

达梦技术手册

## DM8\_DIsql 使用手册

Service manual of DM8\_DIsql



# 前言

## 概述

本文档主要介绍如何使用 DM 的命令行交互式工具 DIsql，以及它作为数据库访问工具所提供的功能。

## 读者对象





本文档主要适用于 DM 数据库的：

- 开发工程师
- 测试工程师
- 技术支持工程师
- 数据库管理员

## 通用约定

在本文档中可能出现下列标志，它们所代表的含义如下：

表 0.1 标志含义

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

在本文档中可能出现下列格式，它们所代表的含义如下：

表 0.2 格式含义

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

## 访问相关文档

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

您也可以通过访问我们的网站阅读或下载 DM 的各种相关文档。

## 联系我们

如果您有任何疑问或是想了解达梦数据库的最新动态消息，请联系我们：

网址：[www.dameng.com](http://www.dameng.com)

技术服务电话：400-991-6599

技术服务邮箱：[dmtech@dameng.com](mailto:dmtech@dameng.com)

# 目录

<b>1 功能简介.....</b>	<b>1</b>
<b>2 DISQL 入门.....</b>	<b>2</b>
2.1 启动 DISQL.....	2
2.1.1 在 WINDOWS 系统中启动 DIsql .....	2
2.1.2 命令行启动 DIsql .....	4
2.2 切换登录 .....	10
2.2.1 LOGIN /LOGOUT .....	10
2.2.2 CONN[ECT] /DISCONN[ECT] .....	11
2.3 使用 DISQL.....	12
2.4 退出 DISQL.....	12
<b>3 DISQL 环境变量设置 .....</b>	<b>14</b>
3.1 DISQL 环境变量 .....	14
3.2 SET 命令用法 .....	16
3.3 用 SET 命令设置环境变量详解 .....	17
3.3.1 AUTO[COMMIT] .....	17
3.3.2 DEFINE .....	17
3.3.3 ECHO .....	18
3.3.4 FEED[BACK] .....	18
3.3.5 HEA[DING] .....	18
3.3.6 LINESHOW.....	19
3.3.7 NEWP[AGE] .....	20
3.3.8 PAGES[IZE] .....	20
3.3.9 TIMING .....	20
3.3.10 TIME.....	20
3.3.11 VER[IFY] .....	21
3.3.12 LONG.....	22
3.3.13 LINESIZE .....	23

3.3.14	SERVEROUT [PUT] .....	23
3.3.15	SCREENBUFSIZE .....	24
3.3.16	CHAR_CODE.....	25
3.3.17	CURSOR.....	25
3.3.18	AUTOTRACE.....	25
3.3.19	DESCRIBE .....	26
3.3.20	TRIMS [POOL] .....	26
3.3.21	LOBCOMPLETE .....	26
3.3.22	COLSEP.....	26
3.3.23	KEEPDATA .....	27
3.3.24	AUTORECONN .....	27
3.3.25	NEST_COMMENT.....	27
3.3.26	NULL_ASNULL.....	28
3.3.27	CMD_EXEC .....	29
3.3.28	CHARDEL.....	30
3.3.29	FLOAT_SHOW .....	31
3.3.30	CONSOLE_PRINT .....	32
3.3.31	SQLCODE.....	33
3.3.32	SQL_LINESHOW.....	33
3.3.33	NULL_SHOW.....	34
3.3.34	SQLPROMPT.....	35
3.3.35	ISQL_MODE.....	35
3.3.36	WRAP.....	36
3.3.37	CTRL_INFO.....	37
3.3.38	RETRY_CONN.....	39
3.3.39	RETRY_CONN_TIME.....	40
3.4	SHOW 命令查看环境变量.....	41
3.4.1	显示当前环境变量 .....	41
3.4.2	显示初始化参数 .....	41
3.5	使用配置文件设置环境变量.....	42

---

<b>4 DISQL 常用命令 .....</b>	<b>44</b>
4.1 帮助 HELP .....	44
4.2 输出文件 SPOOL.....	44
4.3 切换到操作系统命令 HOST .....	45
4.4 获取对象结构信息 DESCRIBE.....	46
4.5 定义本地变量 DEFINE 和 COLUMN.....	52
4.5.1 DEFINE .....	52
4.5.2 COLUMN .....	57
4.6 查看执行计划 EXPLAIN.....	59
4.7 设置异常处理方式 WHENEVER .....	60
4.8 查看下一个结果集 MORE.....	60
4.9 显示 SQL 语句或块信息 LIST .....	61
4.10 插入大对象数据 .....	61
4.11 缓存清理 CLEAR .....	61
<b>5 如何在 DISQL 中使用脚本 .....</b>	<b>63</b>
5.1 编写脚本 .....	63
5.2 使用 START 命令运行脚本 .....	63
5.3 使用 EDIT 命令编辑脚本.....	65
5.4 如何在脚本中使用变量 .....	66
5.4.1 脚本带参数值 .....	66
5.4.2 脚本中定义参数值 .....	68
5.4.3 接收用户交互式输入参数值 .....	69
5.5 使用 PROMPT 命令传递信息 .....	69

# 1 功能简介

DIsql 是 DM 数据库的一个命令行客户端工具，用来与 DM 数据库服务器进行交互。

DIsql 是 DM 数据库自带的工具，只要安装了 DM 数据库，就可以在应用菜单和安装目录中找到。

DIsql 识别用户输入，将用户输入的 SQL 语句打包发送给 DM 数据库服务器执行，并接收服务器的执行结果，并按用户的要求将执行结果展示给用户。为了更好地与用户交互和展示执行结果，用户也可以在 DIsql 中执行 DIsql 命令，这些命令由 DIsql 工具自身进行处理，不被发送给数据库服务器。

SQL 语句在 DIsql 中执行完后都被保存在一个特定的内存区域中，用户可以通过上下键查找到这些保存在内存中的 SQL 语句（某些操作系统可能不支持此操作），并可以进行修改，然后再次执行。DIsql 命令执行完后不保存在内存区域中。

表 1.1 SQL 语句和 DIsql 命令的区别

SQL 语句	DIsql 命令
ANSI 标准	DM 内部标准
语言	命令
关键字不可缩写	关键字可缩写
部分语句以分号结束，部分语句以 / 结束	分号可有可无，/ 完全用不到
可以更新表中的数据	不能更新表中的数据

SQL 语句的用法在《DM8\_SQL 语言使用手册》中详细说明。本文档重点介绍 DIsql 命令的使用。

## 2 DIsql 入门

这一章介绍如何启动 DIsql 并成功登录到数据库、如何远程登录到其他数据库、如何使用以及如何退出 DIsql。

### 2.1 启动 DIsql

为了使用 DIsql，必须首先要启动 DIsql。DIsql 工具可以广泛用于各种操作系统，如 WINDOWS、LINUX 等。

启动之后，当出现“SQL>”符号时，用户就可以利用 DM 提供的 SQL 语句和数据库进行交互操作了，需要注意的是，在 DIsql 中 SQL 语句应以分号“;”结束。对于执行语句块，创建触发器、存储过程、函数、包以及模式等时需要用“/”结束。

#### 2.1.1 在 WINDOWS 系统中启动 DIsql

WINDOWS 环境下，有两种启动 DIsql 的方式。第一种是启动安装软件后生成的程序菜单，第二种是启动安装目录下自带的 DIsql 工具。

##### 2.1.1.1 程序菜单启动

如果在 WINDOWS 环境中安装了 DM 数据库产品，那么可以在应用菜单中找到



，直接双击即可启动。然后使用 LOGIN 或 CONN 命令登录到指定数据库。LOGIN 或 CONN 命令下文有详细介绍。

以 LOGIN 为例，登录到 IP 地址为 192.168.1.150 的机器上，用户名和密码为：SYSDBA/SYSDBA，端口号为 5236。其他全部敲回车，采用缺省输入。密码不会回显到屏幕上。



```
SQL> LOGIN
服务名:192.168.1.150
用户名:SYSDBA
密码:
端口号:5236
SSL路径:
SSL密码:
UKEY名称:
UKEY PIN码:
MPP类型:
是否读写分离(y/n):
协议类型:

服务器[192.168.1.150:5236]:处于普通打开状态
登录使用时间      : 3.647(ms)
SQL>
```

图 2.1 菜单启动登录界面

也可以全部直接回车，采用缺省输入，登录到本地 DM 数据库。缺省值请参考下文 LOGIN 命令。

### 2.1.1.2 自带 DIsql 工具启动

DIsql 工具位于 DM 数据库安装目录的 bin 子目录下，例如 DM 数据库的安装目录为 D:\dmdbms，则 DIsql 位于 D:\dmdbms\bin\DIsql.exe。双击启动，然后输入用户名、密码，就可登录到本地 DM 数据库实例。密码不会回显到屏幕上。也可以全部直接回车，采用缺省输入，缺省值为 SYSDBA/SYSDBA。

```
disql V8
用户名:SYSDBA
密码:

服务器[LOCALHOST:5236]:处于普通打开状态
登录使用时间      :6.969(ms)

SQL>
```

图 2.2 自带 DIsql 工具登录界面

如果后续操作想登录到其他 DM 数据库实例，可使用 LOGIN 或 CONN 命令。

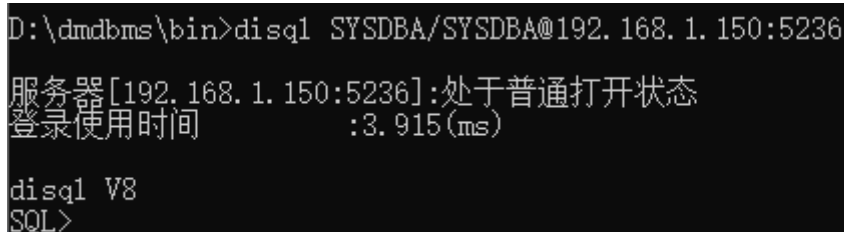
## 2.1.2 命令行启动 DIsql

命令行启动 DIsql 适用于任何操作系统平台。下面以 WINDOWS 系统为例。

### 2.1.2.1 命令行启动

从命令行启动 DIsql 并登录到数据库。在命令行工具中找到 DIsql 所在安装目录 D:\dmdbms\bin，输入 disql 和登录方式后回车。登录方式在下一节详细介绍。

登录界面如下：



```
D:\dmdbms\bin>disql SYSDBA/SYSDBA@192.168.1.150:5236
服务器[192.168.1.150:5236]:处于普通打开状态
登录使用时间:3.915(ms)

disql V8
SQL>
```

图 2.3 命令行启动登录界面

### 2.1.2.2 DIsql 登录方式

DIsql 的登录方式。

语法如下：

---

DIsql 用法 1:disql -h|help 显示 disql 版本信息和帮助信息

DIsql 用法 2:disql [ [<option>] [<logon> |{/NOLOG}] [<start>] ]

<option>::=[-L] [-S]

<logon>::={(<username>[/<password>]) | /}[@<connect\_identifier>][<option>]

[<os\_auth>]

<connect\_identifier> ::=<svc\_name> | {<host>[:<port>]} | <unixsocket\_file>

<option>::= #{ <extend\_option>=<value>{,<extend\_option>=<value>} } //此行外层{ }

是为了封装参数之用，书写时需要保留

<os\_auth>::= AS {SYSDBA|SYSSSO|SYSAUDITOR|USERS|AUTO}

<start>::=<`运行脚本`>|<start 运行脚本>|<直接执行语句>|<直接执行 SET 命令>

<`运行脚本`>::=<`file\_path`> [<PARAMETER\_VALUE>{ <PARAMETER\_VALUE>}]

---

---

<start 运行脚本>::=START <file\_path> [<PARAMETER\_VALUE>{ <PARAMETER\_VALUE>}]

<直接执行语句>::= -E "<SQL 语句>{;<SQL 语句>}"

<直接执行设置 DIsql 属性命令>::= -C "<SET 命令 | COLUMN 命令>"

---

文中语法符号规定：<>内的内容是必选项；

[]内的内容是可选项；{}内的内容可以出现一次或多次；



说明：

|为或者；::=为定义符。后文语法用法与此相同。

-h|help: h 或 help 表示显示 DIsql 版本信息和帮助信息。

[-L] [-S]: -L 表示只尝试登录一次；-S 表示设置 DIsql 界面为隐藏模式，隐藏 <SQL>标识符。

{{<username>[/<password>]} | /}: <username>[/<password>]为用户名和密码。普通登录方式时用户名必写，密码缺省为 SYSDBA。/表示采用操作系统身份验证方式登录或利用 wallet 文件登录。采用操作系统身份验证方式登录时无需指定用户名和密码，即使指定也会被忽略。利用 wallet 文件登录时，dm\_svc.conf 文件中的配置项 WALLET\_LOCATION 必须非空，客户端会通过用户输入的服务名以及 WALLET\_LOCATION 配置项指定的 wallet 文件路径自动获取 wallet 文件中服务名所对应的用户名和密码，因此用户无需输入用户名和密码，若用户输入了用户名和密码，则优先使用用户输入的用户名和密码登录数据库，关于利用 wallet 文件登录数据库的更多详细介绍请参考手册《DM8 安全管理》。如果<password>中含有特殊字符，因为特殊字符在操作系统中需要被特殊处理，因此特殊字符书写的时候需要按照要求的格式。本节末尾会针对包含特殊字符的 <password>如何书写，进行详细介绍。

<svc\_name>: 服务名。服务名在 dm\_svc.conf 中配置。dm\_svc.conf 的配置请参考《DM8 系统管理员手册》。例如：在 dm\_svc.conf 中配置服务名 dmsvc = (192.168.1.150:5236, 192.168.1.150:5237)。然后就可以使用服务名登录了：DIsql SYSDBA/SYSDBA@dmsvc。使用服务名的好处是第一个 IP 连不通，会自动连接下一个。

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

192.168.0.248:8888。<host>如果是 IPv6 的地址，需要用[]指明是 IPv6 地址，例如[fe80::1e6f:65ff:fed1:3724%6]。

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

```
./disql SYSDBA/SYSDBA@/data/sdb/DAMENG/foo.sock#{inet_type=UNIXSOCKET}
```

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

现支持的扩展选项如下：

extend_option	value
mpp_type	MPP 登录属性，此属性的设置对非 MPP 系统没有影响。取值 GLOBAL 和 LOCAL，缺省为 GLOBAL。GLOBAL 表示 MPP 环境下建立的会话为全局会话，对数据库的导入导出操作在所有节点进行；LOCAL 表示 MPP 环境下建立的会话为本地会话，对数据库的导入导出操作只在本地节点进行
inet_type	网络通信协议类型。取值 UDP/TCP/IPC/RDMA/UNIXSOCKET，分别对应 UDP 协议、TCP 协议、IPC（共享内存）、RDMA（远程直接内存访问）、UNIXSOCKET(unix domain socket - IPC)协议。缺省为 TCP
ssl_path	通信加密的 SSL 数字证书路径，缺省为不使用加密。数字证书路径由用户自己创建，将相应的证书需放入该文件夹中。其中服务器证书必须与 dmserver 目录同级，客户端目录可以任意设置。和 ssl_pwd 一起使用。 各用户只能使用自己的 SSL 数字证书，例如 SYSDBA 账户只能使用 \bin\CLIENT_SSL\SYSDBA 下的证书和密码，如果证书没有密码可以用缺省或任意数字代替。 例如： <pre>./disql SYSDBA/SYSDBA@192.168.1.64:5236#{ssl_path=/home/dmdbms/bin/client_ssl/SYSDBA,ssl_pwd=12345}"</pre>
ssl_pwd	通信加密的 SSL 数字证书密码。和 ssl_path 一起使用。缺省为不加密
proxy_client	被代理的用户名。 例如：被代理用户 USER1 的口令和密码为 USER111/USER111。代理用户 USER2 的用户名和密码为 SYSDBA/SYSDBA。 首先，赋予用户 USER2 代理用户 USER1 权限，使用户 USER2 可以认证登录用户 USER1。 <pre>SQL&gt;ALTER USER USER1 GRANT CONNECT THROUGH USER2;</pre> 然后，使用代理用户 USER2 就能登录被代理用户 USER1 的数据库。 <pre>./disql SYSDBA/SYSDBA@192.168.1.64:5236#{inet_type=tcp,proxy_client=USER1}"</pre>

例如，一个包含<option>扩展选项的例子。

```
./disql SYSDBA/SYSDBA@192.168.1.64:5236#{mpp_type=local,inet_type=tcp}"
```

AS <SYSDBA|SYSSSO|SYSAUDITOR|USERS|AUTO>：操作系统身份验证。用户可以通过将操作系统用户加入到操作系统的 dmdba|dmssso|dmauditor 用户组来使用操作系统用户登录数据库，分别对应数据库的 SYSDBA|SYSSSO|SYSAUDITOR 用户。还可以通过将操作系统用户加入到操作系统的 dmusers 用户组来使用操作系统用户登录数据库，对应数据库的同名用户。AUTO 表示按顺序自动匹配数据库用户类型。操作系统身份验证无需输入用户名和密码，若输入用户名和密码将会被忽略。操作系统身份验证仅在 DM 安全版本中才提供支持，详情请参考《DM8 安全管理》。

/NOLOG：表示在未登录 DM 服务器的情况下启动 disql。此时可以进行 DIsql 的显示设置和本地变量操作。如果没有 /NOLOG 选项必须登录服务器，不带参数的时候提示输入用户名和密码，此时的用户名和密码用法参考<logon>。且登录三次失败后退出 DIsql。

**<start>命令中：** <`运行脚本>既可以在 DIsql 启动时使用，也可以在进入 DIsql 界面之后使用。而<start 运行脚本>只能在进入 DIsql 界面之



**注意：** 后才能使用。

<`运行脚本>：`符号运行 sql 脚本文件。如果在 linux 环境下使用，<`运行脚本>外需要加上单引号，里如：`" < file\_path >"'。<file\_path>是运行的 sql 脚本文件的路径，建议使用绝对路径。例如：e:\a.sql 为包含了一条 select \* from dual; 语句的脚本文件。

```
disql SYSDBA/SYSDBA@192.168.1.64:5236 `e:\a.sql
```

或

```
SQL> `e:\a.sql
```

<PARAMETER\_VALUE>：传给<file\_path>脚本文件中本地变量的参数值，将其中的参数内容传给变量&1, &2, &3...以此类推。例如：e:\b.sql 为包含了一条 select name,type from V\$PARAMETER where name='&1'; 语句的脚本文件。

```
SQL> start E:\b.txt CTL_PATH
```

```
SQL> select name,type from V$PARAMETER where name='&1';
```

执行结果如下：

```
原值 1:select name,type from V$PARAMETER where name='&1';
```

```
新值 1:select name,type from V$PARAMETER where name='CTL_PATH';
```

行号	NAME	TYPE
-----		
1	CTL_PATH	READ ONLY

<start 运行脚本>: 使用 START 命令运行 sql 脚本文件。<file\_path>和<PARAMETER\_VALUE>的用法和<运行脚本>相同。例如:

```
SQL> start e:\a.sql
```

<直接执行语句>: 使用-E 参数, 将在运行 DIsql 时直接执行后续的一条或多条 SQL 语句, 查询结果不显示行号、执行时间以及执行号等信息, 且不受-C 参数控制。例如:

```
disql SYSDBA/SYSDBA -E "SELECT TOP 1 * FROM SYSOBJECTS; SELECT TOP 1 * FROM V$CMD_HISTORY"
```

<直接执行设置 DIsql 属性命令>: 使用-C 参数, 将在运行 DIsql 时直接执行后续 SET 命令或 COLUMN 命令。若存在多个-C, 只有最后一个-C 生效。若 SET 命令和 COLUMN 命令在同一个-C 中, 则 COLUMN 命令和写在 COLUMN 命令之后的命令均不生效。例如:

```
disql SYSDBA/SYSDBA -C "SET LONG 1000 PAGESIZE 0"。
```



注意:

<password>比较特殊, 因为<password>中会用到各种特殊字符。特殊字符在操作系统中需要被特殊处理。不同操作系统, 书写方式不同。

<password>中特殊字符的书写规范:

### 1. 不同的操作系统

#### 1) WINDOWS 系统

- DIsql 的关键字符, DIsql 的要求对连接串的特殊字符需要使用双引号括起来 "aaaa/aaaa", 操作系统的要求需要再在最外加双引号和转义 "" "aaaa/aaaa" ""。例如: 用户名为 user01, 密码为 aaaa/aaaa, 那么连接串要写成: disql user01/"" "aaaa/aaaa" ""。

- 空格, 需要使用双引号括起来作为一个整体 (这是操作系统的要求)。例如: 用户名为 user01, 密码为 aaaa aaaa, 那么连接串要写成: disql user01/"aaaa aaaa"。

● 双引号，DIsql 要求对双引号需要使用双引号括起来，同时双引号需要转义  
 "aaaa"aaaa"; 操作系统要求再对双引号转义和最外层加双引号""aaaa""aaaa""。  
 例如：用户名为 user01，密码为 aaaa"aaaa"，那么连接串要写成：disql  
 user01/""aaaa""aaaa""。

## 2) LINUX 系统

LINUX 环境下，密码中的特殊字符处理过程既要考虑操作系统的要求，又要考虑 DIsql 的要求。

首先，操作系统的要求。

bash 的引号设计为：在单引号中，所有的特殊字符都失去其特殊含义；在双引号中，特殊字符包括：美元符(\$)、反引号(`)、转义符(\)、感叹号(!)。  
 如果密码中没有单引号的，只有外面加单引号就可以解决了；如果密码只有单引号，那么可以将单引号用双引号括起来；如果既有单引号又有美元符(\$)、反引号(`)、转义符(\)、感叹号(!)四个特殊字符，那么在特殊字符之前全部加\转义就好了。例如：

'aaaa\aaaa' 传给 disql 为 aaaa\aaaa。

"aaaa'aaaa" 传给 disql 为 aaaa'aaaa。

"aaa'\\$aaaa" 传给 disql 为 aaa'\$aaaa。

其次，在操作系统要求的基础上，增加 DIsql 对关键字和双引号的要求。

● DIsql 的关键字符，DIsql 的要求对连接串的特殊字符需要使用双引号括起来。

例如：密码为 aaaa\aaaa，使用双引号括起来"aaaa\aaaa"，因为此密码中不含有单引号，根据操作系统的要求直接在最外面加单引号。例如：用户名为 user01，密码为 aaaa/aaaa，那么连接串要写成：./disql user01/'"aaaa/aaaa"'

● 双引号，DIsql 要求对双引号需要使用双引号括起来，同时双引号需要转义。例如：密码为 aaa"aaaa，那么根据 DIsql 的要求加双引号同时转义为"aaa""aaaa"，因为没有单引号，根据操作系统的要求直接加单引号。例如：用户名为 user01，密码为 aaa"aaaa，那么连接串要写成：./disql user01/'"aaa""aaaa"'

● 单引号，根据操作系统的要求，只能将单引号放入双引号中。例如：用户名为 user01，密码为 aaaa'aaaa，那么连接串要写成：./disql user01/"aaaa'aaaa"。

● 单引号+操作系统下的特殊字符，根据操作系统的要求，因为单引号只能放在双引号内，同时双引号中还有一些特殊字符不能被识别需要加反斜杠转义。例如：用户名为

user01，密码为 aaa'\$aaaa，使用双引号括起来，同时对\$加反斜杠转义。那么连接串要写成：./disql user01/"aaa'\\$aaaa"。

● 单引号+双引号，根据操作系统的要求，单引号需要放在双引号中，在双引号中表示双引号则使用反斜杠转义双引号。例如：用户名为 user01，密码为 aaa"'aaaa，根据 DIsql 的要求双引号作为特殊字符，需要使用双引号在括起来，同时使用双引号对双引号转义"aaa"'aaaa";同时考虑操作系统的要求，因为含有单引号，只能将整个密码放入双引号中，同时对双引号使用反斜杠转义，那么连接串要写成：./disql user01/"\"aaa\"\"'\aaaa\""。

## 2. 如何转义双引号

- 1) DIsql 的要求使用双引号对双引号内的双引号转义。
- 2) WINDOWS 命令行，使用双引号或者反斜杠对双引号内的双引号转义。
- 3) LINUX 命令行，使用反斜杠对双引号内的双引号转义。

## 2.2 切换登录

用户进入 DIsql 界面后，如果想切换到其他 DM 数据库实例。有两种实现方式：一是使用 LOGIN 命令；二是使用 CONN 命令。登录到远程数据库，必须在服务名处使用 IP 地址或网络服务名。

### 2.2.1 LOGIN /LOGOUT

在 DIsql 界面中，使用 LOGIN/LOGOUT 命令登录/退出远程数据库。

#### 1. LOGIN 登录主库建立会话

直接输入 LOGIN 命令后，屏幕会提示输入登录信息。

```
SQL> LOGIN
服务名:
用户名:
密码:
SSL路径:
SSL密码:
UKEY名称:
UKEY PIN码:
MPP类型:
是否读写分离(y/n):
协议类型:
```

图 2.4 login 登录提示信息



服务名：用于连接上服务器的服务名。此处的服务名是一个统称，有三种选择：数据库服务名、IP 地址[:端口号]、或 UNIXSOCKET 文件路径名。其中，UNIXSOCKET 文件路径名用法例如：/home/te/foo.sock。服务名缺省的情况下，服务器的 IP 地址为 LOCALHOST，LOCALHOST 表示本地服务器，端口号为 5236。

用户名和密码：缺省均为 SYSDBA，密码不回显。

SSL 路径和 SSL 密码：用于服务器通信加密，不加密的用户不用设置，缺省为不设置。

UKEY 名称和 UKEY PIN 码：供使用 UKEY 的用户使用，普通用户不用设置，缺省为不使用。

MPP 类型：参见上一节<MPP\_TYPE>，MPP 类型是 MPP 登录属性，此属性的设置对非 MPP 系统没有影响。此属性的有效值为 GLOBAL 和 LOCAL，缺省为 GLOBAL。

是否读写分离(y/n)：缺省为 n。如果输入 y，会提示：读写分离百分比(0-100)。用户根据需要输入相应的百分比，如果输入的百分比不合法，那就相当于没有设置。

协议类型：缺省为 TCP，可选 TCP|UDP|IPC（共享内存）|RDMA（远程直接内存访问）|UNIXSOCKET。

登录成功后会显示登录时间。

## 2. LOGOUT 从登录主库注销会话

LOGOUT 命令从登录主库注销会话。断开连接而不退出 DIsql。

```
SQL>LOGOUT
```

## 2.2.2CONN[ECT] /DISCONN[ECT]

### 1. CONN[ECT] 连接

在 DIsql 界面中，使用 CONN[ECT] 命令登录远程数据库。

语法如下：

---

```
CONN[ECT] <logon>
```

```
<logon>::={{<username>[/<password>]} | /}[@<connect_identifier>][<option>]
```

```
[<os_auth>]
```

```
<connect_identifier> ::=<svc_name> | {<host>[:<port>]} | <unixsocket_file>
```

```
<option>::= #{ <exetend_option>=<value>{,<extend_option>=<value>} } //此行外层
```

```
{ }是为了封装参数之用，书写时需要保留
```

---

---

```
<os_auth>::= AS {SYSDBA|SYSSSO|SYSAUDITOR|USERS|AUTO}
```

---

<logon>中更详细的参数介绍请参考 [2.1.2.2 DIsql 登录方式](#)。



注意:

使用 **CONN[ECT]** 命令建立新会话时，会自动断开先前会话。

示例如下:

```
SQL>CONN SYSDBA/SYSDBA@192.168.1.150
```

## 2. DISCONN[ECT] 断开连接

DISCONN[ECT]: 断开连接而不退出 DIsql。与 logout 功能一样。

```
SQL>DISCONN
```

## 2.3 使用 DIsql

以一个简单的查询例子来说明如何使用 DIsql。只需要输入一条 SQL 语句，回车即可。DIsql 将 SQL 语句发送给 DM 数据库服务器并显示服务器返回的结果。SQL 语句如何书写请参考《DM8\_SQL 语言使用手册》。

```
SQL>select top 5 name,id from sysobjects;
```

执行结果如下:

行号	NAME	ID
1	SYSOBJECTS	0
2	SYSINDEXES	1
3	SYSCOLUMNS	2
4	SYSUSER\$	3
5	SYSCONS	4

已用时间: 0.415 (毫秒). 执行号:518.

## 2.4 退出 DIsql

使用 EXIT/QUIT 命令，退出 DIsql。

语法如下：

---

EXIT|QUIT

---

示例如下：

```
SQL>EXIT
```

## 3 DIsql 环境变量设置

使用 SET 命令可以对当前 DIsql 的环境变量进行设置。并通过 SHOW 命令来查看当前系统中环境变量的设置情况。也可以通过配置文件使 DIsql 启动时自动设置一批环境变量。

### 3.1 DIsql 环境变量

DIsql 中的系统环境变量，汇总如下：

表 3.1 DIsql 命令的快速参考

序号	变量名称	属性	用途
1	AUTO[COMMIT]	<ON OFF (缺省值)>	设置自动提交
2	DEFINE	<c (默认的变量前缀是&)   ON (缺省值)   OFF>	定义本地变量
3	ECHO	<ON (缺省值)   OFF>	显示脚本中正在执行的 SQL 语句
4	FEED[BACK]	<6 (缺省值)   n   ON   OFF>	显示当前 SQL 语句查询或修改的行数
5	HEA[DING]	<ON (缺省值)   OFF>	显示列标题
6	LINESHOW	<ON (缺省值)   OFF>	显示行号
7	NEWP[AGE]	<1 (缺省值)   n   NONE>	设置页与页之间的分隔
8	PAGES[IZE]	<14 (缺省值)   n>	设置一页有多少行数
9	TIMING	<ON (缺省值)   OFF>	显示每个 SQL 语句花费的执行时间
10	TIME	<ON OFF (缺省值)>	显示系统的当前时间
11	VER[IFY]	<ON (缺省值)   OFF>	列出环境变量被替换前、后的控制命令文本
12	LONG	<800 (缺省值)   n>	设置大字段类型显示的最大字节数
13	LINESIZE	<screem_length (缺省值, 屏幕宽度)   n>	设置屏幕上一行显示宽度
14	SERVEROUT[PUT]	<ON   OFF (缺省值)> [SIZE <20000 (缺省值)   n>] [FOR[MAT] <WRA[PPED]   WOR[D_WAPPED] (缺省值)   TRU[NCATED]>]	在块中有打印信息时，是否打印，以及打印的格式

## DM8\_DIsql 使用手册

15	SCREENBUFSIZE	<DEFAULT (20M)   n>	设置屏幕缓冲区的长度
16	CHAR_CODE	<GBK   GB18030   UTF8   DEFAULT (缺省值, 操作系统的编码方式)>	设置 SQL 语句的编码方式
17	CURSOR	<STATIC   FORWARDONLY (缺省值)   DEFAULT>	设置 DPI 语句句柄中游标的类型
18	AUTOTRACE	<OFF (缺省值)   NL   INDEX   ON   TRACE   TRACEONLY>	设置执行计划和统计信息的跟踪
19	DESCRIBE	[DEPTH <1 (缺省值)   n   ALL>] [LINE[ NUM] <ON   OFF (缺省值)>] [INDENT <ON (缺省值)   OFF>]	设置 DESCRIBE 的显示方式
20	TRIMS [POOL]	<OFF (缺省值)   ON>	设置 spool 文件中每行的结尾空格
21	LOBCOMPLETE	<OFF (缺省值)   ON>	设置大字段数据是否从服务器全部取出
22	COLSEP	[text]	设置列之间的分割符。缺省为一个空格
23	KEEPDATA	<ON   OFF (缺省值)>	是否为数据对齐进行优化, 或者保持数据的原始格式。ON 不优化, OFF 对齐优化。缺省为 OFF
24	AUTORECONN	<ON (缺省值)   OFF>	是否自动重新连接。ON 是, OFF 否。缺省为 ON
25	NEST_COMMENT	<ON   OFF (缺省值)>	是否支持多行注释嵌套。ON 是, OFF 否。缺省为 OFF
26	NULL_ASNULL	<ON   OFF (缺省值)>	在绑定参数输入时, 是否将输入的 NULL 当作数据库的 null 处理。ON 是, OFF 否。缺省为 OFF
27	CMD_EXEC	<ON (缺省值)   OFF>	是否执行文件中 “/” 命令。ON 是, OFF 否。缺省为 ON
28	CHARDEL	[text]	设置字符串的限定符。缺省为一个空格
29	FLOAT_SHOW	<0 (缺省值)   float_length>	设置 FLOAT、DOUBLE 类型按科学计数法显示的分界长度。缺省为 0, 代表全部按科学计数法显示
30	CONSOLE_PRINT	<ON (缺省值)   OFF>	控制台是否打印查询结果。ON 是, OFF 否。缺省为 ON
31	SQLCODE	<ON   OFF (缺省值)>	控制台是否打印 SQLCODE 返回

			值。ON 是，OFF 否。缺省为 OFF
32	SQL_LINESHOW	<ON (缺省值)   OFF>	输入多行 SQL 语句时，是否打印 SQL 语句的行号。ON 是，OFF 否。缺省为 ON
33	NULL_SHOW	<ON (缺省值)   OFF>	空数据是否显示为 NULL。ON 是，OFF 否。缺省为 ON
34	SQLPROMPT	<text>	设置 DIsql 的命令行前缀标识
35	ISQL_MODE	<0 (缺省值)   1   2>	结果集打印方式。0：DM 数据库的打印方式；1：兼容 ORACLE 的打印格式；2：兼容 GP 的打印格式。缺省为 0
36	WRAP	<ON (缺省值)   OFF>	是否折行打印结果集。ON 是，OFF 否。缺省为 ON 仅在 ISQL_MODE 为 1 时，该参数有效
37	CTRL_INFO	<0 (缺省值)   1   2   3>	设置 DIsql 控制信息
38	RETRY_CONN	<0 (缺省值)   n>	设置 DIsql 尝试自动重连的次数。自动重连时间间隔由 RETRY_CONN_TIME 设置
39	RETRY_CONN_TIME	<0 (缺省值)   n>	用于当 RETRY_CONN 为 n 时，设置 DIsql 自动重连时间间隔。单位：秒

## 3.2 SET 命令用法

SET 命令用于设置 DIsql 系统环境变量。

语法如下：

---

```
SET <system_variable><value>{ <system_variable><value>}
```

---

<system\_variable>：变量名称，参考 3.1 节。

<value>：属性。

可以同时 SET 多个环境变量，如：Set heading on timing on。一旦 SET 之后某个环境变量出错，那么该变量之后的将不再起作用。

### 3.3 用 SET 命令设置环境变量详解

本节详细介绍如何使用 SET 命令对环境变量进行设置。

#### 3.3.1 AUTO[COMMIT]

设置当前 session 是否对修改的数据进行自动提交。

语法如下：

---

```
SET AUTO[COMMIT] <ON|OFF>
```

---

ON：表示打开自动提交，所有执行的 SQL 语句的事务将自动进行提交。

OFF：表示关闭自动提交，所有执行的 SQL 语句的事务将由用户显式提交，为默认设置。

#### 3.3.2 DEFINE

是否使用 DEFINE 定义本地变量。

语法如下：

---

```
SET DEFINE<c (默认的变量前缀是&) | ON (缺省值) | OFF>
```

---

c：表示打开 DEFINE 功能，同时定义前缀变量符号，c 为定义的前缀符号。

ON：表示打开 DEFINE 功能，使用默认前缀符号&。

OFF：表示不使用 DEFINE 功能。

示例如下：

例 打开 DEFINE 功能，并设置#为变量前缀。

```
SQL> SET DEFINE #
```

```
SQL> SELECT #A FROM DUAL;
```

输入 A 的值:1

执行结果如下：

```
原值 1:SELECT #A FROM DUAL;
```

```
新值 1:SELECT 1 FROM DUAL;
```

```
行号      1
```

```
-----
```

```
1          1
```

已用时间: 0.178 (毫秒). 执行号:722.

### 3.3.3 ECHO

在用 START 命令执行一个 SQL 脚本时, 是否显示脚本中正在执行的 SQL 语句。

语法如下:

```
SET ECHO <ON (默认值) | OFF>
```

### 3.3.4 FEED [BACK]

是否显示当前 SQL 语句查询或修改的总行数。

语法如下:

```
SET FEED[BACK] <6 (缺省值) | n | ON | OFF>
```

n: 表示结果大于 n 行时, 才显示结果的总行数。

ON: 打开显示开关, 使用缺省值 6。

OFF: 关闭显示开关。

示例如下:

```

LOCALHOST: 5236 / SYSDBA
SQL> SET FEED 4
SQL> SELECT TOP 5 NAME FROM SYSOBJECTS;
行号      NAME
-----
1      SYSUSERS
2      USER_MVIEWS
3      USER_OBJECTS
4      USER_SOURCE
5      USER_TABLES
5 rows got
已用时间: 0.256<毫秒> . 执行号:31.
SQL>

```

图 3.1 FEED[BACK]用法

### 3.3.5 HEA [DING]

是否显示列标题。



语法如下：

---

```
SET HEA[DING] <ON (缺省值) |OFF>
```

---

当 SET HEADING OFF 时，在每页的上面不显示列标题，而是以空白行代替。

示例如下：

```

C:\. LOCALHOST : 5236 / SYSDBA
SQL> set heading on
SQL> SELECT TOP 5 NAME FROM SYSOBJECTS;
行号      NAME
-----
1      SYSUSERS
2      USER_MVIEWS
3      USER_OBJECTS
4      USER_SOURCE
5      USER_TABLES

5 rows got

已用时间: 0.210<毫秒>. 执行号:33.
SQL>

```

图 3.2 HEA[DING]用法

### 3.3.6 LINESHOW

LINESHOW 设置是否显示行号。

语法如下：

---

```
SET LINESHOW<ON (缺省值) |OFF >;
```

---

默认为每行输出打印行号。

示例如下：

```

C:\. LOCALHOST : 5236 / SYSDBA
SQL> SET LINESHOW on
SQL> SELECT TOP 5 NAME FROM SYSOBJECTS;
行号      NAME
-----
1      SYSUSERS
2      USER_MVIEWS
3      USER_OBJECTS
4      USER_SOURCE
5      USER_TABLES

5 rows got

已用时间: 0.213<毫秒>. 执行号:37.
SQL>

```

图 3.3 LINESHOW 用法

### 3.3.7 NEWP[AGE]

设置页与页之间的分隔。

语法如下：

---

```
SET NEWP[AGE] <1 (缺省值) | n | NONE>
```

---

当 SET NEWPAGE 0 时，在每页的开头有一个换号符，即“^”符号。

当 SET NEWPAGE n 时，在页和页之间隔着 n 个空行。

当 SET NEWPAGE NONE 时，在页和页之间没有任何间隔。

### 3.3.8 PAGES[IZE]

设置一页有多少行数。

语法如下：

---

```
SET PAGES[IZE] <14 (缺省值) | n>
```

---

如果设为 0，则所有的输出内容为一页，并且当 ISQL\_MODE 不等于 2 时，不显示列标题。缺省值为 14。

当 ISQL\_MODE = 1 时，如果 PAGESIZE < NEWPAGE + 2，则不显示列标题。

当 ISQL\_MODE = 2 时，PAGESIZE 取任意值，都显示列标题。

### 3.3.9 TIMING

显示每个 SQL 语句花费的执行时间。

语法如下：

---

```
SET TIMING<ON (缺省值) | OFF>
```

---

### 3.3.10 TIME

显示系统的当前时间。

语法如下：

---

```
SET TIME<ON | OFF (缺省值) >
```

---

### 3.3.11 VER[IFY]

VER[IFY] 是否列出环境变量被替换前、后的控制命令文本。缺省值为 ON，表示列出命令文本。

语法如下：

---

```
SET VER[IFY] < ON (缺省值) | OFF>
```

---

示例如下：

例 1 设置 VER[IFY] 为 OFF，不列出命令文本。

```
SQL> SET VERIFY OFF
SQL> SELECT &A FROM DUAL;
```

输入 A 的值:3

执行结果如下：

```
行号          3
-----
1              3
已用时间: 0.558 (毫秒). 执行号:733.
```

例 2 设置 VER[IFY] 为 ON，列出命令文本。

```
SQL> SET VERIFY ON
SQL> SELECT &A FROM DUAL;
```

输入 A 的值:3

执行结果如下：

```
原值 1:SELECT &A FROM DUAL;
新值 1:SELECT 3 FROM DUAL;
行号          3
-----
1              3
已用时间: 0.127 (毫秒). 执行号:734.
```

### 3.3.12 LONG

设置 BLOB、CLOB、CHAR、VARCHAR、BINARY、VARBINARY、CLASS 等类型一列能显示的最大字节数。

语法如下：

```
SET LONG <800 (缺省值) | n>
```

示例如下：

```
SQL> CREATE TABLE TEST(C1 TEXT);
```

```
SQL> INSERT INTO TEST VALUES('DIsql 是 DM 数据库的一个命令行客户端工具，用来与 DM 数据库服务器进行交互。');
```

例 1 设置能显示的最大字节数为 10。

```
SQL> SET LONG 10
```

```
SQL> SELECT * FROM TEST;
```

执行结果如下：

[70004]:字符串截断。

行号	C1
----	----

-----

1	DIsql 是 D
---	-----------

已用时间：0.237(毫秒)。执行号：55109。

例 2 设置能显示的最大字节数为 100。

```
SQL> SET LONG 100
```

```
SQL> SELECT * FROM TEST;
```

执行结果如下：

行号	C1
----	----

-----

1	DIsql 是 DM 数据库的一个命令行客户端工具，用来与 DM 数据库服务器进行交互。
---	--

已用时间：0.230(毫秒)。执行号：55110。

### 3.3.13 LINESIZE

设置屏幕上一行显示宽度。

语法如下：

---

```
SET LINESIZE <screen_length (缺省值, 屏幕宽度) |n>
```

---

当 ISQL\_MODE 取值为 0 或 1 时, LINESIZE 取值范围为自 1 至 32767 的整数。

当 ISQL\_MODE 取值为 2 时, LINESIZE 取值范围为自 0 至 32767 的整数, 取值为 0 时, 表示一行数据在同一行显示。

### 3.3.14 SERVEROUT [PUT]

在块中有打印信息时, 是否打印, 以及打印的格式。设置之后, 可以使用 DBMS\_OUTPUT 包打印 (认为 DBMS\_OUTPUT 包已经创建)。

语法如下：

---

```
SET SERVEROUT[PUT] <ON | OFF (缺省值)> [SIZE <20000 (缺省值) |n>]
[FOR[MAT] <WRA[PPED] | WOR[D_WRAPPED] (缺省值) | TRU[NCATED]>]
```

---

ON/OFF: 是否打印。

SIZE: 打印的最大长度。

WORD\_WRAPPED: 按照单词分隔。

TRUNCATED: 单词被截断。

FORMAT: 按照服务器返回的显示, 不做格式化。

示例如下：

对比两个结果, 不难发现, 第二个结果中的单词被截断。

例 1 设置为按照单词分隔, 显示宽度为 20。

```
SQL> SET SERVEROUTPUT ON FORMAT WORD_WRAPPED

SQL> SET LINESIZE 20

SQL>BEGIN

    DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');

    DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');

END;
```

```
/
```

执行结果如下：

```
If there is nothing  
left to do  
shall we continue with  
plan B?
```

PL/SQL 过程已成功完成

已用时间：1.140(毫秒)．执行号：729．

例 2 设置为单词被截断，显示宽度为 20。

```
SQL> SET SERVEROUTPUT ON FORMAT TRUNCATED
```

```
SQL> SET LINESIZE 20
```

```
SQL>BEGIN
```

```
2          DBMS_OUTPUT.PUT_LINE('If there is nothing left to do');
```

```
3          DBMS_OUTPUT.PUT_LINE('shall we continue with plan B?');
```

```
4      END;
```

```
5      /
```

执行结果如下：

```
If there is nothing
```

```
shall we continue wi
```

PL/SQL 过程已成功完成

已用时间：0.330(毫秒)．执行号：731．

### 3.3.15 SCREENBUFSIZE

设置屏幕缓冲区的长度。用来存储屏幕上显示的内容。单位为字节。

语法如下：

---

```
SET SCREENBUFSIZE<DEFAULT (20M) | n>
```

---

n：有效值范围 1~50M。

### 3.3.16 CHAR\_CODE

设置 SQL 语句的编码方式。

语法如下：

---

```
SET CHAR_CODE <GBK | UTF8 | DEFAULT (缺省值, 操作系统的编码方式)>
```

---

### 3.3.17 CURSOR

设置 DPI 语句句柄中游标的类型。

语法如下：

---

```
SET CURSOR <STATIC | FORWARDONLY (缺省值) | DEFAULT>
```

---

### 3.3.18 AUTOTRACE

设置执行计划和统计信息的跟踪。

语法如下：

---

```
SET AUTOTRACE <OFF (缺省值) | NL | INDEX | ON | TRACE | TRACEONLY>
```

---

当 SET AUTOTRACE OFF 时，停止 AUTOTRACE 功能，常规执行语句。

当 SET AUTOTRACE NL 时，开启 AUTOTRACE 功能，不执行语句，如果执行计划中有嵌套循环操作，那么打印 NL 操作符的内容。

当 SET AUTOTRACE INDEX (或者 ON) 时，开启 AUTOTRACE 功能，不执行语句，如果有表扫描，那么打印执行计划中表扫描的方式、表名和索引。

当 SET AUTOTRACE TRACE 时，开启 AUTOTRACE 功能，执行语句，打印执行计划。此功能与服务器 EXPLAIN 语句的区别在于，EXPLAIN 只生成执行计划，并不会真正执行 SQL 语句，因此产生的执行计划有可能不准。而 TRACE 获得的执行计划，是服务器实际执行的计划。

当 SET AUTOTRACE TRACEONLY 时，开启 AUTOTRACE 功能，执行语句，打印执行计划。此功能与 TRACE 区别在于对于查询语句集不打印结果集。

### 3.3.19 DESCRIBE

设置 DESCRIBE 对象结构信息的显示方式。

语法如下：

---

```
SET DESCRIBE [DEPTH <1 (缺省值) | n | ALL>] [LINE[NUM] <ON | OFF (缺省值)>]  
[INDENT <ON (缺省值) | OFF>]
```

---

DEPTH: 结构信息显示至第 N 层, 默认只显示第 1 层。取值范围是 1~40。当设置为 ALL 时, DEPTH 为 40。

LINENUM: 是否显示对象行号, 成员显示父亲的行号。

INDENT: 当对象的类型是复合类型时, 是否通过缩进的方式显示成员信息。

### 3.3.20 TRIMS[POOL]

设置 TRIMS[POOL], 和 SPOOL 功能结合使用。

语法如下：

---

```
SET TRIMS[POOL] <OFF (缺省值) | ON>
```

---

对于 SPOOL 文件, 是否去除输出每行的结尾空格, 缺省为 OFF。

### 3.3.21 LOBCOMPLETE

设置 LOBCOMPLETE, 是否从服务器中全部取出大字段数据。

语法如下：

---

```
SET LOBCOMPLETE <OFF (缺省值) | ON>
```

---

对于大字段数据, 是否从服务器全部取出, 防止死锁的发生; 与显示长度不同, 即便是全部取出, 也可以只显示一部分。

### 3.3.22 COLSEP

设置列之间的分割符。

语法如下：

---

```
SET COLSEP[text]
```

---



如果 text 包含空格或标点符号，请用单引号扩起来。默认为一个空格。

### 3.3.23 KEEPDATA

是否为数据对齐进行优化，或者保持数据的原始格式。

语法如下：

---

```
SET KEEPDATA <ON|OFF (缺省值)>
```

---

OFF：表示为保证数据的对齐格式，DIsql 对服务器传回的字符串数据，将其中的换行符、TAB 都转换为空格。缺省为 OFF。

ON：表示关闭对齐优化。

### 3.3.24 AUTORECONN

是否进行自动重新连接。

语法如下：

---

```
SET AUTORECONN <ON (缺省值) | OFF>
```

---

是否进行自动重新连接，设置为 ON 时，使用上次连接的属性设置进行自动重连，缺省为 ON。

### 3.3.25 NEST\_COMMENT

是否支持多层注释嵌套。DIsql 中支持注释，注释必须由起始符号 “/\*” 开始，由结束符号 “\*/” 结束。注释内容还可以嵌套其他的注释，嵌套的注释也同样需由 “/\*” 开始和 “\*/” 结束。

语法如下：

---

```
SET NEST_COMMENT <ON|OFF (缺省值)>
```

---

ON：是，支持多层注释。

OFF：否，只支持一层注释。缺省为 OFF。

示例如下：

当 SETNEST\_COMMENT ON 时，打开支持多层注释的嵌套。下面这个例子就无法停止，因为有两个注释起始符号，只有一个注释结束符号。所以程序一直处于输入状态。当 SET NEST\_COMMENT OFF 时，只支持一行注释，程序只要找到一组 /\* 和 \*/ 就会结束。

```
SQL>SET NEST_COMMENT OFF
/****** abcd fdddef
/******
*/
CREATE TABLE TEST(C1 INT);
```

### 3.3.26 NULL\_ASNULL

在绑定参数输入时，是否将输入的 NULL 当做数据库的 null 处理。ON 是，OFF 否。缺省为 OFF。

语法如下：

---

```
SET NULL_ASNULL<ON|OFF (缺省值)>
```

---

ON：是。

OFF：否。缺省为 OFF。

示例如下：

```
drop table tm;
create table tm(c1 int,c2 varchar);
insert into tm values(null,'a');
commit;
```

例 1 设置为不将输入的 NULL 当做数据库的 null 处理。

```
SET NULL_ASNULL OFF
insert into tm values(?,?);
请输入参数 1 的值:null
请输入参数 2 的值:null
insert into tm values(?,?);
[-70011]:无效的转换字符串.
```

例 2 设置为将输入的 NULL 当做数据库的 null 处理。

```
SET NULL_ASNULL ON
```

```
insert into tm values(?,?);
```

请输入参数 1 的值:null

请输入参数 2 的值:null

影响行数 1

### 3.3.27 CMD\_EXEC

是否执行 sql 脚本文件中 “/” 命令，ON 是，OFF 否。缺省为 ON。

语法如下：

```
SET CMD_EXEC <ON (缺省值) | OFF>
```

ON：是。缺省为 ON。

OFF：否。

示例如下：

test.sql 脚本位于 d:\目录下。test.sql 内容如下：

```
select 1;
```

```
/
```

```
Commit;
```

例 1 缺省 CMD\_EXEC 为 ON 的情况。

```
SQL> `d:\test.sql
```

```
SQL> select 1;
```

执行结果如下：

```
行号          1
```

```
-----
```

```
1              1
```

已用时间：0.842(毫秒)。执行号:1664。

```
SQL> /
```

执行结果如下：

```
行号          1
```

```
-----
```

1	1
---	---

已用时间：0.832(毫秒)．执行号:1665．

```
SQL> commit;
```

操作已执行

已用时间：1.061(毫秒)．执行号:1666．

#### 例 2 修改 CMD\_EXEC 为 OFF。

```
SQL>SET CMD_EXEC OFF
```

```
SQL> `d:\test.sql
```

```
SQL> select 1;
```

执行结果如下：

行号	1
----	---

```
-----
```

1	1
---	---

已用时间：0.866(毫秒)．执行号:1667．

```
SQL> commit;
```

操作已执行

已用时间：0.800(毫秒)．执行号:1668．

### 3.3.28 CHARDEL

设置字符串的限定符。

语法如下：

```
SET CHARDEL [text]
```

如果 text 包含空格或标点符号，则需要用单引号扩起来。默认为一个空格。

示例如下：

例 1 设置字符串限定符为\$。

```
SQL> SET CHARDEL $
```

```
SQL> SELECT 'aa';
```

执行结果如下：

行号	'aa'
1	\$aa\$

例 2 设置字符串限定符为"。

```
SQL> SET CHARDEL '"';
```

```
SQL> SELECT 'aa';
```

执行结果如下：

行号	'aa'
1	"aa"

### 3.3.29 FLOAT\_SHOW

设置 FLOAT、DOUBLE 类型数据按科学计数法显示的分界长度。

语法如下：

```
SET FLOAT_SHOW <0(缺省值) | float_length>
```

当数据直接打印长度超过 float\_length 时以科学计数法显示，否则直接显示。

float\_length 取值范围为 0~350，为 0 代表总是以科学计数法显示。

示例如下：

```
SQL> DROP TABLE T1;

SQL> CREATE TABLE T1(X FLOAT);

SQL> INSERT INTO T1 VALUES(11.00011);

SQL> COMMIT;
```

例 1 设置数据按科学计数法显示的分界长度为 50。

```
SQL> SET FLOAT_SHOW 50

SQL> SELECT * FROM T1;
```

执行结果如下：

行号	X
----	---

```
-----
```

1	11.000110
---	-----------

例 2 设置为总是以科学计数法显示数据。

```
SQL> SET FLOAT_SHOW 0
SQL> SELECT * FROM T1;
```

执行结果如下：

```
行号      X
-----
```

1	1.1000110000000000E+01
---	------------------------

### 3.3.30 CONSOLE\_PRINT

控制台是否打印查询结果和执行时间，ON 是，OFF 否。缺省为 ON。

语法如下：

```
SET CONSOLE_PRINT <ON(缺省值)|OFF|OFF_WITH_TIME >
```

ON：是，打印执行结果和执行时间。缺省为 ON。

OFF：否，既不打印执行结果，也不打印执行时间。

OFF\_WITH\_TIME：否，不打印执行结果但是打印执行时间。

示例如下：

例 1 缺省 CONSOLE\_PRINT 为 ON，打印查询结果。

```
SQL> select 1;
```

执行结果如下：

```
行号      1
-----
```

1	1
---	---

已用时间：0.440(毫秒)。执行号:1664。

例 2 修改 CONSOLE\_PRINT 为 OFF，不打印查询结果。

```
SQL> set console_print off
SQL> select 1;
```

### 3.3.31 SQLCODE

控制台是否打印 SQLCODE 返回值，ON 是，OFF 否。缺省为 OFF。SQLCODE 返回 0 表示当前 SQL 语句执行成功，SQLCODE 返回 1 表示当前 SQL 语句执行失败。

语法如下：

---

```
SET SQLCODE <ON|OFF(缺省值)>
```

---

ON：是。

OFF：否。缺省为 OFF。

示例如下：

例 1 缺省 SQLCODE 为 OFF，控制台不打印 SQLCODE 返回值。

```
SQL> select 1;
```

执行结果如下：

```
行号      1
```

```
-----
```

```
1          1
```

```
已用时间：2.136(毫秒)．执行号：700．
```

例 2 修改 SQLCODE 为 ON，控制台打印 SQLCODE 返回值。

```
SQL> set sqlcode on
```

```
SQL> select 1;
```

执行结果如下：

```
行号      1
```

```
-----
```

```
1          1
```

```
已用时间：0.402(毫秒)．执行号：701．
```

```
SQLCODE=0: SQL statement executed successfully
```

### 3.3.32 SQL\_LINESHOW

输入多行 SQL 语句时，是否打印 SQL 语句的行号，ON 是，OFF 否。缺省为 ON。

语法如下：

---

```
SET SQL_LINESHOW <ON (缺省值) | OFF>
```

---

ON: 是。缺省为 ON。

OFF: 否。

示例如下:

例 1 缺省 SQL\_LINESHOW 为 ON, 打印 SQL 语句的行号。

```
SQL> select
2   'a'
3   ;
```

执行结果如下:

```
行号      'a'
-----
1         a
```

例 2 修改 SQL\_LINESHOW 为 OFF, 不打印 SQL 语句的行号。

```
SQL> set sql_lineshow off
SQL> select
'a'
;
```

执行结果如下:

```
行号      'a'
-----
1         a
```

### 3.3.33 NULL\_SHOW

空数据是否显示为 NULL, ON 是, OFF 否。缺省为 ON。

语法如下:

---

```
SET NULL_SHOW <ON (缺省值) | OFF>
```

---

ON: 是。缺省为 ON。

OFF: 否。

示例如下:



例 1 缺省 NULL\_SHOW 为 ON，设置空数据显示为 NULL。

```
SQL> select null;
```

执行结果如下：

行号	NULL
-----	
1	NULL

例 2 修改 NULL\_SHOW 为 OFF，设置空数据不显示为 NULL。

```
SQL> set null_show off
```

```
SQL> select null;
```

执行结果如下：

行号	NULL
-----	
1	

### 3.3.34 SQLPROMPT

设置 DIsql 命令行的前缀标识

语法如下：

---

```
SET SQLPROMPT < text >
```

---

<text>为任意的字符串。<text>中可以包含 DEFINE 变量名。缺省则 SQL>为前缀。其中 \_USER 和 \_connect\_identifier 两个系统变量代表“系统用户名”和“IP 地址：端口号”。

示例如下：

```
SQL> set SQLPROMPT "_USER'@'_connect_identifier>"
SYSDBA@LOCALHOST:5236>
SYSDBA@LOCALHOST:5236> set SQLPROMPT "GYF-COMPUTER>"
GYF-COMPUTER>
```

### 3.3.35 ISQL\_MODE

结果集打印方式。

语法如下：

---

```
SET ISQL_MODE <0 (缺省值) | 1 | 2>
```

---

0：DM 数据库的打印方式。缺省为 0。

1：兼容 ORACLE 的打印格式。

2：兼容 GP 的打印格式。

### 3.3.36 WRAP

是否折行打印结果集，ON 是，OFF 否。缺省为 ON。仅在 ISQL\_MODE 为 1 时，该参数有效。

语法如下：

---

```
SET WRAP <ON (缺省值) | OFF>
```

---

ON：是。缺省为 ON。

OFF：否。

示例如下：

设置 ISQL\_MODE 为 1，LINESIZE 为 10，准备数据如下。

```
SQL> set isql_mode 1;

SQL> set linesize 10;

SQL> drop table test;

SQL> create table test(c1 varchar(50), c2 varchar(50));

SQL> insert into test values('damengshujuku', 'damengshujuku');

SQL> commit;
```

例 1 缺省 WRAP 为 ON。查询结果根据设置的 LINESIZE 进行折行显示。

```
SQL> select * from test;
```

执行结果如下：

```
行号      C1
-----
          C2
          -----
```

```

1      damengshuj
      uku
      damengshuj
      uku

```

例 2 设置 WRAP 为 OFF。查询结果不折行显示，若实际数据长度超过设置的 LINESIZE，则将打印结果截断。

```

SQL> set wrap off;
SQL> select * from test;

```

执行结果如下：

```

行号      C1
-----
1      damengshuj
行被截断

```

### 3.3.37 CTRL\_INFO

设置 DIsql 的回显打印信息。

语法如下：

---

```
SET CTRL_INFO <0 (缺省值) | 1 | 2 | 3>
```

---

0：没有特殊设置。缺省为 0。

1：控制 -E 不强制关闭行号、执行时间以及执行号等信息的展示。

2：控制 DIsql 的回显信息按 5 列打印，具体如下：

- ✓ 第 1 列：语句的 SQL 类型，如：DELETE、UPDATE、SELECT 等；
- ✓ 第 2 列：语句的执行结果，执行成功则为 0，执行失败则为错误码；
- ✓ 第 3 列：语句的影响行数，DDL 和 PLsql 类型语句的影响行数均默认为 0，  
仅 DELETE、UPDATE、SELECT 类型语句存在具体的影响行数；
- ✓ 第 4 列：语句的执行时间；
- ✓ 第 5 列：语句执行失败时的错误信息，若语句执行成功，则不打印该列。

设置为 2 时，不再打印执行过程中的报错信息。

3：同时进行 1、2 的设置。

4: 控制语句块多个结果集全部自动显示, 不需要通过 more 命令显示结果集。

示例如下:

例 1 使用 -E 命令直接执行 SQL 语句, 不显示行号、执行时间以及执行号等信息

```
./disql SYSDBA/SYSDBA@192.168.100.163:5254 -E "SELECT * FROM TEST;"
```

执行结果如下:

服务器[192.168.100.163:5254]:处于普通打开状态

登录使用时间 : 2.635 (ms)

disql V8

C1

---

ABC

例 2 利用 -C 命令设置 CTRL\_INFO 为 1, 控制 -E 不强制关闭行号、执行时间以及执行号等信息的展示。

```
./disql SYSDBA/SYSDBA@192.168.100.163:5254 -C "SET CTRL_INFO 1" -E "SELECT * FROM TEST;"
```

执行结果如下:

服务器[192.168.100.163:5254]:处于普通打开状态

登录使用时间 : 2.447 (ms)

disql V8

行号 C1

-----

1 ABC

已用时间: 0.702 (毫秒). 执行号: 56100.

例 3 设置 CTRL\_INFO 为 2, 控制 DIsql 的回显信息按 5 列打印。

```
SQL> SET CTRL_INFO 2
```

```
SQL> SELECT * FROM TEST;
```

执行结果如下:

```

行号      C1
-----
1          ABC
SELECT 0 1 0.925

SQL> SELECT * FROM TEST22;
SELECT -2106 0 0.604 第 1 行附近出现错误：无效的表或视图名[TEST22]

```

### 3.3.38 RETRY\_CONN

设置 DIsql 尝试自动重连的次数。该环境变量主要用于写自动化脚本，当连接失败导致 SQL 语句执行失败时，DIsql 尝试自动重新连接数据库，如果在设置次数内连接成功，则重新执行相应 SQL 语句。需要注意的是，该环境变量与 AUTORECONN 并无关系。

语法如下：

---

```
SET RETRY_CONN <0 (缺省值) | n>
```

---

0：不自动重连。缺省为 0。

n：自动重连的次数。

示例如下：

例 设置 DIsql 尝试自动重连的次数为 1000，如果 DIsql 在 1000 次内连接服务器成功，则重新执行相应 SQL 语句。

```

./disql SYSDBA/SYSDBA@192.168.100.163:5254 -C "SET RETRY_CONN 1000" -E

"select 1;"

[-70019]:网络通讯失败.

disql V8

连接丢失

连接丢失

连接丢失

服务器[192.168.100.163:5254]:处于普通打开状态

已连接

```

1

1

### 3.3.39 RETRY\_CONN\_TIME

用于当 RETRY\_CONN 为 n 时，设置 DIsql 自动重连的时间间隔。

语法如下：

---

```
SET RETRY_CONN_TIME < 0|n >
```

---

0：缺省值，无时间间隔进行重连。

n：间隔 n 秒重连一次。

示例如下：

例 设置 DIsql 尝试自动重连的次数为 20，如果 DIsql 在 20 次内连接服务器成功，则重新执行相应 SQL 语句。自动重连时间间隔为 10s。

```
./disql SYSDBA/SYSDBA -C "SET RETRY_CONN 20 RETRY_CONN_TIME 10" -E " select
1;"
```

```
[-70028]:创建 SOCKET 连接失败.
```

```
disql v8
```

```
连接丢失
```

```
连接丢失
```

```
连接丢失
```

```
服务器[LOCALHOST:5236]:处于普通打开状态
```

```
已连接
```

```
1
```

```
1
```

### 3.4 SHOW 命令查看环境变量

通过使用 SHOW 命令，用户就可以快速而方便的了解到 DIsql 环境的当前环境变量设置和初始化参数。

SHOW 可以显示一个或多个变量。显示多个变量时中间加空格，当其中某一变量出错之后，后面的仍会继续显示。

#### 3.4.1 显示当前环境变量

显示指定的环境变量。

语法如下：

---

```
SHOW <system_variable>{<system_variable>}
```

---

<system\_variable>:环境变量。

示例如下：

例 显示 HEADING 和 TIMING 两个变量。

```
SQL> SHOW HEADING TIMING

HEADING ON.

TIMING ON
```

#### 3.4.2 显示初始化参数

显示所有包含指定字符串的初始化参数。

语法如下：

---

```
SHOW PARAMETER[S] [<parameter_name>]
```

---

<parameter\_name>: 包含在初始化参数名中的字符串。<PARAMETER\_NAME>为空、单引号或双引号时，则显示所有初始化参数。若<PARAMETER\_NAME>为多个字符串，字符串之间可用空格隔开，只有第一个字符串生效，后面的字符串会被忽略。

示例如下：

例 显示包含 SORT 的初始化参数。

```
SQL> SHOW PARAMETERS SORT

行号      PARA_NAME      PARA_VALUE
```

1	SORT_BUF_SIZE	20
2	SORT_BLK_SIZE	1
3	SORT_BUF_GLOBAL_SIZE	1000
4	SORT_FLAG	1
5	SORT_OPT_SIZE	0
6	TSORT_OPT	1
7	BASE_SORT_CPU	36000
8	LARGE_SORT_CPU	40000
9	SMALL_SORT_CPU	200
10	NLS_SORT_TYPE	0
11	SORT_ADAPTIVE_FLAG	1

### 3.5 使用配置文件设置环境变量

DIsql 在连接成功数据库时会自动运行两个配置文件 glogin.sql 和 login.sql。glogin.sql 文件中的设置永久生效，该配置文件需要用户自行创建在 \$DM\_HOME/bin/disql\_conf 路径下，其中 \$DM\_HOME 为 DM 的安装目录，需要用户在操作系统配置 DM\_HOME 环境变量，disql\_conf 目录需要用户自行创建。login.sql 文件为用户自定义配置文件，对其存放路径不做限制。

glogin.sql 文件和 login.sql 文件的执行顺序如下：

1. 默认在 \$DM\_HOME/bin/disql\_conf 路径下查找 glogin.sql 文件并执行；
2. 默认在当前工作目录下（注意不是 DIsql 工具所在的目录，而是启动 DIsql 时所在的目录）查找 login.sql 文件并执行，若未找到则执行步骤 3；
3. 判断操作系统是否配置了 DM\_SQLPATH 环境变量，如果配置了该变量则在对应路径下查找 login.sql 文件并执行。

DIsql 执行上述配置文件时并不会在 DIsql 窗口打印信息，如果没有找到上述配置文件则忽略。

例如，用户配置 DM\_HOME 环境变量后，在 \$DM\_HOME/bin/disql\_conf 目录下创建 glogin.sql 文件，内容如下：



```
SET FEEDBACK ON
```

```
SET TIMING ON
```

另外，用户在当前工作目录下创建了 login.sql 文件，内容如下：

```
SET CTRL_INFO 2
```

DIsql 启动并连接数据库服务器后，上述两个配置文件中的设置立即生效。

## 4 DIsql 常用命令

### 4.1 帮助 HELP

DIsql 帮助命令，可以帮助用户查看其他命令的具体用法。用户可以看到其他命令系统显示的内容，概括为：

- 命令的标题
- 命令的文本描述
- 命令的简写（例如，AUTO 可以代替 AUTOCOMMIT）
- 可以向命令传递的强制参数和可选参数

HELP 显示指定命令的帮助信息。

语法如下：

---

```
HELP|? [topic]
```

---

topic：命令名称或者命令名称的首字母，查询某一命令用法或者某一字母开头的命令用法。

示例如下：

```
SQL> HELP DEFINE
```

```
DEFINE
```

```
-----
```

设置变量值，或者显示已定义的变量信息。

```
DEF[INE] [variable] | [variable = text]
```

### 4.2 输出文件 SPOOL

将查询结果输出到指定文件。

语法如下：

---

```
SPOOL {<file> | OFF }
```

```
<file>: : = <file_path> [CRE[ATE] | REP[LACE] | APP[END]] [NO_PRINT]
```

---

<file\_path>：指定文件的绝对路径

CRE[ATE]: 创建指定的文件，若指定的文件已存在，则报错，默认方式

REP[LACE]: 创建指定的文件，若指定的文件已存在，则替换它

APP[END]: 将输出内容追加到指定文件的末尾

NO\_PRINT: 控制台不打印查询结果

OFF: 关闭 SPOOL 输出



**注意:**

只有 SPOOL OFF 之后，才能在输出文件中看到完整的输出内容。

示例如下:

环境变量 CONSOLE\_PRINT 缺省为 ON，具体请参见 [3.3.30 CONSOLE PRINT](#)。

```
SQL> spool d:\b.sql
```

```
SQL> select 1;
```

```
行号          1
```

```
-----
```

```
1            1
```

```
已用时间: 0.400 (毫秒). 执行号:325.
```

```
SQL> spool off
```

执行上述语句，控制台打印查询结果，并将查询结果输出到指定文件 d:\b.sql 中。

指定 NO\_PRINT。

```
SQL> spool d:\b.sql no_print
```

```
SQL> select 2;
```

```
SQL> spool off
```

执行上述语句，控制台不打印查询结果，只将查询结果输出到指定文件 d:\b.sql 中。

## 4.3 切换到操作系统命令 HOST

使用 HOST 命令可以不用退出 DIsql 就能执行操作系统命令。如果单独执行 host，则能够直接从 DIsql 界面切换到操作系统，之后可使用 EXIT 回到 DIsql 界面。

语法如下：

---

```
HOST [<command>]
```

---

<command>：操作系统命令。

示例如下：

```
SQL>HOST DIR
```

## 4.4 获取对象结构信息 DESCRIBE

获取表或视图、存储过程、函数、包、记录、类的结构描述。

语法如下：

---

```
DESC[RIBE] [<模式名>.]<对象名>;
```

---

各对象获取的内容略有不同：

- 表或视图获取的内容包括列名、列数据类型、列是否可以取空值。
- 函数、过程、类获取的内容包括两类：1) 存储函数/过程名，类型（函数或过程），函数返回值类型；2) 参数名，参数数据类型、参数输入输出属性、参数缺省值。
- 包获取的内容分为两类：1) 包内存储函数/过程名，类型（函数或过程），函数返回值类型；2) 包内参数名，参数数据类型、参数输入输出属性、参数缺省值。
- 记录获取的内容为：参数名，参数数据类型，参数是否可以取空值。

示例如下：

例 1 获取表 sysgrants 的结构描述。

```
SQL> desc sysgrants;
```

执行结果如下：

行号	NAME	TYPEV	NULLABLE
-----			
1	URID	INTEGER	N
2	OBJID	INTEGER	N
3	COLID	INTEGER	N
4	PRIVID	INTEGER	N
5	GRANTOR	INTEGER	N

```
6          GRANTABLE CHAR(1) N
```

```
6 rows got
```

已用时间: 2.408(毫秒). 执行号:753.

例 2 获取存储过程的结构描述。

创建一个存储过程:

```
CREATE OR REPLACE PROCEDURE PROC_1(A IN OUT INT) AS
B INT;
BEGIN
A:=A+B;
EXCEPTION
    WHEN OTHERS THEN NULL;
END;
/
```

获取存储过程 PROC\_1 的结构描述:

```
SQL>describe PROC_1;
```

行号	NAME	TYPEV	IO	DEF	RT_TYPE
1	PROC_1	PROC			
2	A	INTEGER	INOUT		

-----

1 PROC\_1 PROC

2 A INTEGER INOUT

已用时间: 22.679(毫秒). 执行号:757.

例 3 获取包的结构描述

创建数据库表:

```
CREATE TABLE Person(Id INT IDENTITY, Name VARCHAR(100), City VARCHAR(100));
INSERT INTO Person(Name, City) VALUES('Tom','武汉');
INSERT INTO Person(Name, City) VALUES('Jack','北京');
INSERT INTO Person(Name, City) VALUES('Mary','上海');
```

创建包:

```
CREATE OR REPLACE PACKAGE PersonPackage AS
    E_NoPerson EXCEPTION;
    PersonCount INT;
```

```

Pcur CURSOR;

    PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100));

    PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100));

PROCEDURE RemovePerson(Pid INT);

    FUNCTION GetPersonCount RETURN INT;

    PROCEDURE PersonList;

END PersonPackage;

/

```

创建包主体:

```

CREATE OR REPLACE PACKAGE BODY PersonPackage AS

    PROCEDURE AddPerson(Pname VARCHAR(100), Pcity varchar(100) )AS

    BEGIN

        INSERT INTO Person(Name, City) VALUES(Pname, Pcity);

        PersonCount = PersonCount + SQL%ROWCOUNT;

    END AddPerson;

    PROCEDURE RemovePerson(Pname VARCHAR(100), Pcity varchar(100)) AS

    BEGIN

        DELETE FROM Person WHERE NAME LIKE Pname AND City like Pcity;

        PersonCount = PersonCount - SQL%ROWCOUNT;

    END RemovePerson;

    PROCEDURE RemovePerson(Pid INT) AS

    BEGIN

        DELETE FROM Person WHERE Id = Pid;

        PersonCount = PersonCount - SQL%ROWCOUNT;

    END RemovePerson;

    FUNCTION GetPersonCount RETURN INT AS

    BEGIN

        RETURN PersonCount;

    END GetPersonCount;

    PROCEDURE PersonList AS

```

```

DECLARE

V_id INT;

V_name VARCHAR(100);

V_city VARCHAR(100);

BEGIN

IF PersonCount = 0 THEN

    RAISE E_NoPerson;

END IF;

    OPEN Pcur FOR SELECT Id, Name, City FROM Person;
LOOP

    FETCH Pcur INTO V_id,V_name,V_city;

    EXIT WHEN Pcur%NOTFOUND;

    PRINT ('No.' + (cast (V_id as varchar(100))) + ' ' + V_name + '来自' +
V_city );

    END LOOP;

    CLOSE Pcur;

END PersonList;

BEGIN

    SELECT COUNT(*) INTO PersonCount FROM Person;

END PersonPackage;

/

```

获取包 PersonPackage 的结构描述:

```
SQL>describe PersonPackage;
```

执行结果如下:

行号	NAME	TYPEV	IO DEF RT_TYPE
1	ADDPERSON	PROC	
2	PNAME	VARCHAR(100)	IN
3	PCITY	VARCHAR(100)	IN

```

4          REMOVEPERSON    PROC
5          PNAME            VARCHAR(100) IN
6          PCITY            VARCHAR(100) IN
7          REMOVEPERSON    PROC
8          PID              INTEGER      IN
9          GETPERSONCOUNT  FUNC                INTEGER
10         PERSONLIST      PROC

```

10 rows got

已用时间: 23.196(毫秒). 执行号:764.

#### 例 4 使用 DEPTH 显示列的结构信息

```

CREATE TYPE ADDRESS AS OBJECT
(
  STREET  VARCHAR2(20),
  CITY    VARCHAR2(20)
);
/

CREATE TYPE ADDRESS1 AS OBJECT
(
  NO      INT,
  SADDR   ADDRESS
);
/

```

#### 创建表 EMPINFO:

```

CREATE TABLE EMPINFO
(
  LAST_NAME VARCHAR2(30),
  EMPADDR   ADDRESS,
  SMADDR    ADDRESS1,
  JOB_ID    VARCHAR2(20),
  SALARY    NUMBER(7,2)
);

```

#### 设置 DESCRIBE 的显示方式 :

```
SQL>SET DESCRIBE DEPTH 1 LINENUM ON INDENT ON;
```



获取表 EMPINFO 的结构描述如下：

```
SQL>DESC EMPINFO;
```

行号	ID	PID	NAME	TYPEV	NULLABLE
1	1		LAST_NAME	VARCHAR(30)	Y
2	2		EMPADDR	ADDRESS	Y
3	3	2	STREET	VARCHAR(20)	
4	4	2	CITY	VARCHAR(20)	
5	5		SMADDR	ADDRESS1	Y
6	6	5	NO	INTEGER	
7	7	5	SADDR	ADDRESS	
8	8		JOB_ID	VARCHAR(20)	Y
9	9		SALARY	DEC(7, 2)	Y

9 rows got

已用时间：18.325(毫秒)。执行号：768。

设置 DEPTH 为 1 时，表 EMPINFO 的结构信息只显示至第一层。LINENUM 为 ON 时，显示行号 ID、PID 信息；反之，行号 ID、PID 信息不显示；INDENT 为 ON 时，NAME 的显示方式发生了缩进；反之，不发生缩进。

重新设置 DESCRIBE 的显示方式，并获取表 EMPINFO 的结构描述如下：

```
SQL>SET DESCRIBE DEPTH 2 LINENUM ON INDENT ON;
```

```
SQL>DESC EMPINFO;
```

执行结果如下：

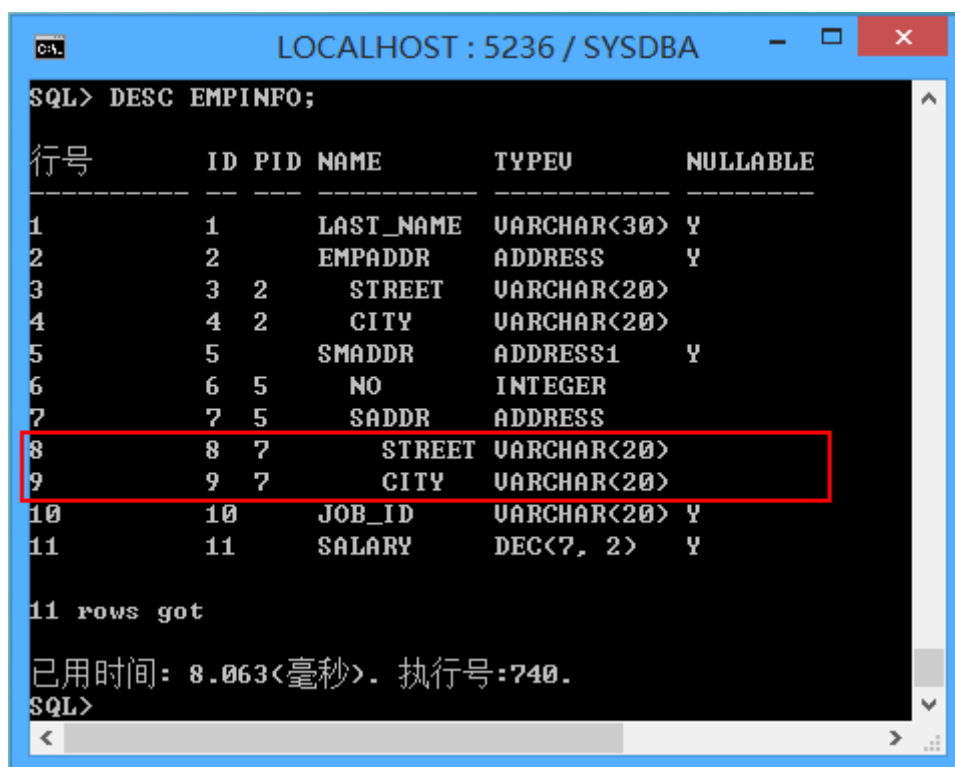


图 4.1 查看 DEPTH 为 2 时显示结果

由上图可见，与 DEPTH 为 1 时对比，表 EMPINFO 的结构描述增加了属于第二层的两列信息：STREET 和 CITY。

## 4.5 定义本地变量 DEFINE 和 COLUMN

定义本地变量的命令有两个：一是 DEFINE；二是 COLUMN。

### 4.5.1 DEFINE

用来定义一个本地变量的替代变量，然后对该变量赋一个 CHAR 类型的值；或者输出变量的值和类型。

语法如下：

---

```
DEF[INE] [<VARIABLE=text>|< VARIABLE >]
```

---

DEF[INE] VARIABLE = text: 申明一个变量，如果该变量存在，则重新赋值，否则新生成一个变量，并进行赋值。

DEF[INE] VARIABLE: 如果该变量存在，则输出特定 VARIABLE 的值和类型，否则报错。

DEF[INE]: 输出 DIsql 中所有的变量的值和类型。

该命令定义的替代变量在当前的 DIsql 环境和/NOLOG 环境中均可以起作用。

当使用该命令定义变量时，如果变量值包含空格或区分大小写，则用引号引注。另外，使用“DEFINE 变量名”可以检查变量是否已经定义。

DEFINE 定义的变量会保存在环境 DIsql 环境中，可以在 SQL 语句中使用。默认的量前缀是&。

示例如下：

```
SQL>DEF VAR=666;
SQL>select * from sysobjects where id=&VAR;
```

如果 var 没有定义，会提示输入变量的值；没有定义的 var 不会保存在 DIsql 环境中。

关闭变量替换：

```
SET DEFINE OFF
```

例 1 Define 变量与其他字符之间的连接字符是点号 ‘.’。

```
SQL>SET DEFINE ON
SQL>DEF VAR1 = C;
SQL>DEF VAR2 = TEST1;
SQL>DROP TABLE TEST1;
SQL>CREATE TABLE TEST1(C1 INT);
SQL>INSERT INTO TEST1 VALUES(1);
SQL>COMMIT;
SQL>SELECT &VAR1.1 FROM TEST1;
```

执行结果如下：

行号	C1
1	1

已用时间：0.428(毫秒)．执行号:702.

```
SQL> select &var2..c1 from test1;
```

执行结果如下：

原值 1:select &var2..c1 from test1;

新值 1:select TEST1.c1 from test1;

行号	C1
----	----

-----

1	1
---	---

已用时间: 0.355(毫秒). 执行号:703.

例 2 DEFINE 变量定义为整型。

SQL>SET DEFINE ON //打开 DEFINE 变量定义

SQL>DEFINE C1=1 //定义变量 C1 为 1

SQL>SELECT &C1 FROM DUAL;

执行结果如下:

原值 1:SELECT &C1 FROM DUAL;

新值 1:SELECT 1 FROM DUAL;

行号	1
----	---

-----

1	1
---	---

已用时间: 0.477(毫秒). 执行号:704.

//在存储函数中的使用

SQL>CREATE OR REPLACE FUNCTION F1(C1 INT)RETURN INT IS

BEGIN

C1=&C1;

RETURN(C1);

END;

/

SQL> SELECT F1(0);

执行结果如下:

行号	F1(0)
----	-------

```
-----
```

1	1
---	---

已用时间：0.355(毫秒)．执行号：707．

```
SQL>DEFINE C2=(2+3*4) //定义变量 C2 为表达式,定义为表达式时必须加括号
```

```
SQL> SELECT &C2*4 FROM DUAL;
```

执行结果如下：

```
原值 1:SELECT &C2*4 FROM DUAL;
```

```
新值 1:SELECT (2+3*4)*4 FROM DUAL;
```

行号	(2+(3*4))*4
----	-------------

```
-----
```

1	56
---	----

已用时间：0.490(毫秒)．执行号：708．

例 3 DEFINE 变量定义为字符型。

```
SQL>SET DEFINE ON //打开 DEFINE 变量定义
```

```
SQL>DEFINE C3="'OG'" //定义变量 C3 为'OG'
```

一种使用 DEFINE 字符串变量的方式

```
SQL> SELECT &C3 FROM DUAL;
```

执行结果如下：

```
原值 1:SELECT &C3 FROM DUAL;
```

```
新值 1:SELECT 'OG' FROM DUAL;
```

行号	'OG'
----	------

```
-----
```

1	OG
---	----

已用时间：0.470(毫秒)．执行号：709．

```
SQL>SELECT LCASE(&C3) FROM DUAL; //引用变量为函数参数
```

执行结果如下：

```
原值 1:SELECT LCASE(&C3) FROM DUAL;
```

```
新值 1:SELECT LCASE('OG') FROM DUAL;
```

```
行号          LCASE('OG')
```

```
-----
```

```
1              og
```

已用时间：14.052(毫秒)．执行号：59．

```
SQL>SET DEFINE ON //打开 DEFINE 变量定义
```

```
SQL>DEFINE C4=OG //定义变量 C4 为 OG
```

另外一种使用 DEFINE 字符串变量的方式

```
SQL>SELECT '&C4' FROM DUAL;
```

执行结果如下：

```
原值 1:SELECT '&C4' FROM DUAL;
```

```
新值 1:SELECT 'OG' FROM DUAL;
```

```
行号          'OG'
```

```
-----
```

```
1              OG
```

已用时间：0.173(毫秒)．执行号：711．

```
SQL>CREATE OR REPLACE FUNCTION F1(&C4 INT)RETURN INT IS
```

```
BEGIN
```

```
&C4=777;
```

```
RETURN(&C4);
```

```
END;
```

```
/
```

```
SQL> SELECT F1(0);
```

执行结果如下：

```
行号          F1(0)
```

```
-----
```

```
1              777                      //返回值 OG=777
```

已用时间：0.460(毫秒)．执行号：713．

例 4 DEFINE 变量定义为日期类型。

```
SQL>SET DEFINE ON //打开 DEFINE 变量定义
```

```
SQL>DEFINE C5="DATE'2015-10-01'"
```

```
SQL>SELECT &C5+1 FROM DUAL; //引用变量值加 1 天
```

执行结果如下：

```
原值 1:SELECT &C5+1 FROM DUAL;
```

```
新值 1:SELECT DATE'2015-10-01'+1 FROM DUAL;
```

```
行号          DATE'2015-10-01'+1
```

```
-----
```

```
1            2015-10-02
```

```
已用时间: 28.478 (毫秒). 执行号:714.
```

```
SQL>SELECT &C5+INTERVAL '01-02' YEAR TO MONTH FROM DUAL; //引用变量值与日期类型作  
运算
```

执行结果如下：

```
原值 1:SELECT &C5+INTERVAL '01-02' YEAR TO MONTH FROM DUAL;
```

```
新值 1:SELECT DATE'2015-10-01'+INTERVAL '01-02' YEAR TO MONTH FROM DUAL;
```

```
行号          DATE'2015-10-01'+INTERVAL+'01-02'YEARTOMONTH
```

```
-----
```

```
1            2016-12-01
```

```
已用时间: 0.482 (毫秒). 执行号:715.
```

## 4.5.2 COLUMN

定义一个本地列或表达式。

语法如下：

```
COL[UMN] [<column | expr> [<option>]]
```

```
<option> ::= NEW_VALUE variable |
```

```
FOR[MAT] <format>
```

```
<format> ::= An | an
```

COL[UMN]：列举出所有的 COLUMN 变量信息。

COL[UMN] column | expr: 列举出某个 column 或 expr, 如果存在, 则输出信息, 否则报错。

COL[UMN] column | expr option: option 目前仅支持 NEW\_VALUE 和 FORMAT。

NEW\_VALUE: 表示该 column|expr 的值同时作为变量 variable 存在。但如果该变量未赋值, 通过 DEFINE 查询时, 不会显示该变量。查询结果的最后一个值赋给变量 variable。

FOR[MAT]: 表示该 column|expr 的列打印长度。‘A’ 或 ‘a’ 是列打印长度的前缀, n 是列打印长度的值, n 的取值范围为 1~60000。仅当 ISQL\_MODE = 1 且 COLUMN <column | expr> 为 ON 时, 该参数有效。定长类型的列的打印长度不受该参数影响。

**示例如下:**

准备数据如下:

```
SQL> DROP TABLE TEST;

SQL> CREATE TABLE TEST(C1 VARCHAR(50));

SQL> INSERT INTO TEST VALUES('DAMENGSHUJUKU');

SQL> INSERT INTO TEST VALUES('A');

SQL> INSERT INTO TEST VALUES('B');

SQL> COMMIT;
```

例 1 将列 C1 的值作为变量 DVAR, 缺省为 COLUMN C1 ON。

```
SQL> COLUMN C1 NEW_VALUE DVAR

SQL> SELECT * FROM TEST;
```

执行结果如下:

```
行号      C1
-----
1          DAMENGSHUJUKU
2           A
3           B

SQL> DEFINE DVAR

DEFINE DVAR      = "B"      (CHAR)
```



例 2 设置 ISQL\_MODE 为 1，C1 列的列打印长度为 8，缺省 WRAP 为 ON，表示折行打印结果集，缺省为 COLUMN C1 ON。

```
SQL> SET ISQL_MODE 1

SQL> COLUMN C1 FORMAT A8

SQL> SELECT * FROM TEST;
```

执行结果如下：

行号	C1
1	DAMENGSH UJUKU
2	A
3	B

通过如下方式设置是否将变量 variable 或列打印长度 format 与 column|expr 相关联，缺省值为 ON，表示已关联。

语法如下：

---

```
COL[UMN] <column | expr> <OFF|ON(默认值)>
```

---

示例如下：

```
SQL>COLUMN C1 OFF
```

## 4.6 查看执行计划 EXPLAIN

用 EXPLAIN 命令来查看查询语句的执行计划。

语法如下：

---

```
EXPLAIN <sql_clause>
```

---

<sql\_clause>请参考《DM8\_SQL 语言使用手册》。

示例如下：

```
SQL>EXPLAIN select count(*) from sysobjects;
```

## 4.7 设置异常处理方式 WHENEVER

用 WHENEVER 命令可以设置异常处理方式，继续执行或退出 DIsql。

语法如下：

---

```
WHENEVER SQLERROR

CONTINUE [ COMMIT | ROLLBACK | NONE ] |

EXIT [ SUCCESS | FAILURE | WARNING | n | <variable> | : <bindvariable> ]

[ COMMIT | ROLLBACK ]
```

---

n 和<variable>的返回值受限于操作系统，在不同平台下，会有所不同，例如：  
UNIX 系统只用一个字节来存 code，所以返回值的范围只在 0~255 之间。

示例如下：

```
SQL>whenever sqlerror exit 1

SQL>select c1 from dual;

select c1 from dual;
```

执行结果如下：

```
第 1 行附近出现错误[-2111]:无效的列名[C1]。

//windows 系统下，输入 echo %ERRORLEVEL%，查看返回值为：1

//linux 系统下，输入 echo $?，查看返回值为：1
```

## 4.8 查看下一个结果集 MORE

当结果集过多，屏幕只能显示一个时，用户可以使用 MORE 命令切换到下一个结果集。

---

MORE | MR

---

例如，当执行如下语句时，用户想查看更多的结果集，可以使用 MORE 命令。

```
begin

select top 10 * from v$dm_ini;

select top 10 * from sysobjects;

select * from dual;

end

/
```

## 4.9 显示 SQL 语句或块信息 LIST

显示最近执行的 SQL 语句或者 PL/SQL 块信息。不显示 DIsql 命令。

语法如下：

---

```
L[IST] [n | n m | n * | n LAST | * | * n | * LAST | LAST] 或者;
```

---

n ,m : 数值 SQL 行号。

\*: 当前行号。

LAST: 最后一行。

## 4.10 插入大对象数据

当插入语句中包含大对象数据文件时，使用@。

---

@<插入语句>

---

<插入语句>，请参考《DM8\_SQL 语言使用手册》，其中大数据的插入值格式为：

---

@'path'

---

示例如下：

例如，在 test 表中插入大对象 e:\DSC\_1663.jpg。

```
create table test(a int,b image);
@insert into test values(1,@'e:\DSC_1663.jpg');
```

## 4.11 缓存清理 CLEAR

清理指定操作本地缓存。

语法如下：

---

```
CL[EAR] <option>
```

---

```
<option> ::= [COL[UMNS] | SQL | SCR[EEN] | BUFF[ER]]
```

---

COL[UMNS]: 清理所有的 COLUMN 变量信息。

SQL: 清理本地 SQL 缓存信息。

SCR[EEN]: 清理 DIsql 终端屏幕信息。

BUFF[ER]: 同 SQL 功能一样，清理本地 SQL 缓存信息。

示例如下：

```
SQL> COLUMN CVAR NEW_VALUE DVAR
```

```
SQL> col
```

```
COLUMN CVAR ON
```

```
NEW_VALUE DVAR
```

```
SQL> cl col
```

```
columns 已清除
```

```
SQL> col
```

```
SQL>
```

## 5 如何在 DIsql 中使用脚本

用户不必在每次使用数据库的时候都编写常用的 SQL 语句和 PL/SQL 程序块，而是可以将它们保存到称为脚本的文件中。这些脚本专门为反复执行的各种任务而设计。本节主要介绍如何编写脚本、运行脚本。

### 5.1 编写脚本

使用一种文本编辑器来编写 SQL 脚本，比如 notepad 文本等。

一个简单的脚本例子，在文本中编写脚本并保存为 D:\test.sql，内容如下：

```
drop table t01;

create table t01(c1 varchar(100), c2 varchar(100));

    begin

        for i in 1..10 loop

            insert into t01 values('a'||i, 'b'||i);

        end loop;

    end;

/
```

### 5.2 使用 START 命令运行脚本

运行脚本必须使用<start>命令。<start>命令中与脚本有关的是<`运行脚本>和<start 运行脚本>。<直接执行语句>在 DIsql 登录时候使用，与脚本无关，此处不做介绍。

语法如下：

---

<start>::=<`运行脚本>|<start 运行脚本>|<直接执行语句>

<`运行脚本>::=`<file\_path> [<PARAMETER\_VALUE>{ <PARAMETER\_VALUE>}]

<start 运行脚本>::=START <file\_path> [<PARAMETER\_VALUE>{ <PARAMETER\_VALUE>}]

<直接执行语句>::= -E "<SQL 语句>{;<SQL 语句>}"

---

<file\_path>: 脚本的路径, 建议使用绝对路径。

<PARAMETER\_VALUE>: 传递进入脚本的参数值。

脚本可以在启动 DIsql 时就运行, 或者在进入 DIsql 之后再运行。如果在启动时运行, 只能使用<`运行脚本>; 如果在进入 DIsql 之后, 使用<`运行脚本>或者<start 运行脚本>来运行脚本都可以。



注意:

<`运行脚本>中的符号`位于键盘第二排左起第一个。

如果在 LINUX 环境下使用<`运行脚本>, 则符号`需要加\或'进行转义。

例如: `./disql SYSDBA/SYSDBA \ `/dev/test.sql`

### 1. 启动 DIsql 时, 运行脚本。

```
disql SYSDBA/SYSDBA `D:\test.sql
```

执行结果如下:

```
D:\dmdbms\bin>disql SYSDBA/SYSDBA `D:\test.sql
服务器[LOCALHOST:5236]:处于普通打开状态
登录使用时间          :17.928(ms)

disql V8
SQL> drop table t01;
操作已执行
已用时间: 12.322(毫秒). 执行号:512.
SQL> create table t01(c1 varchar(100), c2 varchar(100));
操作已执行
已用时间: 1.400(毫秒). 执行号:513.
SQL> begin
for i in 1..10 loop
insert into t01 values('a' ||i, 'b' ||i);
end loop;
end;
DMSQL 过程已成功完成
已用时间: 0.303(毫秒). 执行号:514.
SQL>
```

图 5.1 启动 DIsql 时运行脚本结果

### 2. 进入 DIsql 之后, 运行脚本。

```
SQL>start D:\test.sql
```

或

```
SQL>`D:\test.sql
```

执行结果如下:

```

SQL> start d:\test.sql
SQL> drop table t01;
操作已执行
已用时间: 11.738(毫秒). 执行号:515.
SQL> create table t01(c1 varchar(100), c2 varchar(100));
操作已执行
已用时间: 1.350(毫秒). 执行号:516.
SQL> begin
for i in 1..10 loop
insert into t01 values('a' ||i, 'b' ||i);
end loop;
end;
DMSQL 过程已成功完成
已用时间: 0.293(毫秒). 执行号:517.
SQL>

```

图 5.2 进入 DIsql 后运行脚本结果



注意:

DISQL 在运行完脚本后会自动执行一个提交动作

## 5.3 使用 EDIT 命令编辑脚本

DIsql 中使用 EDIT 命令来编辑指定的脚本文件。

语法如下:

---

```
ED[IT] [<file_name>]
```

---

<file\_name>: 指定待编辑的脚本文件。

如果指定文件不存在, 则创建该文件。

如果省略文件<file\_name>, 则只会修改缓冲区中的最后一条 SQL 语句。DIsql 自动打开系统缺省的文本编辑器 (WINDOWS 下使用 notepad), 复制缓冲区中最后一条 SQL 语句到文本中, 这时用户可以对其中的内容进行编辑。修改完成之前, DIsql 一直处于等待状态。修改完毕, 保存文件后, 被修改的内容就会被写入缓存区。这对于修改错误命令很方便。当操作系统为 LINUX 或 UNIX 时, 使用 vi 进行编辑。

示例如下:

```
SQL>EDIT D:\test.sql
```

或

```
SQL>edit
```

## 5.4 如何在脚本中使用变量

替换变量主要用来进行 SQL、PLSQL 与用户的交互，可以运行时输入，也可提前输入。

替换变量前带有一个前缀标志符（默认是&），DIsql 在命令中遇到替换变量时，用真实值去代替，相当于 c 语言中的宏定义。真实值来源于三个地方：

1. 脚本参数带入
2. 脚本中直接定义
3. 用户动态输入

DIsql 中根据 SET DEFINE 命令开启本地变量功能并定义变量前缀符号。默认符号&作为变量的前缀。详细用法请查看 DEFINE 命令。

### 5.4.1 脚本带参数值

脚本带参数值，参数名必须是数字。

#### 5.4.1.1 变量名是数字

在脚本中通过&n 来引用参数，n 为 1 表示为第一个参数，2 表示第二个参数，依次类推。如现有表 test，其建表和初始化数据语句如下：

```
create table test(id int) ;  
insert into test values(11) ;  
insert into test values(12) ;  
insert into test values(15) ;
```

脚本 D:\test.sql 如下：

```
select * from test where id = &1;  
select * from test where id = &2;  
select * from test where id = &3;
```

DIsql 要求传入的参数值个数要与脚本中的变量个数一一对应。比如脚本 D:\test.sql 中有三个变量&1、&2、&3，则要求传入的参数值也必须是三个。如果传入参数值个数不匹配，如 n 为 3，但执行时只带了 2 个参数，DIsql 就会在屏幕上提示输入参数。



例 输入三个参数值 11、12、13：

```
SQL>`D:\test.sql 11 12 13
```

执行结果如下：

```
SQL> `D:\test.sql 11 12 13
SQL> select * from test where id = &1;
原值 1:select * from test where id = &1;
新值 1:select * from test where id = 11;

行号      ID
-----
1         11

已用时间: 0.456(毫秒). 执行号:534.
SQL> select * from test where id = &2;
原值 1:select * from test where id = &2;
新值 1:select * from test where id = 12;

行号      ID
-----
1         12

已用时间: 0.229(毫秒). 执行号:535.
SQL> select * from test where id = &3;
原值 1:select * from test where id = &3;
新值 1:select * from test where id = 13;
未选定行

已用时间: 0.217(毫秒). 执行号:536.
SQL>
```

图 5.3 脚本带参数用法

#### 5.4.1.2 参数书写要求

因为参数是原样替换，因此如果 SQL 语句中字符串要求用单引号，那么定义的参数值也应该包含单引号；另外如果字符串中有特殊字符，需要使用双引号将整个字符串作为一个整体，需要注意的是，如果作为整体的字符串中有双引号作为内容，需要将内容的双引号转义。

如果参数值是数字，写法没有特殊要求。

如果参数值是字符串，应该用单引号扩起，如果字符串有空格，应该在单引号外面，再加上一个双引号扩起。如脚本 D:\test.sql：

```
create table test(a varchar);

insert into test values('hello');

insert into test values('hello world');
```

//脚本 D:\test.sql 内容如下:

```
select * from test where a = &1;
```

```
select* from test where a = &2;
```

注意参数的写法, 执行语句如下:

```
SQL>`D:\test.sql 'hello' "'hello world'"
```

执行结果如下:

```
SQL> `D:\test.sql 'hello' "'hello world'"
SQL> create table test(a varchar);
操作已执行
已用时间: 1.446(毫秒). 执行号:538.
SQL> insert into test values('hello');
影响行数 1

已用时间: 0.190(毫秒). 执行号:539.
SQL> insert into test values('hello world');
影响行数 1

已用时间: 0.132(毫秒). 执行号:540.
SQL> select * from test where a = &1;
原值 1:select * from test where a = &1;
新值 1:select * from test where a = 'hello';

行号      A
-----
1         hello

已用时间: 0.328(毫秒). 执行号:541.
SQL> select* from test where a = &2;
原值 1:select* from test where a = &2;
新值 1:select* from test where a = 'hello world';

行号      A
-----
1         hello world

已用时间: 0.186(毫秒). 执行号:542.
SQL>
```

图 5.4 字符串参数书写用法

## 5.4.2 脚本中定义参数值

使用 DEFINE 命令定义变量值, 格式: DEFINE 标识符 = 值。

如脚本 D:\test.sql:

```
define n=1

define s=DIsql

select &n from dual;
```

```
select '&s' from dual;
```

如果变量没有定义，那么在通过&引用时，DIsql 会提示输入。

### 5.4.3 接收用户交互式输入参数值

很多时候，在执行脚本时，我们希望有些信息根据脚本的提示，让用户动态输入。这种情况非常好实现，满足下面两个条件即可。

1. 运行脚本时不带参数
2. 脚本中不定义参数

如脚本 D:\test.sql:

```
select &x from dual;
```

执行结果如下：

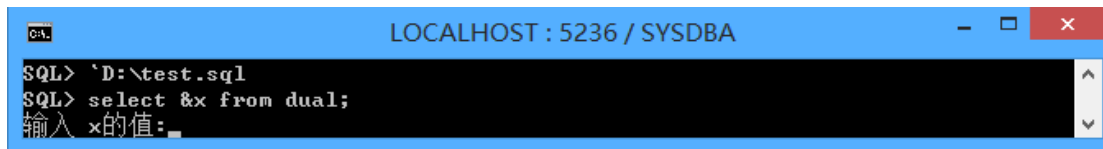


图 5.5 交互式输入参数值用法

## 5.5 使用 PROMPT 命令传递信息

PROMPT 命令会在屏幕上输出一行信息。这非常有助于在存储脚本中向用户传送信息。

语法如下：

```
PROMPT <输出内容>
```

例如，编写一个查询，要提供用户看到数据的纯文本描述信息。用户就可以使用 PROMPT 命令完成这项工作。将如下脚本存储到名为 prompt.sql 的文件中：

```
prompt 部分 ini 参数和 dminit 建库参数信息（系统值、最小值和最大值）；  
select top 3 * from v$dm_ini;
```

执行脚本：

```
SQL> `f:\prompt.sql
```

执行结果如下：

```
SQL> `f:\prompt.sql  
SQL> prompt 部分 ini 参数和 dminit 建库参数信息（系统值、最小值和最大值）；
```

部分 ini 参数和 dminit 建库参数信息（系统值、最小值和最大值）

```
SQL> select top 3 * from v$dm_ini;
```

行号	PARAMETER	VALUE	MIN_VALUE	MAX_VALUE
	MPP_CHK_SESS_VALUE			
	FILE_VALUE	DESCRIPTION		PARAMETER_TYPE
1	CTL_PATH	D:\dmdbms\data\DAMENG\dm.ctl	NULL	NULL
	D:\dmdbms\data\DAMENG\dm.ctl			
	D:\dmdbms\data\DAMENG\dm.ctl	path of dm.ctl		READ ONLY
2	CTL_BAK_PATH	D:\dmdbms\data\DAMENG\ctl_bak	NULL	NULL
	D:\dmdbms\data\DAMENG\ctl_bak			
	D:\dmdbms\data\DAMENG\ctl_bak	backup path of dm.ctl		READ ONLY
3	CTL_BAK_NUM	10	1	100
	10			
	10	backup num of dm.ctl		SYS

已用时间：3.931(毫秒) . 执行号:29.

咨询热线：400-991-6599

技术支持：dmtech@dameng.com

官网网址：www.dameng.com



**武汉达梦数据库股份有限公司**  
**Wuhan Dameng Database Co.,Ltd.**

地址：武汉市东湖新技术开发区高新大道999号未来科技大厦C3栋16—19层

16th-19th Floor, Future Tech Building C3, No.999 Gaoxin Road, Donghu New Tech Development Zone,Wuhan,Hubei Province,China

电话：(+86) 027-87588000 传真：(+86) 027-87588810

---