SYS (
create_flag int
)

功能说明：

初始化或清除类型别名运行环境

参数说明：

create_flag：1 表示创建，0 表示删除

返回值：

无

举例说明：

初始化类型别名运行环境

```
CALL SP_INIT_DTYPE_SYS(1);
```

2. SF_CHECK_DTYPE_SYS

定义：

int
SF_CHECK_DTYPE_SYS ()

功能说明：

系统类型别名系统的启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统类型别名系统的启用状态

```
SELECT SF_CHECK_SYSTEM_PACKAGES;
```

3. SP_DTYPE_CREATE

定义：

```
SP_DTYPE_CREATE (
name        varchar(32),
base_name   varchar(32),
len         int,
scale       int
)
```

功能说明：

创建一个类型别名。最多只能创建 100 个类型别名

参数说明：

name：类型别名的名称

base_name：基础数据类型名

len：基础数据类型长度，当为固定精度类型时，应该设置为 NULL

scale：基础数据类型刻度

返回值：

无

举例说明：

创建 VARCHAR(100)的类型别名‘STR’

```
CALL SP_DTYPE_CREATE('STR', 'VARCHAR', 100, NULL);
```

4. SP_DTYPE_DELETE

定义：

```
SP_DTYPE_DELETE (
name        varchar(32)
)
```

功能说明：

删除一个类型别名

参数说明：

name：类型别名的名称

返回值：

无

举例说明：

删除类型别名‘STR’

```
CALL SP_DTYPE_DELETE('STR');
```

14）杂类函数

1. TO_DATETIME

定义：

```
CHAR *
TO_DATETIME(
    year int,
    month int,
    day int,
    hour int,
    minute int
)
```

功能说明：

将 int 类型值组合并转换成日期时间类型

参数说明：

year：年份

month：月份

day：日

hour：小时

minute：分钟

返回值：

日期时间值

举例说明：

将整型数 2010，2，2，5，5 转换成日期时间类型

```
SELECT   TO_DATETIME (2010,2,2,5,5);
```

查询结果：

```
2010-02-02 05:05:00.000000
```

2. SP_SET_ROLE

定义：

```
int
SP_SET_ROLE (
    ROLE_NAME     varchar(128),
    ENABLE        int
)
```

功能说明：

设置角色启用禁用

参数说明：

ROLE_NAME: 角色名。

ENABLE：0/1 (0：禁用 1：启用)

返回值：

无

举例说明：

禁用角色 ROLE1

```
SP_SET_ROLE ( 'ROLE1', 0);
```

3. SP_CREATE_SYSTEM_VIEWS

定义

```
int
SP_CREATE_SYSTEM_VIEWS (
CREATE_FLAG          int
)
```

功能说明：

创建或删除系统视图。系统视图包括：USER_MVIEWS、USER_OBJECTS、USER_SOURCE、USER_TABLES、USER_TAB_COLUMNS、USER_CONSTRAINTS、USER_SYS_PRIVS、ER_TAB_PRIVS、USER_COL_PRIVS、USER_ROLE_PRIVS、USER_INDEXES、USER_IND_COLUMNS、USER_CONS_COLUMNS、DBA_OBJECTS、DBA_DATA_FILES、DBA_DB_LINKS、DBA_INDEXES、DBA_IND_COLUMNS、DBA_SEQUENCES、DBA_SYNONYMS、DBA_TABLES、DBA_TABLESPACES、DBA_TAB_COLUMNS、DBA_USERS、DBA_VIEWS、DBA_SYS_PRIVS、DBA_TAB_PRIVS、DBA_COL_PRIVS、DBA_ROLE_PRIVS、SESSION_PRIVS、SESSION_ROLES、YSAUTH$。

参数说明：

CREATE_FLAG：为 1 时表示创建视图；为 0 表示删除视图

返回值：

无

举例说明：

创建系统视图

```
SP_CREATE_SYSTEM_VIEWS(1);
```

4. SF_CHECK_SYSTEM_VIEWS

定义：

```
int
SF_CHECK_SYSTEM_VIEWS()
```

功能说明：

系统视图的启用状态检测。

返回值：

0：未启用；1：已启用

举例说明：

获得系统视图的启用状态

```
SELECT SF_CHECK_SYSTEM_VIEWS;
```

5. SP_DYNAMIC_VIEW_DATA_CLEAR

定义：

```
void
SP_DYNAMIC_VIEW_DATA_CLEAR  (
```

```
    VIEW_NAME      varchar(128)
)
```

功能说明：

清空动态性能视图的历史数据，仅对存放历史记录的动态视图起作用。

参数说明：

VIEW_NAME：动态性能视图名

返回值：

无

举例说明：

清空动态性能视图 V$SQL_HISTORY 的历史数据

```
SP_DYNAMIC_VIEW_DATA_CLEAR ( 'V$SQL_HISTORY');
```

6. SP_INIT_DBG_SYS

定义：

```
void
SP_INIT_DBG_SYS(
    CREATE_FLAG          int
)
```

功能说明：

创建或删除 DBMS_DBG 系统包

参数说明：

CREATE_FLAG：为 1 时表示创建 DBMS_DBG 包；为 0 表示删除该系统包

返回值：

无

举例说明：

创建 DBMS_DBG 系统包

```
SP_INIT_DBG_SYS(1);
```

7. SF_CHECK_DBG_SYS

定义：

```
int
SF_CHECK_DBG_SYS ()
```

功能说明：

系统的 DBG 系统包启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得 DBG 系统包的启用状态

```
SELECT SF_CHECK_DBG_SYS;
```

8. SP_INIT_GEO_SYS

定义：

```
void
SP_INIT_GEO_SYS(
    CREATE_FLAG          int
)
```

功能说明：

创建或删除 DBMS_GEO 系统包

参数说明：

CREATE_FLAG：为 1 时表示创建 DBMS_GEO 包；为 0 表示删除该系统包

返回值：

无

举例说明：

创建 DBMS_GEO 系统包

```
SP_INIT_GEO_SYS(1);
```

9. SF_CHECK_GEO_SYS

定义：

```
int
SF_CHECK_GEO_SYS ()
```

功能说明：

系统的 GEO 系统包启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得 GEO 系统包的启用状态

```
SELECT SF_CHECK_GEO_SYS;
```

10. SP_INIT_JOB_SYS

定义：

```
void
SP_INIT_JOB_SYS (
    CREATE_FLAG         int
)
```

功能说明：

创建或删除存储作业相关的对象、历史记录等信息的表以及 DBMS_JOB 系统包

参数说明：

CREATE_FLAG：为 1 时表示创建存储作业相关的对象、历史记录等信息的表以及 DBMS_JOB 系统包；为 0 表示删除这些表和 DBMS_JOB 包

返回值：

无

举例说明：

创建存储作业相关的对象、历史记录等信息的表以及 DBMS_JOB 系统包

```
SP_INIT_JOB_SYS (1);
```

11. SP_CREATE_SYSTEM_PACKAGES

定义：

```
void
SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG         int
)
```

功能说明：

创建或删除除了 DBMS_DBG、DBMS_GEO 和 DBMS_JOB 以外的所有系统包

参数说明：

CREATE_FLAG：为 1 时表示创建除了 DBMS_DBG、DBMS_GEO 和

DBMS_JOB 以外的所有系统包；为 0 表示删除这些系统包

返回值：

无

举例说明：

创建除了 DBMS_DBG、DBMS_GEO 和 DBMS_JOB 以外的所有系统包

```
SP_CREATE_SYSTEM_PACKAGES (1);
```

12. SF_CHECK_SYSTEM_PACKAGES

定义：

```
int
SF_CHECK_SYSTEM_PACKAGES（）
```

功能说明：

系统包的启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统包的启用状态

```
SELECT SF_CHECK_SYSTEM_PACKAGES;
```

13. SP_INIT_INFOSCH

定义：

```
void
SP_INIT_INFOSCH (
    CREATE_FLAG         int
)
```

功能说明：

创建或删除信息模式

参数说明：

CREATE_FLAG：为 1 时表示创建信息模式；为 0 表示删除信息模式

举例说明：

创建信息模式

```
SP_INIT_INFOSCH (1);
```

14. SF_CHECK_INFOSCH

定义：

```
int
SF_CHECK_INFOSCH ()
```

功能说明：

系统的信息模式启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统信息模式的启用状态

```
SELECT SF_CHECK_INFOSCH;
```

15. SP_INIT_CPT_SYS

定义：

```
int
```

SP_ INIT_CPT_SYS (
FLAG int
)

功能说明：

初始化数据捕获环境

参数说明：

FLAG： 1 表示初始化环境；0 表示删除环境

举例说明：

初始化数据捕获环境

```
SP_INIT_CPT_SYS(1);
```

16. SF_CHECK_CPT_SYS

定义：

int
SF_CHECK_CPT_SYS ()

功能说明：

系统的数据捕获环境启用状态检测

返回值：

0：未启用；1：已启用

举例说明：

获得系统数据捕获环境的启用状态

```
SELECT SF_CHECK_CPT_SYS;
```

15）存储加密函数

1. CFALGORITHMSENCRYPT

定义：

CFALGORITHMSENCRYPT(
src varchar,
algorithm int,
key varchar
)

参数说明：

src：被加密的数据，数据类型为 VARCHAR
algorithm：采用的加密算法，INT 类型，不可以为 NULL；
key：采用的密钥，VARCHAR 类型，不可以为 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 VARCHAR

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列表中。

2．CFALGORITHMSDECRYPT

定义：

CFALGORITHMSDECRYPT(
src varchar,
algorithm int,
key varchar
)

参数说明：

src：密文，数据类型为 VARCHAR
algorithm：采用的加密算法，INT 类型，不可以为 NULL；
key：采用的密钥，VARCHAR 类型，不可以为 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 CFALGORITHMSDECRYPT(c1, 514, '仅供测试使用') FROM enc_001;
```

查询结果：

```
tt
```

3．SF_ENCRYPT_BINARY

定义：

SF_ENCRYPT_BINARY(
src varbinary,
algorithm int,
key varchar
)

参数说明：

src：被加密的数据，数据类型为 VARBINARY
algorithm：采用的加密算法，INT 类型，不可以为 NULL；
key：采用的密钥，VARCHAR 类型，不可以为 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'));
```

这样就将加密后的数据存放到列表中。

4．SF_DECRYPT_TO_BINARY

定义：

SF_DECRYPT_TO_BINARY(
src varbinary,
algorithm int,
key varchar
)

参数说明：

src：密文，数据类型为 VARBINARY

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_002;
```

查询结果：

```
0x12345678EF
```

5．SF_ENCRYPT_CHAR

定义：

```
SF_ENCRYPT_CHAR(
    src varchar,
    algorithm int,
    key varchar
)
```

参数说明：

src：被加密的数据，数据类型为 VARCHAR/CHAR

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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'));
```

这样就将加密后的数据存放到列表中。

6．SF_DECRYPT_TO_CHAR

定义：

```
SF_DECRYPT_TO_CHAR(
    src varbinary,
    algorithm int,
    key varchar
)
```

参数说明：

src：密文，数据类型为 VARBINARY

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_003;
```

查询结果：

```
测试数据
```

7．SF_ENCRYPT_DATE

定义：

```
SF_ENCRYPT_DATE(
    src date,
    algorithm int,
    key varchar
)
```

参数说明：

src：被加密的数据，数据类型为 DATE

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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'));
```

这样就将加密后的日期类型数据存放到列表中。

8．SF_DECRYPT_TO_DATE

定义：

```
SF_DECRYPT_TO_DATE(
    src varbinary,
    algorithm int,
    key varchar
)
```

参数说明：

src：密文，数据类型为 VARBINARY

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_004;
```

查询结果：

```
2011-1-1
```

9．SF_ENCRYPT_DATETIME

定义：

```
SF_ENCRYPT_DATETIME(
    src datetime,
    algorithm int,
    key varchar
)
```

参数说明：

src：被加密的数据，数据类型为 DATETIME

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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'));
```

这样就将加密后的日期时间类型数据存放到列表中。

10．SF_DECRYPT_TO_DATETIME

定义：

```
SF_DECRYPT_TO_DATETIME(
    src varbinary,
    algorithm int,
    key varchar
)
```

参数说明：

src：密文，数据类型为 VARBINARY

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_005;
```

查询结果：

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

11．SF_ENCRYPT_DEC

定义：

```
SF_ENCRYPT_DEC(
```

src dec,
algorithm int,
key varchar
)

参数说明：

src：被加密的数据，数据类型为 DEC

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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'));
```

这样就将加密后的 DEC 数据存放到列表中。

12．SF_DECRYPT_TO_DEC

定义：

SF_DECRYPT_TO_DEC(
src varbinary,
algorithm int,
key varchar
)

参数说明：

src：密文，数据类型为 VARBINARY

algorithm：采用的加密算法，INT 类型，不可以为 NULL；

key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_006;
```

查询结果：

```
3.141590000000
```

13．SF_ENCRYPT_TIME

定义：

SF_ENCRYPT_TIME(
src time,
algorithm int,
key varchar
)

参数说明：

src：被加密的数据，数据类型为 TIME
algorithm：采用的加密算法，INT 类型，不可以为 NULL；
key：采用的密钥，VARCHAR 类型，不可以为 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
'));
```

这样就将加密后的时间类型数据存放到列表中。

14. SF_DECRYPT_TO_TIME

定义：

```
SF_DECRYPT_TO_TIME(
    src varbinary,
    algorithm int,
    key varchar
)
```

参数说明：

src：密文，数据类型为 VARBINARY
algorithm：采用的加密算法，INT 类型，不可以为 NULL；
key：采用的密钥，VARCHAR 类型，不可以为 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') FROM enc_007;
```

查询结果：

```
12:12:12.0
```

16）编目函数调用的系统函数

1. SF_GET_BUFFER_LEN

定义：

```
SF_GET_BUFFER_LEN(
    name varchar,
    length int,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR
length：列的精度，INT 类型；

scale：列的刻度，INT 类型；

功能说明：

根据某列数据类型名 name 和列的精度 length 刻度 scale 获取该列存贮在硬盘上的长度。

返回值：

列存贮长度，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_BUFFER_LEN('VARCHAR',3,2);
```

查询结果：

```
3
```

2. SF_GET_DATA_TYPE

定义：

```
SF_GET_DATA_TYPE(
    name varchar,
    scale int,
    version int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：时间间隔类型刻度，INT 类型；

version：版本号，INT 类型。

功能说明：

根据数据类型关键字获取对应的 SQL 数据类型。

返回值：

数据类型值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_DATA_TYPE('VARCHAR',3,2);
```

查询结果：

```
12
```

3. SF_GET_DATE_TIME_SUB

定义：

```
SF_GET_DATE_TIME_SUB(
    name varchar,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：类型刻度，INT 类型；

功能说明：

获得时间类型的子类型。

返回值：

时间类型子类型值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_DATE_TIME_SUB('datetime',2);
```

查询结果：

```
3
```

4. SF_GET_DECIMAL_DIGITS

定义：

```
SF_GET_DECIMAL_DIGITS(
    name varchar,
    scale int
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

scale：预期类型刻度，INT 类型；

功能说明：

根据某数据类型名和预期的刻度获取该数据类型的实际刻度。

返回值：

类型实际刻度值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_DECIMAL_DIGITS('INT',2);
```

查询结果：

```
0
```

5. SF_GET_SQL_DATA_TYPE

定义：

```
SF_GET_SQL_DATA_TYPE(
    name varchar
)
```

参数说明：

name：某列数据类型名，数据类型为 VARCHAR

功能说明：

根据某数据类型名返回该数据类型的 SQL 数据类型值

返回值：

SQL 数据类型值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_SQL_DATA_TYPE('INT');
```

查询结果：

```
4
```

6. SF_GET_SYS_PRIV

定义：

```
SF_GET_SYS_PRIV(
    privid int
)
```

参数说明：

privid：数据类型为 INT。取值范围：数据库权限 4096-4241，对象权限 8192-8198。除了 4123、4204 和 4205。

功能说明：

获得 privid 所代表的权限操作。

返回值：

前导字符串，数据类型为 VARCHAR

举例说明：

执行：

```
SELECT SF_GET_SYS_PRIV(4096);
SELECT SF_GET_SYS_PRIV(4099);
```

查询结果：

```
CREATE DATABASE
CREATE LOGIN
```

7. SF_GET_OCT_LENGTH

定义：

```
SF_GET_OCT_LENGTH(
    name varchar,
    length int
)
```

参数说明：

name：数据类型名，数据类型为 VARCHAR

length：类型长度，INT 类型；

功能说明：

返回变长数据类型的长度。

返回值：

类型长度，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_OCT_LENGTH('VARCHAR',3);
```

查询结果：

```
3
```

8. SF_GET_COLUMN_SIZE

定义：

```
SF_GET_COLUMN_SIZE(
    name varchar,
    length int,
    scale int
)
```

参数说明：

name：数据类型名，数据类型为 VARCHAR

length：类型长度，数据类型为 INT；

scale：时间类型刻度，INT 类型；

功能说明：

返回列的大小。

返回值：

类型长度，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_COLUMN_SIZE('INT',2,0);
```

查询结果：

```
10
```

9. SF_GET_TABLES_TYPE

定义：

```
SF_GET_TABLES_TYPE(
    type varchar
)
```

参数说明：

type：表的类型名，数据类型为 VARCHAR

功能说明：

返回表类型名。

返回值：

表类型名，数据类型为 VARCHAR

举例说明：

执行：

```
SELECT SF_GET_TABLES_TYPE('UTAB');
```

查询结果：

```
TABLE
```

10. SF_GET_SCHEMA_NAME_BY_ID

定义：

```
SF_GET_SCHEMA_NAME_BY_ID(
    schid int
)
```

参数说明：

schid：模式 ID，INT 类型；

功能说明：

返回模式名。

返回值：

模式名，数据类型为 VARCHAR

举例说明：

执行：

```
SELECT SF_GET_SCHEMA_NAME_BY_ID(150994945);
```

查询结果：

```
SYSDBA
```

11. SF_COL_IS_IDX_KEY

定义：

```
SF_COL_IS_IDX_KEY(
    key_num int,
    key_info varbinary,
    col_id int
)
```

参数说明：

key_num：键值个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型；

功能说明：

判断所给的 colid 是否是 index key。

返回值：

是否索引键，数据类型为 INT

举例说明：

执行：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1334
SELECT INDS.KEYNUM, INDS.KEYINFO, SF_COL_IS_IDX_KEY(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM (SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS, SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1334 AND OBJ_INDS.PID = 1334 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
1   000041   1        --ID1
0            0        --聚集索引
```

12. SF_GET_INDEX_KEY_ORDER

定义：

```
SF_GET_INDEX_KEY_ORDER(
    key_num int,
    key_info varbinary,
    col_id int
)
```

参数说明：

key_num：索引键个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型；

功能说明：

获得当前 column 的 key 的排序。

返回值：

类型长度，数据类型为 VARCHAR

举例说明：

执行：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1135
SELECT SF_GET_INDEX_KEY_ORDER(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM (SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS, SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1135 AND OBJ_INDS.PID = 1135 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
A
NULL
```

13. SF_GET_INDEX_KEY_SEQ

定义：

```
SF_GET_INDEX_KEY_SEQ(
```

key_num int,
key_info varbinary,
col_id int
)

参数说明：

key_num：索引键个数，数据类型为 INT

key_info：键值信息，数据类型为 VARBINARY

col_id：列 ID，INT 类型；

功能说明：

获得当前 column 所在的 key 序号。

返回值：

当前列 KEY 序号，数据类型为 INT

举例说明：

执行：

```
CREATE TABLE TT1(C1 INT);
CREATE INDEX ID1 ON TT1(C1);
SELECT ID FROM SYSOBJECTS WHERE NAME LIKE 'TT1';//1135
SELECT SF_GET_INDEX_KEY_SEQ(INDS.KEYNUM, INDS.KEYINFO, COLS.COLID) FROM (SELECT ID,PID,NAME FROM SYSOBJECTS WHERE SUBTYPE$='INDEX') AS OBJ_INDS, SYSCOLUMNS AS COLS, SYSINDEXES AS INDS WHERE COLS.ID = 1135 AND OBJ_INDS.PID = 1135 AND INDS.ID = OBJ_INDS.ID;
```

查询结果：

```
1
-1
```

14. SF_GET_UPD_RULE

定义：

SF_GET_UPD_RULE(
rule varchar(2)
)

参数说明：

rule：规则名，数据类型为 VARCHAR，rule 参数只会用到 rule[0]，取值为’’/’C’/’N’/’D’，分别表示 no act/cascade/set null/set default。

功能说明：

解析在 SYSCONS 中 faction 中保存的外键更新规则。

返回值：

外键更新规则值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_UPD_RULE('C');
```

查询结果：

```
0
```

15. SF_GET_DEL_RULE

定义：

SF_GET_DEL_RULE(
rule varchar(2)
)

参数说明：

rule：规则名，数据类型为 VARCHAR，rule 参数只会用到 rule[1]，取值为''/'C'/'N'/'D'，分别表示 no act/cascade/set null/set default。

功能说明：

解析在 SYSCONS 中 faction 中保存的外键删除规则。

返回值：

外键删除规则值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_DEL_RULE('AC');
```

查询结果：

```
0
```

16. SF_GET_OLEDB_TYPE

定义：

```
SF_GET_OLEDB_TYPE(
    name varchar
)
```

参数说明：

name：类型名，数据类型为 VARCHAR

功能说明：

获得 OLEDB 的数据类型长度。

返回值：

数据类型值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_OLEDB_TYPE('INT');
```

查询结果：

```
3
```

17. SF_GET_OLEDB_TYPE_PREC

定义：

```
SF_GET_OLEDB_TYPE_PREC(
    name varchar,
    length int    )
```

参数说明：

name：类型名，数据类型为 VARCHAR

length：类型长度，数据类型为 INT

功能说明：

获得 OLEDB 的数据类型的精度。

返回值：

数据类型的精度值，数据类型为 INT

举例说明：

执行：

```
SELECT SF_GET_OLEDB_TYPE_PREC('INT',2);
```

查询结果：

```
10
```

18. SP_GET_TABLE_COUNT

定义：

SP_GET_TABLE_COUNT(
table_id int)

参数说明：

table_id：类型长度，数据类型为 INT

功能说明：

获得表行数。

返回值：

表的行数，数据类型为 INT

举例说明：

执行：

```
SELECT SP_GET_TABLE_COUNT(1097);
```

查询结果：

```
69
```

19. SF_OLEDB_TYPE_IS_LONG

定义：

SF_OLEDB_TYPE_IS_LONG(
typename varchar)

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否较长。

返回值：

0 或者 1，数据类型为 INT

举例说明：

执行：

```
SELECT SF_OLEDB_TYPE_IS_LONG('IMAGE');
SELECT SF_OLEDB_TYPE_IS_LONG('INT');
```

查询结果：

```
1
0
```

20. SF_OLEDB_TYPE_IS_BESTMATCH

定义：

SF_OLEDB_TYPE_IS_BESTMATCH(
typename varchar
)

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否精确匹配类型。

返回值：

0 或者 1，数据类型为 INT

举例说明：

执行：

```
SELECT SF_OLEDB_TYPE_IS_BESTMATCH('INT');
SELECT SF_OLEDB_TYPE_IS_BESTMATCH('IMAGE');
```

查询结果：

```
1
0
```

21. SF_OLEDB_TYPE_IS_FIXEDLEN

定义：

```
SF_OLEDB_TYPE_IS_FIXEDLEN(
    typename varchar    )
```

参数说明：

typename：类型名，数据类型为 VARCHAR

功能说明：

判断类型是否为定长类型。

返回值：

0 或者 1，数据类型为 INT

举例说明：

执行：

```
SELECT SF_OLEDB_TYPE_IS_FIXEDLEN('INT');
SELECT SF_OLEDB_TYPE_IS_FIXEDLEN('IMAGE');
```

查询结果：

```
1
0
```

附录 4 DM 技术支持

如果您在安装或使用 DM 及其相应产品时出现了问题，请首先访问我们的 Web 站点 http://www.dameng.com/。在此站点我们收集整理了安装使用过程中一些常见问题的解决办法，相信会对您有所帮助。

您也可以通过以下途径与我们联系，我们的技术支持工程师会为您提供服务。

达梦数据库（武汉）有限公司

地址：武汉市关山一路特 1 号光谷软件园 C6 栋 5 层

邮编：430073

电话：(+86)027-87588000

传真：(+86)027-87588000-8039

达梦数据库（北京）有限公司

地址：北京市海淀区北三环西路 48 号数码大厦 B 座 905

邮编：100086

电话：(+86)010-51727900

传真：(+86)010-51727983

达梦数据库（上海）有限公司

地址：上海市闸北区江场三路 28 号 301 室

邮编：200436

电话：(+86)021-33932716

传真：(+86)021-33932718

地址：上海市浦东张江高科技园区博霞路 50 号 403 室

邮编：201203

电话：（+86）021-33932717

传真：（+86）021-33932717-801

达梦数据库（广州）有限公司

地址：广州市荔湾区中山七路 330 号荔湾留学生科技园 703 房

邮编：510145

电话：(+86)020-38371832

传真：(+86)020-38371832

达梦数据库（海南）有限公司

地址：海南省海口市玉沙路富豪花园 B 座 1602 室

邮编：570125

电话：(+86)0898-68533029

传真：(+86)0898-68531910

达梦数据库（南宁）办事处

地址：广西省南宁市科园东五路四号南宁软件园五楼

邮编：530003
电话：（+86）0771-2184078
传真：（+86）0771-2184080

达梦数据库（合肥）办事处
地址：合肥市包河区马鞍山路金帝国际城 7 栋 3 单元 706 室
邮编：230022
电话：（+86）0551-3711086

达梦数据库（深圳）办事处
地址：深圳市福田区皇岗路高科利大厦 A 栋 24E
邮编：518033
电话：0755-83658909
传真：0755-83658909

技术服务：
电话：400-648-9899
邮箱：tech@dameng.com