DBMaster

SQL命令与函数参考手册

SYSCOM

SYSCOM Computer Engineering Co./Corporate Headquarters

B1, 2-7F No. 115 Emei Street, Wanhua District,
Taipei City 108, Taiwan (R.O.C.)

www.dbmaker.com

www.dbmaker.com.tw/service

Document No.645049-237378/DBM54CN-M03312017-SQLR

发行日期: 2017-03-31

对于本手册中没有体现的关于产品最新功能的描述，请在安装完成SYSCOM DBMaster 软件后阅读 README.TXT 文件。

注意事项
本手册中有关软件的描述，均以该软件所提供的使用许可为基础。

对于授权许可的详细信息，请与您的经销商联系。关于计算机产品的特殊用途的市场性与适用性，经销商不会给予任何说明和保证。因外界因素如地震、过热、过冷和潮湿而引起产品的任何损坏以及由于使用不正确的电压和不兼容的软硬件而引起的损失和损坏，经销商概不负责。

虽然该手册的内容已经过仔细核对，但错误再所难免。若手册再有改动，不另行通知。还请见谅。

目录

1 简介

欢迎使用DBMaster SQL命令与函数参考手册。DBMaster是一个功能强大且使用灵活的SQL数据库管理系统（DBMS），它支持交互式的结构化查询语言（SQL），Microsoft开放式数据库连接（ODBC）标准接口以及嵌入式的ESQL/C语言。DBMaster也支持Java工具的接口，针对COBOL语言的DCI接口。由于DBMaster完全遵循开放式的架构，以及标准ODBC接口，而市面上的绝大多数开发工具，因为都支持标准ODBC，所以您可以有更多的选择用来编译自定义应用程序。

DBMaster可以很容易地从个人使用的“单用户数据库”升级到企业级的“分布式数据库”。无论您的数据库是何种结构，DBMaster先进的安全性、完整性和可靠性都能保证用户重要数据的安全。另外，DBMaster的跨平台支持特性则在您需要作硬件升级时，提供给您最佳的扩展弹性。

DBMaster提供了卓越的多媒体处理能力，可以让您存储、查询、恢复和操纵各种类型的多媒体数据。利用DBMaster提供的二进制大型对象（BLOB），可以让您的多媒体数据完全享有DBMaster先进的安全性和灾难恢复功能。而利用文件对象（File Object）数据类型，也可以让您的应用程序能够直接编辑DBMaster数据库的外部文件。

1.1 其它相关文件

除了本手册外，DBMaster还提供了一整套的数据库管理系统（DBMS）手册，您可以根据特定需要参考以下手册。

- 有关DBMaster的性能和特征，请参考*DBMaster指南*。
- 有关设计、管理和维护DBMaster数据库的信息，请参考*数据库管理员手册*。
- 有关如何管理DBMaster的信息，请参考*服务器管理工具用户手册*。
- 有关DBMaster的配置信息，请参考*配置管理工具用户手册*。
- 有关DBMaster功能的信息，请参考*数据库管理工具用户手册*。
- 有关dmSQL命令行工具的使用方法，请参考*dmSQL使用手册*。
- 有关DCI COBOL接口的详细信息，请参考*DBMaster DCI用户手册*。
- 有关嵌入式ESQL/C语言的语法和使用，请参考*ESQL/C程序员参考手册*。
- 有关ODBC API及JDBC API的信息，请参考*ODBC程序员参考手册*及*JDBC程序员参考手册*。
- 有关DBMaster的错误信息及警告信息，请参考*错误信息参考手册*。
- 有关SQL存储过程的使用方法，请参考*SQL存储过程用户手册*。

1.2 技术支持

在软件评估期间，Syscom Computer Engineering Co.(“Syscom”)会为您提供30天的免费email支持和电话支持。当软件注册后，我们还会再为您提供30天的免费技术支持。因此，您就可以获得60天的免费技术支持。在30天或60天的免费技术支持到期后，Syscom仍将继续通过电子邮件的方式为您提供技术支持。

您除了可以获得免费的技术支持外，还可以以20%的零售价购买其它产品。具体详情和价格请与sales@dbmaker.com保持联系。

您可以通过任何一种方式（普通信件、电话或email）与Syscom技术支持保持联系，请登录至：www.casemaker.com/support 以获取详细信息。我们建议您在联系Syscom技术支持之前，请先查询当前数据库的常见问题解答。

无论您以何种方式与Syscom的技术支持联系时，请务必写上以下有效信息：

- 产品的名称及版本号
- 注册号
- 注册的客户名和地址
- 供应商/发行者的地址
- 系统平台及计算机系统配置
- 错误信息出现前执行的详细操作
- 如果可以，请提供错误信息和编号
- 其它相关信息

1.3 文档协定

为方便用户的阅读和使用，本手册使用了一种标准的排版约定：注释、程序、示例和命令行都用缩进排版的方式进行了特别的设置。

协定	描述
斜体字	斜体字指必须提供的信息，如用户名、表名等。以斜体字出现的文字是应该用实际的名字来替代的。除此之外，斜体字还可用于引入新单词，偶尔也在文档中使用斜体字以示强调。
黑体字	黑体字表示文件名、数据库名、表名称、字段名、用户名和其它数据库对象。它也用于强调程序执行步骤中的菜单命令。
关键字	文字段落中，SQL语言使用的关键字都是以大写字母出现的。
小型大写字母	小型大写字母指的是键盘上的某个键。两个键之间的加号(+) 说明在按第二个键的同时按住第一个键不放。而两个键之间的逗号(,) 说明按第二个键之前必须释放第一个键。
注释	包含一些重要的信息。
➲ **程序**	指明下面将列出一些处理过程。许多任务都是以这种格式来描述的，为用户提供这些处理过程的合理顺序。
➲ **例**	示例显示在屏幕上，一般包括正文，用作详细阐述。
命令行	包括文本，这些命令都可以输入计算机中，显示在屏幕上。通常用于显示SQL命令的输入输出或dmconfig.ini中的内容。

表 1-1 文档协定

2 SQL基础

本手册适用于在DBMaster中使用SQL语言的任何人。从使用dmSQL命令行工具执行即席查询（ad-hoc queries）的用户，到使用ESQL/C和DBMaster ODBC兼容接口来开发用户应用程序的程序员，都可以使用本手册。

同时，本手册也提供了结构化查询语言（Structured Query Language）的参考以及每条SQL指令的语法。手册中的示例、图表可帮助您更清楚地理解其中的内容。

2.1 语法图

手册中的语法图说明了SQL命令的语法。在命令行中构造一条命令的时候，语法图能够给您提供帮助。语法图的使用很简单，跟随图中的线条从箭头开始到结束即可。在这过程中不能被绕过的元素是在命令中必须的，可以绕过的元素是可选的。另外，图中的and/or选项也为您在使用语法图时提供了灵活性。

斜体字部份代表数据库对象的真实名称，而在语法图中，我们用这些斜体字占位符来代替应该填写的真实名称。如下图，用数据库的真实表名替换斜体字“*table_name*”。在实例数据库中，当您需要在Customers表上执行操作时，应该使用“Customers”来替换“*table_name*”。

注意语法图中的箭头，有时候一条命令会包含多个项目，这表现在语法图中将是一个循环。正如图中的循环路径所示：字段名区域包括多个字段名，它们以逗号分隔。

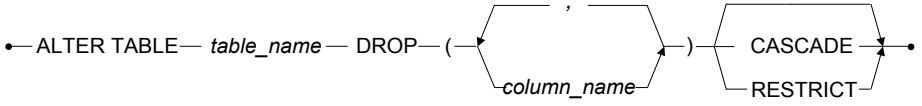


图 2-1 A语法图范例

2.2 数据类型

在表中定义字段时需要为该字段选择一种数据类型。您最好能够理解每个字段的用途，这样才能选择正确的数据类型。选择错误的数据类型不仅会浪费数据库的空间，而且还会给应用程序增加额外的任务，即将错误的数据类型转换成可用的类型。

DBMaster支持以下几种数据类型：

BIGINT、BINARY(size)、BIGSERIAL、CHAR(size)、NCHAR(size)、DATE、DECIMAL(NUMERIC)、DOUBLE、FILE、FLOAT、INTEGER、JSONCOLS、LONG VARBINARY(BLOB)、LONG VARCHAR(CLOB)、REAL、OID、SERIAL(start)、SMALLINT、TIME、TIMESTAMP、VARCHAR(size)、NVARCHAR(size)以及媒体类型。

BIGINT

BIGINT类型是一种精确的带正负号的数值类型，其精度和宽度分别为19位和0位。BIGINT类型数据的存储位为8个字节，其有效范围为9,223,372,036,854,775,807至−9,223,372,036,854,775,808。

如果您想将超过BIGINT或INTEGER最大有效值的数据转存为BIGINT或INTEGER类型，DBMaster会报出类型转换错误的信息，并对该操作不予执行。

➲ **例1**

```
37654
```

➲ **例2**

```
857823
```

BIGSERIAL(start)

DBMaster使用BIGSERIAL数据类型为数据库中的每一个表分配连续的整数并进行唯一识别。DBMaster能够内部自行管理这些整数，每个数字都是在每次使用DBMaster时自动产生的。

当为BIGSERIAL字段设定初始值时，就需要在定义字段时给可选参数START赋值，当START参数被忽略时初始值默认为1.。每张表中只能有一个BIGSERIAL类型的字段。

产生的BIGSERIAL序列数是一个整型数据，这个整数是精确的数字数据类型，其精度和宽度分别为19和0位，占8个字节的存储空间，其取值范围是-9,223,372,036,854,775,808~9,223,372,036,854,775,806。

当您新增一笔记录时，把BIGSERIAL字段设为空值（NULL），那么DBMaster就会为新插入记录的SERIAL字段自动增加一个序列数，该序列值以1递增。

当您在新增记录中为BIGSERIAL字段赋了一个整数值，那么DBMaster不会再产生数字，而直接使用您输入的数值，同时系统中的内部序列数也不会递增1。如果您输入的整数值比在数据库中的序列数记数器的值还大，那么下次DBMaster自动产生的数字就会重设，从您新输入的值开始生成新的值。

➲ **例1**

```
10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007
```

➲ **例2**

```
10000, 10001, 5000, 10002, 10003, 11000, 11001, 11002
```

BINARY (size)

BINARY类型是一种定长的数据类型，用于存储二进制数据。BINARY字段的最大长度为4KB、8KB、16KB、32KB，当创建一个BINARY类型字段时，用户可以输入size参数的值。如果录入的BINARY类型字段长度小于size，那么余下的部分将填充0。BINARY字段默认的最小长度为1字节，最大长度为8056字节。

像输入CHAR类型数据一样，在输入字符数据时，应在其前后加上单引号(' ')。不过在BINARY类型字段中存储的不是您实际输入的字符，而是等价于该字符ASCII码的十六进制数。

您也可以直接输入十六进制数，当然也要加上单引号，并且还要在其后附加字母x(' 'x)以示其为十六进制数据。十六进制中每个字节需要两位数来表示，因此输入的十六进制数应是偶位数。

➲ **例1**

```
'AaBbCcDdEe'x
```

➲ **例2**

```
'41614262436344644565'x
```

CHAR (size)

CHAR类型是一种定长的数据类型，用于存储您从键盘输入的字符。CHAR类型字段的最大长度为4KB、8KB、16KB、32KB，在定义CHAR类型字段时用户可输入参数size的值。CHAR类型字段默认的最小长度为1字节，最大长度为8056字节。

如果录入CHAR类型字段的数据长度小于size，那么余下的部分将以空格填充。输入CHAR类型数据时，必须在其前后加上单引号(' ')。每个双字节字符都占用2个字节的存储空间，所以在计算双字节字符所占用的存储空间时，需要考虑这一点。

➲ **例1**

```
''This is a CHAR string.'
```

➲ **例2**

```
'This is another CHAR string.'
```

DATE

DATE类型数据有两种：DATE函数和DATE字段。前者表示当前日期，后者表示一个特定的日期。DATE类型是一个定长的数据类型，用来存储日期（年、月、日）。DATE类型数据所占的存储空间为4个字节，可以表示的日期范围为0001年—9999年。

DATE类型的数据有多种输入/输出格式。如果数据库中的日期数据显示不正确或无法输入有效的日期时，请检查它们的输入/输出格式是否正确。

- **例1a**

```
'0001/01/01'
```

- **例1b**

```
'0001/01/01'd
```

- **例1c**

```
DATE '0001/01/01'
```

- **例2a**

```
'1999/12/31'
```

- **例2b**

```
'1999/12/31'd
```

- **例2c**

```
DATE '1999/12/31'
```

DECIMAL (NUMERIC)

DECIMAL类型是精确的带正负号的数值类型，它具有可变精度和宽度。其中，精度指小数点前后数字的总位数，默认的精度值是17位，最大为38位。而宽度则指小数点后的位数，默认的宽度是6位。

在使用DECIMAL类型字段存储数据时，所需要的实际空间并不是依默认精度和宽度或您在定义字段时所确定的精度和宽度而定，而是根据您所输入的值而定。

下面的公式可计算该类型数据所占用的存储空间：

$$\#\text{ of bytes} = \frac{p+1}{2} + 2$$

例如，9283.83本来应该存储6个字节。

其具体的计算为：

$$\#\text{ of bytes} = \frac{p+1}{2} + 2$$
$$= \frac{6+1}{2} + 2$$
$$= 5.5$$

如果您想将超过DECIMAL类型最大值的数据（如FLOAT或DOUBLE类型）转存到DECIMAL类型字段中，那么DBMaster将显示一条类型转换错误信息，并对此操作不予执行。DECIMAL类型可以缩写为DEC。

➲ **例1**

```
3452.8373645
```

➲ **例2**

```
736.383732652
```

DOUBLE

DOUBLE类型是不精确的带符号数值的数据类型，尾数精度可到15位，其精度指尾数中小数点前后数字的总位数。DOUBLE类型的数据存储为8个字节，其有效输入范围为：1.0E308～–1.0E308。

能输入的最小值为：**1.0E-308 和 –1.0E-308**。

➲ **例1**

```
2.89837457884451E285
```

➲ **例2**

```
-1.93873634847372E-174
```

FILE

FILE类型是一种占48个字节的结构化数据类型，该数据类型类似于CLOB及BLOB类型，它将现存文件以及其它数据存储为DBMaster能连接的外部文件。DBMaster并不将这些数据存储为一个内部对象，而是存储为一个外部文件。这种存储方式使第三方工具在存取和操作数据库后，不用再将数据重新导入数据库来记录数据的改变。一个文件对象的路径长度不能超过255个字符。

FILE类型字段存储了系统表中的记录参考信息，数据库使用系统表来查找文件对象信息。当查询FILE类型字段时，您并不能看到该字段的真正内容，DBMaster给您显示的是存储在系统表中三个视图之一的相关信息，或文件名、文件大小和文件内容。

FILE类型有两种存储型态：系统文件对象和用户文件对象。系统文件对象可将现有文件拷贝到数据库文件对象的目录下，并指定一个唯一的名字，由数据库来管理这些文件，将没有记录连接的文件自动删除。当用户断开现有文件的连接时，用户文件对象将为该文件创建一个链接。因为这是用户创建的文件，所以当数据库没有记录连接到文件时，它将不会被删除。在您将文件作为用户文件对象连接到数据库之前，DBMaster必须拥有该文件的读取权限。

当多条记录连接到同一文件时，DBMaster并不会产生额外的文件，所有记录都将共享同一个文件对象以节省磁盘空间。不过针对每个用户而言，好像每条记录都连接到一个专用的文件对象。当更新一个共享文件时，DBMaster会产生一个新文件，同时，其它记录共享的文件也不会发生任何变动，其他用户仍然连接的是原始文件。这样在一条记录对一个文件进行修改时，并不会影响其它记录对文件的共享。

FLOAT

FLOAT类型是一种不精确的带正负号的数值数据类型，尾数精度可到15位，其精度指小数点前后数字总位数。默认的FLOAT类型的数据存储为8个字节，其有效范围为：1.0E308～ –1.0E308。可以通过关键字**DB_FltDb**将默认的FLOAT类型设置为REAL类型或DOUBLE类型。

能输入的最小值为：**1.0E-308** 和 **–1.0E-308。**

➲ **例1**

```
2.89837457884451E285
```

➲ **例2**

```
-1.93873634847372E-174
```

INTEGER

INTEGER类型是一种精确的带正负号的数值数据类型，其精度和宽度分别为10位和0位。INTEGER类型数据的存储位为4个字节，其有效范围为2,147,483,647～-2,147,483,648。

如果您想将超过INTEGER最大有效值的数据（如DOUBLE等类型）转存为INTEGER类型，DBMaster会报出类型转换错误的信息，并对该操作不予执行。INTEGER数据类型可以简写成INT。

➲ **例1**

```
393848
```

➲ **例2**

```
-298376
```

JSONCOLS

JSONCOLS类型是一种动态字段集。DBMaster支持动态字段，但动态字段并不出现在表定义中。动态字段是源于JSON字符串的键值，且仅能在表中已有字段被声明为JSONCOLS类型的前提下才能使用。有关动态字段的详细信息，请参考*数据库管理员手册*中的*使用动态字段*章节。有关JSONCOLS字段的详细信息，请参考*数据库管理员手册*中的*使用JSONCOLS类型*章节。动态字段在表中被存储为JSCONLS类型，该数据类型是LONG VARBINARY的派生类型。

➲ **例1**

创建一个包含JSONCOLS类型字段的表：

```
dmSQL> CREATE TABLE student(name CHAR(30), info JSONCOLS);
```

或：

```
dmSQL> CREATE TABLE student(name CHAR(30));
dmSQL> ALTER TABLE student ADD COLUMN info JSONCOLS;
```

使用JSONCOLS类型字段的字段名向表**student**中插入数据：

```
dmSQL> INSERT INTO student(name,info) VALUES
('jessia','{"desk_id":3,"birthday":"1986-09-19","score":90}');
1 rows inserted
```

```
dmSQL> INSERT INTO student(name,info) VALUES
('pine','{"desk id":4,"birthday":"1987-03-03","score":95}');
1 rows inserted
```

使用“SELECT *”语句查询表**student**：

```
dmSQL> SET blobwidth 80;
dmSQL> SELECT * FROM student;
       NAME                              INFO
================= ================================================
jessia            {"score":90,"birthday":"1986-09-19","desk_id":3}
pine              {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

使用JSONCOLS类型字段的字段名查询表**student**：

```
dmSQL> SELECT name, info FROM student;
       NAME                              INFO
================= ================================================
jessia            {"score":90,"birthday":"1986-09-19","desk_id":3}
pine              {"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
```

使用JSONCOLS类型字段的字段名更新表**student**中的数据：

```
dmSQL> UPDATE student SET info = '{"desk_id":7, "birthday":"1986-09-
19","score":88}' WHERE name='jessia';
1 rows updated
```

将字段**birthday**的数据类型更改为DATE类型：

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN birthday DATE;
dmSQL> SELECT info FROM student;
                                INFO
================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
2 rows selected
dmSQL> INSERT INTO student(name,desk id,birthday,score) VALUES ('mike','8','1985-
02-15','92');
dmSQL> SELECT info FROM student;
                                INFO
================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
```

```
{"BIRTHDAY":477244800000,"DESK ID":"8","SCORE":"92"}
3 rows selected
```

在JSONCOLS类型字段**info**上创建文本索引：

```
dmSQL> CREATE TEXT INDEX idx_stu ON student(INFO);
```

在JSONCOLS类型字段**info**上创建视图：

```
dmSQL> CREATE VIEW view1 AS SELECT info FROM student;
dmSQL> SELECT * FROM view1;
                                INFO
====================================================================
{"score":88,"birthday":"1986-09-19","desk_id":7}
{"score":95,"birthday":"1987-03-03","desk_id":4}
{"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
3 rows selected
```

➲ **例2**

以下操作基于表**student**。有关表**student**详详细信息，请参考例**1**。

使用动态字段的字段名向表**student**中插入数据：

```
/* implicit data conversion is closed by default */
dmSQL> INSERT INTO student(name,score) VALUES(?,?);
dmSQL/Val> 'demi','85';    /* it is ok */
1 rows inserted
dmSQL/Val> 'finly',82;     /* INT cannot be converted to CHAR */
ERROR (9629): 数值列表语法错误
dmSQL/Val> END;
dmSQL> SET itcmd ON;
dmSQL> INSERT INTO student (name,score) VALUES(?,?);
dmSQL/Val> 'finly',82;     /* using implicit data conversion */
1 rows inserted
dmSQL/Val> END;
dmSQL> SET itcmd OFF;
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('linda','1','1982-
01-01','91');
1 rows inserted
dmSQL> INSERT INTO student(name,desk_id,birthday,score) VALUES('glow','2','1984-
03-25','93');
1 rows inserted
```

```
dmSQL> INSERT INTO student (name,desk id,birthday,score)
VALUES('kitty','abc','1980-02-27','97');
1 rows inserted
```

使用“SELECT *”语句查询表**student**：

```
dmSQL> SELECT * FROM student;
     NAME                          INFO
============= ============================================


jessia          {"score":88,"birthday":"1986-09-19","desk_id":7}
pine            {"score":95,"birthday":"1987-03-03","desk_id":4}
mike            {"BIRTHDAY":477244800000,"DESK_ID":"8","SCORE":"92"}
demi            {"SCORE":"85"}
finly           {"SCORE":"82"}
linda           {"BIRTHDAY":378662400000,"DESK_ID":"1","SCORE":"91"}
glow            {"BIRTHDAY":448992000000,"DESK_ID":"2","SCORE":"93"}
kitty           {"BIRTHDAY":320428800000,"DESK_ID":"abc","SCORE":"97"}
8 rows selected
```

使用动态字段的字段名查询表**student**：

```
dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID      BIRTHDAY              SCORE
=============  ===========  ============  =====================
jessia          7            19*           88
pine            4            19*           95
mike            8            19*           92
demi            NULL         NU*           85
finly           NULL         NU*           82
linda           1            19*           91
glow            2            19*           93
kitty          abc           19*           97
8 rows selected
```

使用动态字段的字段名更新或删除表**student**中的数据：

```
dmSQL> UPDATE student SET score='88' WHERE name='linda';
1 rows updated
dmSQL> DELETE FROM student WHERE desk_id='2';
1 rows deleted
```

为表中的动态字段添加描述信息：

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN desk_id INT;
```

```
dmSQL> ALTER TABLE student ADD DYNAMIC COLUMN score DOUBLE;
```

向表**student**中插入数据：

```
dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jane','12','1982-
05-07',96);
ERROR (6150): [DBMaster] expression/predicate中，insert/update数据类型和字段类型不符
或compare/operand数据类型和字段类型不符
dmSQL> INSERT INTO student(name, desk_id, age, score) VALUES('jim',8,'1984-09-
26',98);
1 rows inserted
dmSQL> SELECT name, desk_id, birthday, score FROM student;
     NAME         DESK_ID       BIRTHDAY            SCORE
==============  ============  ============  ======================
jessia                    7  1986-09-19    8.80000000000000e+001
pine                      4  1987-03-03    9.50000000000000e+001对比maker、
mike                      8  1985-02-15    9.20000000000000e+001
demi                   NULL        NULL    8.50000000000000e+001
finly                  NULL        NULL    8.20000000000000e+001
linda                     1  1982-01-01    8.80000000000000e+001
kitty                  NULL  1980-02-27    9.70000000000000e+001
jim                       8        NULL    9.80000000000000e+001
8 rows selected
```

更改动态字段**score**的数据类型：

```
dmSQL> ALTER TABLE student MODIFY DYNAMIC COLUMN score TYPE TO INT;
```

在动态字段**desk_id**上创建索引：

```
dmSQL> CREATE INDEX idx1 ON student(desk_id);
```

删除动态字段**birthday**的描述信息：

```
dmSQL> ALTER TABLE student DROP DYNAMIC COLUMN birthday;
```

LONG VARBINARY (BLOB)

BLOB类型是一种变长的数据类型，用来存储任何二进制数据。每一个BLOB字段的最大长度不能超过8 TB。与BINARY数据类型不同的是，BLOB类型只在数据库中存储您输入的字节，而并不会以0来填充多余的空间。

如同输入CHAR类型的数据一样，您得在数据的前后加上单引号(' ')。不过在BLOB类型的字段中存储的不是您实际输入的字符，而是等价于该字符ASCII码的十六进制数据。

您也可以直接输入十六进制数，当然也要加上单引号，并且还要在后面附加字母x(' 'x)以示其为十六进制数据。十六进制中的每个字节用两位数来表示，因此在输入十六进制数时应是偶数位数。

➲ **例1**

```
'AaBbCcDdEe'x
```

➲ **例2**

```
'41614262436344644565'x
```

LONG VARCHAR (CLOB)

CLOB类型是一种变长数据类型，用来存储任何从键盘输入的字符。CLOB类型字段的最大长度不能超过8 T。

与CHAR类型数据不同的是，CLOB类型只在数据库中存储您输入的字符，而不会以空格来填充多余的空间。当您在CLOB字段输入数据时，应在数据前后加上单引号(' ')。每个双字节字符占用2个字节的存储空间，所以在您计算这些字段所占空间时，就应该考虑这一点。

➲ **例1**

```
'This is a varchar string.'
```

➲ **例2**

```
'This is another varchar string.'
```

NCHAR (size)

NCHAR类型是一种定长的数据类型，用来存储Unicode字符。在UTF16 Little-Endian(LE)编码制中，每个Unicode字符占两个字节。Size参数决定了字段中字符的个数，所以必须在定义NCHAR字段时输入size参数，该参数值默认可在1～4028中取值。

如果NCHAR类型字段的数据长度小于size，那么余下的部分将以空格填充。输入NCHAR类型数据时，应该在这些Unicode字符前后加上单引号(' ')，并且还应加上前缀‘N’。

➲ **例1**

下面给出了Unicode数据的输入范例：

```
N'Unicode Data'
```

如果NCHAR数据是以十六进制的格式输入，那么应该在这些十六进制串前后加上单引号，并在后面附加字母‘u’。

➲ **例2**

下例表示以十六进制格式输入的Unicode数据，其中包含三个Unicode字符：

```
'610a620b63f1'u
```

当您在向Unicode字段录入数据而没有加前缀‘N’时，DBMaster会自动将该数据转换为Unicode类型。如果将Unicode字符输入到CHAR类型字段中，那么DBMaster会根据**dmconfig.ini**中的关键字**DB_LCode**值对输入的Unicode字符做相应的转换，不能转换成本地码的字符将以□来表示。

NCHAR类型的同义字包括NATIONAL CHAR(size)和NATIONAL CHARACTER(size)。

NVARCHAR (size)

NVARCHAR类型是一种可变长的数据类型，用来存储Unicode字符。在UTF16 Little-Endian (LE)编码制中，每个Unicode字符占2个字节的存储空间。Size参数决定字段中的字符个数，所以必须在定义NVARCHAR字段时输入size参数，它的默认取值范围为1～4028。

如果NVARCHAR类型字段的数据小于size，那么余下的部分将不会以空格填充。输入NVARCHAR类型数据时，应该在这些Unicode字符的前后加上单引号(' ')，并且加上前缀‘N’。

➲ **例1**

以下给出了Unicode字符的输入范例：

```
N'Unicode Data'
```

如果NVARCHAR数据以十六进制的格式输入，那么应该在这些十六进制串的前后加上单引号，并在后面附加字母‘u’。

➲ **例2**

下例表示以十六进制格式输入的Unicode数据，其中包含三个Unicode字符：

```
'610a620b63f1'u
```

如果您在往Unicode字段录入数据时没有加前缀‘N’，那么DBMaster会自动将数据转换为Unicode类型。如果将Unicode字符输入到VARCHAR类型的字段，那么DBMaster会根据**dmconfig.ini**中的关键字**DB_LCode**值，对输入的Unicode字符做相应的转换。不能转换成本地码的字符将以□来表示。

NVARCHAR类型的同义字包括：NATIONAL CHAR VARYING(size)、NCHAR VARYING(size)、NATIONAL VARCHAR(size)和NATIONAL CHARACTER VARYING(size)。

OID

OID(object identifier)类型是一种特殊的数据类型，为存储在数据库中的对象、记录或BLOB分配一个唯一的ID。该结构化的数据类型的精度和宽度分别为10位和0位，并占据16个字节的存储空间。当插入记录时，DBMaster会自动为每条记录产生并插入一个对象编号。对象编号是由DBMaster在内部管理及维护的，用户不能直接使用它。

OID值的产生和数据库中对象的存储位置有关，这意味着两个连续生成的对象编号并不一定连续。

OID字段在表中是个隐藏字段，用SELECT * FROM CUSTOMERS这样的查询语句是不会显示其值的。如果要查询OID字段，则应在查询语句中使用‘OID’作为字段名。

尽管您可以使用对象编号来查询或修改数据，但在使用SQL语句时，这种方式并不是通用的。对象编号一般用于内部编程接口，并不直接在交互式dmSQL环境中使用。

REAL

REAL类型是一种不精确的带符号的数值类型，它的尾数精度可到7位。精度是指尾数中小数点前后数字的总位数。REAL类型占用4字节的存储空间，它的有效范围为3.402823466E38 –3.402823466E38。能输入的最小值为1.175494351E-38 至 –1.175494351E-38。如果试图将一个大于允许范围的数值，如DOUBLE类型的一个数值，转换为REAL，DBMaster将显示类型转换错误，并且转换失败。

➲ **示例1**

```
3.583837E34
```

➲ **示例2**

```
-1.873653E-21
```

SERIAL (start)

SERIAL类型是一种特殊的数据类型，用以产生一串序列数字。DBMaster为数据库中的每张表分配一个整数，并通过这些整数为相应的表生成一个唯一的序号。DBMaster能够内部自行管理和维护这些序号，每个数字都是在每次使用DBMaster时自动产生的。

如果您想为SERIAL字段设定初始值，那么就需要在定义字段时给可选参数START赋值，否则系统将字段的初始值默认为1。每张表中只能有一个SERIAL类型的字段。

因为系统产生的序列数实际上是一个整型数据，所以SERIAL数据类型拥有整型（INTEGER）数据的所有特性。因此SERIAL类型和INTEGER类型一样，都是精确的带符号的数据类型，其精度和宽度分别为10和0位，占4个字节的存储空间，其取值范围都是–2,147,483,648～2,147,483,646。

当您新增一笔记录时，把SERIAL字段设为空值（NULL），那么DBMaster就会为新插入记录的SERIAL字段自动增加一个序列数，该序列值以1递增。

如果您在新增记录中为SERIAL字段赋了一个整数值，那么DBMaster不会再产生数字，而直接使用您输入的数值，同时系统中的内部序列数也不会递增1。如果您输入的整数值比在数据库中的序列数记数器的值还大，那么下次DBMaster自动产生的数字就会重设，从您新输入的值开始生成新的值。

➲ **例1**

```
100, 101, 102, 103, 104, 105, 106, 107
```

➲ **例2**

```
100, 101, 50, 102, 103, 110, 111, 112
```

SMALLINT

SMALLINT类型是一种精确的带符号的数值数据类型，其精度和宽度分别为5位和0位。SMALLINT占2个字节的存储空间，取值范围为-32,768～32,767。

如果您想将超过SMALLINT最大有效值的数据（如INTEGER或DOUBLE等类型）转存为SMALLINT类型，那么DBMaster会报出类型转换错误信息，并对该操作不予执行。

➲ **例1**

```
4769
```

➲ **例2**

```
8376
```

TIME

TIME类型数据有两种：TIME函数和TIME字段。Time函数表示当前时间，其值随时间不断改变；而TIME字段则用于存放某个固定时刻。这两种时间类型都是定长数据类型，占4个字节的存储空间。除非您在输入时

间值时，明确地标识了‘AM’或‘PM’，否则系统默认的时间值输入格式为24小时制。

这两种时间类型都有多种输入/输出格式。如果数据库中的时间数据不能正确显示，或者不能输入您认为有效的时间值，请检查它们的输入/输出格式是否正确。

➲ **例1a**

```
'22:04:05'
```

➲ **例1b**

```
'22:04:05't
```

➲ **例1c**

```
TIME '22:04:05'
```

➲ **例2a**

```
'10:04:05 PM'
```

➲ **例2b**

```
10:04:05 PM't
```

➲ **例2c**

```
TIME 10:04:05 PM'
```

TIMESTAMP

TIMESTAMP类型数据有两种：TIMESTAMP函数和TIMESTAMP字段。TIMESTAMP函数用于取得当前时间，其值随时间不断变化。而TIMESTAMP字段则存放某个特定时刻。

这两种TIMESTAMP类型都是定长的数据类型，可用来存储日期和时间数据，占11个字节的存储空间，精度和宽度分别是17和10位。有效的日期值是从0001年到9999年。系统默认的时间值输入格式是24小时制，除非您在输入时间值的时候明确标识了‘AM’或‘PM’。

两种TIMESTAMP数据类型都按照TIME类型和DATE类型的输入格式来输入日期和时间。如果数据库中的时间数据不能正确显示，或者不能输入您认为是有效的值时，请检查它们的输入/输出格式是否正确。

➲ **例 1a**

```
'1997/01/01 10:02:03'
```

➲ **例 1b**

```
'1997/01/01 22:02:03'ts
```

➲ **例 1c**

```
TIMESTAMP '1997/01/01 10:02:03'
```

➲ **例 2a**

```
'01.01.1997 22:02:03'
```

➲ **例 2b**

```
'01.01.1997 22:02:03'ts
```

➲ **例 2c**

```
TIMESTAMP '01.01.1997 22:02:03'
```

VARCHAR (size)

VARCHAR类型是一种变长的数据类型，用来存储输入的字符。VARCHAR类型字段的最大长度为4KB、8KB、16KB或32KB，在创建VARCHAR字段时，应为参数size赋值。VARCHAR类型字段默认的最小长度为1字节，最大长度为8056字节。

数据库中只存储您输入的字符。在输入VARCHAR字段值时应在字符串前后加上单引号(' ')。每个双字节字符占用2个字节的存储空间，所以在您计算字段所占用的存储空间时，应考虑这一点。

➲ **例1**

```
' This is a VARCHAR string.'
```

➲ **例2**

```
' This is another VARCHAR string.'
```

媒体类型

大型对象字段可以定义为媒体类型，这有助于媒体处理，例如在Microsoft Word文档中的全文搜索等功能。媒体类型包括：MsWordType、HtmlType、XmlType、MsPPTType、MsExcelType、

PDFType、MsWordFileType、HtmlFileType、XmlFileType、MsPPTFileType、MsExcelFileType以及PDFFileType。

媒体类型实际上是现有数据类型的派生类型。MsWordType、MsPPTType、MsExcelType、PDFType、HtmlType以及XmlType继承于LONG VARBINARY类型，HtmlType和XmlType继承于LONG VARCHAR，而MsWordFileType、HtmlFileType以及XmlFileType继承于FILE类型。了解这一点对您使用ALTER TABLE来修改字段数据类型是很重要的。每种媒体类型数据的特性都继承其相应数据类型的特性。

XMLTYPE 的特性有：

- XML完整格式检查：插入/更新的xml内容必须是格式完全正确的。
- XML有效性：在创建xmltype字段时，选择指定一个有效的UDF，DBMaster使其xml内容有效。
- XML数据以原来的格式被保存。
- 使用XPath搜索：选择指定一个 xpath并使用extract 函数来查询/定位XML数据中的节点。
- 更新由XPath指定的XML内容。
- 在XPath extract上创建索引：通过对频繁查询的xpath表达式建立索引可加快查询速度。
- 不允许将xmltype字段或其他数据类型修改为xmltype。

➲ **例**

```
dmSQL> CREATE TABLE minutes (id INT, meeting_date DATE, doc MSWORDFILETYPE);
dmSQL> INSERT INTO minutes VALUES (1, 3/3/2003, 'c:\meeting\20030303.doc');
```

2.3 数据转换

您可以按照如下方式转换数据类型：

- 当一个对象的数据移到另一个对象，或两个对象之间的数据进行比较或组合时，数据可能需要从一个对象的数据类型转换为另一个对象的数据类型。
- 将SQL结果字段、返回代码或输出参数中的数据移到某个程序变量中时，必须将这些数据从DBMaster系统数据类型转换成该变量的数据类型。
- 当表达式包含不同类型的数据时，需进行数据转换以确保数据的兼容性。

DBMaster支持隐式和显式转换数据类型。

建议用户使用显式转换，而不是隐式转换或自动转换，原因如下：

- 使用显式数据类型转换函数时，SQL语句易于理解。
- 隐式数据类型转换可能会对性能产生一定影响，尤其是字段值的数据类型被转换成常量数据类型时。
- 隐式转换取决于转换内容，且每次转换方式可能会有所不同。

显式数据转换

用户可使用如下SQL转换函数来对数据进行显式转换：CAST、DATETOSTR、TIMETOSTR、TIMESTAMPTOSTR、TO_DATE。

CAST允许用户将一个类型的输出数据转换为另一个数据类型，详细信息请参考*SELECT*章节。

DATETOSTR函数用于将DATE类型数据转换为指定格式的字符串，详细信息请参考*DATETOSTR*章节。

TIMETOSTR函数用于将TIME类型数据转换为指定格式的字符串，详细信息请参考*TIMETOSTR*章节。

TIMESTAMPTOSTR函数用于将TIMESTAMP类型数据转换为指定格式的字符串，详细信息请参考*TIMESTAMPTOSTR*章节。

TO_DATE函数用于将选定的字符串转换为DATE类型，详细信息请参考*TO_DATE*章节。

隐式数据转换

当转换成功时，DBMaster会自动将数值从一种数据类型转换为其它类型，主要包括数值型数据和字符型数据之间的转换。数值型数据类型可以是INTEGER (INT、SERIAL)、SMALLINT、BIGINT、BIGSERIAL、FLOAT、DOUBLE、DECIMAL；字符型数据类型可以是CHAR、VARCHAR、NCHAR、NVARCHAR。使用隐式数据转换之前，用户需通过命令“SET itcmd ON”或将**DB_ItcMd**的值设为1来启用该功能。

下表列出所有有效转换，转换方向为X行到Y列。

Xy	int (serial)	small-int	bigint (bigserial)	decimal	double	float	(var) char	n(var) char
int(serial)	Y	Y	Y	Y	Y	Y	Y	Y
smallint	Y	Y	Y	Y	Y	Y	Y	Y
bigint(bigserial)	Y	Y	Y	Y	Y	Y	Y	Y
decimal	Y	Y	Y	Y	Y	Y	Y	Y
double	Y	Y	Y	Y	Y	Y	Y	Y
float	Y	Y	Y	Y	Y	Y	Y	Y
(var)char	Y	Y	Y	Y	Y	Y	Y	Y
n(var)char	Y	Y	Y	Y	Y	Y	Y	Y

表 2-1 隐式数据转换表

DBMaster根据以下规则进行隐式数据类型转换：

- 在INSERT操作中，DBMaster会把插入值转换为插入字段的数据类型。
- 在算术运算中（算术运算符：+、-、*、/），DBMaster会把字符型数据转换为数值型数据。
  - **a)** 若运算符一侧是字符型数据，另一侧是数值型数据，则DBMaster会把该字符型数据转换为另一侧数值型数据的数据类型。

b) 若运算符两侧都是字符型数据，同时都是常量字符数据，DBMaster会把两侧的字符型数据都转换为合适的数据类型。比如，在表达式'123'+'123.456'+'1.23e45'中，'123'、'123.456'、'1.23e45'将会分别被转化为123（INT型）、123.456 (DECIMAL型)、1.23e45 (DOUBLE型)，否则DBMaster将会把字符型数据转化为DOUBLE型。

- 在比较运算中（比较运算符：>、>=、=、<=、<、!=、<>、IN、IS NULL），DBMaster会把运算符右侧的数据转换为运算符左侧数据的数据类型。
- 在连接运算中（连接运算符：||、CONCAT），DBMaster会把数值型数据转换为字符型数据。

请注意DBMaster也能隐式转换用户自定义函数（UDF）的参数及默认值。

➲ **例 1**

对于如下语句，DBMaster会把CHAR（VARCHAR）类型的数据隐式转换为INT类型的数据。

```
dmSQL> SET itcmd ON;
dmSQL> CREATE TABLE t1 (c1 INT);
dmSQL> INSERT INTO t1 VALUES ('123');
dmSQL> SELECT * FROM t1 WHERE c1 = '123';
dmSQL> UPDATE t1 SET c1='456'+111;
dmSQL> DELETE FROM t1 WHERE c1 = '678'-111;
```

➲ **例 2**

对于如下语句，DBMaster会把DECIMAL类型的数据隐式转换为NCHAR（NVARCHAR）类型的数据。

```
dmSQL> CREATE TABLE t2 (c1 NCHAR(20), c2 NVARCHAR(20));
dmSQL> INSERT INTO t2 VALUES (12345.6789, 222.222);
dmSQL> SELECT * FROM t2 WHERE c1 = 12345.6789 AND c2 = 222.222;
dmSQL> UPDATE t2 SET c1 = -6789.12345;
```

➲ **例 3**

对于如下语句，DBMaster会隐式转换用户自定义函数（UDF）的参数。

```
dmSQL> CREATE TABLE t1 (c1 INT, c2 CHAR(10), c3 NCHAR(10));
```

```
dmSQL> INSERT INTO t1 VALUES(abs('-10'), -abs(-10), abs(10));
dmSQL> SELECT * FROM t1;
    C1           C2                        C3
==========  ==========  ====================================
        10  -10         3100300000000000000000000000000000000000
1 rows selected

dmSQL> SELECT * FROM t1 WHERE abs(c1) = abs(c2);
    C1           C2                        C3
==========  ==========  ====================================
        10  -10         3100300000000000000000000000000000000000
1 rows selected

dmSQL> SELECT * FROM t1 WHERE abs(c1) = abs(c3);
    C1           C2                        C3
==========  ==========  ====================================
        10  -10         3100300000000000000000000000000000000000
1 rows selected

dmSQL> SELECT abs(c1), abs(c2), abs(c3) FROM t1;
    ABS(C1)                    ABS(C2)               ABS(C3)
====================== ===================== ==================
1.00000000000000e+001   1.00000000000000e+001  1.00000000000000e+001
1 rows selected

dmSQL> SELECT concat(c1,c2), concat(c1,-123), concat(1.234e8, c1) FROM t1;
    CONCAT(C1,C2)        CONCAT(C1,-123)     CONCAT(1.234E8, C1)
==================== ================== ====================
10-10                10-123              12340000010
1 rows selected
```

➲ **例 4**

对于如下语句，DBMaster会隐式转换默认值。

```
dmSQL> CREATE TABLE t1 (c1 INT DEFAULT '123456');
dmSQL> INSERT INTO t1 VALUES(DEFAULT);
1 rows inserted
dmSQL> SELECT * FROM t1;
     C1
==========
```

```
     123456
1 rows selected

dmSQL> CREATE TABLE t1 (c1 CHAR(20) DEFAULT 123456);
dmSQL> INSERT INTO t1 VALUES(DEFAULT);
1 rows inserted
dmSQL> SELECT * FROM t1;
     C1
============
      123456
1 rows selected
```

2.4 RESERVED WORDS

下面列表中的关键字不应被当作普通标志符。如果您将它们作为普通标志符来使用，那么DBMaster会报出“ERR_RESERVED_WORD”的错误信息，并且不会执行您的指令。

ABSOLUTE | ACTION | ADD | ADMIN | AFTER | AGGREGATE | ALIAS | ALLOCATE | ALTER | AND | ANY | ARE | ARRAY | AS | ASC | ASSERTION | ASENSITIVE | AT | AUTHORIZATION | BEFORE | BEGIN | BIGINT | BIGSERIAL | BINARY | BIT | BLOB | BOOLEAN | BOTH | BREADTH | BREAK | BY | CALL | CASCADE | CASCADED | CASE | CAST | CATALOG |CHAR | CHECK | CLASS | CLOB | CLOSE | COLLATE | COLLATION | COLUMN | COMMIT | COMPLETION| CONDITION | CONNECT | CONT | CONNECTION | CONSTRAINT | CONSTRAINTS | CONSTRUCTOR | CONTINUE | CORRESPONDING | CREATE | CROSS | CUBE | CURRENT | CURRENT_DATE | CURRENT_PATH | CURRENT_ROLE | CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_USER | CURSOR | CYCLE| DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT | DEFERRABLE | DEFERRED | DELETE | DEPTH | DEREF | DESC | DESCRIBE | DESCRIPTOR | DESTROY| DESTRUCTOR | DETERMINISTIC | DICTIONARY | DIAGNOSTICS | DISCONNECT | DISTINCT |DO |DOMAIN | DOUBLE | DROP | DYNAMIC | EACH | ELSE | ELSEIF | END | END-EXEC | EQUALS | ESCAPE | EVERY | EXCEPT| EXCEPTION | EXEC | EXECUTE | EXIT | EXTERNAL | FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FROM| FREE | FULL | FUNCTION | GENERAL | GET | GLOBAL | GO | GOTO | GRANT | GROUP | GROUPING | HANDLER | HAVING | HOLD | HOST | IDENTITY | IF | IGNORE | IMMEDIATE | IN | INDICATOR | INITIALIZE | INITIALLY | INNER | INOUT | INPUT | INSENSITIVE | INT | INTEGER | INTERSECT | INTO | IS | ISOLATION | ITERATE | JOIN | KEY | LANGUAGE | LANGUAGE SQL | LARGE | LAST | LATERAL | LEADING | LEAVE | LESS | LEVEL | LIKE | LIMIT | LOCAL | LOCALTIME | LOCALTIMESTAMP | LOCATOR | LOOP | MAP | MATCH | MODIFIES | MODIFY | MODULE |

NAMES | NATIONAL | NATURAL | NCHAR | NCLOB | NEXT | NO | NONE | NOT | NULL | NUMERIC | NVARCHAR | OBJECT | OF | OFF | ON | ONLY | OPEN | OPERATION | OPTION | OR | ORDINALITY | OUT | OUTER | OUTPUT | PAD | PARTIAL | PATH | POSTFIX | PREFIX | PREORDER | PREPARE | PRESERVE | PRIMARY | PRIOR | PRIVILEGES | PROCEDURE | READ | READS | REAL | RECURSIVE | REFERENCES | REFERENCING | RELATIVE | REPEAT | RESTRICT | RESULT | RETURN | RETURNS | REVOKE | ROLE | ROLLBACK | ROLLUP | ROUTINE | ROW | ROWS|SAVEPOINT | SCHEMA | SCROLL | SCOPE | SEARCH | SECTION | SELECT| SENSITIVE | SEQUENCE | SERIAL | SESSION | SESSION_USER | SET | SETS | SHORT | SIZE | SMALLINT | SOME | SPECIFIC | SPECIFICTYPE | SQL | SQLCODE | SQLEXCEPTION | SQLSTATE | SQLWARNING | START | STATIC | STATISTICS | STOP | STRUCTURE | SYSTEM_USER | TABLE | TEMPORARY | TERMINATE | THAN | THEN | TIME | TIMESTAMP | TIMEZONE_HOUR| TIMEZONE_MINUTE | TO | TRACE | TRAILING | TRANSACTION | TRANSLATION | TREAT | TRIGGER | TRUE | UNDER | UNION | UNKNOWN | UNTIL | UNNEST | UPDATE | USAGE | USING | VALUE | VALUES | VARBINARY | VARBPTR | VARCHAR | VARCPTR | VARIABLE | VARYING | VIEW | WHEN | WHENEVER | WHERE | WHILE | WITH | WITHOUT | WORK | WRITE | ZONE

3 SQL命令

DBMaster提供了全面的SQL查询语言。SQL(Structured Query Language)是一种ANSI规定的标准的查询语言，目前的标准是ANSI-99。本章中所有DBMaster的SQL命令，都支持ANSI-99标准。

3.1 ABORT BACKUP

ABORT BACKUP命令可用于退出在线备份。如果在备份操作中有错误出现或打算以后再执行备份，那么您可以取消当前的备份操作。只有具备SYSADM、SYSDBA或DBA权限的用户才能执行ABORT BACKUP命令。

备份模式决定了DBMaster是否可以执行在线增量备份以及备份哪些内容。DBMaster提供了三种备份模式：不备份（NON-BACKUP）、备份数据（ BACKUP-DATA）以及备份数据和BLOB（BACKUP-DATA-AND-BLOB）。用户可通过三种方式来设置备份模式：您可以利用配置文件**dmconfig.ini**中的**DB_BMode**关键字来设置备份模式，也可以用SQL命令来设定备份模式，还可以利用服务器管理员工具（Server Manager）来设定备份模式。

NON-BACKUP 模式对上次完整备份后新插入或修改的数据不提供备份。用户可以利用日志文件从出现错误的程序中完全恢复一个数据库，但在介质损坏的情况下，一些丢失的数据就可能无法恢复了。在执行检查点（checkpoint）后，可以立即回收没有被当前事务使用的日志记录。一旦这些回收的日志文件被重写，数据库就只能恢复到上次完整备份的状态了。

BACKUP-DATA 模式可为上次完整备份后新插入或修改的数据（不包括BLOB数据）提供恢复保障。在此种模式下，DBMaster可以执行在线增量备份，但只有非BLOB数据才会被存储到备份文件中。用户可以利用日志从错误的程序中完全恢复数据库，也可以从损坏的介质中部分恢复数据库。执行检查点（checkpoint）后，没有在当前事务中使用的记录只有在做过备份后才能被回收。

BACKUP-DATA-AND-BLOB模式对上次完整备份后新插入或修改的数据（包括BLOB数据）提供恢复保障。在此种模式下，DBMaster可以执行在线增量备份，所有数据都将被存储到备份文件中。用户可以用日志从错误的程序和损坏的介质中完全恢复数据库。您可以利用上次的备份将数据库（包括BLOB数据）完全恢复至介质发生错误的状态。执行检查点

（checkpoint）后，没有在当前事务中使用的记录只有在做过备份后才能被回收。

执行ABORT BACKUP命令后并不会改变数据库的备份模式，数据库仍然保持其原来的备份模式。

图 3-1 ABORT BACKUP语法图

➲ **例**

下面是退出备份的操作：

```
BEGIN BACKUP
ABORT BACKUP
```

3.2 ABORT CONNECTION

ABORT CONNECTION命令用于中断一个活动连接，而不是断开与数据库的连接。只有具备DBA、SYSDBA或SYSADM权限的用户才可以执行ABORT CONNECTION命令。

ABORT CONNECTION命令与KILL CONNECTION命令功能相似，唯一的区别是ABORT CONNECTION命令不会断开连接，且该连接将回滚到最后提交的状态。该命令主要用于中断耗时较长的查询，而不是断开连接。执行该命令不会释放该用户占有的所有锁资源。

请注意被中断的连接将会回滚。若该连接处于活动状态，它将会被立即中断并回滚；若该连接处于非活动状态，它将会在执行下一条SQL语句时被中断并回滚。

connection_id ..... 将要中断连接的id值。

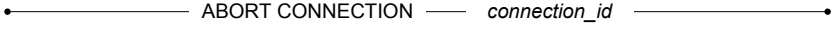


图3-2 ABORT CONNECTION语法图

➲ **例**

下例中断了ID值为**12345**的连接。

```
ABORT CONNECTION 12345
```

3.3 ADD TO GROUP

ADD TO GROUP命令可将用户添加到一个已存在的组中，这样用户就拥有该组对象当前及以后的操作权限。只有具备SYSADM、SYSDBA或DBA权限的用户才可以执行ADD TO GROUP命令。

用户组简化了数据库中众多用户对象权限的管理。将用户或用户组添加到组中，任何赋予该组的权限也自动赋予组中的所有成员。

加入到组中的成员仍然保持其原有权限，移出该组的成员将不再拥有该组的任何权限，但仍将保留自己被直接授予的对象权限或在其他组中的权限。

只要一个组A的成员中不包含组B，那么在往组B中添加成员时，ADD TO GROUP命令中的用户名（user_name）可以被替换为组A的名称（group_name）。用户名和组名的长度不能超过128个字符，可以包括任何字母、数字、下划线以及$和#符号，但不能以数字开头。

user_name.....至少拥有连接权（connect）的现有用户名。

group_name.....现有的组名。

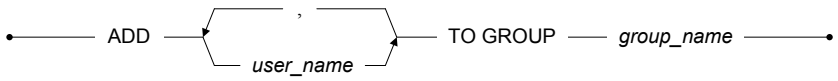


图*3-3* ADD TO Group语法图

➲ **例1**

此例说明如何将用户**Joe**和**John**加入到组**Manager**中。

```
dmSQL> ADD Joe, John TO GROUP Manager;
```

➲ **例2**

下例说明如何将组**FullTime**和**PartTime**添加到组**Staff**中。

```
dmSQL> ADD FullTime, PartTime TO GROUP Staff;
```

➲ **例3**

下例说明如何将用户**Bill**和组**FlexTime**添加到组**Staff**中。

```
dmSQL> ADD Bill, FlexTime TO GROUP Staff;
```

3.4 ADD TRACE

ADD TRACE命令可在单个表中添加追踪功能用来记录详细的OLD\NEW数据。实际上，它是通过三个内部触发器（插入/更新/删除）来实现的。这些操作会记录在追踪表里，OLD\NEW数据也将被作为额外信息复制到**DBNAME_currentdate_###.TXT**里。只有表所有者、具备DBA、SYSDBA或SYSADM权限的用户才可以执行ADD TRACE命令。

注意 *DB_LgSvr必须等于或大于4。否则，详细信息将会被忽略，日志文件也不会写入任何信息。*

table_name .....现有的单个表名

图 3-4 ADD TRACE语法图

➲ **例**

在表**tb1**里添加跟踪功能，并且插入、更新、删除记录。

```
dmSQL> ADD TRACE ON tb1;
dmSQL> INSERT INTO tb1 VALUES (1, 'abc');
1 rows inserted
dmSQL> UPDATE tb1 SET c2 = 'xyz' WHERE c1=1;
1 rows updated
dmSQL> DELETE FROM tb1;
1 rows deleted
```

3.5 ALTER DATAFILE

ALTER DATAFILE命令可通过增加页数来扩大数据文件或BLOB文件的大小。只有拥有SYSADM、SYSDBA或DBA权限的用户才能执行该命令。

数据库中的文件是存储数据的物理单元。操作系统用于管理文件，而DBMS用于管理文件中的数据。DBMaster使用的文件类型有：数据、BLOB及日志。

数据文件和BLOB文件用于存储用户和系统数据，尽管这两种文件的特性相似，但DBMaster对它们的管理方式却不尽相同，这样有助于提高系统的性能。数据文件用于存储表和索引数据，而BLOB文件则用于存储二进制大型对象。

日志文件是一种特殊的文件，它记录了用户对数据库更改的实时和历史记录以及数据库更改后的状态。使用日志文件可以撤消失败事务对数据库的更改，或在数据库灾难恢复时将已提交的事务重新写入数据库。日志文件只能由数据库管理系统（DBMS）使用，它并不能存储用户数据。

为了保证数据库中的数据独立性，操作系统中的数据不应直接被存取。因此，每个数据库文件都有两个名字：物理文件名（*physical file name*）和逻辑文件名（*logical file name*）。前者是操作系统使用的文件名，后者是在数据库中使用的文件名。这两个文件名可通过**dmconfig.ini**文件进行交互。

使用ALTER DATAFILE命令时，需要指明逻辑文件名。如果文件的总页数小于2147483645并且有足够的磁盘空间，那么您可以为每个文件指定的页数为1～2147483645 。一个表空间的总文件大小不得超过8TB。

file_name ..... 要扩大页数的逻辑文件名

number..... 要增加的页数

图 3-5 ALTER DATAFILE语法图

➲ **例1**

下面是摘自于**dmconfig.ini**文件的记录。加上逻辑和物理文件名，一共4个数据库文件。其中逻辑文件名在左边，物理文件名在右边。

```
customer_data = d:\dbmaster\tutorial\database\custdata.db 500
customer_blob = d:\dbmaster\tutorial\database\custblob.bb 1000
```

➲ **例2**

下例说明了如何为文件**customer_data**增加**1000**页。

```
dmSQL> ALTER DATAFILE customer_data ADD 1000 PAGES;
```

➲ **例3**

来自于同一个**dmconfig.ini**文件中的**customer_data**文件页数增加了500页。

```
customer_data = d:\dbmaster\tutorial\database\custdata.db 1500
customer_blob = d:\dbmaster\tutorial\database\custblob.bb 1000
```

3.6 ALTER INDEX RENAME

ALTER INDEX RENAME命令可更改现有表中已存在的索引名称。该动作只会影响系统目录中的索引名称，而不会重建数据库中的索引。只有表的拥有者、DBA或者具有INDEX 权限的用户才可以执行ALTER INDEX RENAME命令。

index_name……索引的原名

new_index_name……更改后索引的新名

table_name..... 索引所在的表名

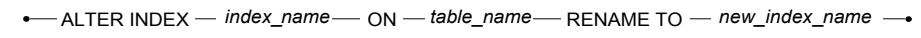


图 3-6 ALTER INDEX RENAME语法图

➲ 例

```
dmSQL> ALTER INDEX ix1 ON tb_tem RENAME TO ix_new;
```

3.7 ALTER PASSWORD

ALTER PASSWORD命令用来修改用户密码。普通用户可以修改他们当前的密码，而系统管理员（SYSADM）可以修改任何用户的密码。

当普通用户想修改自己的密码时，应该使用如下命令：ALTER PASSWORD *old_password* TO *new_password*。当SYSADM想要修改密码时，应该用如下的命令：ALTER PASSWORD OF *user_name* TO *new_password*。注意：只有SYSADM才可以使用第二个命令。

当修改密码时，用户的当前密码必须和他在数据库中存储的密码一致。如果用户原来没有密码，那么在命令中可输入关键字NULL作为其当前密码。同样，要删除密码，只需将新密码值赋为NULL即可。

密码的最大长度不能超过16个字符，可以包含任何字母、数字、下划线以及$和#符号，但不能以数字开头。

user_name.....要更改密码的用户名

old_password.....用户的当前密码

new_password...用户要设置的新密码

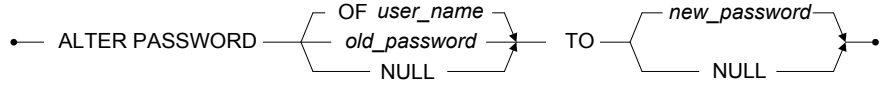


图 3-7 ALTER PASSWORD语法图

➲ **例1**

下例说明了如何为一个没有密码的用户设置一个新密码**abcdef**。

```
dmSQL> ALTER PASSWORD NULL TO abcdef;
```

➲ **例2**

下例说明了普通用户如何将密码abcdef改为**a23456**。

```
dmSQL> ALTER PASSWORD abcdef TO a23456;
```

➲ **例3**

下例说明了如何删除自己的密码**a23456**。

```
dmSQL> ALTER PASSWORD a23456 TO NULL;
```

➲ **例4**

下例说明了**SYSADM**如何在不输入用户**John**当前密码的情况下，将其密码更改为**abcedf**。

```
dmSQL> ALTER PASSWORD OF John TO abcdef;
```

3.8 ALTER REPLICATION ADD REPLICATE

ALTER REPLICATION ADD REPLICATE命令可用来在表复制机制中增加一个远程表，您可以根据自己的需要增加远程表。表的拥有者或拥有DBA、SYSDBA或SYSADM权限的用户可以执行该指令。

表复制可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，用户可以不去干预。

表复制有两种类型：同步（*synchronous*）表复制和异步（*asynchronous*）表复制。同步表复制指当我们对本地表做任何修改时，远程表就会立刻被更新；异步表复制按照既定的时间计划将本地表中的更改复制到远程表中。ALTER REPLICATION ADD REPLICATE命令对*同步和异步*表复制都起作用。

DBMaster中的*同步*表复制遵循全局事务（global transaction）规则，即把数据复制到远端的表当作本地事务的一个完整的部分，这意味着如果对远端表的更新操作失败了，那么对本地表的更新事务也不会成功。

事务是运作的逻辑单元，它将同时完成一项或多项对数据库的操作，在执行这些操作后，数据库仍处于一致状态。事务必须是独立的，执行事务后只可能有两种状态：事务正常完成，更改数据库中数据；事务失败，数据库中数据不发生任何变化。

在DBMaster中，*异步*表复制利用事务日志将数据复制到远程表中。DBMaster将本地表的更改都存储在事务日志中，然后根据既定时间计划复制到远程表中。通过使用事务日志，DBMaster可独立地处理远程和本地事务，即使在不能连接远程表的情况下，仍可以对本地表进行更改。异步表复制可以在网络或远程数据库出错的情况下，不影响本地操作。因为在错误排除之前，DBMaster会不断尝试向远程复制。

在修改表复制机制时，需要指定复制名（replication name）、要复制的本地表名以及远程目的表名。无论是本地表还是远程表，它们都必须是已存于各自数据库中的表。

如果您在使用命令时没有指明要复制的本地表字段，那么DBMaster会复制整张表。只有在创建表复制时才能指定要复制的本地表字段。在没有指明复制的字段名而复制整张表时，本地表和远程表中的字段名和数据类型必须相同。

如果本地表和远程表的字段名不同，那么您在创建表复制机制时，需要给出远程表的字段列表。本地表中的字段将按从左到右的顺序复制到远程表的相应字段中。您也可以同时指明远程表和本地表中的字段，使其一一对应。但无论在何种情况下，两张表的主键（Primary Key）的字段数和主键的数据类型都必须一致。

DBMaster将对象名、拥有者名及表名结合起来作为复制的完整名称。因此，同一张表上的每个复制名都应该是唯一的。

DBMaster是按表复制创建者的安全性及对象权限来进行同步表复制的。但如果远程数据库明确规定了以数据库链的方式来存取，那么DBMaster将按照数据库链的安全性和对象权限来进行同步表复制。

DBMaster按照远程用户的安全性及对象权限进行异步表复制，这些远程用户是在使用CREATE SCHEDULE命令，由子句IDENTIFIED BY指定的。在创建表复制之前，应使用CREATE SCHEDULE命令先创建一个复制计划。

CLEAR DATA、FLUSH DATA、CLEAR AND FLUSH DATA这三个关键字是可选的，它们是在定义表复制时规定具体操作的关键字。如果选择了关键字CLEAR DATA，则在创建表复制时，系统会先将远程表中的数据删除；如果选择了关键字FLUSH DATA，则在创建表的同时，系统会将本地表中满足条件的数据先复制到远程表中；如果选择了CLEAR AND FLUSH DATA关键字，则在创建表复制的同时，先清除远程表中的数据，再将满足条件的数据插入到远程表中；如果以上三项您都不选的话，系统将维持目的表的原状。

关键字NO CASCADE也是可选的，其作用是进行级联复制。如同大多数的组织，命令是由最高层逐层往下传到最基层。数据从节点A复制到节点

B，然后再复制到节点C，这是一种典型的级联复制。而在No-Cascade模式中，数据从节点A复制到节点B，同时，B也将其数据复制到A。如果您想通过这种方式复制数据，就应该开启NO CASCADE选项。如果没做任何规定，系统则默认CASCADE为开启状态。

replication_name.....要添加远程表的表复制名。

local_table_name.....创建表复制的本地表名。

remote_table_name..远程表名。

column_name .....远程表中要复制的字段名。

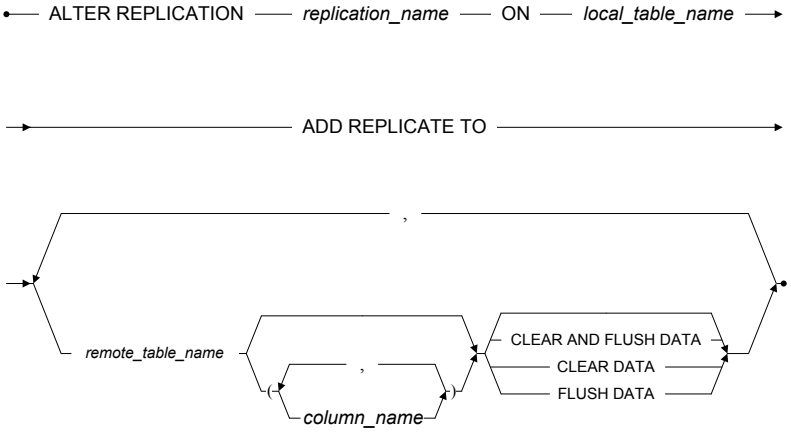


图 3-8 ALTER REPLICATION ADD REPLICATE语法图

➲ **例1**

下例说明了如何更改创建在本地表**Employeesinfo**中的表复制**EmpRep**。数据被复制到远程数据库的**Div1Emp**表中，该远程数据库**Div1Office**的配置信息在本地**dmconfig.ini**文件中列出。并且这两张表中的字段名和数据类型是相同的。

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp;
```

➲ **例2**

若使用关键字**CLEAR DATA**，DBMaster会在复制之前先清除远程表中的所有数据。其语法如下：

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp CLEAR DATA;
```

➲ **例3**

使用关键字**FLUSH DATA**，DBMaster会在复制之前将本地表中的数据发送到远程表中。其语法如下：

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp FLUSH DATA;
```

➲ **例4**

若使用关键字**CLEAR AND FLUSH DATA**，DBMaster会先将远程表中的所有数据先删除，然后再将本地表中的数据发送到远程表中。其语法如下：

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div1Office:Div1Emp CLEAR AND FLUSH DATA;
```

➲ **例5**

下例是在已有表复制中增加目的表：远程数据库**Div2Office**中的**Div2Emp**表和库**Div3Office**中的**Div3Emp**表，同时在本地的**dmconfig.ini**文件中包含了这两个远程数据库的配置信息。

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo ADD REPLICATE TO
                            Div2Office:Div2Emp CLEAR DATA,
                            Div3Office:Div3Emp FLUSH DATA;
```

3.9 ALTER REPLICATION DROP REPLICATE

ALTER REPLICATION DROP REPLICATE命令可将远程目的表从现有表复制中删除。当您不需要将数据复制到表中时，就可以使用该命令从表复制中将其删除。当然，只有表的拥有者或拥有DBA、SYSDBA或SYSADM权限的用户才有权执行该命令。

*表复制*可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，不需要用户干预。

表复制有两种类型：*同步（synchronous）*表复制和*异步（asynchronous）*表复制。同步表复制指当我们对本地表做任何修改时，远程表就会立刻被更新；异步表复制按照既定的时间计划将本地表中的更改复制到远程表中。ALTER REPLICATION ADD REPLICATE命令对同步和异步表复制都起作用。

DBMaster中的*同步*表复制遵循全局事务（global transaction）规则，即把数据复制到远端的表当作本地事务的一个完整的部分，这意味着如果对远端表的更新操作失败了，那么对本地表的更新事务也不会成功。事务是运作的逻辑单元，它将同时完成一项或多项对数据库的操作，在执行这些操作后，数据库仍处于一致状态。事务必须是独立的，执行事务后只可能有两种状态：事务正常完成，更改数据库中数据；事务失败，数据库中数据不发生任何变化。

在DBMaster中，*异步*表复制利用事务日志将数据复制到远程表中。DBMaster将对本地表的修改存储在事务日志中，然后根据既定的时间计划复制到远程表中。通过使用事务日志，DBMaster可独立地处理远程和本地事务，即使在不能连接远程表的情况下，仍可以执行对本地表的更改。异步表复制允许在网络或远程数据库出错的情况下继续执行，因为在错误排除前，DBMaster会不断尝试向远程复制。

要在表复制机制中删除一个远程表，则需要指定表复制名、本地表名以及远程表名。若要删除多个表，只需将要删除的表名列出即可。如果本地表被删除，那么为该表创建的复制也将自动被删除。

replication_name...要进行删除表操作的表复制名。

local_table_name...创建表复制的本地表名。

remote_table_name ...要从表复制机制中删除的远程表名。

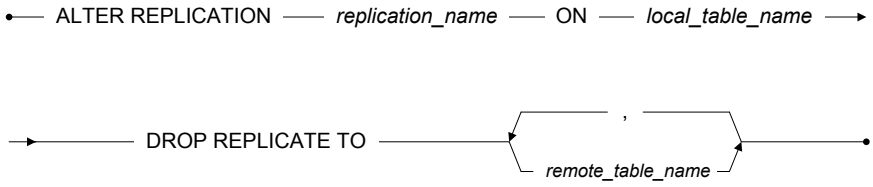


图 3-9 ALTER/DROP REPLICATION语法图

➲ **例1**

下例表示从创建在本地表**Employeesinfo**的表复制机制**EmpRep**中删除远程表**Div1Emp**。

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo DROP REPLICATE TO Div1Emp;
```

➲ **例2**

下例表示从创建在本地表**Employeesinfo**的表复制机制**EmpRep**中删除远程表**Div2Emp**、 **Div3Emp**和**Div4Emp**。

```
dmSQL> ALTER REPLICATION EmpRep ON Employeesinfo DROP REPLICATE TO Div2Emp,
Div3Emp, Div4Emp;
```

3.10 ALTER SCHEDULE

ALTER SCHEDULE命令可以为异步表复制修改复制的时间计划。同步表复制无需使用复制计划，所以ALTER SCHEDULE命令对同步表复制没有影响。只有具备DBA、SYSDBA或SYSADM权限的用户才能执行该命令。

表复制可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，用户可以不去干预。

表复制主要有两种类型：同步（synchronous）表复制和异步（asynchronous）表复制。同步表复制指当我们对本地表做任何修改时，远程表就会立刻被更新；而异步表复制则按照既定的时间计划将本地表中的更改复制到远程表中。所有ALTER SCHEDULE命令都只对异步表复制起作用。

关键字BEGIN AT指明第一次做异步表复制的日期和时间。日期的格式必须是：*yyyy/mm/dd*，其中*yyyy*表示年份，范围为1970—2038年；*mm*表示月份，其范围为01—12；*dd*表示日，其范围为01—31。时间的格式必须是：*hh:mm:ss*，其中*hh*表示小时，其范围为00—23；*mm*表示分钟，其范围为00—59；*ss*表示秒，其范围为00——59。可以看出，要使用BEGIN AT关键字就必须包含日期和时间两组值。如果您在复制运行的情况下将第一次复制的时间改为将来的某个时间，那么还没有复制到远程数据库中的数据将会保留在您的数据库中，直到您所设定时间到来时才开始复制。

关键字EVERY规定了异步表复制中相继发生的两次复制的时间间隔。这个时间间隔可以是时/分/秒，也可以是天数或前两者的结合。您可使用EVERY *hh:mm:ss*格式来设定时/分/秒的值，其中*hh*是00到23之间的整数，*mm*是00到59之间的数，*ss*是00到59之间的数值。使用EVERY *d*

DAYS格式来设定间隔的天数，其中*d*是1到365之间的整数。使用EVERY *d* DAYS AND *hh:mm:ss*格式将以上二者结合起来设定时间值。

在执行SQL命令时可能会出现一些错误，比如锁超时或根据全日志事务回滚到保存点，关键字RETRY则是指当这些错误发生时，DBMaster重试表数据复制的次数。您可以使用RETRY *n* TIMES格式来设定重试的次数，其中*n*代表重试的次数，它的范围为0到2147483647，默认值是***0***。

如果因为网络错误或远程数据库出错而不能连接到远程服务器，那么DBMaster会等到下一个复制计划到来时，重新复制上一次没有复制成功的数据。如果碰到回滚事务的请求，DBMaster会只重试复制一次，如果数据仍然复制失败，那么只有等到一个复制计划再执行该任务。

关键字AFTER是可选项，它与关键字RETRY结合起来，用于指明出现错误时两次重试之间的时间间隔。使用Use AFTER *s* SECONDS格式来指明这一间隔的秒数，SECONDS的取值范围为0到2147483647，其默认值为***5***。

当远程数据库中的数据发生更改而使表复制不能进行时，可用关键字ON ERROR来指明DBMaster应采取的动作。这些使复制不能正常进行的情况有：DBMaster试图在远程表中删除一条并不存在的记录或向远程表中插入已存在的记录。当这些错误发生时，DBMaster为用户提供两个解决方案：STOP ON ERROR和IGNORE ON ERROR。STOP ON ERROR表示当这种错误发生时，DBMaster将终止复制；IGNORE ON ERROR表示DBMaster可以忽略错误，继续进行数据的复制，系统默认的设置是忽略错误（IGNORE ON ERROR）。

使用关键字IDENTIFIED BY可以指定连接远程数据库时使用的用户名和密码。这里指定的用户必须是已存在于远程数据库中，并具备表的INSERT, DELETE, UPDATE等权限的用户。用户可执行的操作由赋予该用户的安全级别和对象权限决定。

在修改时间计划时还应指明远程数据库名，并且该名字不能是一个数据库链接（database link）。复制的时间计划被修改后，所有异步表复制都将按照新的复制计划进行。

yyyy/mm/dd..... 开始复制的日期。

hh:mm:ss ..... 1. 开始复制的时间。

.....2. 复制的时间间隔。

d .....向远程表复制的天数间隔。

n .....发生错误时重试的次数。

s .....错误出现时，两次重试之间的秒数。

user_name .....远程数据库中的用户名。

password.....远程数据库用户的密码。

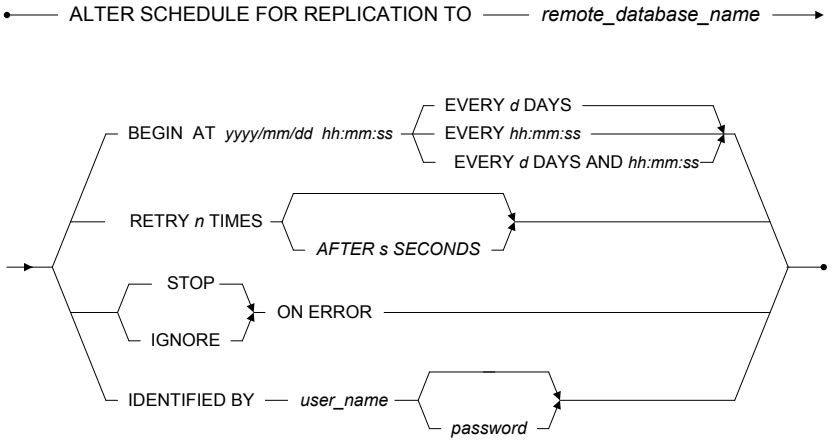


图 3-10 ALTER SCHEDULE语法图

➲ **例1**

下例表示如何修改一个名为**EmpRep**的异步表复制计划。在出现**锁超时**（**lock time-out**）或根据完整日志将事务**回滚**（**rollback**）到**存储点**（**savepoint**）等错误时，重试的次数设为**3**，重试的时间间隔为**5秒**。

```
dmSQL> ALTER SCHEDULE FOR REPLICATION TO EmpRep
                RETRY 3 TIMES AFTER 5 SECONDS;
```

➲ **例2**

下例表示如何修改一个名为**EmpRep**的异步表复制计划。当远程数据库中的数据发生更改，使得表复制不能进行时，DBMaster所采取的行动是终止复制（**STOP ON ERROR**）。

```
dmSQL> ALTER SCHEDULE FOR REPLICATION TO EmpRep
               STOP ON ERROR;
```

➲ **例3**

下例表示如何修改一个名为**EmpRep**的异步表复制计划。例子中设置了连接远程数据库的**用户名**及**密码**。

```
dmSQL> ALTER SCHEDULE FOR REPLICATION TO EmpRep
               IDENTIFIED BY RepUser rdejpe88;
```

3.11 ALTER TABLE ADD COLUMN

ALTER TABLE ADD COLUMN命令用于修改表的定义并在表中添加字段。只有表的拥有者、DBA或对指定表有ALTER权限的用户才可以执行该命令。

定义字段时需要指明字段名、字段的数据类型或者所属定义域。您也可以在一条命令中添加多个字段，只要保证表中的字段总数不超过2000个。

table_name .....要增加字段的表名。

column_definition…字段定义。

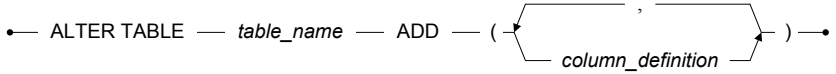


图3-11 ALTER TABLE ADD COLUMN语法图

字段定义

每个字段都应指定其数据类型。DBMaster支持以下数据类型：BINARY、BIGINT、BIGSERIAL、CHAR、DATE、DECIMAL、DOUBLE、FILE、FLOAT、INTEGER、BLOB、CLOB、OID、SERIAL、SMALLINT、TIME、 TIMESTAMP和VARCHAR。

您可以不使用标准的数据类型而使用用户自定义的数据类型（定义域）来定义字段。定义域可用来设定字段的数据类型和默认值以及字段的完整性约束。您可以使用关键字DEFAULT和CHECK来定义字段的默认值和完整性约束（下面会提到这两个关键字的具体用法）。如果您在定义字段时重新设定了字段的默认值，那么字段的默认值就会是新设定的值；如果您在定义字段时定义了完整性约束，那么该字段的条件约束将是定义域中的约束加新定义的约束。

关键字NULL/NOT NULL是可选的，这两个关键字决定了在插入一个字段时其值是否可以为空（NULL）。NULL关键字指在插入记录时，该字段上可以不输入数值。NOT NULL关键字指在插入新记录时，该字段必须输入数值。NOT NULL只能在表中没有数据时才能使用，否则它就可能与本来是空值的字段发生冲突，导致字段无法创建。

关键字USER/SYSTEM是可选的，这两个关键字决定了用户是否可以使用INSERT/UPDATE语句来更改字段的默认值。若用户没有设置该关键字，则默认为USER。当使用USER关键字时，用户可以更改字段的默认值；而使用SYSTEM关键字时，用户不能更改字段的默认值。

关键字DEFAULT也是可选的。当插入一条新记录时，如果没有输入该字段值，就可以将DEFAULT关键字所指定的值作为该字段的默认值。常数、内置函数的值以及空值（NULL）都可以作为字段的默认值。其中只有不含自变量的内置函数，如PI( )、NOW( )或USER( )才能作为定义的默认值。如果用NULL作为默认值，那么在定义字段时就不能再使用关键字NOT NULL。当用户在定义字段时使用的是定义域而不是标准的DBMaster数据类型时，最好不要再使用DEFAULT关键字，因为定义域中一般都包括默认值（DEFAULT）。

关键字ON UPDATE也是可选的。该关键字用于设置字段的默认值是否随其它字段值的更新而自动更新。

关键字CHECK也是可选的，用来指明输入字段值的取值范围或字段的条件约束。CHECK后面的表达式一般都是取值为真（true）或假（false）的表达式。可以将关键字VALUE和CHECK结合起来代表字段值。如果SQL命令不满足CHECK所设定的条件，它就不能被执行。如果用户使用定义域而不是标准的DBMaster数据类型来定义字段，那么就可以不使用关键字CHECK，因为定义域中一般都包括了它们自己的CHECK条件约束。

关键字GIVE是可选的，用户可用它来为表中已经存在的每条记录设定新字段的值。如果您没有用GIVE来设定插入值，则DBMaster会将新字段的默认值设为NULL。因为SERIAL类型的字段值不能为NULL，所以在增加一个SERIAL类型的字段时，必须用GIVE指定字段值。常数、内置函数的值以及空值（NULL）都可以用作GIVE值。如果使用NULL作为GIVE值，字段在定义时就不能再使用关键字NOT NULL了。同样，在插

入一个SERIAL字段时，您可以将关键字SEQUENTIAL/SEQ和GIVE一起使用。这些关键字表明DBMaster将从定义SERIAL字段时所指定的开始值起，向现有记录中插入序列值。这些序列值随着新记录的插入而不断增加。

关键字BEFORE/AFTER是可选的，这两个关键字用于指明新插入字段相对于已有字段的位置。如果使用BEFORE关键字，那么DBMaster会在指定字段的前面（紧挨的左边）插入新字段。如果使用AFTER关键字，那么DBMaster会在指定字段的后面（紧挨的右边）插入新字段。如果您没有使用这两个关键字指定新字段在表中的相对位置，那么DBMaster会将新字段插入到表中最右边字段的后面。

向表中加入新字段对在该表上创建的视图、同义字不会产生任何影响。字段名的最大长度不能超过128个字符，它可以包括字母、数字、下划线以及$和#符号，但不能以数字开头。

column_name .....新字段名。

data_type .....新字段的数据类型。

domain_name .....新字段使用的域名。

literal.....没有输入字段值时使用的默认数值。

constant .....没有输入字段值时使用的常量数值。

function_name .....没有输入字段值时的内置函数名。

constraint_name .....用于字段的约束名。

boolean_expression....取值为真/假的表达式。

column_name_a .....表中已有的字段名，新字段将放在它的后面。

column_name_b.....表中已有的字段名，新字段将放在它的前面。

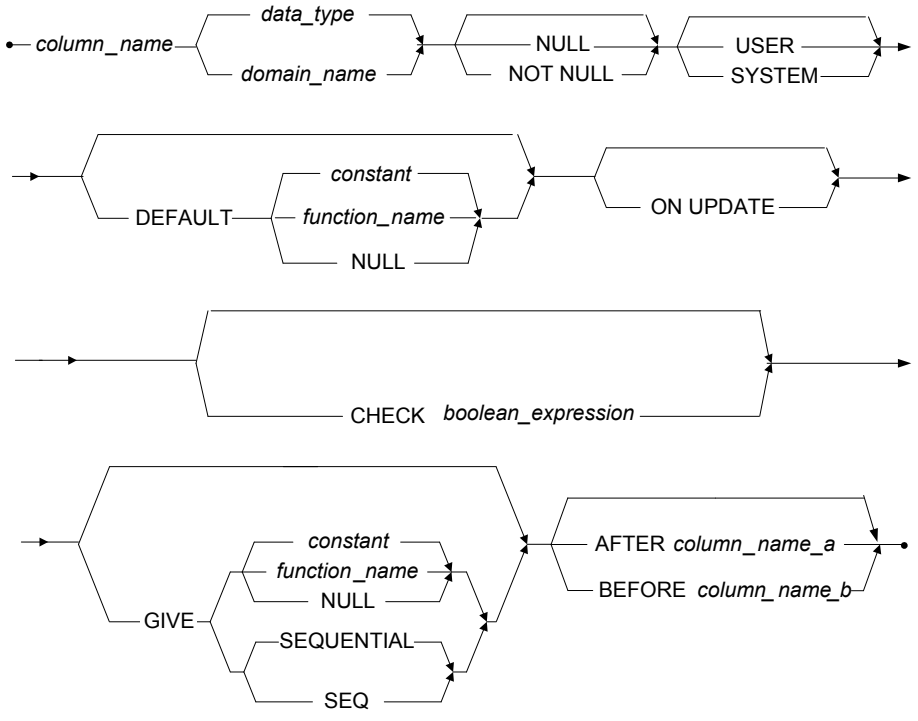


图 3-12 COLUMN DEFINITION语法图

➲ **例1**

下例说明是如何在**Employeesinfo**表中添加一个名为**HireDate**的日期（**DATE**）类型字段。

```
dmSQL> ALTER TABLE Employeesinfo ADD (HireDate DATE);
```

➲ **例2**

下例同上，只是增加了关键字**NOT NULL**，表示在新插入记录时必须输入该字段值。

```
dmSQL> ALTER TABLE Employeesinfo ADD (HireDate DATE NOT NULL);
```

➲ **例3**

下例同上，只是增加了**DEFAULT**关键字，用来设定当字段输入值为空时的默认值。只有在没有使用**NOT NULL**关键字的字段中输入数据时，

DEFAULT关键字才会起作用。本例中，没有输入数据的字段将使用内置函数**NOW( )**的值。

```
dmSQL> ALTER TABLE Employee ADD (HireDate DATE NOT NULL DEFAULT NOW());
```

➲ **例4**

下例同上，只是增加了**ON UPDATE**关键字，用来设定**HireDate**字段的值随其它字段值的更新而自动更新。

```
dmSQL> ALTER TABLE Employeesinfo ADD (HireDate DATE NOT NULL DEFAULT NOW() ON
UPDATE);
```

➲ **例5**

下例同上，只是增加了关键字**CHECK**，用来设定**HireDate**字段输入值的取值范围，关键字**VALUE**代表字段的输入值。

```
dmSQL> ALTER TABLE Employee ADD (HireDate DATE NOT NULL DEFAULT NOW() CHECK VALUE
> '01/01/1995');
```

➲ **例6**

下例同上，只是通过用户自定义的定义域**D_ValidDates**而不是时间（**DATA**）类型来定义字段。如果使用了定义域，一般就不再使用**DEFAULT**和**CHECK**这两个关键字了，因为定义域内通常都包含了**DEFAULT**和**CHECK**子句。

```
dmSQL> ALTER TABLE Employee ADD (HireDate D_ValidDates NOT NULL);
```

3.12 ALTER TABLE ADD DYNAMIC COLUMN

ALTER TABLE DROP COLUMN命令用来添加动态字段的描述信息。只有表的所有者、DBA、SYSDBA、SYSADM或对表有ALTER权限的用户才可以执行该命令。

定义一个JSONCOLS字段后，动态字段可以不被定义而直接使用。动态字段的默认数据类型是VARCHAR(256)，用户可以使用ALTER TABLE ADD DYNAMIC COLUMN命令将该默认数据类型更改为其它数据类型。此外，用户也可以在插入动态字段时使用该命令以声明动态字段的数据类型。

然而，若用户没有在插入数据之前执行ALTER TABLE ADD DYNAMIC COLUMN命令，而是在插入数据之后使用该命令声明动态字段的数据类型，那么，如果插入的数据无法转换为声明类型数据，该数据在查询结果中将显示为NULL，且数据库不报错。

有关动态字段的详细信息，请参考*数据库管理员手册*中的*使用动态字段*章节。有关JSCONSOLS字段的详细信息，请参考*数据库管理员手册*中的*使用JSCONSOLS字段*章节。

table_name ..... JSONCOLS字段所在表的表名。

column_name .... 要添加描述信息的动态字段的字段名。

data_type ..... 要添加描述信息的动态字段的数据类型。

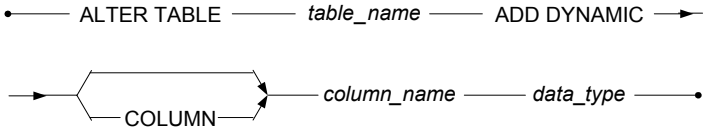


图 3-13 ALTER TABLE ADD DYNAMIC COLUMN语法图

➲ 例

下例说明如何给动态字段添加描述信息。

```
dmSQL> CREATE TABLE books(name CHAR(50),info JSONCOLS);
dmSQL> INSERT INTO books(name,id,price) VALUES('C language','abc','19');
1 rows inserted
dmSQL> INSERT INTO books(name,id,price) VALUES('College english','2','32');
1 rows inserted
dmSQL> ALTER TABLE books ADD DYNAMIC COLUMN id INT;
dmSQL> ALTER TABLE books ADD DYNAMIC COLUMN price FLOAT;
dmSQL> SELECT name,id,price FROM books;
               NAME                      ID           PRICE
================================== ========== ====================
C language                            NULL    1.900000000000e+001
College english                             2 3.200000000000e+001
2 rows selected
```

3.13 ALTER TABLE DROP COLUMN

ALTER TABLE DROP COLUMN命令用来更改一个现有表的定义并删除以前定义的字段。只有表的所有者、DBA、SYSDBA、SYSADM或对表有ALTER权限的用户才可以执行该命令。

当用户不再需要表中的某些字段时，可以使用该命令将这些字段删除。当表的主键或外键定义在这些字段上，或该字段带有定义的视图时，您可以根据需要选择删除的方式。DBMaster为你提供两种可选择的方式。

关键字CASCADE/RESTRICT是可选的。它们决定了删除字段时，是否删除或检查参照它的对象。选择关键字CASCADE，指删除字段时将删除所有参照该字段的对象；选择关键字RESTRICT则不会删除参照任一个视图定义、外键或约束的字段。RESTRICT关键字保证了只有没被其它对象参照的字段才可以被删除。

table_name..... 要删除字段的表名。

column_name.... 要删除的字段名。

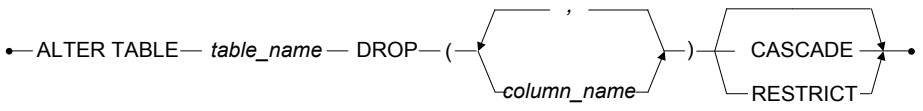


图 3-14 ALTER TABLE DROP COLUMN语法图

➲ **例1**

此命令是从表**Employeesinfo**中删除**BirthDate**字段。

```
dmSQL> ALTER TABLE Employeesinfo DROP (BirthDate);
```

➲ **例2**

下面的命令是从表**Employeesinfo**中删除**BirthDate**和**HireDate**字段。

```
dmSQL> ALTER TABLE Employeesinfo DROP (BirthDate, HireDate);
```

➲ **例3**

下面的命令是从表**Employeesinfo**中删除**BirthDate**字段，并一同删除建在该字段上的视图**EmpView**。

```
dmSQL> CREATE VIEW EmpView AS SELECT BirthDate FROM Employeesinfo;
dmSQL> ALTER TABLE Employeesinfo DROP (BirthDate) CASCADE;
```

3.14 ALTER TABLE DROP DYNAMIC COLUMN

ALTER TABLE DROP COLUMN命令用来删除动态字段的描述信息，但并不删除该动态字段。只有表的所有者、DBA、SYSDBA、SYSADM或对表有ALTER权限的用户才可以执行该命令。

请注意，若用户将JSONCOLS字段或该JSONCOLS字段的所在表删除，那么系统将自动删除该JSONCOLS所包括动态字段的描述信息。

有关动态字段的详细信息，请参考*数据库管理员手册*中的*使用动态字段*章节。有关JSCONSOLS字段的详细信息，请参考*数据库管理员手册*中的*使用JSCONSOLS字段*章节。

table_name..... 要删除描述信息的动态字段所在表的表名。

column_name.... 要删除描述信息的动态字段的字段名。

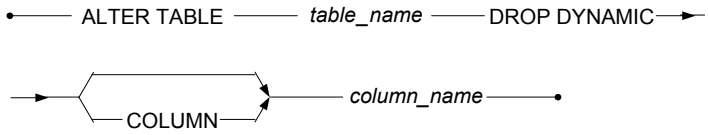


图 3-15 ALTER TABLE DROP DYNAMIC COLUMN 语法图

➲ 例

下例说明如何删除动态字段的描述信息。有关表**books**的详细信息，请参考*ALTER TABLE ADD DYNAMIC COLUMN*章节。

```
dmSQL> ALTER TABLE books DROP DYNAMIC COLUMN id;
dmSQL> SELECT name,id,price FROM books;
    NAME                ID                  PRICE
============ ======================== ==============
C language    abc                            *9e+001
College engl* 2                              *3e+001
2 rows selected
```

3.15 ALTER TABLE DROP FOREIGN KEY

ALTER TABLE DROP FOREIGN KEY命令用于修改现有表的定义，删除以前定义的外键。只有表的所有者、DBA或对指定表有ALTER权限的用户才可以执行该命令。

键（*key*）是标识表中每条记录的字段或字段组合。这些组成键的字段被称做键字段（*key columns*）。唯一键（*unique key*）是指任何两条记录在该键域上的字段值都不相同。

主键（ *primary key*）是唯一标识表中每条记录的键。如果没有主键，用户就无法区分表中包含相同数值的两条记录。数据库管理系统（*DBMS*）不允许在数值相同的字段上创建主键或输入与现有主键值相同的数值。

外键（ *foreign key*）是对应于另外一张表的主键或索引的键。这样，通过两张表相应字段的相同值创建起主从（即参照）关系。主表（被参照表）中包含了主键或唯一索引，从表（参照表）中包含了对应主表主键的外键。

参照完整性（*Referential integrity*）确保了子键值，即从表（参照表）中的外键值对应于父键值，主表（被参照表）中的主键值或唯一索引是一致的。用外键创建的主从关系会强制执行两张表之间的参照完整性。通过外键的定义，DBMaster将自动支持两张表之间的参照完整性约束。要向从表中增加新记录，该从表中的值必须已存在于主表中。同样，若要从主表中删除记录，必须先删除子键中相对应的记录。

当子键参照主键时，参照完整性可能不允许对主键执行修改或删除，参照动作则提供了解决这种问题的办法。参照动作定义了当您删除或修改主键时，DBMaster在所有相应子键上应执行的操作。参照动作可以发生在修改和删除这两种动作上，DBMaster提供了四种参照动作：CASCADE、SET NULL、SET DEFAULT以及NO ACTION。CASCADE指修改和删除动作同时发生在从表的子键上；SET NULL指将从表中相应的子键设为空；SET DEFAULT指将从表中的相应子键设为字段默认值；NO ACTION指DBMaster强制执行参照完整性规则，即对于

主表中的修改和删除操作不予执行。创建外键时如果没指定参照动作，DBMaster将默认参照动作为无动作（NO ACTION）。

当用户不再需要表中的外键时，就可以使用ALTER TABLE DROP FOREIGN KEY命令将其删除。删除外键后，DBMaster将不会在从表（参照表）上强制执行参照完整性或参照动作。在没有外键的情况下，您可以在从表中输入主表中不存在的字段值或更新\删除主表中的字段值。使用该命令必须小心谨慎。

table_name ..... 要删除外键的表名。

key_name ..... 要删除的外键名。

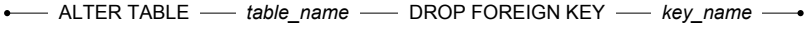


图 3-16 ALTER TABLE DROP FOREIGN KEY语法图

➲ **例**

下例是如何从**Salary**中删除名为**fkey**的外键。

```
dmSQL> ALTER TABLE Salary DROP FOREIGN KEY fkey;
```

3.16 ALTER TABLE DROP PRIMARY KEY

ALTER TABLE DROP PRIMARY KEY命令可用来修改现有表的定义并删除以前定义的主键。只有表的拥有者、DBA或在指定表上拥有ALTER和INDEX权限的用户才能执行该命令。

键（*key*）是标识表中每条记录的字段或字段组合。这些组成键的字段被称做键字段（*key columns*）。

唯一键（*unique key*）是指任何两条记录在该键域上的字段值都不相同。

主键（ *primary key*）是唯一标识表中每条记录的键。如果没有主键，用户就无法区分表中包含相同数值的两条记录。数据库管理系统（*DBMS*）不允许在数值相同的字段上创建主键或输入与现有主键值相同的数值。

外键（ *foreign key*）是对应于另外一张表的主键或索引的键。这样，通过两张表相应字段的相同值创建起主从（即参照）关系。主表（被参照表）中包含了主键或唯一索引，从表（参照表）中包含了对应主表主键的外键。

参照完整性（*Referential integrity*）确保了子键值，即从表（参照表）中的外键值对应于父键值，主表（被参照表）中的主键值或唯一索引是一致的。用外键创建的主从关系会强制执行两张表之间的参照完整性。通过外键的定义，DBMaster将自动支持两张表之间的参照完整性约束。要向从表中增加新记录，该从表中的值必须已存在于主表中。同样，若要从主表中删除记录，必须先删除子键中相对应的记录。

当用户不再需要表中的主键时，可以用ALTER TABLE DROP PRIMARY KEY 命令将其删除。创建外键时，DBMaster将会强制执行参照完整性。在删除主键之前必须先删除参照它的所有外键。删除主键后，DBMaster将不再要求每条记录要有唯一的键值，用户可以插入两条完全相同的记录，这可能导致数据库的不一致性，所以在执行该命令前，系统会提出警告。

table_name..... 要删除主键的表名。

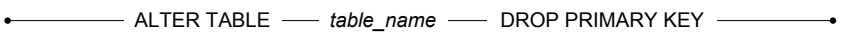


图 3-17 ALTER TABLE DROP PRIMARY KEY语法图

➲ **例**

下例是如何从**Employeesinfo**表中删除主键（**Primary Key**）。

```
dmSQL> ALTER TABLE Employeesinfo DROP PRIMARY KEY;
```

3.17 ALTER TABLE FOREIGN KEY

ALTER TABLE FOREIGN KEY命令用来修改现有表的定义并增加新的外键。要执行ALTER TABLE FOREIGN KEY命令，您必须是DBA或表的拥有者，拥有指定表的ALTER权限，或在包含主键的表和字段上拥有REFERENCE权限。

键（*key*）是标识表中每条记录的字段或字段组合。这些组成键的字段被称做键字段（*key columns*）。唯一键（*unique key*）表示任何两条记录在这键域上的字段值都不相同。

主键（ *primary key*）是唯一标识表中每条记录的键。如果没有主键，用户就无法区分表中包含相同数值的两条记录。数据库管理系统（*DBMS*）不允许在数值相同的字段上创建主键或输入与现有主键值相同的数值。

外键（ *foreign key*）是对应于另外一张表的主键或索引的键。这样，通过两张表相应字段的相同值创建起主从（即参照）关系。主表（被参照表）中包含了主键或唯一索引，从表（参照表）中包含了对应主表主键的外键。

当子键参照主键时，参照完整性可能不允许对主键执行修改或删除，参照动作则提供了解决这种问题的办法。参照动作定义了删除或修改主键时，DBMaster在所有相应子键上应执行的操作。参照动作可以发生在修改和删除这两种动作上，DBMaster提供了四种参照动作：CASCADE、SET NULL、SET DEFAULT以及NO ACTION。

关键字ON UPDATE/ON DELETE是可选的。这两个关键字表明DBMaster执行的参照动作是修改主键还是删除主键。参照动作分四种：CASCADE、SET NULL、SET DEFAULT和NO ACTION。

级联（CASCADE）指修改和删除动作同时发生在从表的子键上。当主表中某条记录的主键值发生更改时，从表中相应的外键值将发生同样的更改；或者在主表中删除某条记录的主键值时，从表中相应的外键值也将被删除。

设为空值（SET NULL）是指当您更改或删除记录的主键时，DBMster将相应的子键设为空值（NULL）。如果您在子键上设定了不为空（NOT NULL）的限制，那么在这里您就不能使用SET NULL动作。

设为默认值（SET DEFAULT）是指当您更改或删除记录的主键时，DBMaster将相应子键设为字段默认值。如果您在定义字段时，设置子键的默认值为空值（NULL），并且又在定义子键的字段时设定了不为空的限制，那么在这里您就不能使用SET DEFAULT动作。

无动作（NO ACTION）将使DBMaster执行一般的参照完整性规则。创建外键时如果没指定参照动作，DBMaster将参照动作默认为无动作（NO ACTION）。

一张表中的外键数一般是没有限制的。父键可能是主键，也可能是唯一性索引，但父键必须在子键之前创建。父键和子键的字段数量、数据类型和字段长度必须一致。两张表中两组键的字段顺序可以不同，不过在使用ALTER TABLE FORIGN KEY命令时应按主键的字段顺序列出外键字段。一般主表中的主键字段可以不列出，由系统默认设置。

外键中的字段可以有空值。如果外键含有空值，那么它将自动满足参照完整性。您可以在同义字上创建外键，不过不能在视图上创建。外键名的最大长度不能超过128个字符，它可以包含数字、字母、下划线以及 $ 和#符号，但不能以数字开头。

table_name ..... 要添加外键的表名。

key_name ..... 新外键名。

column_name ..... 1. 包含在外键中的字段名。

..... 2. 被外键参照的字段名。

parent_table_name.....外键参照的表名（主表或被参照表）。

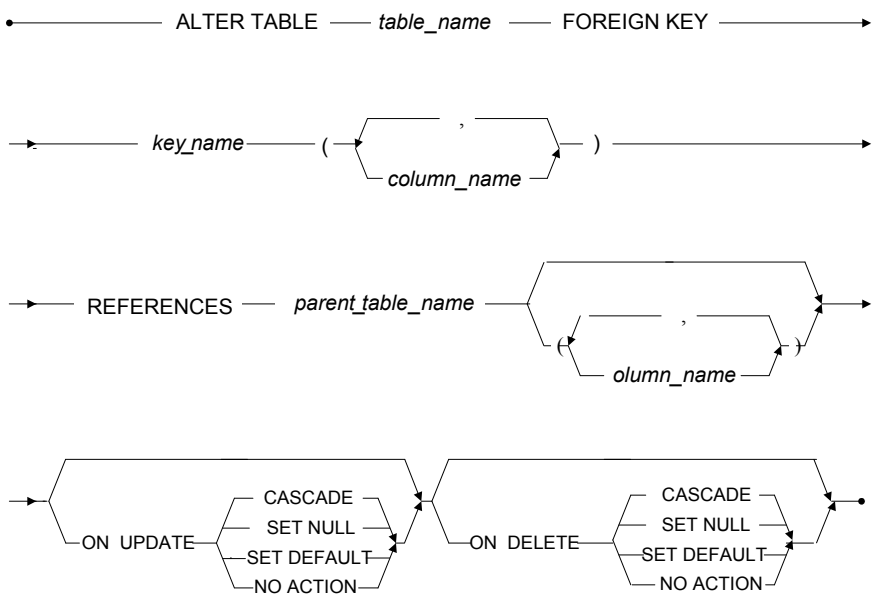


图 3-18 ALTER TABLE FOREIGN KEY 语法图

➲ **例1**

下例是如何在表**Accounts**的**CustNo**字段上创建一个名为**fkey_CNo**的外键，主表为**Customers**。本例中没有给出父键的字段，DBMaster将**Customers**表中的主键默认为父键。需要注意的是在执行该命令前，表**Customers**中的主键必须先创建。

```
dmSQL> ALTER TABLE Accounts FOREIGN KEY fkey CNo (CustNo)
                        REFERENCES Customers;
```

➲ **例2**

下例同上，只是明确指出将**CustNo**字段作为父键。**CustNo**字段可以是**Customers**表的主键或唯一性索引，但必须在执行该命令前就创建好。

```
dmSQL> ALTER TABLE Accounts FOREIGN KEY fkey_CNo (CustNo)
                        REFERENCES Customers (CustNo);
```

➲ **例3**

下例是如何在**Invoice**表的**PartNo**和**StockNo**字段上创建一个名为**fkey_No**的外键，主表是**Stock**。表**Invoice**中的字段顺序是（**PartNo**，**SuppNo**），这与表**Stock**中相应字段的顺序不一致，后者的顺序是（**SuppNo**，**PartNo**）。但只要在使用命令时，表字段列表中相应字段的顺序相同，那么命令还是可以正确执行的。

```
dmSQL> ALTER TABLE Invoice FOREIGN KEY fkey_No (SuppNo, PartNo)
                    REFERENCES Stock (SuppNo, PartNo);
```

➲ **例4**

下例同上，只是定义了DBMaster需要执行的参照动作。关键字**ON UPDATE SET DEFAULT**指明DBMaster在更新或删除记录中的父键值时，将相应的子键值设为字段的默认值。关键字**ON DELETE SET NULL**指明DBMaster在删除父键时，将相应的子键值设为空值（**NULL**）。

```
dmSQL> ALTER TABLE Invoice FOREIGN KEY fkey_No (SuppNo, PartNo)
                     REFERENCES Stock (SuppNo, PartNo)
                     ON UPDATE SET DEFAULT
                     ON DELETE SET NULL;
```

3.18 ALTER TABLE MODIFY COLUMN

ALTER TABLE MODIFY COLUMN命令用来修改表中现有字段的定义。只有表的拥有者、DBA、SYSDBA、SYSADM或拥有指定表ALTER权限的用户才可以执行该命令。

table_name .....…要修改字段的表名。

column_name ....…要修改的字段名。

column_definition…新定义的字段。

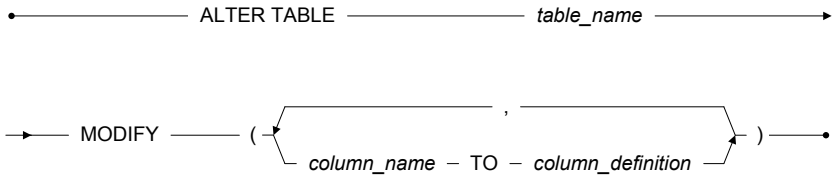


图 3-19 ALTER TABLE MODIFY COLUMN语法图

字段定义

定义字段时需要指明字段名、字段的数据类型或者所属定义域。您也可以在一条命令中修改多个字段，直到达到最大字段数（252个）。

为每个要修改的字段指定其数据类型。DBMaster支持以下数据类型：BINARY、CHAR、DATE、DECIMAL、DOUBLE、FILE、FLOAT、INTEGER、BLOB、CLOB、OID、SERIAL、SMALLINT、TIME、TIMESTAMP和VARCHAR。

您可以不使用标准的数据类型，而使用自定义的定义域来定义字段。定义域可用来设定字段的数据类型和默认值以及字段的完整性约束。您可以使用关键字DEFAULT和CHECK来定义字段的默认值和完整性约束（下面会提到这两个关键字的具体用法）。如果您在定义字段时重新设

定了字段的默认值，那么字段的默认值就会是新设定的值；如果您在定义字段时定义了完整性约束，那么该字段的条件约束将是定义域中的约束加新定义的约束。

关键字NULL/NOT NULL是可选的，这两个关键字决定了在插入一个字段时其值是否可以为空（NULL）。NULL关键字指在插入新字段时，可以不输入数值。NOT NULL关键字指在插入新字段时必须输入数值。如果以前的字段使用了NULL关键字，那么除非表为空或使用了GIVE关键字，否则修改字段时不能使用关键字NOT NULL。

关键字USER/SYSTEM是可选的，这两个关键字决定了用户是否可以使用INSERT/UPDATE语句来更改字段的默认值。若用户没有设置该关键字，则默认为USER。当使用USER关键字时，用户可以更改字段的默认值；而使用SYSTEM关键时，用户不能更改字段的默认值。

关键字DEFAULT也是可选的。当插入一条新记录时，如果没有输入字段值，就可以将DEFAULT关键字所指定的值作为字段的默认值。常数、内置函数的值以及空值（NULL）都可以作为字段的默认值。其中只有不含自变量的内置函数，如PI( )、NOW( )或USER( )才能作为定义的默认值。如果用NULL作为默认值，那么在定义字段时就不能再使用关键字NOT NULL。当用户在定义字段时使用的是定义域而不是标准的DBMaster数据类型时，最好不要再使用DEFAULT关键字，因为定义域中一般都包括默认值（DEFAULT）。

关键字ON UPDATE也是可选的。该关键字用于设置字段的默认值是否随其它字段值的更新而自动更新。

关键字CHECK也是可选的，用来指明输入字段值的取值范围或字段的条件约束。CHECK后面的表达式一般都是取值为真（true）或假（false）的表达式。可以将关键字VALUE和CHECK结合起来代表字段值。如果SQL命令不满足CHECK所设定的条件，它就不能被执行。如果用户使用定义域而不是标准的DBMaster数据类型来定义字段，那么就可以不使用关键字CHECK，因为定义域中一般都包括了它们自己的CHECK条件约束。

关键字GIVE是可选的, 用户可用它来设定插入到表中的每条记录的修改字段的值。如果您试图将字段从NULL更改为NOT NULL，并且又没有用

GIVE来设定更改值，那么DBMaster将拒绝更改字段。同样，在将一个字段更改为SERIAL类型时，您可以将关键字SEQUENTIAL/SEQ和GIVE结合起来使用。这些关键字表明DBMaster将从定义SERIAL字段时所指定的开始值起，向现有记录中插入序列值。这些序列值随着新记录的插入而不断增加。

关键字BEFORE/AFTER是可选的，这两个关键字用于指明要修改的字段相对于其他字段的位置。如果使用BEFORE关键字，那么DBMaster会修改在指定字段的前面（紧挨的左边）的字段。如果使用AFTER关键字，那么DBMaster会修改在指定字段后面的（紧挨的右边）字段。如果您没有使用这两个关键字指定修改字段的相对位置，那么DBMaster将保留它的位置。

更改表中的字段将会影响表中的所有视图和存储命令，但对表上创建的同义字不产生任何影响。字段名最大长度不能超过128个字符，可以包含字母、数字、下划线以及$和#符号，但不能以数字开头。

column_name .....…要修改的字段名。

data_type .....修改后的字段数据类型。

domain_name ....….....修改后字段所属的定义域。

Literal .....….....没有给字段输入值时，系统使用的默认数值。

constant .....….....没有给字段输入值时，系统使用的默认常量。

function_name ...….....没有给字段输入值时，系统使用的内置函数名。

constraint_name ….....应用于字段的约束名。

boolean_expression…取值为真假的表达式。

column_name_a .....表中的字段，修改后的字段放在它的后面。

column_name_b .....表中的字段，修改后的字段放在它的前面。

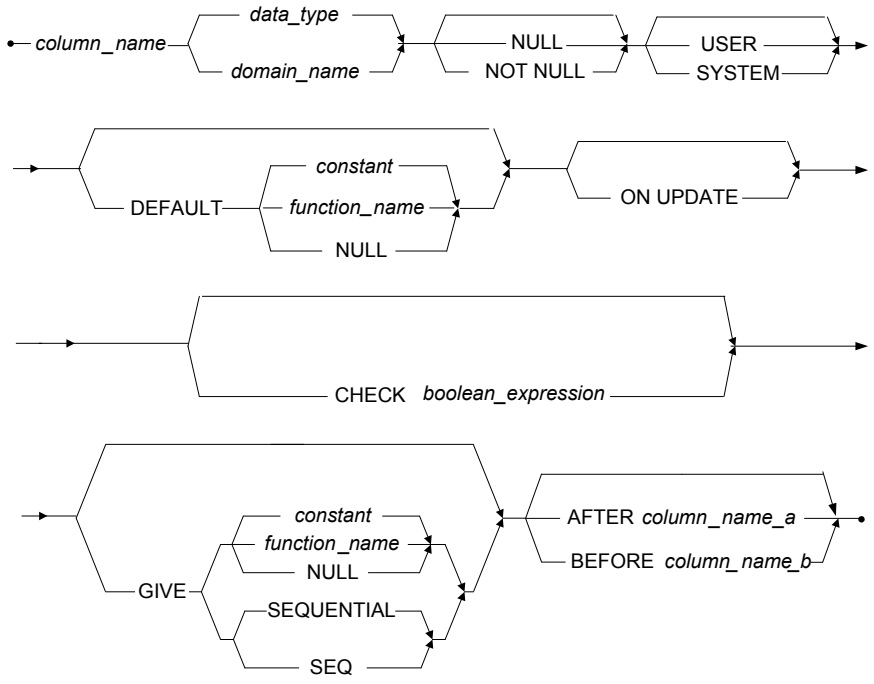


图 3-20 字段定义语法图

➲ **例1**

下例表示如何修改表**Employeesinfo**中的**Phone**字段长度，方法是将字段的数据类型从**CHAR(15)**更改为**CHAR(20)**。

```
dmSQL> ALTER TABLE Employeesinfo MODIFY (Phone TO Phone CHAR(20));
```

➲ **例2**

下例同上，只是增加了**NOT NULL**关键字，表示为新插入的字段输入数值。如果原有记录在这些字段上含有空值（**NULL**），那么系统将自动设定这些字段的值为**GIVE**关键字所指定的值。

```
dmSQL> ALTER TABLE Employeesinfo MODIFY (Phone TO Phone CHAR(20)
                                         NOT NULL
                                         GIVE '000-0000');
```

➲ **例3**

下例表示如何更改表**LineItems**中，字段**Quantity**和**Amount**的数据类型，将它们的数据类型从**SMALLINT**更改为**INT**。

```
dmSQL> ALTER TABLE LineItems MODIFY (Quantity TO Quantity INT,
                                     Amount TO Amount INT);
```

3.19 ALTER TABLE MODIFY DYNAMIC COLUMN

ALTER TABLE MODIFIY COLUMN命令用来更改动态字段的现有描述信息。只有表的所有者、DBA、SYSDBA、SYSADM或对表有ALTER权限的用户才可以执行该命令。

动态字段仅支持更改数据类型。

有关动态字段的详细信息，请参考*数据库管理员手册*中的*使用动态字段*章节。

table_name..... 要修改描述信息的动态字段所在表的表名。

column_name.... 要修改描述信息的动态字段的字段名。

data_type..... 用于更改的动态字段数据类型。

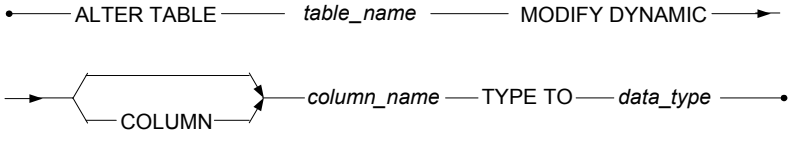


图 3-21 ALTER TABLE MODIFY DYNAMIC COLUMN语法图

➲ **例**

下例说明如何删除名为**price**的动态字段的描述信息。有关表**books**的详细信息，请参考*ALTER TABLE ADD DYNAMIC COLUMN*章节。

```
dmSQL> ALTER TABLE books MODIFY DYNAMIC COLUMN price TYPE TO INT;
dmSQL> SELECT name,id,price FROM books;
    NAME                ID                  PRICE
============ ======================== ==============
C language    abc                               19
College engl* 2                                 32
2 rows selected
```

3.20 ALTER TABLE PRIMARY KEY

ALTER TABLE PRIMARY KEY命令用于来更改现有表的定义并为表增加一个主键。只有表的拥有者、DBA或者拥有该表ALTER和INDEX权限的用户才可以执行这条命令。

键（*key*）是标识表中每条记录的字段或字段组合。唯一键（*unique key*）表示任何两条记录在键域上的字段值都不相同。

主键（ *primary key*）是唯一标识表中每条记录的键。如果没有主键，用户就无法区分表中包含相同数值的两条记录。数据库管理系统（*DBMS*）不允许在数值相同的字段上创建主键或输入与现有主键值相同的数值。

外键（ *foreign key*）是对应于另外一张表的主键或索引的键。这样，通过两张表相应字段的相同值创建起主从（即参照）关系。主表（被参照表）中包含了主键或唯一索引，从表（参照表）中包含了对应主表主键的外键。

参照完整性（*Referential integrity*）确保了子键值，即从表（参照表）中的外键值对应于父键值，主表（被参照表）中的主键值或唯一索引是一致的。用外键创建的主从关系会强制执行两张表之间的参照完整性。通过外键的定义，DBMaster将自动支持两张表之间的参照完整性约束。要向从表中增加新记录，该从表中的值必须已存在于主表中。同样，若要从主表中删除记录，必须先删除子键中相对应的记录。

主键要求每条记录的键值都具有唯一性，这就保证了表中数据的完整性。因为主键的值不能相同且不能为空值，所以在定义主键时应设置不为空（NOT NULL）的约束条件。

每张表只能拥有一个主键，因此您不能为主键命名，DBMaster会在系统内部为每张表的主键生成一个名为PrimaryKey的唯一性索引，并且自动维护该索引。因为DBMaster会自动在主键上创建索引，所以您不必在主键字段上另建索引以试图提高查询效率。

主键最多可包含32个字段，且这些字段所占的空间不能超过4000个字节。您可以在同义字上创建主键，但不能在视图上创建主键。在同义字上创建主键实际上就是在基本表上创建主键。

table_name..... 要创建主键的表名。

column_name.... 主键包含的字段名。

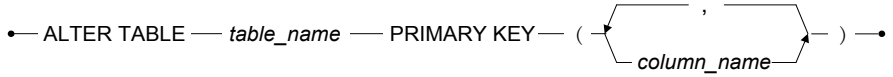


图 3-22 ALTER TABLE PRIMARY KEY语法图

➲ **例**

下例是如何在表**Customers**中的**CustNo**字段上创建一个主键。其中的字段**CustNo**必须定义**NOT NULL**约束条件，并且每条记录在**CustNo**字段上的值唯一或表为空。

```
dmSQL> ALTER TABLE Customers PRIMARY KEY (CustNo);
```

3.21 ALTER TABLE RENAME

ALTER TABLE RENAME命令用来更改现有表的表名。只有表的拥有者、DBA以及拥有指定表ALTER权限的用户才可以执行该命令。

当表上仅有索引和/或全文索引时，就可以修改表名。如果有与表相关的存储命令、存储过程、触发器、外键等存在的情形，则不支持RENAME命令。

table_name .....….要修改的表名。
new_table_name….修改后的表名。

图 3-23 ALTER TABLE RENAME语法图

3.22 ALTER TABLE SET OPTIONS

ALTER TABLE SET OPTIONS命令用来更改现有表的定义并更改表的选项。只有表的拥有者、DBA或拥有指定表ALTER权限的用户才能执行该命令。

在存取表中数据时，可利用LOCK MODE来指定DBMaster采用的锁模式（即锁级别）。DBMaster有三种锁模式：表、页和行。页锁定是系统默认的锁模式。用户可以在SYSTABLE表中查询字段LOCKMODE来确定表的锁定模式。

LOCK MODE TABLE的作用是锁定整张表，该模式使用户不能同时存取被锁定的表，这将降低事务的并发性。但是这种锁定模式使用的锁资源较少，因此在系统控制区中占用的存储空间较少。

LOCK MODE PAGE的作用是锁定一个数据页，该模式是对事务并发性和锁资源的折中考虑。这种模式提供了适当的并发性，因为其他用户也可以存取锁定页以外的数据页。

LOCK MODE ROW的作用是锁定一行，该模式允许用户存取锁定行以外的其它数据，因此这种模式比页锁定更进一步地提高了事务的并发行。但这种锁定模式与前两种模式相比，占用的锁资源更多，并且在系统控制区中占用的存储空间较多。

填充系数（FILLFACTOR）指数据页被填充的百分比。这种为将来更新记录保留一定存储空间的方法，提高了数据页中存取数据的效率。参数*number*的取值范围为50—100，因此填充系数的值可设为50%—100%。您可以查询系统表SYSTABLE中的FILLFACTOR字段来获得表的填充系数。

无缓存（NOCACHE）可用来限制表扫描时，分配给该表的数据页缓冲区数量。DBMaster将所有的缓冲数据页存储为缓冲链的形式，使用最频繁的数据页将排在链头部而最不常用的数据页将排在末端。如果选择了NOCACHE这一选项，扫描表时存取的数据页将被放置在缓冲区链的末端。缓冲区链末端的内容在下一次存取数据前会被覆盖掉，并且在表扫描的过程中，后来存取的数据页会覆盖先前的数据页。在表上设置无缓

存以后，DBMaster在扫描表时，就只会使用一个数据页来充当缓冲区存放表中的数据。您可以查询系统表SYSTABLE中的CACHEMODE字段来获得表的缓冲模式。

序列数（SERIAL）选项用来重置序列字段的计数。这样您就可以不必修改表而使序列字段从一个新的数字开始计数。

ALTER TABLE SET OPTIONS命令对这个表上所创建的视图和同义字不产生任何影响。

table_name .....要修改选项的表名。

number.....填充系数（FILLFACTOR）的值。

n .....两次统计表发生更新的间隔天数。

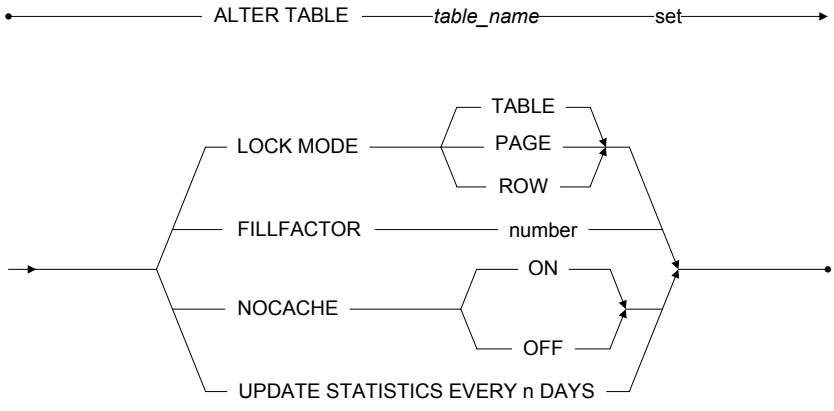


图 3-24 ALTER TABLE SET OPTIONS语法图

➲ **例1**

下例是如何将表**Customers**的锁定模式（**LOCK MODE**）设置为表（**TABLE**）锁定。

```
dmSQL> ALTER TABLE Customers SET LOCK MODE TABLE;
```

➲ **例2**

下例是如何将表**Customers**的锁定模式（**LOCK MODE**）设置为页（**PAGE**）锁定。

```
dmSQL> ALTER TABLE Customers SET LOCK MODE PAGE;
```

➲ **例3**

下例是如何将表**Customers**的锁定模式（**LOCK MODE**）设置为行（**ROW**）锁定。

```
dmSQL> ALTER TABLE Customers SET LOCK MODE ROW;
```

➲ **例4**

下例是如何将表**Customers**的填充系数（**FILLFACTOR**）设为**90%**。

```
dmSQL> ALTER TABLE Customers SET FILLFACTOR 90;
```

➲ **例5**

下例是如何开启表**Customers**的无缓存（**NOCACHE**）选项。

```
dmSQL> ALTER TABLE Customers SET NOCACHE ON;
```

➲ **例6**

下例是如何取消表**Customers**的无缓存（**NOCACHE**）选项。

```
dmSQL> ALTER TABLE Customers SET NOCACHE OFF;
```

➲ **例7**

下例是如何将表**tb_tmp**的**SERIAL**字段的开始计数值更改为**100**。

```
dmSQL> ALTER TABLE tb_tmp SET SERIAL 100;
```

3.23 ALTER TABLE TO ANOTHER TABLESPACE

ALTER TABLE TO ANOTHER TABLESPACE命令用于将表移动到另一个表空间，若索引和表位于相同的表空间，可同时将索引移动到另一个表空间；若索引和表位于不同的表空间，则无法将索引移动到另一个表空间。只有表的拥有者、DBA或拥有该表ALTER和INDEX权限的用户才有权执行该命令。

设置FASTCOPY ON，用户可加快将表移动到另一个表空间的速度。移动表时，系统将直接将数据页复制到另一个数据页。在一次复制过程中，数据库仅操作一次日志文件且无需使用缓存，因此减少了大量重复的日志操作。

移动表到另一个表空间可将该表存储到其它的磁盘，以免磁盘已满时无法存储数据。

将表移动到其它表空间存在以下限制：

- 不能移动系统表、临时表或视图到其它表空间；
- 不能移动永久表到系统表空间或临时表空间；
- 永久表的索引不能创建在临时表空间中；
- 临时表的索引只能创建在临时表空间中；
- 系统表的索引只能创建在系统表空间中；
- 不能在一个表内复制数据。

table_name .....……将要移动的表名。

tablespace_name……表将要被移动到的表空间名。

图 3-25 ALTER TABLE TO ANOTHER TABLESPACE语法图

➲ 例

下例将位于表空间**ts_mode**中名为**Employeesinfo**的表移动到表空间**ts_new**。

```
dmSQL> ALTER TABLE Employeesinfo MOVE TABLESPACE ts_new;
```

3.24 ALTER TABLESPACE

ALTER TABLESPACE命令可为现有的表空间增加文件，也可用来将表空间的类型在自动扩展与固定表空间，或在只读和读写表空间之间转换。只有DBA、SYSDBA或SYSADM才有权执行该命令。

对于大多数用户而言，他们并不关心数据在计算机上的物理存储模式。DBMaster将物理存储模式的细节隐藏在关系型数据模式下，因此用户看到的只是数据的逻辑存储模式。

在DBMaster的物理存储模式中，文件是数据库中数据的物理存储结构。文件一般是由操作系统管理的，不过在UNIX的裸设备中，文件中的数据是由数据库管理系统管理的。DBMaster在日常操作中使用三种类型的文件：数据文件（Data）、二进制大对象（BLOB）和日志文件（Journal）。

日志文件是一种特殊的文件，它对数据库所做的更改提供了实时或历史记录，并且记录了数据库更改后的状态。使用日志文件可使数据库能撤销失败事务对数据库所做的更改，或在数据库执行灾难恢复时，将已提交的事务重新写入数据库。日志文件只能由数据库管理系统（DBMS）使用，它并不能存储用户数据。

数据文件和BLOB文件用于存储用户和系统数据。尽管这两种文件的特性相似，但DBMaster对它们的管理方式却不尽相同，这样有助于提高系统的性能。数据文件用于存储表和索引数据，而BLOB文件则用于存储二进制大对象。

在DBMaster的逻辑存储模式中，表空间是数据的逻辑存储结构，可将数据库分割成几个DBMS容易管理的区域。一个表空间可能包含几个表和索引，表空间中的数据物理存储在文件中，但由数据库管理系统统一管理。表空间分为五种：固定表空间、自动扩展表空间、系统表空间、只读表空间和读写表空间。

固定表空间是大小固定的表空间，可以包含一个或多个数据文件或BLOB文件。您可以通过扩大表空间中文件的大小或向表空间中增加文件的方式来手动扩展表空间。在新增文件之前，应首先在配置文件

dmconfig.ini中设置文件的逻辑文件名、物理文件名以及文件的初始大小。一个固定表空间中最多可包含32767个文件，所有文件所占空间的总和不能超过8TB。在UNIX中，固定表空间可以放在裸设备上。

注意 *有关裸设备的详细信息，请参照Unix系统文档。*

自动扩展表空间是视需要而自动延伸的表空间，其中至少要包含一个数据文件，也可用来存储BLOB文件。自动扩展表空间可以自动扩展表空间的大小，这是与固定表空间有所不同的。DBA可对这两种类型表空间中的表进行组织管理，在向自动扩展表空间中新增文件之前，您应该先在配置文件**dmconfig.ini**中设置文件的逻辑文件名、物理文件名以及文件的初始大小。但要注意自动扩展表空间不支持裸设备。

创建数据库时，DBMaster会自动生成系统表空间。每个数据库只有一个系统表空间，用来存储系统表，而系统表是用来记录存储模式、安全和状态信息的。如果不是在Unix裸设备上创建的数据库，那么系统表空间就是一个自动扩展表空间。系统表空间包含一个数据文件和一个BLOB文件。系统表空间也可以更改成固定表空间。

你可以使用关键字SET AUTOEXTEND OFF将自动扩展表空间更改为固定表空间。为了限制表空间所占磁盘空间的大小，您可以将自动扩展表空间更改为固定表空间。

注意 *只要还有磁盘空间，自动扩展表空间中的文件就可以不断扩展大小，但文件的总大小不能超过8TB。*

只读表空间不允许用户在表空间中执行任何修改的操作，然而，只读表空间自有它的优点：

- 不必经常备份。只读表空间在它设为只读之后仅需要做一次备份即可。
- 数据库的恢复变得更容易 。当发生故障时，只读表空间的优点是不需要从介质故障中恢复。
- 只读的表空间比可更改的表空间占有更少的系统资源。（没有锁资源）

可以使用SET READ ONLY命令把一个可读写的表空间改为只读表空间。

而通过SET READ WRITE命令把只读表空间改为可读写的表空间。

关键字ADD DATAFILE可用来向表空间中新增数据文件或BLOB文件，向表空间中新增的文件并不一定位于同一物理磁盘上。在Unix系统中，文件可以存储在裸设备中，这时DBMaster可以不通过操作系统调用而直接向裸设备中写入文件，这种方式提高了在普通文件上的读取速度和操作效率。

正如前面所提到的，构成表空间的文件在数据库中是用逻辑文件名来参照的，这将有利于保持数据的物理独立性。DBMaster在配置文件**dmconfig.ini**中将逻辑文件名映射为物理文件名，下面将有例子说明。新建文件时，如果没有另外指定目录或路径，DBMaster会将文件存储在系统默认的数据库目录下，该默认目录是由**dmconfig.ini**文件中的**DB_DbDir**关键字所指定的。

逻辑文件名不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。物理文件名包括驱动器名和路径名在内，最长不能超过256个字符，它可以包含任何操作系统允许的字符，但要注意物理文件名中不能包含空格。

新增文件时，用户可以使用关键字TYPE=DATA和TYPE=BLOB来指定文件类型，文件的默认类型是数据文件。

同时，您还可以指定文件的大小：数据文件中包含的数据页数或BLOB文件中包含的BLOB帧数。一个数据页的大小可以是4KB、8KB、16KB或32KB，而一个BLOB帧的大小是8KB到256KB。DBMaster可根据需要扩充自动扩展表空间的大小。如果您想知道BLOB的帧大小，可查看数据库配置文件**dmconfig.ini**中的关键字**DB_BfrSz**。

tablespace_name…要修改的表空间名。

file_name .....….要在表空间中新增的文件名。

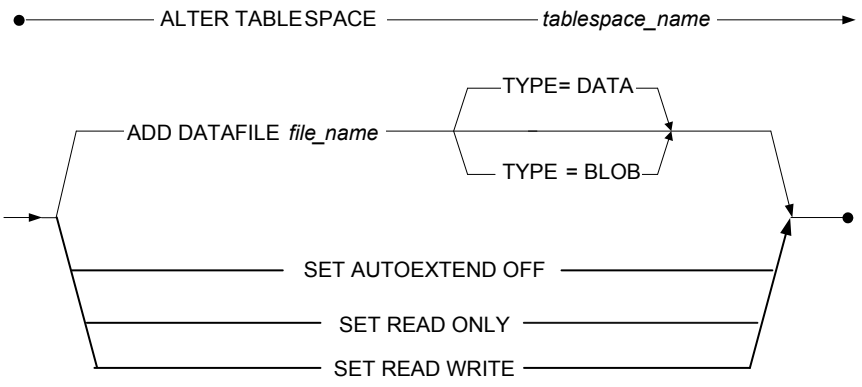


图 3-26 ALTER TABLESPACE语法图

➲ **映射1**

您在执行例1的命令前，需要在**dmconfig.ini**文件中加上如下代码，用于将逻辑文件名映射为物理文件名，并且指定数据文件所包含的页数。如果设定的每一页大小为4KB，那么下述文件则占400KB的空间。

```
file1=c:\dbmaster\databases\f1.db 100
```

➲ **例1**

下例向表空间**ts_new**中新增一个名为**f1.db**的文件，根据映射1的设定，**f1.db**的逻辑文件名为**file1**。

```
dmSQL> ALTER TABLESPACE ts_new ADD DATAFILE file1 TYPE=DATA;
```

➲ **映射2**

您在执行例2中的命令之前，需要在**dmconfig.ini**文件中增加如下代码，用来将逻辑文件名映射为物理文件名，并且指定BLOB文件所含的帧数。在本例中，因为使用了默认的帧大小，即8KB，所以该BLOB文件的大小为4000KB。

```
file2=c:\dbmaster\databases\f2.bb 500
```

➲ **例2**

下例是将表空间的类型从自动扩展更改为固定表空间以及如何向表空间**ts_mode**中新增一个名为**f2.bb**的文件。根据映射2的设定，**f1.bb**的逻辑文件名为**file2**。

```
dmSQL> ALTER TABLESPACE ts mode SET AUTOEXTEND OFF;
dmSQL> ALTER TABLESPACE ts_mode ADD DATAFILE file2 TYPE=BLOB;
```

➲ **例3**

下例将表空间的类型从可读写改为只读表空间。

```
dmSQL> ALTER TABLESPACE ts_mode SET READ ONLY;
```

3.25 ALTER TABLESPACE DROP DATAFILE

ALTER TABLESPACE DROP DATAFILE 命令将从表空间中删除空数据文件。只有DBA、SYSDBA或者SYSADM才可以执行该命令。

当从表空间中删除数据文件时，您必须确保该文件是空的。如果文件中包含数据，该命令将被中止并且返回一个错误信息。同时，用户不能删除表空间中的唯一文件。也不能删除系统表空间中的系统数据文件以及默认表空间中的默认的数据文件。

该命令只能删除逻辑文件，所以在提交该命令后，用户需要手动删除物理文件和dmconfig.ini中的信息。

tablespace_name... 数据文件所属的表空间名。

file_name.....要删除的数据文件名。

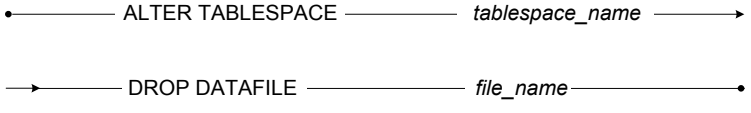


图 3-27 ALTER TABLESPACE DROP DATAFILE语法图

➲ **例**

下例将从表空间**ts_new**删除数据文件**tsfile1**。

```
dmSQL> ALTER TABLESPACE ts_new DROP DATAFILE tsfile1;
```

3.26 ALTER TRIGGER ENABLE

ALTER TRIGGER ENABLE命令可用来设定触发器是处于开启（ENABLE）还是禁用（DISABLE）状态。只有表的所有者、DBA、SYSDBA、SYSADM才可以执行该命令。

触发器是一种数据库服务器机制，所设定的事件一旦发生，就会导致触发器自动执行一些事先定义好的指令。这可以使数据库执行一些标准SQL指令不能执行的复杂或非常规操作。因为触发器是由数据库服务器控制的，所以无论事件是由哪个用户或哪个应用程序所造成的，触发器都能保证数据的一致性。每次当用户或应用程序造成一个触发事件时，DBMaster将自动执行触发器中预先定义的指令，用户和应用程序不必对此操心。

创建一个触发器后，该触发器将自动处于启用状态。当您在测试某些会导致触发器执行的数据库操作时，可以使用DISABLE关键字来禁用触发器。禁用触发器并不是从数据库中删除触发器，您可以使用关键字ENABLE来重新启用该触发器。

trigger_name.....要启用或禁用的触发器名。

table_name .....与触发器相关的表名。

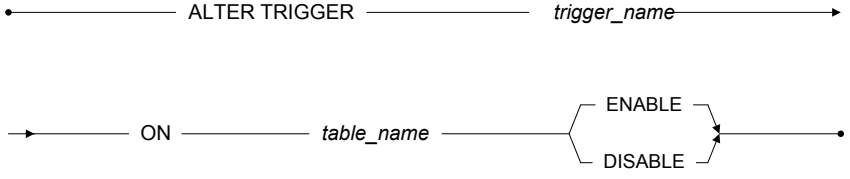


图 3-28 ALTER TRIGGER ENABLE语法图

➲ **例1**

下例是如何禁用在表**Employeesinfo**上的触发器**Trig_emp**。

```
dmSQL> ALTER TRIGGER Trig_emp ON Employeesinfo DISABLE;
```

➲ **例2**

下例是如何启用在表**Employeesinfo**上的触发器**Trig_emp**。

```
dmSQL> ALTER TRIGGER Trig_emp ON Employeesinfo ENABLE;
```

3.27 ALTER TRIGGER REPLACE

ALTER TRIGGER REPLACE命令可用来替换触发器。只有表的所有者、DBA、SYSDBA、SYSADM才可以执行该命令。

触发器是一种数据库服务器机制，所设定的事件一旦发生，就会导致触发器自动执行一些事先定义好的指令。这可以使数据库执行一些标准SQL指令不能执行的复杂或非常规操作。因为触发器是由数据库服务器控制的，所以无论事件是由哪个用户或哪个应用程序所造成的，触发器都能保证数据的一致性。每次当用户或应用程序造成一个触发事件时，DBMaster将自动执行触发器中预先定义的指令，用户和应用程序不必对此操心。

在修改触发器或替换触发器时需要指定触发器名，还应该指出新的触发动作、触发时间、触发事件、触发表以及触发器类型。

注意 *ALTER TRIGGER REPLACE命令只作用于要更改或要替换的触发表。*

触发器不同于其它数据库对象，DBMaster并不是只将触发器名作为触发器的名称，而是将触发器名和表名结合来作为触发器的完整名称。所以同一张表上的触发器名必须唯一。触发行为中所定义的SQL操作是以触发器创建者的安全级别和对象权限来执行的，并非以执行触发事件的使用者权限范围来执行。

用户可使用关键字BEFORE/AFTER来指明数据库服务器应何时根据触发事件执行相应地触发动作以及执行该触发动作的时间。关键字BEFORE表示数据库服务器应该在触发事件发生之前执行触发动作，而关键字AFTER表示数据库服务器应该在触发事件发生之后执行相应的触发动作。

关键字INSERT/DELETE/UPDATE表示可引起触发器执行的触发事件。使用INSERT/DELETE关键字和使用关键字UPDATE时有些不同。INSERT指在表中新增记录时触发相应的动作；DELETE指从表中删除记录时触发动作；UPDATE指更新表的字段时触发相应的动作。您可以使

用“UPDATE OF 字段列表”这样的格式来指定触发事件是在哪些字段上执行更新。

注意 *一个更新触发器中的字段名必须是唯一的。*

关键字ON后面的表名是需要替换触发器的触发表名。触发表必须是数据库中永久存在的表，而不能是临时表、视图或同义字。

trigger_name..... 需要做替换的触发器名。

column_name .... 更新触发事件所更新的字段名。

table_name ..... 新的触发器的触发表名。

sql_statement .... 触发事件中要执行的指令。

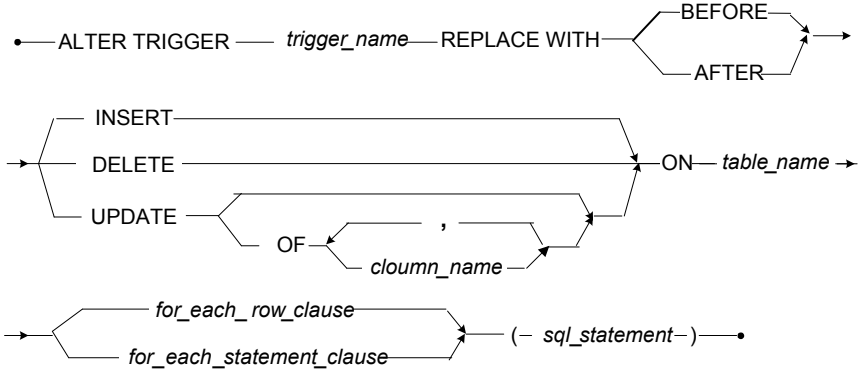


图 3-29 ALTER TRIGGER REPLACE语法图

For Each Row子句

关键字REFERENCING可用来为关键字OLD和NEW定义别名。当您创建一个行触发时，您需要在触发动作中指明是否需要参照触发事件发生之前或之后字段中的值。如果您的表名恰好就是OLD或NEW，那么您就可以使用REFERENCING关键字指代的别名来代替关键字OLD和NEW。

FOR EACH ROW子句代表行级触发，即如果触发事件里有一笔记录被更改，那么触发动作就会执行一次。如果触发事件不是针对记录而进行的，那么关键字FOR EACH ROW定义的行触发器将不会被执行。

关键字WHEN表示当指定记录符合条件时，DBMaster将执行一次触发。在发生触发事件时，WHEN子句会对每条记录进行估算。如果估算的结果为真，那么对这条记录的触发动作将会执行，如果估算的结果为假，则触发动作将被跳过。WHEN条件的结果只会影响触发动作的执行，而不会影响激活触发事件的SQL语句。

old_name .....…触发动作执行之前，用来参照字段旧值的别名。

new_name .....…触发动作执行之后，用来参照字段新值的别名。

search_condition …触发动作执行前需要满足的条件。

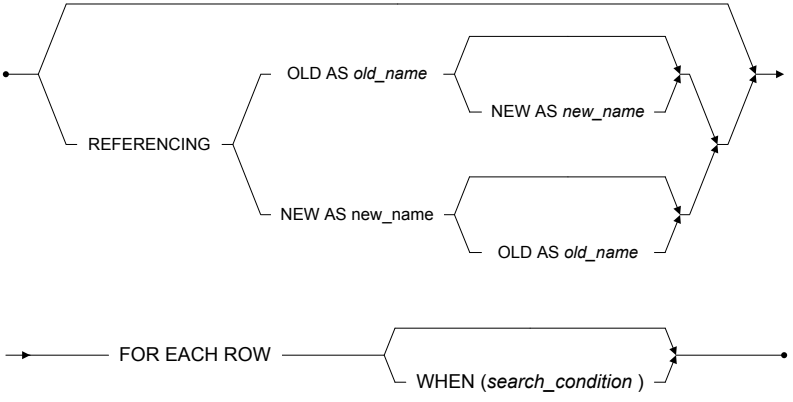


图 3-30 For Each Row Clause语法图

For Each Statement子句

关键字FOR EACH STATEMENT指语句触发，表示针对每一个触发事件，触发动作只会执行一次。如果使用该关键字，即使触发事件的语句没有对记录进行操作，也可以执行触发动作。

触发器执行的语句称为触发动作（*trigger action*）。触发动作可以是INSERT、UPDATE、DELETE或是执行存储过程（EXECUTE PROCEDURE）的语句。定义触发动作时，执行的程序只可以是诸如PI( )、NOW( )或USER( )这种不需要输入自变量的内置函数。触发器要执行的存储过程中不能包含COMMIT、ROLLBACK或SAVEPOINT等事务控制语句。

在每个触发事件上，您可以使用触发动作时机（包括关键字BEFORE和AFTER）和触发型态（包括关键字FOR EACH ROW 和FOR EACH STATEMENT）来产生四种触发。例如，您可以为新增（INSERT）触发事件产生四种触发：BEFORE/FOR EACH STATEMENT、BEFORE/FOR EACH ROW、AFTER/FOR EACH ROW以及AFTER/FOR EACH STATEMENT。

注意　　*Update和Delete触发事件支持以上触发时刻。*

触发事件定义时如果没有使用表更新（UPDATE），而使用的是字段更新（UPDATE OF），那么可以用*trigger action time/trigger type combination*模式来为每个字段产生一种触发。如果一张表有四个字段，您可以创建四个UPDATE OF触发：BEFORE/FOR EACH STATEMENT、BEFORE/FOR EACH ROW、AFTER/FOR EACH ROW以及AFTER/FOR EACH STATEMENT。定义触发时，UPDATE OF和UPDATE是不能同时使用的。表中所有UPDATE OF触发的字段是彼此互斥的，因此在替换触发器时不能使用其它UPDATE OF触发中使用的字段。

FOR EACH STATEMENT

图 3-31 For Each Statement Clause语法图

➲ **例1**

下例中的触发器原来是表**Employeesinfo**上的行触发器（**FOR EACH ROW**）**Trig_emp**，本例中将把它改为语句触发（**FOR EACH STATEMENT**）。

```
dmSQL> ALTER TRIGGER Trig_emp REPLACE WITH
                    BEFORE UPDATE ON Employeesinfo
                    FOR EACH STATEMENT
            (INSERT INTO NameChange VALUES (OLD.FName, OLD.LName,
                                            NEW.FName, NEW.LName));
```

➲ **例2**

下面，我们把例1中表**Employeesinfo**上的触发器**Trig_emp**的触发事件从**UPDATE**更改为**INSERT**。

```
dmSQL> ALTER TRIGGER Trig_emp REPLACE WITH
                    AFTER INSERT ON Employeesinfo
                    FOR EACH STATEMENT
            (INSERT INTO NameChange VALUES (OLD.FName, OLD.LName,
                                            NEW.FName, NEW.LName));
```

➲ **例3**

下面，我们把例2中表**Employees**上的触发器**Trig1**的触发动作从**INSERT**更改为**EXECUTE PROCEDURE**。

```
dmSQL> ALTER TRIGGER Trig1 REPLACE WITH
                    AFTER INSERT ON Employees
                    FOR EACH STATEMENT
            (EXECUTE PROCEDURE LogTime);
```

3.28 BEGIN BACKUP

使用BEGIN BACKUP命令可使数据库处于一种特殊的状态，在这种状态下，不需要其他用户断开数据库的连接或关闭数据库，系统将自动备份数据库中的所有文件。只有DBA、SYSDBA或SYSADM才可以执行这项命令。

存储介质错误是指计算机系统中二级存储器或辅助存储器的损坏。最常用的二级辅助存储设备一般指硬盘。存储介质错误通常是由磁盘的物理损坏所引起的，如读写头损坏、火灾、地震或超过磁盘物理极限的强震动、高压力等。

当发生介质错误时，数据库中的文件不可避免地会丢失。这时就需要数据文件或备份来恢复数据库。您应该周期性的对数据库文件进行备份，以备在遭到介质损坏的情况下恢复数据库，以下有几种数据库备份类型。

在线备份指数据库正在运行的情况下执行的备份。在这种情况下，数据库管理员不用关闭数据库，用户也不用断开与数据库的连接。在线备份不要求用户在执行数据备份时有任何动作，因此对用户是比较方便的。一个数据库管理系统必须具备在线备份数据库的功能。

离线备份指在关闭数据库后执行的备份。数据库管理员必须制定一个关闭数据库的时间表来断开数据库的连接。离线备份要求用户必须完成所有的现行事务且断开数据库，因此用户有可能觉得不太方便。一个数据库管理系统不一定必须具备离线备份数据库的功能。

完整备份是指在一个时间点上备份数据库系统中的所有文件，包括所有数据、日志文件等。完整备份是整个数据库的拷贝，因此会需要大量的存储空间，但是用它可以快速恢复数据库。

差异备份是基于数据的上次完整备份，即差异基础。差异备份仅备份至差异基础被创建时而发生改动的数据。差异基础通常用于一些成功的差异备份中。在执行恢复操作时，完整备份和相应的差异备份一起将被恢复为一个完整的数据库。

增量备份是指备份上次完整备份后，发生改动的日志文件。也就是说，增量备份中只包含上次完整备份后数据库的改动。由于增量备份只拷贝日志文件，所以它只需要较少的存储空间，但在恢复数据库时会花费较多的时间。

DBMaster提供五种类型的备份：离线完整备份（offline full backups）、在线完整备份（online full backups）、在线差异备份（online differential backups）在线增量备份（online incremental backups）以及在线增量备份至当前（online incremental to current backups）。在执行增量备份之前，您必须作一个完整备份（无论是在线还是离线）。当介质错误发生后，如果没有执行完整备份，而仅执行了增量备份，那么您就有可能无法恢复数据库。

在执行离线完整备份之前，您必须确保所有用户都断开了与数据库的连接，并且数据库已经关闭。如果数据库关闭时发生了错误，那么数据库管理系统将有可能无法完成备份操作，数据库也将无法恢复。您可以使用离线备份将数据库恢复到关闭数据库时的状态，数据库关闭后，就开始备份所有的数据、BLOB以及日志文件。

在执行在线完整备份之前，您必须先设置数据库的备份模式。数据库的备份模式分为三种：NON-BACKUP、BACKUP-DATA和BACKUP-DATA-AND-BLOB。您可以通过BEGIN BACKUP命令开始执行一个备份，当所有的数据和BLOB文件都备份完成后，通过END BACKUP DATAFILE命令结束备份。接着开始备份所有日志文件，备份完成后执行END BACKUP JOURNAL命令来结束备份，并将数据库返回到正常的操作状态。使用在线完整备份可以将数据库从上次完整备份时执行END BACKUP DATAFILE命令后的状态，恢复到拷贝当前日志文件后的状态。

在执行差异备份之前，您必须将数据库的备份模式设置为NON-BACKUP、BACKUP-DATA或BACKUP-DATA-AND-BLOB。在创建差异备份之前，您必须首先进行完整备份，即创建一个差异基础。DBMaster的差异备份仅备份数据文件（例如，*.DB和 *.BB），不备份日志文件。这是因为日志文件的改变非常频繁，所以当执行差异备份时，只备份有用的日志块。

在执行在线增量备份或在线增量备份至当前之前，您必须将数据库的备份模式设置为BACKUP-DATA或BACKUP-DATA-AND-BLOB。

只有具备数据库文件读取权限的用户才可以执行离线完整备份，并且只有具备DBA、SYSDBA或SYSADM权限的用户才可以执行在线备份。另外，同一时间只能有一个用户执行在线备份。

您可以随时使用ABORT BACKUP命令退出在线备份。执行该命令后，您将无法使用那些未完成的备份文件来恢复数据库。

不管数据库处于何种备份模式，即使是处于NON-BACKUP模式，您也可以执行在线完整备份和在线差异备份。但对于在线增量备份，只有在数据库处于BACKUP-DATA或BACKUP-DATA-AND-BLOB模式下才可以执行。

备份模式是指数据库管理系统在执行在线增量备份时，备份数据库中的哪些数据。DBMaster提供了三种方法来修改在线或离线备份的模式：您可以通过配置文件**dmconfig.ini**中的**DB_BMode**关键字来更改离线备份模式，在dmSQL中使用SQL SET命令设置在线备份模式，或通过DBMaster的服务器管理工具更改在线备份模式。

NON-BACKUP模式对上次完整备份后新插入或修改的数据不提供备份。在这种模式下，数据库不能执行在线增量备份。用户可利用日志文件从错误的程序中完全恢复数据库，但在介质损坏的情况下，丢失的数据可能就无法恢复了。在执行检查点后可立即回收没有被当前事务使用过的日志记录，一旦这些回收的日志文件被重写，数据库就只能恢复到了上次完整备份时的状态了。

如果您想通过更改关键字**DB_BMode**的方式来将数据库的备份模式更改为NON-BACKUP，那么您只需使用任何一种文本编辑器将**dmconfig.ini**配置文件打开，并将**DB_BMode**的值设为0即可。您也可以在在线完整备份中，通过SET BACKUP OFF命令将备份模式设置为NON-BACKUP。该命令必须在BEGIN BACKUP命令之后，END BACKUP JOURNAL之前执行，并且是在执行在线完整备份期间执行。

BACKUP-DATA模式可为上次完整备份后新插入或修改的数据（但不包括BLOB数据）提供恢复保障。在此模式下，DBMaster可以执行在线增量备份，但因为BLOB数据的更新并没有记录在日志文件中，所以只有非

BLOB数据才会被存储到备份文件中。数据库恢复后，上次完整备份后新插入或修改的BLOB数据将由空值（NULL）代替。您可以手动更新所有记录中的BLOB数据，用户可利用日志文件从错误程序中完全恢复数据库，也可以从介质损坏中部分恢复数据库。

如果您想通过更改关键字**DB_BMode**的方式来将数据库的备份模式更改为BACKUP-DATA，那么您只需使用任何一种文本编辑器将**dmconfig.ini**配置文件打开，将**DB_BMode**的值设为1即可。您也可以在执行在线完整备份的过程中，通过SET DATA BACKUP ON命令将备份模式设置为BACKUP-DATA，该命令必须在BEGIN BACKUP命令之后、END BACKUP JOURNAL之前使用，并且是在在线完全备份期间执行。

BACKUP-DATA-AND-BLOB模式对上次完整备份后新插入或更改的数据（包括BLOB数据）提供恢复保障。在这种模式下，DBMaster可以执行在线增量备份，所有数据都将存储到备份文件中。用户可以利用日志文件从程序错误和介质损坏中完全恢复数据库。您可以利用上次的备份将数据库（包括BLOB数据）完全恢复到介质错误发生点的状态。执行检查点后，没有在当前事务中使用的记录只有在做过备份后才能被回收。

如果您想通过更改关键字**DB_BMode**的方式来将数据库的备份模式更改为BACKUP-DATA-AND-BLOB，那么您只需使用任何一种文本编辑器将**dmconfig.ini**配置文件打开，将**DB_BMode**的值设置为2即可。您也可以在在线完整备份中，使用SET BLOB BACKUP ON命令将备份模式设为BACKUP-DATA-AND-BLOB。该命令必须在BEGIN BACKUP命令之后、END BACKUP JOURNAL之前使用，并且是在在线完整备份期间执行。

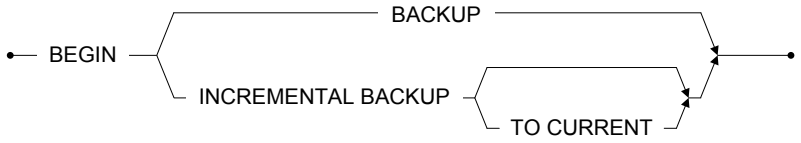


图 3-32 BEGIN BACKUP语法图

➲ **例**

下例展示了一个在线完整备份的全过程。首先发出命令**BEGIN BACKUP**通知DBMaster执行完整备份，然后使用操作系统命令将所有的数据和BLOB文件拷贝到指定位置，接下来发出命令**END BACKUP DATAFILE**，之后使用操作系统命令将所有的日志文件拷贝到指定位置，最后发出**END BACKUP JOURNAL**命令。如果备份成功，数据库将返回到平常的操作状态。

```
BEGIN BACKUP
    Copy data and BLOB files to backup location using OS commands
    Change backup mode if desired
    Abort the backup if desired
END BACKUP DATAFILE
    Copy Journal files to backup location using OS commands
    Change the backup mode if desired
    Abort the backup if desired
END BACKUP JOURNAL
```

3.29 BEGIN WORK

BEGIN WORK是一个可选命令，用于脚本文件中标明事务的开始。

DBMaster一般忽略该命令。

图 3-33 BEGIN WORK语法图

➲ 例

下例说明了**BEGIN WORK**命令是如何在脚本文件中标明事务开始的，其中的文本可以放在脚本文件的任何一个地方。

```
BEGIN WORK
    ...
    SQL Command
    SQL Command
    ...
COMMIT WORK
```

3.30 CHECK

CHECK命令用于检查指定的数据库对象，以确保数据的一致性。如果在查询的过程中，数据库返回不一致或错误的结果，或者接收到频繁或异常的错误信息，那么用户就需要检查数据的一致性。只有DBA、SYSDBA或SYSADM才可以执行该这项命令。

DBMaster提供CHECK命令来检查数据库、索引、表、文件、表空间以及系统目录的一致性。检查这些数据库对象的一致性是一个费时又耗资源的动作，因此只有在必要的时候才能使用CHECK命令，并且尽量避开数据库使用的高峰时刻执行该命令，这样就可以减少给用户带来的不便。

在检查数据库对象之前，DBMaster会首先检查系统表以确保所有目录信息都是正确有效的。如果在系统目录中发现了任何错误，DBMaster都将停止检查，因为这意味着数据库可能存在严重的不一致性。检查完系统命令表后，接下来检查的将是目标对象的物理结构和数据完整性。DBMaster在检查一个对象的同时，将检查所有与之相关的对象。之后DBMaster将陆续检查索引、数据页、文件以及表。

某些类型的错误是可以修补的，删除并重建索引可帮助您解决大部分出现在索引上的错误。对于出现问题的表，您可以先载出表中的所有记录，然后删除并重建该表，最后将前面载出的数据重新载入到新表中。

如果发现数据库的确存在不一致性的问题，那么您就应该立即备份这个数据库，包括所有数据和日志文件。DBMaster可在灾难恢复之后，可以修理部分不一致的错误。您可以按如下步骤来操作：首先启动DBMaster灾难恢复程序，然后关闭数据库再重启数据库；数据库重启后执行CHECK命令，看看错误是否已被纠正。

如果不一致性的问题仍然存在，那么请您联系CASEMaker客户服务，CASEMaker技术支持会帮助您修复数据库。

注意 *有关如何在您所在地区联系CASEMaker技术支持，请参看软件许可协议。*

*tablespace_name…*需要检查的表空间名。

file_name .....…需要检查的文件名。

table_name .....…需要检查的表名。

index_name .....…需要检查的索引名。

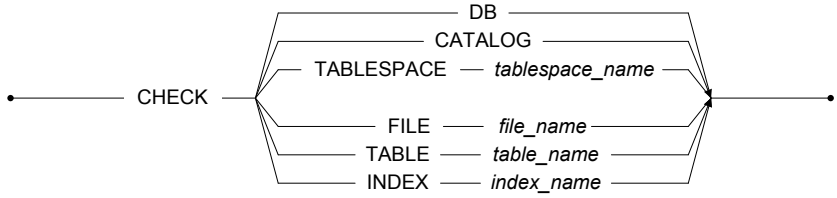


图 3-34 CHECK语法图

➲ **例1**

下例将检查表**Customers**中的数据一致性。

```
dmSQL> CHECK TABLE Customers;
```

➲ **例2**

下例将检查表**Customers**中索引**idxCustNum**的一致性。

```
dmSQL> CHECK INDEX Customers.idxCustNum;
```

➲ **例3**

下例将检查文件**customer_data**中的数据页或BLOB帧的一致性。

```
dmSQL> CHECK FILE customer_data;
```

➲ **例4**

下例将检查指定表空间**ts_new**中数据库对象的一致性。这些对象可能包括文件、表、数据页以及表中的数据。

```
dmSQL> CHECK TABLESPACE ts_new;
```

➲ **例5**

下面的命令将检查数据库系统目录的一致性。

```
dmSQL> CHECK CATALOG;
```

➲ **例6**

下面的命令将检查所有数据库对象的一致性。

```
dmSQL> CHECK DB;
```

3.31 CHECKPOINT

使用CHECKPOINT命令可使DBMaster强制执行检查点（checkpoint）。在数据库活动频繁，且在很少做备份或重启数据库的情况下，应该执行检查点。只有DBA、SYSDBA或SYSADM才有权执行该命令。

发生检查点之后，数据库将处于一个干净状态。这时候，DBMaster会将缓冲区中的所有日志记录以及所有脏数据页写入磁盘，并且回收不再被备份或恢复所使用的日志块。DBMaster可以回收那些记录着一些非活动的事务的日志块，而这些事务在在最早的活动事务开始前就已结束。

执行检查点会节省数据库在发生系统故障后的启动时间。DBMaster会将最后一次的检查点时间以及当时的事务情形写到日志文件的文件头处。在恢复数据库时，DBMaster将应用这些信息来决定哪些事务应该不做，哪些应该重做，哪些事务应该取消。

DBMaster在执行在线备份，数据库启动、终止或日志文件空间不足时自动执行检查点。执行检查点会花费一段时间，这段时间的长短取决于从上次执行检查点到目前的事务量。自动执行检查点时，所有正在运行的事务都必须等待，直到该操作完成。如果日志文件已满，并且检查点后回收的空间不足以完成当前事务时，DBMaster会取消该事务。这种情况下就应该重做被取消事务中的所有命令。

数据库管理员应定期执行CHECKPOINT命令来手动执行检查点，这样可以避免事务处理过程中的不必要中断。定期手动执行检查点将可以节省启动、终止和备份数据库的时间，缩短了事务的等待时间，同时也减少了日志文件空间不足的可能性。数据库管理员可依据数据库活动的频率来决定手动执行检查点的最佳时间。

图 3-35 CHECKPOINT语法图

➲ 例

下面的命令可强制系统执行检查点。

```
CHECKPOINT
```

3.32 CLOSE DATABASE LINK

CLOSE DATABASE LINK命令可用来断开与远程数据库的连接。使用该命令可断开一条连接或同时断开多条连接。任何连接到远程数据库的用户都可以执行该命令。

数据链创建了与远程数据库的连接，使用户可以通过本地的数据库来访问远程的数据库，数据库链接还提供了额外的安全性信息。数据库链接可以使用户以另外一个用户名连接到远程数据库中，或使用公用数据库链接来连接远程数据库，使用公用数据库链接并不要求您在连接时提供帐号。

执行CLOSE DATABASE LINK命令并指定数据库链名称后，如果连接中没有正在处理的事务，那么DBMaster就会关闭这条和远程数据库的链接。执行CLOSE DATABASE LINK命令并指定了远程数据库名后，DBMaster将关闭所有与远程数据库的链接。如果被关闭的链接正在处理事务，那么该链接仍将继续。同时DBMaster将返回错误信息，系统将等到事务完成后重新关闭链接。

关键字NONACTIVE用于关闭所有没被当前事务使用的链接。使用该关键字后，如果您在执行CLOSE DATABASE LINK命令时恰好有个链接正被使用，那么该链接将继续保持。DBMaster必须等到该连接上的事务完成后才能再次关闭它。

关键字ALL keyword用于关闭所有和远程数据库的链接。使用该关键字后，如果您在执行CLOSE DATABASE LINK时命令时恰好有个链接正被使用，那么该链接将继续保持，同时DBMaster将返回错误信息。您必须等到该链接上的事务全部完成后才能关闭它。

link_name.....连接到远程数据库的数据库链名。

remote_database_name… 远程数据库名。

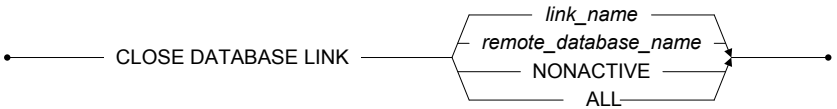


图 3-36 CLOSE DATABASE LINK语法图

➲ **例1**

下面的命令将关闭数据库链**FieldLink**。

```
dmSQL> CLOSE DATABASE LINK FieldLink;
```

➲ **例2**

下面的命令将关闭远程数据库**FieldOffice**上的所有链接。这个远程数据库已在本地**dmconfig.ini**中定义。

```
dmSQL> CLOSE DATABASE LINK FieldOffice;
```

➲ **例3**

下面的命令将关闭所有没被事务使用的连接。

```
dmSQL> CLOSE DATABASE LINK NONACTIVE;
```

➲ **例4**

下面的命令将关闭所有没被事务使用的连接，如果有一个连接正被使用，那么DBMaster将返回错误信息，并保持该连接。

```
dmSQL> CLOSE DATABASE LINK ALL;
```

3.33 COMMIT WORK

COMMIT WORK命令用于提交当前事务。执行该命令后，DBMaster将自动开始一个新事务。任何具备CONNECT权限或更高权限的用户都可以执行该命令。

事务是单一的逻辑工作单元，或者是为保持数据库一致性而需要一起完成的一条或多条操作。事务必须是独立的，执行事务后只可能有两种状态：事务正常完成，更改数据库中数据；或事务失败，数据库中数据不发生任何变化。

例如，假设您要在数据库中存储两种不同的信息：已发送到客户的货物记录、现存货物（包括其数量）的记录。当一款货物被送往客户后，发送清单里应增加已发送的这款货物及其数量。同时必须从存货清单中减去这款货物的已发送量。如果以上两项操作没有被作为单一的逻辑工作单元一起完成，那么数据库将处于不一致状态。如果已发送了货物，但数量没从存货单里减去，那么存货单中的货物数量将高于实际数量；如果从存货单里减去没有发送的货物，那么存货单中的货物数量将低于实际数量。以上两项操作必须作为一个单一事务，要么都完成，要么都失败。

如果一个事务成功完成并且更改了数据库中的数据，那么这个事务就已经被提交。如果事务失败而数据库中的数据保持不变，这时候事务就被回滚。

执行COMMIT WORK命令后，DBMaster会将事务对数据的更改写入数据库中。注意，COMMIT WORK命令只是把当前事务所做的更改写入数据库中。如果数据库运行在AUTOCOMMIT模式下，那么这条命令就不是必须的。

AUTOCOMMIT模式用来控制DBMaster提交事务的时间。如果开启了AUTOCOMMIT模式，那么每条命令将会作为一个单独的事务。事务完成后按Enter键可自动提交事务；如果操作中出现错误，系统将自动回滚事务。如果关闭AUTOCOMMIT模式，那么两条COMMIT WORK之间的所有命令将被视为一个事务。您可以执行COMMIT WORK来提交事务对数据所作的更改或执行ROLLBACK WORK命令来回滚数据的更改。

数据库出现故障时，DBMaster将自动回滚没有提交的事务。被回滚的事务所作的更改如果映射到了数据库，那么在数据库重新启动后将重做这些事务中的命令。

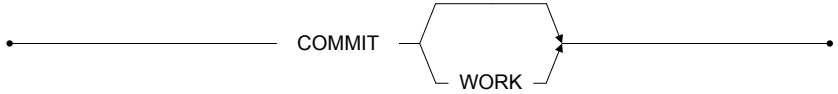


图 3-37 COMMIT WORK语法图

➲ **例**

下例是在**AUTOCOMMIT**关闭的情况下，提交介于前后两个**COMMIT WORK**命令之间，所有SQL命令所作的更改操作。

```
COMMIT WORK
    ...
    SQL Command
    SQL Command
    ...
COMMIT WORK
```

3.34 CREATE COMMAND

CREATE COMMAND命令用来创建一个新存储命令。使用存储命令可使您更快、更方便地执行常用的SQL数据操纵指令。只有具备RESOURCE或更高安全权限的用户，以及具备执行SQL命令所需的所有安全和对象权限的用户才可以执行该命令。

存储命令是是编译过的SQL数据操纵指令，它们以可执行的文件格式永久存储在数据库中。用户可以反复执行存储命令，而不用等待DBMaster编译和优化这条指令。存储命令和存储过程非常相似，不同之处在于，前者只包含一条指令，但不能包含程序逻辑。

您在生成存储命令的时候，需要指定存储命令名和有效的SQL数据操作命令，如SELECT、INSERT、UPDATA或DELETE。SQL命令中可以使用主变量来代表字段。这使得用户在执行命令时可为字段赋一个实际的值。如果您想在存储命令中使用主变量，那么就应该将命令中的数据或字段值用问号（？）来代替。

执行一个包含主变量的指令时，其中主变量的值可以是内置函数的返回值、NULL关键字、DEFAULT关键字或其它主变量值。值得注意的是，只有使用诸如RAND( )、CURDATE( )及NOW( )这些不含有自变量的内置函数时，才能为主变量赋值；如果要为主变量赋空值，必须先确保主变量代表的值可以接受空值。执行中输入的参数数量必须等于定义存储命令时的主变量个数。

如果您在删除存储命令参照的表或字段，或者更改表、修改字段定义或使用关键字BEFORE和AFTER来增加新字段，那么存储命令将变得无效，并且将不能再被使用。如果在新增字段时没有使用BEFORE和AFTER关键字，那么这种表修改将不会对存储命令产生影响。当一个存储命令不在有意义后，我们可以将它从数据库中删除。

每个存储命令名在这个数据库中都必须唯一。存储命令的名字不能超过128个字符，它可以是数字、字母、下划线以及$和#符号，但不能以数字开头。

OR REPLACE：若存储命令已存在，可指定OR REPLACE选项重建该存储命令，即用户可使用该子句更改已存在存储命令的定义。

command_name……新建的存储命令名。

select_statement…..合法的SELECT语句

insert_statement……合法的INSERT语句。

update_statement….合法的UPDATE语句。

delete_statement……合法的DELETEA语句。

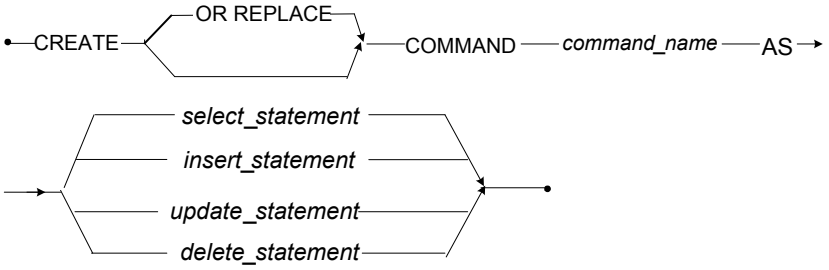


图 3-38 CREATE COMMAND语法图

➲ **例1**

下面的命令将生成一个名为**sc_select**的存储命令，其功能是在表**Employeesinfo**中查询所有姓是以**A**字母开头的员工信息。

```
dmSQL> CREATE COMMAND sc_select AS SELECT * FROM Employeesinfo WHERE LastName
LIKE 'A%';
```

➲ **例2**

下面的命令将生成一个名为**sc_update**的存储命令，其功能是将主变量的值赋予表**Employeesinfo**的**Manager**字段。

```
dmSQL> CREATE COMMAND sc_update AS UPDATE Employeesinfo SET Manager = ? WHERE
Manager = ?;
```

或：

```
dmSQL> CREATE COMMAND OR REPLACE sc_update AS UPDATE Employeesinfo SET Manager
= ? WHERE Manager = ?;
```

3.35 CREATE DATABASE LINK

CREATE DATABASE LINK命令可以创建一个公用或私用的数据库链来连接远程数据库。数据库链可以使用户以存取本地数据库对象的方式来存取远程数据库对象。只有DBA、SYSDBA或SYSADM才可以通过该命令来创建公用数据库链，而只有具备CONNECT权限或更高权限的用户才可以通过该命令来创建私用数据库链。

数据链创建了与远程数据库的连接，使用户可以通过本地的数据库来访问远程的数据库，尽管您可以直接连接到远程数据库上，但是使用数据库链来连接数据库是更有益的因为它还提供了额外的安全性信息。数据库链可以使用户以不同的用户名连接到远程数据库中，或使用公用数据库链接来连接远程数据库，使用公用数据库链接并不要求您在连接时提供帐号。

创建数据库链时应输入链名和远程数据库名。本地和远程数据库的**dmconfig.ini**文件都必须包含对方数据库的配置信息。数据库配置信息必须包含对方数据库服务器的IP地址和端口号。您可以在配置文件中使用关键字**DB_SvAdr**来设定IP地址，使用关键字**DB_PtNum**来设定端口号。

关键字PUBLIC/PRIVATE是可选的。这两个关键字用来设置数据库链的类型是公用还是私用。公用链可供数据库中的所有用户使用，而只有创建者才能使用自己所建的私用链。只有DBA、SYSDBA或SYSADM才能创建公用链，而数据库中的任何合法用户都可以创建私用链。如果公用数据库链和用户的私用数据库链的名字相同，那么用户将不能使用公用数据库链。创建数据库链时，如果没有指定是公用（public）还是私用（private），DBMaster将默认为私用数据库链。

关键字IDENTIFIED BY是可选的。这个关键字用来指定连接远程数据库时的用户名和密码。这里指定的用户名必须是远程数据库中存在的、具备CONNECT权限或更高权限的用户。用户使用数据库链连接数据库时，可以执行的操作应该根据用户在数据库中被赋予的安全和对象权限而定。如果连接远程数据库时没有指定用户名，那么DBMaster将使用本地数据库中的当前用户名来连接。

数据库链名的最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

link_name ..... 数据库链名。

remote_db_name…要连接的远程数据库名。

user_name.....在远程数据库中，具备CONNECT或更高权限的用户名。

password ..... 远程数据库中的用户密码。

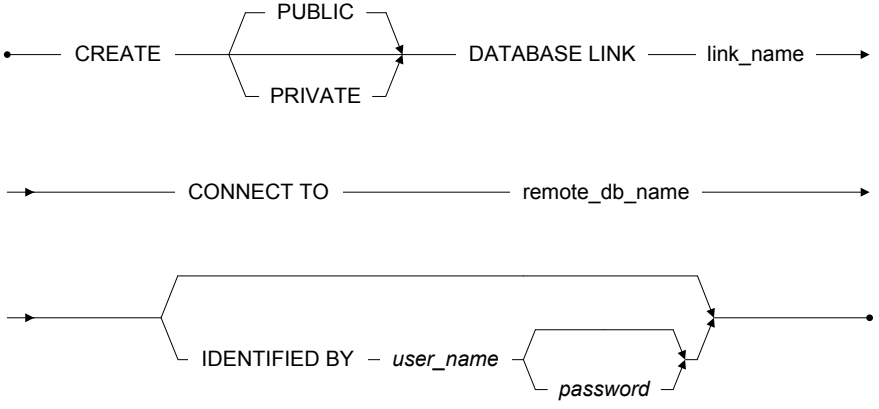


图 3-39 CREATE DATABASE LINK语法图

➲ **例1**

下例创建了一个名为**FieldLink**的公用数据库链来连接远程数据库**FieldOffice**。创建这个连接的用户必须是在本地数据库中具备DBA、SYSDBA或SYSADM安全权限的用户，并且在本地和远程数据库中使用的是同一个用户名。用户使用这个链，以远程数据库中链创建者的帐户连接到远程数据库。用户的权限是由远程数据库中同名帐户的安全及对象权限决定的。

```
dmSQL> CREATE PUBLIC Database LINK FieldLink CONNECT TO FieldOffice;
```

➲ **例2**

下例创建了一个名为**FieldLink**的数据库链来连接远程数据库**FieldOffice**。创建这个链的用户必须具备DBA、SYSDBA或SYSADM权限。用户通过这个链以用户名**LinkUser**和密码**dil3ryx9**来连接远程数据库。用户的权限是由远程数据库中该用户的安全及对象权限决定的。

```
dmSQL> CREATE PUBLIC Database LINK FieldLink CONNECT TO FieldOffice
                          IDENTIFIED BY LinkUser dil3ryx9;
```

➲ **例3**

下例创建了一个名为**FieldLink**的私用数据库链来连接远程数据库**FieldOffice**。这个创建数据库链的用户必须在本地和远程数据库中具有相同的用户名。用户通过这个链以本地数据库中的同名用户连接到远程数据库。用户的权限和远程数据库中同名用户的安全性及对象权限相同。如果系统中还存在与该私用数据库链同名的公用数据库链，那么系统将使用私用数据库链。

```
dmSQL> CREATE PRIVATE Database LINK FieldLink CONNECT TO FieldOffice;
```

➲ **例4**

下例创建了一个名为**FieldLink**的私用数据库链来连接远程数据库**FieldOffice**。用户通过这个链以用户名**Vivian**和密码**a23456**来连接远程数据库。用户的权限和该用户名的安全及对象权限相同。当您在本地和远程数据库中有不同的用户名时，这种连接方式是很有用的。如果系统中还存在与这个私用数据库链同名的公用数据库链，那么系统将使用私用数据库链。

```
dmSQL> CREATE PRIVATE Database LINK FieldLink CONNECT TO FieldOffice
                          IDENTIFIED BY Vivian a23456;
```

3.36 CREATE DOMAIN

CREATE DOMAIN命令用来新建一个可选择默认值和可选择完整性约束的定义域。任何具备RESOURCE或更高安全权限的用户，都可以使用该命令。

定义域是一种用户自定义的数据类型，可用来设定数据类型、默认值、字段约束。在CREATE TABLE或ALTER TABLE ADD COLUMN命令中定义字段时，就可以使用定义域代替标准数据类型来为字段定义一组有效的输入值。

例如，创建一个基于DATA数据类型的定义域，该定义域的默认值为NOW( )，其有效的输入值是1900年1月1日至今。用这个定义域创建的字段都将继承这些特性，检查这些字段的完整性约束，而无需您每次在创建同样类型的字段时都制定默认值和约束值。

在创建定义域时必须要指定数据类型，不过默认值和约束条件是可选的。数据类型必须是DBMaster支持的除SERIAL之外的任何数据类型。您可以使用DEFAULT关键字来设定默认值，用关键字CHECK来设定条件约束。

在CREATE DOMAIN子句中，可以使用TEXT CONVERTER语法来创建定义域。当在创建定义域的语句中指定了TEXT CONVERTER语法，DBMaster可以通过TEXT CONVERTER函数来将CLOB、NCLOB、BLOB和FILE类型转换为纯文本类型，以便创建全文索引和PURETEXT() UDF。TEXT CONVERTER函数名称需要包含一个与BLOB类型相关的参数，返回的类型必须是CLOB或NCLOB数据类型，否则将返回错误。最多可以使用TEXT CONVERTER语法创建**32767**个定义域。

关键字DEFAULT是可选的。插入新记录时，如果某个字段上没有插入值，那么系统会自动将这些字段上的值设为DEFAULT指定的默认值。常量、内置函数的值或关键字NULL都可用来作为默认值。需要注意的是：在创建定义域时，只有诸如PI( )、NOW( )或者USER( )这样没有自变量的内置函数才可以当作默认值。如果默认值设定为NULL，那么在定义字段时就不能再使用关键字NOT NULL了。

关键字CHECK是可选的。这个关键字可用来为输入的值指定约束条件。CHECK后面的表达式一般都是取值为真（true）假（false）的任何一种表达式。可以将关键字VALUE和关键字CHECK一同使用，用来代表字段的值。如果一条SQL命令不满足CHECK所设定的条件，它就不能被执行。如果用户用定义域而不是标准DBMaster数据类型来定义字段，关键字CHECK最好不要再使用，因为定义域中一般都包括了CHECK语言定义的条件约束。

使用定义域来设定默认值和条件约束就等于使用标准的字段定义方法来设定默认值和条件约束。不过，如果您先在定义域中设定了默认值，然后又在定义字段时设定了默认值，那么这个默认值将覆盖定义域中预先设定的默认值。而如果您在定义字段时又添加了约束条件，那么这个字段的条件约束就会变成旧的定义加上新增的条件约束。

因此，您必须保证字段定义时的条件约束和定义域中的条件约束并不冲突。当使用定义域定义字段时，DBMaster并不会检查新旧条件约束是否冲突，但冲突的条件约束可能让您无法插入或更新某些甚至全部数据。

定义域名的最大长度不能超过128个字符，它可以包含数字、字母、下划线以及$和#字符，但不能以数字开头。

注意 *只有不带自变量的函数才可以在定义域中使用。*

domain_name .....要创建的定义域名。

data_type .....定义域指定的数据类型。

constant.....在没有给字段输入值时，系统将使用这个参数代替常数值。

function_name.....在没有给字段输入值时，系统将使用这个参数代替内置函数值。

constraint_name.....应用于字段的约束名。

boolean_expression.....取值为真假的表达式。

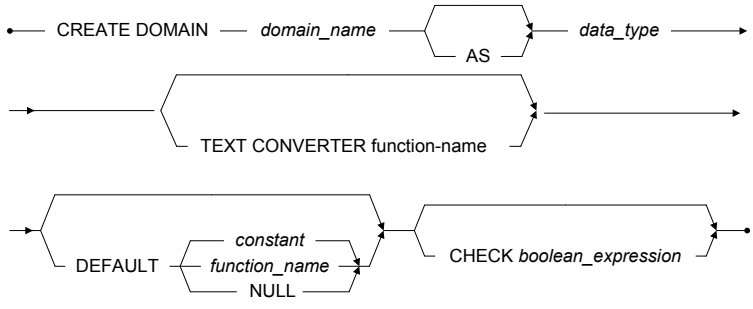


图 3-40 CREATE DOMAIN语法图

➲ **例1**

下例创建了一个以**INTEGER**类型为基础的定义域**AllNum**。

```
dmSQL> CREATE DOMAIN AllNum AS INTEGER;
```

➲ **例2**

下例创建了一个以**INTEGER**类型为基础的定义域**AllNum**。其默认值为***0***。

```
dmSQL> CREATE DOMAIN AllNum AS INTEGER DEFAULT 0;
```

➲ **例3**

下例创建了一个以**INTEGER**类型为基础的定义域**AllNum**。条件约束不为空。

```
dmSQL> CREATE DOMAIN AllNum AS INTEGER CHECK VALUE IS NOT NULL;
```

➲ **例4**

下例创建了一个以**INTEGER**类型为基础的定义域**PosNum**。默认值是***0***，取值范围是0到100。

```
dmSQL> CREATE DOMAIN PosNum AS INTEGER DEFAULT 0 CHECK VALUE >= 0 AND VALUE <=
100;
```

➲ **例5**

下例创建了一个以**DATE**类型为基础的定义域**ValidDate**。使用**NOW( )**函数作为默认值，取值是01/01/1900到NOW( )之间的时间值。

```
dmSQL> CREATE DOMAIN ValidDate AS DATE
                        DEFAULT NOW()
                        CHECK VALUE > '01/01/1900' AND VALUE <= NOW();
```

3.37 CREATE GROUP

CREATE GROUP命令用来创建新的用户组。组中的每个用户都具有赋给该组的所有对象权限，只有SYSADM、SYSDBA或DBA才能执行这条命令。

当数据库中的用户数量很多时，组可以简化对象权限的管理。您可以将要求同样对象权限的所有用户集合成一个组。任何赋予组的对象权限都将自动赋予该组中的所有成员。创建一个新组后，您可以使用ADD TO GROUP命令来将用户添加到组中。

DBMaster支持嵌套组，即将一个组作为成员加入到另一个组中，不过值得注意的是，将一个组添加到另一个组时不能构成循环。如果group2是group1中的一个成员，那么您就不能将group1作为一个成员添加到group2中，并且也不能将group1作为成员添加到group1中。向一个组中添加组成员与添加用户成员的方法是一样的。

组的名字不能是SYSTEM、PUBLIC或GROUP，并且不能和已有的用户和组重名。组名最大长度不能超过128个字符，可以包含字母、数字、下划线以及$和#符号，但不能以数字开头。

group_name..... 要创建的新组名。

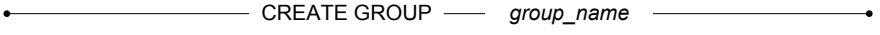


图 3-41 CREATE GROUP语法图

➲ **例**

下例创建了一个名为**Manager**的组。

```
dmSQL> CREATE GROUP Manager;
```

3.38 CREATE HASH INDEX

哈希索引只能创建在内存表上。使用哈希索引可以使用户快速的存取存储在哈希索引中的数据。哈希索引还可以提高比较运算和连接运算的执行效率。您可以使用CREATE HASH INDEX *index_name* ON *table_name* (*column_name*, …) [*bucket n*]命令创建一个哈希索引，其中的*index_name*代表创建哈希索引的名称，*table_name*为内存表的名称，*column_name*为内存表中相关联的字段名，该字段不能指定排序顺序：asc/desc。bucket n用于设置创建哈希表的数组大小。

index_name .....要创建的新的哈希索引名。

table_name .....创建的哈希索引所在的内存表名。

column_name ....哈希索引中使用的字段名。

bucket n .....设置哈希索引中的数组值。

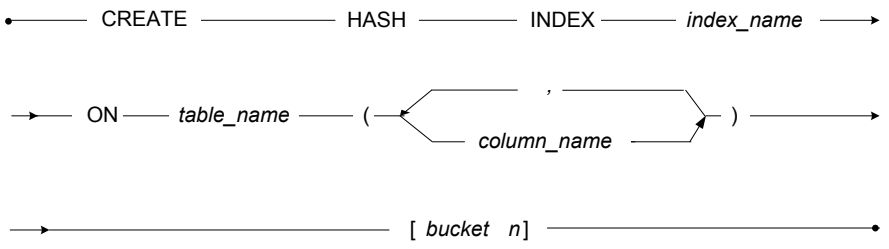


图 3-42 CREATE HASH INDEX语法图

➲ **例**

下例在内存表**tb_mem**上创建了哈希索引**idx1**，其中使用了字段**c01_int**和**c02_char**，数组值是31。

```
dmSQL> CREATE HASH INDEX idx1 ON tb_mem (c01_int, c02_char) BUCKET 31;
```

3.39 CREATE INDEX

CREATE INDEX命令用于在现有表上新建一个索引。因为索引可以不搜索整张表而快速定位到指定记录，所以使用索引可加快查询的速度。只有表的拥有者，DBA或具备INDEX权限的用户才能执行该指令。

索引是一种在表上通过一个或多个字段（例如键）或字段的表达式的值来快速查找指定记录的机制。索引包含的数据和键字段中的数据一样，但是这些数据通过组织和排序使得检索速度得到了提高。一旦在表上创建了索引，它的操作对数据库的用户而言就是透明的了，数据库管理系统就可以使用索引来提高查询速度。

创建索引时需要指定索引名、创建索引的表名以及表的键字段名。您可以在一个或多个字段上创建索引，但是字段数最多不能超过32个。表中的任何字段都可以创建索引。DBMaster同时还限制了索引的最大记录不能超过4000个字节。

对经常频繁使用的表达式创建索引，可大大提高查询性能。对于XML字段，在XML UDF：**extract()**和**extractvalue()**上创建索引也可提高xpath的查询速度。请注意EXtract()和Extractvalue()之间的主要区别是：**EXtract()**允许多值、单值或零值，而不支持升降序或唯一索引。**Extractvalue()**仅允许UDF的结果为单值或零值，如果UDF的结果为多值，那么创建索引或插入新记录操作将失败；同时，**Extractvalue()**支持升降序或唯一索引。

过滤索引（条件索引）是一种带有WHERE条件子句的优化索引。当用户从一个定义明确的数据子集中查找符合条件的索引值时，过滤索引不失为一种最佳选择。也就是说，过滤索引在查找索引值之前就已经插入到数据页中，且过滤索引的数据不包括表的所有数据行，而是通过过滤条件（WHERE子句）选取数据行的一部分。过滤索引使用过滤谓词选择表中的部分数据行，同一个全表索引相比，一个设计巧妙的过滤索引不仅能够提高数据库的查询性能，还能降低表索引的维护和储存成本。

WHERE子句允许以下谓词组合，包括：

- 表的任一字段

- 常数值
- 比较符。=，>，>=，<，<=，!=。例：c1>=3
- Like。例：c3 like ‘abc’
- NULL和NOT NULL。例：c4 is null
- in list。例：c5 in (1,3,5)
- 运算符。+，-，*，/ 。例：c1+c2>5
- 用户自定义函数（UDF）。例：abs(c6)>5
- BLOB运算符。match, contain
- AND组合。例：c1=3 and c2=5 and c3=7
- OR组合。例：c1=3 or c2=5 or c3=7

WHERE子句不允许出现以下语句：

- 子查询
- 主变量
- 混合AND和OR，例：c1=3 or c2=5 and c3=7

请参考下面的XPath规则将有助于创建索引：

- 不能包含谓词
- 不能包含函数
- 必须有一个绝对路径
- 仅支持’child’axis
- 应该有一个只包含叶子节点的节点集（一般类型的元素节点或属性节点）
- 针对所有元素节点qname必须相同
- 针对所有属性节点，每个属性明必须相同
- 必须基于属性节点或元素叶子节点

- 不能是复杂的非叶子节点或评论节点，例如：'/order/items/item/@product' 或 '/order/date'
- 不允许位置 '/order/items/item[1]/@product'
- 允许函数 'count(/order/items/item)'
- 不允许表达式

关键字UNIQUE是可选的，这个关键字可用来指定索引是否具备唯一性。在唯一性索引中，不允许多笔记录具有相同的键值，也不允许包含重复的数据。索引中的每个空值都被看作是一个唯一的值，所以在唯一性索引中就有可能存在多条键值为空的记录。在一个非空表上创建唯一性索引时，DBMaster将检查所有的键值是否唯一。如果存在重复值，DBMaster会返回错误信息，并且不能创建这个索引。而在表上创建了唯一性索引后，DBMaster会检查新增或更新的记录键值是否和已有的键值重复，如果重复，操作将无法执行。默认情况下，DBMaster创建的是非唯一性索引。所以，如果您想要创建一个唯一性索引，就必须使用关键字UNIQUE 。

关键字AUTO是可选的，这个关键字用来指定自动索引后台程序是否自动执行索引。其行为类似于非唯一性索引，但它可通过自动索引后台程序自动创建或删除。若设置AUTOCOMMIT为ON，则DBMaster仅需*Update(U)*锁便可创建自动索引，这意味着DBMaster允许其它用户同时对表进行查询。若要创建自动索引，用户需指定关键字AUTO。

关键字ASC/DESC是可选的，这两个关键字用来指定索引列的排序是递增还是递减。您还可以逐个字段的指定排序，这就可能导致某些字段递增，而某些字段递减。在某些情况下，索引字段的排序可能会影响查询结果输出的顺序。如果索引是递减的，那么就算您没有指定查询的排序，查询的结果也有可能是降序排列的。您可以使用ORDER BY子句来指定字段的排序顺序，DBMaster默认的字段排序顺序是递增（ASC）。

关键字FILLFACTOR也是可选的，这个关键字用来指定索引页可填充的比例。填充系数为更新现有记录预留了空间，这就优化了数据库对索引页的使用。该参数的取值范围为1～100，也就是说，填充系数是从1%到100%。如果索引创建后对表的更新很频繁，您就可以将填充系数设定得

低一些（如50%），这样才可以为新键值留下足够的空间；如果您不需要经常更新表，那么您就可将填充系数保留为默认值，即100% 。

您向表中载入数据时，每载入一条记录，DBMaster就会对这个表上的索引更新一次。因此，您最好在载入所有数据后再创建索引，这样比在载入数据前就创建索引更有效率。

每张表的索引名必须唯一。索引名最长不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

索引可以和创建它们的表存储在不同的表空间。

index_name .....…新建的索引名。

table_name .....…索引所在的表。

column_name .....…索引使用的字段。

number.....…填充系数值。

tablespace_name...存储索引的表空间名。

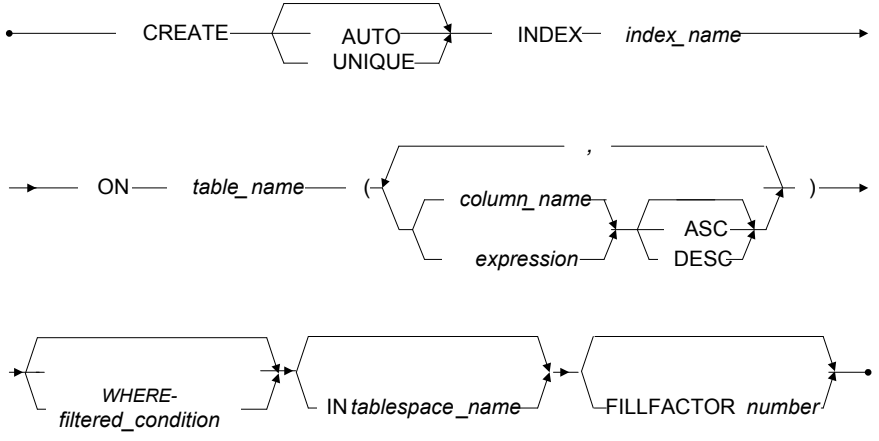


图 3-43 CREATE INDEX语法图

➲ **例1**

下例在表**Employeesinfo**的字段**FName**和**LName**上创建一个名为**NameIndex**的索引，该索引是非唯一性索引，可以包含重复值。

```
dmSQL> CREATE INDEX NameIndex ON Employeesinfo (FName, LName);
```

➲ **例2**

下例在表**Employeesinfo**的**FName**和**LName**字段上创建一个名为**NameIndex**的索引，这两个字段都是降序排列。

```
dmSQL> CREATE INDEX NameIndex ON Employeesinfo (FName DESC, LName DESC);
```

➲ **例3**

下例在表**Classes**的**Course**和**Section**字段上创建一个名为**ClassIndex**的索引，该索引是唯一性索引，不允许包含重复值。

```
dmSQL> CREATE UNIQUE INDEX ClassIndex ON Classes (Course, Section);
```

➲ **例4**

下例在表**Classes**的**Course**和**Section**字段上创建一个名为**ClassIndex**的索引，这个索引是唯一性索引，不能包含重复值，索引页的填充系数为**80%**。

```
dmSQL> CREATE UNIQUE INDEX ClassIndex ON Classes (Course, Section) FILLFACTOR 80;
```

➲ **例5**

下例在表**Classes**的**Course**和**Section**字段的表达式上创建一个名为**ExprIndex**的索引，这个索引是唯一性索引，不能包含重复值，索引页的填充系数为**80%**。

```
dmSQL> CREATE UNIQUE INDEX ExprIndex ON Classes (concat(Course,Section))
FILLFACTOR 80;
```

➲ **例6**

下例在表**tb_staff**的**ID**和**NAME**字段上创建一个名为**AUTO_ID_2**的索引（降序排列）。

```
dmSQL> CREATE AUTO INDEX AUTO_ID_2 ON tb_staff (ID DESC, NAME);
```

➲ **例7**

下例在表**tb_salary**的表达式**basepay+bonus** 上创建一个名为**AUTO_1DX_expr** 的自动索引（降序排列）。

```
dmSQL> CREATE AUTO INDEX AUTO_IDX_expr ON tb_salary (basepay+bonus DESC);
```

➲ **例8**

下例在表**tb_salary**上通过**where**条件子句创建一个名为**FILIDX_income**的过滤索引。

```
dmSQL> CREATE INDEX FILIDX_income ON tb_salary(basepay+bonus,tax)WHERE ID>30;
```

3.40 CREATE PROCEDURE

CREATE PROCEDURE命令用于生成新的存储过程。使用存储过程可使数据库引擎避免重复编译并优化SQL命令，从而提高频繁重复任务的性能。拥有RESOURCE安全权限或更高安全权限的用户以及拥有执行SQL语句所必需的安全权限和对象权限的用户可以使用CREATE PROCEDURE命令。

存储过程是已编译的SQL数据操作语句，以可执行文件的格式永久存储在数据库中。它可以作为一个命令在交互式SQL中执行，也可以被应用程序、触发行为或其他存储过程调用。

创建存储过程时，用户需指定存储过程名和有效的SELECT, INSERT, UPDATE, DELETE等SQL数据操作语句。在SQL语句中，主变量用于代替字段值，实际值将在后来执行命令时分配给字段。使用（？）替换数据或字段值，以便在存储命令中使用主变量。

FROM FILE

OR REPLACE：若存储过程已存在，可指定OR REPLACE选项重建该存储过程，即用户可使用该子句更改已存在存储过程的定义。

file_name ..... ……用于创建存储过程的文件名。

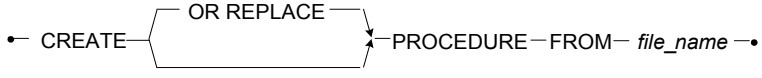


图 3-44 CREATE PROCEDURE FROM FILE语法图

➲ **例**

创建或替换存储过程：

```
dmSQL> CREATE PROCEDURE FROM 'file-name';
dmSQL> CREATE OR REPLACE PROCEDURE FROM 'file-name';
```

ESQL SP

OR REPLACE：若存储过程已存在，可指定OR REPLACE选项重建该存储过程，即用户可使用该子句更改已存在存储过程的定义。

module_name ....……要创建存储过程的模块名。

procedure_name ....……要创建的存储过程名。

procedure_parameter……要创建的存储过程参数。

procedure_returen_results...要创建的存储过程返回的结果集。

注意 *不支持在存储过程中执行 CREATE OR REPLACE COMMAND 和 CREATE OR REPLACE PROCEDURE 命令；当AUTOCOMMIT设置为OFF时，也不支持 CREATE OR REPLACE PROCEDURE命令。*

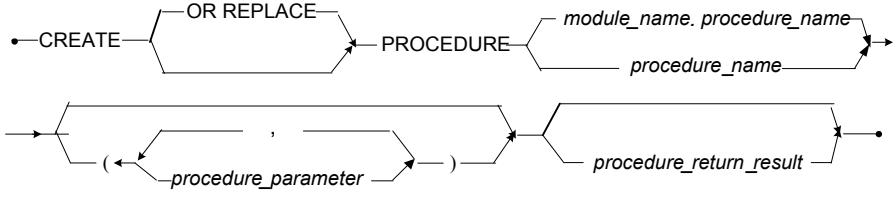


图 3-45 CREATE PROCEDURE语法图

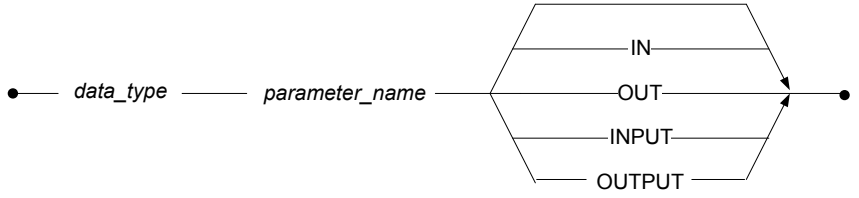


图 3-46 CREATE PROCEDURE: procedure_parameter语法图

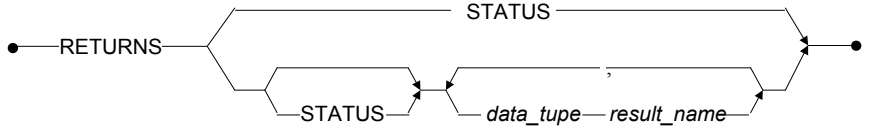


图 3-47 CREATE PROCEDURE: procedure_return_result语法图

➲ **例： 创建或替换存储过程**

-从文件创建ESQL存储过程：

```
dmSQL> CREATE OR REPLACE PROCEDURE FROM 'proc1.ec';
```

-编辑**.ec**文件：

```
EXEC SQL CREATE OR REPLACE PROCEDURE proc1 (char(10) i1, char(10) i2 output)
 returns char(10) o1, char(10) o2;
{
                EXEC SQL BEGIN CODE SECTION;
                EXEC SQL select FName from tb_staff where LName =:i1 into:i2;
                EXEC SQL returns select * from tb_staff into :o1,:o2;
                EXEC SQL END CODE SECTION;
}
```

JAVA SP

OR REPLACE：若存储过程已存在，可指定OR REPLACE选项重建该存储过程，即用户可使用该子句更改已存在存储过程的定义。

module_name.... ……要创建存储过程的模块名。

procedure_name ....……要创建的存储过程名。

procedure_parameter.……要创建的存储过程参数。

data_type..…….... 返回变量数据类型。

variable_name...……返回变量名。

package.class.method.....要创建存储过程的java方法。

argtype..…..... ……java方法参数类型。

java_sourcecode_jar_file.....jar源代码的物理jar文件。

related_jar_file... ……..... 逻辑jar文件。

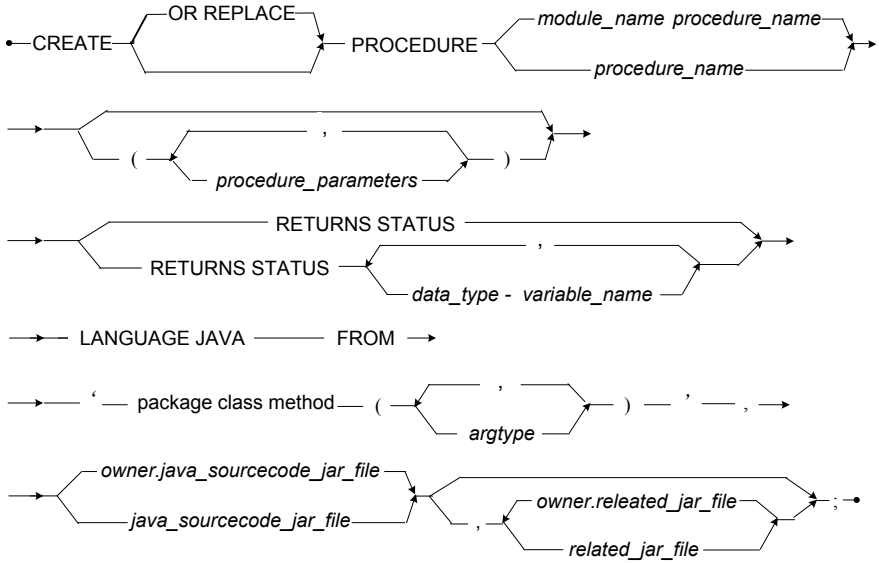


图 3-48 CREATE JavaSP语法图

➲ **例： 创建或替换存储过程**

-编辑java文件*AddStaff.java*：

```
package staff;
import java.sql.*;

public class AddStaff
{
                    // Add an row into the tb staff table
                     public static void addStaff(String fName, String lName)
                     throws Exception
                     {
                                    // Register DBMaster JDBC Driver
                                    Class.forName("dbmaster.sql.JdbcOdbcDriver");
                                    // Connect to database
                                     Connection conn =
DriverManager.getConnection("jdbc:default:connection");

                                     // Prepare SQL statement
```

```
                                    PreparedStatement pstmt =
conn.prepareStatement("insert into tb staff values(?,?)");

                                    // Set values of the dynamic SQL argument
                                    pstmt.setString(1, fName);
                                    pstmt.setString(2, lName);

                                    // Execute the dynamic SQL statement
                                    pstmt.execute();

                                   // Close the dynamic SQL statement
                                   pstmt.close();

                                   // Close the connection
                                      conn.close();
            }
}
```

-在DOS命令行下编译文件**AddStaff.java**，将在当前目录下创建一个名为**AddStaff.class**的文件：

```
javac AddStaff.java
```

-复制**AddStaff.class**到*dir\staff*。

-打包类，将在当前目录下创建文件**addStaff.jar**：

```
jar cvf addStaff.jar   staff\AddStaff.class
```

-在**<DB_SpDir>**下创建目录*jar\SYSADM*，将文件**addStaff.jar**移动到***<DB_SpDir>****jar\SYSADM*。

-添加**jarfile**：

```
dmSQL> ADD JARFILE addStaff addStaff.jar;
```

-创建或替换JAVA存储过程：

```
dmSQL> CREATE OR REPLACE PROCEDURE addStaff(char(12) fname,char(12) lname)
RETURNS STATUS LANGUAGE JAVA FROM
'staff.AddStaff.addStaff(String,String)',addStaff;
```

SQL SP

OR REPLACE: 若存储过程已存在，可指定OR REPLACE选项重建该存储过程，即用户可使用该子句更改已存在存储过程的定义。

module_name ....……要创建存储过程的模块名。

procedure_name ....……要创建的存储过程名。

procedure_parameter……要创建的存储过程参数。

date_type ..... 返回变量的数据类型。

variable_name ..... 返回变量的名字。

sp_declare_main.....……要创建的存储过程的主声明变量部分。

sp_statement_main.....……要创建的存储过程的主语句部分。

注意 *不支持在存储过程中执行CREATE OR REPLACE COMMAND和 CREATE OR REPLACE PROCEDURE命令；当AUTOCOMMIT设置为OFF时，也不支持CREATE OR REPLACE PROCEDURE命令。*

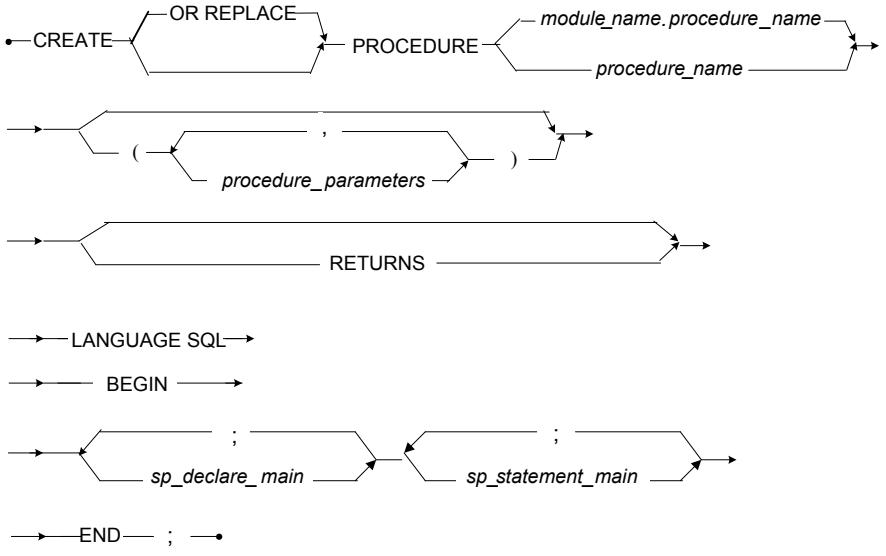


图 3-49 CREATE SQL SP语法图

➲ **例： 创建或替换存储过程**

-从文件创建SQL存储过程：

```
dmSQL> CREATE OR REPLACE PROCEDURE FROM 'proc1.sp';
```

-编辑**.sp**文件：

```
dmSQL> CREATE OR REPLACE PROCEDURE proc1
LANGUAGE SQL
BEGIN
     DECLARE cur CURSOR WITH RETURN FOR select * from tb_staff;
     OPEN cur;
END;
```

3.41 CREATE REPLICATION

CREATE REPLICATION命令用于生成一个新的表复制。复制、同义字、视图是不能创建在临时表上的。只有表的拥有者、DBA、SYSDBA或SYSADM才能执行该命令。

表复制可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，用户可以不去干预。

表复制主要有两种类型：同步（*synchronous*）表复制和异步（*asynchronous*）表复制。同步表复制指当我们对本地表做任何更改时，远程表就会立刻被更新；异步表复制是按照既定的时间计划存储本地表中的更改，然后再将这些更新复制到远程表中。使用CREATE REPLICATION命令可以创建同步和异步表复制。

DBMaster中的同步表复制遵循全局事务规则，即本地事务和将数据复制到远程表是不可分的两个步骤，这意味着如果向远程数据库中复制数据失败，那么在本地表上执行的事务也不会成功。

DBMaster中，异步表复制利用事务日志来将数据复制到远程表中。DBMaster将本地表的更改都存储于事务日志中，然后根据既定的时间计划复制到远程表中。通过使用事务日志，DBMaster可独立处理远程和本地事务，即使在不能连接远程表的情况下，仍可执行对本地表的更改。异步表复制允许网络异常或远程数据库出错的情况下而不影响本地操作，因为在错误排除前，DBMaster会不断尝试向远程复制数据。

在创建表复制机制时，需要指定复制名、要复制的本地表名以及远程目的表名。无论是本地表还是远程表，它们都必须是已存在于各自数据库中的表。

如果没有指明本地表中要复制的字段，那么DBMaster将复制整张表，这时，本地表和远程表必须具有相同的字段名和数据类型。本地表中的字

段将按从左到右的顺序复制到远程表字段列表中的相应字段。您也可以同时指明远程表和本地表字段，让它们一一对应。但在任何一种情况下，两张表的主键以及组成主键的数据类型应该相同。

DBMaster并不是用复制名来区分复制的，而是将对象名、拥有者名及表名结合起来作为复制的完整名称。因此同一张表上的所有复制名必须唯一。DBMaster是按表复制创建者的安全性及对象权限进行同步表复制的。但如果远程数据库明确规定了以数据库链的方式来存取的话，那么DBMaster将按照数据库链的安全性和对象权限来进行同步表复制。DBMaster按照远程用户的安全性及对象权限进行异步表复制，这些远程用户是在使用CREATE SCHEDULE命令时，由子句IDENTIFIED BY指定的。

关键字ASYNC是可选的，该关键字表示创建的复制是异步表复制。创建异步表复制之前，应该先为包含远程表的远程数据库创建一个复制计划。如果没有使用这个关键字，那么DBMaster将默认生成同步表复制。

关键字WHERE是可选的。向远程表中复制数据时，可以使用它引导的子句来指明搜索条件。只有符合搜索条件的记录才会被DBMaster复制到远程表中，更多有关WHERE子句的信息可参看SELECT命令。

CLEAR DATA、FLUSH DATA、CLEAR AND FLUSH DATA这些关键字是可选的，它们是在定义表复制时规定具体操作的关键字。如果选择CLEAR DATA，则在创建表复制时，系统会将远程表中的数据先行删除；如果选择关键字FLUSH DATA，则在创建表复制时，系统会将本地表中满足条件的数据先行复制到远程表中；CLEAR AND FLUSH DATA关键字是指在创建表复制的同时，先清除远程表中的数据，再将满足条件的数据插入到远程表中。

关键字NO CASCADE也是可选的，该关键字只有在异步表复制时才可以使用，其作用是进行级联复制。下面我们将举例说明什么是级联复制。级联复制就像多数组织一样，命令由最高层逐层向下传递到最基层。如数据从节点A复制到节点B，然后再复制到节点C，这是一种典型的级联复制。而在No-Cascade模式中，数据从节点A复制到节点B，同时，B也将其数据复制到A。如果您想以这种方式来复制数据，就应该开启NO CASCADE选项。如果没做任何规定，系统则会默认CASCADE是开启状态。

如果您更改表和字段定义或使用关键字BEFORE和AFTER来新增字段，而这些表或字段是被一个同步表复制所参照，那么该表复制将变得无效且不能再继续使用。如果在没有使用关键字BEFORE和AFTER时新增字段，那么这个操作就不会对同步表复制产生任何影响。更改表对异步表复制不产生影响。当一个复制无效时，您可以将它从数据库中删除。删除表或表中的某个字段时，如果您选择了CASCADE关键字，系统将自动删除创建在这张表上的复制。

复制名的最大长度不能超过128个字符，可以包括数字、字母、下划线以及$和#字符，但不能以数字开头。

replication_name…….要创建的表复制名。

local_table_name……. 本地表名。

column_name……1. 本地表的字段名。

.....……..2. 远程表的字段名。

search_condition……..复制数据必须满足的条件。

remote_table_name….远程数据库中的表名。

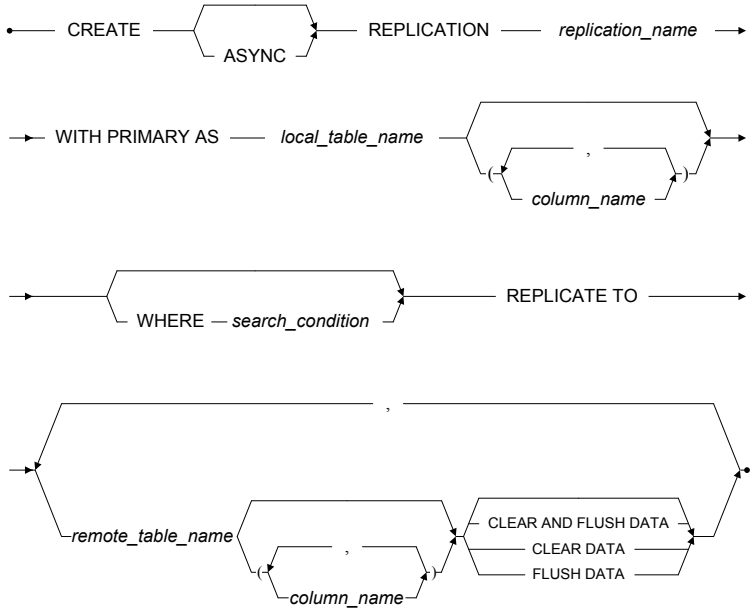


图 3-50 CREATE REPLICATION语法图

➲ **例1**

下例为本地表**Employeesinfo**创建一个名为**EmpRep**的表复制。本地数据库中的 **dmconfig.ini**配置文件列出了远程数据库**FieldOffice**的配置信息。远程表也叫**Employees**，这两张表的字段名和数据类型都相同。

```
dmSQL> CREATE REPLICATION EmpRep WITH PRIMARY AS Employeesinfo
                          REPLICATE TO FieldOffice:Employeesinfo;
```

➲ **例2**

下例同上。不同的是本例中指定了复制时的具体操作：先清除远程表中的数据，再将本地表中的所有数据复制到远程表中。

```
dmSQL> CREATE REPLICATION EmpRep WITH PRIMARY AS Employeesinfo
                          REPLICATE TO FieldOffice:Employeesinfo
                          CLEAR AND FLUSH DATA;
```

3.42 CREATE SCHEDULE

CREATE SCHEDULE命令可用来为异步表复制生成一个复制时间计划。同步表复制中并不需要使用复制计划，所以该命令对同步表复制不产生任何影响。只有DBA、SYSDBA或者SYSADM才能执行该命令。

表复制可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，用户可以不去干预。

关键字NO CASCADE也是可选的。这个关键字只有在复制类型是异步表复制时才可以使用，其作用是进行级联复制。下面我们将举例说明什么是级联复制。级联复制就像多数组织一样，命令由最高层逐层向下传到最基层。例如数据从节点A复制到节点B，然后再复制到节点C，这是一种典型的级联复制。而在No-Cascade模式中，数据从节点A复制到节点B，同时B也将其数据复制到A。如果您想以这种方式复制数据，就应该开启NO CASCADE选项。如果没做任何规定，系统将默认CASCADE为开启状态。

异步表复制不仅可以在两个DBMaster数据库之间进行，而且可以将DBMaster中的数据库复制到Oracle、SYBASE、INFORMIX和Microsoft SQL Server数据库中，这种类型的复制被称作异类表复制。异类表复制使得DBMaster可在异构环境下与其它数据库共存。因为DBMaster在将数据复制到第三方远程数据库之前需要预处理这些数据，所以在异构环境中创建复制计划时就需要指明第三方数据库管理系统的类型。您可以使用ORACLE、SYBASE、INFORMIX和MICROSOFT关键字

来指代第三方数据库，其中ORACLE表示远程数据库是Oracle数据库，SYBASE表示远程数据库是SYBASE数据库，INFORMIX 表示远程数据库是INFORMIX数据库，而MICROSOFT则表示远程数据库是Microsoft SQL Server数据库。

创建异类表复制时不能使用CLEAR DATA、FLUSH DATA或CLEAR AND FLUSH DATA关键字。复制开始前您应该在第三方远程数据库中手动删除或插入数据以使表处于初始状态。此外，第三方数据库上不能执行模式检查（schema checking），模式检查可确保本地表和远程表中字段和数据类型的兼容性。在创建异步表复制计划时，应使用关键字WITH NO CHECK来阻止DBMaster执行模式检查（有关关键字WITH NO CHECK的描述，请参看本节随后内容）。DBMaster使用ODBC驱动程序管理器（ODBC Driver Manager）来执行异类表复制，同时DBMaster服务器必须运行在Windows 平台上，第三方远程数据库可运行在任何Windows和UNIX平台上。

关键字BEGIN AT指明第一次做异步表复制的日期和时间。日期格式必须是：yyyy/mm/dd， 其中yyyy表示年份，其范围是1970到2038年；mm表示月份，范围是01到12；dd表示日，范围是01到31。时间的格式必须是hh:mm:ss，其中hh表示小时，其范围为00到23；mm表示分钟，其范围为00到59；ss表示秒，范围为00到59。可以看出，要使用BEGIN AT关键字就必须包含日期和时间两组值。如果您在复制运行的情况下将第一次复制的时间改为将来的某个时间，那么还没有复制到远程数据库中的数据将会保留在您的数据库中，直到您所设定时间到来时才开始复制。

关键字EVERY规定了异步表复制中相继发生的两次复制的时间间隔。该时间间隔可以是时/分/秒，也可以是天数或前两者的结合。您可以使用EVERY hh:mm:ss格式来设定时/分/秒的值，其中hh是00到23之间的整数，mm是00到59之间的数，ss是00到59之间的数值。使用EVERY d DAYS格式来设定间隔的天数，其中d是1到365之间的整数。使用EVERY d DAYS AND hh:mm:ss格式来设定以上二者结合的时间值。

在执行SQL命令时可能会出现一些错误，比如锁超时错误或根据全日志将事务回滚到保存点错误，关键字RETRY指当这些错误发生后，DBMaster重试表复制的次数。您可以使用RETRY n TIMES格式来设定重试的次数，其中n代表重试的次数，它的范围为0到2147483647，默认值是***0***。在没有使用RETRY关键字的情况下，如果碰到网络错误或远程数据库出错，或其它需要回滚事务的错误时， DBMaster将会等到下一个复制计划到来时重新复制这次没复制成功的数据。

关键字AFTER是可选的，它与关键字RETRY一起使用，用来指明出现错误的情况下，两次重试之间的时间间隔。使用Use AFTER *s* SECONDS格式来指明这一间隔的秒数，SECONDS的取值范围是0到2147483647，其默认值是***5***。

当远程数据库中的数据发生更改而使表复制不能进行时，可用关键字ON ERROR来指明DBMaster应采取的动作。这些使复制不能正常进行的情况有：DBMaster试图在远程表中删除一条并不存在的记录或向远程表中插入已存在的记录。当这些错误发生时，DBMaster为用户提供两个解决方案：STOP ON ERROR和IGNORE ON ERROR。STOP ON ERROR表示当这种错误发生时，DBMaster将终止复制；IGNORE ON ERROR表示DBMaster可以忽略错误，继续进行数据的复制，系统默认的设置是忽略错误（IGNORE ON ERROR）。

关键字WITH NO CHECK是可选的。因为DBMaster不能检查第三方数据库的结构，所以在创建异类表复制时应该使用该关键字。在使用这个关键字后，用户就应当承担起模式检查的责任以确保本地表和远程表中的字段和数据类型是兼容的。创建同类表复制（从一个DBMaster数据库复制到另一个DBMaster数据库）时无需此关键字。

使用关键字IDENTIFIED BY可以指定连接远程数据库时使用的用户名和密码。这里指定的用户必须是已存在于远程数据库中且具备表的INSERT、DELETE、UPDATE等权限的用户。将表中的数据复制到远程数据库时，用户在远程表上可执行的操作由赋予该用户的安全性和对象权限决定。

remote_database_name…远程数据库中的名称，并且不能是数据库链。

yyyy/mm/dd.....开始复制的日期。

hh:mm:ss .....1. 开始复制的时间。

.....2. 两次复制的时间间隔。

d .....两次复制间隔的天数。

n .....错误发生时，复制重试的次数。

s .....错误发生时，两次重试之间的秒数。

user_name..... 远程数据库中帐户的用户名。

password ..... 远程数据库中帐户的密码。

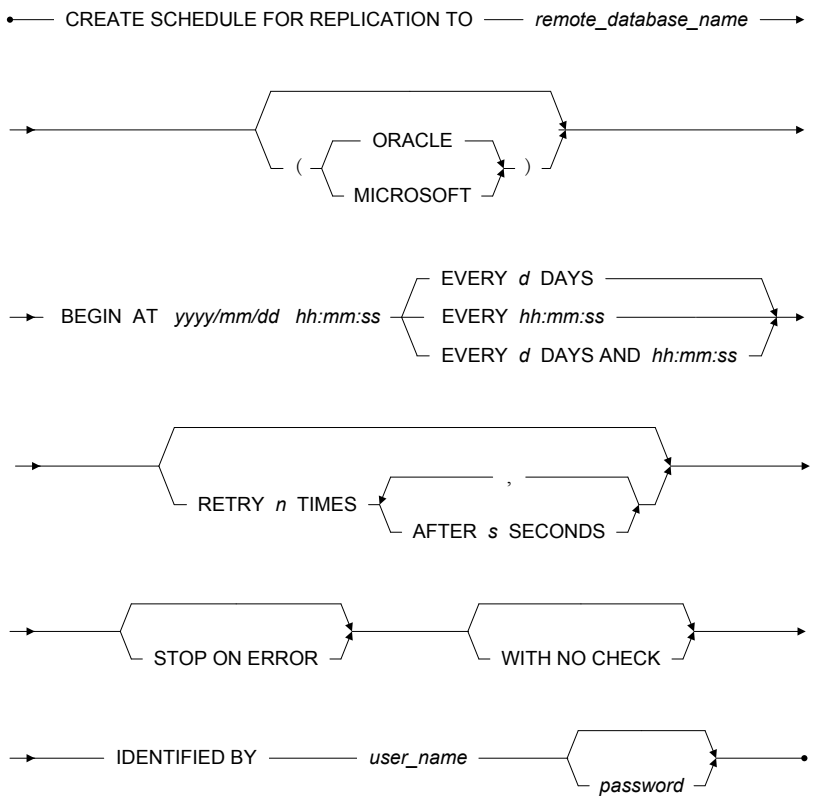


图 3-51 CREATE SCHEDULE语法图

➲ **例1**

下例为异步表复制**EmpRep**创建了一个复制计划，首次复制的日期和时间设定在将来的某个时间上，两次复制的时间间隔为**7天零12小时**。表中的所有数据都将等到新时间到来后才能被复制。

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep
```

```
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00;
```

➲ **例2**

下例同上，只是本例中将发生错误、锁超时或根据全日志事务回滚到保存点后重试复制的次数设为**3次**，两次重试之间的时间间隔为**5秒**。

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS;
```

➲ **例3**

下例同上，并且还设置了当远程数据库中的数据发生更新使得复制不能进行时，DBMaster采取的动作是停止（**STOP**）复制。

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR;
```

➲ **例4**

下例同上，并且本例中还设置了连接远程数据库的用户名和密码分别是：**RepUser**和**rdejpe88**。

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR
                IDENTIFIED BY RepUser rdejpe88;
```

➲ **例5**

本例是一个异类表复制，使用关键字**WITH NO CHECK**来阻止DBMaster对远程数据库执行模式的检查。用户要确保远程数据库中的字段和数据类型和本地数据库中的兼容。本例中创建的复制计划同例**4**，同时还使用了关键字**ORACLE**来表明远程表处于*Oracle 8.0*数据库中。

```
dmSQL> CREATE SCHEDULE FOR REPLICATION TO EmpRep (ORACLE)
                BEGIN AT 2001/10/10 00:00:00 EVERY 7 DAYS AND 12:00:00
                RETRY 3 TIMES AFTER 5 SECONDS
                STOP ON ERROR
                WITH NO CHECK
                IDENTIFIED BY RepUser rdejpe88;
```

3.43 CREATE SCHEMA

CREATE SCHEMA命令可以创建一个新模式，并将该模式加入当前数据库系统中。模式实质上是数据库对象的统称，它包括的数据库对象有：表、视图、索引、同义字、触发器、定义域、存储指令和存储过程。这些对象的名称可以和存在于其它模式中的对象重名。访问数据库对象时，需要验证它们的名字。验证数据库对象的名字应该以模式名作为前缀表示它们的名称。

只有具备RESOURCE或更高权限的用户才可以创建模式。如果在创建模式时省略了*user_name*，那么该模式的创建者将成为系统的默认用户。只有具备DBA权限的用户才可以为其他用户创建模式。

如果用户拥有DBMaster的connect权限，那么DBMaster会为该用户创建一个默认模式。模式名即为用户的名称，并且该模式名必须唯一。如果要创建的模式名和数据库中已有的模式同名，那么系统将返回一条错误信息。

对模式的所有者存在以下约定：

- 如果使用了AUTHORIZATION子句来指定用户名，那么给出的用户名就代表模式的所有者名；如果省略了模式名，那么子句中指定的用户名将成为新建的模式名
- 如果没有使用AUTHORIZATION子句，那么创建CREATE SCHEMA语句的用户就成为模式的所有者。

schema_name ... .要创建模式的名称。

user_name ..... .新建模式所有者的名称。

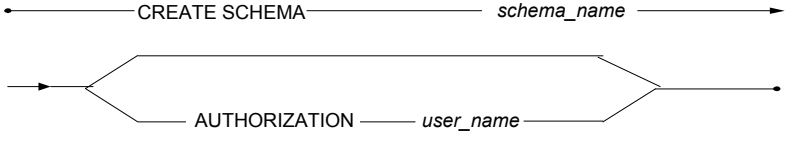


图 3-52 CREATE SCHEMA语法图

➲ **例1**

具备RESOURCE权限的用户**YUBIN**，新建一个名为**schm_def**的模式。那么**YUBIN**将成为该模式的默认所有者。

```
dmSQL> CREATE SCHEMA schm_def;
```

➲ **例2**

具备DBA权限的用户新建了一个以用户**YUBIN**作为所有者的模式，由于创建模式时没有指定模式名，因此**YUBIN**就成为默认的模式名。

```
dmSQL> CREATE SCHEMA AUTHORIZATION YUBIN;
```

注意 *记住用户何时被授予连接权限是很重要的。DBMaster会自动为该用户生成一个与用户名同名的模式。如果数据库中已存在一个同名模式，那么系统将返回一条错误信息。*

➲ **例3**

在本例中，一个具备DBA权限的用户新建了一个以**YUBIN**为所有者的模式**schm_auth**。

```
dmSQL> CREATE SCHEMA schm_auth AUTHORIZATION YUBIN;
```

➲ **例4**

在本例中，具备DBA权限的用户新建了一个名为**inventory**的模式，随后又为该模式创建了两个模式对象：**inventory-part**和**partind**。最后为表**inventory-part**中的用户YUBIN赋予所有对象权限。注意，用户YUBIN并非拥有模式**inventory**的所有权限。

```
dmSQL> CREATE SCHEMA inventory;
dmSQL> CREATE TABLE inventory.part (partNo smallint not null, quantity int);
dmSQL> CREATE INDEX partind ON inventory.part (partNo);
dmSQL> GRANT ALL ON inventory.part TO YUBIN;
```

3.44 CREATE SYNONYM

CREATE SYNONYM命令用来为现存表或视图新建一个同义字。您不能在临时表或其它同义字上创建同义字。只有表或视图的拥有者、DBA、SYSDBA或SYSADM才有权执行该命令。

DBMaster通常把表或视图名和它的拥有者名、对象名结合起来充当整个表或视图的完整名称。为了简化表和视图的名称，DBMaster允许使用同义字。

同义字就是表或视图的别名。它不占存储空间，系统只需将它的定义存储在系统目录中。用户可以使用同义字来存取相应的表或视图，而不必记忆或键入完整的表或视图名称。

为表或视图创建的多个同义字名必须唯一，这样用户可以直接引用同义字，而不用在其前面加上所有者名。如果一个用户的某个表和同义字同名，那么在DBMaster中，这个名称代表的将是表而不是同义字，这时如果要使用那个同义字所指向的表就只能键入完整的表名称。删除表或视图并且选择CASCADE关键时，DBMaster将删除为它们创建的同义字。

同义字名最长不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

OR REPLACE：若同义字已存在，可指定OR REPLACE选项重建该同义字，即用户可使用该子句更改已存在同义字的定义。

synonym_name . .同义字的名称。

table_name . . . . . .创建同义字的表名称。

view_name . . . . . .创建同义字的视图名称。

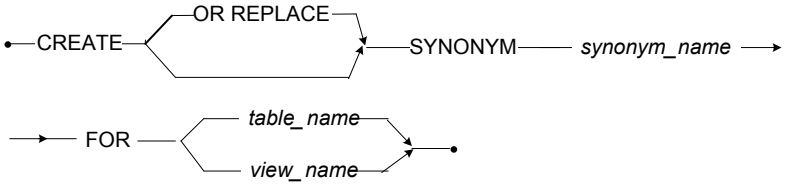


图 3-53 CREATE SYNONYM语法图

➲ **例1**

下例为用户**User1**的表**AllEmployees**创建一个名为**AllEmp**的同义字。在之后的SQL命令中就可以使用该同义字**AllEmp**来代替表的完整名称**User1**.**AllEmployees**。

```
dmSQL> CREATE SYNONYM AllEmp FOR User1.AllEmployees;
```

➲ **例2**

下例为用户**User2**的视图**SalesEmployees**创建一个名为**SalesEmp**的同义字。在之后的SQL命令中就可以使用该同义字**SalesEmp**来代替视图的完整名称**User2.SalesEmployees**。

```
dmSQL> CREATE SYNONYM SalesEmp FOR User2.SalesEmployees;
```

3.45 CREATE TABLE

CREATE TABLE命令可用来生成一张新表。生成新表时，您应该指明表所属的表空间。默认情况下，DBMaster会在系统表空间里创建表。任何具备RESOURCE或更高安全权限的用户都可以执行该指令。

在关系数据库中，表是存储数据的基本单元，您输入到数据库的任何信息都将存储到表里。每张表代表现实世界某一类型的对象，并包含这类对象的信息。这些对象可能是真实对象，比如客户或产品等，也有可能是抽象对象，比如订单或交易等。数据库中每张表的名称都必须唯一，这个名称通常指出了存储在表中的对象类型，表将相关对象的信息存储在表的行和列中。

行又称记录或元组，其中存储的信息定义了某类实体具有同样的特性，每一行表示这类实体的一个实例。此外，系统以实体的一个或多个特性来区分每一行。行并不是按照某个特殊的顺序排列的，您也不能保证行的任何两次排列一定是同样的顺序。

列又称字段或属性，其中存储的信息定义了实体的特性。每一列代表一个属性或实体实例数据中的某一项。系统用列名和数据类型来区分每一列，因此每个列名必须唯一。表中的列即使被重新排列，也不会影响SQL查询。

约束或规则是用来确保数据完整性的。创建表时，您可以在列上使用定义域、字段完整性约束或表完整性约束来保证数据库中数据的完整性。

定义域约束是在创建定义域时定义的，它作用于所有使用定义域定义的字段。在插入新记录或更新现有记录时，系统会计算每个定义域的约束条件。定义域约束包括NULL/NOT NULL限制、默认值以及CHECK约束。

字段完整性约束是定义在某一个列上的，并不影响表中的其它列。插入新记录或更新现有记录时，系统会计算每个字段的约束条件。字段约束包括NULL/NOT NULL限制、默认值以及CHECK约束。

表完整性约束是定义在一组列上的。插入新记录或更新现有记录时，如果所有定义域约束和字段约束条件的计算结果都为真，系统将会计算每

个表的约束条件。只有表约束条件的计算结果也为真时，才能执行插入或更新操作。表约束包括唯一性（UNIQUE）约束、CHECK约束、主键和外键。

创建表至少要指明表名和字段定义。一张表至少得拥有一个字段，最多可包含2,000个字段，不过请注意，表可支持的最大字段数还受到数据页大小的影响。

DBMaster通常将表名和所有者名结合起来作为表的完整名称。表名称的最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。表名称在数据库中必须唯一。只有具备DBA权限的用户才能以特定的表所有者名来创建表。表所有者必须是已存在于数据库中的用户，默认的表所有者是表的创建者。要注意的是表名称区分大小写。

定义字段时，至少要指明字段名、数据类型或定义域。下面将给出定义字段时关键字的语法和用法。

table_name .....……要创建的表名称。

column_definition……字段的定义。

primary_key_defintion…主键的定义。

foreign_key_definition…外键的定义。

constraint_name ……..应用于表上的约束条件。

tablespace_name……表所属的表空间名。

boolean_expression……取值为真或假的表达式。

number.....……填充系数的值。

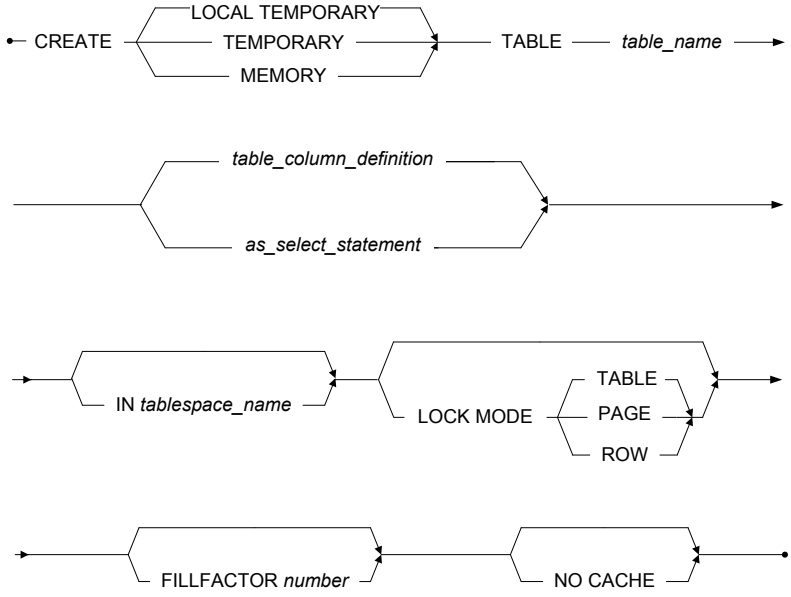


图 3-54 CREATE TABLE语法图

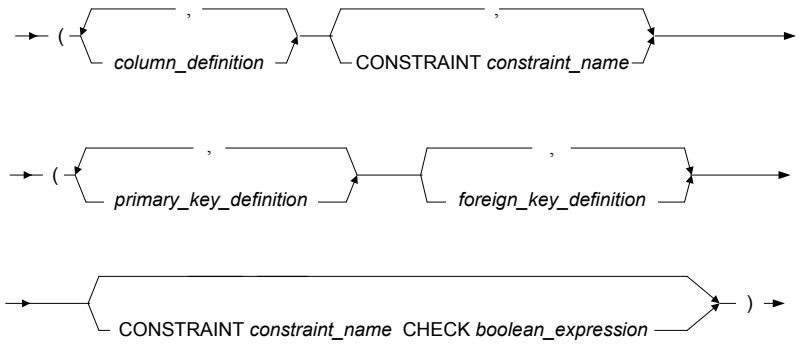


图 3-55 CREATE TABLE: table_column_definition语法图

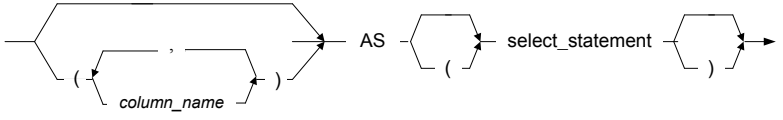


图 3-56 CREATE TABLE: as_select_statement语法图

字段定义

DBMaster通常用所有者名、表名和字段名来作为表中字段的完整名称。字段名最长不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。字段名在同一张表中必须是唯一的，字段名区分大小写。

DBMaster支持如下一些数据类型：BINARY、BIGINT、BIGSERIAL、CHAR、DATE、DECIMAL、DOUBLE、FLOAT、FILE、INTEGER、BLOB、CLOB、OID、SERIAL、SMALLLINT、TIME、TIMESTAMP以及VARCHAR。

您可以不使用标准的数据类型，而使用用户自定义的定义域来定义字段。定义域可用来设定字段的数据类型、默认值以及字段的完整性约束。使用关键字DEFAULT和CHECK来定义字段的默认值和完整性约束（下面会提到这两个关键字的具体用法）。如果您在定义字段时重新设定了字段的默认值，那么字段的默认值将会是新设定的值；如果您在定义字段时定义了完整性约束，那么这个字段的条件约束将是定义域中的约束加上新定义的约束。

关键字NULL/NOT NULL是可选的，这两个关键字表示新插入的字段值是否可以为空。NULL关键字表示在新插入字段时，可以不输入数值；NOT NULL关键字表示新插入字段时必须输入值，NOT NULL只能在表中没有数据时才能使用，如果在表中有数据的情况下使用该关键字，它就可能会与本来为空的字段发生冲突，系统的默认值是***NULL*** 。

关键字USER/SYSTEM是可选的，这两个关键字决定了用户是否可以使用INSERT/UPDATE语句来更改字段的默认值。若用户没有设置该关键字，则默认为USER。当使用USER关键字时，用户可以更改字段的默认值；而使用SYSTEM关键时，用户不能更改字段的默认值。

关键字DEFAULT也是可选的。当插入一条新记录时，如果没有输入字段的值，就可以使用DEFAULT关键字所指定的值作为字段的默认值。常数、内置函数的值以及空值（NULL）都可以作为字段的默认值。只有不含自变量的内置函数，如PI( )、NOW( )或者USER( )才能作为默认值。如果用NULL来作为默认值的话，定义字段时就不能再使用关键字NOT NULL了。

关键字ON UPDATE也是可选的。该关键字用于设置默认值字段的值是否随其它字段值的更新而自动更新。

关键字CHECK也是可选的，它用来指明可输入的字段值的取值范围或字段的约束条件。CHECK后面的表达式一般都是取值为真或假的表达式。您可以将关键字VALUE和CHECK一起使用来代表字段值。如果SQL命令不能满足CHECK所设定的条件，那么它就不能被执行。

column_name .... 新建的字段名称。

data_type ..... 定义字段的数据类型。

domain_name .... 代替标准数据类型来定义字段的定义域。

literal.....没有输入字段值时使用的默认值。

constant .....没有输入字段值时使用的默认常量。

function_name .....没有输入字段值时使用的内置函数名称。

constraint_name …….约束名。

boolean_expression…取值为真或假的条件表达式。

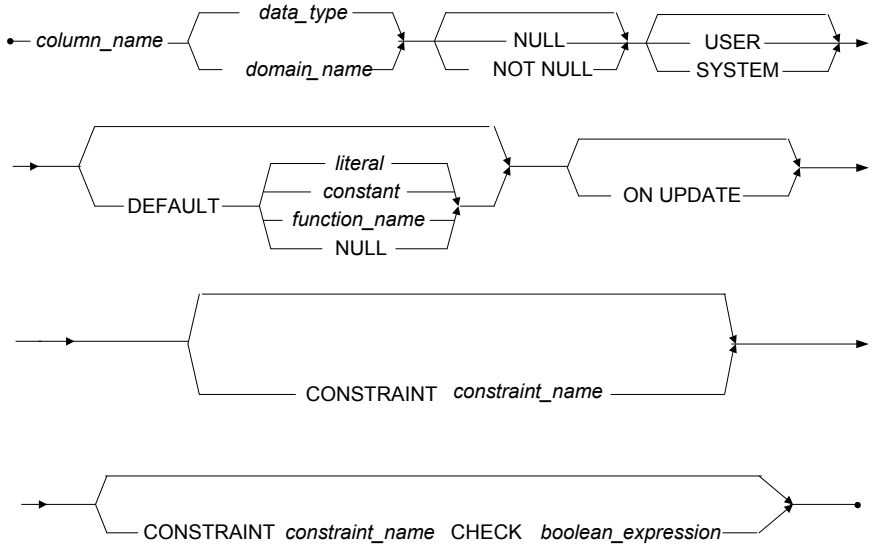


图 3-57 Column Definitions语法图

主键和唯一性定义

键（*key*）是标识表中每条记录的字段或字段组合。这些组成键的字段被称作键字段（*key columns*）。唯一键（*unique key*）代表任何两条记录在键域上的字段值都不相同。

主键（ *primary key*）是唯一标识表中每条记录的键。如果没有主键，就无法区分表中包含重复值的两条记录。数据库管理系统不允许在数值相同的字段上创建主键，也不允许主键含有相同的值。

主键要求每条记录的键值具有唯一性，这保证了表中数据的完整性。因为主键的值不能相同且不能为空，所以在定义主键字段时应加上不为空（NOT NULL）的约束条件。主键最多可包含32个字段，这些字段所占的空间不能超过4,000个字节。

一张表只能创建一个主键或唯一键，因此您不能为主键命名，DBMaster会在系统内为每张表的主键生成一个名为PrimaryKey唯一性索引，并且自动维护这个索引。因为DBMaster会自动在主键上创建索引，所以您就不必在主键字段上另建索引以图提高查询效率。

constraint_name …新创建的约束名。

column_name .... …主键包含的字段名。

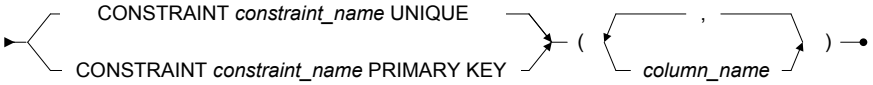


图 3-58 主键和唯一性定义语法图

外键定义

外键（*foreign key*）是对应于另外一张表的主键或唯一索引的键。这样，通过两张表相应字段的相同值建立起主从（即参照）关系。主表（被参照表）中包含了主键或唯一索引，从表（参照表）中包含了对应主表主键的外键。

参照完整性（*Referential integrity*）确保了子键值与相应的父键值是一致的。用外键创建的主从关系会强制执行两张表之间的参照完整性。通过外键的定义，DBMaster将自动支持两张表之间的参照完整性约束。要向从表中增加新记录，该从表中的值必须已存在于主表中。同样，若要从主表中删除记录，必须先删除子键中相对应的记录。

当子键参照主键时，参照完整性可能不允许对主键执行修改或删除，参照动作则提供了解决这种问题的办法。参照动作定义了当您删除或修改主键的值时，DBMaster在所有相应子键上应执行的操作。参照动作可以

发生在修改和删除这两种动作上，DBMaster提供了四种参照动作：CASCADE、SET NULL、SET DEFAULT以及NO ACTION。

关键字ON UPDATE/ON DELETE是可选的。这两个关键字表明当修改或删除主表中的字段值时，DBMaster对相应的从表要执行的参照动作。参照动作分四种：CASCADE、SET NULL、SET DEFAULT和NO ACTION。

级联（CASCADE）指修改和删除动作同时发生在从表的子键上。当主表中某条记录的主键值发生更改时，从表中相应的外键值将发生同样的更改；或者在主表中删除某条记录的主键值时，从表中相应的外键值也将被删除。

设为空值（SET NULL）是指当您更改或删除记录的主键时，DBMaster将相应的子键设为空值（NULL）。不过如果您在子键上定义了不为空（NOT NULL）的限制，那么您在此就不能使用SET NULL了。

设为默认值（SET DEFAULT）是指当您更改或删除记录的主键时，DBMaster将相应子键设为字段默认值。如果您在定义字段时将子键的默认值设为空，并且在定义子键时使用了不为空的限制，那么您在此就不能使用SET DEFAULT了。

无动作（NO ACTION）指DBMaster强制执行参照完整性规则。创建外键时如果没指定参照动作，DBMaster将默认为NO ACTION。

一张表中的外键数一般是没有实际限制的。父键可能是主键，也可能是唯一性索引，但父键必须在子键之前创建。父键和子键的字段数量、数据类型或字段长度必须一致。两张表中两组键的字段顺序可以不同，不过在定义外键时就应按主键字段的顺序列出外键字段，主表中的主键字段的顺序是系统默认的。

外键中的字段可以有空值。如果外键含有空值，那么它将自动满足参照完整性。您可以在同义字上创建外键，但不能在视图上创建。外键名最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

constraint_name ...要创建的约束名。

key_name .....…创建的外键名称。

column_name .....1. 外键包含的字段名。

....2. 被外键参照的主键中包含的字段名称。

parent_table_name.....被外键参照的表名称，即主表。

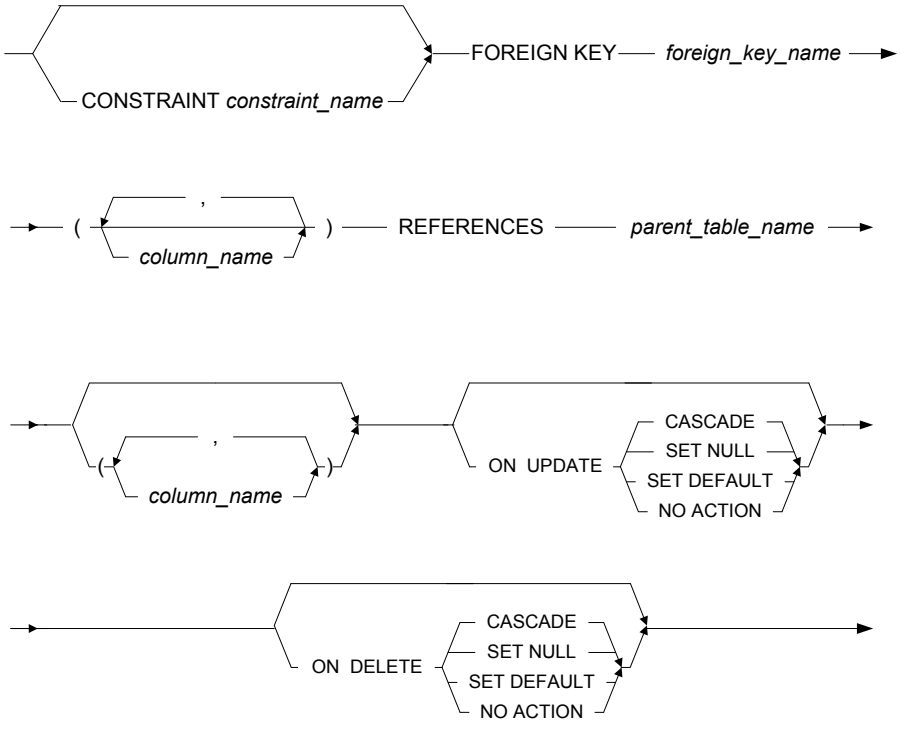


图 3-59 外键定义语法图

表的选项

DBMaster提供了大量选项以供创建表时使用。您可以使用如下关键字来设置选项的动作：TEMPORARY/TEMP、MEMORY、IN、CHECK、LOCK MODE、NOCACHE以及FILLFACTOR。

关键字TEMPORARY/TEMP是可选的，这两个关键字用于标明被创建的表是临时表还是永久表。存取临时表中的数据要快于存取永久表，这是因为临时表上不使用锁，也不用将临时表的操作写入日志文件中。不过临时表只能被表的所有者使用，并且在数据库断开后自动被删除。此外，您还可以随时使用DROP TABLE命令来删除临时表。

关键字 MEMORY 也是可选的。在DBMaster中，内存表的大多数功能和用途与固定表是一致的，不同之处在于内存表是临时表，它们的生命周期是以数据库连接为基础的。这意味着当用户删除所建的内存表或断开与数据库的连接时，内存表将从数据库中删除。与固定表不同的是，内存表只存储在创建它们的连接内存中，这些表不能被其它连接使用。用户只能在表中选择或插入数据，而不能更新或删除已有的数据。内存表支持事务控制操作，如提交，回滚，定义存储点以及回滚存储点。

以上这些关键字用于定义系统所创建的表是临时表而非永久表。因为临时表不会被锁定，并且系统也不会为临时表写日志记录，所以在临时表中存取数据是比较快的。不过，临时表只能被表的所有者使用，并且在断开与数据库的连接时，会自动删除所有临时表。并且，您也可以在已连接的数据库中，使用DROP TABLE命令来删除临时表。

关键字IN是可选的，该关键字用于指出创建表的表空间名。表空间是DBMaster的逻辑存储区域，它可以将数据库分割成几个容易管理的区域。系统根据逻辑分组或频繁使用表的存储位置来分割表。默认情况下，表是创建在系统表空间中的。

表定义中的关键字CHECK也是可选的，该关键字的用法和字段定义中的关键字CHECK的用法相似。它用来确保多个字段的取值都是在合法的取值范围里。CHECK后面的表达式一般都是取值为真或假的表达式。在这个表达式中可将字段名和关键字CHECK一同使用来代表字段值。如果SQL命令不能满足CHECK所设定的条件，它就不能被执行。

关键字LOCK MODE是可选的，该关键字表示在存取表中数据时，DBMaster应采用的锁模式（即锁定级别）。DBMaster有三种锁模式：表、页和行。页锁定是系统默认的锁定模式。用户可利用SYSTABLE表中的查询字段LOCKMODE来确定表的锁定模式。

LOCK MODE TABLE的作用是锁定整张表，该模式可阻止其他用户同时存取被锁定的表，这就降低了事务的并发性。当然这种锁定模式占用较少的锁资源，因此在系统控制区（SCA）中占用的存储空间较少。

LOCK MODE PAGE的作用是锁定一个数据页，该模式是对事务并发性和锁资源的折中考虑。这种模式提供了适当的并发行，因为它允许用户存取锁定页以外的其它数据页。

LOCK MODE ROW的作用是锁定一行，该模式可允许用户存取除锁定行以外的其他数据，因此这种模式比页锁定更进一步地增加了事务的并发行。但是这种模式与前两种模式相比占用的锁资源较多，在*SCA*中占用的存储空间也较多。

填充系数（FILLFACTOR）指数据页被填充百分比。这种为将来更新记录保留一定存储空间的方法，提高了数据页中存取数据的效率。参数*number*的取值范围为50～100，因此填充系数的值可设为50%～100%。您可以查询系统表SYSTABLE中的FILLFACTOR字段来获得表的填充系数。

无缓存（NOCACHE）可用来限制表扫描时，分配给该表的数据页缓冲区数量。DBMaster将所有的缓冲数据页存储为缓冲链的形式，将最近使用的数据页排在链头部而最久未用的数据页将排在末端。如果选择了NOCACHE这一选项，扫描表时存取的数据页将被放置在缓冲区链的末端。缓冲区链末端的内容在下一次存取数据前被覆盖掉，并且在表扫描的过程中，后来存取的数据页会覆盖先前的数据页。这将有效的限制表扫描过程中，仅使用一个数据页来充当缓冲区存放表中的数据。您可以查询系统表SYSTABLE中的CACHEMODE字段来确定表的缓冲模式。

如果您创建了一张表，那么您就是该表的所有者。您将拥有该表的所有对象权限，并且可将这些对象权限分配给其他用户。作为表的所有者，即使您的安全权限被降为CONNECT，也将继续拥有该表的所有对象权限。

注意 *DBMaster中，CHECK和CHECK VALUE的语法都已更新为SQL 99中的语法。*

➲ **例1**

下例是在系统表空间中创建了一个名为**Scores**的表，表中的字段有：**StudentNo, Math, English, Science**以及**History**，这些字段的数据类型都是**INTEGER**。

```
dmSQL> CREATE TABLE Scores (StudentNo INTEGER,
                            Math INTEGER,
                         English INTEGER,
                         Science INTEGER,
                         History INTEGER);
```

➲ **例2**

下例是在表空间**StudentRecords**中创建了一张和上例一样的表，所有字段都不能为NULL，并且字段**Math, English, Science**以及**History**的默认值是***0***，表的所有者是**Madison**。

```
dmSQL> CREATE TABLE Madison.Scores
(StudentNo INTEGER NOT NULL,
                            Math INTEGER NOT NULL DEFAULT 0,
                         English INTEGER NOT NULL DEFAULT 0,
                         Science INTEGER NOT NULL DEFAULT 0,
                         History INTEGER NOT NULL DEFAULT 0)
              IN StudentRecords;
```

➲ **例3**

下例同例2，不过本例还在字段**Math, English, Science**以及**History**中添加了字段约束：字段值的取值范围是0到100。

```
dmSQL> CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                            Math INTEGER NOT NULL DEFAULT 0
                                         CHECK VALUE >= 0 AND VALUE <= 100,
                         English INTEGER NOT NULL DEFAULT 0
                                         CHECK VALUE >= 0 AND VALUE <= 100,
                         Science INTEGER NOT NULL DEFAULT 0
                                         CHECK VALUE >= 0 AND VALUE <= 100,
                         History INTEGER NOT NULL DEFAULT 0
                                         CHECK VALUE >= 0 AND VALUE <= 100)
```

```
            IN StudentRecords;
```

➲ **例4**

下例同上，只是在本例中定义了表约束以确保**Math, English, Science**和**History**字段值的总和小于400。

```
dmSQL> CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                          Math INTEGER NOT NULL DEFAULT 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       English INTEGER NOT NULL DEFAULT 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       Science INTEGER NOT NULL DEFAULT 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       History INTEGER NOT NULL DEFAULT 0
                                       CHECK VALUE >= 0 AND VALUE <= 100)
            IN StudentRecords
            CHECK Math + English + Science + History <= 400;
```

➲ **例5**

本例同上例，只是将锁定模式设定为页（PAGE）锁定，填充系数（FILLFACTOR）设置为90%，并且开启了**NOCACHE**这一选项。

```
dmSQL> CREATE TABLE Scores (StudentNo INTEGER NOT NULL,
                          Math INTEGER NOT NULL DEFAULT = 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       English INTEGER NOT NULL DEFAULT = 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       Science INTEGER NOT NULL DEFAULT = 0
                                       CHECK VALUE >= 0 AND VALUE <= 100,
                       History INTEGER NOT NULL DEFAULT = 0
                                       CHECK VALUE >= 0 AND VALUE <= 100)
            IN StudentRecords
            CHECK Math + English + Science + History <= 400
            LOCK MODE PAGE
            FILLFACTOR 90
            NOCACHE;
```

➲ **例6a**

```
dmSQL> CREATE TABLE computer(id INT, buy time TIMESTAMP DEFAULT '2012-03-04
12:12:12', price int); //now attributes of buy_time is USER
```

```
dmSQL> INSERT INTO computer VALUES(1, '2012-10-10 10:10:20', 3400); //value of
buy time will be replaced with '2012-10-10 10:10:20' which is specified by the
user
1 rows inserted
dmSQL> INSERT INTO computer VALUES(2, '2012-10-11 10:10:20', 5400);
1 rows inserted
dmSQL> select * from computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20               3400
         2 2012-10-11 10:10:20               5400
2 rows selected
dmSQL> UPDATE computer SET price=3200 WHERE id=1; //value of buy_time will not be
updated
1 rows updated
dmSQL> select * from computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
         1 2012-10-10 10:10:20               3200
         2 2012-10-11 10:10:20               5400
2 rows selected
```

➲ **例6b**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP DEFAULT '2012-
03-04 12:12:12' ON UPDATE); //now attributes of buy_time is USER and ON UPDATE
dmSQL> UPDATE computer SET price=3000 WHERE id=1; //value of buy_time will be
replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> select * from computer;
    ID              BUY TIME              PRICE
========== ========================== ==========
         1 2012-03-04 12:12:12               3000
         2 2012-10-11 10:10:20               5400
2 rows selected
dmSQL> UPDATE computer SET price=3000, buy_time='2012-10-10' WHERE id=1;//value
of buy_time will be replaced with '2012-10-10' which is specified by the user
1 rows updated
dmSQL> select * from computer;
    ID              BUY_TIME              PRICE
========== ========================== ==========
```

```
          1 2012-10-10 00:00:00                3000
          2 2012-10-11 10:10:20                5400
2 rows selected
```

➲ **例6c**

```
dmSQL> ALTER TABLE computer MODIFY (buy_time TO buy_time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12'); //now attributes of buy_time is SYSTEM
dmSQL> INSERT INTO computer VALUES(3, '2012-11-10 10:10:20', 4700); //value of
buy_time will not be replaced with '2012-11-10 10:10:20' which is specified by
the user.
1 rows inserted
dmSQL> INSERT INTO computer VALUES(4, '2012-12-11 10:10:20', 2800);//value of
buy_time will not be replaced with '2012-12-11 10:10:20' which is specified by
the user.
1 rows inserted
dmSQL> select * from computer;
    ID               BUY_TIME              PRICE
========== ========================== ==========
          1 2012-10-10 00:00:00                3000
          2 2012-10-11 10:10:20                5400
          3 2012-03-04 12:12:12                4700
          4 2012-03-04 12:12:12                2800
4 rows selected
dmSQL> UPDATE computer SET price=4500 WHERE id=3; //value of buy_time will not be
updated.
1 rows updated
dmSQL> select * from computer;
    ID               BUY TIME              PRICE
========== ========================== ==========
          1 2012-10-10 00:00:00                3000
          2 2012-10-11 10:10:20                5400
          3 2012-03-04 12:12:12                4500
          4 2012-03-04 12:12:12                2800
4 rows selected
```

➲ **例6d**

```
dmSQL> ALTER TABLE computer MODIFY (buy time TO buy time TIMESTAMP SYSTEM DEFAULT
'2012-03-04 12:12:12' ON UPDATE); //now attributes of buy time is SYSTEM and ON
UPDATE
```

```
dmSQL> UPDATE computer SET price=4000, buy time='2015-01-01' WHERE id=3; //value
of buy time will be replaced with the default value'2012-03-04 12:12:12'
1 rows updated
dmSQL> select * from computer;
    ID               BUY_TIME              PRICE
========== ========================= ==========
         1 2012-10-10 00:00:00               3000
         2 2012-10-11 10:10:20               5400
         3 2012-03-04 12:12:12               4000
         4 2012-03-04 12:12:12               2800
4 rows selected
```

CREATE TABLE AS SELECT

CREATE TABLE AS SELECT命令用来创建一张表，该表的字段定义和表中的数据源自于SELECT语句。使用该语句创建表的字段定义就如同执行CREATE VIEW命令，为表插入数据就如同于执行SELECT INTO命令。

➲ 例

下例创建了一张表**Scores70**，它的字段和数据来自于表**Scores**中Math字段（数学成绩）大于70的**StudentNo**和**Math**。

```
dmSQL> CREATE TABLE Scores70 AS SELECT StudentNo, Math FROM Score WHERE Math >
70 IN tablespace1;
```

3.46 CREATE TABLESPACE

CREATE TABLESPACE命令用来生成一个新的表空间。新的表空间可以为数据库增加物理存储空间，只有DBA、SYSDBA或SYSADM才可以执行该命令。

DBMaster利用关系型数据模式来隐藏数据的物理存储模式，而用户所看到的数据存储模式实际上是逻辑存储模式。在DBMaster的物理存储模式中，文件是数据库数据的物理存储结构。文件一般由操作系统管理，不过在UNIX的裸设备中，文件中的数据是由数据库管理系统来管理的。DBMaster在日常操作中使用三种类型的文件：数据文件（Data），二进制大型对象（BLOB）和日志文件（Journal）。

数据文件和BLOB文件可用来存储用户和系统数据，尽管这两种文件具有相似的特性，但DBMaster对它们的管理方式却不尽相同，这将有助于提高系统的性能。数据文件用于存储表和索引的数据，而BLOB文件则用来存储二进制大型对象。

日志文件是一种特殊的文件，它是对数据库所做更改的实时或历史记录，并且记录了数据库改变后的状态。使用日志文件可使数据库撤销失败事务对它所做的更改，或在数据库灾难恢复时将已提交的事务重新写入数据库。日志文件只能由数据库管理系统使用，它并不能存储用户数据。

在DBMaster的逻辑存储模式中，表空间是数据的逻辑存储结构，可将数据库分割成几个DBMS容易管理的区域。一个表空间可能包含几个表和索引，表空间中的数据物理存储在文件中，但由数据库管理系统统一管理。表空间分为五种：固定表空间、自动扩展表空间、系统表空间、只读表空间和读写表空间。

固定表空间是大小固定的表空间，可以包含一个或多个数据文件或BLOB文件。您可以通过扩大表空间中文件的大小或向表空间中增加文件的方式来手动扩展表空间。在新增文件之前，应首先在配置文件**dmconfig.ini**中设置文件的逻辑文件名、物理文件名以及文件的初始大小。一个固定表空间中最多可包含32,767个文件，所有文件所占空间的总和不能超过8TB。在UNIX中，固定表空间可以放在裸设备上。

注意 *有关裸设备的信息，请查阅Unix系统文档。*

自动扩展表空间（autoextend *tablespaces*）是视需要而自动延伸的表空间。固定表空间和自动扩展表空间都可以包含一个或多个数据文件和BLOB文件。自动扩展表空间也有可能出现空间用完的情况，这是因为这种表空间中的文件大小不能超过8TB或磁盘已满。您可以向表空间中增加文件或扩大现有文件的大小来手动扩展表空间。向自动扩展表空间中插入大量数据时，可预先分配空间以提高执行效率。但要注意的是，自动扩展表空间是不能用在裸设备上的。

创建数据库时，DBMaster会自动生成系统表空间。每个数据库只有一个系统表空间，用来存储系统表，而系统表是用来记录整个数据库的存储模式、安全和状态信息的表。系统表空间是一种特殊的自动扩展表空间。系统表空间包含一个数据文件和一个BLOB文件，这两个文件是在创建表空间时自动生成的，不能用来存储用户数据。系统表空间可以更改为固定表空间。同样，系统表空间也不能在裸设备中使用。

关键字AUTOEXTEND是可选的，用来表示创建的表空间是否是自动扩展表空间。当系统需要额外的空间时，自动扩展表空间可自动增加自己的大小。任何时候，自动扩展的表空间和固定表空间都可以相互转化。

关键字BACKUP BLOB是可选的。当数据库处于BACKUP_DATA_AND_BLOB模式时，可使用该关键字来指示DBMaster是否要备份表空间中的BLOB数据。当BACKUP BLOB设为ON时，处于BACKUP_DATA_AND_BLOB模式下的DBMaster将备份表空间中的所有BLOB数据；当BACKUP BLOB设为OFF时，那么不管DBMaster处于何种备份模式下，都不会备份BLOB数据。

为了确保数据库中的数据独立性，操作系统文件不应该在数据库中直接被存取。为此，每个数据库文件都有两个名字：物理文件名和逻辑文件名。前者是操作系统使用的文件名，后者是在数据库中使用的文件名。这两个文件名可通过**dmconfig.ini**文件进行交互。在执行CREATE TABLESPACE命令前，首先应打开**dmconfig.ini**文件，在数据库配置信息中设置文件的逻辑文件名、物理文件名以及文件的初始大小（见示例）。

在创建表空间时，关键字DATAFILE指定了创建文件的逻辑名称和类型。在表空间类型允许并且磁盘空间足够的情况下，表空间中最多可容纳32,767个文件。每个表空间中至少得包含一个数据文件。如果您想在表空间里增加文件，可使用ALTER TABLESPACE命令来操作。

关键字TYPE的作用是指定DBMaster所创建的文件是数据文件还是BLOB文件。用法如下：TYPE=DATA表示生成一个新的数据文件，TYPE=BLOB表示生成一个新的BLOB文件。如果没有用TYPE来指明文件类型，系统将默认生成一个数据文件。

新建文件时，如果没有另外指定目录或路径，DBMaster会将文件存储在系统默认的目录下，该目录是由**dmconfig.ini**中的**DB_DbDir**关键字所指定的。数据文件的初始大小由数据页来衡量，而BLOB文件的初始大小由BLOB帧来衡量。

一个数据文件可包含2～2147483647个数据页，将页数乘以**DB_PgSiz**的值就能得到文件的实际大小。一个BLOB文件可包含2-524,287个帧。将文件中的帧数量乘以**dmconfig.ini**文件中的**DB_BfrSz**值，就可得到该BLOB文件的实际大小。

表空间中的文件不一定都位于同一个磁盘上，您可以将这些文件存储在不同的磁盘上的，也可以将它们存放在同一磁盘的不同目录上。在Unix系统中，文件可以存储在裸设备的固定表空间中，这时DBMaster可以不用通过操作系统的调用，直接向裸设备中写入文件，这种方式提高了在普通文件上的读取速度和操作效率。

表空间名和逻辑文件名的最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头，表空间名区分大小写。

物理文件名，包括驱动器名和路径名在内最长不能超过255个字符，它可以包括任何操作系统允许的字符，但要注意的是不能包含空格，物理文件名是否区分大小写要视操作系统而定。

tablespace_name……新建的表空间名。

file_name ..... ……物理表空间的逻辑文件名。

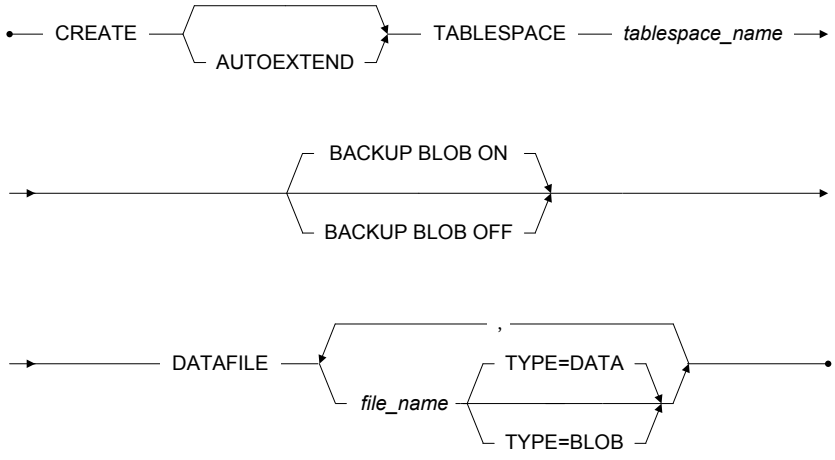


图 3-60 CREATE TABLESPACE语法图

➲ **映射1**

您在执行例1中的命令前，需要在**dmconfig.ini**文件中加上如下一行代码，用来将逻辑文件名映射为物理文件名，并且指定数据文件所包含的页数或**BLOB**文件所包含的帧数。对于数据文件，如果采用默认的数据页大小为**8KB**，那么下面所述的文件将占**800KB**；对于**BLOB**文件，因为使用了默认的帧大小，即**32KB**，那么下面的文件将占**3200KB**。

```
datafile = c:\dbmaster\database\ ts_reg_df.db 100
blobfile = c:\dbmaster\database\ ts_reg_bf.bb 100
```

➲ **例1**

下例是创建一个名为**ts_reg**的固定表空间，并在其中创建逻辑文件名为**datafile**的数据文件和名为**blobfile**的BLOB文件。其中表空间所包含的文件不能超过32767个。

```
dmSQL> CREATE TABLESPACE ts_reg DATAFILE datafile TYPE=DATA, blobfile TYPE=BLOB;
```

➲ **映射2**

您在执行例2中的命令前，需要在**dmconfig.ini**文件中加上如下一行代码，用来将逻辑文件名映射为物理文件名，并且指定数据文件所含的页数或**BLOB**文件所含的帧数。对于数据文件，如果采用默认的数据页大小为**8KB**，那么下面所述文件将占**800KB**；对于**BLOB**文件，因为使用了帧默认的大小，即**32KB**，那么下面的文件所占的空间为**3200KB**。

```
datafile = c:\dbmaster\database\ts_ext_df.db 100
blobfile = c:\dbmaster\database\ts_ext_bf.bb 100
```

➲ **例2**

下例是创建一个名为**ts_ext**的自动扩展表空间，并在其中创建一个逻辑文件名为**datafile**的数据文件和名为**blobfile**的BLOB文件，这时您可能就不能向这个表空间中再加入数据文件或**BLOB**文件了。

```
dmSQL> CREATE AUTOEXTEND TABLESPACE ts_ext DATAFILE datafile TYPE=DATA,
                                          blobfile TYPE=BLOB;
```

3.47 CREATE TEXT INDEX

DBMaster可创建两种类型的全文索引：特征全文索引和反向全文索引。创建特征全文索引的表空间与创建索引的字段表空间相同，而反向全文索引则创建一个单独的文件上，这样可提高大索引的执行效率。

CREATE TEXT INDEX命令可在一个或多个字段上创建全文索引。使用全文索引执行检索时，系统可不用搜索整张表而快速地在包含文本的字段中查询到指定的单词，这提高了全文查询的效率。只有表的所有者、DBA、SYSDBA、SYSADM或在指定表上拥有INDEX权限的用户才可以执行该命令。

全文索引是一种快速存取记录的机制，这些记录的字段一般包含一个或多个单词和短语。全文索引包含了文本字段中查询到的所有文本的陈述。在这种索引中，数据被重新组织编码，这样的检索方式比直接查询表要快得多。全文索引的具体操作对数据库的用户而言是透明的，数据库管理系统可使用它来提高全文查询效率。

创建全文索引时，需要指明索引名、表名和字段名。全文索引可以创建在CHAR、VARCHAR、CLOB、NCHAR、NVARCHAR、NCLOB或FILE类型的字段上。全文索引不能创建在系统表、临时表或视图上。

使用Order By子句可以在一个字段上查询一个或多个单词，并且在另一字段上对查询的结果进行排序。创建全文索引时，如果使用了Order By Column，那么DBMaster将按照Order By Column所指定的顺序输出查询结果。例如，您想查询*content*字段，并且将查询结果在*post time*字段上排序，您只需在查询命令后面加上Order By Post Time子句。如果使用了order by子句，DBMaster则必须对查询结果进行排序，不过排序是比较费时的。如果您在创建全文索引时就使用了Order By *Post Time*子句，那么就算您没有在查询命令后加上Order By子句，也能得到一个在Post time字段上排序后的结果。您可以用ASC或DESC关键字来指定排序是递增还是递减，默认的排序顺序是升序。重建索引后，Order By Column这一子句可对新增的部分进行影响，不过它不能将新旧字段一起重新排序。

如果向表中载入数据，DBMaster并不能自动更新该表上的全文索引，因此您应该在表创建索引之前，尽可能地将所有数据载入完毕。全文索引创建后，新加的记录即使符合查询条件，用match运算也找不到这些记录。为了在查询时能搜索到这些新增加的记录，您必须使用REBUILD TEXT INDEX命令来重建全文索引。

每张表的全文索引名必须唯一，名字的最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#，但不能以数字开头。

特征全文索引（Signature Text Index）

特征全文索引可以创建在所有字符类型的字段上，包括CHAR、VARCHAR、LONG VARCHAR、NCHAR、NVARCHAR、NCLOB以及FILE类型。一张表可以包含多个全文索引，一个全文索引也可以创建在多个字段上。

TOTAL TEXT SIZE是创建全文索引字段的所有文档总大小，单位为MB，它的范围是1-200，默认值为32。该值是DBMaster用来估计和进行优化效率的，它不是用来限定一个字段能容纳多少个文档。如果该值超过了200MB，那么就只能够使用200MB，或创建反向全文索引来提高查询效率。

SCALE用以预测索引占整个字段大小的比率。如果您将TOTAL TEXT SIZE 设为20并且估算出索引要使用10MB的存储空间，那么您就应该将SCALE设为50（即50%）。Scale值越大，查询效率越高。其取值范围是10-200，默认值是***40***。

text_index_name……...要创建的全文索引名。

table_name..... ……创建全文索引的表名称。

column_name……..创建索引的字段名称。

order_column_name….排序的字段名。

number..... ……用来设置参数SCALE和TOTAL TEXT SIZE的值。

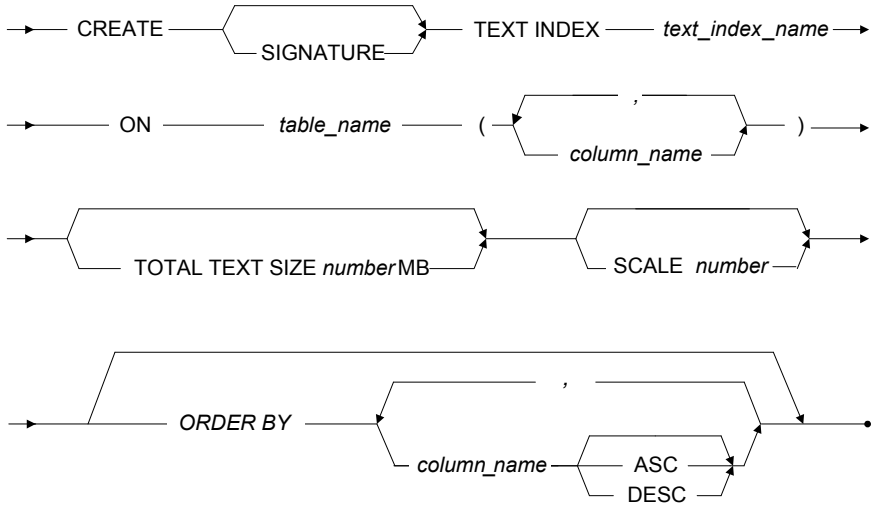


图 3-61 CREATE SIGNATURE TEXT INDEX语法图

➲ **例1**

下例表示在表**Employeesinfo**的**FName**字段上创建一个名为**TxtIdx**的特征全文索引。所有参数都使用系统默认值，查询结果在**Emp_ID**字段上排序。

```
dmSQL> CREATE SIGNATURE TEXT INDEX TxtIdx ON Employeesinfo(FName) ORDER BY
Emp_ID;
```

➲ **例2**

下例表示在表**Employeesinfo**的**FName**字段上创建了一个名为**TxtIdx**的特征全文索引。字段大小的估计值是20MB，所建索引大小占字段大小的50% 。

```
dmSQL> CREATE SIGNATURE TEXT INDEX TxtIdx ON Employeesinfo(FName) TOTAL TEXT SIZE
20 MB SCALE 50;
```

反向全文索引（Inverted File Text Index）

CREATE IVF TEXT INDEX命令可在指定字段上创建一个新的反向全文索引。反向全文索引可以替代标准索引并且能提高查询效率，尤其是在字段中的数据超过200MB的情况下。

只有表的所有者、DBA、SYSDBA或SYSADM才能创建反向全文索引。

反向全文索引属于操作系统的文件系统，由数据库管理系统来管理。创建索引时就应该指定反向全文索引应存储的位置。DBMaster可管理反向全文索引根目录下的子目录。

text_index_name.....新创建的全文索引名。

table_name.....创建全文索引的表名称。

column_name.....创建全文索引的字段名。

path.....存储索引的完整路径名。

order_column_name.....排序的字段名。

number.....用来设置参数SCALE和TOTAL TEXT SIZE的值。

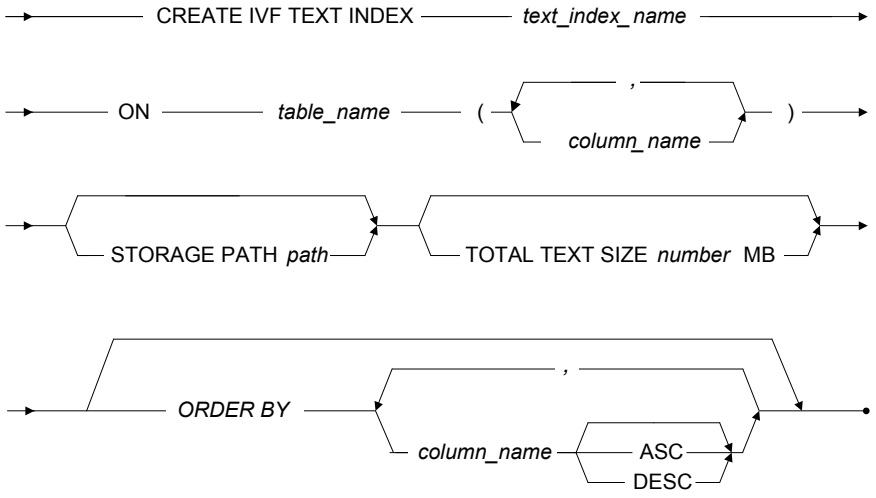


图 3-62 CREATE IVF TEXT INDEX 语法图

➲ **例1**

下例在表**Employeesinfo**的**LName**字段上创建了一个名为**Txtldx**的反向全文索引，其中索引的参数都采用默认值。

```
dmSQL> CREATE IVF TEXT INDEX TxtIdx ON Employeesinfo(LName);
```

➲ **例2**

下例在表**Employeesinfo**的**LName**字段上创建了一个名为**TxtIdx**的反向全文索引，该索引存储于逻辑文件**DB_IvfDir**所指的目录中，字段大小的估计值是100MB。

```
dmSQL> CREATE IVF TEXT INDEX TxtIdx ON Employeesinfo(LName) STORAGE PATH
DB_IVFDIR TOTAL TEXT SIZE 100 MB ORDER BY Emp_ID ASC;
```

3.48 CREATE TRIGGER

使用CREATE TRIGGER命令可在表上生成一个新的触发器。使用触发器可以在数据库上按用户要求执行命令，而这些命令可能是标准SQL命令无法完成的。只有表的拥有者、DBA、SYSDBA或SYSADM，具有执行触发器中所定义的SQL命令所需的所有安全和对象权限的用户才能执行该命令。

触发器是一种数据库服务器机制，所设定的事件一旦发生，就会导致触发器自动执行一些事先定义好的指令。这可以使数据库执行一些标准SQL指令不能执行的复杂或非常规操作。因为触发器是由数据库服务器控制的，所以无论事件是由哪个用户或哪个应用程序所造成的，触发器都能保证数据的一致性。触发器的操作对用户而言是透明的。

创建触发器时，您需要指定触发器的名称，触发时间（也就是相对于触发事件触发器被执行的时间），触发事件（即引起触发器被执行的事件），触发表（即创建触发器的表），触发型态（即执行触发器的类型）以及触发动作（即执行触发器时数据库执行的操作，一般为一些SQL命令）。删除表时，如果选择CASCADE关键字，在这个表上创建的所有触发器都将自动被删除。

DBMaster将触发器名和表名结合起来作为触发器的完整名称，所以同一张表上的触发器名必须唯一。触发动作中所定义的SQL操作是以触发器创建者的安全和对象权限来执行，并非以执行触发事件使用者的权限来执行。

用户可使用关键字BEFORE/AFTER来指明数据库服务器应何时根据触发事件来执行相应的触发动作，即触发动作的时机。关键字BEFORE表示数据库服务器应该在触发事件发生之前执行触发动作，关键字AFTER表示数据库服务器应该在触发事件发生之后执行相应的触发动作。

关键字INSERT/DELETE/UPDATE代表可以引起触发器执行的事件，也就是触发事件。INSERT指在表中新增记录时触发相应的动作；DELETE指从表中删除记录时触发动作；UPDATE指更新表的字段时触发相应的动作。如果您希望某些字段更改后再执行触发器，那么您可以使用“UPDATE OF”来指定一个字段列表，用于表示触发事件是在哪些字段

上的更新。使用“UPDATE OF”指定一个字段列表来限制该表某些字段的UPDATE触发器动作。

关键字ON后面的表名指触发器是创建在哪个表上的，也就是触发表。触发表必须是数据库中永久存在的表，而不能是一些临时表、视图或者同义字。每个触发器只能创建在一张表上。

OR REPLACE：若触发器已存在，可指定OR REPLACE选项重建该触发器，即用户可使用该子句更改已存在触发器的定义。

trigger_name.....所创建的触发器名。

column_name ....创建触发器的字段名。

table_name .....创建触发器的表名。

sql_statement.....触发器被执行时所要执行的SQL命令。

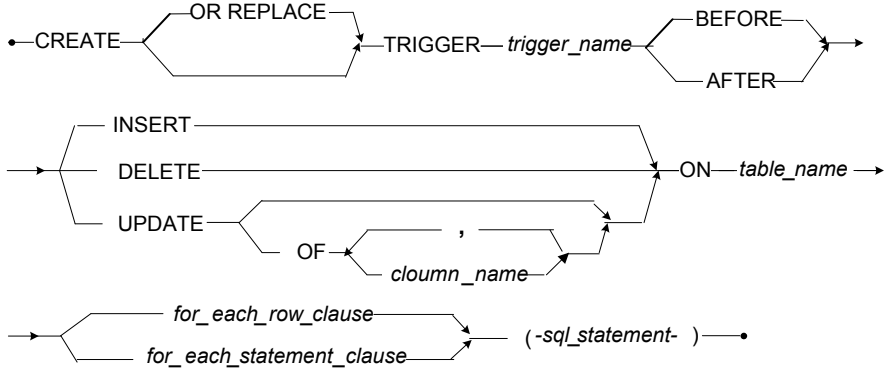


图 3-63 CREATE TRIGGER语法图

For Each Row子句

REFERENCING关键字可用来为关键字OLD和NEW定义别名。当您创建一个行触发时，您经常需要在触发动作中指明是否需要参照触发事件发生之前或之后字段中的值。如果您的表名恰好就是OLD或NEW，那么您

就可以使用REFERENCING关键字指代的别名来代替关键字OLD和NEW。

FOR EACH ROW关键字代表行级触发，即如果触发事件里有一笔记录被更改，那么触发动作就会执行一次。如果触发事件不是针对记录而进行的，那么关键字FOR EACH ROW定义的行触发器将不会被执行。

关键字WHEN的意思是当符合条件的记录被更改一次时，DBMaster将执行一次触发器。当发生触发事件时，WHEN子句会针对每条记录来计算。如果每条记录WHEN条件计算出来的结果是真，则对于这条记录的触发动作就会执行，如果WHEN条件计算结果是假，则触发动作将被跳过。WHEN条件的结果只会影响触发动作的执行，而不会影响触发事件的状态。

old_name ..... 触发动作执行之前，用来参照字段旧值的别名。

new_name ..... 触发动作执行之后，用来参照字段新值的别名。

search_condition ....触发动作执行前需要满足的条件。

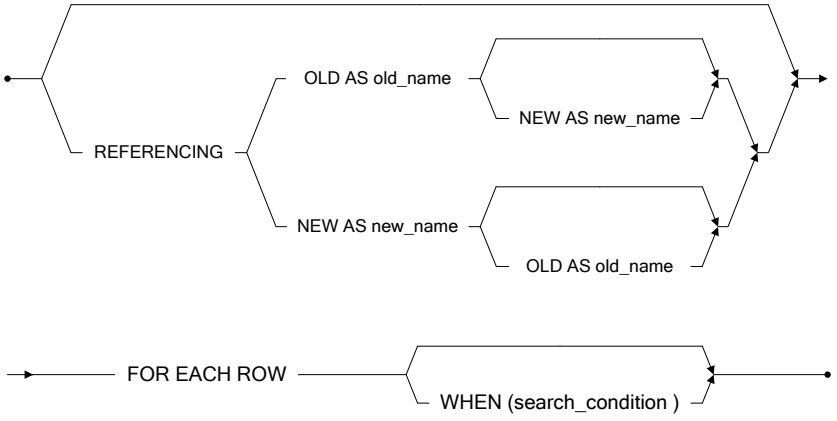


图 3-64 For Each Row Clause 语法图

For Each Statement子句

关键字FOR EACH STATEMENT指语句触发，表示针对每一个触发事件，触发动作只会执行一次。如果使用该关键字，即使触发事件的语句没有对记录进行操作，也可以执行触发动作。

触发器执行的语句称为触发动作（*trigger action*）。触发动作可以是INSERT、UPDATE、DELETE或是执行存储过程（EXECUTE PROCEDURE）的语句。定义触发动作时，执行的程序只可以是诸如PI( )、NOW( )或USER( )这种不需要输入自变量的内置函数。触发器要执行的存储过程中不能包含COMMIT、ROLLBACK或SAVEPOINT等事务控制语句。

在每个触发事件上，您可以使用触发动作时机（包括关键字BEFORE和AFTER）和触发型态（包括关键字FOR EACH ROW 和FOR EACH STATEMENT）的组合来产生四种可能的触发。例如，您可以为新增（INSERT）触发事件产生四种触发：BEFORE/FOR EACH STATEMENT、BEFORE/FOR EACH ROW、AFTER/FOR EACH ROW以及AFTER/FOR EACH STATEMENT。在UPDATE和DELETE触发器上您也可以采取相同的结合来产生四种触发。

触发事件定义时如果没有使用表更新（UPDATE），而使用的是字段更新（UPDATE OF），那么可以用*trigger action time/trigger type combination*模式来为每个字段产生一种触发。如果一张表有四个字段，您可以创建四个UPDATE OF触发：BEFORE/FOR EACH STATEMENT、BEFORE/FOR EACH ROW、AFTER/FOR EACH ROW以及AFTER/FOR EACH STATEMENT。定义触发时，UPDATE OF和UPDATE是不能同时使用的。

每张表上的触发器名必须唯一，最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

FOR EACH STATEMENT

图 3-65 For Each Statement Clause 语法图

➲ **例1**

下例在**Employeesinfo**表上创建了一个名为**Trig_update**的**UPDATE**触发器，将更新前后的值插入到表**NameChange**中。该触发动作是在行更新之前被执行的，并且与字段的更新顺序无关。

```
dmSQL> CREATE TRIGGER Trig_update BEFORE UPDATE ON Employeesinfo
                    FOR EACH ROW
                   (INSERT INTO NameChange
                         VALUES (OLD.FName, OLD.LName,
                                 NEW.FName, NEW.LName));
```

➲ **例2**

下例在表**Employeesinfo**上创建了一个名为**Trig_insert**的**INSERT**触发器，作用是在**Employeesinfo**中插入一条新记录同时执行存储过程**SendMail**。本例中通过关键字**REFERENCING**为**OLD**和**NEW**这两个关键字定义了别名。每向表中插入一条记录后，系统就会执行一次触发。

```
dmSQL> CREATE TRIGGER Trig_insert AFTER INSERT ON Employeesinfo
                    REFERENCING OLD AS pre NEW AS post
                    FOR EACH ROW
                   (EXECUTE PROCEDURE SendMail(pre.FName,
                                               pre.LName,
                                               WelcomeMessage));
```

➲ **例3**

下例在表**Orders**上创建了一个名为**Trig_update**的**UPDATE**触发器，其作用是在更新**Orders**表时执行存储过程**LogTime**。不管触发事件更新了多少条记录，更新事件前触发动作都将只执行一次。

```
dmSQL> CREATE TRIGGER Trig_update BEFORE UPDATE ON Orders
                    FOR EACH STATEMENT
                   (EXECUTE PROCEDURE LogTime);
```

➲ **例4**

假设数据库中存在名为tb_staff和tb_change的两个表，如下所示：

```
dmSQL> CREATE TABLE tb_staff (FName char(10), LName char(10));
dmSQL> CREATE TABLE tb_change (new_FName char(10), new_LName char(10), old_FName
                               char(10), old_LName char(10));
```

创建或替换触发器trig_update：

```
dmSQL> CREATE OR REPLACE TRIGGER trig_update BEFORE UPDATE ON tb_staff
       FOR EACH ROW   (INSERT INTO tb_change VALUES (NEW.FName, NEW.LName,
       OLD.FName, OLD.LName));
```

3.49 CREATE VIEW

CREATE VIEW命令可为现有表或视图创建一个新视图。只有拥有RESOURCE权限的基础表的所有者、或在指定表和视图上拥有SELECT权限的用户才可以执行该指令。

视图是基于现有表或视图的虚拟表。视图对用户而言就像一个包含了指定行和列的真实表，然而它并不能像真实表一样永久存储在数据库中。通过视图所看到的数据实际上是存储在原始表中的数据，而并非物理存储于数据库中的，存储于数据库的只是视图的定义和用户定义的视图名。视图的定义实际上是一条SQL查询，使用视图时，DBMaster将使用这条查询语句从原始表中获得数据。

您可以为数据库的每个用户提供不同的视图，这种限制数据存取的方式使用户只能看到他权限范围内的数据，从而加强了数据库的安全性。视图可以让用户不受后台数据库结构变化的影响，因为即使在基础表发生改动后，视图仍能为用户提供稳定的数据库映射。

视图可将几个表的数据连接或分组，并将这些数据存放在一张表中，这就简化了数据库中数据的组织。用户可为查询返回的结果加上条件限制来显示基础表中记录的子集。

相对于使用实表而言，使用视图有两个不足之处：性能和对更新的限制。在视图上执行查询的效率比直接在原始表上执行查询的效率低。要在视图上查询数据，数据库必须先检索视图的定义，然后将这个定义中的查询创建在原始表中，接下来才能执行查询，最后显示查询结果。如果将视图定义在多个表时，那么您就不能在视图上执行更新操作，因为数据库无法在这种复杂的视图上执行更新命令。

视图定义中的SELECT语句中不能包含INTO子句。目前，DBMaster仅支持创建在单一表上的视图的更新操作。

定义视图时可以指定字段。这里指定的字段数必须和SELECT语句中指定的字段数一致。如果没有指定字段名，那么视图将继承基本表中的所有字段。

视图名和字段名最大长度不能超过128个字符，可以包含数字、字母、下划线以及$和#符号，但不能以数字开头。

OR REPLACE：若视图已存在，可指定OR REPLACE选项重建该视图，即用户可使用该子句更改已存在视图的定义。

view_name.....所创建的视图名。

column_name .....视图中的字段名。

select_statement…定义视图内容的SELECT语句。

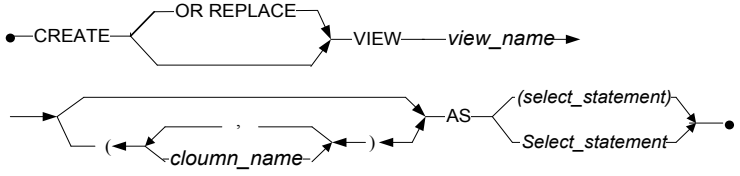


图 3-66 CREATE VIEW语法图

➲ **例**

下例在表**Employeesinfo**上创建一个名为**View_emp**的视图。

```
dmSQL> CREATE VIEW View_emp AS SELECT Name, Salary FROM Employeeinfo WHERE Salary
> 50000;
```

或

```
dmSQL> CREATE OR REPLACE VIEW View_Emp AS SELECT Name, Salary FROM Employeesinfo
WHERE Salary >= 100000;
```

3.50 DECLARE SET

DECLARE SET命令用于定义当前连接的连接变量。只有该当前连接的用户可以执行该命令，且该变量只在当前连接有效。

CV是一个仅能在当前连接中定义的连接变量。各个连接的连接变量互不影响。也就是说，只有连接变量拥有者有权使用这些连接变量，其它连接无法获得或使用它们。

对用户来说，连接变量是当前连接中SQL命令的全局变量，可在dmSQL命令行工具和SQLSP中使用。一旦该连接断开，所有连接变量将会被自动释放。

执行该命令将会存储一个通过类型和值定义的值。CV可替换任一SQL命令的表达式值。

使用CV时，用户需在变量名前添加符号@，否则dmSQL工具会将该变量名识别为字段名或其它标识符。此外，CV名称不区分大小写。

data_type ....CV数据类型。

@variable_name...变量名。

expression ..... …表达式的结果是该变量的值。

该表达式可以是简单的赋值表达式，也可以是C函数和Lua函数，如内置函数和用户自定义函数。

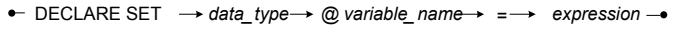


图 3-67 DECLARE SET语法图

➲ **例1**

下例创建一个名为**aa**的连接变量，并将其值设置为1，类型设置为INT。

```
dmSQL> DECLARE SET INT @aa = 1;
dmSQL> SELECT @aa;
          @AA
======================
1
```

➲ **例2**

下例创建一个名为**bb**的连接变量，并将其值设置为syscom，类型设置为CHAR(20)。

```
dmSQL> DECLARE SET CHAR(20) @bb = 'syscom';
dmSQL> SELECT @bb;
          @BB
======================
SYSCOM
```

➲ **例3**

下例创建一个名为**cc**的连接变量，并将其值设置为表达式结果。

```
dmSQL> DECLARE SET INT int @cc1 = 100+200;
dmSQL> DECLARE SET INT @cc2 = @cc1+300;
dmSQL> SELECT @cc1;
          @CC1
======================
               300
dmSQL> SELECT @cc2;
          @CC2
=======================
600
```

➲ **例4**

下例创建一个名为**dd**的连接变量，并将其值设置为表达式结果。

```
dmSQL> DECLARE SET CHAR(40) @dd = CONCAT('abcd','efgh');
dmSQL> SELECT @dd;
          @DD
======================
abcdefgh
```

➲ **例5**

下例创建一个名为**ee**的连接变量，并将其值设置为表达式结果。

```
dmSQL> DECLARE SET DOUBLE @ee = 9.999999*100;
dmSQL> SELECT @EE;
         @EE
======================
9.99999900000000e+002
```

3.51 DELETE

DELETE命令可删除表中符合条件的记录，该指令一次只能删除一张表中的记录，但系统表中的记录是不能被删除的。只有表的拥有者、DBA、SYSDBA、SYSADM或在指定表上拥有DELETE权限的用户才能执行该指令。DBMaster只删除符合条件的记录，同时只有在ODBC程序中才能使用游标。

关于查询条件的更多信息可参看SELECT命令的WHERE子句。

table_name.....…要删除记录所在的表名。

search_condition...执行删除命令时记录应该满足的条件。

cursor_name.....…该命令将删除游标名所指的记录。

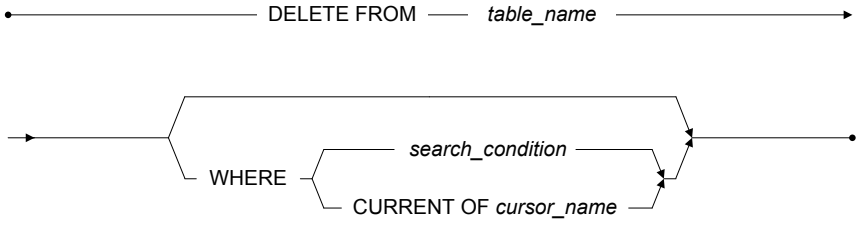


图 3-68 DELETE语法图

➲ **例1**

下例是将表**Employeesinfo**中职工编号为1234的记录删除。

```
dmSQL> DELETE FROM Employeesinfo WHERE Emp_ID = '1234';
```

➲ **例2**

下例是将职工姓名中以**John**开头的记录从表**Employeesinfo**中删除。

```
dmSQL> DELETE FROM Employeesinfo WHERE FName LIKE 'John%';
```

3.52 DROP COMMAND

DROP COMMAND命令用来删除数据库中现有的存储命令。只有存储命令的拥有者、DBA、SYSDBA或SYSADM才能执行该指令。

存储命令是编译过的SQL数据操纵指令，它们以可执行文件的格式永久存储在数据库中。用户可反复执行存储命令，而不用等待DBMaster编译和优化这条指令。存储命令和存储过程有些相似，不同之处在于前者只包含一条指令，但不包含程序逻辑。

如果您在删除存储命令参照的表或字段，或者更改表、修改字段定义或使用关键字BEFORE和AFTER来增加新字段，那么存储命令将变得无效，并且将不能再被使用。如果在新增字段时没有使用BEFORE和AFTER关键字，那么这种表修改将不会对存储命令产生影响。当一个存储命令不在有意义后，我们可以将它从数据库中删除。

IF EXISTS：确保在存储命令不存在的情况下不报错。

command_name ……需要从数据库中删除的存储命令名。

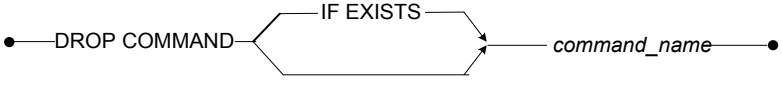


图 3-69 DROP COMMAND语法图

➲ **例**

下例中将删除一个名为**sc_select**的存储命令。

```
dmSQL> DROP COMMAND sc_select;
```

或：

```
dmSQL> DROP COMMAND IF EXISTS sc_select;
```

3.53 DROP DATABASE LINK

DROP DATABASE LINK命令可用来从数据库中删除现有的公用或私用数据库链。只有私用数据库链的所有者才能删除自己所建的数据库链，也只有DBA、SYSDBA或SYSADM才能删除公用数据库链。

数据库链创建了与远程数据库的连接，使用户可以通过本地的数据库来访问远程的数据库。数据库链还提供了额外的安全性信息。数据库链可以使用户以不同的用户名连接到远程数据库中，或使用公用数据库链接来连接远程数据库，使用公用数据库链接并不要求您在连接时提供帐号。

关键字PUBLIC/PRIVATE是可选的，这两个关键字可用来设置要删除的数据库链的类型是公用还是私用。公用链可供数据库中的所有用户使用，只有创建者才能使用自己所建的私用链。如果执行删除数据库链时没有指明数据库链的类型，那么DBMaster将默认删除私用数据库链。

link_name ..... 将从数据库中删除的数据库链名。

图 3-70 DROP DATABASE LINK语法图

➲ **例1**

下例将删除一个名为**FieldLink**的私用数据库链。

```
dmSQL> DROP PRIVATE DATABASE LINK FieldLink;
```

➲ **例2**

下例将删除一个名为**FieldLink**的公用数据库链。

```
dmSQL> DROP PUBLIC DATABASE LINK FieldLink;
```

3.54 DROP DOMAIN

DROP DOMAIN命令可用来删除数据库中现有的定义域。只有定义域的所有者、DBA、SYSDBA或SYSADM才有权执行该指令。

定义域是一种用户定义的数据类型，可用来集中设定数据类型、默认值、字段约束。在CREATE TABLE或ALTER TABLE ADD COLUMN命令中定义字段时，就可以使用定义域来替代标准的数据类型，为字段定义一组有效的输入值。

关键字CASCADE/RESTRICT是可选的。它们决定了在删除定义域时，是将参照它的对象一并删除还是检查该定义域是否被参照。如果选择CASCADE关键字，数据库会删除所有参照该定义域的依赖对象，并且用定义域的定义来替换字段定义。如果选择RESTRICT关键字，数据库不会删除任何一个被表定义参照的定义域。RESTRICT关键字确保了只有没被任何对象参照的定义域才可以被删除。

domain_name.....要从数据库删除的定义域名。

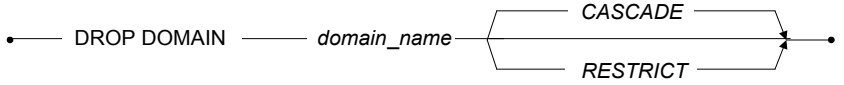


图 3-71 DROP DOMAIN 图

➲ **例**

下例将删除一个名为**ValidDate**的定义域。

```
dmSQL> DROP DOMAIN ValidDate;
```

3.55 DROP GROUP

DROP GROUP命令可以从数据库中删除一个组，只有DBA、SYSDBA或SYSADM才可以执行该指令。

当数据库中的用户数很多时，组可以简化对象权限的管理。您可以将要求同样对象权限的用户集合到一个组中，任何赋予该组的对象权限都将自动赋予组里的所有成员。DBMaster支持嵌套组，即将一个组作为成员加入到另一个组中，不过值得注意的是：将一个组添加到另一个组时不能构成循环。

当在数据库中删除组时，组中的所有成员都将失去赋给该组的所有权限。但这些成员将继续保持直接赋予他们的权限和继承于其它组的权限。PUBLIC组是不能被删除的，它由DBMaster在系统内部进行管理。

group_name..... 要删除的组名。

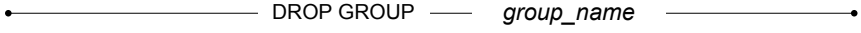


图 3-72 DROP GROUP 语法图

➲ **例**

下例从数据库中删除一个名为**Manager**的组。

```
dmSQL> DROP GROUP Manager;
```

3.56 DROP INDEX

DROP INDEX命令可从数据库中删除现有表的索引。只有表的所有者、DBA、SYSDBA、SYSADM或拥有指定表INDEX权限的用户才能执行该命令。

索引是一种在表中通过一个或多个字段（例如键）值来快速查找指定记录的机制。索引包含的数据和键字段中的数据一样，但这些数据通过组织和排序使得检索速度得到了提高。一旦在表上创建了索引，它的操作对用户而言就是透明的，数据库管理系统（DBMS）在必要时会使用索引来提高查询速度。

您可以从数据库中删除系统表以外的其它任何表的索引。如果一个索引被其它表的外键参照，那么在删除该索引之前应首先删除这些外键。如果一个索引变成碎片且降低系统效率，那么您就应该将它删除，然后重建一个密集型且没有碎片的索引。

index_name .....要删除的索引名。

table_name .....从哪个表上删除索引。

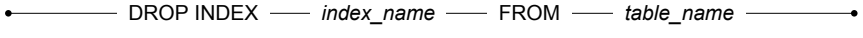


图 *3-73* DROP INDEX语法图

➲ **例**

下例是删除表**Employeesinfo**的索引**NameIndex**。如果有外键参照该索引，那么在删除它之前要先删除这些外键。

```
dmSQL> DROP INDEX NameIndex FROM Employeesinfo;
```

3.57 DROP PROCEDURE

DROP PROCEDURE命令可从数据库中删除现有的存储过程。只有表的所有者、DBA、SYSDBA、SYSADM或拥有指定表PROCEDURE权限的用户才能执行该命令。

IF EXISTS..... 确保在存储过程不存在的情况下不报错。

procedure-name..... 要从数据库中删除的存储过程名。

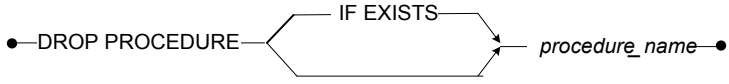


图 3-74 DROP PROCEDURE语法图

➲ **例**

下例删除存储过程**sp_proc1**：

```
dmSQL> DROP PROCEDURE sp_proc1;
```

或：

```
dmSQL> DROP PROCEDURE IF EXISTS sp_proc1;
```

3.58 DROP REPLICATION

DROP REPLICATION命令可从数据库中删除现有表的复制。只有表的所有者、DBA、SYSDBA或SYSADM才能执行该命令。

表复制可在远程目标表中生成源表的一个完整或部分的复本。远程用户可以操作复制过来的数据，同时这些数据又与另一个远端数据库保持同步。这样，每一个数据库都可以快速有效地响应数据请求，而不用对每次请求都通过传输速度很慢的网络连接到另一台机子上。表复制不同于数据库复制，表复制的同步机制由数据库管理系统（*DBMS*）自己执行，用户可以不去干预。

表复制有两种类型：同步表复制和异步表复制。同步表复制指当我们对本地表做任何更改时，远程表就会立刻被更新，而异步表复制是按照既定的时间计划来存储本地表中的更改，然后将这些更新复制到远程表中。DROP REPLICATION命令可删除这两种表复制。

*replication_name...*要删除的表复制名。

*table_name .....*从哪个表上删除表复制。

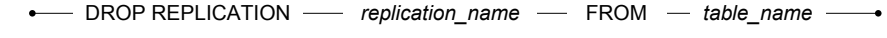


图 3-75 DROP REPLICATION 语法图

➲ **例**

下例将删除表**Employeesinfo**上的表复制**EmpRep**。

```
dmSQL> DROP REPLICATION EmpRep FROM Employeesinfo;
```

3.59 DROP SCHEDULE

DROP SCHEDULE命令可删除远程数据库的表复制计划。删除复制计划前应先删除所有与之相关的异步表复制。只有本地表的所有者、DBA、SYSDBA或SYSADM才可以执行该命令。

用户可使用这条命令来删除异步表复制的复制计划。在删除复制计划之前，应首先删除所有与之相关的异步表复制。这包含复制计划中，将数据复制到远程数据库的所有异步表复制。

remote_database_name...要删除复制计划的远程数据库名。

图 3-76 DROP SCHEDULE 语法图

➲ **例**

下例将在远程数据库**DivOneDb**中删除表复制计划。

```
dmSQL> DROP SCHEDULE FOR REPLICATION TO DivOneDb;
```

3.60 DROP SCHEMA

DROP SCHEMA命令用来删除当前数据库中的指定模式。模式实质上是数据库对象的统称，它包括的数据库对象有表、视图、索引、同义字、触发器、定义域、存储指令和存储过程。这些对象的名称可以和存在于其它模式中的对象重名。访问模式对象时，需要以模式名作为前缀表示它们的名称。

只有模式的创建者或具备DBA权限的用户才可以删除数据库中的模式。

关键字CASCADE/RESTRICT是可选的。他们决定在删除模式时，是一同删除参照它的其他对象，还是请数据库检查是否有参照。如果选择CASCADE关键字，数据库会将所有参照该模式的其他对象一起删除；如果选择RESTRICT关键字，数据库便会检查是否有其他对象参照该模式，该关键字确保只有没有被参照的模式才允许被删除。

schema_name...要删除的模式名。

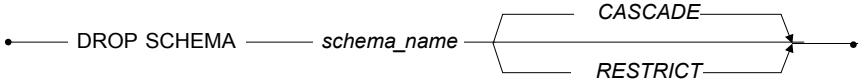


图 3-77 DROP SCHEMA 语法图

3.61 DROP SYNONYM

同义字（*synonym*）就是表或视图的别名，它不占用存储空间，系统只需将它的定义存储在系统目录中即可。您可以在一张表或视图上创建多个同义字，但是每个同义字的名字必须唯一。DROP SYNONYM命令用来删除表或视图的同义字。只有同义字的所有者、DBA、SYSDBA或SYSADM才能执行该命令。

DBMaster通常将表或视图名称结合它的拥有者来作为整个表或视图的完整名称。因此，为了简化表和视图的名称，DBMaster允许使用同义字。

这允许用户直接使用同义字，而不必再加上所有者名。如果一个用户的某个表与同义字同名，那么在DBMaster中，该名称将代表表而非同义字，这时如果要使用该同义字所指向的表就只能键入完整的表名称。当表或视图被删除时，如果选择了CASCADE 关键字，则在它们之上创建的所有同义字都将被删除。

数据库中除了系统表以外的其他所有表的同义字都可以被删除。DBMaster在系统内部管理所有创建在系统表上的同义字，因此不允许用户删除它们。

IF EXISTS：确保在同义字不存在的情况下不报错。

synonym_name ..... 要从数据库中删除的同义字名称。

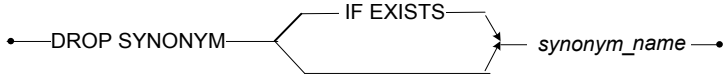


图 3-78 DROP SYNONYM 语法图

➲ **例**

下例将删除**Employeesinfo**表中的同义字**Staff**。

```
dmSQL> DROP SYNONYM Staff;
```

或：

```
dmSQL> DROP SYNONYM IF EXISTS Staff;
```

3.62 DROP TABLE

DROP TABLE命令用于删除一张表。只有表的所有者、DBA、SYSDBA或SYSADM才有权执行该命令。

关键字CASCADE/RESTRICT是可选的。它们决定了在删除表时，是将参照它的对象一并删除还是检查该表是否被参照。如果选择CASCADE关键字，数据库会将所有参照该表的对象一起删除；如果选择RESTRICT关键字，数据库会检查是否有其它对象参照该表，确保只有没被其它对象参照的表才允许被删除。

IF EXISTS .....确保在表不存在的情况下不报错。

table_name .....要从数据库中删除的表名称。

CASCADE.....删除表相关对象，例如，索引、外键、同义字、视图及触发器。

RESTRICT.....确保只能删除没有相关对象（例如索引、外键、同义字、视图及触发器）的表。

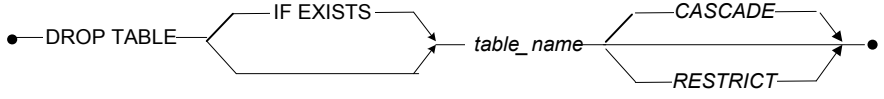


图 3-79 DROP TABLE语法图

➲ **例**

下例删除表**Employeesinfo**。

```
dmSQL> DROP TABLE Employeesinfo;
```

或：

```
dmSQL> DROP TABLE IF EXISTS Employeesinfo;
```

3.63 DROP TABLESPACE

DROP TABLESPACE命令用于删除一个表空间。只有DBA、SYSDBA或SYSADM才能执行该命令。

删除表空间时，DBMaster将同时删除表空间中的所有逻辑文件。您可以使用操作系统命令来手动删除与这些逻辑文件相对应的物理文件以释放磁盘空间。如果表空间中含有表，那么在删除该表空间之前必须先删除表空间中的所有表。

tablespace_name...要从数据库中删除的表空间的名称。

图 3-80 DROP TABLESPACE 语法图

➲ **例**

下例是从数据库中删除一个名为**ts_emp**的表空间。删除这个表空间前将首先删除表空间中的表。

```
dmSQL> DROP TABLESPACE ts_emp;
```

3.64 DROP TEXT INDEX

DROP TEXT INDEX命令用于从数据库中删除某个字段上的特征全文索引或反向（IVF）全文索引。只有表的所有者、DBA、SYSDBA、SYSADM或在指定表上拥有INDEX权限的用户才能执行该命令。

全文索引是一种快速存取记录的机制，这些记录的字段中一般包含一个或多个单词和短语。全文索引包含了在文本字段查询到的所有文本陈述。在这种索引中，数据被重新组织编码，这样的检索方式比直接查询表要快得多。一旦为表创建了全文索引，它的具体操作对数据库的用户而言就是透明的了，只是由数据库管理系统（DBMS）在必要时使用来提高全文查询效率。

text_index_name...要删除的全文索引名。

table_name .....…要从哪个表上删除全文索引。

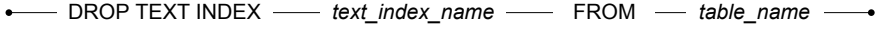


图 3-81 DROP TEXT INDEX语法图

➲ **例**

下例是删除表**Employeesinfo**的全文索引**TxtIdx**。

```
dmSQL> DROP TEXT INDEX TxtIdx FROM Employeesinfo;
```

3.65 DROP TRIGGER

DROP TRIGGER命令用于删除一个触发器。只有表的所有者、DBA、SYSDBA或SYSADM才有权执行该命令。

触发器（*trigger*）是一种数据库服务器机制，只要所设定的事件发生，就导致触发自动执行一些事先定义好的指令。这使得数据库可以执行一些用标准SQL指令所不能执行的复杂操作或非常规操作。因为触发器是由数据库服务器控制的，所以，无论事件是由哪个使用者或哪个应用程序所造成的，触发器的执行都能保证数据的一致性。触发器的操作对用户而言是透明的，因为只要用户或应用程序产生触发事件，DBMaster就自动执行触发器。

如果您删除了触发器参照的表或字段，或更改了表结构，或在更改表结构增加字段时使用了关键字BEFORE和AFTER，那么触发器将变得无效并且将不能再被使用。如果在新增字段时没有使用BEFORE和AFTER关键字，那么这种表修改将不会对触发器产生影响。当一个触发器变得无效时，我们可以将它从数据库中删除。删除表时，在这个表基础上创建的任何触发器都将自动被删除。

IF EXISTS..... 确保在触发器不存在的情况下不报错。

trigger_name..... 要删除的触发器名称。

table_name..... 删除触发器所属的表。

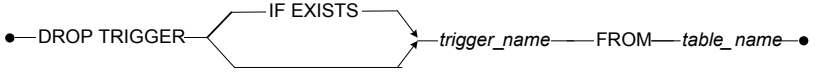


图 3-82 DROP TRIGGER语法图

➲ **例**

下例是删除表**Employeesinfo**上的触发器**Trig_emp**。

```
dmSQL> DROP TRIGGER Trig_emp FROM Employeesinfo;
```

或：

```
dmSQL> DROP TRIGGER IF EXISTS Trig_emp FROM Employeesinfo;
```

3.66 DROP VIEW

DROP VIEW命令用于删除一个视图。只有视图的所有者、DBA、SYSDBA或SYSADM才有权执行该命令。

删除视图将会使基于该视图的其它视图无效，系统视图不能被删除。

关键字CASCADE/RESTRICT是可选的，它们决定在删除视图时，是将参照该视图的其它对象一并删除，还是检查该视图是否存在参照对象。如果选择CASCADE关键字，数据库会将参照该视图的其它对象一并删除；如果选择RESTRICT关键字，数据库会检查该视图是否被任何一个视图定义或同义字参照，如果存在，则该删除操作失败。这样确保了只有没被参照的视图才可以被删除，但是系统视图不能被删除。

IF EXISTS：确保在视图不存在的情况下不报错。

view-name.....要从数据库中删除的视图名。

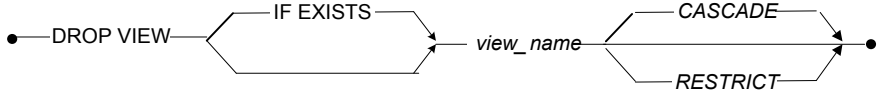


图 3-83 DROP VIEW语法图

➲ **例1**

下例将删除一个名为**SalesStaff**的视图。

```
dmSQL> DROP VIEW SalesStaff;
```

或：

```
dmSQL> DROP VIEW IF EXISTS SalesStaff;
```

➲ **例2**

在下例中，如果存在同义字或者其它视图参照**SalesStaff**，它将不允许被删除。

```
dmSQL> DROP VIEW SalesStaff RESTRICT;
```

3.67 END BACKUP

END BACKUP命令用于结束在线备份时，数据库所处的备份状态。只有DBA、SYSDBA或SYSADM才能执行该命令。

执行在线完整备份时，可以在三种备份模式下启动数据库：NON-BACKUP、BACKUP-DATA或BACKUP-DATA-AND-BLOB，然后使用BEGIN BACKUP命令开始备份。您可以使用操作系统命令或数据库备份程序来备份所有数据文件和BLOB文件。当这些文件都备份完成后，发出END BACKUP DATAFILE命令结束数据备份；接着再使用操作系统命令或数据库备份程序开始备份所有日志文件，备份完成后发出END BACKUP JOURNAL命令来结束备份，并将数据库返回到平常的操作状态。使用在线完整备份可将数据库从上次完整备份时执行END BACKUP DATAFILE命令后的状态，恢复到拷贝当前日志文件后的状态。

```
BEGIN BACKUP; //手动复制所有数据文件copy
END BACKUP DATAFILE; //手动复制所有日志文件
END BACKUP JOURNAL; //完整备份结束
```

执行在线差异备份时，可以在三种备份模式下启动数据库：NON-BACKUP, BACKUP-DATA或BACKUP-DATA-AND-BLOB。用户做差异备份时可不使用手动备份方式。请注意，差异备份是在之前的完整备份基础之上并仅备份差异基础创建时被更改的数据。

在进行在线增量备份或在线增量备份至当前之前，您必须先将数据库的备份模式设为BACKUP-DATA或BACKUP-DATA-AND-BLOB。

您可以随时使用ABORT BACKUP命令退出在线备份。关于该命令的详细信息，可查看ABORT BACKUP命令。执行这条命令后，您将不能使用这个未完成的备份中的文件来恢复数据库。因此您最好将这些备份文件删除，以免这些文件在您恢复数据库时和有用的备份文件混淆。

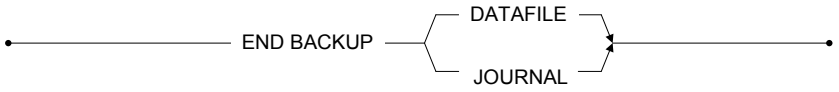


图 3-84 END BACKUP语法图

➲ **例**

下例展示了一个在线完整备份的全过程。首先发出命令**BEGIN BACKUP**通知DBMaster执行完整备份。然后，拷贝所有的数据和BLOB文件到指定位置。拷贝完成后，发出命令**END BACKUP DATAFILE**。之后拷贝所有的日志文件到指定位置。最后发出**END BACKUP JOURNAL**命令。如果备份成功，数据库将返回到平常的操作状态。

```
BEGIN BACKUP
    Copy data and BLOB files to backup location using OS commands
    Change backup mode if desired
    Abort the backup if desired
END BACKUP DATAFILE
    Copy Journal files to backup location using OS commands
    Change the backup mode if desired
    Abort the backup if desired
END BACKUP JOURNAL
```

3.68 EXECUTE COMMAND

EXECUTE COMMAND命令用于执行一个存储命令。使用存储命令能使您更快，更方便地执行常用的SQL数据操纵指令。只有DBA、SYSDBA、SYSADM或具备EXECUTE权限的用户才能执行这条命令。

存储命令是编译过的SQL数据操作指令，它们以可执行文件的格式永久存储在数据库中。用户可以反复执行存储命令，而不用等待DBMaster编译和优化这条指令。存储命令和存储过程有些相似，不同之处在于前者中只包含一条指令，不能包含程序逻辑。

存储命令的SQL命令中可以使用主变量来代表字段。这使得用户在执行命令时可以为字段赋一个实际的值，而并不一定只能在生成存储命令时就给字段赋值。如果您想在存储命令中使用主变量，那么就应该将命令中的数据或者字段值用问号（？）来代替。

执行一个包含主变量的指令时，其中主变量的值可以赋为常量、内置函数的值、NULL关键字、DEFAULT关键字或者其他主变量的值。需要注意的是，只有诸如RAND( )、CURDATE( )以及NOW( )这些不含有自变量的内置函数才能为主变量赋值；而要为主变量赋空值的话，必须要先确保主变量代表的值可以接受空值。执行中输入的参数的数量必须要等于定义存储命令时主变量的个数。

command_name …要执行的存储命令的名称。

value ..... …存储命令中主变量对应的输入参数。

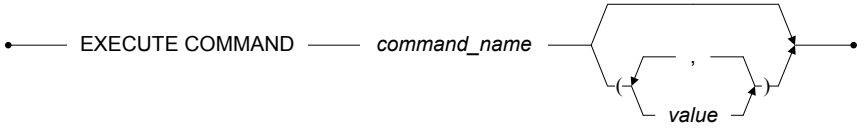


图 3-85 EXECUTE COMMAND 语法图

➲ **例1**

下面的命令是执行一个名为**sc_select**的存储命令，该存储命令没有输入参数。

```
dmSQL> EXECUTE COMMAND sc_select;
```

➲ **例2**

下面的命令是执行一个名为**sc_input**的存储命令，该存储命令中有两个输入参数。

```
dmSQL> EXECUTE COMMAND sc_input(10002, 10006);
```

3.69 GRANT（执行权限）

该命令的作用是将数据库中可执行对象的执行权限赋予一个独立的用户。只有对象的所有者、DBA、SYSDBA或SYSADM才能执行这条命令。

执行（EXECUTE）权限用来控制用户可以执行哪些数据库对象。DBMaster有三种可执行的对象：存储命令（stored command）、存储过程（stored procedure）、项目（project）。

关键字COMMAND表示对象是存储命令。只有具备执行存储命令包含的SQL指令所需的所有对象权限和安全权限，并且具备EXECUTE权限的用户才能执行这条指令。

关键字PROCEDURE表示赋予用户的是存储过程的EXECUTE权限。执行存储过程时，只有EXECUTE权限是必需的。

关键字PROJECT表示被赋予了EXECUTE权限的对象是一个包含一个或多个存储过程的项目。如果给一个项目赋予了EXECUTE权限，那么该项目中的所有存储过程都将拥有EXECUTE权限。

对象的所有者是在数据库中创建这个可执行对象的用户。这个所有者、DBA、SYSDBA或SYSADM自动拥有在这个对象上的EXECUTE权限。如想将EXECUTE权限赋给所有用户实际上就是将EXECUTE权限赋给PUBLIC，这样所有现有的和将来的用户都将获得这个可执行对象上的EXECUTE权限。

executable_name...数据库中可执行对象的名称。

user_name..... …将权限赋予哪个用户。

group_name..... …将权限赋予哪个组。

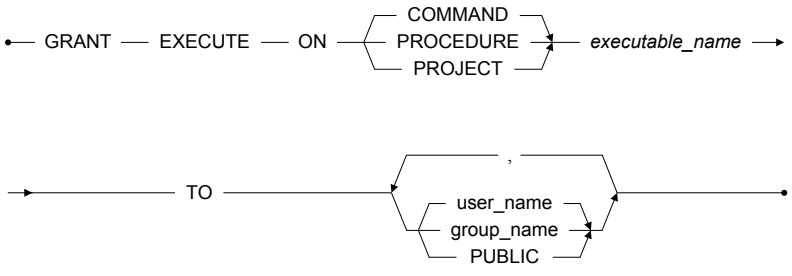


图 3-86 GRANT（执行权限）语法图

➲ **例1**

下例是将存储命令**ListUserTables**上的**EXECUTE**权限赋予用户**Vivian**。

```
dmSQL> GRANT EXECUTE ON COMMAND ListUserTables TO Vivian;
```

➲ **例2**

下例是将存储过程**ShowUsers**的**EXECUTE**权限赋予用户**Jenny**和**John**以及组**Managers**。

```
dmSQL> GRANT EXECUTE ON PROCEDURE ShowUsers TO Jenny, John, Managers;
```

➲ **例3**

下例使用了关键字**PUBLIC**，来将项目中所有存储过程的**EXECUTE**权限赋予数据库中所有的用户。

```
dmSQL> GRANT EXECUTE ON PROJECT InternetFunc TO PUBLIC;
```

3.70 GRANT（对象权限）

该命令的作用是将数据库对象的存取权限赋予指定用户。只有对象的所有者、DBA、SYSDBA或SYSADM才可以执行该指令。

对象权限用来控制用户可存取哪些数据库对象以及可执行的操作。DBMaster中有七种对象权限：SELECT、INSERT、DELETE、UPDATE、INDEX、ALTER以及REFERENCE。关键字ALL和ALL PRIVILEGES用来将一个对象上的所有权限赋给用户。

- *SELECT*权限指可以在数据库对象中查询数据。该权限可应用于整个对象，但不能应用于一个特定字段。
- *INSERT*权限指可以在数据库对象中插入新数据。该权限可应用于一个特定字段。
- *DELETE*权限指可以从数据库对象中删除数据。该权限可应用于整个对象，但不能应用于一个特定字段。
- *UPDATE*权限指可以更新数据库对象中的数据。该权限可以应用于一个特定字段。
- *INDEX*权限指可以在数据库对象上创建一个索引。该权限可应用于整个对象，但不能应用于一个特定字段。
- *ALTER*权限指可以修改一个数据库对象的结构。该权限可应用于整个对象，但不能应用于一个特定字段。
- *REFERENCE*权限指可以在数据库对象上创建参照完整性约束，如创建外键。该权限可应用于一个特定的字段。

对象的拥有者是指创建该对象的用户。对象的所有者、DBA、SYSDBA或SYSADM可自动获得该对象的所有权限。系统表的所有者是虚拟用户SYSADM。所有用户，包括SYSADM都具备系统命令表上的SELECT权限，系统命令表上的其它权限是不能另外赋给其他用户的。

如果给用户赋予了特定字段的权限，那么您就不能同时给他赋予整个数据库对象的权限，这两种权限必须分两次赋予。一次对用户赋予特定字段上的权限，一次对用户赋予这张表上的权限。将对象权限赋予PUBLIC

的意思是将对象权限赋予数据库中的所有用户，这样，数据库中的现有或将来用户都将获得这个对象权限。

column_name ....赋予哪个字段上的对象权限。

table_name .....赋予哪个表上的对象权限。

user_name .....将对象权限赋予哪个用户。

group_name.....将对象权限赋予哪个组。

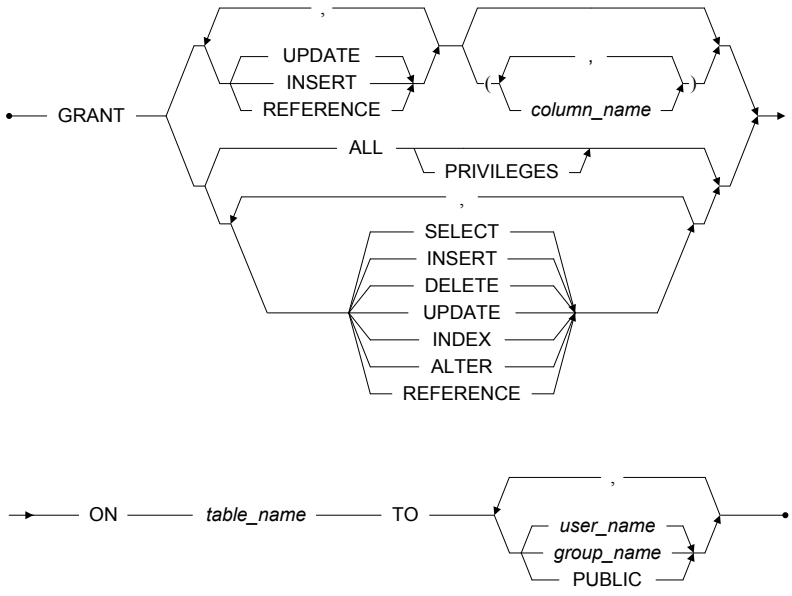


图 3-87 GRANT（对象权限）语法图

➲ **例1**

下例是将表**Checks**上的SELECT、NSERT和UPDATE对象权限赋予用户**Vivian**。

```
dmSQL> GRANT SELECT, INSERT, UPDATE ON Checks TO Vivian;
```

➲ **例2**

下例将表**Checks**中**Amount**和**PayDate**字段的INSERT、UPDATE和REFERENCE权限赋予用户**Jenny**。

```
dmSQL> GRANT INSERT, UPDATE, REFERENCE (Amount, PayDate) ON Checks TO Jenny;
```

➲ **例3**

下例将表**Checks**上的所有对象权限赋予用户**John**。

```
dmSQL> GRANT ALL ON Checks TO John;
```

3.71 GRANT（安全权限）

这个GRANT命令可用来创建一个新用户或更改现有用户的安全权限。只有SYSDBA或SYSADM才能执行该命令。新建一个数据库时，DBMaster将自动生成默认用户SYSADM，此时该用户是没有密码的。您最好在创建数据库后立即修改SYSADM的密码以防止非授权用户的访问。在SYSADM将安全权限赋予其他用户之前，SYSADM是数据库中唯一的授权用户。

SYSADM可将CONNECT、RESOURCE、DBA、SYSDBA和ACCESS这四个安全权限授予其他用户。当数据库新增加一个用户时，该用户将自动获得CONNECT权限。一旦创建了一个新用户，SYSADM就可以给这个用户授予更高的安全权限。高权限用户拥有低权限用户的所有权限。只有SYSADM、SYSDBA用户才可以将安全权限授予其他用户。SYSADM拥有SYSDBA权限用户的所有权限，且只有SYSADM才能授予用户SYSDBA权限。

用户在连接数据库之前，必须要具备CONNECT安全权限。一旦用户被授予CONNECT权限，那么他就成为数据库的合法用户。对所有用户而言，要给他授予其他安全权限，必须先对他授予CONNECT权。具备CONNECT权限的用户可以在数据库中创建临时表，或在允许他访问的数据上执行查询操作。

具备RESOURCE安全权限的用户可以创建、修改、删除表，定义域和索引。作为对象的所有者，如果拥有RESOURCE权限，那么他就有权将自己所有的对象的权限授予其他用户或取消其他用户的对象权限，也可以在自己所有的对象上创建同义字和视图。

具有DBA权限的用户除了可以执行RESOURCE权限的动作外，还可以创建表空间和文件。具备DBA权限的用户可以将除系统对象以外的其他用户所有对象的权限授予另外的用户或者取消对象权限。

具有SYSDBA权限的用户除了可以执行DBA权限的动作外，还可以将CONNECT、RESOURCE、DBA和ACCESS权限授予其它用户，且可以授予、更改或回收具有DBA权限用户所拥有的对象权限。此外，还可以更改除SYSADM和其它SYSDBA权限用户之外的用户密码。

用户名的最大长度不能超过128个字符，密码最大长度不能超过16个字符，可以包含字母、数字、下划线以及$和#符号，但不能以数字开头。

ACCESS/ALLOW权限允许用户通过指定的IP地址连接数据库。这就有效保护了数据库，避免恶意的连接。IP地址是标准的网络协议格式，可以包含数字和“*”号。

BLOCK权限禁止用户通过指定的IP地址连接数据库。同样有效保护了数据库，避免恶意的连接。IP地址是标准的网络协议格式，可以包含数字和“*”号。

user_name..... 将安全权限授予哪个用户。

password ..... 用户密码。

ip_address ..... 授予安全权限的用户地址。

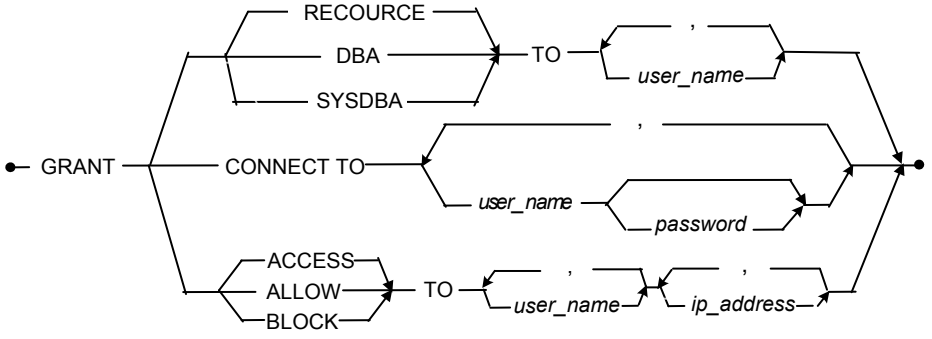


图3-88 GRANT（安全权限）语法图

➲ **例1**

下例是将**CONNECT**权限授予用户**vivian**和**jenny**。

```
dmSQL> GRANT CONNECT TO vivian, jenny;
```

➲ **例2**

下例将**CONNECT**权限授予用户**vivian**和**jenny**，这两个用户的密码分别为**shuka828**和**grala833**。

```
dmSQL> GRANT CONNECT TO vivian shuka828, jenny grala833;
```

➲ **例3**

下例将**RESOURCE**权限授予用户**vivian**和**jenny**。

```
dmSQL> GRANT RESOURCE TO vivian, jenny;
```

➲ **例4**

下例将**DBA**权限授予用户**vivian**和**jenny**。

```
dmSQL> GRANT DBA TO vivian, jenny;
```

➲ **例5**

下例将**ACCESS**权限授予用户**vivian**和**jenny**，允许其访问地址分别是**192.4.55.3**和**219.3.44.***。

```
dmSQL> GRANT ACCESS TO vivian,jenny '192.4.55.3','219.3.44.*';
```

3.72 INSERT

INSERT命令用来在表中插入新记录，但不能在系统表中插入记录。只有表的所有者、DBA、SYSADM、SYSDBA或在整张表或指定字段上拥有INSERT权限的用户才能执行该命令。

您可以使用该命令在表中插入一条记录，插入的值由关键字VALUES来指定。插入值可以是常量、内置函数的值或ODBC API程序中绑定的变量值。同样，您还可以使用该命令在表中插入一组记录，插入值可以是在其它表上使用SELECT语句所查询到的结果。查询所得记录中的字段数据类型必须和插入表的字段类型一致。

使用INSERT命令时，需要指定需要插入值的字段，并按一定顺序排列这些字段。如果INSERT命令中没有指定字段，就代表插入所有字段，且顺序是创建字段时的顺序。在这种情况下，必须为每个字段插入值，即使这个值是空值。如果插入的值与字段类型不匹配，DBMaster会自动将这个值转换为适当的数据类型。如果在某个字段上没有插入值，那么系统就把这个字段上的默认值插入该字段。

从表利用外键和主表连接，因此在从表中插入数据时，需要遵守参照完整性约束。插入值如果是非空的话，您将要在从表中插入的值必须是在主键中存在的值，因此在从表中插入值之前，应首先在主键中插入相应的值。

如果想插入一个带有单引号的字符串，您需要用两个连续的单引号来代替字符串中的那个单引号。除非字符串中刚好有偶数个单引号，不然DBMaster将一直等待另一个单引号来结束这个字符串。如果想在字段中插入默认值，您只需在插入时，让这个字段为空或使用关键字DEFAULT即可。

OR REPLACE：OR REPLACE选项是可选的。当两条记录的某些字段值相同时，该选项用于确保DBMaster使用新记录来替换旧记录。也就是说，若用户插入该表的记录已经存在于表中（根据主键和唯一性索引判断），那么旧记录将会从表中删除，然后新记录插入到该表中。否则，新记录将直接插入到该表中。

若要使用OR REPLACE选项，用户需同时拥有INSERT权限和DELETE权限。使用OR REPLACE选项的INSERT语句将会返回一个计数，该计数用于表示受影响记录的行数，是删除行数和添加行数的总和。

请注意，OR REPLACE选项仅当该表拥有主键或唯一性索引时才有意义。主键和唯一性索引均用于确认新记录是否与旧记录重复，如果二者不存在，INSERT语句是否使用OR REPLACE选项将无任何区别，因此在执行INSERT语句时，新记录将会被直接插入到该表中，这将导致该表中存在重复的记录。如果将要插入的记录是其它表的查询结果且包含大量数据，则不推荐使用该选项，因为这将降低插入效率。

此外，我们也不推荐在包含大量数据的字段上创建唯一性索引，因为这样不仅不利于快速查询，而且当用户执行INSERT OR REPLACE语句插入数据时可能会返回错误8332：“表达式或谓词需要的内存太大”。

table_name .....要插入新记录的表名。

column_name ....要插入值的字段名。

literal.....插入值。

constant .....插入的常量。

bind_variable.....插入的绑定在程序中的变量。

select_statement...SELECT语句。

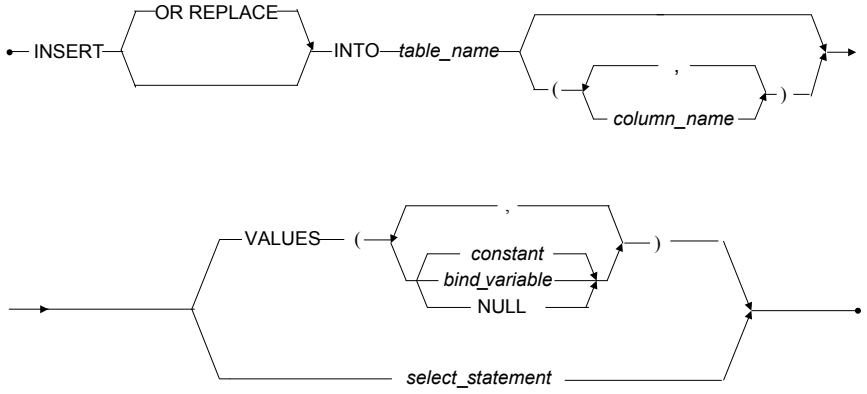


图 3-89 INSERT语法图

➲ **例1**

下例是在表**Employeesinfo**中插入一条新记录。

```
dmSQL> INSERT INTO Employeesinfo VALUES (1234, 'John', '01/01/1998', 2500);
```

➲ **例2**

下例是在表**Employeesinfo**的字段**Emp_ID**, **FName**和**HireDate**中插入新值。

```
dmSQL> INSERT INTO Employeesinfo (Emp_ID, FName, HireDate)
     VALUES (1234, 'John', '01/01/1998');
```

➲ **例3**

下例是在表**Employeesinfo**的字段**Emp_ID**, **FName**和**HireDate**中插入一组新记录。插入的值是在表**TempStaff**的相应字段上查询**Emp_ID**字段的值大于**10567**的结果。

```
dmSQL> INSERT INTO Employeesinfo (Emp_ID, FName, HireDate)
     SELECT Emp_ID, FName, HireDate FROM TempStaff WHERE Emp_ID > 10567;
```

➲ **例4**

下例是在表**TB_TMP**的**CHAR**字段中插入一个包含一个单引号的字符串，而在其它两个字段上使用关键字**DEFAULT**插入字段的默认值。

```
dmSQL> INSERT INTO TB_TMP VALUES ('Joe''s Diner', DEFAULT, DEFAULT);
```

➲ **例 5**

下例是在表**Employeesinfo**中插入一条新纪录，该表的主键为**Emp_ID**，且字段**FName**上有一个名为**idx2**的唯一性索引。

```
dmSQL> INSERT INTO Employeesinfo VALUES (1,'BB', '01/01/1986');
```

或

```
dmSQL> INSERT OR REPLACE INTO Employeesinfo VALUES (1,'BB', '01/01/1986');
```

3.73 KILL CONNECTION

KILL CONNECTION命令的作用是结束用户和数据库的连接，只有DBA、SYSDBA或SYSADM才有权执行该命令。

执行该命令后将释放这个用户加在数据库上的所有锁资源。当一个用户占有的资源被另一个用户要求来执行具有更高优先级的操作，或数据库管理员要关闭数据库，而还有些用户仍连接在数据库上时就可以使用这条命令。

connection_ID.. ...要终止的连接ID。

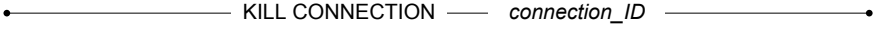


图 3-90 KILL CONNECTION语法图

➲ **例**

下例是终止ID号为**12345**的用户连接。

```
dmSQL> KILL CONNECTION 12345;
```

3.74 LOAD STATISTICS

LOAD STATISTICS命令可用来将包含DBMaster数据库统计数据的文本文件中的统计信息载入到数据库中。您可以用UNLOAD STATISTICS命令来为数据库生成一个统计文件。这个文件可以用任何一种ASCII文本编辑器来编辑，并且可以进行修改以便为测试或其它目的提供统计数据。只有DBA、SYSDBA或SYSADM才可以执行该命令。

file_name .....要调用的、包含统计数据的文件名。

图 3-91 LOAD STATISTICS语法图

➲ **例**

下例是将统计文件**stat.dat**载入到数据库中。

```
dmSQL> LOAD STATISTICS FROM stat.dat;
```

3.75 LOCK TABLE

LOCK TABLE命令用于控制其他用户对表的访问。只有表的所有者、DBA、SYSDBA、SYSADM或拥有SELECT（在SHARE模式下锁定表）、UPDATE或DELETE权限（在EXCLUSIVE模式下锁定表）的用户才有权执行该命令。

这条命令在共享（SHARE）或互斥（EXCLUSIVE）模式下锁定表，以此来控制用户对表的访问。SHARE模式允许多个用户同时读取表，但不能同时写表。如果表处于SHARE模式下，其他用户就不能在表中插入、更新或删除记录。EXCLUSIVE模式不允许多个用户同时读写表。如果表处于EXCLUSIVE模式下，那么其他用户就不能在表中查询、插入、更新或删除记录。

使用该命令可减少数据库操作中要求的锁数量。如果默认的锁定级别是页或行，那么使用该命令可将锁定级别更改为表锁定，这样就可以避免系统中存在过多较低级别的锁定，因为较高级别的锁定包含了较低级别对象的锁定。一般来说，用户不必去做这些工作，因为当系统中有太多的锁定时，DBMaster会自动更新表的锁定级别。

关键字WAIT/NO WAIT是可选的。这两个关键字的意思是请求的锁不能立即得到时，DBMaster是否要等待。如果选择NO WAIT，那么DBMaster将不等待，而是返回一个错误信息表明锁定要求不能被满足。DBMaster等待的时间是由**dmconfig.ini**文件中的关键字**DB_LTimo**决定的。如果没有指定是选择WAIT还是NO WAIT，系统默认是***WAIT***。

table_name ..... 要修改哪个表的锁定设置。

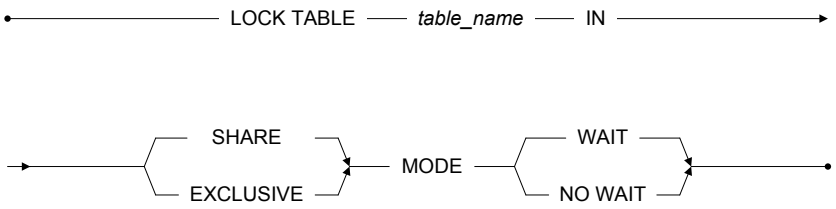


图 3-92 LOCK TABLE语法图

➲ **例1**

下例在共享（**SHARE**）模式下锁定表**Employeesinfo**，同时选择**WAIT**选项。

```
dmSQL> LOCK TABLE Employeesinfo IN SHARE MODE WAIT;
```

➲ **例2**

下例是在互斥（**EXCLUSIVE**）模式下锁定表**Employeesinfo**，同时选择**NO WAIT**选项。

```
dmSQL> LOCK TABLE Employeesinfo IN EXCLUSIVE MODE NO WAIT;
```

3.76 REBUILD COMMAND

REBUILD COMMAND命令可用来重建存储命令。只有DBA、SYSDBA和SYSADM可以执行该命令。

重建存储命令可避免该存储命令的执行效率降低。例如，若用户在一个拥有少量记录的表上创建存储命令，随着表记录的增加，该存储命令的执行效率将会变得更加糟糕。

用户可使用REBUILD COMMAND语法重建存储命令，或在更新统计值时数据库自动重建存储命令。

command_name .....要重建的存储命令名。

图 3-93 REBUILD COMMAND 语法图

➲ **例**

下例将重建存储命令**sc_select**。

```
dmSQL> REBUILD COMMAND sc_select;
```

3.77 REBUILD INDEX

REBUILD INDEX命令可用来重建一张表上的索引。只有表的所有者、DBA、SYSDBA、SYSADM或在指定表上拥有INDEX权限的用户才可以执行该命令。

索引（*index*）是一种在表中通过一个或多个字段（例如键）值来快速查找指定记录的机制。索引包含的数据和键字段中的数据一样，但是这些数据经过组织和排序使得检索速度得到了提高。一旦在表上创建索引，它的操作对数据库的用户而言就是透明的了，数据库管理系统（DBMS）在必要时就将使用索引来提高查询速度。

重建一个数据密集，没有碎片的索引，这样将有助于提高效率。

index_name .....要重建的索引名。

table_name .....要重建索引的表名。

图 3-94 REBUILD INDEX语法图

➲ **例**

下例将重建表**Employeesinfo**的索引**NameIndex**。

```
dmSQL> REBUILD INDEX NameIndex FOR Employeesinfo;
```

3.78 REBUILD INDEX IN ANOTHER TABLESPACE

REBUILD INDEX命令可用来在另外一个表空间中重建表的索引，且该表原索引将自动删除。只有表的所有者、DBA、SYSDBA或在指定表上同时拥有ALTER和INDEX权限的用户才可以执行该命令。

注意 *不能在临时表空间中重建固定表的索引。*

注意 *临时表的索引只能重建在临时表空间中。*

注意 *系统表的索引只能重建在系统表空间中。*

index_name ..... 要重建的索引名。

table_name ..... 重建索引所属的表名。

tablespace_name..... 重建索引所属的表空间名。

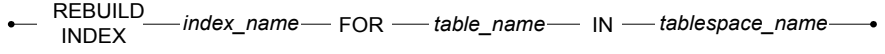


图 3-95 REBUILD INDEX IN ANOTHER TABLESPACE语法图

➲ **例**

下例将在表空间**ts_new**中为存储在表空间**ts_mode**里的表**Employeesinfo**重建其索引**NameIndex**。

```
dmSQL> REBUILD INDEX NameIndex FOR Employeesinfo IN ts_new;
```

3.79 REBUILD TEXT INDEX

REBUILD TEXT INDEX命令可为表重建反向（IVF）全文索引或特征（signature）全文索引。重建全文索引的作用是将新插入到数据库的数据包含进全文索引中。只有表的所有者、DBA、SYSDBA、SYSADM或在指定表上拥有INDXE权限的用户才能执行这条命令。

全文索引是一种快速存取记录的机制，这些记录的字段一般包含一个或多个单词或短语。全文索引包含了在文本字段查询到的所有文本的陈述。在这种索引中，数据被重新组织编码，这样的检索方式比直接查询表要快得多。全文索引的具体操作对数据库的用户而言就是透明的了，只是由数据库管理系统（DBMS）在必要时使用来提高全文查询的效率。

当向表中载入数据后，DBMaster并不能自动更新表上的全文索引，因此建议您在为表创建索引之前，尽可能地载入所有数据。全文索引创建后，新加入的记录即使符合查询条件，也无法用match运算找到这些记录。为了能在查询时搜索到这些新增加的记录，您必须使用REBUILD TEXT INDEX命令来重建全文索引。

增加（incremental）选项是REBUILD TEXT INDEX命令的默认设置。该选项的作用是为在全文索引创建后才插入到表的那部分文本创建部分索引，且这些索引附加在当前索引之后，这样在做全文检索时就能搜索到这些文本。完全（full）选项的作用是删除原来的全文索引，在新的全文检索基础上重建新的全文索引。

text_index_name...要重建的全文索引名。

table_name .....…要重建全文索引的表名。

incremental .....…创建部分索引附加在当前索引之后。

full.....…删除当前索引，创建一个新的全文索引。

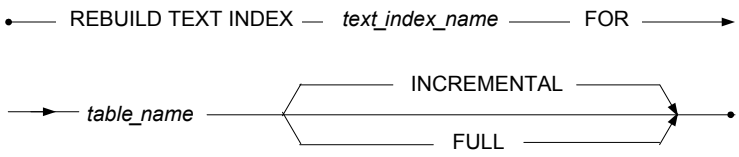


图 3-96 REBUILD TEXT INDEX语法图

➲ **例**

下例是重建表**Employeesinfo**的全文索引**Txtldx**。

```
dmSQL> REBUILD TEXT INDEX TxtIdx FOR Employeesinfo;
```

3.80 REMOVE FROM GROUP

REMOVE FROM GROUP命令的作用是将某个用户移出现有的组。执行该命令后，这个用户将失去授予该组的所有对象权限，但仍将保留被直接授予的任何权限。只有SYSADM、SYSDBA或DBA才可以执行这条指令。

当数据库中用户数很多时，组可以简化对象权限的管理。您可以将用户或组集合在一个组里。任何赋予群组的对象权限都将自动赋予群组里的所有成员。

如果您给组授予了对象权限后再将某个用户加入该组，那么这个用户除了拥有本身被直接授予的对象权限外，还将获得那些授予给这个组的所有对象权限。

您也可以将一个组移出另一个组。只要被移出的组成员不是您正在使用的组中成员，您就可以执行该命令，执行时只需将组名替代用户名即可。用户名和组名最大长度不能超过128个字符，可以包含字母、数字、下划线以及$和#符号，但不能以数字开头。

user_name.....将被移出组的用户名。

group_name.....用户将从哪个组中移出。

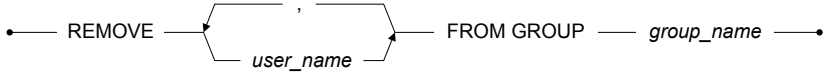


图 3-97 REMOVE FROM GROUP语法图

➲ **例1**

下例是将用户**Vivian**移出组**SalesStaff**。

```
dmSQL> REMOVE Vivian FROM GROUP SalesStaff;
```

➲ **例2**

下例是将组**NYSalesStaff**移出组**SalesStaff**。

```
dmSQL> REMOVE NYSalesStaff FROM GROUP SalesStaff;
```

3.81 REMOVE TRACE

REMOVE TRACE命令可删除记录了详细OLD\NEW数据的单个表中的跟踪功能。只有表所有者，具备DBA、SYSDBA、 SYSADM权限的用户才可以执行此命令。

table_name.....现有的单个表名。

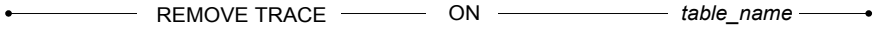


图3-98 REMOVE TRACE语法图

3.82 RESUME SCHEDULE

RESUME SCHEDULE命令用来恢复被暂停的异步表复制的复制计划。只有本地表的所有者、DBA、SYSDBA或SYSADM才可以执行这条命令。

remote_database_name…需要恢复复制计划的远程数据库的名称。

图 3-99 RESUME SCHEDULE语法图

➲ 例

下例将恢复远程数据库**DivOneDb**的复制计划。

```
dmSQL> RESUME SCHEDULE FOR REPLICATION TO DivOneDb;
```

3.83 REVOKE（执行权限）

该REVOKE命令用于取消授予用户或组在数据库可执行对象上的执行权限。只有对象的所有者、DBA、SYSDBA或SYSADM才能执行这条命令。

执行权限可用来控制一个用户可以执行哪些数据库的可执行对象。DBMaster的可执行对象包括存储命令、存储过程和项目(project)。

关键字COMMAND表示取消存储命令上的EXECUTE权限。只有具备执行存储命令包含的SQL指令所需的所有对象权限和安全权限，并且具备EXECUTE权限的用户才能执行这条指令。

关键字PROCEDURE表示取消存储过程上的EXECUTE权限。执行存储过程时，只有EXECUTE权限是必需的。

关键字PROJECT表示取消EXECUTE权限的对象是一个包含一个或多个存储过程的项目。如果取消一个项目上的EXECUTE权限，那么该项目中所有存储过程上的EXECUTE权限都将被取消。

只有对象的所有者、DBA、SYSDBA或SYSADM才能自动获得对象的EXECUTE权限。将PUBLIC权限取消的意思就是将所有用户在这个对象上的EXECUTE权限取消，这样所有现有用户都将失去在这个可执行对象上的EXECUTE权限。

executable_name...数据库中可执行对象的名称。

user_name..... …要取消其执行权限的用户名称。

group_name..... …要取消其执行权限的组的名称。

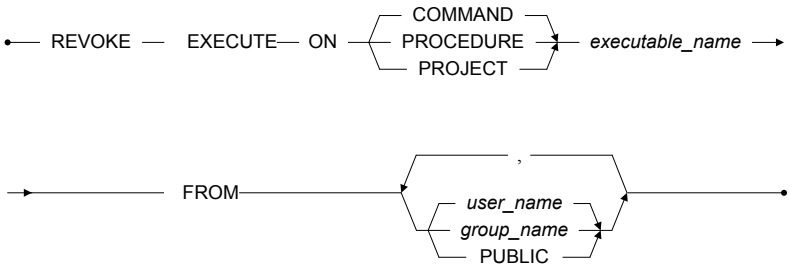


图 3-100 REVOKE（执行权限）语法图

➲ **例1**

下例将取消用户**Vivian**在存储命令**ListUserTables**上的EXECUTE权限。

```
dmSQL> REVOKE EXECUTE ON COMMAND ListUserTables FROM Vivian;
```

➲ **例2**

下例将取消用户**Jenny**和**John**以及组Managers在存储过程**ShowUsers**上的EXECUTE权限。

```
dmSQL> REVOKE EXECUTE ON PROCEDURE ShowUsers FROM Jenny, John, Managers;
```

➲ **例3**

下例中使用关键字**PUBLIC**，来取消所有用户在**InternetFunc**中所有存储过程上的**EXECUTE**权限。

```
dmSQL> REVOKE EXECUTE ON PROJECT InternetFunc FROM PUBLIC;
```

3.84 REVOKE（对象权限）

该REVOKE命令可用来取消用户和组对数据库对象的存取权限。只有对象的所有者、DBA、SYSDBA或SYSADM才可以执行这条指令。

对象权限可用来控制一个用户可访问哪些数据库对象以及在这些对象上可执行的操作。DBMaster中有七种对象权限：SELECT、INSERT、DELETE、UPDATE、INDEX、ALTER以及REFERENCE。关键字ALL和ALL PRIVILEGES可用来取消用户在对象上的所有对象权限。

- *SELECT*权限指用户可以在数据库对象中查询数据。这个权限可应用于整个对象，但不能应用于一个特定字段。
- *INSERT*权限指可以在数据库对象中插入新数据。这个权限可应用于一个特定字段。
- *DELETE*权限指可以从数据库对象中删除数据。这个权限可应用于整个对象，但不能应用于一个特定字段。
- *UPDATE*权限指可以更新数据库对象中的数据。这个权限可以应用于一个特定字段。
- *INDEX*权限指可以在数据库对象上创建一个索引。这个权限可应用于整个对象，但不能应用于一个特定字段。
- *ALTER*权限指可以修改一个数据库对象的结构。这个权限可以应用于整个对象，但不能应用于一个特定字段。
- *REFERENCE*权限指可以在数据库对象上创建参照完整性约束，例如创建外键。这个权限可以应用于一个特定的字段。

系统表的所有者是虚拟用户SYSADM。所有用户，包括SYSADM都具备在系统命令表上的SELECT权限，系统命令表上的这个对象权限是不能被取消的。

要想取消用户在某个字段上和整个数据库上的权限，必须使用两次REVOKE命令，一次是取消用户在指定字段上的权限，一次是取消用户在整张表上的权限。取消PUBLIC权限也就取消了所有用户的对象权限，这样，数据库中的现有用户都将失去这个对象上的权限。

column_name ....取消用户在哪个字段上的权限。

table_name .....取消用户在哪个表上的权限。

user_name .....要被取消对象权限的用户名。

group_name.....要被取消对象权限的组名。

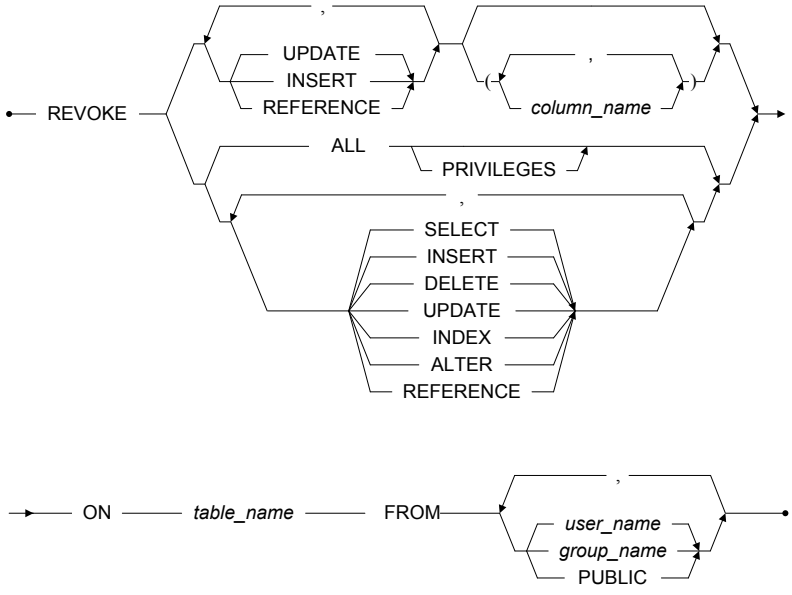


图 3-101 REVOKE（对象权限）语法图

➲ **例1**

下例将取消用户**Vivian**在表**Checks**上的**SELECT**、**INSERT**和**UPDATE**对象权限。

```
dmSQL> REVOKE SELECT, INSERT, UPDATE ON Checks FROM Vivian;
```

➲ **例2**

下例将取消用户**Jenny**在表**Checks**中的字段**Amount**和**PayDate**上的**INSERT, PDATE**和**REFERENCE**三个对象权限。

```
dmSQL> REVOKE INSERT, UPDATE, REFERENCE (Amount, PayDate) ON Checks FROM Jenny;
```

➲ **例3**

下例将取消用户**John**在表**Checks**上的所有授权。

```
dmSQL> REVOKE ALL ON Checks FROM John;
```

3.85 REVOKE (安全权限)

这个REVOKE命令的作用是从数据库中删除某个用户或更改某个用户的安全权限，只有SYSADM或具备SYSDBA权限的用户才可以执行这条命令。

SYSADM可以取消用户的DBA、SYSDBA、 ESOURCE、CONNECT以及ACCESS权限。取消某个用户的CONNECT权限实际上就是将这个用户的ID从数据库中删除。一旦某个用户的ID被删除，那么这个用户将不能再连接数据库。您取消一个较低级别的权限时并不会取消更高级别的权限，但取消用户的CONNECT权限时将相应取消该用户的所有权限。

SYSDBA权限具有所有DBA的权限，不同的是SYSDBA权限还可以使用REVOKE命令回收除SYSADM、SYSDBA外其它用户的DBA、RESOURCE、CONNECT、ACCESS权限。若用户的SYSDBA权限被回收，则该用户仍然拥有DBA权限。

DBA权限具有所有RESOURCE的权限，不同的是DBA权限还允许用户创建表空间和文件。具备DBA权限的用户可以将除了系统对象以外的其他用户对象权限授予另外的用户或者取消对象权限。

具备RESOURCE权限的用户可以创建、修改、删除表、定义域和索引。作为对象的所有者，如果拥有RESOURCE权限，那么他就有权将自己所属的对象权限授予其他用户或取消其他用户的对象权限，也可以在自己所有的对象上创建同义字和视图。

用户在连接数据库之前，必须要具备CONNECT安全权限。一旦用户被授予CONNECT权限，那么他就成为数据库的合法用户。对于用户而言，要给他授予其它安全权限，必须首先对他授予CONNECT权限。具备CONNECT权限的用户可以在数据库中创建临时表或在允许他存取的数据上执行查询操作。

ACCESS/ALLOW权限允许用户通过指定的IP地址连接数据库。这就有效保护了数据库，避免恶意的连接。IP地址是标准的网络协议格式，可以包含数字和“*”号。

BLOCK权限禁止用户通过指定的IP地址连接数据库。同样也有效保护了数据库，避免恶意的连接。IP地址是标准的网络协议格式，可以包含数字和“*”号。

若要回收用户指定的所有IP检查规则，可使用REVOKE ALLOW/BLOCK FROM user_name ALL命令，其中ALL表示所有IP地址。

若REVOKE命令用于回收用户的RESOURCE、DBA、SYSDBA权限，则在用户下次连接数据库时生效。

user_name..... 要被取消安全权限的用户名。

ip_address ..... 要被取消安全权限的用户地址。

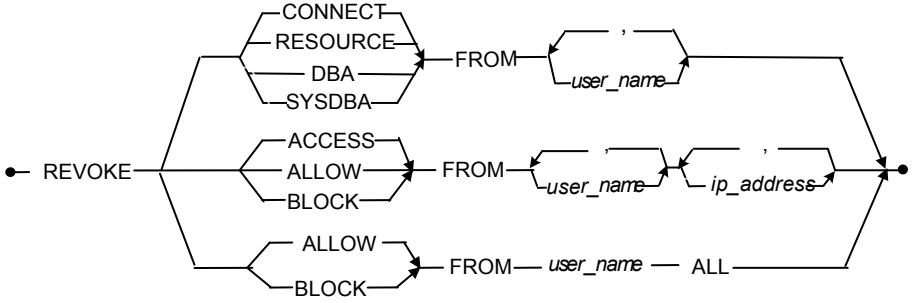


图 3-102 REVOKE (安全权限)语法图

➲ **例1**

下例将取消用户**vivian**和**jenny**的**DBA**权限。

```
dmSQL> REVOKE DBA FROM vivian, jenny;
```

➲ **例2**

下例将取消用户**vivian**和**jenny**的**RESOURCE**权限。

```
dmSQL> REVOKE RESOURCE FROM vivian, jenny;
```

➲ **例3**

下例将取消用户**vivian**和**jenny**的**CONNECT**权限，也就是取消这两个用户的所有权限，并将这两个用户从数据库中删除。

```
dmSQL> REVOKE CONNECT FROM vivian, jenny;
```

➲ **例4**

下例将取消用户**vivian**和**jenny**通过地址**192.55.3.4**和**219.5.3.***访问的**ACCESS** 权限。

```
dmSQL> REVOKE ACCESS FROM Vivian,jenny '192.55.3.4','219.5.3.*';
```

3.86 ROLLBACK

ROLLBACK命令可将事务回滚至事务开始前的状态，或回滚至预先定义的存储点。任何具备CONNECT或更高权限的用户都可以执行该指令。

您可以使用ROLLBACK命令来回滚当前事务中的命令对数据库的更改，并用这个命令来释放事务所占用的锁资源。不过，如果数据库处于AUTOCOMMIT模式下，那么这条命令就不起作用。

同样，您也可以使用这条命令来回滚当前事务对数据库的部分更改。存储点之后执行的命令都可以被回滚，但不能回滚存储点之前执行的命令。回滚到存储点时，事务仍处于激活状态，并且不释放所占用的锁资源。

savepoint_name ...恢复到哪个存储点。

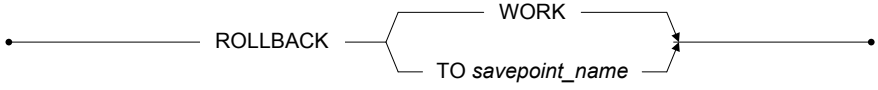


图 3-103 ROLLBACK语法图

➲ **例1**

下例回滚当前事务，也就是终止当前事务。这种情况下，系统将释放这个事务所占的锁资源。

```
dmSQL> ROLLBACK WORK;
```

➲ **例2**

下例将回滚存储点**SavePoint1**之后执行的所有命令，但保持存储点之前执行的命令。事务继续处于激活状态，同时它所占的锁资源也不会被释放。

```
dmSQL> ROLLBACK TO SavePoint1;
```

3.87 SAVEPOINT

SAVEPOINT命令可为当前事务设置存储点，并给该存储点命名。具备CONNECT或更高权限的用户都可以执行这条指令。

SAVEPOINT命令可以和ROLLBACK一起使用，用来回滚当前事务中的一部分指令。执行时只需在ROLLBACK命令中指出存储点的名称，DBMaster就可以回滚存储点后执行的所有指令。回滚后，事务仍处于激活状态，并不释放它所占用的锁资源。

如果在ROLLBACK中指定的存储点不存在，那么DBMaster将回滚这个事务，并且返回一条错误信息。这时，系统将终止这个事务，并释放它所占的所有锁资源。如果在同一事务中为两个不同的存储点设置同一个名称，那么系统将取消第一个存储点，而将这个名称指派给第二个存储点。

savepoint_name....存储点的名称。

图 3-104 SAVEPOINT语法图

➲ **例**

下例是在当前事务中设置了一个名为**SavePoint1**的存储点。

```
dmSQL> SAVEPOINT SavePoint1;
```

3.88 SELECT

您可以通过SELECT命令来查询、检索数据库中的数据，并将查询的结果显示出来。只有表的所有者、DBA、SYSDBA、SYSADM或在指定表上拥有SELECT权限的用户才能执行这条命令。

SELECT命令的执行结果是一组记录，也就是满足指定条件的结果集。执行命令时需要指定要查询的表名称或视图名称、出现在结果集中的数据所需要满足的条件、输出结果数据的顺序。SELECT语句还可能是若干条单一指令的结合（UNION）。

select ..... SELECT子句列出了要检索的字段。

from..... FROM子句列出要检索的字段所属的表。

where ..... WHERE子句指定查询的返回值所需要满足的条件。

group by..... GROUP BY子句按某个字段分组，以计算每个组的聚集。

having ..... HAVING子句指定可以显示的聚集所需要满足的条件。

order by..... ORDER BY子句指定字段的排序。

for browse ..... FOR BROWSE子句的意思是查询数据时只需获得共享锁。

into ..... INTO子句用以指定查询结果将被插入到哪个表中。

limit..... LIMIT子句用来限制SELECT语句返回的记录数。返回的记录从忽略n行后开始计算。

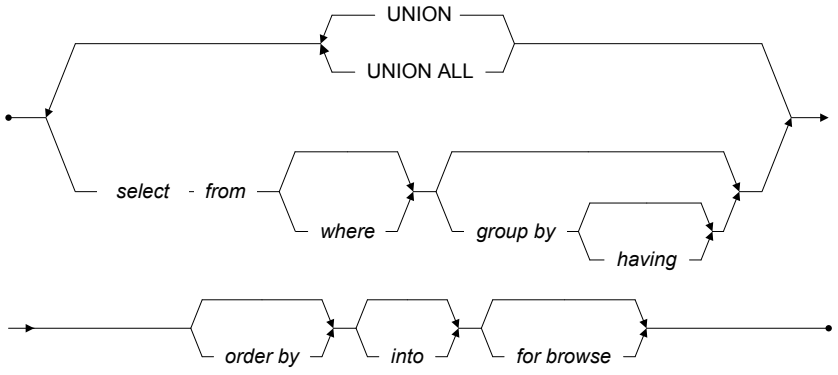


图 3-105 SELECT（使用FROM）语法图

省略FROM的SELECT语句

省略FROM的SELECT语句一般用于获得用户自定义函数（UDF）或表达式的值。它不要求用户在查询中使用FROM子句来指定是在哪个表中查询。因此，用户也就不能在这种省略FROM的SELECT查询中指定字段名或表名。

省略FROM子句的SELECT查询不能和如下子句一起使用：WHERE、GROUP BY、HAVING、ORDER BY、DISTINCT以及UNION。

图 3-106 SELECT（省略FROM）语法图

➲ **例**

```
dmSQL> SELECT abs(100), COS(100.0);
```

SELECT 子句

SELECT子句包含关键字SELECT以及要出现在结果集中的数据库对象或表达式列表。关键字ALL或DISTINCT用于指定是否允许返回相同记录。如果在SELECT语句中没有指定ALL或DISTINCT，那么DBMaster将默认返回所有记录。

结果列表中的值可能是字段名、表达式，也可能是常量或星号（*）。星号表示查询结果将显示源表中的所有字段。您在写字段列表的时候可以给字段名或星号加上源表名作为其前缀。

在字段列表中您可以使用四种基本类型的表达式、常量、函数以及集合函数。如果在字段列表中有常量，那么查询结果的每条记录在这个字段上都将返回这个常量值。集合函数为每一组记录返回一个值，这类函数通常用在包含GROUP BY子句的查询中。

您可以在字段列表中使用“OID”，将和每条记录相关的对象编号（OID）作为一个字段名。OID实际上是一个隐藏字段，数据库中每条记录在这个字段上的值都是唯一的。OID值并不一定是连续的。

您可以使用显示标签来为结果集里的字段或不是来自于字段的表达式生成的值指定一个临时名称。使用关键字AS来指定指派给结果集中某个字段的标签名称。

expression ..... 在结果集中返回值的表达式。

column_name .... 检索的字段名称。

label ..... 在结果集中作为字段的标签，这个名称和源表中的字段名称不同。

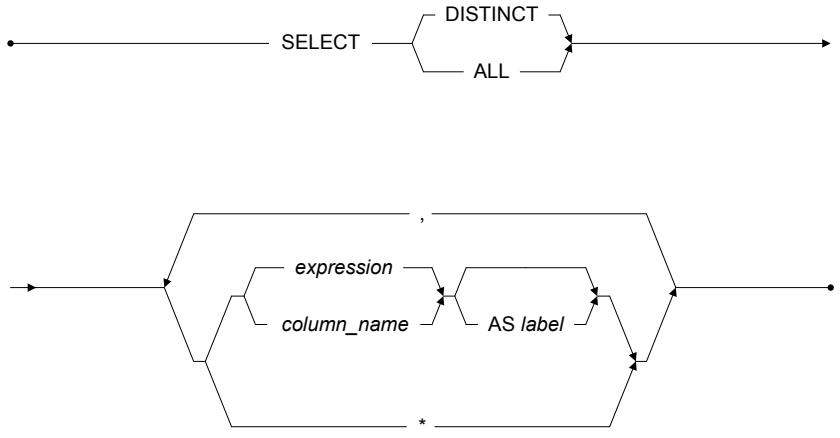


图 3-107 SELECT子句语法图

FROM子句

FROM子句列出的是将要从哪些表或视图中查询数据。如果字段名的出处不确定，那么这个子句将标明字段的出处。该查询源可能是一个表名、视图名、查询结果或同义字名。它可能是一个单一的源，也可能是一个外部源，外部源由连接多个源的OUTER关键字所指定。

您也可以为表另外指定一个关联名，并且可以在SELECT语句的其它子句中使用它。这可以增加语句的可读性。自连接情况下使用表的关联名是很有用的。

➲ **例**

下例在表**t2**的字段**c2**上分组，并查询每一组在字段**c1**上的最大值。最后为子查询的结果集指定了一个关联名**t3**。

```
dmSQL> SELECT * FROM (SELECT MAX(c1) FROM t2 GROUP BY c2) AS t3 (c1);
```

您可以使用外部连接（OUTER JOIN）关键字：OUTER, LEFT OUTER, JOIN或LEFT JOIN等来创建外部连接。在一个SELECT语句中可以有多

个OUTER JOIN关键字。OUTER JOIN关键字之前的所有源都必须是处于主导地位的源，而OUTER JOIN关键字之后的所有源必须都是处于辅助地位的源。您应该在FROM子句中指定所有的外部连接表的顺序，而在WHERE子句中指定外部连接的要素。WHERE子句中的完整连接要素会被作为外部连接要素，而另外的要素则应在外部连接要素之前求值。

DBMaster支持ANSI和ODBC外部连接，其语法是在ON子句中指定外部连接要素。WHERE子句中的其它要素应在外部连接要素之后求值。

您可以使用关键字CROSS JOIN来指定两个表交叉连接的结果。返回的记录和旧版也就是非SQL92版的连接中没使用WHERE子句的记录一样。结果和在FROM子句的表列表中使用逗号（,）的结果是一样的。

➲ **例**

```
dmSQL> SELECT * FROM t1 CROSS JOIN t2 CROSS JOIN t3 WHERE t1.c1 = t2.c1 AND t2.c2
= t3.c3;
```

以上的查询结果和下面的查询结果相同。

➲ **结果**

```
dmSQL> SELECT * FROM t1,t2,t3 WHERE t1.c1 = t2.c1 AND t2.c2 = t3.c3;
```

DBMaster 3.5和以后的版本中，您可以手动指定查询中使用哪种类型的扫描，也可以指定扫描中使用哪个索引。此外，即使您最近没有更新数据库的统计值，DBMaster查询优化器也会自动决定使用最有效的扫描类型来进行扫描。

source ..... 从哪个表或查询结果中检索数据。

alias..... 在其它子句中使用的源别名。

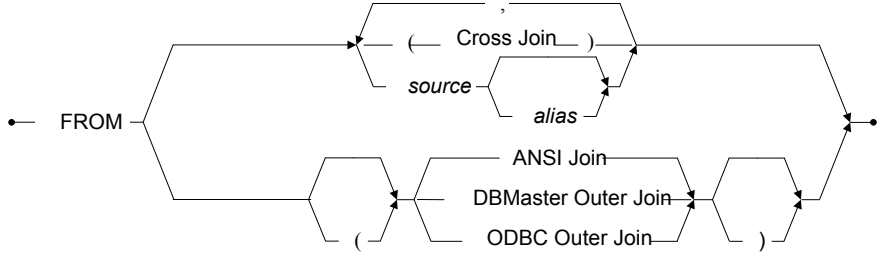


图 3-108 FROM子句语法图

强制索引扫描

通过下列语句来强制索引扫描。

```
table_name (INDEX [=] idx_name [ASC|DESC])
```

您也可以使用0来代表表扫描，用1来代表主键索引扫描。

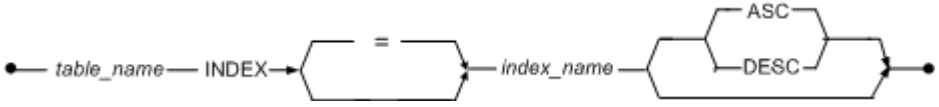


图 3-109强制索引扫描语法图

➲ **例1**

通过指定**0**来强制表扫描。

```
dmSQL> SELECT * FROM tb_tmp (INDEX=0);
```

➲ **例2**

通过指定**1**来强制主键索引扫描。

```
dmSQL> SELECT * FROM tb_tmp (INDEX=1);
```

➲ **例3**

强制执行在索引**idx1**上的索引扫描。

```
dmSQL> SELECT * FROM t1 (INDEX idx1);
```

➲ **例4**

允许查询优化器决定在表**t1**上使用何种扫描，而在表**t2**的索引**idx1**上强制执行一个索引扫描。

```
dmSQL> SELECT * FROM t1, t2 (INDEX idx1);
```

强制索引扫描与别名

下面的语法用来强制执行索引扫描并为表指定一个别名。

```
table_name (INDEX [=] idx_name) aliasname
```

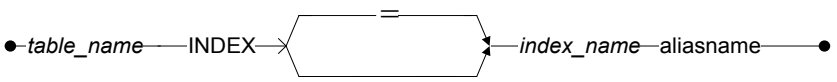


图 3-110强制索引扫描与别名语法图

➲ **例**

下例在索引**idx1**上执行强制索引扫描，并且为表指定了别名**a,b**：

```
dmSQL> SELECT * FROM t1 (INDEX idx1) a, t1 b WHERE a.c1 = b.c1;
```

强制索引扫描与同义字

下面的语法通过同义字强制执行索引扫描。

```
synonym_name (INDEX [=] idx_name)
```

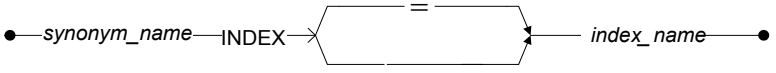


图 3-111强制索引扫描与同义字语法图

➲ **例**

下例使用同义字**s1**在索引**idx1**上强制执行索引扫描。

```
dmSQL> SELECT * FROM s1 (INDEX idx1);
```

强制索引扫描与视图

下面的语法用于创建视图时强制索引扫描。

```
view_name (INDEX [=] idx_name)
```

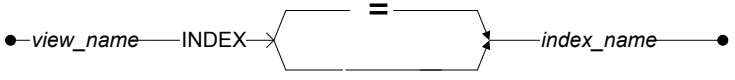


图 3-112强制索引扫描与视图语法图

➲ **例1**

下例在创建视图**v1**时，在索引**idx1**上强制执行索引扫描。

```
dmSQL> CREATE VIEW v1 as SELECT * FROM t1 (INDEX idx1);
```

对视图进行查询时，不能使用强制索引扫描。

➲ **例2**

错误的用法将会返回一个错误信息。

```
dmSQL> SELECT * FROM v1 (INDEX idx1);
```

强制全文索引扫描

下面的语法用来执行强制全文索引扫描。

```
table_name (TEXT INDEX [=] idx_name)
```

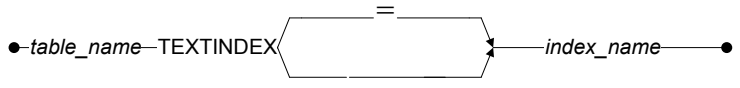


图 3-113强制全文索引扫描语法图

➲ **例**

在索引**tidx1**上强制执行一个全文索引扫描。

```
dmSQL> SELECT * FROM t1 (TEXT INDEX tidx1);
```

SOURCE SUBCLAUSE

FROM子句中的源（*source*）可能是表名称，也可能是一组查询结果。当源是一组查询结果时，其语法参考下图。

*Correlation_name...*表示子查询的查询结果。

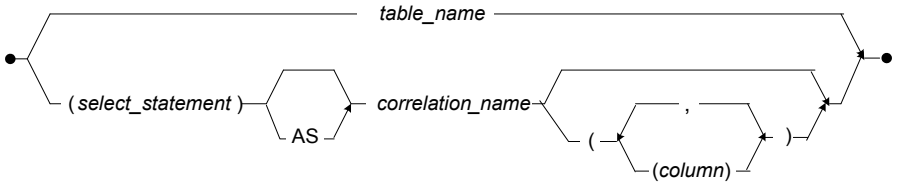


图 3-114 Source subclause 语法图

WHERE子句

WHERE子句可用来指定查询条件和多表查询的连接标准。如果一条记录满足查询条件，那么它将作为结果集的一部分被返回。关于如何在WHERE子句中使用SELECT语句和子查询，请参看子查询主题。

在引用字符串时可以使用百分号（%）和下划线（_）来作为通配符。其中百分号用来匹配0或多个字符，而下划线则只能匹配一个字符。ESCAPE子句是可选的，其作用是允许用户定义一个转义字符，以便引用包含百分号和下划线的字符串，而不会将百分号和下划线作为通配符。使用两个连续的单引号来表示字符串中的单引号。

WHERE子句中谓词的可以是一些简单的运算符，如下所示：

- *关系运算符*—它们可以是下列的运算符: >、>=、<=、<、=、and <>。当关系运算符任一边的表达式满足了这个运算符所创建的比较关系时，这个比较关系运算符的条件就被满足了。
- *BETWEEN* — 这个比较谓词的形式是：*x BETWEEN y AND z*。当*BETWEEN*关键字左边的值或表达式位于关键字条件*AND*所指定的范围内，也就是位于*BETWEEN*右边的两个表达式的范围内时，BETWEEN条件也就被满足了。
- *IN* — 这个比较谓词的形式是：*x IN (y, z, ...)*；当关键字*IN*左边的值或表达式包含在关键字右边的值列表中时，*IN*条件就满足了。
- *IS NULL* — 这个谓词的形式是：*x IS NULL*；当关键字*IS NULL*左边的值或表达式是空值时，*IS NULL*条件就被满足了。

- *IS NOT NULL* — 这个谓词的形式是：*x IS NOT NULL*；如果*IS NOT NULL*左边的值或表达式不为空值时，*IS NOT NULL*条件也就被满足了。
- *LIKE* — 这个谓词的形式是：*x LIKE 'y' ESCAPE 'z'*；当关键字LIKE左边的字符串值或表达式满足关键字右边的字符串时，LIKE条件就被满足了。注意，这里的字符串是大小写敏感的。
- *MATCH* — 这个谓词的形式是：*x NOT CASE MATCH 'y'*；当MATCH关键字右边引用的字符串和关键字左边的字符串值或表达式匹配时，*MATCH*条件就满足了。关键字*NOT*用以表示不匹配，而关键字*CASE*的作用则是让查询变得大小写敏感，这两个关键字都是可选的。
- *CONTAIN* — 这个谓词的形式如下：*x NOT CASE CONTAIN 'y'*；当*CONTAIN*关键字右边所引用的字符串和关键字左边的字符串或者表达式的任何一部分匹配的话，*CONTAIN*条件也就被满足了。关键字*NOT* 用来表示不包含，而关键字*CASE*的作用则是让查询表的大小写敏感，这两个关键字都是可选的。
- *CONTAINS* – 当相关字段的字符串和“string pattern”指定的字符串匹配时，包含运算符的条件也就被满足了。

  这个运算符的语法如下：[NOT] CONTAINS (column || column [|| column]…, 'string pattern'[, option string])

  ➲ **例**

  本例中的SELECT语句将从**mcol**表的**c4**字段中查询满足条件的记录。其中需要满足的条件是：**c1**和**c4**字段都必须包含字符串'**Mail Server**'。可选项CASE的作用是使查询区分大小写。

```
dmSQL> SELECT c4 FROM mcol WHERE CONTAINS (c1 || c4, 'Mail
Server', 'CASE');
```

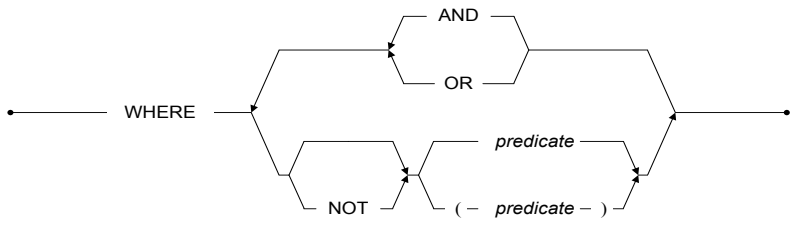


图 3-115 WHERE Clause语法图

CAST

CAST允许用户将一个类型的输出数据转换为另一个数据类型。下面的表列出了所有的合法类型转换。表中是将行**X**的数据类型转换成列**Y**的数据类型。表中的值包括N和Y，其中N表示两种数据类型间不能相互转换，Y表示可以转换。

Numeric、Character和Date/Time数据类型都包含了多种数据类型。Numeric类型包含integer（int, serial）、smallint、float、double以及decimal等类型；Character类型包含char和varchar等类型；而Date/Time类型则包含date、time、timestamp等类型。

Xy	int (serial)	small-int	decimal	double	float	(var) char	(var) binary	date	time	time-stamp	file	blob	clob
int(serial)	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
smallint	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
decimal	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
double	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
float	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N
(var)char	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N
(var)binary	N	N	N	N	N	Y	N	N	N	N	N	N	N
date	N	N	N	N	N	Y	N	Y	N	Y	N	N	N
time	N	N	N	N	N	Y	N	N	Y	N	N	N	N
time-stamp	N	N	N	N	N	Y	N	Y	Y	Y	N	N	N
file	N	N	N	N	N	Y	Y	N	N	N	Y	N	N
blob	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y
clob	N	N	N	N	N	Y	Y	N	N	N	N	Y	Y

表 3 -1 CAST Conversion Table

➲ **例1**

下例展示了如何在WHERE中使用CAST()。

```
dmSQL> SELECT * FROM t1 WHERE CAST(c1 AS CHAR(20)) LIKE '2001%';
```

➲ **例2**

下例是如何在表达式中使用CAST()。

```
dmSQL> SELECT CAST(c1+c2 AS CHAR(10)) FROM t1;
```

➲ **例3**

下例是如何嵌套使用CAST()。

```
dmSQL> SELECT CAST(CAST(123 AS CHAR(10)) || CAST(45 AS CHAR(10)) AS INT) FROM t1;
```

CASE

CASE是SQL 99中的一个函数。

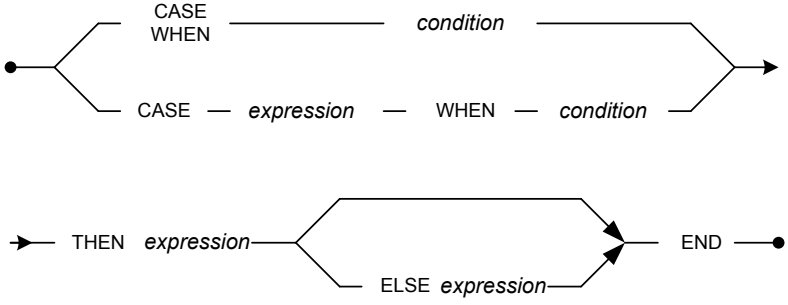


图 3-116 CASE语法图

➲ **例1**

CASE WHEN p1 THEN v1 ELSE CASE WHEN p2 THEN v2 ELSE… ELSE vn END…END。这条语句的意思是如果条件p1的值为真则执行v1，如果p2为真则执行v2，依此类推直到vn。这条语句的执行如下：

```
dmSQL> SELECT CASE WHEN c1=3 THEN c2 ELSE CASE WHEN c1=5 THEN c3 ELSE c4 END END
FROM t1;
```

➲ **例2**

CASE c1 WHEN d1 THEN v1 ELSE CASE c1 WHEN d2 THEN v2 ELSE…ELSE vn END…END。这条语句的意思是如果c1=d1则执行v1，如果c1=d2则执行v2，依此类推直到vn。这条语句的执行如下：

```
dmSQL> SELECT CASE c1 WHEN 3 THEN c2 ELSE CASE c1 WHEN 5 THEN c3 ELSE c4 END END
FROM t1;
```

➲ **例3**

CASE WHEN p1 THEN v1 WHEN p2 THEN v2 WHEN…ELSE vn END。这条语句的意思是如果条件p1为真则执行v1，如果p2为真则执行v2，依此类推直到vn。这条语句的执行如下：

```
dmSQL> SELECT CASE WHEN c1=3 THEN c2 WHEN c1=5 THEN c3 ELSE c4 END FROM t1;
```

COALESCE

COALESCE是SQL 99中的一个函数。COALESCE (v1, v2, v2,….vn) 和“if v1 IS NOT NULL then v1 else if v2 IS NOT NULL then v2 else…….else vn”等价。

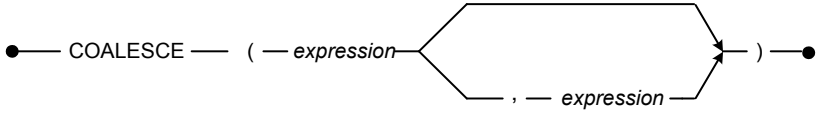


图 3-117 COALESCE语法图

➲ **例1**

```
dmSQL> SELECT COALESCE(c1, 7) FROM t1;
```

➲ **例2**

```
dmSQL> SELECT COALESCE(c1, c2, c3, 7) FROM t1;
```

NULLIF

NULLIF是SQL 99中的一个函数。NULLIF(v1, v2) 等价于“if v1=v2 then NULL else v1”。

图 3-118NULLIF语法图

➲ **例1**

```
dmSQL> SELECT NULLIF(c1, 7) FROM t1;
```

➲ **例2**

```
dmSQL> SELECT NULLIF(t1.c1, t2.c1) FROM t1, t2;
```

IFNULL

IFNULL是ODBC中的一个函数。IFNULL (v1, v2)等价于coalesce(v1,v2)，它们都等价于“if v1 is not null, then v1 else v2”。

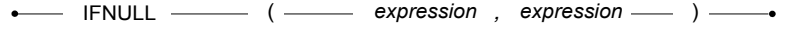


图 3-119IFNULL 语法图

➲ **例1**

```
dmSQL> SELECT IFNULL(c1, 7) FROM t1;
```

➲ **例2**

```
dmSQL> SELECT IFNULL(t1.c1, t2.c1) FROM t1, t2;
```

复合比较

您可以使用逻辑运算符AND、OR或NOT来将单一的条件组合成复合条件。关键字AND表示两个查询条件都必须为真；关键字OR表示两个查询条件只需满足其中一个，也可以两个都满足；而关键字NOT则表示查询使条件为假的记录。

➲ **例1**

```
dmSQL> SELECT * FROM Customer
       WHERE City NOT IN ('LA', 'NY') AND Age > 40;
```

➲ **例2**

```
dmSQL> SELECT * FROM Orders
```

```
      WHERE Price > 10,000 OR Ship_Date = TODAY;
```

连接条件

连接条件（*join condition*）是关系运算符在两个字段上的比较，这两个字段来自于不同的表。（例如：**Orders.CusNum = Customer.CusNum**）。

当在WHERE子句中使用连接条件来为两张不同表中的字段创建联系时，就将两个表连接在一起了。连接的结果是创建了一个临时的复合表，在这个表中来自于每个表的满足连接条件的两条记录将连接成一个单一的记录。DBMaster中有四种连接类型：两表连接（ two-table-joins）、多表连接（multiple table-joins）、自连接（self-joins）以及外部连接（outer-joins）。

ON <SEARCH_CONDITION>

ON <search_condition> 指定了创建连接的条件，该条件一般使用字段和比较运算符，但也可以使用谓词。

➲ **例**

```
dmSQL> SELECT ProductID, Suppliers.SupplierID
       FROM Suppliers JOIN Products
       ON (Suppliers.SupplierID = Products.SupplierID);
```

ANSI OUTER-JOIN

外部连接是将满足外部连接条件的两个或多个表连接起来。外部连接条件是两个表的字段上的比较或关系运算。左边表中的每条记录都将返回，对应右边表的所有外部条件不能被满足的列将用空值补齐。

下图显示了ANSI JOIN语法以及优化线索图：

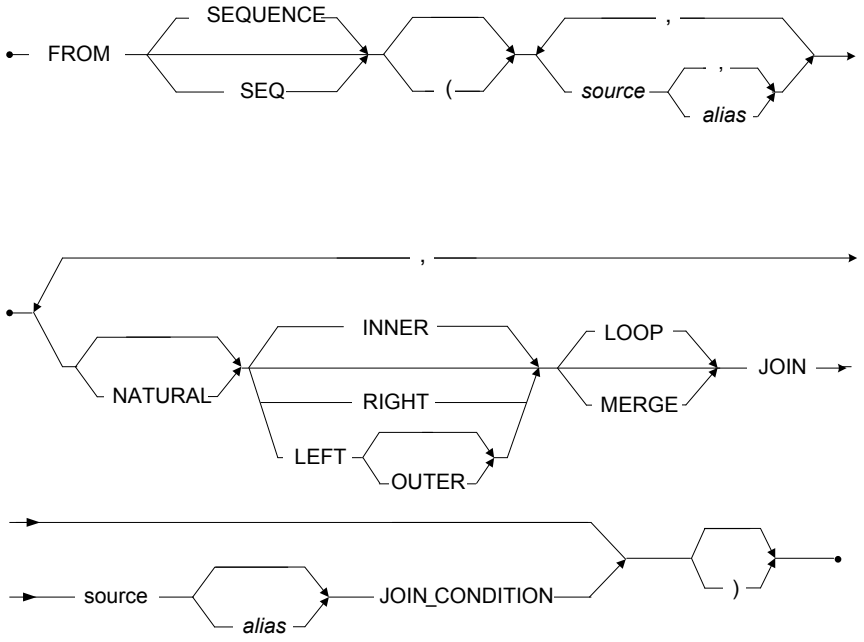


图 3-120 ANSI Join 语法图

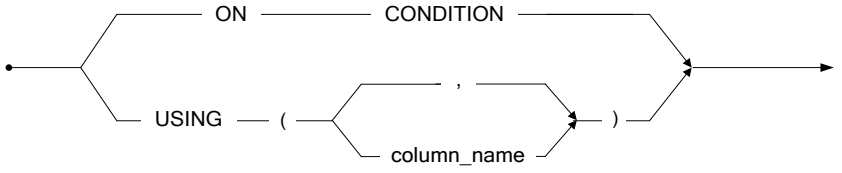


图 3-121 ANSI Join Condition 语法图

关键字SEQUENCE、SEQ、LOOP和MERGE用来作为优化器的线索，它们不是ANSI的语法。如果连接执行中使用了指定的关键字，优化器将

选择执行计划；如果它们对连接的执行没有任何影响，优化器也不会返回任何错误信息。

如果指定SEQUENCE/SEQ关键字，连接的执行顺序就会和SQL命令中的表的连接顺序一样，表连接的执行顺序不会被优化器所改变。当执行的是外部连接时，这两个关键字不会起作用。

➲ **例1**

```
dmSQL> SELECT * FROM SEQ t1 INNER JOIN t2 ON t1.c1=t2.c1 INNER JOIN t3 ON
t1.c2=t3.c2;
```

关键字LOOP/MERGE为内连接或外连接指定了连接执行方法。当指定了连接的执行方法，连接表的连接执行顺序就不会被改变。LOOP关键字指定在进行内连接或外连接时，优化器将使用嵌套连接；而MERGE关键字指定在进行等值的内连接或外连接时，优化器将使用merge连接。

➲ **例2**

```
dmSQL> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.c1=t2.c1;
```

DBMASTER OUTER-JOIN

以下是旧版DBMaster使用的语法，这和ANSI外部连接语法的区别在于此外部连接的要素是由DBMaster的查询优化器来决定的。下面的语法不支持RIGHT-JOIN，用户不能将它和ANSI外部连接的语法混用。

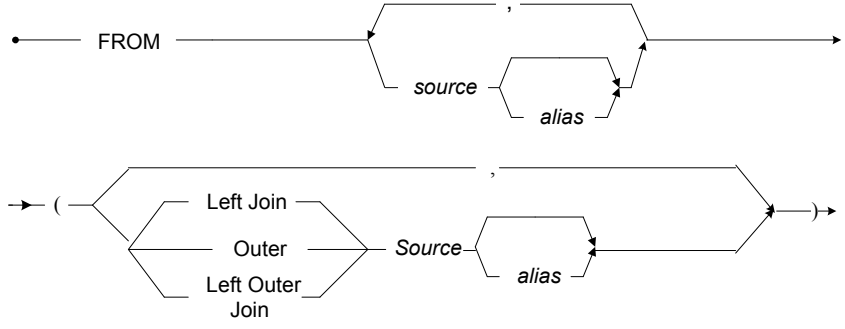


图 3-122DBMaster Outer-Join语法图

ODBC OUTER-JOIN

ODBC Outer-Join和ANSI Outer-Join的语法相似，不同的是该语法中的所有选项都必须使用。

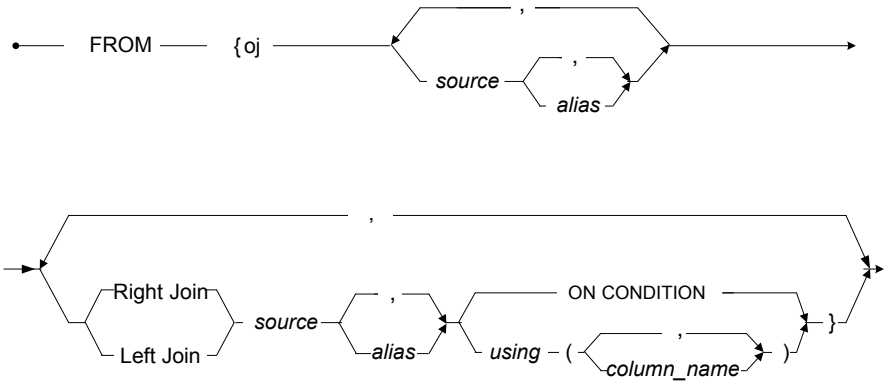


图 3-123 ODBC Outer-Join 语法图

SELF-JOIN

该连接是表自身的连接，您需要在FROM子句中两次列出表的名称，每次为表指派不同的别名。在**WHERE**子句中可以使用别名来指代这“两个”表。假设表**Employeesinfo**中有**Manager_ID**字段（经理职工号）。

➲ **例**

下例是表**Employeesinfo**的自连接，查询结果将列出所有职工的姓名以及他们的经理的姓名。

```
dmSQL> SELECT e.FName AS Emp, m.Fname AS Manager
                                FROM Employeesinfo e, Employeesinfo m
                                 WHERE e.Manager_Id = m.Emp_Id;
```

RIGHT-JOIN

Right-Join中，如果外部条件不满足，结果集会返回右边表中的所有记录（包含右边表中不满足连接条件的记录），而对应左边表的列将用空值补齐。因此查询的结果是以上记录加两张表做内部连接后返回的记录。

➲ **例**

```
dmSQL> SELECT * FROM t1 RIGHT JOIN t2 ON t1.c1 = t2.c1;
```

INNER-JOIN

使用INNER JOIN后，查询结果将返回所有满足连接条件的一对记录。同时两张表中不能满足连接条件的记录将被抛弃。如果查询中只用了JOIN关键字，那么系统默认的连接类型是INNER-JOIN类型。

➲ **例1**

```
dmSQL> SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1;
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 JOIN t2 ON t1.c1 = t2.c1;
```

➲ **结果**

```
dmSQL> SELECT * FROM t1, t2 WHERE t1.c1 = t2.c1;
```

NATURAL JOIN

如果NATURAL关键字用在了JOIN类型之前，您就不能在该语句中使用ON条件来指定连接条件或者使用USING关键字来列出连接字段列表。NATURAL JOIN将在连接表的共有字段名上执行等值连接。NATURAL JOIN的执行结果同在USING字段列表中指定了所有的共有字段一样。“select *”的映射列表将首先显示用来作为连接的字段，接着将显示所连接表中的其它剩余字段。

➲ **例1**

```
dmSQL> SELECT * FROM t1 NATURAL INNER JOIN t2;
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 NATURAL LEFT JOIN t2;
```

ON 条件

ON条件为连接表指定了连接条件。

➲ **例1**

```
dmSQL> SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1;
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 LEFT JOIN t2 ON t1.c1 = t2.c1;
```

USING 字段列表

USING字段列表用来指定在表连接中用来作为连接的字段列表。如果指定了关键字USING，那么列表中的字段必须是存在而且在连接表中是可以比较的。返回的结果同在ON条件中指定等值连接一样。“select *”的映射列表将首先显示用来作为连接的字段，接着将显示所连接表中的其它剩余字段。

➲ **例1**

```
dmSQL> SELECT * FROM t1 INNER JOIN t2 USING (c1, c2);
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 LEFT JOIN t2 USING (c1) LEFT JOIN t3 USING (c1);
```

两表连接

两表连接指连接条件只涉及两张表。

➲ **例1**

下例是一个两表连接，同字段**Dept_id**来连接两张表**Emp_Name**和**Dept_Name**。

```
dmSQL> SELECT FName, Dept_Name FROM Employeesinfo, Department
                        WHERE Employeesinfo.Dept_ID = Department.Dept_Id;
```

➲ **例2**

下例是两张表的外部连接，目的是查询表**Department**中的所有记录和相应的项目（project），如果在**Project**表中有不属于该部门的项目，则项目名（**Proj_Name**）设为空值。

```
dmSQL> SELECT Dept_id, Dept_Name, Proj_Name FROM Department d outer Project p
```

```
                                    WHERE d.Dept_id = e.Dept_Id;
```

多表连接

多表连接是一种通过为每对表设定连接条件来连接多个表的连接。连接条件是来自于每个表的两个字段上的比较或关系运算。

➲ **例**

下例是一个三个表的连接，目的是选出工程部门**Engineering**中所有成员的所有项目。

```
dmSQL> SELECT Dept_Name, Proj_Name FROM Department d, Project p, Employeesinfo e
                          WHERE d.Dept_id = e.Dept_Id AND
                          p.Emp_Id = e.Emp_Id AND
                          Dept_Name = 'Engineering';
```

FORCED LOOP JOIN

下面是在两张表之间强制嵌套连接的一般语法：

```
table_name { INNER | OUTER } LOOP JOIN table_name
```

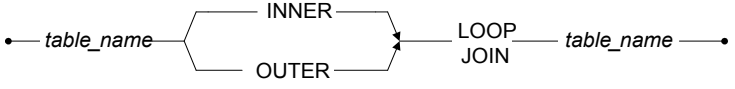


图 3-124 Force Loop Join 语法图

该类型的连接只能应用在INNER JOIN或OUTER JOIN 语法中。

➲ **例1**

```
dmSQL> SELECT * FROM t1 INNER LOOP JOIN t2 ON t1.c1=t2.c1;
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 OUTER LOOP JOIN t2 ON t1.c1=t2.c1;
```

FORCED MERGE JOIN

下面是在两张表之间强制Merge连接的一般语法：

```
table_name { INNER | OUTER } MERGE JOIN table_name
```

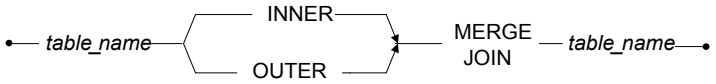


图 3-125 Force Merge Join语法图

如果连接中不能使用Merge连接，指定该连接也是没有用的，但是系统不会返回错误信息。

➲ **例1**

```
dmSQL> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.c1=t2.c1;
```

➲ **例2**

```
dmSQL> SELECT * FROM t1 OUTER MERGE JOIN t2 ON t1.c1=t2.c1;
```

FORCED JOIN SEQUENCE

下面的语法用来强制指定表的连接顺序。指定后，连接顺序就不能改变。

```
SELECT ..... FROM [SEQUENCE | SEQ] table_name_list
```

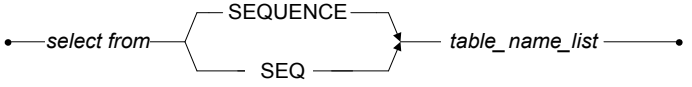


图 3-126 Force Join Sequence语法图

➲ **例1**

```
dmSQL> SELECT * FROM SEQUENCE t1, t2, t3 WHERE t1.c1=t2.c1 AND t2.c2=t3.c2;
```

➲ **例2**

```
dmSQL> SELECT * FROM SEQ t1 INNER JOIN t2 ON t1.c1=t2.c1 INNER JOIN t3 ON
t1.c2=t3.c2;
```

GROUP BY子句

您可以使用**GROUP BY**子句来为每个组生成一个聚集数据。这里的组指的是在某些字段上具有相同值的一组记录。为每个组计算的集合结果将作为结果集中的一条记录。这个集合结果的名称可以是字段名或显示标签。

SELECT语句中使用GROUP BY有一定的限制。在组上的查询列表必须符合下列条目中的一条：

- 用来聚集组中所有记录的集合函数，只产生一个值。
- 分组字段名，也就是在GROUP BY子句中所列的字段。
- 常量。
- 结合以上三条的表达式。

实际应用中，一个GROUP BY查询通常包含一个分组字段和一个集合函数。GROUP BY子句指定的包含空值字段的记录将被分到同一个组。

USING HASH/SORT子句用来作为优化器的优化线索。当指定了USING HASH语句，优化器将为GROUP BY选择哈希方法来执行；但当GROUP BY产生太多的组时，即使指定了USING HASH关键字，优化器也不会使用哈希方法。当指定了USING SORT关键字，如果在GROUP BY 子句中的分组字段上建有索引，优化器将试图使用一个索引扫描或者在执行GROUP BY子句时，选择执行计划来排序GROUP BY字段。

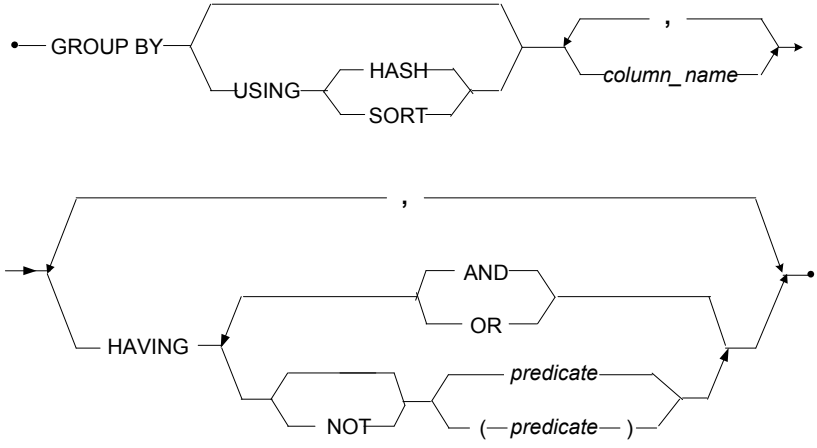


图 3-127 GROUP BY Clause语法图

➲ **例1**

下例中，使用**SELECT**来查询部门ID（**Dept_Id**）和每个员工的平均工资（**AVG(salary)**）。然后求部门编号为**ID1**的平均工资。

```
dmSQL> SELECT Dept_Id, AVG(Salary) FROM Employeesinfo
                        GROUP BY Dept_Id;
dmSQL> SELECT Dept_Id AS ID1, AVG(Salary) FROM Employeesinfo
                        GROUP BY ID1;
```

➲ **例2**

下例通过哈希方法，来查询表**Employeesinfo**中每个部门的部门号**Dept_Id**和平均工资**AVG(salary)**。

```
dmSQL> SELECT Dept_Id, AVG(Salary) FROM Employeesinfo
                        GROUP BY Dept_Id USING HASH;
```

FORCED GROUP BY METHOD

下面是强制连接顺序的一般语法：

```
GROUP BY column_name_list [USING SORT | USING HASH] having .....
```

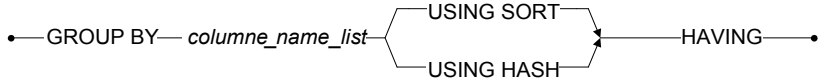


图 3-128 Force Group by Method 语法图

➲ **例1**

```
dmSQL> SELECT c1,c2,COUNT(*) FROM tb_test GROUP BY c1,c2 USING HASH;
```

➲ **例2**

```
dmSQL> SELECT c1,c2,COUNT(*) FROM tb_test GROUP BY c1,c2 USING SORT HAVING
SUM(c3)>0;
```

HAVING子句

HAVING子句可用来选择或删除表中的分组，HAVING子句中可以包含子查询。关于子查询的信息可参看SUBQUERY部分。

➲ 例

下例将查询总销售额超过一百万美元的部门所有产品的平均销售额和部门名称。

```
dmSQL> SELECT Dept_Name, AVG(Amount) FROM Sales
                              GROUP BY Dept_Name
                              HAVING SUM(Amount) > 1000000;
```

ORDER BY子句

查询结果中的记录并不总是按某一特定顺序进行排序的。因此您可以使用ORDER BY子句来指定查询结果按照某个或某些字段上的值进行排序。

关键字ASC/DESC用于指定查询结果是按升序（即最小的值排在最前面）还是降序排列，系统默认的排序顺序是升序。空值被认为是比非空值大的值。如果使用关键字ASC来指定排序顺序，那么空值将会排在任何非空值的后面。

column_name .... SELECT列表中的字段或者显示标签，查询结果将按照这个字段上的值进行排序。

column_number...排序的字段或表达式在SELECT列表中所处的位置。

expression ..... 按照这个表达式来对查询结果排序。

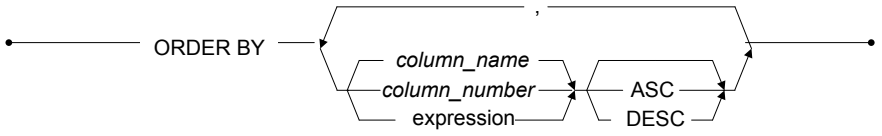


图 3-129 ORDER BY Clause语法图

➲ **例1**

下例将按照表中的name字段上的升序来排列查询结果，name值一样时将按照age字段上的降序来排列。

```
dmSQL> SELECT Name, Address, Age FROM Customer
                      ORDER BY Name, Age DESC;
```

➲ **例2**

下例在ORDER BY子句中使用了字段位置（cloumn number）和显示标签这两个参数。

```
dmSQL> SELECT Dept_Id, Salary + Bounce AS Total_Com, FName
                          FROM Employeesinfo
                          ORDER BY 1, Total_Com;
```

组合运算 (UNION OPERATOR)

您可以使用UNION运算来将两个或多个查询结果组合成一个结果。在UNION查询中，组合结果将删除相同的记录，这使得组合结果中的每条记录都是唯一的。如果您确定所有查询结果中没有相同的记录，或者您想保留结果中的相同记录，那么您可以使用UNION ALL关键字。UNION ALL将保留查询的所有结果，并且运行起来比UNION快。

能使用UNION来组合的查询结果是有一定的限制的，限制如下：

- 两个查询结果中所包含的字段数要相同。
- 每个查询结果中相应字段的数据类型必须兼容，而且不能有相同的字段名。第一个查询结果的字段名将作为组合结果的字段名。
- 如果要使用*ORDER BY*子句，那么就应该在最后一个*SELECT*查询后面指出排序字段在*SELECT*列表中的位置。

➲ **例1**

下例说明了如何在一个**SELECT**语句中使用**UNION**子句。

```
dmSQL> SELECT C1, C2 FROM T1
       UNION
       SELECT C3, C4 FROM T2
       ORDER BY 2;
```

➲ **例2**

下例说明如何在一个**SELECT**语句中使用**UNION ALL**子句。

```
dmSQL> SELECT 'MOVIE', Event FROM Entertainment WHERE Type = 'MOVIE'
        UNION ALL
        SELECT 'BOOK', Name FROM MyBook;
```

子查询

子查询是另一段出现在WHERE或HAVING子句中的SQL查询语句。子查询一般都放在一对圆括号中，但它和一般的SELECT语句是完全一样的。

子查询的结果必须是一些只包含一个字段的记录。此外，如果子查询的结果要用于关系运算比较中，那么子查询返回的结果只能是一条只包含一个字段的记录。

➲ **例**

下例查询的是工资超过平均工资的职工姓名。

```
dmSQL> SELECT Name FROM Employeesinfo
       WHERE Salary > (SELECT AVG(Salary) FROM Employeesinfo);
```

IN子查询

IN子查询是一种从属关系的测试。如果表达式的值和一条或多条子查询返回的记录相匹配，那么查询的结果将为真。IN子查询中返回的数据应该是一些只包含一个字段的记录。

➲ **例**

下面将查询位于**NY**的部门的所有员工（**employeesinfo**）。

```
dmSQL> SELECT FName FROM Employeesinfo
       WHERE Dept_Id
       IN (SELECT Dept_Id FROM Department WHERE City = 'NY');
```

EXISTS 子查询

EXISTS检查子查询是否返回记录。子查询中有时需要参照当前主查询记录中的字段值，这个字段被称作外部参照（*outer reference*）。下例中的

字段**d**.**Dept_id**就是一个外部参照。在这种查询中可以有多个层次的子查询，外部参照可被任何层次上的子查询参照。

➲ **例**

下面例子将查询至少有一个员工（**EMPLOYEE**）的工资超过**500000**的部门（**Department**）名称。

```
dmSQL> SELECT Dept_Name FROM Department d
       WHERE EXISTS
             (SELECT Dept_Id FROM EMPLOYEEFINFO e
                    WHERE e.Salary > 500000 AND d.Dept_Id = e.Dept_Id);
```

ANY/ALL/SOME子查询

如果在子查询中使用关键字ALL，那么只有当所有的返回值使得比较条件为真时，主查询的条件才为真。如果子查询没有返回值，也就是返回一个空集，主查询的条件也为真。如果返回集不为空，但其中包含NULL，那么主查询的条件将为假。

如果在子查询中使用关键字ANY，那么只要至少有一个返回值使得比较条件为真时，主查询的条件就为真。如果子查询没有返回值，主查询的条件将为假。

➲ **例**

下例查询的是非经理的员工，这些员工的工资至少大于一个经理的工资。

```
dmSQL> SELECT FName FROM Employeesinfo
       WHERE Manager = 'N' AND Salary > ANY
            (SELECT Salary FROM EMPLOYEESINFO WHERE Manager = 'Y');
```

FOR BROWSE子句

关键字FOR BROWSE表示查询中使用浏览模式。在浏览模式中，事务不能要求锁资源，这样其他用户的查询就不会被阻塞。正是因为事务不能要求锁，数据库中的数据才不一定能够重复读取。但是浏览模式对于浏览数据、生成报表是很有用的。

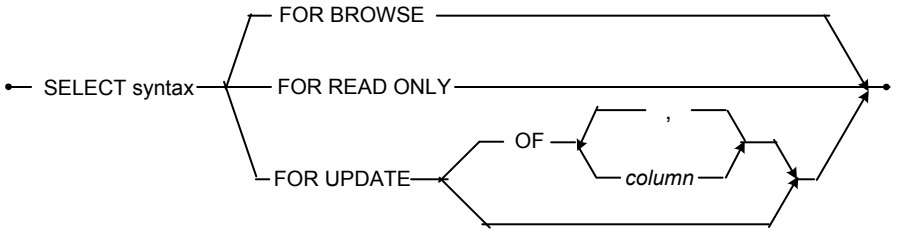


图 3-130 FOR BROWSE Clause语法图

LIMIT

LIMIT用来限制SELECT语句返回的记录数，开始返回行之前忽略*n*行。

offset ..... 指开始返回记录之前忽略多少条记录。

rows ..... 返回的记录的个数。

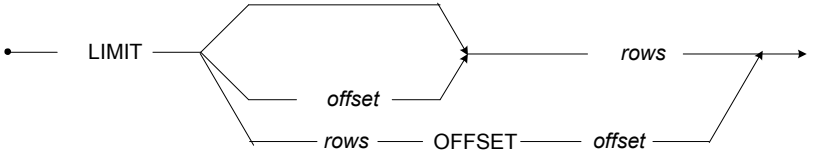


图 3-131 LIMIT语法图

➲ **例**

```
dmSQL> SELECT * FROM tb_test ORDER BY c1 LIMIT 10;
```

聚合函数

聚合函数可对一组值进行计算并返回单一值，DBMaster支持以下内置聚合函数：

- MIN
- MAX
- AVG

- COUNT
- SUM
- XMLAGG

MIN函数用于返回所有输入值的最小值。

MAX函数用于返回所有输入值的最大值。

AVG函数用于返回所有输入值的平均值（算数平均值）。

COUNT函数用于返回符合设定标准的记录数量。

SUM函数用于返回所有输入值的和。

XMLAGG用于返回串联的XML值。

语法如下所示：

```
{AVG|MAX|MIN|SUM|XMLAGG} ([ALL|DISTINCT] expression
[,comparsion_predicate])
|COUNT (* [,comparsion_predicate])
|COUNT ([ALL|DISTINCT] expression [,comparsion_predicate])
```

comparison_predicate……比较表达式

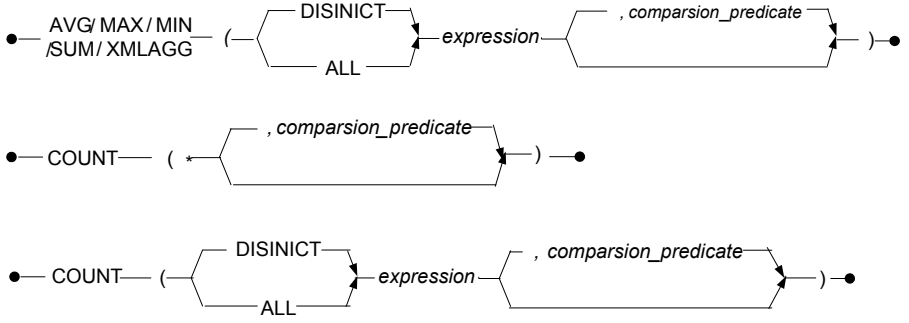


图 3-132 AGGREGATE FUNCTION语法图

➲ 例

```
dmSQL> SELECT COUNT(*) FROM tb_test;
```

聚合函数仅在SELECT语句的选择列表、HAVING子句、GROUP BY子句、ORDER BY子句中作为表达式使用，在其它子句中不能作为表达式使用。

WINDOW函数

WINDOW函数对当前行的一个相关行集执行计算，与聚合函数执行的计算类似。但WINDOW函数的返回值不是一个单一的输出行，这是WINDOW函数不同于常规聚合函数的地方。

语法如下所示：

```
func_name() OVER ([PARTITION_BY_CLAUSE] ORDER_BY_CLAUSE)
```

PARTITION_BY_CLAUSE… 指定用于将结果集划分为分区的字段。WINDOW函数可独立适用于每个分区且对每个分区重新计算。

*ORDER_BY_CLAUSE……*指定用于定义申请WINDOW函数顺序的字段。

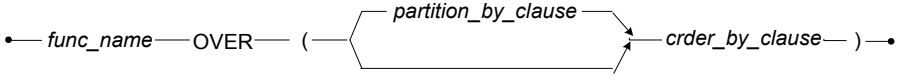


图 3-133 WINDOW FUNCTION语法图

DBMaster支持以下WINDOW函数：

- ROW NUMBER
- RANK
- DENSE_RANK

ROW NUMBER函数用于返回结果集中的某一行在其分区内的序列号，每个分区的序列号从第一行的序列号1开始，函数返回值类型为BIGINT。

RANK函数和DENSE_RANK函数类似，都是用于返回某个值在一组值中的排名，函数返回值类型均为BIGINT。二者不同之处在于：RANK函数在测试值相同时返回非连续排名，而DENSE_RANK函数总是返回连续排名。

➲ **例1**

下例用于返回各类书销量的ROW NUMBER、DENSE以及DENSE_RANK值。

```
dmSQL> SELECT TITLE, BOOK_CATEGORY, SALE_QTY,
       ROW_NUMBER() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS
       ROW_NUMBER,
       RANK() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS RANK,
       DENSE_RANK() OVER (PARTITION BY BOOK_CATEGORY ORDER BY SALE_QTY) AS
DENSE_RANK
       FROM BOOK_STORE;
```

结果如下所示：

```
TITLE    CATEGORY SALE_QTY     ROW_NUMBER          RANK          DENSE_RANK
======== ======== ======== ================ ================ ================
book3    business     20           1                1                1
book2    business     30           2                2                2
book1    business     40           3                3                3
book1    computer     10           1                1                1
book2    computer     20           2                2                2
book3    computer     20           3                2                2
book4    computer     30           4                4                3
```

WINDOW函数的使用有以下限制：

- ORDER BY CLAUSE不能使用order by常数。

➲ **例2**

```
dmSQL> SELECT row_number() OVER (ORDER BY 1) FROM t1;
```

- 查询中使用的所有WINDOW函数的OVER子句必须相同。

➲ **例3**

```
dmSQL> SELECT row_number() OVER (ORDER BY c1), row_number() OVER (ORDER BY c2)
FROM t1;
```

- WINDOW函数不支持GROUP BY或聚合函数。

➲ **例4**

```
dmSQL> SELECT max(c1), row_number() OVER (GROUP BY c1) FROM t1;
```

- INSERT、DELETE或UPDATE不支持WINDOW函数。

- WHERE子句或子查询不支持WINDOW函数。

XML函数

XML函数是一组用于利用SQL数据生成XML内容的函数。

DBMaster支持XML函数，XML函数是SQL语句的一部分，用户可通过dmsql、odbc或jdbc接口使用这些函数。

DBMaster支持以下XML函数：

- xmlelement
- xmlforest
- xmlagg(xml)
- xmlcomment(text)

xmlelement语法如下：

```
xmlelement(name name [, xmlattributes(value AS attname [, ... ])] [,
content, ...])
```

xmlelement表达式可生成名字、属性及内容已给定的xml元素。

*name……*xml元素标签名。若该标签名包含无效的名称字符，则使用hex格式将其替换。例如，若标签名是‘phone number’（phone和number之间存在一个空格），该标签名将被替换为phone_x20_number。

attname… 属性名。

*content……*可以是纯文本、子xml元素或xml注释。

➲ **例1**

```
dmSQL> SELECT XMLELEMENT(name foo, XMLATTRIBUTES(current_date as bar), 'cont',
'ent');

XMLELEMENT(NAME FOO, XMLATTRIBUTES(CURRENT_DATE AS BAR), 'CONT', 'ENT')
================================================================================
<foo bar="2011-08-18">content</foo>

1 rows selected
```

xmlforest语法如下：

```
xmlforest(content [AS name] [, ...])
```

xmlforest表达式可生成名字和内容已给定的xml元素林（序列）。若没有指定名字，且内容值是字段引用，则该序列名称默认为字段名。

➲ **例2**

```
dmSQL> SELECT XMLFOREST(empname, phone) FROM employee;

XMLFOREST(EMPNAME, PHONE)
================================================================================
<empname>Abby</empname><phone>123-1234</phone>
<empname>Alice</empname><phone>234-1234</phone>
<empname>Amber</empname><phone>567-1234</phone>
3 rows selected
```

xmlagg(xml)语法如下：

```
xmlagg(xml) .......
```

xmlagg不同于上述其它函数，它是聚合函数，用于串联各行输入值。Xmlagg的输入应是XML片段，输出是CLOB类型。若无内容，则xml元素显示为空元素，例如<ABC/>。开始或结束标记后无附加新行。

➲ **例3**

```
dmSQL> SELECT XMLAGG(XMLELEMENT(name person, XMLELEMENT(name name, empname)))
FROM employee;

XMLAGG(XMLELEMENT(NAME PERSON, XMLELEMENT(NAME NAME, EMPNAME)))
================================================================================
<person><name>Abby</name></person><person><name>Alice</name></person><person><nam
e>Amber</name></person>

1 rows selected
```

xmlcomment(text)语法如下：

```
xmlcomment(text) .....
```

xmlcomment是用户自定义函数。xmlcomment输入是sql表达式，该表达式可生成nchar或char数据。xmlcomment输出是XML注释格式的字符串，该字符串以<!-- and ends with -->开始。

text……若text包括转义字符（例如，<>、&），则转义字符按实体显示。

➲ **例4**

```
dmSQL> SELECT XMLCOMMENT(empname) FROM employee;

XMLCOMMENT(EMPNAME)
================================================================================
<!--Abby-->
<!--Alice-->
<!--Amber-->

3 rows selected
```

XML函数的使用有以下限制：

- 输入值始终自动转换为char类型，因为用户自定义函数的输入值类型是预定义的。
- 输出均为char型（xmlagg除外）。输出长度有限（和页大小有关）。例如，若**DB_PgSiz**设置为8，则输出字符串长度不大于8056个字符，超大数据将被截断，同时无警告产生。
- 若输入为nchar，同时含有不能被转换为LCODE的字符，则可能产生无效输出。例如，原始数据存储在nchar字段且该数据包含繁体中文字符和日文字符，若将LCODE设置为2，则繁体中文字符不能正确转换。
- 结束标签后无新行生成，不提供特殊XML格式。

3.89 SET CONNECTION OPTIONS

设置连接选项（SET CONNECTION OPTIONS）命令可使用户通过SQL语句来设置连接选项。这对于当用户使用像Delphi这样的前端开发工具来连接数据库并且不能得到ODBC连接句柄时是很有用的，因为用户可以直接用SQL语句来设置连接选项。

下面将详细描述该命令所能设置的连接选项。这些选项分六类：*no value options*、*on/off options*、*number options*、*string options*、*symbol options*以及*transaction options*。

no_value_options.....没有取值的选项。

on_off_options……取值为*on*或*off*的选项。

string_options……..取值为一个单引号字符串，如‘*FOB*’。

number_options…… 取值为一个整数的选项。

symbol_options…… 取值为一组符号，例如 *{delete | close | preserve}* 的选项。

transaction_options…… 取指定事务行为的选项。

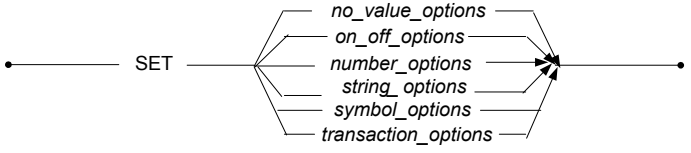


图 3-134 SET CONNECTION OPTIONS语法图

No Value选项

这类选项是没有值的，只是一些简单命令。

SET FLUSH

SET FLUSH是一个复制服务器选项，可将数据复制到从数据库中。

SET SYSINFO CLEAR

该命令用于清除系统信息，重置系统表SYSINFO。

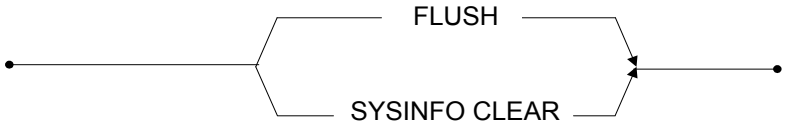


图 3-135 No Value Options语法图

ON/OFF选项

这类选项中的合法选项值是*ON*或*OFF*。有些选项值只能是ON或OFF，而有些选项两者都可以接受。

SET AUTOCOMMIT *ON/OFF*

该命令的作用是设置自动提交模式是激活（*ON*）还是关闭（*OFF*）。

SET BACKUP *OFF*

该命令的作用是将备份模式设置为non-backup，它等价于将**DB_BMode**的值设为0。

SET BKSVR CMP *ON/OFF*

该命令用来设置备份服务器的紧凑备份模式是开启（ON）还是关闭（OFF）。

SET BLOB BACKUP *ON*

该命令将备份模式设置为backup-data-and-blob模式，它等价于将**DB_BMode**设置为2。

SET BROWSE ON/OFF

将连接选项SQL_ATTR_TXN_ISOLATION设置为：SQL_TXN_READ_UNCOMMITTED (ON)或SQL_TXN_SERIALIZABLE

(OFF)。更多信息请参看*ODBC程序员参考手册*中函数*SQLGetInfo*的*SQL_DEFAULT_TXN_ISOLATION*选项。

SET DATA BACKUP *ON*

该命令可将备份模式设置为backup-data模式，它等价于将**DB_BMode**的值设为1。

SET FASTCOPY *ON/OFF*

该命令可设置客户端的连接属性，默认值设为***OFF***。每个连接到数据库的用户均有一个主属性，且各用户的设置互不影响。

SET FREE CATALOG CACHE *ON/OFF*

该命令可设置系统命令缓冲区是处于开启还是关闭状态。

SET ITCMD *ON/OFF*

设置开启/关闭隐式数据转换。

SET JOURNAL *ON/OFF*

只有DBA才能设置日志的开启或关闭状态。

SET LOADAUTOINDEX *ON/OFF*

该命令用于设置执行LOAD DB命令时是否载入所有索引。如果该选项值为ON，则执行LOAD DB命令时将载入所有索引；如果该选项值为OFF，则执行LOAD DB命令时将载入除自动索引外的所有索引。默认值是***OFF***。

SET LOAD SYSTEM DEFAULT *ON/OFF*

如果用户通过INSERT/UPDATE语句给字段赋值，那么在载入数据库的表时，可使用该命令设置含有SYSTEM DEFAULT属性的字段值是否会被覆盖。如果该选项值为ON，该字段值将会被更新为默认值；如果该选项为OFF，该字段值将会被更新为用户指定的值。默认值是***OFF***。

SET REMOVE SPACE PADDING *ON/OFF*

该命令用于设置是否自动删除一个字符串后面的空格。

SET STRING CONCAT *ON/OFF*

该选项可在字符串连接运算符 (||)中使用。如果开启这个选项，那么CHAR类型的数据库中的空格将被删除，如果该选项值为OFF，那么所有空格将继续保留。

SET SYSTEM DEFAULT *ON/OFF*

更新数据时，可使用该命令设置含有SYSTEM DEFAULT属性的字段值是否会被默认值覆盖。如果该选项值为ON，该字段值将会被更新为默认值；如果该选项为OFF，该字段值将会被更新为用户指定的值。默认值是***ON***。

SET SYSTEM INIT *ON/OFF*

只有DBA才能打开或关闭系统模式。在系统模式下可以创建系统表。

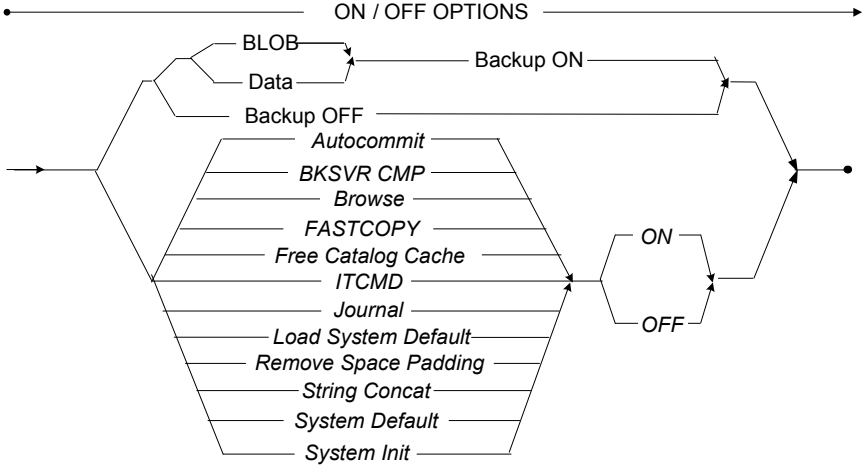


图 3-136 ON/OFF选项语法图

Number选项

这组选项的取值是一些整数，每个选项都有其合法的取值范围。

SET BKSVR JOURNAL FULL *NUMBER*

设置备份服务器的日志填充百分比，取值范围是0-100。

SET BKSVR PID *NUMBER*

设置备份服务器的进程ID，该选项的当前值必须为0。

SET DDB LOGIN TIMEOUT *NUMBER*

设置连接分布式数据库（DDB）时的登录超时值。

SET DDB LOCK TIMEOUT *NUMBER*

该选项用来设置连接分布式数据库（DDB）时的锁定超时值。

SET INPUT PARAM *N* AS CFILE | ASCII

该选项用在使用了参数的INSERT或UPDATE语句前面。如果您想将语句中的一个或多个参数绑定到客户文件中，就可以使用该选项。之后，为对应的参数输入的数据或语句中接下来的参数将被绑定到客户文件中。插入的数据必须是字符类型的数据，并且这个参数必须对应于LONG VARCHAR或LONG VARBINARY类型的字段。

用ALL选项来将所有的参数都绑定到客户文件上。CFILE选项是将参数绑定到客户文件时必须要用的。在SET INPUT PARAM语句的ASCII选项来重置DBMaster，作用是使所有参数都不会被绑定到客户文件上。

number.....指明参数列表中的哪个参数将被绑定到客户文件上。

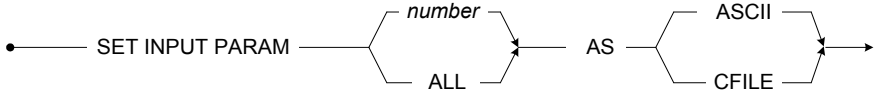


图 3-137 SET INPUT PARAM 选项语法图

➲ 例

这个例子中使用主变量将文件**dmconfig.ini**插入到字段**c3**中。

```
dmSQL> CREATE TABLE tb_attri (c1 INT, c2 INT, c3 LONG VARBINARY);
dmSQL> SET INPUT PARAM 3 AS CFILE;
dmSQL> INSERT INTO tb_attri VALUES (?,?,?);
dmSQL/Val> 2,2,'dmconfig.ini';
dmSQL/Val> end;
```

SET LOCK TIMEOUT *NUMBER*

该命令用来设置DBMaster在返回应用程序前等待锁资源的时间。如果number是个正值，则表示等待数秒；如果是0，则表示不等待，直接返回；如果是个负值，则表示一直等待，直到获得锁资源。

SET MAXTBROW *NUMBER*

检索表时，用该命令设置返回记录的最大值。如果number值为0或为负，则将返回所有查询结果。

SET RPSVR RETRY *NUMBER*

设置复制时，网络错误后重试的次数。

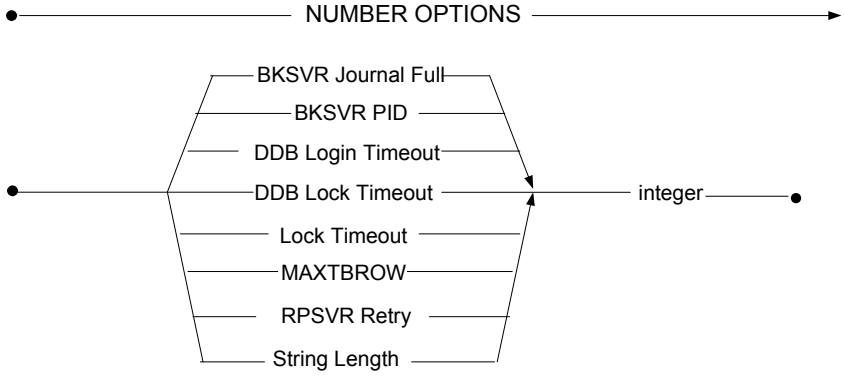


图 3-138 Number 选项语法图

String选项

这组选项的取值是单引号字符串。对于某些选项，其值必须符合一定的格式。

SET BKSVR PATH *STRING*

设置备份日志文件的路径。

SET DATE INPUT FORMAT {ALL | *STRING* }

设置DATE字段的输入格式。

日期字段有效的输入格式有：

格式	例子
'mm/dd/yy'	'02/18/99'
'mm-dd-yy'	'02-18-99'
'dd-mon-yy'	'18-Feb-99'
'mm/dd/yyyy'	'02/18/1999'
'dd/mon/yyyy'	'18/Feb/1999'
'dd-mon-yyyy'	'18-Feb-1999'
'dd.mm.yyyy'	'18.2.1999'

*表 3-2 (*yy/yyyy*: 年，*mm*: 月，*dd*:日)*

如果使用了ALL选项，则表示以上所有的日期格式都允许使用。

SET DATE OUTPUT FORMAT *STRING*

设置DATE字段的输出格式。输出格式见*SET DATE INPUT FORMAT*命令中的表3-2。

SET EXTNAME TO *STRING*

将系统文件对象的扩展名设置为*string*。

SET TIME INPUT FORMAT { ALL | *STRING* }

设置TIME字段的输入格式，ALL选项表示下面的所有格式都可以使用。

您也可以使用以下任一种格式作为时间字段的输入或输出格式。

格式	例子
'hh:mm:ss.fff '	22:10:20.30
'hh:mm:ss'	22:10:20
'hh:mm'	22:10
'hh'	22
'hh:mm:ss.fff tt'	10:10:20.30 PM
'hh:mm:ss tt'	10:10:20 PM
'hh:mm tt'	10:10 PM
'hh tt'	10 PM
'tt hh:mm:ss.fff '	PM 10:10:20.30
'tt hh:mm:ss'	PM 10:10:20
'tt hh:mm'	PM 10:10
'tt hh'	PM 10

表3-3（hh：*时*， mm：*分*，ss：*秒*，fff：*小数部分*， tt：*AM/PM*）

如果使用ALL选项则表示以上所有格式都可以作为TIME字段的输入格式。

SET TIME OUTPUT FORMAT *STRING*

设置TIME字段的输出格式。可能的输出格式和“SET TIME INPUT FORMAT”中的输入格式是相同的，参见表3-3。

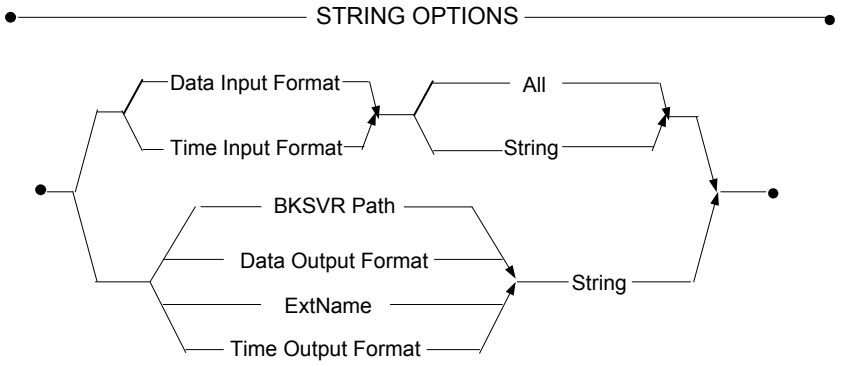


图 3-139 String 选项语法图

Symbol选项

这组选项中，所有选项的取值都为匹配ODBC符号的一组符号。更多信息请参看相关的ODBC连接选项。

SET CB MODE { *CLOSE* | *DELETE* | *PRESERVE* }

该命令用来设置提交事务时的游标动作，有关这三种模式的更多信息，请参看*ODBC程序员参考手册*中函数*SQLGetInfo*的*SQL_CURSOR_COMMIT_BEHAVIOR*选项。

SET CONCAT NULL RETURN { *NULL* | *STRING* }

在使用内置函数CONCAT或字符连接运算符(||)来连接字符串和空值时，您就可以使用该选项了，该选项的默认值是***NULL***。如果将选项设置为*NULL*，则任何字符串连接空值后都将返回空值。如果将选项设置为*STRING*，任何字符串连接空值后都将返回原来的字符串，因为空值在这里被看作是一个空的字符串。

SET DISCONNECT { *DISCONNECT* | *TERMINAT* | *WAIT* }

该命令用来设置SQLDisconnect()的动作。如果选项设置为 ***disconnect***，则立即断开与服务器的连接；***terminate***表示关闭数据库；如果选择 ***wait***，DBMaster将在服务器完全关闭数据库后，才返回SQLDisconnect( )。这是DBMaster的一个内部选项，开发工具可以调用SQLDisconnect( )来关闭数据库。

SET DFO DUPMODE { *COPY* | *NULL* }

在远程表的文件对象字段上执行***select into***时可以用这个选项来决定文件对象的复制。如果选项被设置成***null***，则FILE字段将被设为空（NULL），否则将把远程文件对象复制到本地表中。

SET FO TYPE { *BLOB* | *FILE* }

用哪种SQL数据类型来映射FILE字段。如果选择***FILE***，那FILE字段将被返回为SQL_FILE类型，否则将返回SQL_LONGVARBINARY类型。

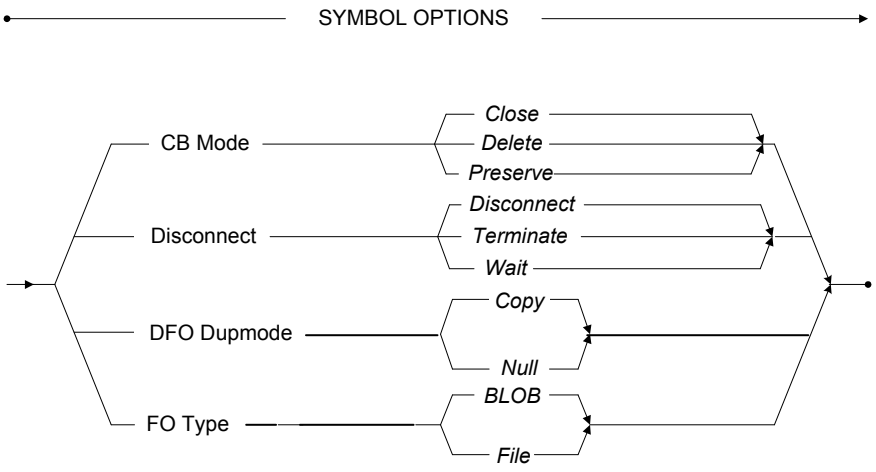


图 3-140 Symbol选项语法图

➲ **例1**

SET BKSVR PID

```
dmSQL> SET BKSVR PID 0;
```

➲ **例2**

SET BKSVR PATH

```
dmSQL> SET BKSVR PATH 'd:\data\backup';
```

➲ **例3**

SET DATE INPUT FORMAT

```
dmSQL> SET DATE INPUT FORMAT ALL;
dmSQL> SET DATE INPUT FORMAT 'yyyy/mm/dd';
```

➲ **例4**

SET DATE OUTPUT FORMAT

```
dmSQL> SET DATE OUTPUT FORMAT 'mm-dd-yy'; // result of DATE column will be like
12-31-99
```

➲ **例5**

SET DDB LOCK TIMEOUT

```
dmSQL> SET DDB LOCK TIMEOUT 20;    // timeout is 20
```

➲ **例6**

SET DDB LOGIN TIMEOUT

```
dmSQL> SET DDB LOGIN TIMEOUT 15;
```

下例将使用数据库***db1***和***db2***中的两个名为***t1***的表，下例中将有这两个表的定义。

➲ **例7**

SET DFO DUPMODE

```
dmSQL> CREATE TABLE t1 (c1 INT, c2 FILE);
```

接下来将***db2***作为***db1***的远程数据库。

➲ **例8**

SET DFO DUPMODE

```
dmSQL> SET DFO DUPMODE null;
```

在表***t1***中插入数据。

➲ **例9**

SET DFO DUPMODE

```
dmSQL> SELECT c1, c2 from DB2:SYSADM.t1 INTO t1;
```

在***db2***中表***t1***的字段***c2***执行SELECT INTO时，本地表***t1***的***c2***字段将为空值。

➲ **例10**

SET DFO DUPMODE

```
dmSQL> SET DFO DUPMODE copy;
```

在本地表***t1***中插入远程表***db2:t1***的查询结果，本地表的***c2***字段上新插入的记录将复制于远程表***db2:t1***的***c2***字段。

➲ **例11**

SET EXTNAME TO

```
dmSQL> SET EXTNAME TO 'FOB';
```

➲ **例12**

SET LOCK TIMEOUT

```
dmSQL> SET LOCK TIMEOUT 30 ;        // timeout is 30 seconds
dmSQL> SET LOCK TIMEOUT 0;          // always wait
dmSQL> SET LOCK TIMEOUT -5;         // always wait
```

➲ **例13**

SET MAXTBROW

```
dmSQL> SET MAXTBROW 10;    // return only first 10 tuples of data
dmSQL> SET MAXTBROW -3;    // return all tuples
```

➲ **例14**

SET SYSTEM INIT

```
dmSQL> SET SYSTEM INIT ON;
dmSQL> CREATE TABLE SYSTEM.t1 (c1 int);
```

➲ **例15**

SET TIME INPUT FORMAT

```
dmSQL> SET TIME INPUT FORMAT ALL;          // all formats accepted
dmSQL> SET TIME INPUT FORMAT 'hh:mm';      // 10:20
```

➲ **例16**

SET TIME OUTPUT FORMAT

```
dmSQL> SET TIME OUTPUT FORMAT 'hh:mm:ss';   // 10:20:55
```

Transaction选项

DBMaster可以将连接选项SQL_ATTR_TXN_ISOLATION设置为：SQL_TXN_READ_UNCOMMITTED (*ON*)或SQL_TXN_SERIALIZABLE(*OFF*)。SQL_TXN_REPEATABLE_READ或SQL_TXN_SERIALIZABLE。更多相关信息可参看*ODBC程序员参考手册*中函数*SQLGetInfo*的 *SQL_DEFAULT_TXN_ISOLATION*选项。

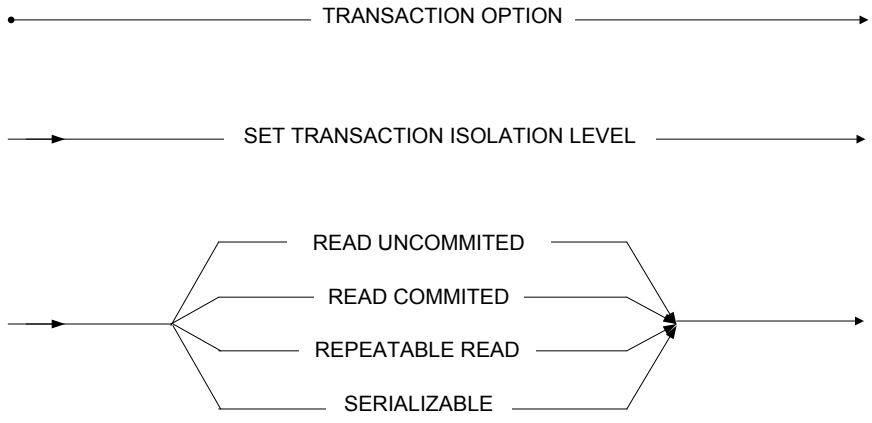


图 3-141 TRANSACTION 选项语法图

3.90 SET CLIENT_CHAR_SET

SET CLIENT_CHAR_SET命令用于指定数据库客户端的字符集编码设置。

在多语言数据库里，客户端能够使用多种编码来连接UTF-8数据库。所以客户端能够设置自己的字符集以区别服务器端。关键字**DB_LCode**用来设置服务器端的语言编码，**DB_CliLCODE**则用于设置客户端的字符集。另外，用户还可以通过命令**SET_CLIENT_CHAR_SET**来设置客户端的字符集。但是，通过该语法设置的字符集的有效范围仅是本次连接，一旦断开本次连接，本语法设置的字符集将变为无效。

如果用户需要知道在数据库服务器端或用户端设置了什么字符集，可以通过**UDF GETSYSINFO()**找回这些设置。

获取服务器端字符集的语法是 SELECT GETSYSINFO('LCODE');

获取客户端字符集的语法是 SELECT GETSYSINFO('CLILCODE');

client-character-set-string 可以在客户端设置字符集

DBMaster支持的可在客户端设置的字符集：

ASCII（英语）
BIG5（繁体中文）
Shift-JIS（日文 Shift-JIS + 半角）
GBK（简体中文）
ISO-8859-1（Latin1编码）
ISO-8859-2（Latin2编码）
ISO-8859-5（Cyrillic编码）
ISO-8859-7（Greek编码）
EUC-JP（日文）
GB18030（简体中文）
Unicode（UTF-8）

ISO-8859-{3,4,9,10,13,14,15,16}、KOI8-R、KOI8-U、KOI8-RU、CP{1250,1251,1252,1253,1254,1257}、CP{850,866}、Mac{罗马语、中

欧语、冰岛语、克罗地亚语、罗马尼亚语 }、Mac{西里尔语、乌克兰语、希腊语、土耳其语 }、Macintosh（欧语系）

ISO-8859-{6,8}、CP{1255,1256}、CP862, Mac{希伯来语、阿拉伯语}（闪族语）

CP932、ISO-2022-JP、ISO-2022-JP-2、ISO-2022-JP-1（日文）

EUC-CN、CP936、EUC-TW、CP950（中文）

EUC-KR、CP949、JOHAB（韩语）

Georgian-Academy、Georgian-PS（格鲁吉亚语）

KOI8-T（塔吉克）

PT154（哈萨克语）

TIS-620、CP874、MacThai（泰语）

MuleLao-1、CP1133（老挝语）

VISCII、TCVN、CP1258（越南语）

图 3-142 SET CLIENT CHARACTER SET语法图

➲ **例**

设置客户端字符集为 BIG5。

```
dmSQL> SET CLIENT_CHAR_SET 'BIG5';
```

3.91 SET ERRMSG_CHAR_SET

SET ERRMSG_CHAR_SET命令用于设置数据库客户端的错误信息输出字符集。

在多语言数据库里，客户端能够设置自己的错误信息输出字符集。

此命令必须指定为这种形式：语言*[_地区][.编码]*，其中**语言**字符串参照ISO-639标准，必须为小写字符，**地区**字符串参照ISO-3166标准，必须为大写字符，**编码**字符串则是DBMaster支持的字符集名称。对于有多个地域区别的语言，应该指出其地域性区别。例如，zh_CN或者zh_TW，仅仅是zh则是无效的。

但是，通过该语法设置的错误信息编码的有效范围仅是本次连接，一旦断开本次连接，本语法设置的编码将变为无效。

存储错误信息的文件存放在*dbmaster/5.4/shared/locale/locale_LANG/*路径下。

要获取客户端错误信息集，用户可以执行命令

SELECT GETSYSINFO('ERRLCODE');

目前，DBMaster只支持四种语言的客户端错误信息：英文、简体中文、繁体中文和日文。可设置的错误信息输出编码可以是四种语言en、zh_CN、zh_TW和ja，或它们与相应字符集的组合，如：

en
en.ASCII
en.ISO-8859-1
en.ISO-8859-2
en.ISO-8859-5
en.ISO-8859-7
en.UTF-8

ja
ja.SHIFT-JIS

ja.UTF-8
ja.EUC-JP

zh_CN
zh_CN.GBK
zh_CN.UTF-8
zh_CN.GB18030

zh_TW
zh_TW.BIG5

zh_TW.UTF-8

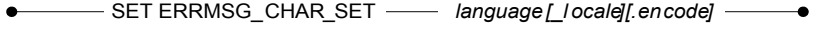


图 3-143 SET ERRMSG_CHAR_SET语法图

➲ **例1**

以下设置将客户端错误信息输出编码设置为**'ja'**。

```
dmSQL> SET ERRMSG_CHAR_SET 'ja';
```

➲ **例2**

以下设置将客户端错误信息输出字符集设置为**'ja'**和字符集**'EUC_JP'**。

```
dmSQL> SET ERRMSG_CHAR_SET 'ja.EUC-JP ';
```

➲ **例3**

以下设置将客户端错误信息输出字符集设置为**'ja'**和字符集**'UTF-8'**。

```
dmSQL> SET ERRMSG_CHAR_SET 'ja.UTF-8';
```

3.92 SUSPEND SCHEDULE

SUSPEND SCHEDULE命令可用来暂停异步表复制的复制计划。在复制计划恢复之前，本地数据库不会尝试连接远程数据库。只有本地表的所有者、DBA、SYSDBA或SYSADM才可以执行这条指令。

您可以使用SUSPEND SCHEDULE命令来暂停异步表复制的复制计划，也可以使用RESUME SCHEDULE命令来恢复复制计划。

remote_database_name....要暂停复制计划的远程数据库名。

图 3-144 SUSPEND SCHEDULE语法图

➲ **例**

下例将暂停远程数据库**DivOneDb**的复制计划。

```
dmSQL> SUSPEND SCHEDULE FOR REPLICATION TO DivOneDb;
```

3.93 SYNC AUTO INDEX

SYNC AUTO INDEX命令可用来即时唤醒自动索引后台程序的处理机制。只有DBA、SYSDBA或SYSADM才可以在自动索引后台程序启动后执行这条指令。

只有在自动索引后台程序启动后且关键字AUTOCOMMIT设置为ON时，用户才可以执行该命令。

图 3-145 SYNC AUTO INDEX语法图

➲ **例**

下例在设置**AUTOCOMMIT**为**ON**后唤醒自动索引后台程序。

```
dmSQL> SYNC AUTO INDEX;
```

3.94 SYNCHRONIZE SCHEDULE

SYNCHRONIZE SCHEDULE命令可使远程数据库中的数据与本地数据库中保持同步，而不必等待下一次复制计划。只有本地表的所有者、DBA、SYSDBA或SYSADM才可以执行这条指令。

如果您想在异步表复制中保持本地和远程表的数据一致性，那么您就可以使用SYNCHRONIZE SCHEDULE命令。

remote_database_name….远程数据库名。

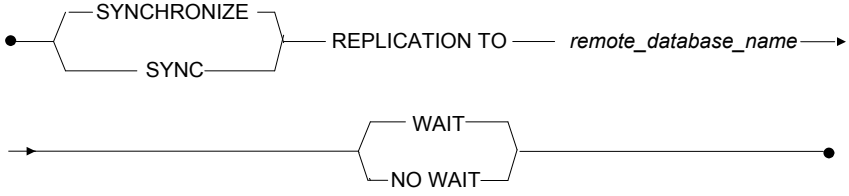


图 3-146 SYNCHRONIZE SCHEDULE语法图

➲ **例**

下面的命令将保持远程数据库**DivOneDb**的复制计划的同步性。

```
dmSQL> SYNCHRONIZE REPLICATION TO DivOneDb;
```

3.95 UNLOAD STATISTICS

UNLOAD STATISTICS命令可将数据库中的统计值载出到外部ASCII文本文件中。您可以编辑这个文件，然后再将修改过的统计值载入到数据库中。只有DBA、SYSDBA或SYSADM才可以执行该命令。

您可以为整个数据库或者一个或多个表载入统计值。每个表中都应指明是载入表统计值还是字段统计值，或是索引统计值，亦或是三种统计值的结合。

DBMaster记录的表统计值有：数据页数、记录数、表中所有记录的平均字节数等信息。DBMaster记录的字段统计值包括：具有唯一值的字段数量、字段的平均字节数、所有抽样值中的次小字段值和次大字段值等信息。DBMaster记录的索引数据统计值包括：索引页的页数、索引数阶层的数目、索引树叶的页数目、唯一键值数、每个键所含的数据页数、索引的聚集数等信息。

object_list.....要载出统计值的数据库对象列表。

file_name .....载出的统计值将放在哪个ASCII文本文件中。

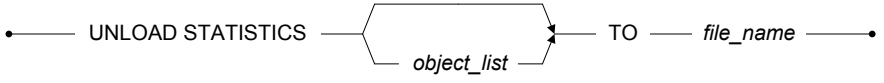


图 3-147 UNLOAD STATISTICS语法图

UNLOAD STATISTICS对象列表

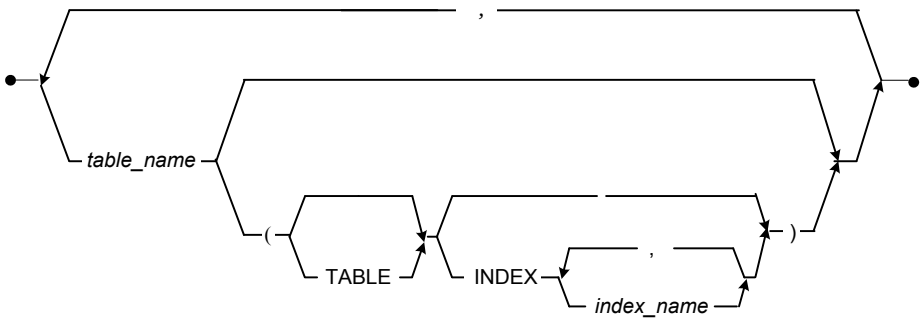


图 3-148 UNLOAD STATISTICS Object List语法图

➲ **例**

下例是将数据库中的所有统计值（**STATISTICS**）载出到**stat.dat**文件中。

```
dmSQL> UNLOAD STATISTICS TO stat.dat;
```

3.96 UPDATE

UPDATE命令用以更新表中的记录，但系统表中的记录不能用此命令来更新。只有表的拥有者、DBA、SYSDBA、SYSADM或在整张表或指定字段上拥有UPDATE权限的用户才可以使用该命令。

当您在更新一个字段时，为这个字段新设置的值必须满足字段的约束条件和参照完整性。更新时，您可以使用DEFAULT关键字来将字段值设为默认值。

table_name .....…..需要修改记录所在的表名。

column_name ....…..需要修改的字段名。

literal.....…..用这个值来修改字段。

expression.....…..用这个表达式返回的值来修改字段。

constant .....…..用这个常量来修改字段。

search_condition.....修改记录时所需要满足的条件。

cursor_name .....…..该游标表明修改的位置（这些游标只有在ODBC编程时才能使用）。

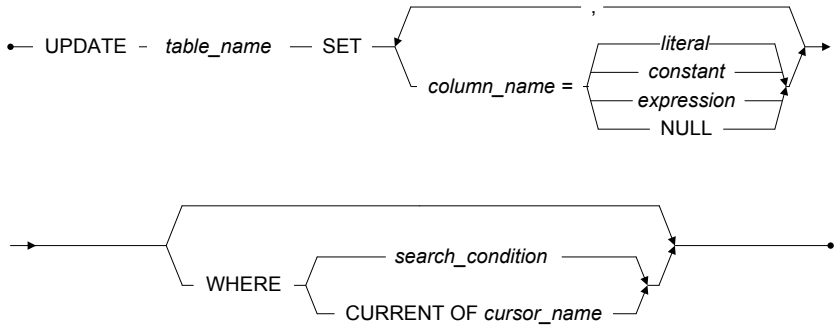


图 3-149 UPDATE语法图

➲ **例1**

下例将说明如何更新**Employeesinfo**表，修改所有名为**Chris**的员工（**employeesinfo**）的工资。

```
dmSQL> UPDATE Employeesinfo SET Salary = 5000 WHERE FName = 'Chris';
```

➲ **例2**

下例将说明如何将所有名为**Chris**的员工（**employees**）的工资提高**10%**。

```
dmSQL> UPDATE Employees SET Salary = Salary*1.10 WHERE Name = 'Chris';
```

3.97 UPDATE STATISTICS

UPDATE STATISTICS命令用于更新数据库的统计值。让统计值反映数据库的当前信息对于提高查询效率是很有帮助的。只有对象的所有者、DBA、SYSDBA或SYSADM才有权执行该指令。

您可以更新整个数据库或者一个或多个表的统计值。在对表使用该命令时，需指定更新的是表统计值还是字段统计值，或是索引统计值，亦或是以上三种统计值的结合。此外，可以用关键字SAMPLE来指定抽样比率，它的值是介于1到100之间的整数。

DBMaster记录的索引统计值包括：索引页的页数、索引数阶层的数目、索引树叶的页数目、唯一键值的数目、每个键所含的数据页数目、索引的聚集数。

ALL：强制更新所有模式对象的统计值。

SAMPLE：采样率以百分比的形式表示，介于1至100之间的整数。

object_list.....需要更新统计值的数据库对象列表。

number.....更新统计值时的抽样比率。

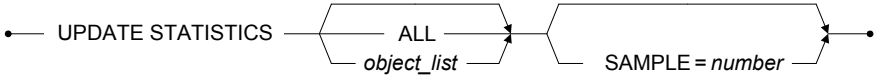


图 3-150 UPDATE STATISTICS语法图

UPDATE STATISTICS对象列表

DBMaster记录的表统计值包括：数据页的页数、记录数、表中所有抽样记录的平均字节数等信息。

DBMaster同时还记录具有唯一值的字段的数量、字段的平均字节数、所有抽样值中次小字段值和次大字段值等信息。

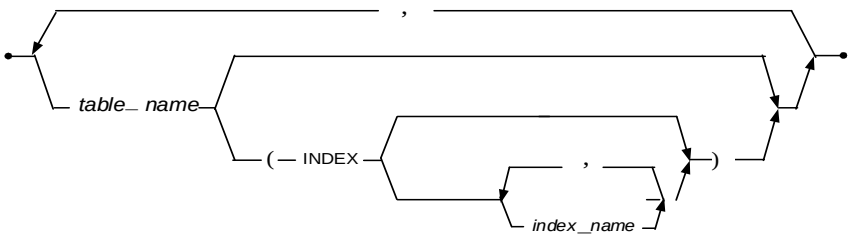


图 3-151 UPDATE STATISTICS 对象列表语法图

➲ **例1**

下例是更新数据库中的所有统计值（**STATISTICS**），并且预设抽样比率为**30%**。

```
dmSQL> UPDATE STATISTICS SAMPLE = 30;
```

➲ **例2**

下例是更新表**table1**上的所有统计值（**STATISTICS**）。

```
dmSQL> UPDATE STATISTICS table1 SAMPLE = 50;
```

➲ **例3**

下例是更新表**table1**的字段**c1**和索引**ixl**上的统计值（**STATISTICS**）。

```
dmSQL> UPDATE STATISTICS table1 c1 (INDEX (ix1));
```

➲ **例4**

下例是更新表**table1**上所有索引的**统计值**。

```
dmSQL> UPDATE STATISTICS table1 (INDEX);
```

➲ **例5**

下例是强制更新数据库所有对象的**统计值**。

```
dmSQL> UPDATE STATISTICS ALL;
```

3.98 UPDATE STATISTICS SET

当以**表设置**模式启动（即**DB_StMod=1**）时，UPDATE STATISTICS SET命令用于为统计更新后台程序的表设置更新统计模式和采样率。

每个表的更新统计模式和统计采样率存储在系统表SYSTABLE中。其中字段UPD_STS_MODE用于存储表统计模式，字段UPD_STS_SAMPLE用于存储表统计采样率。

如果用户通过执行SQL语句UPDATE STATISTICS SET为每个表设置了更新统计选项，就会出现以下4个过滤条件：

- 如果是一个新表，即此表从未执行过更新统计，那么该表会自动执行更新统计。
- 如果表的更新总页数不足20页，那么自动执行更新统计。
- 如果表的更新总页数超过20页，且自最后一次自动更新统计以来，被修改的页数超过2页，将执行自动更新统计。
- 如果此表超过10天未执行过更新统计，则执行自动更新统计。

table_name .....表名

mode_value .....表更新统计方式

0：表的采样率使用 **dmconfig.ini**中**DB_StsSp**设定的值，但需要考虑以上4种过滤条件，默认值为***0***。

1：表的采样率使用***sample_value***设置的表更新统计采样率值，但需要考虑以上4种过滤条件。

2：表的采样率使用***sample_value***设置的表更新统计采样率值，不需要考虑以上4种过滤条件。

sample_value.....表更新统计采样率

-1：智能获取采样率

0：不更新统计

0 ~ 100：表更新统计采样率值，默认值为***100***。

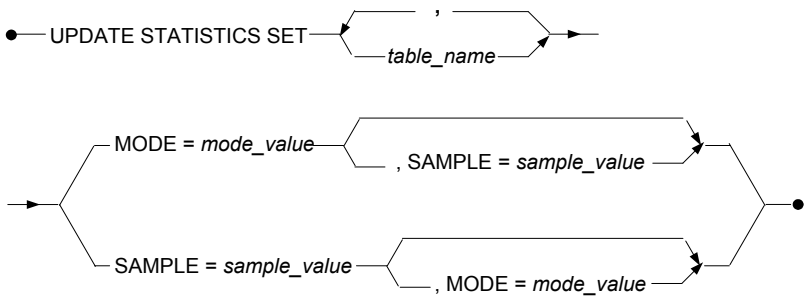


图 3-152 UPDATE STATISTICS SET语法图

➲ **例1**

设置有效表**jeff.tb_staff**的更新统计方式和采样率：

```
dmSQL> UPDATE STATISTICS SET jeff.tb_staff MODE = 1,  SAMPLE = 80;
dmSQL> SELECT TABLE_NAME, TABLE_OWNER, UPD_STS_MODE, UPD_STS_SAMPLE FROM
SYSTABLE;
         TABLE_NAME               TABLE_OWNER     UPD_STS_MODE   UPD_STS_SAMPLE
======================== ======================== ============== ================
TB_STAFF                 JEFF                                1               80
1 rows selected
```

➲ **例2**

设置有效表**jeff.tb_staff**和**jim.tb_salary**的更新统计方式和采样率：

```
dmSQL> UPDATE STATISTICS SET jeff.tb_staff, jim.tb_salary MODE = 1, SAMPLE = 60;
dmSQL> SELECT TABLE_NAME, TABLE_OWNER, UPD_STS_MODE, UPD_STS_SAMPLE FROM
SYSTABLE;
         TABLE_NAME               TABLE_OWNER     UPD_STS_MODE   UPD_STS_SAMPLE
======================== ======================== ============== ================
TB_STAFF                 JEFF                                1               60
TB_SALARY                JIM                                 1               60
2 rows selected
```

3.99 UPDATE TABLESPACE STATISTICS

UPDATE TABLESPACE STATISTICS命令可用来更新表空间的统计信息。保持统计信息反映数据库的当前状态，这有利于提高表空间中的查询效率。只有DBA、SYSDBA或SYSADM才有权执行该命令。

要执行这条命令来更新表空间的统计值，DBMaster必须更新表空间以及相关文件的统计值。

DBMaster记录的表空间统计值包括：数据页数、空闲页数、帧数以及空闲帧数等信息。

DBMaster记录的文件统计值包括：数据页或空闲页数，帧或空闲帧数。

object_list.....要更新统计值的数据库对象列表。

图 3-153 UPDATE TABLESPACE STATISTICS语法图

➲ **例**

下例将更新表空间**DEFTABLESPACE**的统计值（**STATISTICS**）。

```
dmSQL> UPDATE TABLESPACE STATISTICS DEFTABLESPACE;
```

4 函数

DBMaster提供大量的内置函数（built-in functions），同时也允许用户自己创建用户自定义函数（UDF），详细信息请参考如下章节。

4.1 内置函数

DBMaster提供大量的内置函数（built-in functions）。这些函数可用于结果集中的某些字段上或结果集中某些限定记录的字段上。本章按函数类型列出了所有内置函数。每个函数的自变量和返回值都在随后的函数语法图中列出，包括函数的名称、数据类型和取值。

内置函数的类型有：

- 字符串类型函数（String functions）
- 数值类型函数（Numeric functions）
- 日期和时间类型函数（Date and time functions）
- 系统函数（System functions）

4.1.1. ABS

ABS函数返回某个数*number*的绝对值，类型是双精度的浮点（floating-point）类型数据。

number.....Double类型：为哪个数据返回绝对值。

Return value.....Double类型：数据number的绝对值。

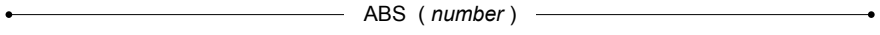


图 4-1 ABS函数语法图

➲ **例**

下例的函数将返回**3.14000000000000e+012**。

```
ABS(-3.14E12)
```

4.1.2. ACOS

ACOS函数将返回一个双精度浮点类型的数据，其值为自变量*number*的反余弦值（arc cosine），返回值必须是介于0到π之间的弧度。

number..... Double类型：为哪个数据求反余弦值。

Return value ..... Double类型：数据number的反余弦值。

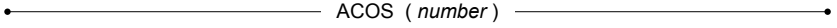


图 4-2 ACOS函数语法图

➲ **例**

下例的函数将返回**1.04719755119660e+000**。

```
ACOS(0.5)
```

4.1.3. ADD_DAYS

ADD_DAYS函数返回的是某个日期加上*number*天数后的日期值，自变量*number*可以是一个负值。

date .....Date类型：原始日期。

number .....Integer类型：增加的天数。

Return value.....Date类型：在原始日期上增加number天后的结果。

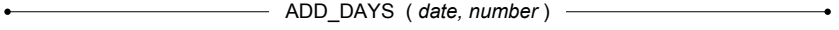


图 4-3 ADD_DAYS函数语法图

➲ **例1**

下例返回的日期是**1999-03-01**。

```
ADD_DAYS('1999-02-24', 5)
```

➲ **例2**

下例返回的日期是**2000-02-29**。

```
ADD_DAYS('2000-02-24', 5)
```

4.1.4. ADD_HOURS

ADD_HOURS函数返回的是某个时间加上*number*小时后的时间值，自变量*number*可以是一个负值。

time ..... Time类型：原来的时间。

number..... Integer类型：要增加几小时。

Return value ..... Time类型：在time上增加number小时后的结果。

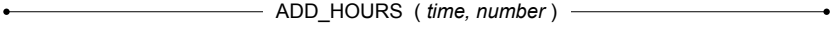


图 4-4 ADD_HOURS函数语法图

➲ **例1**

下例返回的时间是**20:11:12**。

```
ADD_HOURS('10:11:12', 10)
```

➲ **例2**

下例返回的时间是**22:11:12**。

```
ADD_HOURS('10:11:12', -12)
```

4.1.5. ADD_MINS

ADD_MINS函数返回的是某个时间加上*number*分钟后的时间值。自变量*number*可以是一个负值。

time .....Time类型：原来的时间。

number.....Integer类型：要增加几分钟。

Return value.....Time类型：在time上增加*number*分钟后的结果。

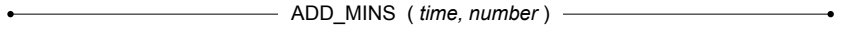


图 4-5 ADD_MINS函数语法图

➲ **例1**

下例返回的时间是**10:21:12**。

```
ADD_MINS('10:11:12', 10)
```

➲ **例2**

下例返回的时间是**09:59:12**。

```
ADD_MINS('10:11:12', -12)
```

4.1.6. ADD_MONTHS

ADD_MONTHS函数返回的是某个日期加上*number*个月后的日期值。自变量*number*可以是个负值。

date ..... Date类型：原来的日期。

number..... Integer类型：要增加几个月。

Return value ..... Date类型：在date上增加number个月后的日期值。

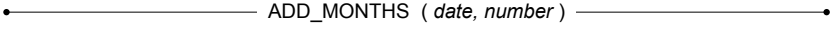


图 4-6 ADD_MONTHS函数语法图

➲ **例1**

下例返回的日期是**1999-07-24**。

```
ADD_MONTHS('1999-02-24',5)
```

➲ **例2**

下例返回的日期是**2001-01-01**。

```
ADD_MONTHS('2000-01-01',12)
```

4.1.7. ADD_SECS

ADD_SECS函数的返回值是某个时间加上*number*秒后的时间值，自变量*number*可以为负值。

time .....Time类型：原来的时间。

number.....Integer类型：要在原来的时间上增加几秒钟。

Return value.....Time类型：在time上增加number秒后的结果。

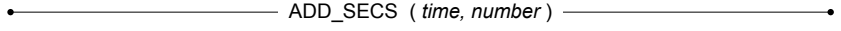


图 4-7 ADD_SECS函数语法图

➲ **例1**

下例返回的时间是**10:11:22**。

```
ADD_SECS('10:11:12',10)
```

➲ **例2**

下例返回的时间是**10:10:52**。

```
ADD_SECS('10:11:12', -20)
```

4.1.8. ADD_YEARS

ADD_YEARS函数的返回值是某个日期加上*number*年后的日期值，自变量 *number*可以为负值。

date ..... Date类型：原来的日期。

number..... Integer类型：要增加几年。

Return value ..... Date类型：在date上增加number年后的日期。

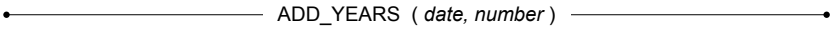


图 4-8 ADD_YEARS函数语法图

➲ **例1**

下例返回的日期是**2001-03-04**。

```
ADD_YEARS('1999-03-04', 2)
```

➲ **例2**

下例返回的日期是**1995-02-28**。

```
ADD_YEARS('2000-02-29', -5)
```

4.1.9. ASCII

ASCII函数返回一个字符串首字符的ASCII码值。如果该字符串中没有字符，那么这个函数将返回0（NULL）。使用该函数时，如果没有为自变量指定字符串，函数将返回错误信息。

string .....String类型：函数将获取该串首字符的ASCII码。

Return value.....Integer类型：string首字符的ASCII码值。

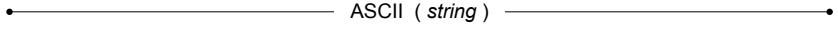


图 4-9 ASCII 函数语法图

➲ **例1a**

下面的函数将返回字符“**A**”的**ASCII**码**65**。

```
ASCII('A')
```

➲ **例1b**

下面的函数将返回字符串“**ABC**”中首字符“**A**”的**ASCII**码**65**。

```
ASCII('ABC')
```

➲ **例2a**

下面的函数将返回字符“**a**”的**ASCII**码**97**。

```
ASCII('a')
```

➲ **例2b**

下面的函数将返回字符串“**abc**”中首字符“**a**”的**ASCII**码**97**。

```
ASCII('abc')
```

➲ **例3a**

下面的函数将返回字符“**1**”的**ASCII**码**49**。

```
ASCII('1')
```

➲ **例3b**

下面的函数将返回字符“**!**”的**ASCII**码**33**。

```
ASCII('!')
```

4.1.10. ASIN

ASIN函数将返回一个双精度的浮点类型的数据，其值为自变量*number*的反正弦（arc sine）值，返回值的取值范围在**-π/2**到**π/2**之间。

number.....Double类型：函数将返回这个数字的反正弦值。

Return value.....Double类型：number的反正弦值。

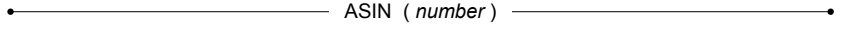


图 4-10 ASIN函数语法图

➲ **例**

下面的函数将返回0.5的反正弦值：5.23598775598299e-001。

```
ASIN(0.5)
```

4.1.11. ATAN

ATAN函数将返回一个双精度的浮点类型数据，其值是number的反正切值（arc tangent），返回值的取值范围是**-π/2**到**π/2**。

number..... Double类型：函数将返回这个数的反正切值。

Return value ..... Double类型：number的反正切值。

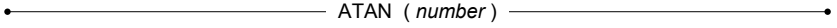


图 4-11 ATAN函数语法图

➲ **例**

下面的函数将返回0.5的反正切值**4.63647609000806e-001**。

```
ATAN(0.5)
```

4.1.12. ATAN2

ATAN2函数返回一个双精度的浮点类型数据，其值是***x*/*y***的反正切值。返回值的范围在-π到π之间。

x .....Double类型：函数将返回两个数比值的反正切，x表示比值的分子。

y .....Double类型：函数将返回两个数比值的反正切，y表示比值的分母。

Return value.....Double类型：***x*/*y***的反正切值。

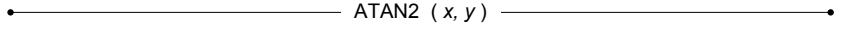


图 4-12 ATAN2函数语法图

➲ **例**

下面的函数将返回*x*/*y*（0.1/0.2）的反正切值**4.63647609000806e-001**。

```
ATAN2(0.1, 0.2)
```

4.1.13. ATOF

ATOF函数的返回值类型是一个双精度的浮点类型数据，其值为*string*参数代表的值。

string ..... String类型：这个函数将string串转换成一个双精度的浮点类型的数据。

Return value ..... Double类型：string串所代表的数值。

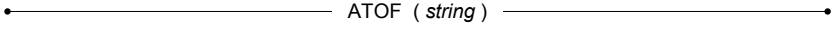


图 4-13 ATOF函数语法图

➲ **例1**

下面的函数将返回一个双精度的浮点数据。其值为字符串“**-12.34**”所代表的数值：**-1.23400000000000e+001**。

```
ATOF('-12.34')
```

➲ **例2**

下面的函数将返回一个双精度的浮点数据。其值为字符串“**-12.34E34**”所代表的数值：**-1.23400000000000e+035**。

```
ATOF('-12.34E34')
```

4.1.14. BLOBLEN

BLOBLEN函数将返回一个BLOB数据所占的字节数。请注意，BLOBLEN返回的最大长度只能是(2^{31}-1)B,即使BLOB实际大小可能大于或等于2^{31}B。BLOBLEN函数可以返回CLOB数据、BLOB数据、NCLOB数据甚至是文件类型对象所占的字节数。

object.....BLOB类型：源BLOB。

Return value.....Integer类型：源BLOB数据所占的字节数。

图 4-14 BLOBLEN函数语法图

➲ **例**

下面的函数将返回BLOB数据“**content**”所占的字节数。

```
BLOBLEN(content)
```

4.1.15. BLOBLENEX

BLOBLENEX函数将输入的BLOB的数据长度返回为一个Decimal类型的值。BLOBLENEX函数可以获取CLOB、BLOB以及NCLOB和FILE类型的数据长度。与BLOBLEN函数不同，即使BLOB大小超过2^{31}B，它也将返回BLOB大小的正确值。

object ..... BLOB类型：BLOB源

Return value ..... Decimal类型：获得的BLOB 源中的BLBO类型数据的长度

图 4-15 BLOBLENEX函数语法图

➲ **例**

下面的函数将返回BLOB数据“**content**”的长度：

```
BLOBLENEX(content)
```

4.1.16. CEILING

CEILING函数返回的是一个等于或大于*number*的整数，这个数是一个双精度的浮点类型数据。

number.....Double类型：函数将返回这个数的等价值或大于这个数的最小整数。

Return value.....Double类型：大于number的最小整数。

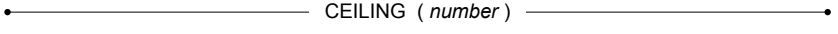


图 4-16 CEILING函数语法图

➲ **例1**

下面函数将返回大于12.3的最小整数：**1.30000000000000e+001**。

```
CEILING(12.3)
```

➲ **例2**

下面函数将返回大于-12.3的的最小整数：**-1.20000000000000e+001**。

```
CEILING(-12.3)
```

4.1.17. CHAR

CHAR函数返回的是ASCII码为*number*的字符。其中*number*必须是有效的ASCII码，取值范围为0到255。否则将是无效的ASCII码，也就不能使用CHAR函数。如果CHAR函数的自变量是一个无效的ASCII码，那么函数将返回一个错误或无效的结果。如果没有在函数中提供*number*自变量，函数将返回错误信息。

number..... Integer类型：字符的ASCII码。

Return value ..... String类型：ASCII码number所代表的字符。

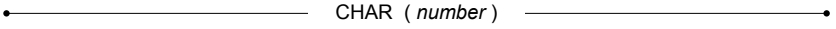


图 4-17 CHAR 函数语法图

➲ **例1**

下面的函数将返回ASCII码为**65**的字符“**A**”。

```
CHAR(65)
```

➲ **例2**

下面的函数将返回ASCII码为**97**的字符“**a**”。

```
CHAR(97)
```

➲ **例3**

下面的函数将返回ASCII码为**49**的字符“**1**”。

```
CHAR(49)
```

➲ **例4**

下面的函数将返回ASCII码为**33**的字符“**!**”。

```
CHAR(33)
```

4.1.18. CHAR_LENGTH

CHAR_LENGTH函数将返回字符串*string*中所包含的字符个数，但不包括字符串尾的空格以及字符串结束符。如果没有在函数中提供*string*自变量，函数将返回错误信息。

string .....String类型：要获得字符个数的字符串。

Return value.....Integer类型：字符串最左边的字符个数。

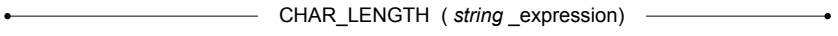


图 4-18 CHAR_LENGTH函数语法图

➲ **例**

下面函数返回的结果是“**4**”。

```
dmSQL> SELECT CHAR_LENGTH(' abc   ');
CHAR_LENGTH(' ABC   ')
============================
                          4
```

4.1.19. CHARACTER_LENGTH

CHARACTER_LENGTH函数返回字符串*string*中所包含的字符数，但不包括字符串尾的空格以及字符串结束符。如果没有在函数中提供*string*自变量，函数将返回错误信息。

string ..... String类型：要获得字符数的字符串。

Return value ..... Integer类型：字符串最左边的字符数。

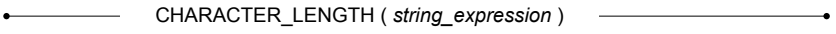


图 4-19 CHARACTER_LENGTH函数语法图

➲ **例**

下面函数返回的结果是“**4**”。

```
dmSQL> SELECT CHARACTER_LENGTH(' abc   ');
CHARACTER_LENGTH(' ABC   ')
============================
                          4
```

4.1.20. CHECKMEDIAFORMAT

CHECKMEDIAFORMAT函数可用来核查BLOB内容是否与指定的媒体类型相匹配。

blob .....要检查的字段名。

Media format:.....String类型：指定媒体格式。支持的格式有：DOC、XLS、 PPT、HTM、XML和PDF。

Return value.....如果字段中的记录与媒体数据类型相匹配，函数将返回True。

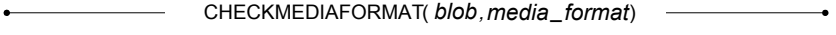


图 4-20 CHECKMEDIATYPE函数语法图

➲ **例**

下例中检查BLOB字段的类型是否为DOC格式。

```
CHECKMEDIAFORMAT(wordcol, 'DOC')
```

4.1.21. CONCAT

CONCAT函数将返回字符串*string1*和*string2*连接后的字符串。结果字符串的前部分是字符串*string1*的内容，而后部分是*string2*的内容。如果在使用函数时没有提供*string1*和*string2*，那么函数都将返回错误信息。

如果自变量的两个字符串中的任一个为NULL，DBMaster将按以下规则来决定返回值。

使用内置函数CONCAT或连接运算符(||) 来将字符串和空值连接时都将返回*NULL*。如果您希望返回字符串而不是空值的话，您必须将SET CONCAT NULL RETURN选项设置为STRING。如果一个空值连接一个空值，那么不管SET CONCAT NULL RETURN的值如何，都将返回空值。

string1 ..... String类型：出现在结果字符串前部分的字符串。

string2 ..... String类型：出现在结果字符串后部分的字符串。

Return value ..... String类型：字符串*string1*和*string2*的连接结果。

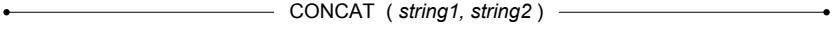


图 4-21 CONCAT函数语法图

➲ **例1**

下面的函数将返回“**master plan**”，请注意第一个字符串后面有空格。

```
CONCAT('master ', 'plan')
```

➲ **例2**

下面的函数将返回“**mastermind**”。

```
CONCAT('master', 'mind')
```

4.1.22. COS

COS函数返回的是*number*的余弦（cosine）值。*Number*是一个弧度，返回值是一个双精度的浮点类型数据。

number.....Double类型：函数将求这个数的余弦值。

Return value.....Double类型：*number*的余弦值。

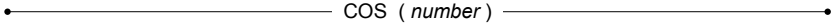


图 4-22 COS函数语法图

➲ **例**

下面函数返回0.5的余弦值：**8.77582561890373e-001**。

```
COS(0.5)
```

4.1.23. COSH

COSH函数将返回*number*的双曲余弦值。其中*number*是一个弧度，返回值是一个双精度的浮点类型数据。

number..... Double类型：函数将求这个数的双曲余弦值。

Return value ..... Double类型：*number*的双曲余弦值。

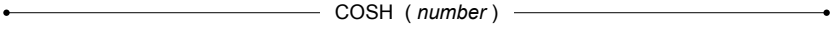


图 4-23 COSH函数语法图

➲ **例**

下面的函数将返回*0.5*的双曲余弦值：**1.12762596520638e+000**。

```
COSH(0.5)
```

4.1.24. COT

COT函数将返回*number*的余切值，其中*number*是一个弧度，而返回值是一个双精度的浮点类型数据。

number.....Double类型：函数将求解这个数的余切值。

Return value.....Double类型：number的余切值。

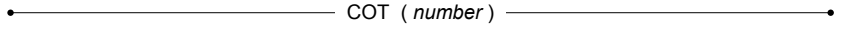


图 4-24 COT函数语法图

➲ **例**

下面的函数将返回*0.5*的余切值：**1.83048772171245e+000**。

```
COT(0.5)
```

4.1.25. CURDATE

CURDATE函数返回当前日期。

Return value ..... Date类型：当前日期。

图 4-25 CURDATE函数语法图

➲ **例**

下面函数的返回值是当前日期。

```
CURDATE()
```

4.1.26. CURRENT_DATE

CURRENT_DATE函数将以DBMaster的默认的date/time/timestamp格式返回当前日期。

*Return value.....*DATE类型：当前日期。

图 4-26 CURRENT_DATE函数语法图

➲ **例1**

下例介绍了该函数的几种用法，您可以直接向表中插入该函数，也可以把它作为SELECT语句中的返回值。

```
dmSQL> INSERT INTO t1 VALUES (CURRENT_DATE);
dmSQL> SELECT CURRENT_DATE;
dmSQL> SELECT c1 FROM t1 WHERE c2 = CURRENT_DATE;
```

➲ **例2**

下例首先在表中插入一条记录，记录中包括有CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP以及CURRENT_USER字段，然后显示记录值并更新表。

```
dmSQL> INSERT INTO sql99t5 VALUES (CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP,
CURRENT_USER);
1 row inserted

dmSQL> SELECT CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER;

CURRENT_TIME  CURRENT_D*    CURRENT_TIMESTAMP            CURRENT_USER
============= ========== ======================= ==========================
16:53:09      2001-09-26 2001-09-26 16:53:09      SYSADM

dmSQL> UPDATE sql99t5 SET c1 = CAST(CURRENT_TIMESTAMP AS CHAR(20)),
                  c2 = CURRENT_DATE,
```

```
                    c3 = CURRENT TIME,
                    c4 = CURRENT TIMESTAMP WHERE c1 = CURRENT USER;
1 row updated
```

4.1.27. CURRENT_TIME

CURRENT_TIME函数将返回当前时间，并以DBMaster的默认时间输出格式来显示。

*Return value.....*TIME类型：当前时间。

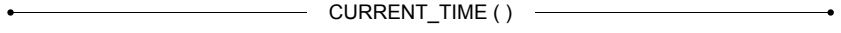


图 4-27 CURRENT_ TIME函数语法图

➲ **例1**

下例介绍该函数的几种用法。您可以直接向表中插入该函数，也可以把它作为SELECT语句中的返回值。

```
dmSQL> INSERT INTO t1 VALUES (CURRENT_TIME);
dmSQL> SELECT CURRENT_TIME;
dmSQL> SELECT c1 FROM t1 WHERE c2 = CURRENT_TIME;
```

➲ **例2**

下例首先在表中插入一条记录，记录中包括有CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP以及CURRENT_USER字段，然后显示记录值并更新表。

```
dmSQL> INSERT INTO sql99t5 VALUES (CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP,
CURRENT_USER);
1 row inserted

dmSQL> SELECT CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER;

CURRENT_TIME  CURRENT_D*    CURRENT_TIMESTAMP            CURRENT_USER
============= ========== ========================== ============================
16:53:09      2001-09-26 2001-09-26 16:53:09      SYSADM

dmSQL> UPDATE sql99t5 SET c1 = CAST(CURRENT_TIMESTAMP AS CHAR(20)),
                  c2 = CURRENT_DATE,
```

```
                    c3 = CURRENT TIME,
                    c4 = CURRENT TIMESTAMP WHERE c1 = CURRENT USER;
1 row updated
```

4.1.28. CURRENT_TIMESTAMP

CURRENT_TIMESTAMP函数将以DBMaster默认的时间戳（timestamp）输出格式返回当前的时间戳。

*Return value.....*TIMESTAMP类型：当前时间戳。

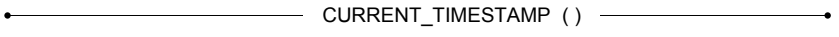


图 4-28 CURRENT_ TIMESTAMP函数语法图

➲ **例1**

下例介绍该函数的几种用法。您可以直接向表中插入该函数，也可以把它作为SELECT语句中的返回值。

```
dmSQL> INSERT INTO t1 VALUES (CURRENT_TIMESTAMP);
dmSQL> SELECT CURRENT_TIMESTAMP;
dmSQL> SELECT c1 FROM t1 WHERE c2 = CURRENT_TIMESTAMP;
```

➲ **例2**

下例首先在表中插入一条记录，记录中包括有CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP以及CURRENT_USER字段，然后显示记录值并更新表。

```
dmSQL> INSERT INTO sql99t5 VALUES (CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP,
CURRENT_USER);
1 row inserted

dmSQL> SELECT CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER;

CURRENT_TIME  CURRENT_D*    CURRENT_TIMESTAMP            CURRENT_USER
============= ========== ========================== ============================
16:53:09      2001-09-26 2001-09-26 16:53:09      SYSADM

dmSQL> UPDATE sql99t5 SET c1 = CAST(CURRENT_TIMESTAMP AS CHAR(20)),
                  c2 = CURRENT_DATE,
```

```
                    c3 = CURRENT TIME,
                    c4 = CURRENT TIMESTAMP WHERE c1 = CURRENT USER;
1 row updated
```

4.1.29. CURRENT_USER

CURRENT_USER函数将返回当前连接在DBMaster数据库上的用户。

*Return value.....*USER类型：当前用户。

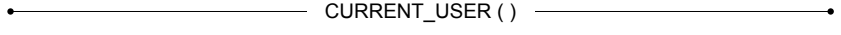


图 4-29 CURRENT_ USER函数语法图

➲ **例**

下例介绍该函数的几种用法。您可以直接向表中插入该函数，也可以把它作为SELECT语句中的返回值。

```
dmSQL> INSERT INTO t1 VALUES (CURRENT_USER);
dmSQL> SELECT CURRENT_USER;
dmSQL> SELECT c1 FROM t1 WHERE c2 = CURRENT_USER;
```

➲ **例2**

首先在表中插入一条记录，记录中包括有CURRENT_TIME、CURRENT_DATE、CURRENT_TIMESTAMP以及CURRENT_USER字段，然后显示记录值并更新表。

```
dmSQL> INSERT INTO sql99t5 VALUES (CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP,
CURRENT_USER);
1 row inserted

dmSQL> SELECT CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, CURRENT_USER;

CURRENT_TIME  CURRENT_D*    CURRENT_TIMESTAMP             CURRENT_USER
============ ========== ========================== ============================
16:53:09      2001-09-26 2001-09-26 16:53:09      SYSADM

dmSQL> UPDATE sql99t5 SET c1 = CAST(CURRENT TIMESTAMP AS CHAR(20)),
                  c2 = CURRENT_DATE,
```

```
                    c3 = CURRENT TIME,
                    c4 = CURRENT TIMESTAMP WHERE c1 = CURRENT USER;
1 row updated
```

4.1.30. CURTIME

CURTIME函数返回当前时间。

Return value.....Time类型：当前时间。

图 4-30 CURRENTTIME函数语法图

➲ **例**

下例返回当前时间。

```
CURTIME ()
```

4.1.31. DATABASE

DATABASE函数返回您当前连接的数据库的名称。您也可以在ODBC程序中调用SQLGetConnectOption的连接选项SQL_CURRENT_QUALIFIER来获得当前连接的数据库的名称。

Return value ..... String类型：当前连接的数据库名。

图 4-31 DATABASE函数语法图

➲ **例**

下面函数的返回值是用户当前连接的数据库名称。

```
DATABASE()
```

4.1.32. DATEPART

DATEPART函数返回的是指定时间戳（*timestamp*）的日期部分。

timestamp .....Timestamp类型：函数从这个时间戳提取日期部分。

Return value.....Date类型：时间戳的日期部分。

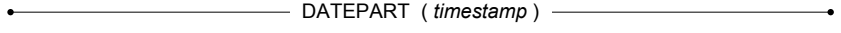


图 4-32 DATEPART函数语法图

➲ **例**

下例函数的返回值是：**1999-08-07**。

```
DATEPART('1999-08-07 10:11:12.123')
```

4.1.33. DAYNAME

DAYNAME函数返回一个字符串，其值是一周的某一天（例如返回值是Sunday、Monday、……、Saturday）。

date ..... Date类型：代表将被转换成一周中某一天的一个具体日期。

Return value ..... String类型：该*date*是一周中的哪一天。

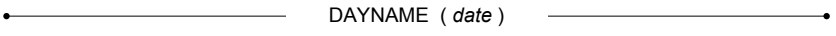


图 4-33 DAYNAME函数语法图

➲ **例**

下面的函数将返回“**Saturday**”。

```
DAYNAME('1999-12-25')
```

4.1.34. DAYOFMONTH

DAYOFMONTH函数将返回*date*日期在一个月中所处的天数，这是一个1到31之间的整数。

date .....Date类型：函数将返回这个日期所处的天数。

Return value.....Integer类型：返回值是给定日期在当月所处的天数。

图 4-34 DAYOFMONTH函数语法图

➲ **例**

下面的函数将返回**23**。

```
DAYOFMONTH('1999-01-23')
```

4.1.35. DAYOFWEEK

DAYOFWEEK函数将返回一个取值在1到7之间的整数，表示日期*date*是在一周中的第几天，其中，1表示星期日，2表示星期一，依此类推，7表示星期六。

date ..... Date类型：函数将计算这个日期是一周的第几天。

Return value ..... Integer类型：*date*日期是一周的第几天。

图 4-35 DAYOFWEEK函数语法图

➲ **例1**

下面函数的返回值是：**3**。

```
DAYOFWEEK('2000-02-29')
```

➲ **例2**

下面函数的返回值是：**6**。

```
DAYOFWEEK('2000-03-03')
```

4.1.36. DAYOFYEAR

DAYOFYEAR函数将返回一个在1到366之间取值的整数，其值表示*date*所指的日期是位于一年中的第几天。只有*date*是闰年的最后一天时，函数才返回366。

date .....Date类型：函数将计算这个日期位于当年的第几天。

Return value.....Integer类型：*date*所指日期是一年的第几天。

图 4-36 DAYOFYEAR函数语法图

➲ **例1**

下面函数的返回值是：**31**。

```
DAYOFYEAR('1999-01-31')
```

➲ **例2**

下面函数的返回值是：**365**。

```
DAYOFYEAR('1999-12-31')
```

4.1.37. DAYS_BETWEEN

DAYS_BETWEEN函数将返回两个日期之间所包含的天数。自变量*date1*可以在*date2*之前，也可以在*data2*之后。

date1 ..... Date类型：两个日期中的一个日期。

date2 ..... Date类型：两个日期中的另一个日期。

Return value ..... Integer类型：两个日期*date1*和*date2*间所包含的天数。

图 4-37 DAYS_BETWEEN函数语法图

➲ **例1**

下面函数的返回值是：**31**。

```
DAYS_BETWEEN('1999-01-15', '1999-02-15')
```

➲ **例2**

下面函数的返回值也是：**31**。

```
DAYS_BETWEEN('1999-02-15', '1999-01-15')
```

4.1.38. DEGREES

DEGREES函数将返回弧度（*radians*）所表示的度数，返回值是一个双精度的浮点类型数据。

radians .....Double类型：要转换成度数的弧度。

Return value.....Double类型：弧度（*radians*）所表示的度数。

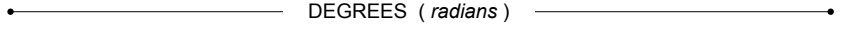


图 4-38 DEGREES函数语法图

➲ **例**

下面的函数将返回弧度3.14所表示的度数：**1.79908747671078e+002**。

```
DEGREES(3.14)
```

4.1.39. DOCTOTXT

函数 DOCTOTXT 用来将微软 Word 文件转换为临时的 BLOB，该BLOB文件为纯文本形式并以unicode编码。它将返回临时BLOB或 NULL。UDF支持office2007-2010版本。

Blob ..... 要转换为纯文本的字段名。

Return value ..... 如果BLOB可转换为纯文本，则返回NCLOB类型的临时BLOB。

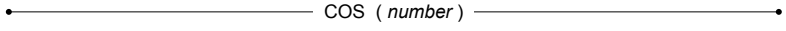


图 4-39 DOCTOTXT函数语法图

➲ **例**

下例说明如何将字段memo转换为纯文本。

```
DOCTOTXT(memo)
```

4.1.40. EXISTSNODE

函数 EXISTSNode 用来检查是否发现指定的节点。

Xmldata…… 要查找的XML

Xpath-expression……用户要使用其查找xml数据

Namespaces……...在xpath-expression中使用的可选项，用来指定命名空间。

Returnvalue……结果将依次排序为 NCLOB。

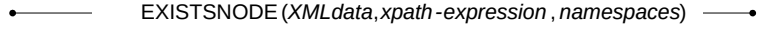


图 4-40 EXISTSNODE函数语法图

➲ 例

下例示范使用XML函数**EXISTSNODE**创建索引。

```
dmSQL> CREATE INDEX idx1 ON t1 (EXISTSNODE(c1, '/order/items/item/@product',
NULL));
```

4.1.41. EXP

EXP函数将返回一个双精度的浮点类型数据，其值为指数函数e^x的值。

x ..... Double类型：指数函数的幂。

Return value ..... Double类型：对e求*x*次幂的结果。

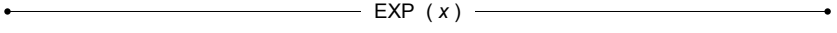


图 4-41 EXP函数语法图

➲ **例**

下面的函数将返回值：**2.71828182845905e+000**。

```
EXP(1)
```

4.1.42. EXTRACT

EXTRACT函数可以返回多值，单值和零值。该函数不允许使用升降序以及唯一索引。

Return value:..... UDF：允许UDF的结果为多值，单值或零值。

图 4-42 EXTRACT函数语法图

➲ **例**

使用XML UDF **EXTRACT**创建索引：

```
dmSQL> CREATE INDEX idx1 ON t1 (EXTRACT(c1, '/order/items/item/@product', NULL));
```

4.1.43. EXTRACTVALUE

EXTRACTVALUE函数仅返回单值或零值，它允许升降序排列以及唯一索引。

Return value: ..... UDF：允许UDF的返回值为单值或零值，不允许返回多值。.

图 4-43 EXTRACTVALUE函数语法图

➲ **例**

使用XML UDF **EXTRACTValue**创建索引：

```
dmSQL> CREATE INDEX idx2 ON t1 (EXTRACTVALUE(c1, '/order/items/item/@product',
NULL));
```

4.1.44. FILEEXIST

FILEEXIST函数的作用是确定*fileobject*指定的文件对象的物理文件是否存在。函数返回1表示文件存在，返回0表示文件不存在。

fileobject.....File类型：函数将检查这个文件对象的物理文件是否存在。

Return value.....Integer类型：表明文件是否存在的布尔值。

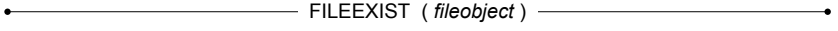


图 4-44 FILEEXIST函数语法图

➲ **例1**

下面函数的返回值是**1**，表明文件是存在的。

```
FILEEXIST(file_column)
```

➲ **例2**

下面函数的返回值是**0**，表明文件是不存在的。

```
FILEEXIST(nofile_column)
```

4.1.45. FILELEN

FILELEN函数返回一个整数，表示*fileobject*文件大小。该函数返回的整数的最大值是$(2^{31}-1)$B，即使文件对象的实际大小大于或等于2^{31}B。自变量 *fileobject*必须是数据库中的FILE数据类型的字段。

fileobject..... File类型：函数将返回这个文件的长度。

Return value ..... Integer类型：文件所占的字节数。

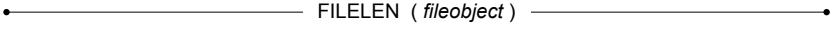


图 4-45 FILELEN函数语法图

➲ **例**

下面函数的返回值是**211**，表示文件占**211**个字节。

```
FILELEN(file_column)
```

4.1.46. FILELENEX

函数FILELENEX返回一个Decimal类型的值，表示文件对象的大小。自变量fileobject必须是数据库中为FILE类型的字段。与函数FILELIN不同，即使文件对象的大小超过2^{31}B，该函数也将返回文件对象大小的正确值。

fileobject.....File类型: 函数将返回这个文件的长度。

Return value .....Decimal类型：File 文件的长度。

图 4-46 FILELENEX函数函数语法图

➲ **例**

下例将返回 **211**，因为 file类型中包含**211**个字节。

```
FILELENEX(file_column)
```

4.1.47. FILENAME

FILENAME函数返回一个字符串，值为*fileobject*的文件名。自变量*fileobject*必须是数据库中的FILE数据类型的字段。

fileobject..... File类型：函数将返回这个文件的文件名。

Return value ..... String类型：文件的文件名。

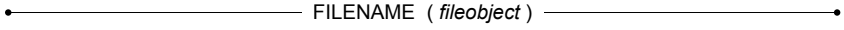


图 4-47 FILENAME函数语法图

➲ **例**

下例将返回：**C:\PATH\MYFILE.FIL**。

```
FILENAME(file_column)
```

4.1.48. FIX

FIX函数返回的是*number*的整数部分。

number.....Double类型：函数将返回这个数的整数部分。

Return value.....Bigint类型：*number*的整数部分。

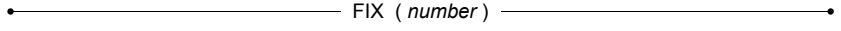


图 4-48 FIX 函数语法图

➲ **例1**

下例将返回**11**。

```
FIX(11.99)
```

➲ **例2**

下例将返回**12**。

```
FIX(12.01)
```

➲ **例3**

下例将返回**–11**。

```
FLOOR(-11.99)
```

➲ **例4**

下例将返回**–12**。

```
FLOOR(-12.01)
```

4.1.49. FLOOR

FLOOR函数将返回一个双精度的浮点类型数据，其值为等于或小于*number*的最大整数。

number..... Double类型：函数将返回等于或小于这个数的最大整数。

Return value ..... Double类型：等于或小于*number*的最大整数。

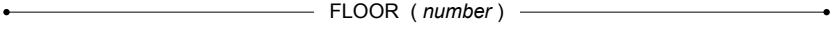


图 4-49 FLOOR 函数语法图

➲ **例1**

下例将返回**1.20000000000000e+001**。

```
FLOOR(12.01)
```

➲ **例2**

下例将返回**1.10000000000000e+001**。

```
FLOOR(11.99)
```

➲ **例3**

下例将返回**-1.20000000000000e+001**。

```
FLOOR(-11.99)
```

➲ **例4**

下例将返回**-1.30000000000000e+001**。

```
FLOOR(-12.01)
```

4.1.50. FREXPE

FREXPE函数将返回一个整数，值为方程 $number = X \times 2^n$ 中的指数***n***的值，其中X的取值范围是：0.5 < $|X|$ < 1。

Number.....Double类型：函数将计算方程 $number = X \times 2^n$ 中的指数***n***。

Return value.....Integer类型：方程 $number = X \times 2^n$ 中指数***n***的值。

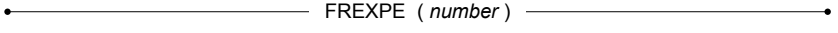


图 4-50 FREXPE函数语法图

➲ **例**

下例的返回值是**3**，因为**X**在**0.5**到**1**(不包含1)之间取值时，只有***n***等于**3**，*number*才有可能等于**4.0**。

```
FREXPE(4.0)
```

4.1.51. FREXPM

FREXPM函数将返回一个双精度的浮点类型数据，其值为方程 $number = X \times 2^n$ 中X的尾数，X的取值范围是0.5 < $|X|$ < 1。

number..... Double类型：函数将求解方程 $number = X \times 2^n$ 中的X的尾数。

Return value ..... Integer类型：方程 $number = X \times 2^n$ 中X的尾数。

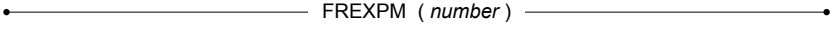


图 4-51 FREXPM函数语法图

➲ **例**

下面例子的返回值是**5.00000000000000e-001**，这意味着*number*等于**4.0**的情况下，若要***n***正好是一个整数，**X**就必须等于**0.5**或**5.00000000000000e-001**。

```
FREXPM(4.0)
```

4.1.52. FTOA

FTOA函数将返回一个包含数字*number*的字符串，该*number*是一个拥有固定小数位的数据。自变量*digits*指定小数点后的尾数，自变量*format*指明返回值是以普通小数格式还是以指数格式返回。

自变量*format*可以取4个值：“f”、“F”、“e”或“E”。使用“f”或“F”将以普通小数的格式返回值，比如*digits*为2时将返回123.45。使用“e”或“E”将以指数格式返回值，比如返回1.23e+02。指数格式的小数可以转化成相应的小数格式。

number.....Double类型：函数把这个数字转换为一个字符串。

digits.....Integer类型：小数点后的位数。

format.....String类型：返回值的格式。

Return value.....String类型：包含*number*的字符串，这个*number*是一个有固定小数位的数据。

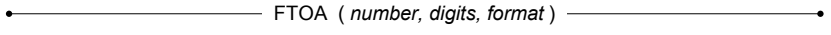


图 4-52 FTOA函数语法图

➲ **例1**

下例的返回值是“**123.46**”。

```
FTOA(123.456789, 2, 'f')
```

➲ **例2**

下例的返回值是“**1.23e+002**”。

```
FTOA(123.456789, 2, 'e')
```

4.1.53. HIGHLIGHT

HIGHLIGHT函数的作用是修改源文件，然后返回修改后的文件。在这个文件中，所有匹配*BollPatn*所指式样的文本前后都会加上标签**preTag**和**endTag**以示突出。

标签所占的空间最大不能超过10000个字节。如果匹配式样中包含有布尔运算符 [&, |, !, (,)]，那么除了包含! (NOT)式样外，其它所有简单查询式样的前后都将加上标签。输入的文档可以是CLOB、file、CHAR或多媒体类型的数据。

如果输入的文档是XMLTYPE类型的数据，HIGHLIGHT函数将返回错误6536，此时用户可先通过调用PURETEXT函数将XMLTYPE类型数据转换为NCLOB类型数据，然后再将满足条件的式样突出显示。

text ..... CLOB类型：源文件。

BoolPatn ..... Char类型：要突出的式样，该式样还可以是一个布尔表达式。

sensitive ..... Integer类型：匹配式样时，1表示对大小写敏感，0表示不敏感。

PreTag ..... Char类型：式样前加的标签，NULL表示不加标签。

EndTag ..... Char类型：式样后加的标签，NULL表示不加标签。

Return value ..... NCLOB类型：修改后的文件。

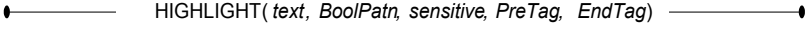


图 4-53 HIGHLIGHT函数语法图

➲ **例1**

下例将返回修改后的文件，文件中的“**Intel**”或“**AMD**” 将被加上了前标签（**preTag**）“**<<**” 和后标签（**endTag**）“**>>**” 以示突出。

```
dmSQL> SELECT HIGHLIGH(content,'Intel | AMD',0,'<<','>>') FROM news WHERE content
MATCH 'Intel| AMD';
```

➲ **例2**

下例将返回修改后的文件，文件中的“**dbmaster**”将被加上了前标签（**preTag**）“**<**”和后标签（**endTag**）“**>**”以示突出。

```
dmSQL> CREATE TABLE tpdf(c1 SERIAL, c2 pdffiletype);
dmSQL> SELECT HIGHLIGHT(c2,'dbmaster',0,'<','>') FROM tpdf;
```

➲ **例3**

下例将返回修改后的文件，文件中的“**dbmaster**”将被加上了前标签（**preTag**）“**<**”和后标签（**endTag**）“**>**”以示突出。

```
dmSQL> CREATE TABLE txml(c1 SERIAL,c2 XMLTYPE);
dmSQL> SELECT HIGHLIGHT(PURETEXT(c2), 'dbmaster',0,'<','>') FROM txml;
```

4.1.54. HITCOUNT

HITCOUNT函数返回的是匹配式样在源文本中出现的频率。

布尔式样出现频率的计算规则如下：

a AND b ：min(count(a)，count(b))

a OR b：count(a) + count(b)

NOT a ：count = 0

text ..... CLOB类型：源文本。

BoolPatn ..... Char类型：要突出的式样，这个式样也可以是一个布尔表达式。

sensitive ..... Integer类型：匹配式样时，1表示对大小写敏感，而0表示不敏感。

Return value ..... Integer类型：源文本中出现目标文本式样的次数。

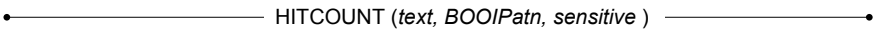


图 4-54 HITCOUNT函数语法图

➲ **例**

下例将返回“**target**”在源数据**content**中出现的次数，该式样对大小写不敏感。

```
HITCOUNT(content, "target", 0)
```

4.1.55. HITPOS

HITPOS函数的作用是返回第*n*个式样在源文本中所处的位置。这个返回的位移可以是：式样的开始位置、结束位置、式样的长度、开始位置（高于24bits）或结束位置（低于8bits），位移是从1开始计算的。

text .....CLOB类型：源文本。

BoolPatn .....Char类型：要突出的式样，这个式样还可以是一个布尔表达式。

sensitive .....Integer类型：匹配式样时，1表示对大小写敏感，而0表示不敏感。

n .....Integer类型：源文本中的第n个式样。

RetType .....Char类型：返回值的类型：

0：开始位移（默认设置)

1：结束位移

2：式样长度（结束位移－开始位移＋1）

3：开始位置（高于24bits)或者结束位置（低于8bits)

Return value.....Integer类型：返回第n个式样在源文本中的位置，如果在源文本中没有找到，那么就返回0。

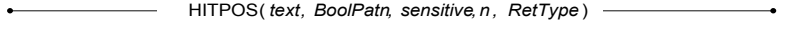


图 4-55 HITPOS 函数语法图

➲ 例

下例将返回**5**、**3**、**5**以及**7**，源文本是“a b A c”。

```
HITPOS(src,'A',   1, 1, 0) = 5  ('A')
HITPOS(src,'A&B' 0, 2, 0) = 3  ('b')
HITPOS(src,'a|b|c', 0, 3, 0) = 5  ('A')
HITPOS(src,'!a&c' 0, 1, 0) = 7  ('c')
```

4.1.56. HMS

HMS函数的返回值是一个*hours*: *minutes: seconds*格式的时间。自变量*hours*表示时间的小时部分，有效的取值范围是0到23之间之间的整数。Hours必须以24小时的格式来输入，因为此处不能输入AM或者PM来表示12小时格式中的时间。自变量*minutes*表示时间的分钟部分，有效的取值范围是0到59之间的整数。自变量*seconds*表示时间的秒钟部分，有效的取值范围是0到59之间的整数。

hours..... Integer类型：时间的小时部分。

minutes ..... Integer类型：时间的分钟部分。

seconds ..... Integer类型：时间的秒钟部分。

Return value ..... Time类型：含有时，分，秒的时间格式。

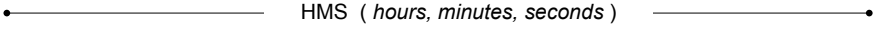


图 4-56 HMS函数语法图

➲ **例1**

下例将返回**10:11:12**，其等价于**10:11:12 AM**。

```
HMS(10, 11, 12)
```

➲ **例2**

下例将返回**22:11:12**，其等价于**10:11:12 PM**。

```
HMS(22, 11, 12)
```

4.1.57. HOUR

HOUR函数将返回一个整数，表示时间的小时部分，在0到23之间取值。

time .....Time类型：函数将返回这个时间的小时部分。

Return value.....Integer类型：*time*的小时部分。

图 4-57 HOUR函数语法图

➲ **例1**

下例将返回**10**。

```
HOUR('10:11:12')
```

➲ **例2**

下例将返回**22**。

```
HOUR('PM 10:11:12')
```

4.1.58. HTMLHIGHLIGHT

HTMLHIGHLIGHT函数将返回一个修改后的数据，其中所有匹配查找式样的文本前都将加上前标签（**preTag**），后面加上后标签（**endTag**）。HTMLHIGHLIGHT函数还可以使用highlight函数来引用HTML文件中的式样，而不破坏HTML文档的结构。

标签所占用的存储空间最大不能超过10000字节。如果匹配式样中包含布尔运算符 [&, |, !, (,)]，那么除了包含! (NOT)式样外，其它所有简单查询式样的前后都将加上标签。输入文档可以是CLOB、file或CHAR类型的数据。并非所有含有内部标签的源文本（包括注释）都将会用标签来加以突出。所有的HTML标签（包括注释）都被看作空格（SPACE）符。例如，如果式样是“DBMaster License”，那么HTML文件“DBMaster
License”就将被加上标签来突出，但是如果HTML文件是“<B>DBM</B>aker”，那么它将不匹配“DBMaster”这个式样。只有<BODY>后的文本才能被加标签来突出。

text ..... CLOB类型：源文本。

BoolPatn ..... Char类型：要突出的式样，该式样还可以是一个布尔表达式。

sensitive ..... Integer类型：匹配式样时，1表示对大小写敏感，0表示不敏感。

PreTag ..... Char类型：式样前加的标签，NULL表示不加标签。

EndTag ..... Char类型：式样后加的标签，NULL表示不加标签。

Return value ..... BLOB类型：在式样上加上标签后的文本。

图 4-58 HTMLHIGHLIGHT函数语法图

➲ 例

下例将返回源文本修改后的内容，其中所有匹配“**Intel**”或“**AMD**”的文本的前后分别加上标签“**<<**”和“**>>**”以示突出。

```
HTMLHIGHLIGHT(content,'Intel | AMD',0,'<<','>>')
```

4.1.59. HTMLTITLE

HTMLTITLE函数将返回一个HTML文件的标题（即包含在<*tilte*>和</*title*>之间的文本）。

object ..... BLOB类型：源HTML文件。

Return value ..... Varchar类型：HTML数据的标题。

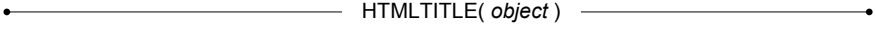


图 4-59 HTMLTITLE函数语法图

➲ **例**

下例将返回HTML数据“**htmlFile**”的标题。

```
HTMLTITLE(htmlFile)
```

4.1.60. HTMTOTXT

函数 HTMTOTXT用来将html文件转换为临时的BLOB文件，该BLOB文件为纯文本形式并以Local code 编码。

Blob.....要转换为纯文本的字段名

Return value.....如果BLOB可以转换为纯文本，将返回CLOB类型的临时文件

图 4-60 HTMTOTXT语法

➲ **例**

下例中将字段memo的内容转换为纯文本。

```
HTMTOTXT(memo)
```

4.1.61. HYPOT

HYPOT函数将返回一个双精度的浮点类型数据，值为一个直角三角形的斜边长度。这个斜边的长度是由方程 $z^2=x^2+y^2$ （勾股定理 Pythagorean Theorem)计算出来的，其中，*z*就是直角三角形的斜边。

x ..... Double类型：直角三角形的一条直角边边长。

y ..... Double类型：直角三角形的另一条直角边边长。

Return value ..... Double类型：直角三角形的斜边的长度。

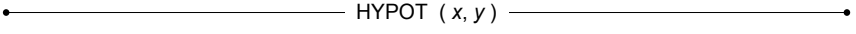


图 4-61 HYPOT函数语法图

➲ **例**

下例将返回**5**。

```
HYPOT(3,4)
```

4.1.62. INSERT

INSERT函数返回一个字符串，该字符串是将*string1*的第*start*开始的*length*个字符用*string2*替换后所得的结果。*start*值表示将从*string1*的第几个字符开始被替换为*string2*。如果*length*值为0，则表示将*string2* 插入到*string1*中，而不替换*string1*中的任何字符。如果参数中任一自变量没有指定值，那么系统将返回错误信息。

如果两个字符串中任何一个包含空值（NULL）或任一整型自变量中包含不标准（*atypical*）值时，DBMaster将按如下规则来决定函数的返回值：

- 如果*string1*包含*NULL*值，那么函数将返回*NULL*值。
- 如果*start*、*length*或*string2*包含*NULL*值，函数将返回*string1*的值。
- 如果*start*值小于0或*length*值小于0，函数将返回*string1*的值。
- 如果*start*值大于*string1*的长度＋1时，函数将返回*string1*的值。

string1 .....String类型：要插入字符的字符串。

start .....Integer类型：在*string2*中从第一个字符开始插入*string1*。

Length. .....Integer类型：*string1*中有几个字符被替换为 *string2*。

string2 .....String类型：在源字符串*string1*中插入该字符串。

Return value.....String类型：在字符串*string1*中插入*string2*后的结果。

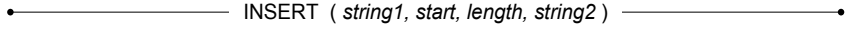


图 4-62 INSERT函数语法图

➲ **例1**

下例将返回“**Good ng!**”。

```
INSERT('morning!', 1, 5, 'Good ')
```

➲ **例2**

下例将返回“**Good morning!**”。

```
INSERT('Good ', 6, 8, 'morning!')
```

➲ **例3**

下例将返回“**Good night!**”。

```
INSERT('Good morning!', 6, 7, 'night')
```

➲ **例4**

下例将返回“**Good morning, sir. Here is your coffee.**”。

```
INSERT('Good morning! Here is your coffee.', 13, 1, ', sir.')
```

4.1.63. INVDATE

INVDATE函数的作用是判断自变量*date*所指定的日期是否是有效日期。函数可能返回的值有：

- 1 表示date为无效日期（例如超出了有效的日期范围）。
- 0 表示date为有效日期（例如‘0001-01-01’到‘9999-12-31’之间的日期）。
- -1 表示date为一个未知值（例如*NULL*值）。

date .....Date类型：函数将检查该日期的有效性。

Return value.....Integer类型：表明要检查的日期是否是有效的布尔值。

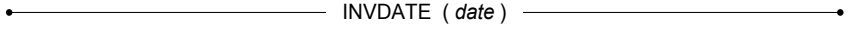


图 4-63 INVDATE 函数语法图

➲ **例**

下例将返回**0**，这表示检查的日期是个有效日期。

```
INVDATE('2000-01-01')
```

4.1.64. INVTIME

INVTIME函数的作用是判断自变量*time*所指定的时间是否是一个有效时间，函数可能的返回值是：

- 1 表示time为无效时间（例如：超出了有效的时间范围）。
- 0 表示time为有效时间（例如：‘00:00:00’到‘24:00:00’之间的时间）。
- -1 表示time是个未知值（例如：*NULL*值）。

time ..... Time类型：函数将检查这个时间的有效性。

Return value ..... Integer类型：表明要检查的时间是否是有效的布尔值。

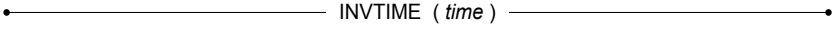


图 4-64 INVTIME函数语法图

➲ **例**

下例将返回**0**，这表明time是一个有效时间。

```
INVTIME('01:01:01')
```

4.1.65. INVTIMESTAMP

INVTIMESTAMP函数的作用是检查自变量*timestamp*所指的时间戳（timestamp）是否有效。函数可能的返回值有：

- 1 表示timestamp为无效时间戳（例如超出了有效的时间戳范围）。
- 0 表示timestamp为有效时间戳（例如‘00:00:00’到‘24:00:00’之间的值）。
- -1 表示timestamp是一个未知值（例如*NULL*值）。

timestamp .....Timestamp类型：函数将检查这个时间戳的有效性。

Return value.....Integer类型：表明要检查的时间戳是否是有效的布尔值。

图 4-65 INVTIMESTAMP 函数语法图

➲ **例**

下例将返回**0**，这表明该时间戳是有效的。

```
INVTIMESTAMP('1999-08-07 10:11:12.123')
```

4.1.66. LAST_DAY

LAST_DAY函数返回自变量*date*中指定月份的最后一天日期。

date ..... Date类型：函数将返回这个日期所在月份的最后一天日期。

Return value ..... Date类型：与*date*同月的最后一天日期。

图 4-66 LAST_DAY 函数语法图

➲ **例1**

下例将返回‘**1996-02-29**’。

```
LAST_DAY('1996-02-08')
```

➲ **例2**

下例将返回‘**2002-12-31**’。

```
LAST_DAY('2002-12-25')
```

4.1.67. LCASE

LCASE函数的作用是将字符串*string*中的所有大写字母转换为小写字母，数字和其它符号不受影响。如果自变量*string*是NULL，那么函数仍返回NULL值。如果没有指定自变量*string*，系统将返回错误信息。

string .....String类型：函数把该文本中的大写字母转换为小写字母。

Return value.....String类型：将*string*中的大写字母转换成小写字母后所得的字符串。

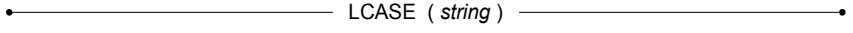


图 4-67 LCASE函数语法图

➲ **例1**

下例将返回“**abcdef**”。

```
LCASE('ABCdef')
```

➲ **例2**

下例将返回“**abc123**”。

```
LCASE('ABC123')
```

➲ **例3**

下例将返回“**abc@#$**”。

```
LCASE('ABC@#$')
```

4.1.68. LDEXP

LDEXP函数将返回一个双精度的浮点类型数据，其值是方程 $number = X \times 2^n$ 中*number*的值。

x ..... Double类型：方程 $number = X \times 2^n$ 中X的值。

n ..... Integer类型：方程 $number = X \times 2^n$ 中的指数值。

Return value ..... Double类型：方程 $number = X \times 2^n$ 中*number*的值。

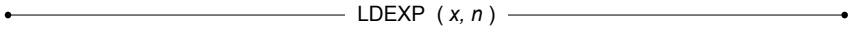


图 4-68 LDEXP函数语法图

➲ **例**

下例将返回**8.00000000000000e+000**。

```
LDEXP(0.5, 4)
```

4.1.69. LEFT

LEFT函数将返回字符串*string*最左边的*count*个字符。如果*count*的值小于0，函数将返回NULL值。如果有任何一个参数没有指定值，函数都将返回错误信息。

string .....String类型：函数将从这个字符串中抽取字符。

count .....Integer类型：抽取字符的个数。

Return value.....String类型：*string*最左边的*count*个字符。

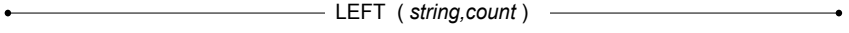


图 4-69 LEFT函数语法图

➲ **例**

下例将返回“**Good**”。

```
LEFT('Good morning!', 4)
```

4.1.70. LENGTH

LENGTH函数返回字符串*string*所包含的字符个数，不包括字符串末尾的空格和字符串结束符。如果没有为自变量*string*提供值，函数将返回错误信息。

string ..... String类型：函数将计算这个字符串中所包含的字符数。

Return value ..... Integer类型：字符串*string*所包含的字符数。

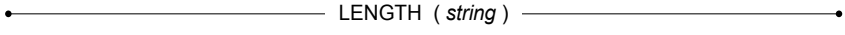


图 4-70 LENGTH函数语法图

➲ **例**

下例的返回值是**13**。

```
LENGTH('Good morning!   ')
```

4.1.71. LOCATE

LOCATE函数返回*string1*第一次出现在*string2*中的位置。您还可以使用自变量*start*来设置从什么位置开始查找*string1*。为自变量*start*赋1，表示从*string2* 的第一个字符开始查询*string1*。如果在*string2*中找不到*string1*，函数将返回0。如果两个字符串中其中一个包含NULL值或自变量*start*中包含不标准的值时，DBMaster将根据以下的规则来决定返回值：

- 如果*string1*包含*NULL*值，函数将返回*NULL*值。
- 如果*string2*或*start*包含*NULL*值，函数将返回0。
- 如果*start*小于或等于0，函数将返回正确的值。
- 如果*start*大于字符串*string2*的长度加1时，函数将返回0。

string1 .....String类型：要定位的字符串。

string2 .....String类型：函数在这个字符串中查找*string1*。

start .....Integer类型：从*string2*的第几个字符开始查找。

Return value.....Integer类型：*string1*第一次出现在*string2*中的位置。

图 4-71 LOCATE函数语法图

➲ **例1**

下例将返回**4**。

```
LOCATE('def', 'abcdefghi', 1)
```

➲ **例2**

下例将返回**0**。

```
LOCATE('def', 'abcdefghi', 5)
```

➲ **例3**

下例将返回**4**。

```
LOCATE('def', 'abcdefghi', 4)
```

➲ **例4**

下例将返回**4**。

```
LOCATE('def', 'abcdefghi', -1)
```

➲ **例5**

下例将返回**0**。

```
LOCATE('def', 'abcdefghi', 10)
```

4.1.72. LOG

LOG函数返回一个双精度的浮点类型数据，其值为*x*的自然对数。

x .....Double类型：函数将计算这个数的自然对数。

Return value.....Double类型：*x*的自然对数。

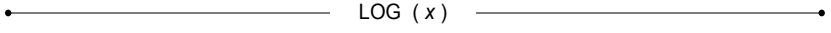


图 4-72 LOG函数语法图

➲ **例**

下例将返回**1.00000000000000e+000**。

```
LOG(2.71828182845905e+000)
```

4.1.73. LOG10

LOG10函数返回一个双精度的浮点类型数据，它的值是对*x*求以10为底的对数。

x ..... Double类型：函数将对这个数计算以10为底的对数。

Return value ..... Double类型：*x*的以10为底的对数。

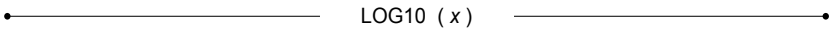


图 4-73 LOG10函数语法图

➲ **例**

下例将返回**2**。

```
LOG10(100)
```

4.1.74. LOWER

LOWER函数执行的运算和函数LCASE相同。作用是把字符串中的所有字符转换成小写字母。

String_expression... String类型：函数将把这个字符串中的字母转换成小写字母。

Return value..... String类型：将大写字母转换成小写字母后所得的字符串。

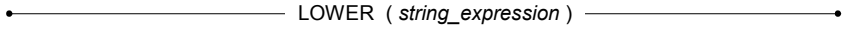


图 4-74 Lower函数语法图

➲ 例

```
dmSQL> SELECT LOWER('ABCDEF');
LOWER('ABCDEF')
===============
abcdef
```

4.1.75. LTRIM

LTRIM函数的作用是将字符串*string*串首空格删除后再返回这个字符串。如果没有为自变量赋值，函数将返回错误信息。

string ..... String类型：函数将删除这个字符串左边的空格。

Return value ..... String类型：删去串首空格后的字符串。

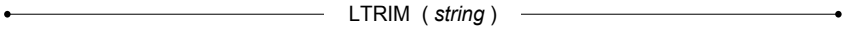


图 4-75 LTRIM函数语法图

➲ **例**

下面函数将返回字符串“**Good morning!**”。

```
LTRIM('   Good morning!')
```

4.1.76. MDY

MDY函数以当前日期格式返回一个以*month*/*day*/*year*组合的日期数据。自变量*month*表示日期中的月份，有效值是1到12的整数。自变量*day*表示日期中的天，有效值是1到31的整数。自变量*year*表示日期中的年，有效值是0001到9999的整数。

month .....Integer类型：日期数据中的月份。

day .....Integer类型：日期数据中的天。

year .....Integer类型：日期数据中的年。

Return value.....Date类型：含有年、月、日的日期格式数据。

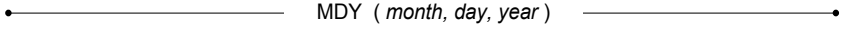


图 4-76 MDY函数语法图

➲ **例1**

当前的日期格式是**yyyy-mm-dd**时，下面的函数将返回日期数据：**1996-02-08**。

```
MDY(2,8,1996)
```

➲ **例2**

当前的日期格式是**mm/dd/yyyy**时，下面的函数将返回日期数据：**02/08/2001**。

```
MDY(2,8,2001)
```

4.1.77. MINUTE

MINUTE函数将返回一个整数，其值是时间*time*中的分，这是一个在0到59之间取值的整数。

time ..... Time类型：函数将返回这个时间的分钟。

Return value ..... Integer类型：时间*time*的分。

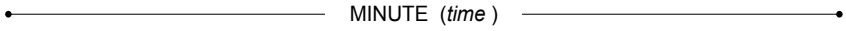


图 4-77 MINUTE函数语法图

➲ **例**

下例的返回值是**11**。

```
MINUTE('10:11:12')
```

4.1.78. MOD

MOD函数将返回一个双精度的浮点类型数据，值是*x*除以*y*的余数或模数。

x .....Double类型：被除数。

y .....Double类型：除数。

Return value.....Double类型：余数。

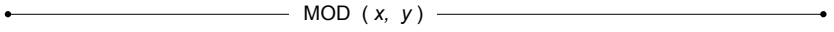


图 4-78 MOD函数语法图

➲ **例**

下例的返回值是**2.00000000000000e+000**。

```
MOD(17, 3)
```

4.1.79. MODFI

MODFI函数返回一个双精度的浮点类型数据，值是*number*的整数部分。

number..... Double类型：函数将截取这个数的整数部分。

Return value ..... Double类型：*number*的整数部分。

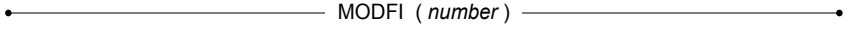


图 4-79 MODFI 函数语法图

➲ **例1**

下例的返回值是：**3.00000000000000e+000**。

```
MODFI(3.1415926535897936)
```

➲ **例2**

下例的返回值是：**-3.00000000000000e+000**。

```
MODFI(-3.1415926535897936)
```

4.1.80. MODFM

MODFM函数将返回一个双精度的浮点类型数据，其值是*number*的尾数部分。

number.....Double类型：函数将截取这个数的尾数部分。

Return value.....Double类型：*number*的尾数部分。

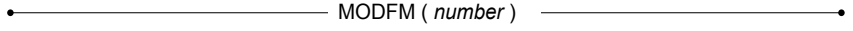


图 4-80 MODFM函数语法图

➲ **例1**

下例的返回值是：**1.41592653589790e-001**。

```
MODFM(3.1415926535897936)
```

➲ **例2**

下例的返回值是：**-1.41592653589790e-001**。

```
MODFM(-3.1415926535897936)
```

4.1.81. MONTH

MONTH函数将返回一个1到12之间的整数，其值是日期*date*中的月份。

date ..... Date类型：函数将返回这个日期所在的月。

Return value ..... Integer类型：日期*date*所在的月。

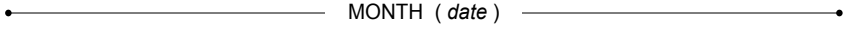


图 4-81 MONTH函数语法图

➲ **例**

下例的返回值是**2**。

```
MONTH('1996-02-29')
```

4.1.82. MONTHNAME

MONTHNAME函数返回一个字符串，表示日期***date***所在月的名称（例如，JAN，FEB，…，DEC等）。自变量***date***必须是一个有效的日期，否则DBMaster会返回错误信息。

date .....Date类型：函数将返回这个日期所在月的名称。

Return value.....String类型：***date***所在月份的名称。

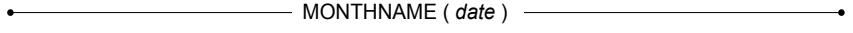


图 4-82 MONTHNAME函数语法图

➲ **例**

下例将返回“**FEB**”。

```
MONTHNAME('1996-02-29')
```

4.1.83. NEXT_DAY

NEXT_DAY函数返回紧接*date*的下一周中，对应*weekday*的日期值。自变量 *weekday*的有效输入值是一周中任一天的名称（Monday，Tueday，…，Sunday等）或是它们的缩写（Mon，Tue，…，Sun等）。*weekday*的值不区分大小写。

date..... Date类型：函数将返回紧接这个日期值之后对应*weekday*的日期。

weekday..... String类型：一周中的哪一天。

Return value ..... Date类型：紧接*date*的下一周中对应*weekday*的日期值。

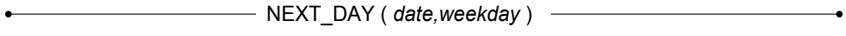


图 4-83 NEXT_DAY函数语法图

➲ **例1**

下面例子的返回值是：**1996-03-04**。

```
NEXT_DAY('1996-02-29', 'Monday')
```

➲ **例2**

下面例子的返回值是：**1996-03-05**。

```
NEXT_DAY('1996-02-29', 'Tuesday')
```

4.1.84. NOW

NOW函数将返回一个时间戳值，包括当前的日期和时间。

Return value.....Timestamp类型：当前的日期和数据。

图 4-84 NOW函数语法图

4.1.85. PDFTOTXT

函数 PDFTOTXT 用来将PDF文件转换为临时 BLOB文件，该BLOB文件为纯文本形式并以unicode编码。它将返回临时BLOB或NULL。请注意支持PDF格式的DBMaster版本为1.2、1.3、1.4、1.5、1.6和1.7。

Blob: ..... 要转换为纯文本的字段名

Return value: ..... 如果BLOB可以转换为纯文本，将返回 NCLOB类型的临时文件

图 4-85 PDFTOTXT函数语法图

➲ **例**

下例中将字段memo的内容转换为纯文本。

```
PDFTOTXT(memo)
```

4.1.86. PI

PI函数将返回一个精度为38、宽度为16的小数、值为常量π（3.1415926535897936）。

Return value..... Decimal类型：π的值。

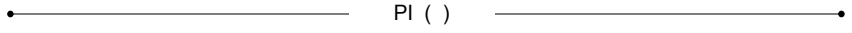


图 4-86 PI函数语法图

4.1.87. POSITION

POSITION函数返回*string1*第一次出现在*string2*中的位置。如果在*string2*中找不到*string1*，函数将返回0。如果两个字符中的其中一个包含NULL值，DBMaster将根据以下规则来决定返回值：

- 如果*string1*包含NULL值，函数返回*NULL* 。
- 如果*string2*包含*NULL*值，函数将返回0。

string1 ..... String类型：要定位的字符串。

string2 ..... String类型：函数将在这个字符串中查询*string1*。

Return value ..... Integer类型：*string1*第一次出现在*string2*中的位置。

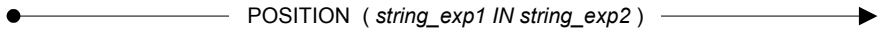


图 4-87 POSITION函数语法图

➲ **例1**

下面函数命令的返回值是**4**。

```
dmSQL> SELECT POSITION('abc' in 'defabcjlkjl');
POSITION('ABC' IN 'DEFABCJLKJL')
================================
                               4
```

➲ **例2**

下面函数命令的返回值是**1**。

```
dmSQL> SELECT POSITION('abc' in 'abcdefghihj');
POSITION('ABC' IN 'ABCDEFGHIHJ')
================================
                               1
```

➲ **例3**

下面函数命令的返回值是**0**。

```
dmSQL> SELECT POSITION('abc' in 'jlkjlkklj');
```

```
POSITION('ABC' IN 'JLKJLKKLJ')
==============================
                             0
```

4.1.88. POW

POW函数返回一个双精度的浮点类型数据，值为x^y的值。

x ..... Double类型：函数将计算这个数的*y*次幂。

y ..... Double类型：指数函数的幂。

Return value ..... Double类型：对*x*求*y*次幂所得的值。

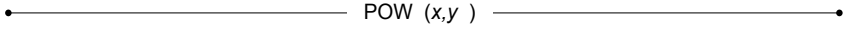


图 4-88 POW函数语法图

➲ **例**

下面例子的返回值是：**8.00000000000000e+000**。

```
POW(2, 3)
```

4.1.89. PPTTOTXT

函数 PPTTOTXT用来将微软PowerPoint文件转换为临时 BLOB文件，该BLOB文件为纯文本形式并以unicode编码。它将返回临时BLOB或NULL。UDF支持office2007-2010版本。

Blob.....要转换为纯文本的字段名。

Return value.....如果BLOB可以转换为纯文本，将返回 NCLOB类型的临时文件。

图 4-89 PPTTOTXT函数语法图

➲ **例**

下例中将字段memo的内容转换为纯文本。

```
PPTTOTXT(memo)
```

4.1.90. PURETEXT

函数 PURETEXT用来将BLOL转换为一个临时BLOB，该临时BLOB为纯文本形式并以unicode编码。

当在媒体类型的字段上或在带有纯文本转换器的定义域上使用PURETEXT函数时，DBMaster将在内部调用纯文本转换功能。

Blob..... 要转换为纯文本的字段名。

Return value ..... 如果BLOB可以转换为纯文本，将返回 NCLOB类型的临时文件。

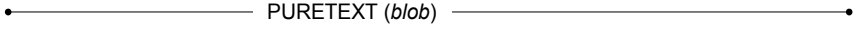


图 4-90 PURETEXT函数语法图

➲ **例**

下例中将字段memo的内容转换为纯文本。

```
PURETOTXT(memo)
```

4.1.91. QUARTER

QUARTER函数返回一个取值在1到4之间的整数，表示*date*日期所处的季度。返回值为1表示时间在1月1号到3月31号之间，依此类推。

date .....Date类型：函数将返回这个日期所处的季度。

Return value.....Integer类型：日期*date*所处的季度。

图 4-91 QUARTER函数语法图

➲ **例**

下例的返回值是**1**。

```
QUARTER('2002-01-20')
```

4.1.92. RADIANS

RADIANS函数返回一个双精度的浮点类型数据，其值是用弧度表示的度数。

degrees..... Double类型：函数将返回这个度数对应的弧度。

Return value ..... Double类型：将度数用弧度表示后的结果。

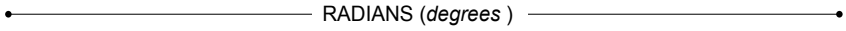


图 4-92 RADIANS函数语法图

➲ **例**

下例的返回值是：**3.14159265358979e+000**。

```
RADIANS(180)
```

4.1.93. RAND

RAND函数返回一个随机的整数。

Return value.....Integer类型：随机数。

图 4-93 RAND函数语法图

4.1.94. REPEAT

REPEAT函数返回一个字符串，这个字符串是重复*count*次*string*后的结果。如果*string*包含NULL值或空字符串时，DBMaster将按如下规则来确定返回值：如果*string*或*count*包含*NULL*值时，函数将返回*NULL*值。如果*count*小于0或*string*为一个空字符串时，函数将返回一个空字符串。如果没有为自变量赋值，那么系统将返回错误信息。

string ..... String类型：要重复的字符串。

count ..... Integer类型：要重复*string*的次数。

Return value ..... String类型：包含*count*个*string*的字符串。

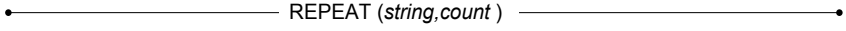


图 4-94 REPEAT函数语法图

➲ **例1**

下例将返回字符串“**Good morning! Good morning!**”。

```
REPEAT('Good morning! ', 2)
```

➲ **例2**

下例将返回字符串“**Zzzz Zzzz Zzzz Zzzz**”。

```
REPEAT('Zzzz ', 4)
```

4.1.95. REPLACE

REPLACE函数的作用是将*string1*中所有匹配*string2*的字符串用*string3*来代替。如果其中一个字符串包含空值，或是一个空字符串(即长度为0的字符串)，DBMaster将按照以下规则来确定返回值：

- 如果*string1*是*NULL*值，函数将返回*NULL*。
- 如果*string2*或*string3*是*NULL*，函数将返回*string1*。
- 如果*string2*是一个空字符串，函数则返回*string1*。

string1 .....String类型：函数将替换这个字符串中的相应字符。

string2 .....String类型：函数将替换*string1*中的这些字符串。

string3 .....String类型：函数将用这个字符串替换*string1*中的*string2*。

Return value.....String类型：将*string1*中的所有*string2*替换为*string3*后的结果。

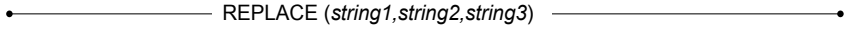


图 4-95 REPLACE函数语法图

➲ **例1**

下例将返回字符串“**Good evening! Good evening!**”。

```
REPLACE('Good morning! Good morning!', 'morning', 'evening')
```

➲ **例2**

下例将返回字符串“**Goodbye Dave.**”。

```
REPLACE('Hello, Dave.', 'Hello,', 'Goodbye')
```

4.1.96. RIGHT

RIGHT函数返回的是*string*最右边的*count*个字符。如果*count* 的值小于0，函数将返回NULL值。如果没有为自变量赋值，函数则返回错误信息。

string ..... String类型：函数将在这个字符串中抽取字符。

count ..... Integer类型：抽取字符的个数。

Return value ..... String类型：*string*最右边的*count*个字符。

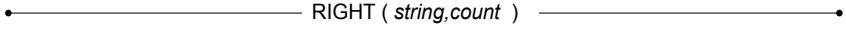


图 4-96 RIGHT函数语法图

➲ **例**

下例将返回字符串“**morning!** ”。

```
RIGHT('Good morning!  ', 10)
```

注意 *在函数的字符串自变量和返回值中，感叹号后面都有两个空格符。*

4.1.97. RND

RND函数返回一个最接近自变量*number*的整数，也就是对*number*四舍五入求整。

number.....Double类型：函数对这个数字四舍五入取整。

Return value.....Bigint 类型：最接近*number*的整数。

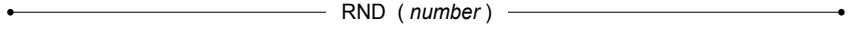


图 4-97 RND函数语法图

➲ **例1**

下例的返回值是**12**。

```
RND(12.01)
```

➲ **例2**

下例的返回值是**12**。

```
RND(12.49)
```

➲ **例3**

下例的返回值是**13**。

```
RND(12.50)
```

➲ **例4**

下例的返回值是**13**。

```
RND(12.99)
```

4.1.98. ROUND

ROUNDS函数返回的是按指定位数进行四舍五入的数值。参数*decimal_places*可能是一个负数，但必须是一个整数。

ROUND函数四舍五入的规则如下：

- 如果参数*decimal_places*省略，函数则将数字四舍五入到小数点处。
- 如果参数*decimal_places*大于*0*，函数则将数字四舍五入到指定的小数位。
- 如果参数*decimal_places*等于*0*，函数则将数字四舍五入到最接近的整数。
- 如果参数*decimal_places*小于*0*，函数则在小数点左侧前几位进行四舍五入。

number..... Double类型：函数对这个数四舍五入取整。

decimal_places.. Integer类型：按此位数对 number 参数进行四舍五入。

Return value ..... Bigint类型：最接近*number*的整数或小数。

图 4-98 ROUND函数语法图

➲ **例1**

下面函数的返回值是**124**。

```
ROUND(123.56)
```

➲ **例2**

下面函数的返回值是**37.269000000000000000**。

```
ROUND(37.269412, 3)
```

➲ **例3**

下面函数的返回值是**125.3611000000000000000**。

```
ROUND(125.361080, 4)
```

➲ **例4**

下面函数的返回值是**8912341.000000000000000000**。

```
ROUND(8912341.123456, 0)
```

➲ **例5**

下面函数的返回值是**1234600.000000000000000000**。

```
ROUND(1234591.123450, -2)
```

注意 ***Round****函数的返回值类型是decimal(38,19)，因此，dmSQL中显示的小数点右边将会有19位数字。*

4.1.99. RTRIM

RTRIM函数的作用是将字符串*string*串末的空格删除，并返回。如果没有为自变量赋值，系统将返回错误信息。

string ..... String类型：函数将删除这个字符串最右边的空格字符。

Return value ..... String类型：删除串尾空格字符后的字符串。

图 4-99 RTRIM函数语法图

➲ **例**

下例将返回“**Good morning!**”。

```
RTRIM('Good morning!  ')
```

注意 *函数字符串自变量的感叹号后有两个空格字符。*

4.1.100. SECOND

SECOND函数将返回一个在0到59之间取值的整数，表示时间*time*的秒。

time .....Time类型：函数将返回这个时间中的秒钟部分。

Return value.....Integer类型：时间*time*中的秒钟。

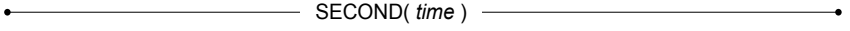


图 4-100 SECOND函数语法图

➲ **例**

下例的返回值是**12**。

```
SECOND('10:11:12')
```

4.1.101. SECS_BETWEEN

SECS_BETWEEN函数返回两个时间之间间隔的秒数。自变量*time1*表示的时间可以比自变量*time2*表示的时间早，也可以比它晚。

time1 ..... Time类型：两个时间中的一个时间。

time2 ..... Time类型：两个时间中的另一个时间。

Return value ..... Integer类型：*time1*和*time2*之间间隔的秒数。

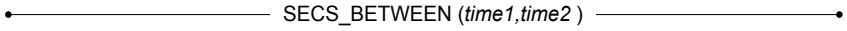


图 4-101 SECS_BETWEEN函数语法图

➲ **例**

下例的返回值是**36000**。

```
SECS_BETWEEN('10:10:10', '20:10:10')
```

4.1.102. SESSION_USER

SESSION_USER函数返回当前连接到DBMaster数据库上的用户。

Return value..... 当前用户名。

图 4-102 SESSION_ USER函数语法图

➲ **例**

下例列出了该函数的几种用法。

```
dmSQL> INSERT INTO t1 VALUES (SESSION_USER);
dmSQL> SELECT SESSION_USER;
dmSQL> SELECT c1 FROM t1 WHERE c2 = SESSION_USER;
```

4.1.103. SIGN

SIGN函数返回的是一个表明*number*正负特征的整数。返回值为+1表示*number*是一个正数，返回值是0表示*number*为0，而如果返回值是-1则表示*number*是一个负数。

number..... Double类型：函数将返回这个数的符号类型。

Return value ..... Integer类型：表示*number*的符号的整数。

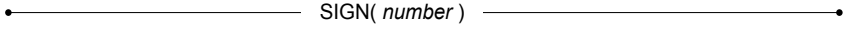


图 4-103 SIGN函数语法图

➲ **例1**

下例的返回值是**1**。

```
SIGN(12.3)
```

➲ **例2**

下例的返回值是**0**。

```
SIGN(0)
```

➲ **例3**

下例的返回值是**-1**。

```
SIGN(-12.3)
```

4.1.104. SIN

SIN函数返回一个双精度的浮点类型数据，该值是用弧度方式表示的自变量*number*的正弦值（sine）。

number.....Double类型：函数将返回这个数的正弦值。

Return value.....Double类型：*number*的正弦值。

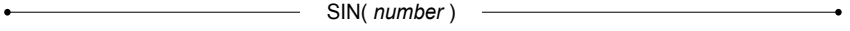


图 4-104 SIN函数语法图

➲ **例**

下例的返回值是**4.79425538604203e-001**。

```
SIN(0.5)
```

4.1.105. SINH

SIN函数返回一个双精度的浮点类型数据，该值是用弧度方式表示的自变量*number*的双曲正弦值。

number..... Double类型：函数将计算这个数的双曲正弦值。

Return value ..... Double类型：*number*的双曲正弦值。

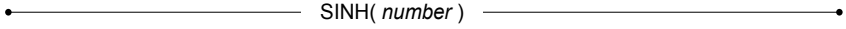


图 4-105 SINH函数语法图

➲ **例**

下例的返回值是**5.21095305493747e-001**。

```
SINH(0.5)
```

4.1.106. SPACE

SPACE函数将返回一个由连续*count*个空格组成的字符串。如果*count*的值小于0，那么函数将返回NULL值。

count .....Integer类型：空格的个数。

Return value.....String类型：连续*count*个空格组成的字符串。

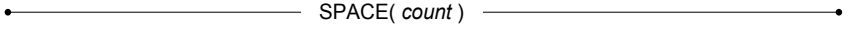


图 4-106 SPACE函数语法图

➲ **例1**

下例将返回一个包含3个空格组成的字符串“ ”。

```
SPACE(3)
```

➲ **例2**

下例将返回字符串“**Good morning!**”，串首有3个空格字符。

```
CONCAT(SPACE(3), 'Good morning!')
```

注意 *返回值第一个字母前面有3个空格。*

4.1.107. SQRT

SQRT函数返回一个双精度的浮点类型数据，其值为*x*的平方根。

x ..... Double类型：函数将计算这个数的平方根。

Return value ..... Double类型：*x*的平方根。

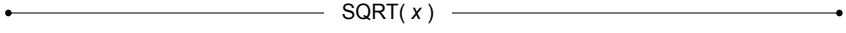


图 4-107 SQRT函数语法图

➲ **例**

下例的返回值是：**1.30000000000000e+001**。

```
SQRT(169)
```

4.1.108. STRTOINT

STRTOINT函数的作用是将字符串转换成一个整数，自变量*string*是NULL时，函数将返回一个NULL值。如果*string*并不能被转换成一个整数，系统将返回错误信息。

string .....String类型：要转换成整数的字符串。

Return value.....Bigint类型：*string*经过转换后所得的整数。

图 4-108 STRTOINT函数语法图

➲ **例**

下例的返回值是**1234**。

```
STRTOINT('1234')
```

4.1.109. SUBBLOB

SUBBLOB函数从自变量*blob*的*start*字节开始，取*length*个字节来生成一个临时BLOB。BLOB字节数是从1开始计数的。这个函数是个附加功能，需要运行DBMaster提供的**libblob.sql**脚本来安装此功能。如果任何一个自变量包含NULL值或者为0时，DBMaster将使用以下的规则来决定函数的返回值。

- 如果*blob*是NULL，函数将返回一个*NULL*值。
- 如果*start*或*length*是*NULL*，函数将返回一个和源*blob*数据相同的临时*BLOB*。
- 如果*start* <= 0或*length* < 0，函数将返回一个*NULL*值。
- 如果*start* >*blob*的总字节数，函数将返回一个*NULL*值。
- 如果*length*为0，函数则返回一个空的临时*BLOB*。

blob ..... BLOB类型：函数将从CLOB、FILE数据中抽取部分数据。

start ..... Integer类型：函数将从*blob*的这个位置开始抽取数据。

length ..... Integer类型：抽取数据的字节数。

Return value ..... BLOB类型：从*blob*中抽取的临时BLOB。

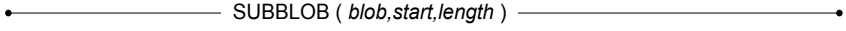


图 4-109 SUBBLOB函数语法图

➲ **例**

下例是从Data BLOB的**1001**字节开始，抽取**100**个字节来生成一个临时BLOB数据。

```
SUBBLOB(Data, 1001, 100)
```

4.1.110. SUBBLOBTOBIN

SUBBLOBTOBIN函数从自变量*blob*的*start*字节开始，取*length*个字节来生成一个二进制串。BLOB字节数是从1开始计数的。这个函数是个附加功能，需要运行DBMaster提供的**libblob.sql**脚本来安装此功能。如果任何一个自变量包含NULL值或者为0时，DBMaster将使用以下规则来决定函数的返回值。

- 如果*blob*是*NULL*，函数将返回一个*NULL*值。
- 如果*start*或*length*是*NULL*，函数将返回和源*blob*数据相同的串。
- 如果*start* <= 0 或*length* < 0，函数将返回一个*NULL*值。
- 如果*start* > *blob*的总字节数，函数将返回一个*NULL*值。
- 如果*length*为0，函数将返回一个空字符串。

blob .....BLOB (BLOB、CLOB、FILE)类型： 函数将从这个数据中抽取数据。

start .....Integer类型：函数将从*blob*的这个位置开始抽取数据。

length .....Integer类型：抽取数据的字节数。

Return value.....Binary string类型：从*blob*中抽出的数据。

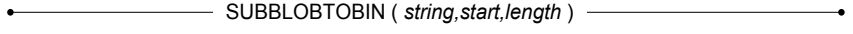


图 4-110 SUBBLOBTOBIN函数语法图

➲ **例**

下例将从BLOB文件Data的**1001**字节开始，到第**1100**之间的100个字节来生成一个二进制串。

```
SUBBLOBTOBIN(Data, 1001, 100)
```

4.1.111. SUBBLOBTOCHAR

SUBBLOBTOCHAR函数从自变量*blob*的*start*字节开始，取*length*个字节来生成一个字符串。BLOB字节数是从1开始计数的。这个函数是个附加功能，需要运行DBMaster提供的**libblob.sql**脚本来安装此功能。如果任何一个自变量包含NULL值或为0时，DBMaster将使用以下规则来决定函数的返回值。

- 如果*blob*是*NULL*，函数将返回一个*NULL*值。
- 如果*start*或*length*是*NULL*，函数将返回和源*blob*数据相同的字符串。
- 如果*start* <= 0或*length* < 0，函数将返回一个*NULL*值。
- 如果*start* > *blob* 的总字节数，函数将返回一个*NULL*值。
- 如果*length*为0，函数则将返回一个空字符串。

blob ..... BLOB类型：函数将从BLOB、CLOB、FILE数据中抽取部分数据。

start ..... Integer类型：函数将从*blob*的这个位置开始抽取数据。

length ..... Integer类型：抽取数据的字节数。

Return value ..... Character String类型：从*blob*中抽出的数据。

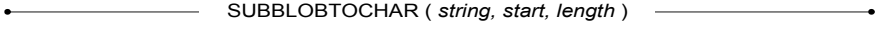


图 4-111 SUBBLOBTOCHAR 函数语法图

➲ **例**

下例将从BLOB 文件Data的**1001**个字节开始，到第**1100**个字节来生成一个字符串。

```
SUBBLOBTOCHAR(Data, 1001, 100)
```

4.1.112. SUBSTRING

SUBSTRING函数返回从*string*的*start*位置开始的*length*个字符。当任一自变量包含NULL值或为0时，DBMaster将根据以下规则来确定函数的返回值。

- 如果*string*为*NULL*，函数将返回一个*NULL*值。
- 如果*start*或*length*是*NULL* ，函数将返回源字符串*string*。
- 如果*start* < 0或*length* < 0，函数将返回一个*NULL*值。
- 如果*start* >= *string*的总长度，函数将返回一个*NULL*值。
- 如果*length*为0，函数则将返回一个空字符串。

string .....String类型：函数将从这个字符串中抽取子串。

start .....Integer类型：函数将从*string*的这个位置开始抽取子串。

length .....Integer类型：抽取的字符个数。

Return value.....String类型：从*string*中抽取的子串。

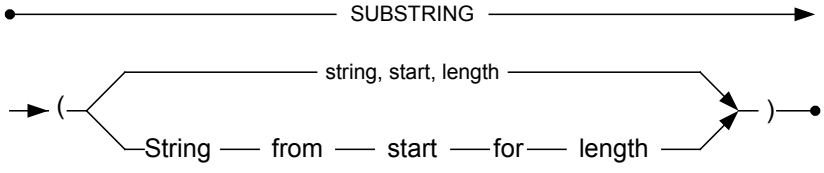


图 4-112 SUBSTRING函数语法图

➲ **例1**

下例将返回字符串“**morning**”。

```
SUBSTRING('Good morning!', 6, 7)
```

➲ **例2**

```
dmSQL> SELECT SUBSTRING(CAST(123456 AS CHAR(10)) FROM LENGTH('abc') for
LENGTH('abc'));
SUBSTRING(CAST(123456 AS CHAR(10)
================================
345
```

➲ **例3**

```
dmSQL> SELECT SUBSTRING('abcdef', 2, 2);
```

4.1.113. TAN

TAN函数的返回值是*number*的正切值（tangent）。其中自变量是以弧度方式表示的，返回值是一个双精度的浮点类型数据。

number.....Double类型：函数将计算这个数的正切值。

Return value.....Double类型：*number*的正切值。

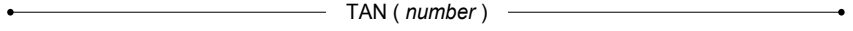


图 4-113 TAN函数语法图

➲ **例**

下例的返回值是**5.46302489843790e -001**。

```
TAN(0.5)
```

4.1.114. TANH

TANH函数将返回自变量*number*的双曲正切值，其中自变量是以弧度方式表示的，而返回值则是一个双精度的浮点类型数据。

Number ..... Double类型：函数将计算这个数的双曲正切值。

Return value ..... Double类型：*number*的双曲正切值。

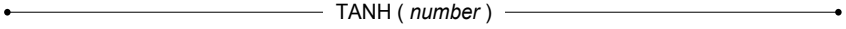


图 4-114 TANH函数语法图

➲ **例**

下例的返回值是**4.62117157260010e-001**。

```
TANH(0.5)
```

4.1.115. TIMEPART

TIMEPART函数将返回时间戳*timestamp*中的时间部分。

timestamp .....Timestamp类型：函数将从这个时间戳中抽取时间部分。

Return value.....Time类型：*Timestamp*的时间部分。

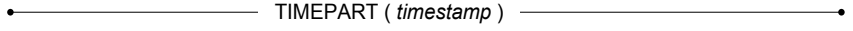


图 4-115 TIMEPART函数语法图

➲ **例**

下例将返回**10:11:12**。

```
TIMEPART('1996-02-29 10:11:12.123')
```

4.1.116. TIMESTAMPADD

TIMESTAMPADD函数将返回在自变量*timestamp*基础上，间隔一定时间段后所得的时间戳。

时间间隔	时间间隔的单位
"f"（ODBC程序中用SQL_TSI_FRAC_SECOND）	小于一秒的单位。
"s"（ODBC程序中使用SQL_TSI_SECOND）	秒
"m"（ODBC程序中使用SQL_TSI_MINUTE）	分
"h"（ODBC程序中使用SQL_TSI_HOUR）	时
"D"（ODBC程序中使用SQL_TSI_DAY）	日
"W"（ODBC程序中使用SQL_TSI_WEEK）	周
"M"（ODBC程序中使用SQL_TSI_MONTH）	月
"Q"（ODBC程序中使用SQL_TSI_QUARTER）	季度
"Y"（ODBC程序中使用SQL_TSI_YEAR）	年

表 4-1 TIMESTAMPADD中时间间隔的参数表

interval ..... String类型：所加时间段的单位。

number..... Integer类型：所加时间段的量。

timestamp ..... Timestamp类型：函数将在这个时间戳上增加一定的时间段。

Return value ..... Timestamp类型：*timestamp* + *interval* × *number*的结果。

TIMESTAMPADD (*interval*,*number*,*timestamp*)

图 4-116 TIMESTAMPADD函数语法图

➲ 例

下例的返回值是：**1996-01-17 06:10:10**。

```
TIMESTAMPADD('h',20,'1996-01-16 10:10:10')
```

4.1.117. TIMESTAMPDIFF

TIMESTAMPDIFF函数返回两个时间戳*timestamp1*和*timestamp2*之间的时间间隔，返回值的单位由自变量*interval*指定。

时间间隔	时间间隔的单位
"f"（ODBC程序中用SQL_TSI_FRAC_SECOND）	小于一秒的单位
"s"（ODBC程序中使用SQL_TSI_SECOND）	秒
"m"（ODBC程序中使用SQL_TSI_MINUTE）	分
"h"（ODBC程序中使用SQL_TSI_HOUR）	时
"D"（ODBC程序中使用SQL_TSI_DAY）	天
"W"（ODBC程序中使用SQL_TSI_WEEK）	周
"M"（ODBC程序中使用SQL_TSI_MONTH）	月
"Q"（ODBC程序中使用SQL_TSI_QUARTER）	季度
"Y"（ODBC程序中使用SQL_TSI_YEAR）	年

表 4-2 TIMESTAMPDIFF 中时间间隔的参数表

interval ..... String类型：两个时间戳的间隔单位。

timestamp1 ..... Timestamp类型：两个要比较的时间戳中的第一个时间戳。

timestamp2 ..... Timestamp类型：两个要比较的时间戳中的第二个时间戳。

Return value ..... Double类型：*timestamp2* - *timestamp1*的结果。

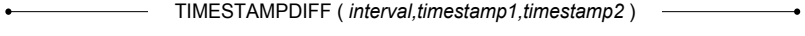


图 4-117 TIMESTAMPDIFF 函数语法图

➲ **例**

下面函数的返回值是**2.40000000000000e+001**。

```
TIMESTAMPDIFF('h','1996-01-16 10:10:10', '1996-01-17 10:10:10')
```

4.1.118. TRIM

TRIM函数是LTRIM函数和RTRIM函数的结合，在trim_char_value_expr中可以指定一个或多个字符，每个字符都被看作是有效的可截去的字符。

如果没有为trim选项指定BOTH、LEADING或TRAILING值，系统将该选项默认为BOTH。默认的trim_char_value_expr字符是空格字符（''）。此外，如果trim_char_value_expr的值为空串（''），函数将返回源字符串。如果trim_source为NULL，不管trim选项和trim字符是怎样的，函数都将返回NULL。LENGTH函数还可以和TRIM函数一起使用，下例将展示这种语法的用法。

leading.....删除trim_sorce前面的trim_string。

trailing.....删除trim_sorce后面的trim_string。

both .....删除trim_sorce前面和后面的trim_string。

若不选择上述三个选项（即：leading、trailing、both），trim函数将删除trim_soruce前面和后面的trim_expr。

trim_expr .....trim_sorce中要删除的字符。

若该参数省略，trim函数将删除trim_souce前面和后面的空格。

trim_source .....要删除的字符串。

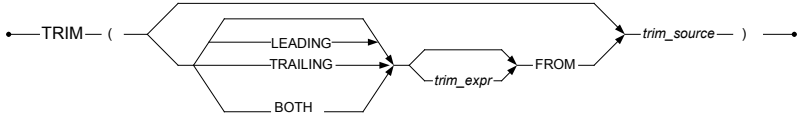


图 4-118 TRIM函数语法图

➲ **例1**

```
dmSQL> SELECT TRIM(both 'a' FROM 'aabcaa');
 TRIM(BOTH 'A' FROM 'AABCAA')
============================
bc
```

➲ **例2**

```
dmSQL> SELECT TRIM(FROM 'aabcaa');
 TRIM(FROM 'AABCAA')
====================
aabcaa
```

➲ **例3**

```
dmSQL> SELECT TRIM('a' FROM 'aabcaa');
 TRIM('A' FROM 'AABCAA')
=========================
bc
```

➲ **例4**

```
dmSQL> SELECT TRIM('abc' FROM 'abckjkjjdcba');
  TRIM('ABC' FROM 'ABCKJKJJDCBA')
=================================
kjkjjd
```

➲ **例5**

```
dmSQL> SELECT TRIM('a c' FROM 'ac ddbc');
 TRIM ('A C' FROM 'AC DDBC')
============================
ddb
```

➲ **例6**

```
dmSQL> SELECT LENGTH(TRIM(leading FROM ' abc '));
    LENGTH(TRIM(LEADING FROM ' ABC '))
========================================
                                     3
```

➲ **例7**

```
dmSQL> SELECT LENGTH(TRIM(leading 'a' FROM 'aabc '));
   LENGTH(TRIM(LEADING 'A' FROM 'AA'))
=====================================
                                    2
```

➲ **例8**

```
dmSQL> SELECT LENGTH(TRIM(trailing FROM 'aabc '));
    LENGTH(TRIM(TRAILING FROM 'AABC'))
====================================
                                   4
```

➲ 例9

```
dmSQL> SELECT LENGTH(TRIM(trailing 'a' FROM 'aabcaa'));
 LENGTH(TRIM(TRAILING 'A' FROM 'AABCAA'))
=========================================
                                       4
```

4.1.119. UCASE

UCASE函数的作用是将*string*中的所有小写字母转化成大写字母。如果自变量*string*是NULL，函数将返回NULL。如果没有为自变量*string*赋值，那么系统将返回错误信息。

string ..... String类型：函数将把该文本中的小写字母转换成大写字母。

Return value ..... String类型：将小写字母转换成大写字母后的文本。

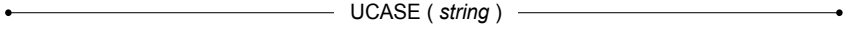


图 4-119 UCASE函数语法图

➲ **例1**

下例将返回字符串“**ABCDEF**”。

```
UCASE('ABCdef')
```

➲ **例2**

下例将返回字符串“**ABC123**”。

```
UCASE('abc123')
```

➲ **例3**

下例将返回字符串“**ABC@#$**”。

```
UCASE('abc@#$')
```

4.1.120. UPPER

该函数的功能和UCASE相同，也就是将字符串中的所有字符都转换成大写字母。自变量*string*为NULL时，函数将返回NULL。

String-expression...String类型：函数将把该文本中的小写字母转换成大写字母。

Return value.....String类型：所有字符被转换成大写字母后的结果。

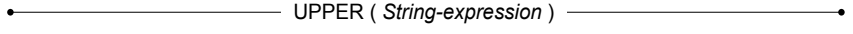


图 4-120 UPPER函数语法图

➲ **例**

```
dmSQL> SELECT UPPER('abcdef');
 UPPER('ABCDEF')
=================
ABCDEF
```

4.1.121. USER

USER函数将返回当前用户的授权名，用户的授权名也可以通过调用SQLGetInfo的SQL_USER_NAME选项来获得。

Return value ..... String类型：当前用户的用户名。

图 4-121 USER函数语法图

4.1.122. UTFConvert

UTFConvert 函数用于UTF-8和UTF-16之间的字符集转换，它包括U8TOU16和U16TOU8两个函数。

UTFConvert 函数在DBMaster安装路径*shared\udf*下，在数据库中它不是默认存在的。如果用户需要，则必须手动创建。

执行以下命令来创建这些函数：

```
dmSQL> CREATE FUNCTION UTFConvert.U8TOU16(long varbinary) RETURNS nclob;
dmSQL> CREATE FUNCTION UTFConvert.U16TOU8(nclob) RETURNS long varbinary;
```

long varbinary .....将被转换为UTF-16的UTF-8内容。

nclob .....将被转换为UTF-8的UTF-16内容。

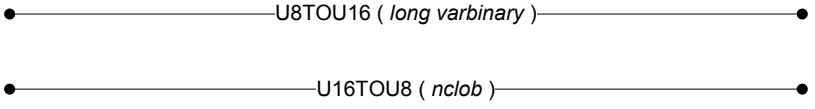


图 4-122 UTFConvert函数语法图

4.1.123. WEEK

WEEK函数返回的是一个取值在**1**到**53**之间的整数，表示日期***date***是处于当年的第几周。

date ..... Date类型：函数将返回这个日期处于当年的第几周。

Return value ..... Integer类型：*date*是处于当年的第几周。

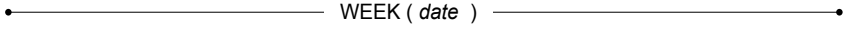


图 4-123 WEEK函数语法图

➲ **例**

2002-02-11是**2002**年的第五周，所以下面例子的返回值是**5**。

```
WEEK('2002-02-01')
```

4.1.124. XLSTOTXT

函数 XLSTOTXT 用来将微软 excel 文件转换为临时BLOB，该BLOB为纯文本形式并以unicode编码，它将返回临时BLOB或NULL。UDF支持office2007-2010版本。

blob .....要转换为纯文本的字段名

Return value.....如果BLOB可以转换为纯文本，将返回 NCLOB类型的临时文件

图4-124 XLSTOTXT函数语法图

➲ **例**

下例中将字段memo的内容转换为纯文本。

```
XLSTOTXT(memo)
```

4.1.125. XMLUPDATE

函数 XMLUPDATE 通过XPath来定位要更新的 xml 数据的位置。

xmldata……要更新的XML内容

xpath-expression……指定要更新的xml数据的位置

namespaces……在Xpath-expression中用来指定命名空间，可选。

replace-content…… 用来代替 Xpath定位部分的内容

returnvalue……更新后的整个 XML 文件

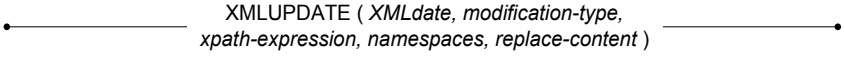


图 4-125 XMLUPDATE函数语法图

4.1.126. YEAR

YEAR函数将返回一个取值在**1**到**9999**之间的整数，表示*date*所处的年份。

date .....Date类型：函数将返回这个日期所处的年份。

Return value.....Integer类型：日期*date*所处的年份。

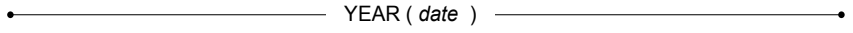


图 4-126 YEAR函数语法图

➲ **例**

下例将返回**2002**。

```
YEAR('2002-02-01');
```

4.2 用户自定义函数

DBMaste允许用户自己创建用户自定义函数（UDF）。UDF被写入DBMaster之后，将会被视为一个新的用法相同的内置函数。

4.2.1. AES_DECRYPT

AES_ENCRYPT函数用于对数据进行加密，以保护那些重要的数据。相应地，AES_DECRYPT函数用于解密之前已经加密的数据，以得到原始数据。

AES_DECRYPT函数的存储路径是DBMaster安装目录*shared**udf*。该函数不是数据库默认创建的。若要使用该函数，用户需手动创建该函数或运行该存储目录下的SQL脚本。

执行如下命令创建AES_DECRYPT函数：

```
dmSQL> CREATE FUNCTION LIBCRYPT.AES_DECRYPT(BINARY(4096),STRING) RETURNS
BINARY(4096);
```

ciphertext .....密文

cipher key .....输入的密钥

plaintext .....原始数据

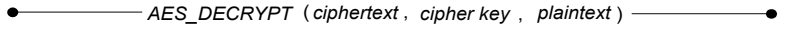


图 4-127 AES_DECRYPT函数语法图

AES_DECRYPT函数支持以下五种数据类型：BINARY(N)、CHAR(N)、VARCHAR(N)、NCHAR(N)、NVARCHAR(N)。如果原始数据是其它类型的数据，将会返回错误（6536）：函数参数和定义的不匹配。

使用16位加密，因此字符串增大，请注意原数据的长度必须和定义的UDF参数相匹配。在不同环境下，用户需修改参数BINARY(n)的值，以确保长度相对于要加密的数据足够大。此外，AES_DECRYPT函数和AES_ENCRYPT函数中参数BINARY(n)的值必须相匹配。

➲ **例**

用户可使用如下语法运行AES_DECRYPT函数。

```
dmSQL> SELECT AES_DECRYPT (Column, 'key') FROM table;
```

下例描述AES_DECRYPT函数的用法：

```
dmSQL> CREATE TABLE DAES(C1 BINARY(1024));
dmSQL> SELECT AES DECRYPT (C1, 'key') FROM DAES; //the result' data type is
BINARY, and please cast the value;
dmSQL> SELECT CAST(AES_DECRYPT (C1, 'key') AS CHAR(200)) FROM DAES;
```

4.2.2. AES_ENCRYPT

AES_ENCRYPT函数用于加密数据，以保护重要的数据。

AES_ENCRYPT函数的存储路径是DBMaster安装目录*shared**udf*。该函数不是数据库默认创建的。若要使用该函数，用户需手动创建该函数或运行该存储目录下的SQL脚本。

执行如下命令创建AES_ENCRYPT函数：

```
dmSQL> CREATE FUNCTION LIBCRYPT.AES_ENCRYPT(BINARY(4096),STRING) RETURNS
BINARY(4096);
```

plaintext .....要加密的原始数据。

cipher key .....输入的密钥。

ciphertext .....密文。

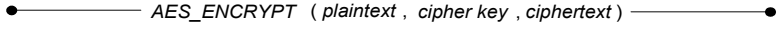


图 4-128 AES_ENCRYPT函数语法图

AES_ENCRYPT函数支持以下五种数据类型：BINARY(N)、CHAR(N)、VARCHAR(N)、NCHAR(N)、NVARCHAR(N)。如果原始数据是其它类型的数据，将会返回错误（6536）：函数参数和定义的不匹配。

使用16位加密，因此字符串增大，请注意原数据的长度必须和定义的UDF参数相匹配。在不同环境下，用户需修改参数BINARY(n)的值，以确保长度相对于要加密的数据足够大。此外，AES_DECRYPT函数和AES_ENCRYPT函数中参数BINARY(n)的值必须相匹配。

➲ 例

用户可使用如下语法运行AES_ENCRYPT函数。

```
dmSQL> SELECT AES_ENCRYPT (Column, 'key') FROM table;
```

下例描述AES_ENCRYPT函数的用法：

```
dmSQL> CREATE TABLE AES(C1 CHAR(200));
dmSQL> INSERT INTO AES VALUES('abc');
dmSQL> SELECT AES_ENCRYPT (C1, 'key') FROM AES INTO DAES;
```

4.2.3. DATETOSTR

DATETOSTR函数用于将DATE类型的数据转换为指定格式的字符串，该DATE类型数据必须是有效数据。

DATETOSTR函数的存储路径是DBMaster安装目录*shared**udf*。该函数不是数据库默认创建的。若要使用该函数，用户需手动创建该函数或运行该存储目录下的SQL脚本。

执行如下命令创建DATETOSTR函数：

```
dmSQL> CREATE FUNCTION datetostr.DATETOSTR(DATE, varchar(20)) RETURNS
varchar(20);
```

date .....将要转换为字符串的日期。

data_format_string　返回字符串的格式，日期将会被转化为该格式的字符串。DBMaster目前支持如下13种格式：mm/dd/yy、mm-dd-yy、dd/mon/yy、dd-mon-yy、mm/dd/yyyy、mm-dd-yyyy、yyyy/mm/dd、yyyy-mm-dd、dd/mon/yyyy、dd-mon-yyyy、dd.mm.yyyy、yyyy.mm.dd、yyyymmdd。此外，格式必须使用小写字母。

Return_value .....将日期转换为该字符串。

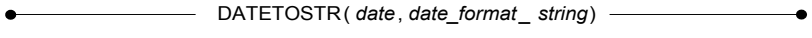


图 4-129 DATATOSTR函数语法图

➲ **例**

下例将日期“**2012-2-12**”转换为“**mm/dd/yy**”格式的字符串：

```
DATETOSTR('2012-12-12','mm/dd/yy')
```

4.2.4. TIMETOSTR

TIMETOSTR函数用于将TIME类型的数据转换为指定格式的字符串，该TIME类型数据必须是有效数据。

TIMETOSTR函数的存储路径是DBMaster安装目录*shared**udf*。该函数不是数据库默认创建的。若要使用该函数，用户需手动创建该函数或运行该存储目录下的SQL脚本。

执行如下命令创建TIMETOSTR函数：

```
dmSQL> CREATE FUNCTION datetostr.TIMETOSTR(TIME, varchar(20)) RETURNS
varchar(20);
```

time .....将要转换为字符串的时间。

time _format_string.....回字符串的格式，时间将会被转化为该格式的字符串。DBMaster目前支持如下13种格式：hh:mm:ss.fff、hh:mm:ss、hh:mm、hh、hh:mm:ss.fff tt、hh:mm:ss tt、hh:mm tt、hh tt、tt hh:mm:ss.fff、tt hh:mm:ss、tt hh:mm、tt hh、hhmmss。此外，格式必须使用小写字母。

Return_value .....将时间转换为该字符串。

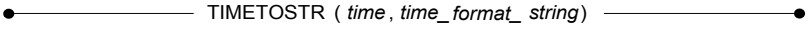


图 4-130 TIMETOSTR函数语法图

➲ 例

下例将时间“**12:10:10**”转换为“**hh:mm:ss:tt**”格式的字符串：

```
TIMETOSTR('12:10:10','hh:mm:ss tt')
```

4.2.5. TIMESTAMPTOSTR

TIMESTAMPTOSTR函数用于将TIMESTAMP类型的数据转换为指定格式的字符串，该TIMESTAMP类型数据必须是有效数据。

TIMESTAMPTOSTR函数的存储路径是DBMaster安装目录*shared\udf*。该函数不是数据库默认创建的。若要使用该函数，用户需手动创建该函数或运行该存储目录下的SQL脚本。

执行如下命令创建TIMESTAMPTOSTR函数：

```
dmSQL> CREATE FUNCTION datetostr.TIMESTAMPTOSTR(TIMESTAMP, varchar(20),
varchar(20)) RETURNS varchar(30);
```

timestamp .....将要转换为字符串的日期和时间。

date_format_string..... 回字符串的格式，日期将会被转化为该格式的字符串。DBMaster目前支持如下13种格式：mm/dd/yy、mm-dd-yy、dd/mon/yy、dd-mon-yy、mm/dd/yyyy、mm-dd-yyyy、yyyy/mm/dd、yyyy-mm-dd、dd/mon/yyyy、dd-mon-yyyy、dd.mm.yyyy、yyyy.mm.dd、yyyymmdd。此外，格式必须使用小写字母。

time_format_string..... 回字符串的格式，时间将会被转化为该格式的字符串。DBMaster目前支持如下13种格式：hh:mm:ss.fff、hh:mm:ss、hh:mm、hh、hh:mm:ss.fff tt、hh:mm:ss tt、hh:mm tt、hh tt、tt hh:mm:ss.fff、tt hh:mm:ss、tt hh:mm、tt hh、hhmmss。此外，格式必须使用小写字母。

Return_value .....将日期和时间转换为该字符串。

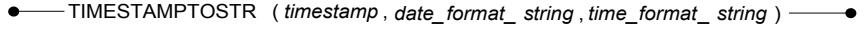


图 4-131 TIMESTAMPTOSTR语法图

➲ **例**

下例将日期和时间“**2012-12-12 12:10:10**”分别转换为“**mm/dd/yy**”格式和“**tt hh:mm:ss**”格式的字符串：

```
TIMESTAMPTOSTR('2012-12-12 12:12:12','mm/dd/yy' 'tt hh:mm:ss')
```

4.2.6. TO_DATE

TO_DATE函数的作用是将指定字符串转换成DATE类型的数据。该字符串可以是任何数据类型，但是在转换成一个日期时，必须是一个有效的日期。TO_DATE函数包含两个自变量：字符串（char_string）和日期格式串（date_format_string）。字符串参数表示要转换的字符串，而日期格式串表示DATE类型结果将以何种方式显示。

TO_DATE 函数的存储路径是DBMaster安装目录*shared\udf*，它并不是数据库默认创建的。如果要使用该函数，用户需要手动创建它或运行该存储目录下的SQL脚本。

可执行下面的命令来创建：

```
dmSQL> CREATE FUNCTION to_date.TO_DATE(varchar(20), varchar(20)) RETURNS DATE;
```

string_expr.....String类型：要转换成日期的字符串。

date_format_string.....日期的格式。**Y**或**y**表示年，**M**或**m**表示月，**D**或**d**表示日。用“*/*”或者“**–**”作为分隔符。

Return value.....返回一个DATE类型的数据。

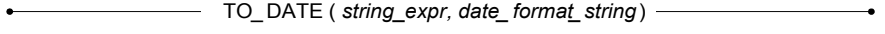


图 4-132 TO_DATE函数语法图

➲ **例1**

```
TO_DATE('991031', 'YYMMDD')
```

➲ **例2**

```
dmSQL> SELECT TRIM(FROM 'aabcaa');
dmSQL> SELECT TO_DATE('2009-Jan-01', 'YYYY-mon-DD');
```

5 系统存储过程

系统存储过程（*System-Stored Procedures*）是一些动态库模块，除非您调用它，否则它们将不会被载入。系统存储过程包括共享对象、XML导入和导出过程。

共享对象是一个带有符号的整型变量，存在于数据库的共享内存DCCA中。对共享对象的存取是比较高效的，并且具备独立的事务处理能力。共享对象不同于数据记录，它们并不存储于数据库中。因此，删除共享对象或关闭数据库后，共享对象的生命周期也就结束了。

每个连接到数据库上的用户都可以访问这些共享对象（在SYSADM加入共享对象之后），如果另外一个用户没有在共享对象上加锁，您就可以设置或获得这些共享对象的值。共享对象是一个带有符号的整型变量（占4个字节）。所有用户在共享对象上拥有同等权限或操作许可，所以任何用户都可以不考虑当前设置而重置共享对象的设置，当然在有锁定的情况下除外。

SYSADM、SYSDBA和DBA还可以使用另外两个系统存储过程（XMLEXPORT和XMLIMPORT）来导入或导出xml文件。

5.1 APPENDBLOB

系统存储过程APPENDBLOB用于逐页将一个大文件插入到一个BLOB/CLOB/FILE类型的字段中，该存储过程被创建在附加可执行文件中。DBMaster在创建数据库时并不初始化该存储过程，因此在使用之前，用户需通过运行*<DBMaster home installation directory>/shared/sp/AppendBlob.sql*对其进行声明。

为了简单起见，以下章节若没有特殊说明， BLOB/CLOB/FILE这三种类型均用BLOB类型表示。

如果没有记录或多条记录与WHERE_STR指定的条件匹配，错误将会发生且返回相关错误信息。

如果由TABLE_NAME、COLUMN_NAME和WHERE_STR指定的单元格的值是NULL或不是BLOB类型，错误将会发生且返回相关错误信息。

DATA_BUFF的最大值是10M比特，因此，如果DATA_BUFF或DATA_LEN的值大于10485760，错误将会发生且返回相关错误信息。

➲ **APPENDBLOB的原型如下：**

```
APPENDBLOB(VARCHAR(128) TABLE_NAME INPUT,
           VARCHAR(128) COLUMN_NAME INPUT,
           VARCHAR(2048) WHERE_STR INPUT,
           BINARY(10485760) DATA_BUFF INPUT,
           INTEGER DATA_LEN INPUT)
```

table_name ..... 包含BLOB类型字段的表名。

column_name ... 附加新数据的BLOB类型字段的字段名。

where_str.....指定单行的条件字符串。

data_buff.....该缓存内的数据将会被附加到由TABLE_NAME、COLUMN_NAME和WHERE_STR指定的BLOB类型字段中。

data_len.....DATA_BUFF中有效数据的长度。如果该长度小于DATA_BUFF的长度，DBMaster将会仅附加长度为DATA_LEN的数据，否则，将会附加缓存中的全部数据。

➲ 例

首先创建一个名为**bbsp**的数据库，接着声明存储过程APPENDBLOB。

```
dmSQL> CREATE DB bbsp;
USE db #1 connected to db:<bbsp> by user:<SYSADM>
dmSQL> RUN 'C:\DBMaster\5.4\shared\sp\AppendBlob.sql';
```

创建一个名为**test_blob1**的表，接着插入一条记录。

```
dmSQL> CREATE TABLE test_blob1(c1 INT,c2 BLOB);
dmSQL> INSERT INTO test_blob1 VALUES(1,?);
dmSQL/Val> &file1;
1 rows inserted
dmSQL/Val> END;
dmSQL> SELECT c1,BLOBLEN(c2) FROM test_blob1;
    C1        BLOBLEN(C2)
==========  =============
         1         81920
1 rows selected
```

调用存储过程APPENDBLOB，将更多数据附加到该BLOB后面。

```
dmSQL> CALL APPENDBLOB('test_blob1','c2','c1=1',?,10);
dmSQL/Val> 'xxxxxyyyyy';
dmSQL/Val> END;
dmSQL> SELECT c1,BLOBLEN(c2) FROM test_blob1;
    C1        BLOBLEN(C2)
==========  =============
         1         81930
1 rows selected
```

5.2 APPENDBLOBBYOID

系统存储过程APPENDBLOBBYOID用于逐页地插入一个大文件，用法和系统存储过程APPEDDBLOB相同，也被创建在附加可执行文件中，用户同样需要在使用之前对其进行声明。如果通过ROW_ID无法找到该行，DBMaster将会报错并返回相关错误信息。

如果由ROW_ID和COLUMN_ORDER指定的单元格的值是NULL或不是BLOB或FILE类型，DBMaster将会报错并返回相关错误信息。

DATA_BUFF的最大值是10M比特，因此，如果DATA_BUFF或DATA_LEN的值大于10485760，DBMaster将会报错并返回相关错误信息。

➲ **APPENDBLOBBYOID的原型如下：**

```
APPENDBLOBBYOID(BINARY(16) ROW_ID INPUT,
                INT COLUMN_ORDER INPUT,
                BINARY(10485760) DATA_BUFF INPUT,
                INT DATA_LEN INPUT)
```

row_id_input ..... 包含BLOB类型字段的记录id。

column_order .... 附加新数据的BLOB类型字段的顺序号。

data_buff..... 该缓存内的数据将会被附加到由ROW_ID、COLUMN_ORDER指定的BLOB类型字段中。

data_len..... DATA_BUFF中有效数据的长度，单位是比特。如果该长度小于DATA_BUFF的长度，DBMaster将会仅附加长度为DATA_LEN的数据，否则，将会附加缓存中的全部数据。

➲ **例**

创建一个名为**test_blob2**的表，接着插入一条记录。

```
dmSQL> CREATE TABLE test_blob2(c1 INT,c2 BLOB);
dmSQL> INSERT INTO test_blob2 VALUES(1,?);
dmSQL/Val> &file1;
1 rows inserted
dmSQL/Val> END;
```

```
dmSQL> SET AUTOMIT OFF;
dmSQL> COMMIT;
1 rows selected
dmSQL> SELECT oid,c1,BLOBLEN(c2) FROM test blob2;
             OID                     C1       BLOBLEN(C2)
============================  ===========  ===========
03000000020000000000000000000000            1        81920
1 rows selected
dmSQL> SELECT column_order FROM system.syscolumn WHERE table_name = 'TEST_BLOB2'
 AND column_name = 'C2';
COLUMN_ORDER
============
          2
1 rows selected
```

调用存储过程APPENDBLOBBYOID，将更多数据附加到file1中。

```
dmSQL> CALL APPENDBLOBBYOID('03000000020000000000000000000000'x,2,?,15);
dmSQL/Val> 'xxxxxyyyyyzzzzz';
dmSQL/Val> END;
dmSQL> COMMIT;
dmSQL> SELECT c1,BLOBLEN(c2) FROM test_blob2;
    C1       BLOBLEN(C2)
===========  ===========
          1        81935
1 rows selected
```

5.3 COPYTABLE

系统存储过程COPYTABLE用于将一张表的定义和数据复制到另一张表中。同时，源表中创建的索引、表/字段的完整性约束以及触发器等也将一同复制到目的表。

系统存储过程COPYTABLE必须在自动提交模式为开启的状态下运行。如果目的表已经存在，执行该存储过程将返回错误信息。在用户将索引重命名标记设为1的情况下，如果索引的命名是以表名为前缀的完整命名，那么复制到目的表后，索引名称将以新表的名称为前缀重新命名。如果您设置了提交数量，那么每复制n笔数据到目的表时，系统将下达一次提交命令。

错误发生时，所有在错误发生之前的操作都将被提交。没有执行的命令将会存储到spusr.log文件中。用户可以在**dmconfig.ini**中，通过**DB_SPLog**参数来设置或查看该文件的存放路径。

➲ **COPYTABLE的原型如下：**

```
COPYTABLE(VARCHAR(32) source_schema_name INPUT,
          VARCHAR(32) source_table_name INPUT,
          VARCHAR(32) destination_schema_name INPUT,
          VARCHAR(32) destination_table_name INPUT,
          VARCHAR(128) tablespace_lock_mode_option_string INPUT,
          VARCHAR(2048) where condition string INPUT,
          INT fg rename index INPUT,
          INT commit_count INPUT)
```

schema_name .. 表的模式名，当设定为NULL或空字符串时代表默认为当前用户。

table_name ..... 源表或目的表名。

tablespace_lock_mode_option_string……选择所在表空间的IN语句或类似于创建表时的锁模式语法。该字符串中的标识符必须遵守SQL语法规则。

where_condition_string…… 类似于SELECT语句中设定的WHERE条件，该字符串中的标识符必须遵守SQL语法规则。

fg_rename_index…… 索引重命名标记。该标记用来表示在源索引名以表名为前缀的情况下，是否以目的表名重新命名该索引。有效值是0或1。

commit_count……每完成n笔记录时提交，有效的取值是0~n。

➲ 例

下例中将表**Scores**中满足**Math > 70**的记录复制到位于不同表空间的表**Scores70**中。该命令不重命名索引，提交的笔数是**10**笔。

```
dmSQL> CALL COPYTABLE('SYSADM', 'Scores', 'SYSADM', 'Scores70', 'in tablespace1',
'Math > 70', 0, 10);
```

5.4 GETCPUNUMBER

系统存储过程GETCPUNUMBER用于获取逻辑处理器的个数。

通过调用系统存储过程GETCPUNUMBER和SETAFFINITY，用户可获得目前的系统状态，也可在DBMaster运行时设置连接的CPU亲和性，并且无需重启数据库。

➲ **GETCPUNUMBER的原型如下：**

```
GETCPUNUMBER (INT CPU_NUMBER OUTPUT)
```

cpu_number……输出参数，逻辑处理器个数。

➲ **例**

下例将调用存储过程GETCPUNUMBER来获得CPU的个数。

```
dmSQL> CALL GETCPUNUMBER(?);
```

5.5 GETSYSTEMOPTION

系统存储过程GetSystemOption用于获取运行时的系统选项，即在数据库运行期间，您可以使用GetSystemOption存储过程获取所有的有效系统选项值。

下表列出了通过调用系统存储过程GetSystemOption获得的所有**option_name**的系统选项值，并简要描述了每个选项名中包含的关键字。有关关键字的详细信息，请参考*数据库管理员手册*。

OPTION_NAME	简述
FODIR	系统文件对象路径（子目录）(DB_FoDir)
LGSVR	服务器日志记录级别(DB_LgSvr)
LGERR	服务器记录错误信息级别(DB_LgErr)
LGSTM	服务器执行日志语句的间隔时间(DB_LgSTm)
LGSYS	服务器日志系统信息(DB_LgSys)
LGFSZ	服务器日志文件大小 (DB_LgFSz)
LGFNO	服务器日志文件个数 (DB_LgFNo)
LGSQL	服务器日志SQL命令 (DB_LgSQL)
LGPLN	服务器日志执行计划 (DB_LgPLn)
LGPAR	服务器日志输入参数值 (DB_LgPar)
LGLCK	当锁超时或死锁发生时，服务器是否记录此时的锁信息(DB_LgLck)
LGDIR	日志服务器的路径(DB_LgDir)
LGDAY	服务器日志文件保存的天数(DB_LgDay)
LGZIP	压缩已关闭的日志文件 (DB_LgZip)
BKCHK	执行完整备份和差异备份之前是否检查数据库 (DB_ BkChk)
BKCMP	压缩备份模式(DB_BkCmp)

BKDIR	储存备份日志文件的目录 (DB_BkDir)
BKFOM	文件对象(FO)备份模式 (DB_BkFom)
BKFRM	用于增量备份时定义日志文件的命名格式 (DB_BkFrm)
BKFUL	定义备份服务被触发后进行增量备份的日志文件填充百分比(DB_BkFul)
BKITV	备份的时间间隔 (DB_BkItv)
BKODR	备份服务存放前一版本完整备份文件的路径 (DB_ BkOdr)
BKRTS	执行完整备份时，是否备份只读表空间文件 (DB_BkRTs)
BKSPM	存储过程(SP)的备份模式 (DB_BkSPm)
BKSVR	是否激活备份服务 (DB_BkSvr)
BKTIM	备份服务第一次执行增量备份的时间 (DB_BkTim)
BKZIP	执行完整备份时，备份服务器是否压缩备份文件 (DB_BkZip)
CTBLM	创建表时使用默认锁模式 (DB_CTbLM)
DBKMX	完整备份后，执行差异备份的最大值 (DB_DbKmx)
DBKTV	差异备份的时间间隔(DB_DbKtv)
DBNAME	当前连接的数据库名
DDBMD	是否在数据库服务器端启动DDB（分布式数据库）功能 (DD_DDBMd)
EATRPT	数据库服务器的订阅者后台服务使用的TCP/IP端口号(DB_EtrPt)
EXTNP	DBMaster自动扩展表空间的大小 (DB_ExtNp)
FBKTM	备份服务第一次执行完整备份的时间 (DB_FBkTm)

FBKTV	完整备份的时间间隔(DB_FBkTv)
FOSUB	存储在每个系统文件对象子目录中的最大文件对象数(DB_FoSub)
FULLBKID	完整备份ID
IDXDP	自动索引后台程序自动删除一个自动索引的阈值(DB_IdxDp)
IDXLN	自动索引后台程序自动创建一个自动索引的阈值(DB_IdxLn)
IDXTM	自动索引后台程序的启动时间(DB_IdxTm)
IDXTV	自动索引后台程序的时间间隔 (DB_IdxTv)
IDXSV	激活自动索引后台程序(DB_IdxSv)
ISOLV	用户连接数据库时的默认事务隔离级别(DB_IsoLv)
LETPT	页锁升级为表锁的锁增加阈值(DB_LetPT)
LETRP	行锁到页锁的锁增加阈值(DB_LetRP)
LIC_ACL	访问控制列表
LIC_BKSERVER	备份服务器
LIC_DBREP	数据库复制
LIC_DCI	数据库COBOL接口
LIC_DDB	分布式数据库
LIC_EDITION	版本
LIC_EXPIREDATE	许可证有效期
LIC_FREETRIAL	免费试用期
LIC_FULLTEXT	全文索引
LIC_HOSTCONN	主机连接
LIC_IOSERVER	IO服务器
LIC_LOCALE	语言环境

LIC_MAXCONN	最大连接数
LIC_MAXDBSIZE	数据库最大空间
LIC_MAXJNFSZ	日志文件的最大值
LIC_MAXPGSIZE	页面的最大值
LIC_NETZC	网络压缩
LIC_PLATFORM	平台
LIC_PRODUCT	产品名称
LIC_SERIALID	串行ID
LIC_STARTDATE	授权起始日期
LIC_UPGRADE	可升级
LIC_USERINFO	用户信息
LIC_VERSION	版本
SQLST	SQL命令监控器的显示模式(DB_SQLSt)
STARTBACKUP	选择激活备份服务器来处理完整或增量备份
STSVR	启动更新统计后台程序 (DB_StSvr)
STMOD	数据库的增量更新统计模式 (DB_StMod)
STSTM	更新统计模式的启动时间 (DB_StsTm)
STSTV	更新统计后台程序的时间间隔 (DB_StsTv)
STSSP	更新统计的数据采样率(DB_StsSp)
USRFO	在数据库中插入用户文件对象(DB_UsrFo)

➲ **GETSYSTEMOPTION的原型如下：**

```
GETSYSTEMOPTION('optionName', ?)
```

optionName.....系统选项名

➲ **例1**

下例语法用于获取备份服务器选项的值：

```
dmSQL> CALL GETSYSTEMOPTION('BKSVR',?);
```

➲ **例2**

下例语法用于获取数据库的失效日期：

```
dmSQL> CALL GETSYSTEMOPTION('LIC_EXPIREDATE',?);
```

5.6 SCHEDULE_ALTER

系统存储过程SCHEDULE_ALTER用于更改已存在的计划。

除SCHEDULE_NAME之外，其它的计划参数均可被更改。若在任务运行时更改计划参数，那么在该任务按照用户计划下次运行时，dmschsvr将载入并使用该任务的新参数值。

➲ **SCHEDULE_ALTER的原型如下：**

```
SCHEDULE_ALTER(VARCHAR(128) SCHEDULE_NAME INPUT,
               VARCHAR(128) TASK_NAME INPUT,
               VARCHAR(512) TIMETABLE INPUT,
               VARCHAR(32) STARTTIME INPUT,
               VARCHAR(32) ENDTIME INPUT)
```

schedule_name.....要更改的已存在计划的名称。

task_name.....计划所包含任务的名称。

starttime.....计划的开始日期和时间；格式是yyyy-mm-dd hh:mm:ss。

schedule_name.....计划的失效日期和时间；格式是yyyy-mm-dd hh:mm:ss。计划后台程序的最小时间单位是分，因此结束时间至少要比开始时间晚1分钟。请注意，通常用户需设定结束时间，但如果设定参数timetable的值为@once或@once m n，用户可不设置结束时间，此时系统将自动认为该时间晚于开始时间1分钟，并将（m*n+1）的值作为结束时间。

timetable.....任务的执行时间表，由以下五个域依次组成：分钟、小时、日期、月、星期，且这五个域之间需用空格隔开。这五个域的取值范围分别是0-59、0-23、1-31、1-12、0-7，且用户可使用以下通配符替换这些取值：星号（*）、逗号（,）、连字符（-）、斜线（/），详细信息如下。

—星号（*）

使用“*”表示执行时间表某域的所有取值是有效的，例如，将小时域的值设为“*”等同于为该域指定所有可能取值。

—逗号（,）

使用“,”隔开执行时间表某域的多个取值是有效的，例如，如果想每过10分钟执行一次命令，用户可将分钟域的值设为“0,10,20,30,40,50”。

—连线符（-）

使用“-”指定执行时间表某域的取值范围是有效的，例如，用户可将小时域的值设为“0-12”，表示上午每小时。

—斜线（/）

使用“/”表示固定时间间隔是有效的，例如，用户可将分钟域的值设为“*/3”，表示每3分钟。

为方便起见，可使用一些简单的特定字符来设置执行时间表，详细信息如下：

字符	简述
@minute	每分钟的第一秒钟
@hourly	每小时的第一分钟
@midnight	每天的第一分钟
@daily	每天的第一分钟
@weekly	每周一的第一分钟
@monthly	每月1日的第一分钟
@once	该任务无论是否执行成功仅执行一次。开始时间由STARTTIME指定，如果STARTTIME的值是“now()”，该任务将在下一分钟开始执行。
@once m n	该任务仅成功执行一次。如果执行失败，该任务将每n分钟执行一次，直至执行成功，且仅能最多尝试执行m次；m的取值范围是1 ~ 525600，n的取值范围是1~1440。

表 5-1 有效特定字符表

➲ 例

下例语法用于更改计划**insert_into_t1**：将执行计划“**10 0,1 * * ***”更改为“**20 2,3 * * ***”。有关计划**insert_into_t1**的详细信息，请参考*SCHEDULE_CREATE*章节。

```
dmSQL> CALL SCHEDULE_ALTER('insert_into_t1', 'insert_t1', '20 2,3 * * *', '2012-12-12 12:00:00', '2015-12-12 12:00:00'); // The task 'insert_t1' will run at 2:20 and 3:20 every day from 2012-12-12 12:00 to 2015-12-12 12:00.
```

5.7 SCHEDULE_CREATE

系统存储过程SCHEDULE_CREATE用于创建计划。

➲ **SCHEDULE_CREATE的原型如下：**

```
SCHEDULE_CREATE(VARCHAR(128) SCHEDULE_NAME INPUT,
                VARCHAR(128) TASK_NAME INPUT,
                VARCHAR(512) TIMETABLE INPUT,
                VARCHAR(32) STARTTIME INPUT,
                VARCHAR(32) ENDTIME INPUT)
```

schedule_name……要创建的计划名称，最多可包括128个字符，由字母、数字、下划线组成， 但第一个字符不能是数字。

task_name……计划所包含任务的名称。

starttime……计划的开始日期和时间；格式是yyyy-mm-dd hh:mm:ss。

endtime……计划的失效日期和时间；格式是yyyy-mm-dd hh:mm:ss。计划后台程序的最小时间单位是分，因此结束时间至少要比开始时间晚1分钟。请注意，通常用户需设定结束时间，但如果设定参数timetable的值为@once或@once m n，用户可不设置结束时间，此时系统将自动认为该时间晚于开始时间1分钟，并将（m*n+1）的值作为结束时间。

timetable……任务的执行时间表，由以下五个域依次组成：分钟、小时、日期、月、星期，且这五个域之间需用空格隔开。这五个域的取值范围分别是0-59,、0-23、1-31、1-12、0-7，且用户可使用以下通配符替换这些取值：星号（*）, 逗号（, ）, 连字符（-）、斜线（/），详细信息如下。

—星号（*）

使用“*”表示执行时间表某域的所有取值是有效的，例如，将小时域的值设为“*”等同于为该域指定所有可能取值。

—逗号（,）

使用“,”隔开执行时间表某域的多个取值是有效的，例如，如果想每过10分钟便执行一次命令，用户可将分钟域的值设为“0,10,20,30,40,50”。

—连线符（-）

使用“-”指定执行时间表某域的取值范围是有效的，例如，用户可将小时域的值设为“0-12”，表示上午每小时。

—斜线（/）

使用“/”表示固定时间间隔是有效的，例如，用户可将分钟域的值设为“*/3”，表示每3分钟。

为方便起见，可使用一些简单的特定字符来设置执行时间表，详细信息如下：

字符	简述
@minute	每分钟的第一秒钟
@hourly	每小时的第一分钟
@midnight	每天的第一分钟
@daily	每天的第一分钟
@weekly	每周一的第一分钟
@monthly	每月1日的第一分钟
@once	该任务无论是否执行成功仅执行一次。开始时间由STARTTIME指定，如果STARTTIME的值是“now()”，该任务将在下一分钟开始执行。
@once m n	该任务仅成功执行一次。如果执行失败，该任务将每n分钟执行一次，直至执行成功，且仅能最多尝试执行m次；m的取值范围是1 ~ 525600，n的取值范围是1~1440。

表 5-2 有效特定字符表

➲ 例

下例语法用于为任务**insert_t1**创建计划**insert_into_t1**。有关任务**insert_t1**的详细信息，请参考*TASK_CREATE*章节。

```
dmSQL> CALL SCHEDULE_CREATE('insert_into_t1', 'insert_t1', '10 0,1 * * *', '2012-12-12 12:00:00', '2015-12-12 12:00:00'); // The task 'insert_t1' will run at 0:10 and 1:10 every day from 2012-12-12 12:00 to 2015-12-12 12:00.
```

5.8 SCHEDULE_DISABLE

系统存储过程SCHEDULE_DISABLE用于禁用计划。

如果计划被禁用，该计划中的原数据仍然存在，但它不能被运行且dmschsvr不能为进程载入该计划。此时，该计划在系统表中的状态被更改为禁用。除一次性计划外，其它新创建的计划默认均为禁止。

➲ **SCHEDULE_DISABLE的原型如下：**

```
SCHEDULE_DISABLE(VARCHAR(128) SCHEDULE_NAME INPUT)
```

schedule_name……要禁用的计划名称。

➲ **例**

下例语法用于禁用计划**insert_into_t1**。有关计划**insert_into_t1**的详细信息，请参考*SCHEDULE_CREATE*章节。

```
dmSQL> CALL SCHEDULE_DISABLE('insert_into_t1');
```

5.9 SCHEDULE_DROP

系统存储过程SCHEDULE_DROP用于删除已存在计划。如果计划被删除，其存储在SYSSCHEDULE中的相关记录也会被删除。

➲ **SCHEDULE_DROP的原型如下：**

```
SCHEDULE_DROP(VARCHAR(128) SCHEDULE_NAME INPUT)
```

schedule_name……要删除的计划名称。

➲ **例**

下例语法用于删除计划**insert_into_t1**。有关计划**insert_into_t1**的详细信息，请参考*SCHEDULE_CREATE*章节。

```
dmSQL> CALL SCHEDULE_DROP('insert_into_t1');
```

5.10 SCHEDULE_ENABLE

系统存储过程SCHEDULE_ENABLE用于启用计划。

调用该存储过程，dmschsvr将载入该计划。通常新创建的计划默认为禁用，因此用户在运行计划之前需要先启用计划。

➲ **SCHEDULE_ENABLE的原型如下：**

```
SCHEDULE_ENABLE(VARCHAR(128) SCHEDULE_NAME INPUT)
```

schedule_name……要启用的计划名称。

➲ **例**

下例语法用于启用计划**insert_into_t1**。有关计划**insert_into_t1**的详细信息，请参考*SCHEDULE_CREATE*章节。

```
dmSQL> CALL SCHEDULE_ENABLE('insert_into_t1');
```

5.11 SCHEDULE_RELOAD

系统存储过程SCHEDULE_RELOAD用于重新载入系统中所有已启用的计划。Dmschsvr每分钟会自动检测是否存在被更改或新创建的计划，如果有，将会重新载入所有已启用的计划。

➲ **SCHEDULE_RELOAD的原型如下：**

```
SCHEDULE_RELOAD
```

➲ **例**

下例语法用于重新载入系统中所有已启用的计划。

```
dmSQL> CALL SCHEDULE_RELOAD;
```

5.12 SCHEDULE_CLEAN

系统存储过程SCHEDULE_CLEAN用于清除多余日志，仅保存最近的日志。只有拥有DBA权限或更高权限的用户才可以调用该存储过程。

➲ **SCHEDULE_CLEAR的原型如下：**

```
SCHELOG_CLEAN(INT RESERVE_DAY INPUT)
```

reserve_day……要删除计划日志的创建时间和最近计划日志之间的天数。计划日志存储在SYSSCHELOG中。reserve_day的取值范围是0~7300（20年）。

➲ **例**

用户最近的日志是10天前创建的，下例语法用于清除创建日期比最近日志早20天的日志，也就是30天前创建的日志。

```
dmSQL> CALL SCHELOG_CLEAN(20);
```

5.13 SETAFFINITY

系统存储过程SETAFFINITY用于设置进程和线程的CPU亲和性。请注意只有SYSADM有权调用该系统存储过程。

通过调用系统存储过程GETCPUNUMBER和SETAFFINITY，用户可获得目前的系统状态，也可在DBMaster运行时设置连接的CPU亲和性，并且无需重启数据库。

CPU亲和性使用亲和性掩码来定义，亲和性掩码中的每一位都代表一个处理器。在DBMaster中，亲和性掩码被定义为char(64)，所以最多有64个CPU。

➲ **SETAFFINITY的原型如下：**

```
SETAFFINITY (INT CONNECTION_ID INPUT, CHAR(64) AFFINITY_MASK INPUT)
```

connection_id……输入参数，连接或服务器的ID。用户可通过命令“select connection_id from sysuser”或检查系统监测器获得该ID。在WINDOWS系统下，该ID是线程ID，在类UNIX系统下，该ID是处理器ID。

affinity_mask……输入参数，CPU亲和性掩码。有效的亲和性掩码由“1”和“0”组成。“1”表示该CPU对于连接有效，“0”表示该CPU对于连接无效。

➲ **例1**

下例是一个8-CPU系统的亲和性掩码。（高位连续的0省略。）

Decimal value	Binary bit mask	Allow run on CPU
1	'1'	0
3	'11'	0和1
7	'111'	0、1和2
15	'1111'	0、1、2和3
31	'11111'	0、1、2、3和4
63	'111111'	0、1、2、3、4和5
127	'1111111'	0、1、2、3、4、5和6
255	'11111111'	0、1、2、3、4、5、6和7

➲ **例2**

设置CPU亲和性之前，用户必须获得某些系统信息，如服务器的CPU个数、每个连接的CPU使用、正确的亲和性掩码。

调用GETCPUNUMBER获得CPU个数：

```
dmSQL> CALL GETCPUNUMBER(?);
```

获得每个连接的CPU使用、正确的亲和性掩码：

```
dmSQL> SELECT connection_id, affinity_mask, priority_level, cpu_usage FROM
sysuser;
```

设置CPU亲和性，使连接运行在0号和1号CPU：

```
dmSQL> SELECT connection_id , user_name FROM sysuser;

    CONNECT*              USER_NAME
================   ======================
          30420    BACKUP_SERVER
          30418    SYSADM

2 rows selected
dMSQL> CALL SETAFFINITY(30418,'11');
```

通过查询sysuser，可获得ID已知连接的CPU亲和性掩码：

```
dmSQL> SELECT affinity_mask FROM sysuser WHERE connection_id = ?;
```

5.14 SETPRIORITY

系统存储过程SETPRIORITY用于设置进程和线程的优先级。请注意只有SYSADM有权调用该系统的存储过程。

通过调用系统存储过程SETPRIORITY，用户可在DBMaster运行时设置连接的优先级而无需重启数据库。请注意，在LINUX系统下升高优先级需要用户拥有root权限，因此在LINUX系统下用户不能升高优先级，而只能降低优先级，但是在WINDOWS系统下无此限制。

➲ **SETPRIORITY的原型如下：**

```
GETCPUNUMBER (INT CONNECTION_ID INPUT,INT PRIORITY_LEVEL INPUT)
```

connection_id……输入参数，连接或服务器的ID。用户可通过命令“select connection_id from sysuser”或检查系统监测器获得该ID。在WINDOWS系统下，该ID是线程ID，在类UNIX系统下，该ID是处理器ID。

priority_level……...输入参数，有五个级别（‘1’、‘2’、‘3’、‘4’和‘5’）可选，‘1’表示最低优先级，‘2’表示较低优先级，‘3’表示正常优先级，‘4’表示较高优先级，‘5’表示最高优先级。默认优先级为***3***。

➲ **例**

设置优先级之前，用户必须获得某些系统信息，如服务器的CPU个数、每个连接的CPU使用、正确的亲和性掩码、优先级。

调用GETCPUNUMBER，获得CPU个数：

```
dmSQL> CALL GETCPUNUMBER(?);
```

获得每个连接的CPU使用、正确的亲和性掩码：

```
dmSQL> SELECT connection id, affinity mask, priority level, cpu usage FROM
sysuser;
```

设置优先级级别：

```
dmSQL> SELECT connection_id , user_name FROM sysuser;

    CONNECT*              USER_NAME
```

```
================   ========================
          30420    BACKUP SERVER
          30418    SYSADM

2 rows selected
dmSQL> CALL SETPRIORITY(30418,'11');
```

通过查询sysuser，可获得ID已知连接的优先级级别：

```
dmSQL> SELECT affinity_mask FROM sysuser WHERE connection_id = ?;
```

5.15 SETSYSTEMOPTION

系统存储过程SetSystemOption用于在运行时设置系统选项。即在数据库运行期间，您可以使用SetSystemOption存储过程更改这些有效的系统选项值。用户可通过调用GetSystemOption来获取系统选项的值。

下表列出了通过调用系统存储过程SetSystemOption获得的所有**option_name**的系统选项值，并简要描述了每个选项名中包含的关键字。有关关键字的详细信息，请参考*数据库管理员手册*。

OPTION_NAME	简述
FODIR	在线更改系统文件对象的存储路径(DB_FoDir)，选项名称为完整路径。如空字符串' ',会使系统文件对象的功能失效。
LGSVR	服务器日志记录级别(DB_LgSvr)
LGERR	服务器记录错误信息级别(DB_LgErr)
LGSTM	服务器执行日志语句的间隔时间(DB_LgSTm)
LGSYS	服务器日志系统信息(DB_LgSys)
LGFSZ	服务器日志文件大小 (DB_LgFSz)
LGFNO	服务器日志文件个数 (DB_LgFNo)
LGSQL	服务器日志SQL命令 (DB_LgSQL)
LGPLN	服务器日志执行计划 (DB_LgPLn)
LGPAR	服务器日志输入参数值 (DB_LgPar)
LGLCK	当锁超时或死锁发生时，服务器是否记录此时的锁信息(DB_LgLck)
LGDIR	日志服务器的路径(DB_LgDir)
LGDAY	服务器日志文件保存的天数(DB_LgDay)
LGZIP	压缩已关闭的日志文件 (DB_LgZip)
BKCHK	执行完整备份和差异备份之前是否检查数据库

	(DB_ BkChk)
BKCMP	压缩备份模式(DB_BkCmp)
BKDIR	储存备份日志文件的目录 (DB_BkDir)
BKFOM	文件对象(FO)备份模式 (DB_BkFom)
BKFRM	用于增量备份时定义日志文件的命名格式 (DB_BkFrm)
BKFUL	定义备份服务被触发后进行增量备份的日志文件填充百分比(DB_BkFul)
BKITV	备份的时间间隔 (DB_BkItv)
BKODR	备份服务存放前一版本完整备份文件的路径 (DB_ BkOdr)
BKRTS	执行完整备份时，是否备份只读表空间文件 (DB_BkRTs)
BKSPM	存储过程(SP)的备份模式 (DB_BkSPm)
BKSVR	是否激活备份服务 (DB_BkSvr)
BKTIM	备份服务第一次执行增量备份的时间 (DB_BkTim)
BKZIP	执行完整备份时，备份服务器是否压缩备份文件 (DB_BkZip)
CTBLM	创建表时使用默认锁模式 (DB_CTbLM)
DBKMX	完整备份后，执行差异备份的最大值 (DB_DbKmx)
DBKTV	差异备份的时间间隔(DB_DbKtv)
DDBMD	是否在数据库服务器端启动DDB（分布式数据库）功能 (DD_DDBMd)
EXTNP	DBMaster自动扩展表空间的大小 (DB_ExtNp)
FBKTM	备份服务第一次执行完整备份的时间 (DB_FBkTm)
FBKTV	完整备份的时间间隔(DB_FBkTv)

FOSUB	存储在每个系统文件对象子目录中的最大文件对象数(DB_FoSub)
IDXDP	自动索引后台程序自动删除一个自动索引的阈值(DB_IdxDp)
IDXLN	自动索引后台程序自动创建一个自动索引的阈值(DB_IdxLn)
IDXTM	自动索引后台程序的启动时间(DB_IdxTm)
IDXTV	自动索引后台程序的时间间隔 (DB_IdxTv)
IDXSV	激活自动索引后台程序(DB_IdxSv)
LETPT	页锁升级为表锁的锁增加阈值(DB_LetPT)
LETRP	行锁到页锁的锁增加阈值(DB_LetRP)
LIC_RELOAD	重新下载许可证
SQLST	SQL命令监控器的显示模式(DB_SQLSt)
STSVR	启动更新统计后台程序 (DB_StSvr)
STMOD	数据库的增量更新统计模式 (DB_StMod)
STSTM	更新统计模式的启动时间 (DB_StsTm)
STSTV	更新统计后台程序的时间间隔 (DB_StsTv)
STSSP	更新统计的数据采样率(DB_StsSp)
STS_ABORT	终止进行中的更新统计
USRFO	在数据库中插入用户文件对象(DB_UsrFo)

➲ **SETSYSTEMOPTION的原型如下：**

```
SETSYSTEMOPTION(VARCHAR(32) OPTION_NAME INPUT,
                VARCHAR(8576) OPTION_VALUE INPUT)
```

option_name……系统选项名。

option_value……系统选项值。

➲ **例1**

下例语法用于激活备份服务器：

```
dmSQL> CALL SETSYSTEMOPTION('BKSVR','1');
```

➲ **例2**

备份服务器处于激活状态时，使用下例语法设置**dmconfig.ini**配置文件中的备份参数：

```
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','1'); //do full backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','2'); //do incremential backup
dmSQL> CALL SETSYSTEMOPTION('STARTBACKUP','3'); //do differential backup
```

➲ **例3**

数据库运行时使用下例语法将更新统计样例设置为**60**：

```
dmSQL> CALL SETSYSTEMOPTION('STSSP', '60');
```

➲ **例4**

使用下例语法中断处于运行状态的更新统计：

```
dmSQL> CALL SETSYSTEMOPTION('STS_ABORT', '14076'); // abort an ongoing update
statistics which connection ID is 14076.
dmSQL> CALL SETSYSTEMOPTION('STS_ABORT', '0'); // the value 0 is a special
connection ID.It means abort all command related to update statistics.
```

➲ **例5**

使用下例语法激活自动索引后台程序：

```
dmSQL> CALL SETSYSTEMOPTION ('IDXSV', '1');
```

➲ **例6**

激活自动索引后台程序之后，使用下例语法在**dmconfig.ini**配置文件中设置合适的自动索引后台程序参数：

```
dmSQL> CALL SETSYSTEMOPTION('IDXTM', '2012-12-12 00:00:00'); // The first time
the auto index daemon starts for the first time at 2012-12-12 00:00:00.
dmSQL> CALL SETSYSTEMOPTION('IDXTV', '2-00:00:00'); // The interval of performing
the auto index daemon is 2 days.
dmSQL> CALL SETSYSTEMOPTION('IDXDP', '60'); // An index which is not used reaches
or exceeds 60 days will be dropped by auto index daemon.
dmSQL> CALL SETSYSTEMOPTION('IDXLN', '10'); // If the same scan log number
reaches or exceeds 10, an auto index will be created according to these log.
```

➲ **例7**

如果用户要更新许可证，调用如下函数重新下载许可证：

```
dmSQL> CALL SETSYSTEMOPTION('LIC_RELOAD', '1');
```

5.16 SETSYSTEMOPTIONW

系统存储过程SetSystemOptionW用于在运行时设置系统选项，并将动态设置写入**dmconfig.ini**文件。该存储过程是对SetSystemOption的扩展，支持SetSystemOption可更改的所有系统选项。下表列出了通过调用系统存储过程SetSystemOptionW获得的**option_name**的系统选项值，并简要描述了每个选项名中包含的关键字。有关关键字的详细信息，请参考*数据库管理员手册*。

数据库运行期间，用户可以调用系统存储过程setSystemOption()设置以上系统选项，也可以调用系统存储过程setSystemOptionW()来设置以上系统选项并将其写入**dmconfig.ini**文件。此外，用户还可以通过调用系统存储过程getSystemOption()获取系统选项信息。

OPTION_NAME	简述
FODIR	在线更改系统文件对象的存储路径(DB_FoDir)，选项名称为完整路径。如空字符串' ',会使系统文件对象的功能失效。
LGSVR	服务器日志记录级别(DB_LgSvr)
LGERR	服务器记录错误信息级别(DB_LgErr)
LGSTM	服务器执行日志语句的间隔时间(DB_LgSTm)
LGSYS	服务器日志系统信息(DB_LgSys)
LGFSZ	服务器日志文件大小 (DB_LgFSz)
LGFNO	服务器日志文件个数 (DB_LgFNo)
LGSQL	服务器日志SQL命令 (DB_LgSQL)
LGPLN	服务器日志执行计划 (DB_LgPLn)
LGPAR	服务器日志输入参数值 (DB_LgPar)
LGLCK	当锁超时或死锁发生时，服务器是否记录此时的锁信息(DB_LgLck)

LGDIR	日志服务器的路径(DB_LgDir)
LGDAY	服务器日志文件保存的天数(DB_LgDay)
LGZIP	压缩已关闭的日志文件 (DB_LgZip)
BKCHK	执行完整备份和差异备份之前是否检查数据库 (DB_ BkChk)
BKCMP	压缩备份模式(DB_BkCmp)
BKDIR	储存备份日志文件的目录 (DB_BkDir)
BKFOM	文件对象(FO)备份模式 (DB_BkFom)
BKFRM	用于增量备份时定义日志文件的命名格式 (DB_BkFrm)
BKFUL	定义备份服务被触发后进行增量备份的日志文件填充百分比(DB_BkFul)
BKITV	备份的时间间隔 (DB_Bkltv)
BKODR	备份服务存放前一版本完整备份文件的路径 (DB_ BkOdr)
BKRTS	执行完整备份时，是否备份只读表空间文件 (DB_BkRTs)
BKSPM	存储过程(SP)的备份模式 (DB_BkSPm)
BKSVR	是否激活备份服务 (DB_BkSvr)
BKTIM	备份服务第一次执行增量备份的时间 (DB_BkTim)
BKZIP	执行完整备份时，备份服务器是否压缩备份文件 (DB_BkZip)
CTBLM	创建表时使用默认锁模式 (DB_CTbLM)
DBKMX	完整备份后，执行差异备份的最大值 (DB_DbKmx)
DBKTV	差异备份的时间间隔(DB_DbKtv)
DDBMD	是否在数据库服务器端启动DDB（分布式数据库）功能 (DD_DDBMd)

EXTNP	DBMaster自动扩展表空间的大小 (DB_ExtNp)
FBKTM	备份服务第一次执行完整备份的时间 (DB_FBkTm)
FBKTV	完整备份的时间间隔(DB_FBkTv)
FOSUB	存储在每个系统文件对象子目录中的最大文件对象数(DB_FoSub)
IDXDP	自动索引后台程序自动删除一个自动索引的阈值 (DB_IdxDp)
IDXLN	自动索引后台程序自动创建一个自动索引的阈值 (DB_IdxLn)
IDXTM	自动索引后台程序的启动时间(DB_IdxTm)
IDXTV	自动索引后台程序的时间间隔 (DB_IdxTv)
IDXSV	激活自动索引后台程序(DB_IdxSv)
LETPT	页锁升级为表锁的锁增加阈值(DB_LetPT)
LETRP	行锁到页锁的锁增加阈值(DB_LetRP)
SQLST	SQL命令监控器的显示模式(DB_SQLSt)
STSVR	启动更新统计后台程序 (DB_StSvr)
STMOD	数据库的增量更新统计模式 (DB_StMod)
STSTM	更新统计模式的启动时间 (DB_StsTm)
STSTV	更新统计后台程序的时间间隔 (DB_StsTv)
STSSP	更新统计的数据采样率(DB_StsSp)
USRFO	在数据库中插入用户文件对象(DB_UsrFo)

➲ SETSYSTEMOPTIONW的原型如下：

```
SETSYSTEMOPTIONW(VARCHAR(32) OPTION_NAME INPUT,
                 VARCHAR(8576) OPTION_VALUE INPUT)
```

option_name……系统选项名。

option_value……系统选项值。

➲ 例

要开启更新统计后台程序，用户需调用**setSystemOptionW**将STSVR的值动态设置为**1**，该动态设置**STSVR=1**将会被写入**dmconfig.ini**文件，如下所示：

调用存储过程**setSystemOptionW**前，**dmconfig.ini**文件如下：

```
[DBSAMPLE5]
; Here omit other keywords
DB_StSvr = 0
```

执行调用**setSystemOptionW('optionName', 'optionValue')**：

```
dmSQL> CALL SETSYSTEMOPTIONW('STSVR', '1');
dmSQL> CALL SETSYSTEMOPTION ('STSVR',?);
OPTION_VALUE : 1
```

调用存储过程**setSystemOptionW**后，**dmconfig.ini**文件如下：

```
[DBSAMPLE5]
; Here omit other keywords
DB_StSvr = 1
```

5.17 SOADD

系统存储过程SOADD的作用是增加共享对象的值。

SOADD的原型如下：

```
SOADD(INTEGER SHID,
      INTEGER ADDEND,
      INTEGER NEW_VAL OUTPUT)
```

shid.....共享对象的编号（id）。

addend... ...增加的数值，可以是正值也可以是负值。

new_val.....共享对象增加后的值。

➲ 例

下例在id为2的共享对象上增加3后，得到的新值等于3。

```
dmSQL> CALL SYSADM.SOADD(2,3,?);
new_val: 3
```

5.18 SOCREATE

系统存储过程SOCREATE的作用是生成共享对象。在使用共享对象前，您可以使用SOCreate（）来生成一个具有指定编号和初始值的共享对象。然后使用该共享对象编号来调用系统存储过程SORead（）、SOSet（）或SOAdd（）（分别用来读取、修改或增加共享对象的值）。由于任何连接在数据库中的用户都可以存取共享对象，因此共享对象必须能支持SOLock（）和SOUnlock以便控制事务的并发性。当不再使用某个共享对象时，就可以使用SODrop（）来删除它。

➲ **SOCREATE的原型如下：**

```
SOCREATE(INTEGER SETID,
         INTEGER INIT_VAL,
         INTEGER SHID OUTPUT)
```

setid……为共享对象分配的id。其中，0表示由系统分配id，其它表示由用户分配id。

init_val……共享对象的初始值。

shid……分配给所建共享对象的id，这是该系统存储过程的输出值。

➲ **例1**

下例将创建一个初始值为0的共享对象，由系统分配的id为0。

```
dmSQL> CALL SYSADM.SOCREATE(0,0,?);
Shid: 1
```

➲ **例2**

下例将创建一个id为2，初始值为0的共享对象。

```
dmSQL> CALL SYSADM.SOCREATE(2,0,?);
Shid: 2
```

5.19 SODROP

当不再使用某个对象时，您就可以使用系统存储过程SODROP来删除这个共享对象。

➲ **SODROP的原型如下：**

```
SODROP(INTEGER SHID)
```

shid ..... 要删除的共享对象的id。

➲ **例**

您可以使用下面的语法来删除id为**1**的共享对象。

```
dmSQL> CALL SYSADM.SODROP(1);
```

5.20 SOLOCK

系统存储过程SOLOCK的作用是锁定一个共享对象。如果一个共享对象被加了锁，那么其他用户就不能再读取、设置或增加该对象的值，不能删除该对象，也不能再在这个对象上加锁或解除该对象的锁定。此时，只有设置锁定的那个用户才可以在这个共享对象上使用另外6个系统存储过程。

➲ **SOLOCK的原型如下：**

```
SOLOCK(INTEGER SHID)
```

shid……要锁定的共享对象id。

➲ **例**

您可以使用下面的语法来锁定id为**1**的共享对象。

```
dmSQL> CALL SYSADM.SOLOCK(1);
```

5.21 SOREAD

系统存储过程SOREAD的作用是读取（获得）一个共享对象中的值。

➲ **SOREAD的原型如下：**

```
SOREAD(INTEGER SHID,
       INTEGER VAL OUTPUT)
```

shid……共享对象的id。

val……..共享对象的值。

➲ **例**

您可以使用下面语法来获得id为**2**的共享对象值。

```
dmSQL> CALL SYSADM.SOREAD(2,?);
val: 3
```

5.22 SOSET

您可以使用系统存储过程SOSET来设置或修改共享对象的值。

➲ **SOSET的原型如下：**

```
SOSET(INTEGER SHID,
      INTEGER NEW_VAL,
      INTEGER OLD_VAL OUTPUT)
```

shid……共享对象的id。

new_val…….共享对象新赋的值。

old_val……..共享对象原来的值，这是该存储过程的输出值。

➲ **例**

您可以用下面的语法来为id是**2**的共享对象设置一个新值**-2**。

```
dmSQL> CALL SYSADM.SOSET(2,-2,?);
old_val: 3
```

5.23 SOUNLOCK

系统存储过程SOUNLOCK可用来解除某个共享对象上的锁定。一个共享对象被加锁后，其他用户就不能再读取、设置或增加该对象的值，不能删除该对象，也不能再在这个对象上加锁或解除该对象的锁定。只有在这个共享对象上设置锁定的用户才可以解除锁定。

➲ **SOUNLOCK的原型如下：**

```
SOUNLOCK(INTEGER SHID)
```

shid……要解除锁定的共享对象的id。

➲ **例**

您可以使用如下语法来解除共享对象1上的锁定。

```
dmSQL> CALL SYSADM.SOUNLOCK(1);
```

5.24 START_DMSCHSVR

系统存储过程START_DMSCHSVR用于启动dmschsvr。

➲ **START_DMSCHSVR的原型如下：**

```
START_DMSCHSVR(VARCHAR(8) TASKRUNNUM INPUT,
               VARCHAR(128) SCHELOGDIR INPUT)
```

taskrunnum……dmschsvr可同时唤醒的任务数。取值范围是1~50，默认值是***30***。

schelogdir……该路径是dmschsvr日志文件目录。其默认路径和**DB_DbDir**指定的路径相同。日志文件名格式是<DB_NAME><_><Date>，例如，DBSAMPLE5_20150135.log。

➲ **例**

下例语法用于启动**dmschsvr**。

```
dmSQL> CALL START_DMSCHSVR ('30', 'C:\DBMaster\5.4\SAMPLES\DATABASE');
```

5.25 STOP_DMSCHSVR

系统存储过程STOP_DMSCHSVR用于停止dmschsvr。

➲ **STOP_DMSCHSVR的原型如下：**

```
STOP_DMSCHSVR
```

➲ **例**

下例语法用于停止**dmschsvr**。

```
dmSQL> CALL STOP_DMSCHSVR;
```

5.26 TASK_ALTER

系统存储过程TASK_ALTER用于更改已存在的任务。

除TASK_NAME之外，其它的任务参数均可被更改。若在任务运行时更改任务参数，那么在该任务按照用户计划下次运行时，该任务将使用新的参数值。

➲ **TASK_ALTER的原型如下：**

```
TASK_ALTER(VARCHAR(128) TASK_NAME INPUT,
           VARCHAR(16) TASK_TYPE INPUT,
          VARCHAR(2048) ACTIONS INPUT)
```

task_name……要更改的已存在的任务名称。

task_type……任务类型。有三个选项：SQL_ STATEMENT（缩写为SQL）、STORE_PROCEDURE（缩写为SP）、EXECUTABLE（缩写为EXEC）。SQL_ STATEMENT表示该任务是SQL语句；STORE_PROCEDURE表示该任务是存储过程；EXECUTABLE表示该任务是可执行程序。

actions……已存在的定期执行任务的行为，该行为必须与任务类型相匹配。

➲ **例**

下例语法用于更改任务**insert_t1**：将行为“**INSERT INTO t1 VALUES(1, 2)**”更改为“**INSERT INTO t1 VALUES(1, 3)**”。有关任务**insert _t1**的详细信息，请参考*TASK_CREATE*章节。

```
dmSQL> CALL TASK_ALTER('insert_t1','SQL_STATEMENT','INSERT INTO t1 VALUES(1,3)');
```

5.27 TASK_CREATE

系统存储过程TASK_CREATE用于创建任务。任务是用户自己定义的，且按照计划运行一次或多次。一个任务是需要执行的行为集合，由计划执行。

➲ **TASK_CREATE的原型如下：**

```
TASK_CREATE(VARCHAR(128) TASK_NAME INPUT,
            VARCHAR(16) TASK_TYPE INPUT,
            VARCHAR(2048) ACTIONS INPUT)
```

task_name……要创建的任务名称，最多可包括128个字符，由字母、数字、下划线组成， 但第一个字符不能是数字。

task_type……要创建的任务类型。有三个选项：SQL_ STATEMENT（缩写为SQL）、STORE_PROCEDURE（缩写为SP）、EXECUTABLE（缩写为EXEC）。SQL_ STATEMENT表示该任务是SQL语句；STORE_PROCEDURE表示该任务是存储过程；EXECUTABLE表示该任务是可执行程序。

actions……将要定期执行任务的行为，该行为必须与任务类型相匹配，且最大值是2K比特。

➲ **例**

下例语法用于创建任务**insert_t1**，该任务的作用是向表**t1**中插入数据。

```
dmSQL> CALL TASK CREATE('insert t1','SQL STATEMENT','INSERT INTO t1
VALUES(1,2)');
```

5.28 TASK_DROP

系统存储过程TASK_DROP用于删除已存在的任务。

若该任务已被添加到计划中，当用户删除该任务时错误将会发生，也就是说，在删除任务前，用户需确认该任务没有被使用。

➲ **TASK_CREATE的原型如下：**

```
TASK_DROP(VARCHAR(128) TASK_NAME INPUT)
```

task_name……要删除的任务名称。

➲ **例**

下例语法用于删除任务**insert_t1**。有关任务**insert_t1**的详细信息，请参考*TASK_CREATE*章节。

```
dmSQL> CALL TASK_DROP('insert_t1');
```

5.29 XMLEXPORT

系统存储过程XMLEXPORT为用户提供一个可编程的界面，用以导出DBMaster中的XML数据。只有SYSADM、SYSDBA或DBA才有权调用这个存储过程。此外，因为XMLEXPORT是系统存储过程，所以不能将这个存储过程的执行权授予其他用户。

XMLEXPORT可以将DBMaster数据库中的表输出到一个XML文件中，还可以一次调用相关存储过程来处理多个表。描述串（description string）中概述了有关XML文件和DBMaster表之间的映射描述。这个描述串将作为一个自变量传递到存储过程中。

➲ **XMLEXPORT的原型：**

```
XMLEXPORT(VARCHAR(256) FILE_PATH,
          VARCHAR(256) DB_TAG,
          VARCHAR(256) XML_HEADER,
          VARCHAR(16000) OBJECT_STR,
          VARCHAR(256) OPTION_STR,
          VARCHAR(256) LOG_PATH)
```

名称	类型	长度(字节)	描述	是否区分大小写
file_path	varchar	256	导出目的xml文件的完整路径	依操作系统而定
db_tag	varchar	256	用户自己定制的数据库标签	区分（输出结果拥有同样的大写字母）
xml_header	varchar	256	用户自己定制的xml标题	区分（输出结果拥有同样的大写字母）
object_str	varchar	16000	被导出的对象的描述串	取决于DBMaster的设置
option_flag	varchar	256	选项标记的描述串	不区分
log_path	varchar	256	客户端错误日志文件的全路径	依操作系统而定

表 5-2 XMLEXPORT自变量列表

构造XMLEXPORT自变量

第一：从数据库中导出的XML文件必须在数据库服务器上生成。file_path是一个完整路径字符串，它作为一个自变量传入相关的存储过程中。

第二：db_tag可用来定制标签。如果没有给这个自变量赋值或赋的值为空字符串，那么系统将采用这个自变量的默认值（数据库的名称）。

第三：自变量object_str的用法：

```
Object_str=:
     { <element> [; <element>…]

<element>=:
     {TABLE_NAME | <select_query>} [#TABLE_TAG]
```

一个元素<element>代表一张表。两个<element>之间的分隔符是分号（；）。如果<element>中的第一个记号是“select”（大小写不敏感），那么这个<element> 则被看作<select_query> [#TABLE_TAG]。否则这个<element>将被看作TABLE_NAME [#TABLE_TAG]。如果<element>是TALBE_NAME [#TABLE_TAG]，那么表的所有字段都将被选中，用户也不可以为每个字段指定自己定制的标签。也就是说，在导出的XML文件中，字段的标签和表中对应字段的名称相同。用户可以使用TABLE_TAG来指定一个自己定制的表标签。如果没有TABLE_TAG，数据库中的表名称则将作为XML文件中的表标签。

如果您想在XML中使用自己定制的字段标签，那么就必须在<element>串中使用<select_query>[#TABLE_TAG]。您可以在<select_query>语句中使用字段的别名，这样就可以在XML文件中用自己定制的字段标签。此时，用户必须在<select_query>中使用“AS”，例如：将“select c1 as name, c2 as type from t2 ”作为<select_query>语句，这样在导出的XML文件中，字段c1的名称将变为“name”标签，字段c2的名称将变成“type”。

第四，用户可以使用选项标记来设置存储过程的选项。每个选项由分号（；）分割。例如，如果您希望将字段名称作为XML元素的属性，在选项串中就应该使用“column_as_attribute”。如果用户不指定选项，那么该选项就是NOT SET，选项串对大小写不敏感。

选项标记	SET	NOT SET
blob_in_separate_file	BLOB/CLOB字段的数据将被导出到不同于XML文件的一个临时文件中。该临时文件的文件名记录在导出文件类型定义中（DTD）。	Blob/Clob字段的数据将被作为XML文件的一部分导出到XML文件中。
column_as_attribute	XML文件中的字段不被作为元素，而是被作为元素的属性。	字段将作为元素导出到XML文件中。
capitalize_tag_name	XML文件中的所有标签名都大写。	所有标签名的大小写和数据库中相应名称的大小写吻合。
file_type_as_link	File类型数据的内容不会被导出，而只将文件名称导出到XML文件中。	File类型数据的内容将被作为XML文件的一部分导出。
no_schema_dtd	并不在生成XML文件的同时，生成结构DTD。	生成XML文件的同时将生成相应的DTD。

表 5-3 XMLEXPORT选项

最后导出XML文件时所生成的日志文件被保存在客户端机器上的log_path目录下。

导出XML文件

假设我们想从一个DBMaster数据库*Customer*中导出两张表（其中一张表叫*card*，另一张叫*contact*）到/usr/john/xmlexport.xml文件中。在这个xmlexport.xml文件中，我们用“EMPLOYEE”作为数据库的自定义标签，“TITLE”是用户为表card自定义的标签，“NUMBER”则是用户为表contact自定义的标签。

此外，我们为表**tb_card**的字段C1、C2、C3以及C4所设置的自定义标签分别是：NUM、FIRST_NAME、LAST_NAME以及JOB，而对于表

tb_contact的字段，我们并没有设置自定义标签。同时，XML文件中的所有标签都要求是大写，所有的BLOB字段的数据都将存储在另外的临时文件中。最后，日志文件将存储为/client/john/xmlexport.log。两个表的内容如下：

```
dmSQL> SELECT * FROM tb_card;
 id              fname              lname                    work
========== ==================== ==================== ====================
         1 Eddie                Chang                Manager
         2 Hook                 Hu                   SoftwareEngineer
         3 ackie                Yu                   SoftwareEngineer
         8 Jerry                Liu                  Manager

dmSQL> SELECT * FROM tb_contact;
 NO           FIRST_NAME           LAST_NAME                PHONE
========== ==================== ==================== ====================
         1 Eddie                Chang                2145678
         2 Hook                 Hu                   2335678
         3 Jackie               Yu                   2346678
         4 Jerry                Liu                  2345671
```

➲ **导出到XML文件中：**

a) File_path是目标XML文件的完整路径。所生成的文件应该是存储在服务器上，因此，这个自变量所指定的路径应该是在服务器上的路径。本例中file_path自变量的值为：“/usr/john/xmlexport.xml”。

b) db_tag是用户为数据库自己定制的标签。NULL或空值字符串表示将使用默认值作为数据库在XML文件中的标签。本例中这个自变量的值为“EMPLOYEE”。

c) 本例中我们将使用的object_str串为：

```
'SELECT ID AS NO, FNAME AS FIRST NAME, LNAME AS LAST NAME, WORK
AS JOB FROM tb_card#TITLE;tb_contact#NUMBER'
```

d) 在option_flag自变量中，我们将使用“capitalize_tag_name;blob_in_separate_file”标记来作为这个自变量的值。

e) log_path自变量的值为：“/client/john/xmlexport.log”作为日志文件的存储路径。

f) 因此，CALL XMLExport语句的格式如下：

```
CALL XMLExport(
'/usr/john/xmlexport.xml',
'EMPLOYEE',
'SELECT ID AS NO, FNAME AS FIRST_NAME, LNAME AS LAST_NAME, WORK
AS JOB FROM tb card#TITLE;tb contact#NUMBER',
'capitalize_tag_name;blob_in_separate_file',
'/client/john/xmlexport.log');
```

g) 导出文件xmlexport.xml的部分内容如下：

```
<EMPLOYEE>
  <TITLE>
    <NO>1</NO>
    <FIRST_NAME>Eddie</FIRST_NAME>
    <LAST_NAME>Chang</LAST_NAME>
    <JOB>Manager</JOB>
  </TITLE>
  <TITLE>
    <NO>2</NO>
    <FIRST_NAME>Hook</FIRST_NAME>
    <LAST_NAME>Hu</LAST_NAME>
    <JOB>SoftwareEngineer</JOB>
  </TITLE>
  <TITLE>
```

```
  <NO>3</NO>
  <FIRST_NAME>Jackie</FIRST_NAME>
  <LAST_NAME>Yu</LAST_NAME>
  <JOB>SoftwareEngineer</JOB>
</TITLE>
<TITLE>
  <NO>4</NO>
  <FIRST NAME>Jerry</FIRST NAME>
  <LAST_NAME>Liu</LAST_NAME>
  <JOB>Manager</JOB>
</TITLE>
<NUMBER>
  <NO>1</NO>
  <FIRST_NAME>Eddie</FIRST_NAME>
  <LAST_NAME>Chang</LAST_NAME>
  <PHONE>2145678</PHONE>
</NUMBER>
<NUMBER>
  <NO>2</NO>
  <FIRST_NAME>Hook</FIRST_NAME>
  <LAST_NAME>Hu</LAST_NAME>
  <PHONE>2335678</PHONE>
</NUMBER>
<NUMBER>
```

```
    <NO>3</NO>
    <FIRST_NAME>Jackie</FIRST_NAME>
    <LAST_NAME>Yu</LAST_NAME>
    <PHONE>2346678</PHONE>
  </NUMBER>
  <NUMBER>
    <NO>4</NO>
    <FIRST NAME>Jerry</FIRST NAME>
    <LAST_NAME>Liu</LAST_NAME>
    <PHONE>2345671</PHONE>
  </NUMBER>
</EMPLOYEE>
```

➲ **我们还可以：**

a) 使用“**column_as_attribute**”选项，再调用XMLExport：

```
CALL XMLExport(
'/usr/john/xmlexport.xml',
'EMPLOYEE',
'SELECT ID AS NO, FNAME AS FIRST_NAME, LNAME AS LAST_NAME, WORK
AS JOB FROM tb_card#TITLE ',
'capitalize_tag_name;blob_in_separate_file;column_as_attribute
','/client/john/xmlexport.log');
```

b) XML文件的部分内容则变为：

```
<EMPLOYEE>
  <TITLE NO="1" FIRST_NAME="Eddie" LAST_NAME="Chang"
JOB="Manager" />
```

```
  <TITLE NO="2" FIRST_NAME="Hook"  LAST_NAME="Hu"
JOB="SoftwareEngineer" />

  <TITLE NO="3" FIRST_NAME="Jackie" LAST_NAME="Yu"
JOB="SoftwareEngineer" />

  <TITLE> NO="4" FIRST NAME="Jerry" LAST NAME="Liu"
JOB="Manager" />

</EMPLOYEE>
```

5.30 XMLIMPORT

系统存储过程XMLIMPORT为用户提供了一个可编程的界面，用以将XML文件中的数据导入到DBMaster数据库中。只有SYSADM、SYSDBA或DBA才可以调用该存储过程。此外，因为XMLIMPORT是系统存储过程，所以它的执行权不能授予其他用户。

XMLIMPORT可以将XML文件中的表导入到DBMaster表中。当从XML文件中导入数据时，用户不用分解文件（分析文件内容、将文件中的数据导入到表中），可以简单地将整个XML文件存储到数据库中。导入的XML文件必须是存在于服务器上的文件，导入XML文件过程中生成的日志文件将被存储到客户端的机器上。

如果您不想分解文件，而希望存储整个XML文件，那么就必须为存储到数据库中的这个XML文件指定一个“键”（key）。这样在查询数据库中的XML文件时就可以使用这个键值。

➲ **XMLIMPORT的原型：**

```
XMLIMPORT(VARCHAR(256) FILE_PATH,
          VARCHAR(16000) OBJECT_STR,
          VARCHAR(256) OPTION_STR,
          VARCHAR(256) LOG_PATH)
```

名称	数据类型	长度(字节)	描述	是否区分大小写
file_path	varchar	256	导入的XML文件的全路径	依操作系统而定
object_str	varchar	16000	被导入的对象的描述串	XML标签大小写敏感，而表名和字段名是否大小写敏感则取决与DBMaster的设置
option_flag	varchar	256	选项标记的描述串	不区分大小写
log_path	varchar	256	客户端的错误日志文件的全路径	依操作系统而定

表 5-4 XMLIMPORT自变量列表

构造XMLIMPORT自变量

首先，要导入到数据库中的XML文件必须是在服务器上生成的。file_path是一个完整路径字符串，它将作为一个自变量传入到相应的存储过程中。

第二，自变量object_str可用来描述导入对象。这些描述信息包括文档等级，用户自己定制的字段标签名和数据库表中字段名之间的映射，还包括用户自己定义的表标签名和数据库中表名称之间的映射。该自变量的格式如下：

```
object_str =:
       { <table_element> [; <table_element>]...}

<table_element> =
       { <document mapping information>#<table mapping information> }

<document mapping information> =:
       {<document level string>[(<column tag names>)]

<document level string> =: {/<level1> [/<level2>/.....]}

<column tag names> =: {<tag1> [, <tag2>]...}

<table mapping information> =: <table import definition>

<table import definition> =: { <insert sql statement> | <target table
name>[(<table column names>)] }

<insert sql statement> =: INSERT INTO <target table name> [(<table column
names>)] VALUES (<value list>)

<table column names> =: {<col1> [, <col2>] ...}

<value list> =: {<insert value>, <insert value>,...}

<insert value> =: {<constant> | <expression>}
```

图 5-1 自变量 object_str语法

如果您不希望分解文件，只将XML文件的内容存储到表中，那么您就应该在<column tag names>中进行特殊的操作，详细信息请参看例5。

<table_element>表示一张表。元素之间的分割符是分号（；）。
<document level string>表示从根等级到表等级的不同文档级别。

```
<root>
   <database>
      <table1>
          <column1>
          </column1>
          <column2>
          </column2>
      </table1>
      <table2>
      </table2>
   </database>
</root>
```

图 5-2 XML File示例

以图5.2中的xml示例文件为基础，导入<database>下的<table1>标签中存储的数据，<table1>由<document level string>指定：*“/root/database/table1”*。

您可以在<column tag names>中指定哪个字段标签将被插入到表中。如果没有指定<column tag names>，就代表插入某个表标签下的所有字段标签。

<table import definition>中，可以使用<INSERT SQL statement>的格式或TABLE_NAME [<table column names>]格式。使用<INSERT SQL statement>时，INSERT SQL语句的格式如下：

```
INSERT INTO <target table name> [(<table column names>)] VALUES (<value list>)
```

<table column names>用来指定要插入哪些字段。如果没有指定<table column names>，就意味着用户希望在目标表中插入所有字段（这里插入字段的语法和普通的INSERT SQL语句的语法相同）。如果<document mapping information>中包含了<column tag names>，那么

<column tag names> 中指定的字段标签数必须等于<value list>中的主变量数。如果<document mapping information>中没有包含<column tag names>，就表示基元素下的所有字段标签都将插入到目标表中。您也可以用DTD文件中的模式信息来检查字段标签数是否和<value list>中的主变量数相等。

<table column names>、<value list>以及<document mapping information>文件中的<column tag names>之间的映射必须是正确的。<column tag names>和<value list>文件中的主变量对应。<table column names>中的字段顺序，<value list>中的值顺序以及<column tag names>中的字段标签顺序一起决定了将在<value list>中插入什么值。

您可以使用<target table name>[(<table column names>)]来指定要在目标表（<target table name>）中插入哪个表。<target table name> 对应<document level string>中的最后一级标签。

如果使用上面的格式，那么就不能插入常量或表达式。如果<document mapping information>中没有指定<column tag names>，那么就不应该有<table column names>出现。如果<document mapping information>中包含了<column tag names>，那么<column tag names>中的字段标签数必须和<table column names>中的字段数相等。

您可以在<table column names>中指定要插入表中的字段。如果没有指定<table column names>，则插入表的所有字段。在这种情况下，<document mapping information>中就不应该再指定<column tag names>。DTD文件中的模式信息可以检查基元素下的所有标签数和目标表的所有字段的数是否相等。

用户应该负责<table column names>和<column tag names>之间的对应关系。<column tag names>中的标签应该对应于<table column names>相应位置的字段。

➲ **例1**

如果〈table column names〉是（c1, c2, c3），〈value list〉为（?,?,?），〈column tag names〉是（tg1, tg2, tg3），那么**tg1**中的值将被插入到字段**c1**中，而**tg2**中的值被插入到字段**c2**中，**tg3**中的值则被插入到字段**c3**中。

➲ **例2**

假设表**t1**有四个字段：**c1**、**c2**、**c3**以及**c4**，同时，在要导入的xml元素中，我们还有四个标签**tg1**、**tg2**、**tg3**、**tg4**。再假设自变量obj_str为“/root/book/order(tg1, tg2)#insert into t1 (c1, c2, c3) values (?,?+3, 5)”。从这个导入对象的描述串中，我们可以确定表**t1**是我们的目标表，插入到**t1**中的**c1**字段值是标签**tg1**的值，插入到字段**c2**中的值是标签**tg2**的值加3，插入到**c3**字段的值是常量5。

➲ **例3**

如果用户没有在<document mapping information>中指定<column tag names>，这表示xml字段标签的顺序和<table column names>中的字段顺序相同，基元素下的所有字段标签值都将被插入到目标表中。

下面假设我们的目标表**t2**中有5个字段：**c1**、**c2**、**c3**、**c4**以及**c5**。同时xml文件中的标签序列为：**tg1**、**tg2**、**tg3**、**tg4**。自变量obj_str的值为“/root/book/order#insert into t2 (c1, c2, c3, c4, c5) values (?, ?, ?, ?, 6)”，标签**tg1**的值将被插入到表**t2**的**c1**字段中，标签**tg2**的值将被插入到字段**c2**中，标签**tg3**的值将被插入到字段**c3**中，标签**tg4**的值将被插入到字段**c4**中，而常量**6**被插入到表**t2**的**c5**字段中。

如果obj_str is的值为“/root/book/order(tg1, tg2, tg3, tag4)#insert into t1 values (?, ?, ?, ?)”，那么它的意思是用户希望将4个标签插入到目标表的所有字段中。**tg1**的值被插入到表**t1**的**c1**字段中，**tg2**的值将被插入到字段**c2**中，**tg3**的值将被插入到字段**c3**中，**tg4**的值将被插入到字段**c4**中。

➲ **例4**

如果obj_str的值为“/root/book/order(tg1, tg2)#insert into t1 values (?, ?, acos(1))”，这意味着将**acos(1)**的值插入到表**t1**的**c3**字段中。

➲ **例5**

对于那些希望在记录中存储整个XML文件，而不希望分解XML文件并存储文件中的内容（例如：分析XML文件，只将XML文件中数据存储到表中）的用户而言，他们必须在<column tag name>中指定一个虚拟标签（virtual tag）。这个特殊的虚拟标签被命名为“_XML_FILE_”。

如果将“_XML_FILE_”当作一个字段标签名，那么这个特殊虚拟标签之前的字段标签所代表的字段就会被作为键值。此外，<value list>文件中对应的值只能是不需要额外定义的一个主变量。

如果导入的对象串为“/root/book/order(tag1, tag2, XML_FILE_)#insert into t2 (c1, c2, c3, c4, c5) values (?+2, ?*5, ?, 7, 8)”，那么整个XML文件将被插入到表t2的c3字段中。

如果导入的对象串为“/root/book/order(tag1, tag2, _XML_FILE_)#customer(firstname, lastname, xml_file)”，也就是说，对象串中的<table_element>为表“customer”，XML文件中的标签tag1的值将插入到“firstname”字段，此外XML文件中的tag2标签的值将被插入到“lastname”字段，而整个XML文件则将被插入到“xml_file”字段中。“firstname”和“lastname”将作为查找指定XML文件的键。

➲ **例6**

如果<column tag names> = <tag1, tag2, tag3>，<table column names> = <c1, c2, c3>，则这里有三对映射：tag1 <-> c1，tag2 <-> c2，tag3 <-> c3。标签名和字段名要么全有要么全无。因此像这样含有空标签名(tag1, ,tag3)的格式是不允许的，空字段名称同样也是不允许的。用户只能要么为所有字段定制标签，要么为所有字段都不定制标签。

因此导入对象串“/root/book/order(tag1, , tag2)#insert into t2 (c1, c2) values (?, ?, ?)”是不允许的。而这样的导入对象串“/root/book/order(tag1, tag2, tag3)#insert into t2 (c1, c2, c3, c4) values (?, ?, ?,)”则是允许的。至于在表t2的字段c4中插入什么值将取决于表的模式信息。

第三，选项标记串不区分大小写。设置了选项标记串后，“column_as_attribute”字段在导入XML文件中被看做元素的属性。如果没有设置选项标记串，字段在XML文件中将被作为一个元素。

```
Option flag=:{[<attribute>[;<attribute>]…]}
<attribute>=:
{
column_as_attribute
}
```

最后，在导入XML文件过程中生成的错误日志文件被存储到客户端机器的log_path目录下。

导入XML文件

假设我们在*/usr/john*目录下有一个名为xmlimport.xml的XML文件，文件内容如下：

```
<ROOT>
   <EMPLOYEE>
     <TITLE>
       <TAG1>1</TAG1>
       <TAG2>Eddie</TAG2>
       <TAG3>Chang</TAG3>
       <TAG4>Manager</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>2</TAG1>
       <TAG2>Hook</TAG2>
       <TAG3>Hu</TAG3>
       <TAG4>SoftwareEngineer</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>3</TAG1>
       <TAG2>Jackie</TAG2>
       <TAG3>Yu</TAG3>
       <TAG4>SoftwareEngineer</TAG4>
     </TITLE>
     <TITLE>
       <TAG1>4</TAG1>
       <TAG2>Jerry</TAG2>
       <TAG3>Liu</TAG3>
       <TAG4>Manager</TAG4>
     </TITLE>
     <NUMBER>
       <NO>1</NO>
       <FIRST_NAME>Eddie</FIRST_NAME>
       <LAST_NAME>Chang</LAST_NAME>
       <PHONE>2145678</PHONE>
     </NUMBER>
```

```
    <NUMBER>
      <NO>2</NO>
      <FIRST NAME>Hook</FIRST NAME>
      <LAST NAME>Hu</LAST NAME>
      <PHONE>2335678</PHONE>
    </NUMBER>
    <NUMBER>
      <NO>3</NO>
      <FIRST_NAME>Jackie</FIRST_NAME>
      <LAST_NAME>Yu</LAST_NAME>
      <PHONE>2346678</PHONE>
    </NUMBER>
    <NUMBER>
      <NO>4</NO>
      <FIRST_NAME>Jerry</FIRST_NAME>
      <LAST_NAME>Liu</LAST_NAME>
      <PHONE>2345671</PHONE>
    </NUMBER>
  </EMPLOYEE>
<ROOT>
```

下面我们将会把记录在**xmlimport.xml**文件中的数据导入到数据库中，该数据库的模式如下：

```
Database Name: DB1
Table Name: CARD(C1 CHAR(30), C2 CHAR(30), C3 CHAR(30), C4 CHAR(30))
Table Name: CONTACT(NO CHAR(30), FIRST_NAME CHAR(30), LAST_NAME CHAR(30), PHONE
CHAR(30))
```

从上面的xml文件中，我们就可以看到<EMPLOYEE>元素下有两个子元素。我们把<EMPLOYEE>元素映射为数据库，<TITLE>和<NUMBER>映射为目标数据库的表。

假设我们要将<TITLE>和<NUMBER>分别导入到表**TB_CARD**和**TB_CONTACT**中，XML文件中标签和数据库中表间的映射如下：

```
/ROOT/EMPLOYEE/TITLE -> /DB TEST/tb card
/ROOT/EMPLOYEE/NUMBER -> /DB_TEST/tb_contact
```

XML文件中的标签和表字段间的映射如下：

*/ROOT/EMPLOYEE/TITLE*下的元素（**<TITLE>**和表**TB_CARD**之间的映射）：

```
TAG1 -> NO
TAG2 -> FIRST NAME
TAG3 -> LAST NAME
TAG4 -> JOB
```

*/ROOT/EMPLOYEE/NUMBER*下的元素（**<NUMBER>**和表**TB_CONTACT**之间的映射）：

```
NO -> NO
FIRST_NAME -> FIRST_NAME
LAST_NAME -> LAST_NAME
PHONE -> PHONE
```

此外，我们可以看到**xmlimport.xml**文件中的字段是被作为XML文件的元素的。最后假设我们的日志文件为*/client/john/xmlimport.log*。

要将数据导入表**TB_CARD**，就应该导入*/ROOT/EMPLOYEE/TITLE*下的元素。标签**TAG1**映射为字段**ID**，标签**TAG2**映射为字段**FNAME**，标签**TAG3**映射为字段**LNAME**，标签**TAG4**映射为字段**WORK**。

要将数据导入到表**TB_CONTACT**中，就应该导入*/ROOT/EMPLOYEE/NUMBER*下的元素。根据假设，<NUMBER>标签下的所有元素和**TB_CONTACT**之间是直接映射的，它们将被导入到**TB_CONTACT**表的相应字段中。

注意，XML标签是大小写敏感的。因此，例子中的**ROOT**、**EMPLOYEE**、**TITLE**、**TAG1**、**TAG2**和**TAG3**都必须大写。至于表名称，字段名称是否区分大小写，这取决于DBMaster的设置。

➲ **根据以上文件执行XMLIMPORT：**

a) 要导入的XML文件必须在服务器上，因此，自变量中指定的完整路径也必须是服务器上的路径。本例中，自变量file_path的值为“/usr/john/xmlimport.xml”。

b) 自变量**object_str**的值如下：

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#INSERT INTO
TB_CARD (ID,FNAME,LNAME,WORK) VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#tb_contact'
```

或者

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB_CARD (ID,FNAME,LNAME,WORK) VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#tb_contact'
```

或者

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#CARD
(C1,C2,C3,C4);/ROOT/EMPLOYEE/NUMBER#contact'
```

c) 这个导入的对象串可以有多种格式：

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#INSERT INTO
TB_CARD (ID,FNAME,LNAME,WORK) VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#TB_CONTACT'
```

因为四个标签对应四个字段，并且标签顺序和字段顺序相同，那么导入的对象串格式还可以这样：

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB CARD (ID,FNAME,LNAME,WORK)
VALUES  (?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#TB_CONTACT'
```

或者

```
'/ROOT/EMPLOYEE/TITLE#INSERT INTO TB_CARD VALUES
(?, ?, ?, ?);/ROOT/EMPLOYEE/NUMBER#TB_CONTACT'
```

因为主变量中不要求做计算，所以导入串的格式也可以是这样：

```
'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3, TAG4)#TB_CARD(ID, FNAME,
LNAME, WORK);/ROOT/EMPLOYEE/NUMBER#TB_CONTACT'
```

d) 由于本例中的字段被作为XML文件元素，因此我们在这里不设置选项标记。如果字段不作为XML元素，我们就可以设置选项标记。

```
option_flag =: {[<attribute> [;<attribute>]...]}

<attribute> =:

{

column_as_attribute

}
```

e) 日志文件存储为：“/client/john/xmlimport.log”。该日志文件中记录了处理存储过程XMLIMPORT时发生的错误信息。

f) 我们现在采用以上一种**obj_str**格式来调用XMLIMPORT：

```
CALL XMLImport(

'/usr/john/xmlimport.xml',

'/ROOT/EMPLOYEE/TITLE(TAG1, TAG2, TAG3,
TAG4)#TB_CARD(ID,FNAME,LNAME,WORK);/ROOT/EMPLOYEE/NUMBER#tb_cont
'',

'/client/john/xmlimport.log');
```

6 dmSQL命令

本章介绍的命令需要在DBMaster中的dmSQL命令行工具中执行。

6.1 CONNECT

CONNECT命令用于创建一个数据库连接。用户名和密码区分大小写，而数据库名不区分大小写。任何具备CONNECT或更高权限的用户都可以执行该命令。

在连接数据库之前，**dmconfig.ini**文件中必须包含目标数据库的配置信息。如果数据库是在本地计算机中创建的，那么这个数据库的配置信息应该已经存在。如果数据库是创建在远程计算机中，您应该根据需要在本地**dmconfig.ini**中添加这个数据库的一些配置信息段。

CONNECT命令可以连接一个单机数据库，此时它的具体作用是启动数据库并创建连接。对于单机数据库，一次只能被一个用户连接。

连接单机数据库之前，需要指定数据库所在目录。您应该在**dmconfig.ini**文件中使用**DB_DbDir**关键字来指定包含这个数据库的目录。

当数据库服务器运行时，您还可以使用CONNECT命令来连接一个客户端/服务器模式的数据库。如果数据库服务器没有运行，那么在连接数据库之前就必须先启动服务器。

连接客户端/服务器数据库之前，需要指定作为DBMaster服务器的主机IP地址及端口号。您应该在**dmconfig.ini**文件中使用**DB_SvAdr**和**DB_PtNum**关键字来指定IP地址和端口号。当然，您还可以用主机名代替IP地址来作为关键字**DB_SvAdr**的值。

DBMaster总是尝试连接客户/服务器型数据库，直到连接超时。连接超时的时间可以由**dmconfig.ini**文件中的**DB_CTimO**关键字来设置。单机数据库中不提供此关键字。

用户名和密码是必须的，密码是NULL时可以忽略它。您还可以在**dmconfig.ini**文件中使用**DB_Usrld**和**DB_PasWd**关键字，这样，您在执行CONNECT命令时就可以省略用户名和密码。关键字**DB_Usrld**设置连接数据库的默认用户名，而**DB_PasWd**设置连接的默认用户密码。因为DBMaster一般是在同一个地方获取用户名和密码的，所以您不能在CONNECT命令中只指定用户名和密码中的一个参数，而将另一个参数

放在配置文件中指定。如果您在CONNECT命令和配置文件中指定了用户名和密码，DBMaster将忽略配置文件中**DB_Usrld**和**DB_PasWd**关键字所指定的用户名和密码。

database_name.要连接的数据库名。

user_name.....连接数据库的用户名。

password.....用户*user_name*当前的密码。

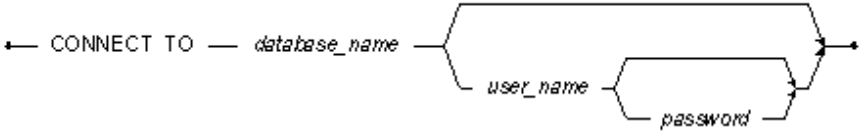


图 6-1 CONNECT语法图

➲ **值1**

dmconfig.ini文件的**Tutor1**配置段中包含关键字**DB_DbDir**，值如下：

```
[TUTOR1]
DB_DbDir = C:\DBMASTER\DATABASE\TUTOR1
```

➲ **例1**

下例是连接一个单用户数据库**Tutor1**，连接用户为**jenny**，密码是**grala833**。

```
dmSQL> CONNECT TO Tutor1 jenny grala833;
```

➲ **值2a**

dmconfig.ini文件的**Tutor2**配置段中包含关键字**DB_SvAdr**和**DB_PtNum**，值如下：

```
[TUTOR2]
DB_SvAdr = 192.72.116.137
DB_PtNum = 35400
```

➲ **值2b**

下面将用主机名来代替IP地址作为关键字**DB_SvAdr**的值。

```
[TUTOR2]
```

```
DB_SvAdr = mars.syscom.com.tw
DB_PtNum = 35400
```

➲ **例2**

下例是连接一个多用户数据库**Tutor2**，连接用户是**amanda**，密码是**grixa944**。

```
dmSQL> CONNECT TO Tutor2 amanda grixa944;
```

➲ **值3**

dmconfig.ini文件的**Tutor2**配置段中包含关键字**DB_SvAdr**、**DB_PtNum**、**DB_UsrId**以及**DB_PasWd**，其值如下：

```
[TUTOR2]
DB_SvAdr = 192.72.116.137
DB_PtNum = 35400
DB_UsrId = vivian
DB_PasWd = shuka828
```

同样的，这里也可以和值**2b**一样将**DB_SvAdr**中的IP地址替换为主机名。

➲ **例3**

下例连接了一个多用户数据库**Tutor2**，连接用户是**vivian**，其密码是**shuka828**。CONNECT命令中没有提供用户名和密码，它们是由**dmconfig.ini**中的关键字**DB_UsrId**和**DB_PasWd**指定的。如果您在CONNECT命令中指定了用户名和密码，那么DBMaster就会忽略由关键字**DB_UsrId**和**DB_PasWd**指定的用户名和密码。

```
dmSQL> CONNECT TO Tutor2;
```

6.2 CREATE DATABASE

CREATE DATABASE命令用来生成一个新的数据库。要执行这条命令，操作系统必须要给予DBMaster在生成数据库目录中的写权限。任何用户都可以执行这条命令。

DBMaster在**dmconfig.ini**文件中存储每个数据库的所有配置信息。您在计算机上可以连接的数据库在这个文件中都有其相应的数据库配置段。**dmconfig.ini**文件是一个ASCII文本文件，可以用文本编辑器来编辑。

每个数据库的配置段是由段名以及该段名后的一个或多个关键字行组成。这个段名是一对方括号括起来的数据库名。段名后的关键字行包括关键字以及相应的值。如果一个关键字要求或支持多个值，您可以使用空格或逗号来分隔每个值。在启动或连接数据库时，不同关键字代表的参数将被DBMaster系统使用。

dmconfig.ini文件中的关键字不区分大小写。关键字的值可能是大小写敏感的，这取决于这个关键字以及数据库运行时所处的操作系统。新建一个数据库时，DBMaster将检查**dmconfig.ini**文件中的数据库配置段。如果文件中存在和您所创建的数据库同名的配置段，那么生成这个数据库时，DBMaster将使用这个同名配置段中的配置信息。如果不存在一个和新数据库同名的配置段，那么在生成这个数据库时，DBMaster将在文件中新增一个配置段，其中关键字的值采用默认值。

创建数据库时，您应该选择一个不同于其它所有可能连接计算机的已有数据库名来作为您的数据库名称。这是因为DBMaster把所有本地和远程数据库的配置信息都存储在一个**dmconfig.ini**文件中，如果两个数据库采用同一个名字将会引起冲突。一旦数据库被创建，您就不能再修改数据库的名称了，除非您载出数据库中的所有数据，然后用新名称重建数据库。数据库的名称长度最大不能超过128个字符，可以包含字母、数字以及下划线，不区分大小写。

在DBMaster的物理存储模式中，文件是物理存储单元，其中包含了数据库中的数据。文件由操作系统管理，而文件中的数据则由DBMS管理。DBMaster有三种类型的文件：数据文件、BLOB文件和日志文件。

数据文件和BLOB文件存储用户和系统数据。尽管这两种文件具有相似的特性，但DBMaster对它们的管理方式却是不同的，这有助于提高执行效率。数据文件存储表和索引的数据，而BLOB文件则存储二进制大型对象。

日志文件是一种特殊的文件，它是对数据库所做更改的实时或历史记录，并且更改后的数据库状态都会存储在这个文件中。使用日志文件可以使数据库撤销失败事务对数据库所做的更改或在数据库发生灾难恢复时，将已提交的事务重新写入数据库。日志文件只能由数据库管理系统（DBMS）使用，它并不能存储用户数据。

在DBMaster的逻辑存储模式中，表空间是数据的逻辑存储结构，用以将数据分割为DBMS可以管理的区域。每个表空间可能包含几张表和索引。表空间中的数据存储在物理文件中，但是由数据库管理系统（DBMS）来管理。表空间分为五种：固定（regular）表空间、自动扩展（autoextend）表空间、只读（read only）表空间、读写（read/write）表空间和系统（system）表空间。

固定表空间（regular tablespaces）有固定的大小，可以包含一个或多个数据文件或BLOB文件。您可以扩大表空间中现有的文件大小或向表空间中增加新文件来手动扩展表空间。一个固定表空间最多可包含32,767个文件，每个表空间中所有文件所占的空间总和不能超过8TB。在UNIX中，固定表空间可以存放在裸设备上。

注意 *有关裸设备（raw devices）的信息，请查阅有关Unix系统文档。*

自动扩展表空间（autoextend tablespaces）可根据需要来自动扩大表空间以存储增加的数据，表空间中所有数据所占空间的总和不能超过8TB。自动扩展表空间中必须至少包含一个数据文件，也可以存储BLOB文件。如果想要在自动扩展表空间中增加新文件，首先应该将这个表空间转换成固定表空间。如果创建表空间时没有BLOB文件，只有一个数据文件，BLOB文件可以在稍后再添加。但要注意的是自动扩展表空间不能用在裸设备上。

只读表空间不允许用户在表空间中执行任何修改的操作，然而只读表空间自有它的优点：

- 不必经常备份。只读表空间在它设为只读之后仅需要做一次备份即可。
- 数据库的恢复变很更容易。当发生故障时，只读表空间的优点是不需要从介质故障中恢复。
- 只读表空间比可更改的表空间占有更少的系统资源。（没有锁资源）

读写表空间允许用户在表空间中执行任何有权限的修改操作。

创建数据库时，DBMaster会自动生成系统表空间。每个数据库只有一个系统表空间，用来存储系统表，而系统表是用来记录存储模式、安全和状态信息的。如果不是在Unix裸设备（*Unix raw device*）上创建的系统表空间，那么系统表空间就是一个自动扩展表空间。系统表空间也可以变更为固定表空间。系统表空间中数据文件的初始大小为600KB，而BLOB文件的初始大小是20KB。

DBMaster将在系统表空间中创建一个系统数据文件和一个系统BLOB文件，并且还在默认用户表空间中新建一个用户数据文件以及用户BLOB文件。除了这些文件，DBMaster还至少创建一个系统日志文件，用来记录数据库事务的日志。

系统文件的默认名是DATABASE.SDB、DATABASE.SBB以及DATABASE.JNL，其中DATABASE是数据库的名字。如果要改变这些默认名，您应该在**dmconfig.ini**文件中为**DB_DbFil**、**DB_BbFil**以及**DB_JnFil**这些关键字设置新值。**DB_DbFil**表示系统数据文件的名称，**DB_BbFil**表示系统BLOB文件的名称，**DB_JnFil**表示系统日志文件的名称。您应该在创建数据库之前就设置好这些值，否则将使用默认名称，并且在数据库创建后，系统文件的名称是不能改变的。

默认的用户文件名是DATABASE.DB和DATABASE.BB，其中DATABASE是数据库的名称。要改变这些默认名称，您应该在**dmconfig.ini**文件中为关键字**DB_UsrDb**和**DB_UsrBb**设置新值。或用**DB_UsrDb**来指定默认用户数据文件的名称和大小，用**DB_UsrBb**来指定默认用户BLOB文件的名称和大小。您可以使用这两个关键字来为默认用户文件设置新的名称以及数据页或BLOB帧的大小，文件名和文件大小

之间用空格或逗号分隔。如果您不希望默认用户文件采用默认名称，您就必须在数据库创建之前设置好这些值。

DBMaster中用来记录数据库事务的日志文件最多可达到8个。如果要创建多个日志文件，您应该在**DB_JnFil**关键字后面增加更多的文件名，这些文件名可用空格或逗号来分隔。DBMaster在创建数据库时自动创建日志文件。数据库创建好后，您可以增加日志文件名，然后在新日志模式下重启数据库，这样就可以为数据库再添加额外的日志文件了。

如果要在文件名中包括文件路径，那么在Windows系统下路径应该包括驱动器名和完整路径。在Unix系统下，可以是完整路径，也可以是相对路径。在默认情况下，文件将被创建在**dmconfig.ini**文件中关键字**DB_DbDir**所指定的目录下，如果没有指定**DB_DbDir**的值，文件就将被创建在应用程序目录下。DBMaster的系统文件名称最长不能超过256个字符，可以包含除空格外，任何操作系统允许的字符和符号。

系统数据文件的默认大小是600KB，系统BLOB文件的默认大小是20KB，日志文件的默认大小是4,000KB。要改变这些默认的文件大小，您应该在**dmconfig.ini**文件中为关键字**DB_BfrSz**和**DB_JnlSz**设置新值。

关键字**DB_BfrSz**用来指定系统BLOB文件中帧的大小，修改该值实际上就是修改系统BLOB文件的大小。如果您不想使用默认大小，那么您就应该在创建数据库时就设置**DB_BfrSz**的值，创建数据库之后是不能更改这个值的。

关键字**DB_JnlSz**用来指定日志块中系统日志文件的大小，日志块是存储日志文件的基本单元。日志块存储的是事务在数据库上的操作记录。每个日志块的大小与数据页的大小一致（数据页的大小由关键字**DB_PgSiz**设定）。只要不超过块大小，每个日志块可以存储任意多的事务信息。您可以设置关键字**DB_JnlSz**的值来指定系统日志文件的大小，**DB_JnlSz**的取值范围是23-524287块。将**DB_JnlSz**的值乘以数据页的大小就可以知道这个文件实际占多少KB的空间。如果您的数据库中有多个日志文件，那么每个日志文件的大小都是**DB_JnlSz**。关键字**DB_JnlSz**的默认值是***1,000*** pages。关键字**DB_JnlSz**的值随时都可以更改，但是只有您在新日志模式下重启数据库后，这个更改才能真正生效。

对于默认用户文件，用户数据文件的默认大小是***600***KB，用户BLOB文件的默认大小是***20***KB。要改变这些默认的文件大小，您应该在**dmconfig.ini**文件中使用关键字**DB_UsrDb**和**DB_UsrBb**来设置新值。

关键字**DB_UsrDb**可用来设置默认用户文件中所包含的数据页数，数据页是存储的基本单位。数据页用来存储表的记录、索引键以及所有数据页能容纳的BLOB数据。只要没有超过页的大小，每个数据页都可以存储任意多的表记录或索引键。**dmconfig.ini**文件中的关键字**DB_PgSiz**指定每个数据页的大小。您可以设置关键字**DB_UsrDb**的值来指定默认用户数据文件的大小，**DB_UsrDb**的取值范围是2-524287个数据页。将这个值乘以关键字**DB_PgSiz**的值就可以知道该文件实际占用多少KB的存储空间。**DB_UsrDb**的默认值是***150***。

关键字**DB_UsrBb**可用来设置默认用户BLOB文件中所含帧的数目，帧是BLOB文件存储的基本单位。BLOB帧存储的是不适合存储在数据页中的二进制大对象，如图像、声频以及视频。每个BLOB帧只能存储一个BLOB，BLOB帧的大小由关键字**DB_BfrSz**来指定，取值范围在8KB到256KB之间。您可以设置关键字**DB_UsrBb**的值来指定默认用户BLOB文件的大小，**DB_UsrBb**的取值范围是2~524,287个帧。将这个值乘以**DB_BfrSz**的值就可以知道这个文件实际占多少KB的存储空间。**DB_UsrBb**的默认值是***2***。

安全模式决定DBMaster是否会用安全权限来控制用户对数据库的存取。DBMaster中有五个级别的安全权限：CONNECT、RESOURCE、DBA、SYSDBA以及SYSADM。

CONNECT权限允许用户连接数据库、查看系统表，访问被所有者、DBA或SYSADM授权的数据库对象。只拥有CONNECT权限的用户是不能新建数据库对象的。在给用户授予其它权限之前，必须先让这个用户获得CONNECT权。

RESOURCE权限允许用户创建和删除表、索引、视图、同义字以及定义域。用户只能删除他们自己创建的表、视图、同义字和定义域。此外，用户可以给其他用户授予或取消自己创建的数据库权限。拥有RESOURCE安全权限的用户同时拥有CONNECT的所有权限。

DBA权限允许用户启动、终止以及备份数据库，也可以管理数据库的资源、表空间和文件，还可以不经授权访问所有表、视图、同义字和定义域。DBA还可以将任何用户创建的数据库对象的访问权限授予其他用户，或者修改、取消这个授权。不过DBA不能授予安全权限，也不能新建组，但是可以在已有组中增加或删除用户。具备DBA安全权限的用户同时拥有RESOURCE和CONNECT的所有权限。

SYSDBA权限允许用户授予或取消非SYSADM及SYSDBA用户的安全权限。拥有该权限的用户还可以创建或删除组、在组中增加或删除用户，并且可以更改非SYSADM及SYSDBA用户的密码。拥有SYSDBA安全权限的用户同时拥有DBA、RESOURCE、CONNECT的所有权限。

SYSADM权限允许用户将安全权限授予所有用户或取消这个授权。拥有该权限的用户还可以创建或删除组、在组中增加或删除用户。SYSADM还可以修改任何用户的密码。每个数据库中只能有一个用户具备SYSADM权限。创建数据库时，DBMaster自动创建这个用户，并为这个用户取名为SYSADM。SYSADM不能将SYSADM安全权限授予其他用户。SYSADM同时拥有SYSDBA、DBA、RESOURCE以及CONNECT的所有权限。

新建数据库之前，您就应该设置安全模式。数据库创建以后，除非您载出数据库中的数据，然后重建数据库，否则，您将不能修改数据库的安全模式。用**dmconfig.ini**文件中的**DB_Secur**关键字来设置安全模式。如果创建数据库时没有设置关键字**DB_Secur**，系统将默认安全模式处于***开启***状态。

当安全模式开启，只有拥有适当安全权限的用户才可以连接数据库。连接数据库时需要输入用户名和密码。DBMaster维护授权用户和他们在数据库中的权限列表，并且还将检查这个列表以决定每个用户可以执行的命令。

当安全模式关闭，任何用户都可以使用任何用户名来连接数据库。并且连接时不需要密码，DBMaster将忽略密码。DBMaster不用维护数据库的用户或安全权限列表，任何用户都可以执行任何命令。

执行CREATE DATABASE命令时，DBMaster先新建一个数据库，然后再重启该数据库，并以SYSADM来连接数据库。创建SYSADM用户时，

DBMaster并不为他设置密码，因此您应该在创建数据库后立即更改SYSADM的密码以防非授权用户对数据库的访问。DBMaster在单用户模式下启动新建的数据库，这可以阻止其他用户在您更改SYSADM密码之前登录数据库。关闭数据库后，在单用户或多用户模式下重启数据库，您修改的密码就可以生效了，并且其他用户也可以连接这个数据库了。

DBMaster默认是单用户模式下启动数据库。如果您希望在多用户模式下启动数据库，您可以在客户端的**dmconfig.ini**文件中设置**DB_SvAdr**和**DB_PtNum**关键字，而在服务器端的**dmconfig.ini**文件中设置**DB_PtNum**关键字。

关键字**DB_SvAdr**用来指定DBMaster服务器的IP地址或服务器主机名。这个关键字只需要在客户端使用，在服务器端是可选的。要设置服务器的IP地址或主机名，**DB_SvAdr**必须是有效的IP地址或主机名。使用主机名时还应该保证您的计算机正确设置了域名服务（DNS）。

关键字**DB_PtNum**用来指定DBMaster服务器绑定的端口号。这个关键字在服务器端和客户端都是需要的。**DB_PtNum**关键字在1025 - 65535之间取值。如果没有指定端口号，DBMaster将采用默认端口号***23000***。

database_name.新建数据库的名字。

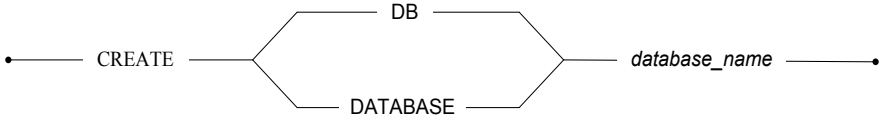


图 6-2 CREATE DATABASE语法图

➲ **例1**

下例中将新建一个名为**Accounts**的数据库，所有参数都采用默认设置。执行这条命令时，这个数据库在**dmconfig.ini**文件中的配置段是不存在的。这条命令在应用程序目录下创建了一个单用户数据库，数据库中使用默认的文件名**ACCOUNTS.SDB**、**ACCOUNTS.SBB**、**ACCOUNTS.DB**、**ACCOUNTS.BB**以及**ACCOUNTS.JNL**，其中**.SDB**的默认大小是800KB，**.DB**文件的默认大小是600KB，**.SBB**和**.BB**文件的默认大小是20KB，**.JNL**文件的默认大小是4000KB。若要在多用户模

式下启动数据库，您应该在数据库创建后，在**dmconfig.ini**文件的Accounts数据库配置段中添加**DB_SvAdr**和**DB_PtNum**这两个关键字。

```
dmSQL> CREATE DATABASE Accounts;
```

➲ **例2**

下例新建一个名为**Accounts**的数据库，数据库在**dmconfig.ini**文件中的设置见下面摘录：

```
dmSQL> CREATE DATABASE Accounts;
```

➲ **摘录**

该命令执行后，DBMaster就在**dmconfig.ini**中创建了这个数据库的配置段。这里命令在*C:\DATABASE\ACCOUNTS*目录下创建了一个单用户数据库，该数据库的安全模式处于开启状态。系统数据文件的文件名为**ACCOUNTS.SDB**，系统BLOB文件的文件名为**ACCOUNTS.SBB**，默认用户数据文件的文件名为**ACNTDATA.DB**，默认用户BLOB文件的文件名为**ACNTBLOB.BB**，三个日志文件的文件名分别为**ACNTHIST.JN1**，**ACNTHIST.JN2**以及**ACNTHIST.JN3**。系统数据文件的大小为800KB，系统BLOB文件的大小为20KB，默认用户数据文件的大小为1000KB，默认用户BLOB文件的大小为8000KB，每个日志文件的大小为2000KB。若要在多用户模式下启动数据库，您应该在数据库被创建之后，在**dmconfig.ini**文件的Accounts数据库配置段添加**DB_SvAdr**和**DB_PtNum**这两个关键字。

```
[ACCOUNTS]
DB_DbDir = C:\DATABASE\ACCOUNTS
DB_DbFil = ACCOUNTS.SDB
DB_BbFil = ACCOUNTS.SBB
DB_UsrDb = ACNTDATA.DB 250
DB_UsrBb = ACNTBLOB.BB 250
DB_BfrSz = 32
DB_JnFil = ACNTHIST.JN1, ACNTHIST.JN2, ACNTHIST.JN3
DB_JnlSz = 500
```

6.3 DEF TABLE

dmSQL的DEF TABLE命令用来显示指定表的结构信息，该命令不能用于系统表。

图 6-3 DEF TABLE语法图

➲ **例1a**

新建一张表：

```
dmSQL> CREATE TABLE tb_tmp(c00_serial SERIAL, c01_int INTEGER, c02_char
CHAR(20));
```

➲ **例1b**

执行这条命令：

```
dmSQL> DEF TABLE tb_tmp;
```

➲ **执行结果：**

```
dmSQL> DEF TABLE tb_tmp;
dmSQL> create table SYSADM.TB_TMP (
 C00 SERIAL  SERIAL(1),
 C01 INT  INTEGER default null ,
 C02 CHAR  CHAR(20) default null )
 in DEFTABLESPACE  lock mode row  fillfactor 100;
```

6.4 DEF VIEW

dmSQL的DEF VIEW命令可用来显示视图的定义，该命令一般不用于系统视图。

图 6-4 DEF VIEW语法图

➲ **例1a**

新建一个视图：

```
dmSQL> CREATE VIEW view_tmp AS SELECT c00_serial, c01_int FROM tb_tmp;
```

➲ **例1b**

执行这条命令：

```
dmSQL> DEF VIEW view_tmp;
```

➲ **执行结果：**

```
dmSQL> DEF VIEW view_tmp;
dmSQL> create view SYSADM.VIEW_TMP as select c00_serial, c01_int from
SYSADM.TB_TMP ;
```

6.5 DISCONNECT

DISCONNECT命令用于断开与数据库的连接。任何具备CONNECT或更高权限的用户都可以执行该命令。

AUTOCOMMIT模式用来控制DBMaster提交事务的时间。如果打开AUTOCOMMIT模式，则每条命令分别被视为一个事务。当事务成功完成后，DBMaster将自动提交事务。如果操作过程中出现错误，系统将自动回滚事务。如果关闭AUTOCOMMIT模式，则所有连续的COMMIT WORK命令所执行的命令被视为同一个事务。

执行COMMIT WORK命令来提交事务对数据库所做的更改，ROLLBACK WORK命令可用来回滚所有对数据库的更改。断开数据库连接时，如果AUTOCOMMIT模式处于关闭状态，那么正在执行的事务将被中断，该事务对数据库所作的任何修改都不会被记录到数据库中。

与多用户数据库断开连接后，数据库仍将处于活动状态，其他用户仍然可以访问该数据库。与运行于UNIX系统上的单用户数据库断开连接时，该数据库将被关闭，而与运行于Windows系统上的多连接数据库断开连接时，只有当您是这该数据库的最后一个连接者时，数据库才会被关闭。

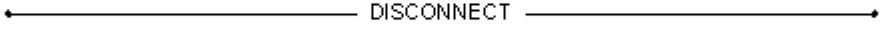


图 6-5 DISCONNECT语法图

➲ **例**

下例将断开与当前数据库的连接。

```
dmSQL> DISCONNECT;
```

6.6 EXPORT

Export命令的作用是从数据库的表中抽取数据，然后再把这些数据插入到文本文件中。命令中要使用2个结构：导出命令界面（export command interface）和描述文件（description file）。前者用来设置命令选项，后者用来设置导出文件的格式。

导出命令界面

Export命令的语法如下：

<data_file> ..... 这是您插入数据的目标文件，该参数是一个完整路径。如果您没有指定这个参数，那么导出文件的名称将是*<table_name>*_out.txt，其中*table_name*为表名称。

TABLE ..... 您可以在这里指定您要导出数据的表。

[DESCRIPTION <description_file>] 这是导出文件中数据格式的描述文件。在这个描述文件中，用户可以为导出文件设置一些规则。更多相关信息请参考*描述文件*部分。如果没有指定描述文件，那么描述文件的名称为*<table_name>_out.dsc*，其中*table_name*为表名称。如果描述文件不存在，那么DBMaster将使用默认的数据输出格式。

默认的文件格式将是一种可变的格式，解释如下：

- TAB符是字段分隔符。
- 换行字符（\n）是行结束符。
- 不存在引号。
- 源表中的所有字段按照它们在表中的顺序导出。

*[LOG <log_file>]*该文件记录了载出数据过程中出现的错误。如果没有设置该选项，系统将使用默认的日志文件名*export.log*来作为日志文件名。

[STOP_ON_ERROR] 该参数表示在出现错误时，停止对数据的载出。如果没有指定该选项，那么在发生错误的情况下，数据的载出仍将继续进行。

```
EXPORT
[INTO <data file>]
TABLE [<owner name>.]<table name>
[DESCRIPTION <description file>]
[LOG <log_file>]
[STOP_ON_ERROR]
```

描述文件

您可以指定描述文件的格式来格式化载出的结果。描述文件的格式有两种：固定格式和可变格式。

固定格式的描述文件

当用户希望垂直排列导出结果的字段时，就可以使用固定格式的描述文件。其中，用于排列的分隔符为空格字符。

FORMAT = FIXED　　该命令表示定长数据文件的描述文件格式。

[LOB_FORMAT= INTERNAL | EXTERNAL]　该选项表示当导出大型对象类型（如blob、clob、nclob、nblob）的字段时，将生成外部文件。使用此命令会为每条记录中的每个大型对象字段生成一个外部文件。如果没有指定该选项，大型对象字段的内容将被包含在数据文件中。

为外部文件取名时，一定要注意以下几点：
blobtempdir<m>\blbtmpf<n>.<tmp | txt>

m 的作用是在本目录下，为新建的目录从1开始指定未使用的最小数字。

例如，如果已存在名为blobtempdir1、blobtempdir2和blobtempdir3的目录，那么新建的用于包含外部文件的目录名为blobtempdir4。

n 的作用是在本目录下，为新建的文件从1开始指定未使用的最小数字。

文件的扩展名是**tmp**还是**txt**是由导出字段的类型（BLOB、FILE、CLOB）决定的。如果字段的类型是BLOB或FILE，那么文件的扩展名为**tmp**，否则，文件的扩展名为**txt**。

server_column_name　该参数列出了将要从数据库导出的源表字段名。如果字段名中包含空格，请用双引号将字段名括住。

column_position. 设置字段在数据文件中的位置（单位为字节）。

*server_columnname*和*column_position*中的多个参数值可由空格分隔。*column_position* 的值由两个数字组成，它们之间用冒号（:）分隔。例如，1:40的意思是用户可以在数据文件的第1个字节到第40个字节之间查找数据。您应该使用空格来垂直排列数据字段。如果源表中的数据超过了字段长度，那么超出的部分将被删除。

```
FORMAT=FIXED
[LOB_FORMAT=INTERNAL | EXTERNAL]
<server_column_name> <column_position>
```

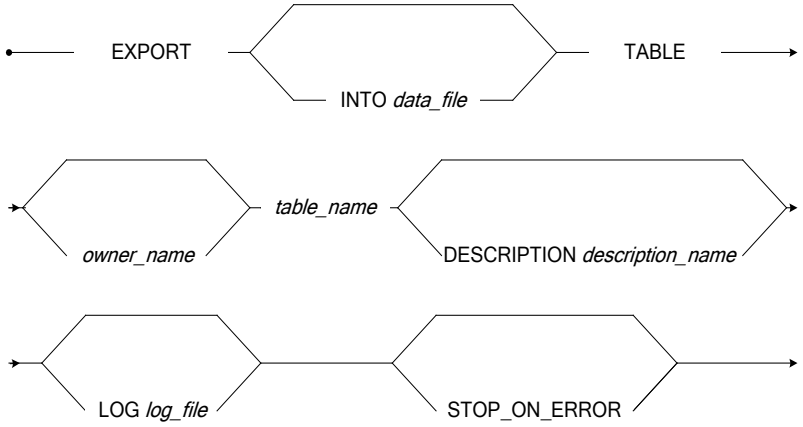


图 6-6 EXPORT语法图

可变格式的描述文件

选中可变格式的描述文件后，用户就可以使用自己指定的分隔符来分隔结果数据中的字段。

FORMAT=VARIABLE 该命令可将导出文件的格式设置成可变格式。

[COLUMN_DELIMITER=<delimiter>] 这条命令的作用是为数据文件中的每个导出字段设置分隔符，该字符应该用单引号括起来。例如，如果将**SPACE** 设置为字段分隔符，您就应该在*<delimiter>*中使用‘ ’。除了一般的字符外，您还可以使用下面转义字符串代表的特殊字符。

CHARACTER	ESCAPE SEQUENCE REPRESENTATION
TAB	\t
NEW LINE	\n

表 6-1 字符和转义序列

例如，如果TAB是分隔符，那么在*<delimiter>*中您应该使用‘\t’。如果没有指定字段分隔符，我们将使用TAB（\t）来作为字段分隔符。选择分隔符时您应该考虑周到。

如果字段分隔符的数量少于目标表中用户指定的字段数量，那么多余的字段将被插入NULL值。

[ROW_TERMINATOR=<row_terminator>] 该字串表示一行的结束。

[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE] 该字串用于指定导出数据是由单引号还是双引号括住。如果数据本身就含有引号，那么导出结果将用两个连续的引号来表示。

[LOB_FORMAT=INTERNAL | EXTERNAL] 该选项表示当导出大型对象类型（如blob、clob、nclob、nblob）的字段时，将生成外部文件。使用此命令会为每条记录中的每个大型对象字段生成一个外部文件。如果没有指定该选项，大型对象字段的内容将被包含在数据文件中。

为外部文件取名时，一定要注意以下几点：
blobtempdir<m>\blbtmpf<n>.<tmp | txt>

m 的作用是在本目录下，为新建的目录从1开始指定未使用的最小数字。

例如，如果已存在名为blobtempdir1、blobtempdir2和blobtempdir3的目录，那么新建的用于包含外部文件的目录名为blobtempdir4。

n 的作用是在本目录下，为新建的文件从1开始指定未使用的最小数字。

文件的扩展名是**tmp**还是**txt**是由导出字段的类型（BLOB、FILE、CLOB）决定的。如果字段的类型是BLOB或FILE，那么文件的扩展名为**tmp**，否则，文件的扩展名为**txt**。

server_column_name 该参数列出了源表中，将要被导出的数据库字段名。参数中的名称顺序代表了导出的字段顺序。如果没有该列表，那么源表中的所有字段都将按照源表中的顺序被导出。

```
FORMAT=VARIABLE
[COLUMN_DELIMITER=<delimiter>]
[ROW_TERMINATOR=<row_terminator>]
[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE]
[LOB_FORMAT=INTERNAL | EXTERNAL]
[<server_column_name>]
```

导入/导出数据规则

下表描述了从文件导入数据或将数据导出到文件中时，必须遵守的规则。

数据类型	导入/导出格式	示例
BINARY	使用十六进制（HEX）格式	要导入二进制数据“0x004D2”，数据文件中应该为004D2
CHAR	全部使用字符	要导入单词“inception”，数据文件中则应为inception
NCHAR	可以使用三种格式：auto格式、hex格式或字符格式。 嵌入格式中使用十六进制字符。 在描述文件中使用标记IMPORT_NCHAR_FORMAT来设置您的选项。 使用NCHAR_AUTO选项是首选使用十六进制（HEX）格式导入数据，若无法导入，则使用字符格式导入数据。 使用NCHAR_HEX格式是使用十六进制（HEX）格式导入数据。	要导入单词“word”，数据文件中应为77006f0072006400或word

数据类型	导入/导出格式	示例
	使用NCHAR_CHAR格式是使用字符格式导入数据。	
VARCHAR	见CHAR类型	
NVARCHAR	见NCHAR类型	
DATE	导出时使用YYYY/MM/DD的格式	要导出/入日期“2003/07/25”，数据文件中则应为2003/07/25
TIME	导出以及导入的格式为HH:MM:SS	要导入时间“14:30:25”，数据文件中则应为14:30:25
TIMESTAMP	TIMESTAMP类型的格式是DATE和TIME类型格式的结合	要导入时间戳“2003/07/25 14:30:25”，数据文件中则应为2003/07/25 14:30:25
DECIMAL	用numeric类型数据来表示	要导入数据“36.82”，数据文件中则应为36.82
DOUBLE	和DECIMAL类型一样用numeric类型数据或采用科学计数法	要导入数据“13e+12”，数据文件中应为：13e+12
FLOAT	见DOUBLE类型	
INTEGER	用整型数据	要导入整数“576”，数据文件中应为576
LONG VARBINARY	可以使用两种格式：嵌入格式或外部文件格式。 嵌入格式中使用十六进制字符。 而外部文件格式中应该制定URL	（1）嵌入格式：用法和BINARY类型的用法一样。 （2）外部文件格式：例如，如果用户

数据类型	导入/导出格式	示例
	使用描述标记LOB_FORMAT来设置您的选项。详细信息请参看描述文件设置。	想导入一个完整路径为“c:\My Document\GRAPH.GIF”的二进制文件，URL的值就必须为“c:\My Document\GRAPH.GIF”
LONG VARCHAR	和LONG VARBINARY相似，可以使用两种格式。只是输入的数据是ASCII串，而不是十六进制串。	（1）嵌入格式和CHAR类型的格式相同 （2）外部文件格式和LONG VARBINARY 类型的格式相同
FILE	对于FILE类型，导入和导出的规则和LONG VARBINARY 类型的规则相同。	
OID	和INTEGER类型的规则一样	
SERIAL	和INTEGER类型的规则一样	
SMALLINT	和INTEGER类型的规则一样	
NULL DATA	对于可变格式，当两个连续的分隔符之间什么都没有时，就被认为是NULL数据。 对于固定格式，当两个字段之间仅为空格字符时就被认为是NULL数据。	

表 6-2 导入/导出数据规则

6.7 IMPORT

Import命令可用来从文本文件中抽取数据，然后再将这些数据插入到数据库的表中。导入命令界面的作用是设置命令选项，而描述文件的作用则是设置导入文件格式。

导入命令界面

导入命令界面为您提供了若干个导入数据选项，这些选项包括对导入数据的中断标准控制、将错误记入日志以及源数据文件的数据编码。源数据文件的格式是在描述文件中描述的。

[<owner_name>.]<table_name>……该字串用来指定数据文件中的数据将被导入到哪个表中。如果没有指定*<owner_name>*，那么当前连接数据库的用户将被指派为表的所有者。

[FROM <data_file>]……该字串代表要导入的实际包含数据的文件。如果您没有指定*data_file*，那么默认的文件名将为*<table_name>_in.txt*，其中*table_name* 为表的名称。例如，如果导入的表名为**t1**并且没有在命令中指定数据文件的文件名，那么默认的文件名则为**t1_in.txt** 。

[DESCRIPTION <description_file>]……这是描述数据文件中数据格式的描述文件。如果没有设置该选项，那么描述文件的文件名将被默认为*<table_name>_in.dsc*，其中*table_name* 为表名称。例如，如果导入的表名称为**t1**并且没有在命令中指定描述文件的名称，那么描述文件的文件名将被默认为**t1_in.dsc** 。如果系统中不存在这个文件，那么系统将使用默认的描述文件格式，即可变描述文件格式。

[LOG <log_file>]……该文件记录了导入数据过程中出现的错误。这个日志文件可以显示引发错误发生的记录，同时也将显示相应的错误信息。如果没有设置该选项，那么系统将使用默认的日志文件名*import.log* 来作为日志文件的名称。

[STOP_ON_ERROR] ……该参数表示在载入数据的过程中如果出现错误，将停止数据的载入。如果没有指定该选项，那么即使在发生错误的情况下，数据的载入仍将继续进行。

```
IMPORT [<owner name>.]<table name>
[FROM <data file>]
[DESCRIPTION <description file>]
[LOG <log file>]
[STOP_ON_ERROR]
```

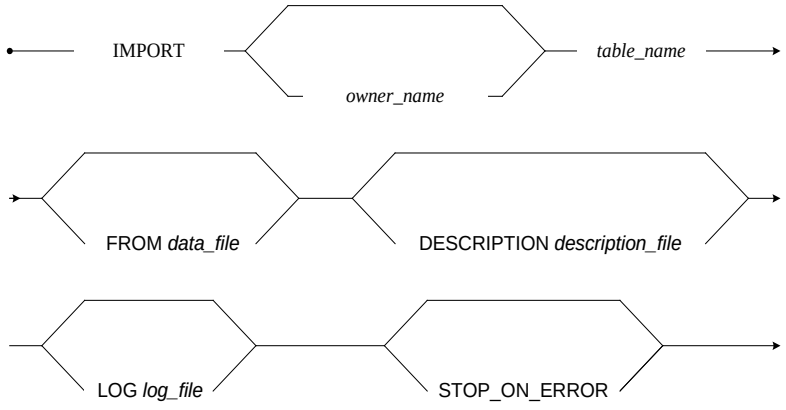


图 6-7 IMPORT语法图

文件描述

您可以使用两种类型的描述文件格式：固定格式和可变格式。描述文件中的错误分析应该尽可能地显示清楚，这样您就可以检查错误信息，确定错误发生的原因。错误信息可以显示分析某个命令时出现的问题。

固定格式的描述文件

FORMAT=FIXED……将描述文件的格式设置为固定格式也就意味着描述文件将为定长的数据文件描述数据格式。

[START_WITH_ROW=<row_number>]……您可以使用这个字串来指定希望从哪条记录开始导入数据。如果您没有指定这一选项，那么 *row_number* 的默认值为1。如果*START_WITH_ROW* 值比数据文件中的所有记录数还大，该命令将不会导入任何数据，*row_number* 必须是一个正数。

[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>]……
您可以指定每个提交事务所导入的记录数。如果没有指定该选项，那么DBMaster会每5条记录提交一次。如果变量的值设为**-1**，那么事务将不被提交，在这种情况下，如果您希望成功地导入数据，就必须以手动的方式提交事务。如果将变量值设为**0**，那么整个导入将被看作一个事务。导入完成后系统将自动发出提交事务的命令。

导入记录时即使出现了错误，提交的记录数仍被增加一条。例如，您设置的字串为：
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=10，并且在导入第四条记录时发生了错误，那么第一到第三条记录以及第五到第十条记录仍将被提交，并且第一到第十条记录依然被看作一个完整的事务单元。当然，如果您指定了STOP_ON_ERROR命令，那么第五到第十条记录将不会被提交，只有第一到第三条记录才会被提交。

该选项只有在auto-commit处于关闭的状态下才能生效。

*[LOB_FORMAT=INTERNAL | EXTERNAL]……*如果clob/blob格式是内部文件，那么数据文件中的文本将被作为数据来导入。否则，文件中的文本被作为外部文件的URL被导入。

*server_column_name……*这个参数列出了将要从数据文件导入到目标表中的字段名称。如果字段名中存在空格或等于（**=**）符号，请用双引号将该字段名括住。

*column_position……*该参数用来设置字段在数据文件中的位置（用字节表示）。

*server_columnname*和*column_position*中的参数值由空格符分隔。*column_position* 的值由两个数字组成，它们之间用冒号（:）分隔。例如，1:40的意思是用户应该在数据文件的第1个字节到第40个字节之间查找数据。您应该使用空格字符来垂直排列数据字段。如果源表中的数据大小超过了字段长度，那么超出的部分将被删除。每一行可以由换行符或换行符加回车来结束，具体取决于执行导入操作的系统是否是Windows平台。如果一行中的内容小于最大长度，那么多于的空间将用空格来填补；如果一行的内容大于最大长度，那么多于的内容将被忽略。

```
FORMAT=FIXED
```



```
[START WITH ROW=<row number>]
[NUMBER OF ROWS FOR EACH TRANSACTION=<number>]
[LOB FORMAT=INTERNAL | EXTERNAL]
<server_column_name> <column_position>
```

注意 *字段server_column_name，column_position 中的值用空格符来分隔。*

➲ **下面是一个用固定模式的描述文件导入数据的例子：**

数据文件如下：

```
Davolio Nancy ......       Sales Representative      Ms.
Fuller Andrew ......       Vice President, Sales     Dr.
Leverling Janet ....       Sales Representative      Ms.
Peacock Margaret ...       Sales Representative      Mrs.
Buchanan Steven ....       Sales Manager             Mr.
Suyama Michael .....       Sales Representative      Mr.
King Robert .............  Sales Representative      Mr.
```

数据文件的描述文件如下：

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
Name 1:20
Position 20:45
Gender 50:54
```

可变格式的描述文件

```
FORMAT=VARIABLE
[START_WITH_ROW=<row_number>]
[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>]
[{COLUMN_DELIMITER=<delimiter>}]
[ROW_TERMINATOR=<row_terminator>]
[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE]
[ESCAPE_CHAR=YES|NO]
[LOB_FORMAT=INTERNAL | EXTERNAL]
[<server_column_name> <column_number>]
```

*FORMAT=VARIABLE……*该字串用于指定文件包含的是变长描述文件的数据格式。

[START_WITH_ROW=<row_number>]……您可以用这个字串来指定希望从哪条记录开始导入数据。如果您没有指定这一选项，*row_number* 的默认值将为***1***。如果*START_WITH_ROW* 的值比数据文件中的总记录数大，那么该命令将不会导入任何数据，*row_number* 必须是一个正数。

[NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=<number>]…….您可以指定每个提交的事务中所导入的记录数。如果没有指定该选项，那么DBMaster会每5条记录就提交一次。如果变量的值设为**-1**，那么事务将不被提交，在这种情况下，如果您还希望成功地导入数据，就必须以手动的方式提交事务。如果变量值为**0**，整个导入将被看作一个事务。导入完成后系统会自动发出提交事务的命令。

导入记录时即使出现了错误，提交的记录数仍将被增加一条。

例如，您设置的字串为：
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=10，并且在导入第四条记录时发生了错误。那么第一到第三条记录以及第五到第十条记录仍被提交，并且第一到第十条记录依然被看作一个完整的事务单元。当然，如果您指定了STOP_ON_ERROR命令，那么第五到第十条记录将不会被提交，只有第一到第三条记录才会被提交。

该选项只有在auto-commit处于关闭的时候才有效。

[COLUMN_DELIMITER=<delimiter>]……这条命令的作用是为数据文件中的每个导入字段设置分隔符，该字符应该用单引号括起来。例如，如果将**SPACE** 设置为字段分隔符，您就应该在*<delimiter>*中使用‘ ’。除了一般的字符外，您还可以使用下面的转义字符串代表的特殊字符。

CHARACTER	ESCAPE SEQUENCE REPRESENTATION
TAB	\t
NEW LINE	\n

表 6-3 字符和转义序列

例如，如果TAB是分隔符，那么在*<delimiter>*中您应该使用‘\t’。如果没有指定字段分隔符，我们将使用TAB（\t）来作为字段分隔符。选择分隔符时您应该考虑周到。

如果字段分隔符的数量少于目标表中用户指定的字段数量，那么多余的字段将被插入NULL值。

[ROW_TERMINATOR=<row_terminator>]……该字串表示一行的结束。*row_terminator* 应该用双引号括起来。字段分隔符的转义字符串规则也可以应用于行结束符。此外，回车符也可以用作转义字符串。

CHARACTER	ESCAPE SEQUENCE REPRESENTATION
CARRIAGE RETURN	\r

表 6-4 字符和转义序列

例如，如果行结束符是回车符加换行符，那么*<row_terminator>*的值就应该是“\r\n”。如果没有指定行结束符，那么换行符（\n）将被默认为行结束符。行结束符中的字符数量不能超过2。

注意，行结束符中不能包含字段分隔符。

[QUOTATION=SINGLE_QUOTE | DOUBLE_QUOTE] ……表示数据源文件字段中的数据是否会被加上引号。选择SINGLE_QUOTE表示字段中的数据将被单引号括住，选择DOUBLE_QUOTE则表示字段中的数据将用双引号括住。

[ESCAPE_CHAR=YES | NO]…… 该字串表示是否使用转义字符（\），默认值为YES。如果使用转义字符，转义字符后的字段分隔符将被看作真实的数据。例如，如果我们把TAB作为字段分隔符，同时将ESCAPE_CHAR的值设为YES，那么\TAB则表示数据TAB，而不是字段分隔符。对于行结束符，\n将被看作真实的数据，因此行将继续而并非结束。这个规则在引号作为转义字符时也适用。

[LOB_FORMAT=INTERNAL | EXTERNAL] ……如果clob/blob格式是内部文件，那么数据文件中的文本将被作为数据来导入。否则，文件中的文本将作为外部文件的URL被导入。

server_column_name……该参数列出了将要从数据文件导入到目标表中的字段名称。如果字段名中包含空格或等于（=）符号，请用双引号将该字段名括住。

*column_number……*该参数用于设置字段在数据文件中的位置（用字节表示）。

*server_column_name*和*column_number*被空格所分隔。

注意 *如果没有指定server_column_name 和 column_number，那么数据文件中的所有字段都将按照数据文件中字段的顺序被导入到目标表中。也就是说，数据文件中的第一个字段将被导入到表中作为第一个字段，数据文件中的第二个字段将被导入到表中作为第二个字段，依次类推。如果数据文件中的字段数量大于目标表中的字段数量，那么数据文件中多余的字段将被忽略，如果数据文件中的字段数量少于目标表中的字段数量，那么目标表中多余的字段将被插入NULL。*

默认可变格式的描述文件

用户可以选择是否为自己的数据文件格式指定描述文件。如果用户没有指定描述文件，那么系统将使用默认的描述文件格式来作为数据文件格式。默认的描述文件格式如下（在Win32平台下，并且ROW_DELIMITER="\r\n"）：

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
COLUMN_DELIMITER="\t"
ROW_TERMINATOR="\n"
```

➲ 下例表示使用可变格式的描述文件来导入数据：

数据文件如下：

```
Davolio Nancy,Sales Representative,Ms.
Fuller Andrew,"Vice President, Sales",Dr.
Leverling Janet,Sales Representative,Ms.
Peacock Margaret,Sales Representative,Mrs.
Buchanan Steven,Sales Manager,Mr.
Suyama Michael,Sales Representative,Mr.
King..............Robert,Sales Representative,Mr.
```

数据文件的描述文件如下：

```
START_WITH_ROW=1
NUMBER_OF_ROWS_FOR_EACH_TRANSACTION=5
COLUMN_DELIMITER=","
```



```
ROW TERMINATOR="\n"
DOUBLE QUOTE
Name 1
Position 2
Gender 3
```

导入/导出数据规则

下表描述了从文件导入数据或将数据导出到文件时，必须遵守的规则。

数据类型	导入/导出格式	示例
BINARY	使用十六进制（HEX）格式	要导入二进制数据“0x004D2”，数据文件中应该为004D2
CHAR	全部使用字符	要导入单词“inception”，数据文件中则应为inception
NCHAR	可以使用三种格式：auto格式、hex格式或字符格式。 嵌入格式中使用十六进制字符。 在描述文件中使用标记IMPORT_NCHAR_FORMAT来设置您的选项。 使用NCHAR_AUTO选项是首选使用十六进制（HEX）格式导入数据，若无法导入，则使用字符格式导入数据。 使用NCHAR_HEX格式是使用十六进制（HEX）格式导入数据。 使用NCHAR_CHAR格式是使用字符格式导入数据。	要导入单词“word”，数据文件中应为77006f0072006400或word
VARCHAR	见CHAR类型	
NVARCHAR	见NCHAR类型	

数据类型	导入/导出格式	示例
DATE	导出时使用YYYY/MM/DD的格式	要导出/入日期“2003/07/25”，数据文件中则应为2003/07/25
TIME	导出以及导入的格式为HH:MM:SS	要导入时间“14:30:25”，数据文件中则应为14:30:25
TIMESTAMP	TIMESTAMP类型的格式是DATE和TIME类型格式的结合	要导入时间戳“2003/07/25 14:30:25”，数据文件中则应为2003/07/25 14:30:25
DECIMAL	用numeric类型数据来表示	要导入数据“36.82”，数据文件中则应为36.82
DOUBLE	和DECIMAL类型一样用numeric类型数据或采用科学计数法	要导入数据“13e+12”，数据文件中应为：13e+12
FLOAT	见DOUBLE类型	
INTEGER	用整型数据	要导入整数“576”，数据文件中应为576
LONG VARBINARY	可以使用两种格式：嵌入格式或外部文件格式。 嵌入格式中使用十六进制字符。 而外部文件格式中应该指定 URL 使用描述标记LOB_FORMAT来设置您的选项。详细信息请参看描述文件设置。	（1）嵌入格式：用法和BINARY类型的用法一样。 （2）外部文件格式：例如，如果用户想导入一个完整路径为“c:\My Document\GRAPH.GIF”的二进制文件，URL的值就必须

数据类型	导入/导出格式	示例
		为c:\My Document\GRAPH.GIF
LONG VARCHAR	和LONG VARBINARY相似，可以使用两种格式。只是输入的数据是ASCII串，而不是十六进制串。	（1）嵌入格式和CHAR类型的格式相同 （2）外部文件格式和LONG VARBINARY 类型的格式相同
FILE	对于FILE类型，导入和导出的规则和LONG VARBINARY 类型的规则相同。	
OID	和INTEGER类型的规则一样	
SERIAL	和INTEGER类型的规则一样	
SMALLINT	和INTEGER类型的规则一样	
NULL DATA	对于可变格式，当两个连续的分隔符之间什么都没有时，就被认为是NULL数据。 对于固定格式，当两个字段之间全是空格字符时就被认为是NULL数据。	

表 6-5 导入/导出数据规则

6.8 LOAD

Load命令是dmSQL提供的一种工具，其功能是将被载出到外部文本文件中的数据库对象再载入到数据库中。载入命令有七个选项：数据库（database）、表（table）、结构（schema），数据（data）、项目（project）、模块（module）和存储过程（procedure）。载入时您所用的选项必须和载出时的选项一样。例如：从一个文本文件载入数据库时，该文本文件必须是在载出数据库应用相同的选项生成的。

载入文本文件时，您需要设置自动执行事务的命令数量，其默认值为1000。数量*n*的大小将影响事务是否完成以及载入的速度。如果*n*值很大，那么日志文件很容易就被装满，这可能会导致事务的失败；如果*n*值很小，这将增加命令执行的时间，降低载入的速度。

如果在载入过程中出现错误，该错误信息将记录到日志文件中，系统可以利用这个日志文件来撤销已执行的命令。日志文件被存储在与被载入的外部文件相同的目录下，这时并不停止载入过程。

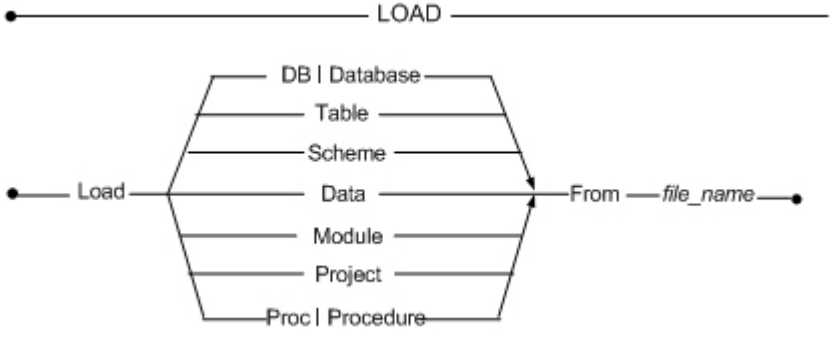


图 6-8 LOAD语法图

LOAD DB [DATABASE]

您可以使用该命令将一个数据库的内容转入另一个新的数据库中。首先，将数据库的内容载出到一个外部文本文件中，然后再使用“load db”命令载入该文本文件中的数据库内容。在载入数据库之前，应该先

新建一个数据库。新数据库的名称可不同于旧数据库的名称。只有DBA、SYSDBA或SYSADM才有权执行该命令。

然而，如果用户在使用命令UNLOAD DB TO *file_name*之前下达了如下命令：SET UNLOAD EXTERNAL '*connection_string*'(*connection_string*的格式为 " DSN =<*数据库名*>；UID =<*用户名*>；PWD =<*密码*>;")，dmSQL将不载出数据到脚本文件empdb.s0中。因此，当用户使用该脚本文件载入数据库时，dmSQL将连接ODBC数据源，从中读取数据并直接将读取的数据保存到本地数据库中。dmSQL在脚本文件中使用"set external [database|db] '*connection_string*'"命令来连接外部数据库，若失败，将返回一条错误信息。dmSQL仅保留最后一次外部数据库的连接，因此在设置新连接时，请先断开之前的连接。此外，由于没有断开连接的命令，用户只有在关闭dmSQL时才能断开外部数据库。如果将loaddb设置为安全模式（safe mode），那么载入是在日志模式开启的状态下进行的。如果在载入过程中出现了错误，那么载入工具将回滚到最后执行的命令，并返回错误信息，同时将错误信息写入日志文件中。

如果将loaddb设置为快速模式（fast mode），则根据DBMaster3.6之前版本的载入规则，该载入过程将在日志模式关闭的情况下进行。将loaddb设置在快速模式下可以加快载入的速度，但一旦载入过程中出现了错误，处于非日志模式下的数据库就将被关闭。

例如，假设载入文件生成了一个表空间，但是没有在**dmconfig.ini**中设置。如果loaddb使用“安全模式”选项，DBMaster将报出如下错误：“ERROR(8002): [DBMaster] 配置文件中必须有关键字”，并回滚这条载入命令。如果loaddb使用“快速模式”选项，DBMaster将报出如下错误：“ERROR(30017), [DBMaster] 无日志模式下发生错误，请关闭数据库”。loaddb的默认选项是“set loaddb safe”。

➲ **例**

下面在DBMaster3.6以上版本中设置**loaddb**的选项：

```
SET LOADDB [SAFE | FAST]
```

LOAD TABLE

该选项的作用是从一个文本文件中载入一个表中的内容，包括表结构和表中的数据。执行该操作时，必须确保表名称的唯一性。

LOAD SCHEMA

该选项允许用户从包含在文本文件的表中只载入表结构，而不载入表中的数据。执行该操作时，必须确保表名称的唯一性。

LOAD DATA

当从外部文本文件载入表中的数据时，相应的表必须是存在的。在3.6以前的版本中，如果在LOAD DATA过程中出现了错误，数据库将被回滚到最后一条成功执行命令后的状态。

如果设置了*loaddata skip error*选项，那么载入数据库过程中的如下错误将被跳过：

ERROR(401).....违反键值唯一的规定。

ERROR(410).....违反参照完整性约束：父键中不存在该值。

ERROR(6521)....表或视图不存在。

ERROR(6002)....此处或附近存在语法错误。

ERROR(6015)....输入的SQL命令不完整。

以上错误可以被系统忽略，载入工具可以继续执行以后的命令。上面的错误都是一些在载入数据过程中最常出现的错误。如果载入数据时设置了*stop*或*stop on error*选项，那么错误发生后这个载入命令都将被回滚。该选项的默认值是*loaddata skip* [*error*]。载入数据过程中出现的所有错误信息都将被写入日志文件。

➲ **例**

DBMaster 3.6及以上的版本支持忽略错误的选项：

```
SET LOADDATA SKIP [ERROR] | STOP [ON ERROR]
```

LOAD MODULE

该选项允许用户从外部文本文件中载入模块。

LOAD PROJECT

该选项允许用户从外部文本文件中载入一个项目。

LOAD PROC [PROCEDURE]

该选项允许用户从外部文本文件中载入一个存储过程。

➲ **例1**

下面的命令是从一个名为“**empdb**”的文件中载入数据库，载入过程中每**100**条命令自动提交一次。系统将在相同目录下生成一个名为“**empdb.log**”的日志文件。

```
dmSQL> LOAD DB FROM empdb 100;
```

➲ **例2**

下面的命令从名为“**empfile**”的文件中载入一张表，载入过程中每**50**条命令自动提交一次。

```
dmSQL> LOAD TABLE FROM empfile 50;
```

➲ **例3**

下面的命令从一个名为“**datafile**”的外部数据文件中载入数据。载入过程中系统默认每**1000**条命令提交一次。

```
dmSQL> LOAD DATA FROM datafile;
```

6.9 SET DUMP PLAN

导出计划包括若干个ON块。查询优化器可以优化查询并且将一个查询分成几个逻辑ON块。简单的查询以及连接查询一般只产生一个ON块，而有些查询里还嵌套子查询，对于这样的复杂查询，就不只产生一个ON块了，其中包括有主块和子块。

查询优化器依据成本规则来为每个ON块选择一个最好的执行方法。查询优化器还可以将每个ON块分解为若干个PL块，而每个PL块表示一个操作，比如扫描、连接……

Set dump plan on…… 打开导出计划选项，DBMaster就会导出后面查询语句的执行计划并执行这些命令。

*Set dump plan off……*关闭导出计划选项，DBMaster只是执行后面的命令而不导出SQL语句的执行计划。该选项是默认的。

*Set dump plan only……*打开这个选项，DBMaster只导出后面语句的执行计划但并不执行这些命令。

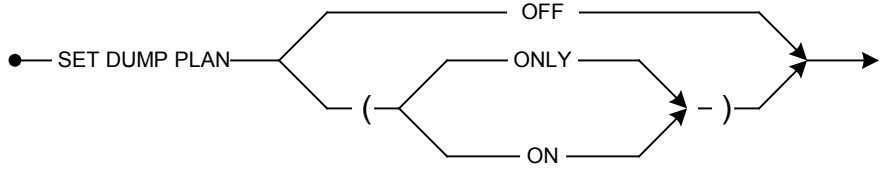


图 6-9 SET DUMP PLAN语法图

➲ 例

```
dmSQL> SET DUMP PLAN ON;
dmSQL> SELECT * FROM tb_tmp ORDER BY c01_int;
dmSQL> SET DUMP PLAN OFF;
```

6.10 START DATABASE

为了能使用户连接到数据库上，您必须先用START DATABASE命令启动数据库。该命令一般只能用在客户端/服务器数据库上。只有DBA、SYSDBA或SYSADM才有权执行该命令。

如果用START DATABASE命令来启动数据库时没有指定用户名和密码，那么系统就使用**dmconfig.ini**文件中由**DB_Usrld**和**DB_PasWd**这两个关键字指定的用户名和密码来连接数据库。

密码采用纯文本的格式，任何有权读取**dmconfig.ini**文件的用户都可以看到。这种密码的显示方式使用很方便，但如果在不安全的计算机系统中，该方式可能会给数据库的安全带来风险。

database_name. 要启动的数据库名称。

user_name..... 启动数据库的用户名称。

password ..... 用户*user_name*的当前密码。

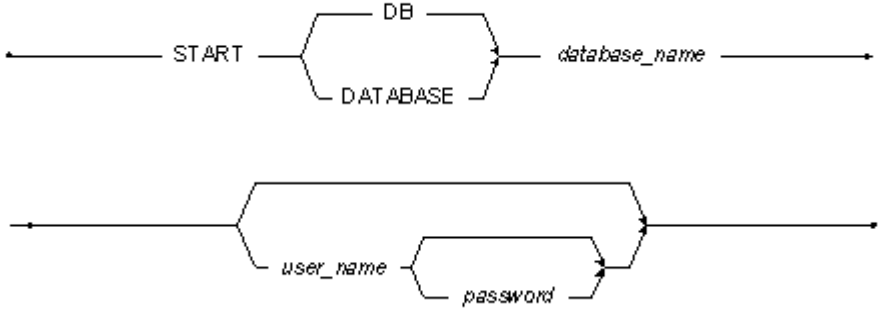


图 6-10 START DATABASE语法图

➲ **例**

下例中用户**vivian**将启动数据库**Employees**，其中**vivian**必须具备DBA、SYSDBA或者SYSADM权限，密码为**shuka828**。

```
dmSQL> START DATABASE Employees vivian shuka828;
```

6.11 TERMINATE DATABASE

TERMINATE DATABASE命令用来关闭数据库，这样其他用户就不能再连接该数据库了。该命令一般只用在客户端/服务器数据库上。只有DBA、SYSDBA或SYSADM才有权执行该命令。

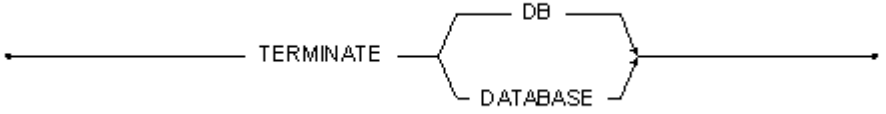


图 6-11 TERMINATE DATABASE语法图

➲ **例**

下例将关闭当前数据库。

```
dmSQL> TERMINATE DATABASE;
```

6.12 UNLOAD

Unload命令是dmSQL提供的一种工具，功能是将数据库中的内容载出到外部文本文件中。载出命令成功完成后，dmSQL将产生两种文本文件。其中一个存储脚本，文件的扩展名是s0，用来创建数据库对象，而另外一些文件存储BLOB数据，文件扩展名是bn。

载出命令有八个选项：数据库（database）、表（table）、结构（schema）、数据（data）、项目（project）、模块（module）、存储过程（procedure）以及存储过程定义（procedure definition）。用户只能载出在其上具有SELECT权限的对象。例如，如果您在一张表上具有SELECT权限，那么您就只能载出该表中的内容。只有DBA、SYSDBA或SYSADM才可以载出一个数据库。

如果要载出的表名称中含有通配符，您只需使用转义符或在该名字两边加上双引号即可。

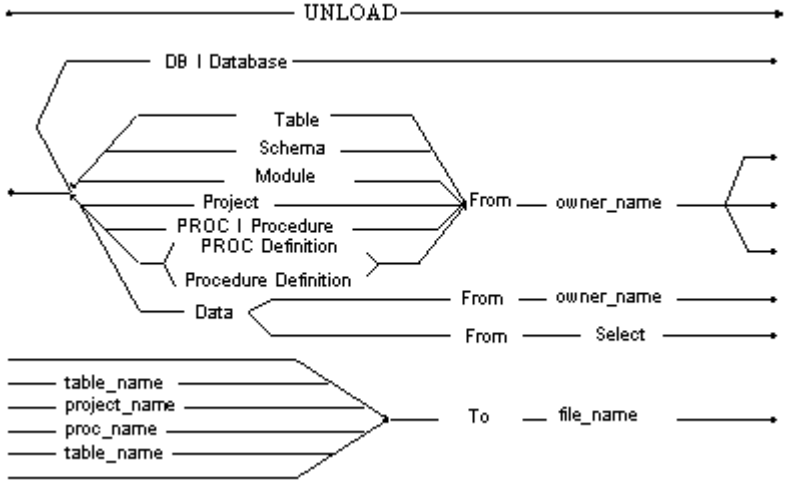
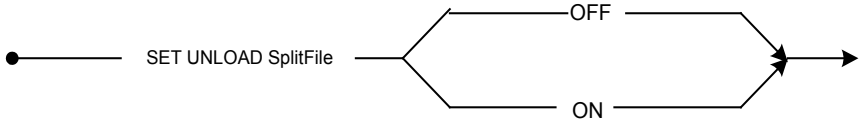


图 6-12 UNLOAD语法图

用户可以指定是否使用**set unload splitfile on/off**命令来拆分载出的脚本文件。一旦开启拆分功能，载出的文件将以表的定义、数据、索引以及其他相关信息等为单位分开存放。默认的设置是关闭该功能。

Set unload splitfile on ……拆分载出的脚本文件。

Set unload splitfle off …… 默认设置，它仅仅将数据库内容载出到文件<external text file name>.bn 和<external text file name>.so。



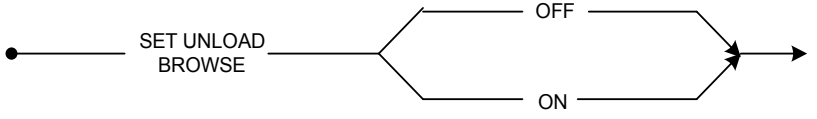
➲ 例

```
dmSQL> SET UNLOAD SPLITFILE ON;
dmSQL> UNLOAD DB TO empdb;
dmSQL> SET UNLOAD SPILTFILE OFF;
```

执行载出数据操作之前，用户可以指定是否使用**set unload browse on/off**命令来确保载出命令和其它DML可同时使用，同时确保载出数据的一致性。默认的设置是关闭该功能。

Set unload browse on ……载出命令和其它DML可同时使用，此时载出数据中包括脏数据。

Set unload browse off ……载出命令和其它DML不能同时使用。



➲ 例

```
dmSQL> SET UNLOAD BROWSE ON;
dmSQL> UNLOAD DB TO empdb;
```

```
dmSQL> SET UNLOAD BROWSE OFF;
```

UNLOAD DB [DATABASE]

DBA、SYSDBA或SYSADM才可以使用该选项来将数据库中的内容载出到一个外部文本文件中。这个文件中将包含有关安全、表空间、数据库对象的定义、索引、同义字、数据等信息。对于每个数据库，dmSQL至少生成两个外部文件：一个是脚本文件，另一个中则包含BLOB数据。

假设**empdb**是外部文本文件的文件名。默认情况下，dmSQL会在当前工作目录下生成这些文件。下面的语句将至少生成两个文本文件：empdb.s0和empdb.b0。如果被导出的BLOB文件empdb.b0超出了操作系统允许的最大长度，dmSQL将生成empdb.b1、empdb.b2等，直到empdb.bn，其中n的最大值为99。dmSQL一般只生成一个脚本文件empdb.s0，最大长度为操作系统允许的最大长度。

➲ **例1**

```
dmSQL> UNLOAD DB TO empdb;
```

然而，如果用户在使用命令UNLOAD DB TO *file_name*之前下达了如下命令：SET UNLOAD EXTERNAL '*connection_string*'（*connection_string*的格式为 "DSN=<*数据库名*>； UID=<*用户名*>；PWD=<*密码*>;"），dmSQL将不会载出数据到脚本文件empdb.s0中，而会在empdb.s0文件中产生"set external db '*connection_string*'"，并且载出的表数据将会出现"load external db from 'select * from *external_table_name*' into *local_table_name*"的提示语句。示例如下：

➲ 例2

```
dmSQL> SET UNLOAD EXTERNAL 'DSN=DBSAMPLE5;UID=SYSADM;PWD=;';
dmSQL> UNLOAD DB TO empdb;
```

以下是脚本文件empdb.s0：

```
…
set external db 'DSN=DBSAMPLE5;UID=SYSADM;PWD=;';
create table Lauser1.Latb3 (
 c1  SMALLINT default null ,
 c2  FLOAT default null ,
 c3  DOUBLE default null ,
 c4  DECIMAL(10, 3) default null ,
```

```
 c5  CHAR(10) default null ,
 c6  BINARY(12) default null )
 in DEFTABLESPACE  lock mode page  fillfactor 100 ;
load external database from 'select * from Lauser1.Latb3' into Lauser1.Latb3;
create  index idx31 on Lauser1.Latb3 ( c1 asc ) in DEFTABLESPACE;
create  index idx32 on Lauser1.Latb3 ( c3 desc ) in DEFTABLESPACE;
create  index idx33 on Lauser1.Latb3 ( c5 asc ) in DEFTABLESPACE;
…
```

UNLOAD TABLE

该选项的作用是将表载出到一个外部文件中，这个文件将记录表的定义、同义字、索引、主键、外键的相关信息以及表中的数据。

在用户名和表名称中可以有通配符“_”和“%”，这有点像DOS下的“?”和“*”。通配符“_”表示一个字符，而“%”则表示多个字符。

UNLOAD SCHEMA

该选项的作用与unload table相似。不过它只能载出一个表的定义，并不能载出表中的数据。通配符的用法和在unload table选项中所描述的用法相同。

UNLOAD DATA

该选项的作用是载出表中的所有数据，但不载出表的定义。Unload data中的通配符用法与上面两个选项中的通配符用法相同。只有在要载出的表上具备SELECT权限的用户才可以执行该命令。

DBMaster 3.6以及以后的版本还支持另外一种语法：**dmSQL>unload data from (select statement) to file_name**。如果select语句是一个连接查询，那么选择查询的字段必须是同一张表中的字段。下面例子的语句是可执行的，但DDL命令、删除、插入或更新语句是不允许在这里使用的。

➲ **例1**

正确的语法：

```
dmSQL> UNLOAD DATA FROM (SELECT tb_doc.c01_int, tb_doc.c02_char FROM tb_doc,
tb_txt WHERE tb_doc.c01_int= tb_txt.c01_int) TO f1;
```

➲ **例2**

错误的语法：

```
dmSQL> UNLOAD DATA FROM (SELECT tb_doc.c01_int, tb_txt.c01_int FROM tb_doc,
tb_txt WHERE tb_doc.c01_int = tb_txt.c01_int) to f1;
```

➲ **例3**

错误的语法，因为查询字段中不能有集合函数或内置函数。

```
dmSQL> UNLOAD DATA FROM (SELECT AVG(c01_int) FROM tb_doc) TO f1;
dmSQL> UNLOAD DATA FROM (select now()FROM tb_doc) TO f1;
```

➲ **例4**

正确语法，select语句中可以使用同义字和视图。

```
dmSQL> UNLOAD DATA FROM (SELECT * FROM syn_tmp WHERE c01_int > 10) TO f1;
dmSQL> UNLOAD DATA FROM (SELECT * FROM view_tmp WHERE c01_int < 10) TO f1;
```

UNLOAD PROJECT

该选项允许用户将一个项目载出到一个外部文本文件中。

UNLOAD MODULE

该选项允许用户将模块载出到一个外部文件中。

UNLOAD [PROC | PROCEDURE]

该选项允许用户将存储过程载出到一个外部文件中。

UNLOAD [PROC DEFINITION | PROCEDURE DEFINITION]

该选项允许用户将存储过程的定义载出到一个外部文件中。

➲ **例1**

下例将载出当前用户的表“**e tab**”，如果表名称中含有空格，那么您需要在表名称两边加上双引号。

```
dmSQL> UNLOAD TABLE FROM "e tab" TO empfile;
```

➲ **例2**

下例将载出**SYSADM**的以**emp**开头的所有表，例如**emptab**、**empname**等。

```
dmSQL> UNLOAD TABLE FROM SYSADM.emp% TO empfile;
```

➲ **例3**

下例载出所有表名为**ktab**的表结构。

```
dmSQL> UNLOAD SCHEMA FROM %.ktab TO kfile;
```

➲ **例4**

下例是载出一个名为**abc%**的表中的数据。

```
dmSQL> UNLOAD DATA FROM abc\% TO abcfile;
dmSQL> UNLOAD DATA FROM "abc%" TO abcfile;
```