



达梦数据库

DM7_DIsql 使用手册

达梦技术手册

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 DIsq1，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
 警告：	表示可能导致系统损坏、数据丢失或不可预知的结果。
 注意：	表示可能导致性能降低、服务不可用。
 小窍门：	可以帮助您解决某个问题或节省您的时间。
 说明：	表示正文的附加信息，是对正文的强调和补充。

在本文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 www.dameng.com 阅读或下载 DM 的各种相关文档。

联系我们

如果您有任何疑问或是想了解达梦数据库的最新动态消息，请联系我们：

网址：www.dameng.com

技术服务电话：400-991-6599

技术服务邮箱：dmtch@dameng.com

目录

1 功能简介	1
2 DISQL 入门	2
2.1 启动 DISQL.....	2
2.2 切换登录	8
2.3 使用 DISQL.....	11
2.4 退出 DISQL.....	11
3 DISQL 环境变量设置	12
3.1 DISQL 环境变量	12
3.2 SET 命令用法	13
3.3 用 SET 命令设置环境变量详解	14
3.4 SHOW 命令查看环境变量	26
4 DISQL 常用命令	27
4.1 帮助 HELP	27
4.2 输出文件 SPOOL.....	27
4.3 切换到操作系统命令 HOST	28
4.4 获取对象结构信息 DESCRIBE.....	28
4.5 定义本地变量 DEFINE 和 COLUMN.....	34
4.6 查看执行计划 EXPLAIN.....	40
4.7 设置异常处理方式 WHENEVER	40
4.8 查看下一个结果集 MORE.....	41
4.9 显示 SQL 语句或块信息 LIST	41
4.10 插入大对象数据	42
4.11 缓存清理 CLEAR	42
5 如何在 DISQL 中使用脚本	43
5.1 编写脚本	43
5.2 使用 START 命令运行脚本	43
5.3 使用 EDIT 命令编辑脚本	45

5.4 如何在脚本中使用变量	45
5.5 使用 PROMPT 命令传递信息	49

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，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《DM7_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，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 192.168.0.38 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。

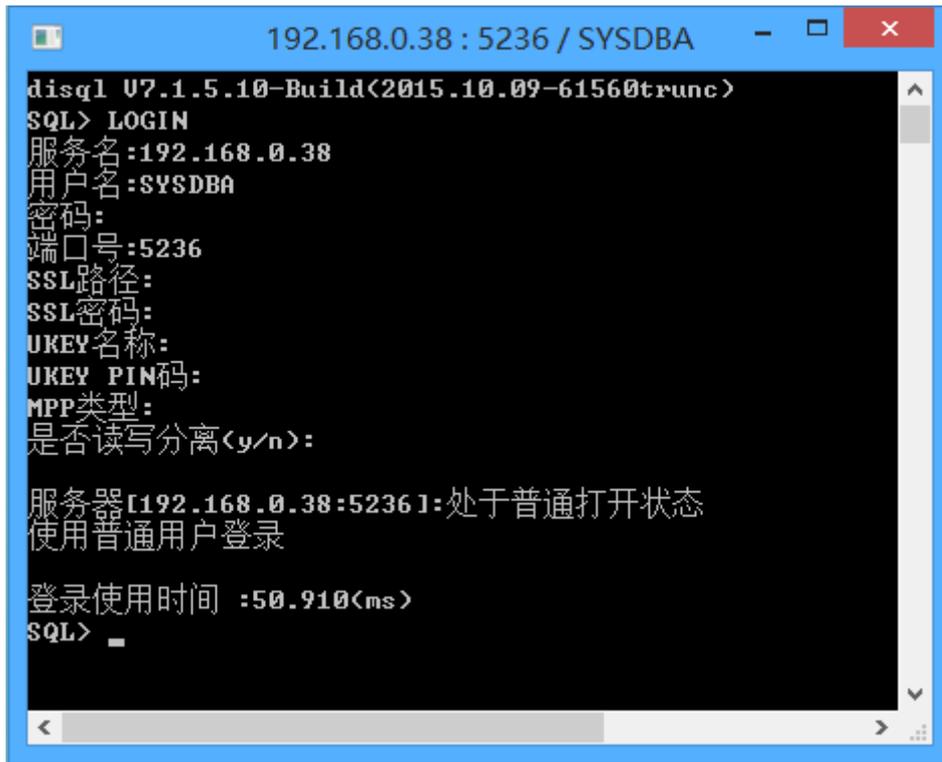


图 2.1 菜单启动登录界面

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。



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：



图 2.3 命令行启动登录界面

2.1.2.2 DIsq1 登录方式

DIsq1 的登录方式。

语法如下：

```
DIsq1 [ [<option>] [<logon> | /NOLOG] [<start>] ]
<option>::=HELP|-H|-S
<logon>::=<username>[/<password>][*<MPP_TYPE>][@<server>][:<port>][?{UDP|TCP}
][#<sslpath>@ssl_pwd]
<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>}"
```



说明：

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
|为或者；::=为定义符。后文语法用法与此相同。

表 2.1 DIsq1 登录方式参数介绍

参数	介绍
HELP -H -S	HELP 或-H 表示显示 DIsq1 版本信息和帮助信息；-S 表示设置 DIsq1 界面为隐藏模式，隐藏命令的<SQL>标识符。
<username>	指定数据库的用户名。缺省用户名为 SYSDBA。
<password>	指定数据库的密码。缺省密码为 SYSDBA。 输入密码时，如遇到特殊字符需要特别处理。特殊字符包括关键字符和双引号等。特殊字符的处理见下文。
<MPP_TYPE>	指 MPP 类型。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，默认为 GLOBAL。 GLOBAL 表示 MPP 环境下建立的会话为全局会话，对数据库的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，对数据库的操作只在本地节点进行。
<server>	指定服务器的 IP 地址或是在 dm_svc.conf 中配置的网络服务名。dm_svc.conf 的配置请参考《DM7 系统管理员手册》的 2.1.1 节。 例如：在 dm_svc.conf 中配置服务名 <u>dmrac_svc =(192.168.0.38:5236, 192.168.0.38:5237)。</u>

	<p>然后就可以使用服务名登录了：</p> <pre>DIsql SYSDBA/SYSDBA@dmrac_svc</pre> <p>使用服务名的好处是第一个 IP 连不通，会自动连接下一个。</p>
UDP TCP	<p>指定使用 UDP 协议或 TCP 协议。缺省为 TCP。</p> <p>例如：DIsql SYSDBA/SYSDBA@localhost:5236?UDP</p>
[#<sslpath>@ssl _pwd]	<p>通信加密中客户端证书存放的地址和客户端证书密钥。各用户只能使用自己的证书，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。</p> <p>例如：DIsql SYSDBA/SYSDBA@192.168.0.38:5236#D:\dmdbms\bin\client_ssl\SYSDBA@12345</p> <p>缺省为不加密。</p>
/NOLOG	<p>表示启动 DIsql 而不登录到服务器。此时可以进行 DIsql 的显示设置和本地变量操作。</p> <p>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。</p>
<start>	<p>运行 DIsql 脚本文件。</p> <p>例如，假设 a.sql 是路径为 “c:\” 的任意脚本文件：DIsql -S SYSDBA/SYSDBA@192.168.0.80:5236 `c:\a.sql`。</p> <p>如果在 linux 环境下使用，<start>外需要加上单引号，如：`'` <file_path >'`。</p>
<file_path>	运行 DIsql 脚本文件的绝对路径。
<PARAMETER_VALU E>	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了 &1, &2, &3...以此类推。
<直接执行语句>	<p>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。例如：</p> <pre>DIsql SYSDBA/SYSDBA -e "SELECT TOP 1 * FROM SYSOBJECTS; SELECT TOP 1 * FROM V\$CMD_HISTORY"。</pre>

<password>中特殊字符的处理方法，不同操作系统，处理方法不同。

1. 不同操作系统

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：`DIsq1 user01/"aaaa aaaa"`
- 双引号，DIsq1 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：`DIsq1 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 DIsq1 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，应该都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 DIsq1 对关键字和双引号的要求。

- DIsq1 的关键字符，DIsq1 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：`./DIsq1 user01/'"aaaa/aaaa"'`。
- 双引号，DIsq1 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 DIsq1 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：`./DIsq1 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： `./DIsq1 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： `./DIsq1 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 DIsq1 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： `./DIsq1 user01/"\"aaa\"\"'aaaa\""`。

2. 如何转义双引号

- 1) DIsq1 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。



<start>命令中： `<`运行脚本>`既可以在 DIsq1 启动时使用，也可以在进入 DIsq1 界面之后使用。而 `<start 运行脚本>`只能在进入 DIsq1 界面之后才能使用。

2.2 切换登录

用户进入 DIsq1 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

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。

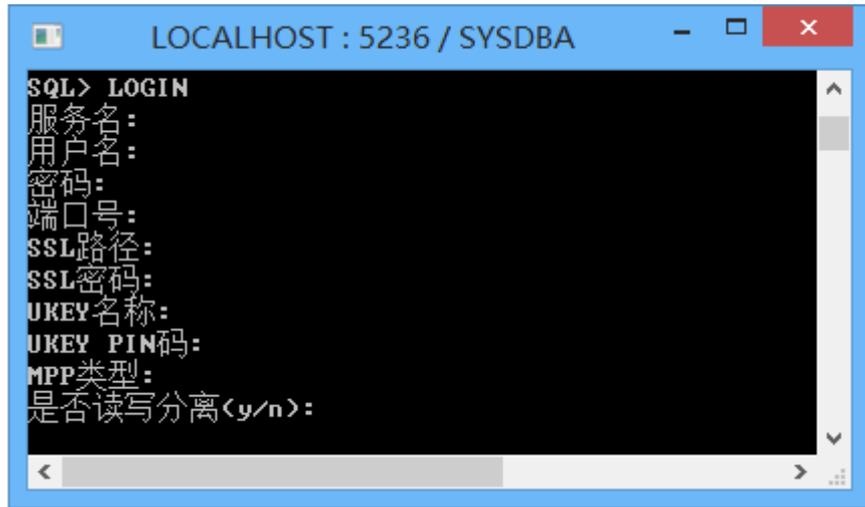


图 2.4 login 登录提示信息

服务名：数据库服务名或 IP 地址。LOCALHOST 表示本地服务器。默认为 LOCALHOST。

用户名和密码：默认均为 SYSDBA，密码不回显。

端口号：默认为 5236。

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，默认为 GLOBAL。

是否读写分离(y/n)：默认 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。

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

CONN[ECT]命令与 login 命令相似，增加的功能是，可以在命令之后直接跟 <username>[/<password>][*<MPP_TYPE>] [@<server>]，或者跟<username>，然后按照 DIsql 提示输入<password>来连接。

语法如下：

```
CONN[ECT] <logon>
<logon>::={<username>[/<password>][*<MPP_TYPE>][@<connect_identifier>]}|
{/ AS <SYSDBA|SYSSSO|SYSAUDITOR|AUTO>}
<connect_identifier>::=[<server>][:<port>][?{UDP|TCP|IPC|RDMA}][#<sslpath>][@
<sslpwd>]
```

<password>: 密码。如果密码中有特殊字符，需要特别处理。特殊字符指密码串中含有的关键字符 (/@:#*[]) 或双引号"。具体处理如下：

- 关键字符，需要用双引号括起连接串中的密码串。例如：用户名 user01，密码为 aaaa/aaaa，那么连接串要写成： conn user01/"aaaa/aaaa"。
- 双引号，需要用双引号括起连接串中的密码串，同时双引号还需要用双引号"或反斜杠/转义。例如：用户名 user01，密码为 aaaa"aaaa，那么连接串要写成 conn user01/"aaaa""aaaa"。

{/AS <SYSDBA|SYSSSO|SYSAUDITOR|AUTO>}：使用不同的管理员账号进行本地登录，指定 AUTO 会进行自动匹配相应的管理员。此方式将以 OS 操作系统身份验证模式登录，不需要输入口令。操作系统身份验证的请参考《DM8 安全管理》的 [基于操作系统的身份验证](#) 章节。例如：CONN / AS SYSDBA。

<server>: 指定服务器的 IP 地址或是在 dm_svc.conf 中配置的网络服务名。dm_svc.conf 的配置请参考《DM8 系统管理员手册》的 2.1.1 节。如果是 IPv6 的地址，需要用 [] 指明是 IPv6 地址，例如 [fe80::1e6f:65ff:fed1:3724%6]。

{UDP|TCP|IPC|RDMA} : 指定使用 UDP 协议、TCP 协议、IPC（共享内存）、RDMA（远程直接内存访问）。缺省为 TCP。例如：CONN SYSDBA/SYSDBA@localhost:5236?TCP。



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@192.168.0.38
```

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《DM7_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	SYSUSERS	16777241
2	USER_MVIEWS	16777243
3	USER_OBJECTS	16777244
4	USER_SOURCE	16777245
5	USER_TABLES	16777246

已用时间: 0.305(毫秒). 执行号:720.

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 DIsq1 环境变量设置

使用 SET 命令可以对当前 DIsq1 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。

3.1 DIsq1 环境变量

DIsq1 中的系统环境变量，汇总如下：

表 3.1 DIsq1 命令的快速参考

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

15	SCREENBUFSIZE	<DEFAULT(20K) n>	设置屏幕缓冲区的长度
16	CHAR_CODE	<GBK UTF8 DEFAULT (默认值, 操作系统的编码方式)>	设置 SQL 语句的编码方式
17	CURSOR	<STATIC FORWARDONLY (默认值) DEFAULT>	设置 DPI 语句句柄中游标的类型
18	AUTOTRACE	<OFF(默认值) NL INDEX ON TRACE>	设置执行计划和统计信息的跟踪
19	DESCRIBE	[DEPTH <1(默认值) n ALL>] [LINE[NUM] <ON OFF(默认值)>] [INDENT <ON OFF(默认值)>]	设置 DESCRIBE 的显示方式
20	TRIMS[POOL]	<OFF(默认值) ON>	设置 spool 文件中每行的结尾空格
21	LOBCOMPLETE	<OFF(默认值) 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(默认值)>	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 OFF(默认值)>	是否自动重新连接。ON 是, OFF 否。缺省为 OFF
25	NEST_COMMENT	<ON OFF(默认值)>	是否支持多行注释嵌套。ON 是, OFF 否。缺省为 OFF
26	NULL_ASNULL	<ON OFF(默认值)>	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(默认值) OFF>	是否执行文件中 “/” 命令。ON 是, OFF 否。缺省为 ON

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 (默认值)>
```

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(默认值)|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，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

```
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:

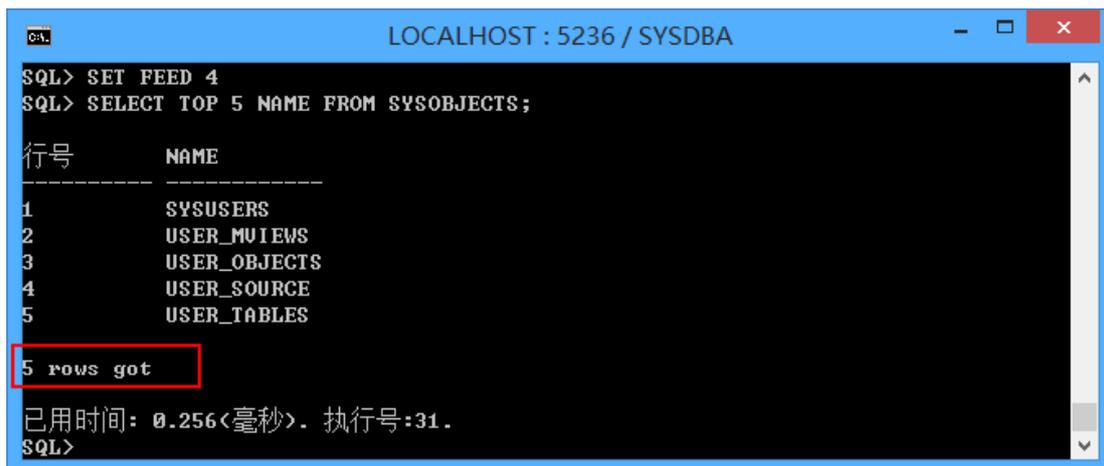
```
SET FEED[BACK] <6 (默认值) |n|ON|OFF>
```

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, 使用默认值 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]

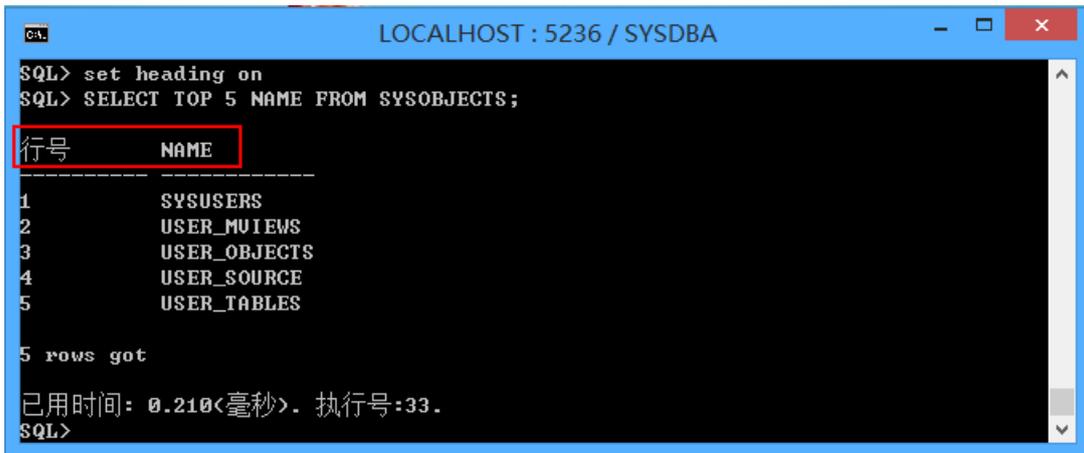
是否显示列标题。

语法如下：

```
SET HEA[DING] <ON (默认值) |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：



```
LOCALHOST : 5236 / SYSDBA
SQL> set heading on
SQL> SELECT TOP 5 NAME FROM SYSOBJECTS;
行号      NAME
-----
1          SYSUSERS
2          USER_MUIEWS
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 3.2HEA[DING]用法

3.3.6 LINESHOW

LINESHOW 设置是否显示行号。

语法如下：

```
SET LINESHOW<ON (默认值) |OFF >;
```

默认为每行输出打印行号。

例如：

```

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 (默认值) |n|NONE>
```

当 SET NEWPAGE 0 时，在每页的开头有一个换号符。

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 (默认值) |n>
```

如果设为 0，则所有的输出内容为一页并且不显示列标题。默认值为 14。

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 (默认值) |OFF>
```

3.3.10 TIME

显示系统的当前时间。

语法如下：

```
SET TIME<ON|OFF (默认值) >
```

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。默认值为 ON，表示列出命令文本。

语法如下：

```
SET VER[IFY] < ON (默认值) |OFF>
```

示例如下：

```
SQL> SET VERIFY OFF

SQL> SELECT &A FROM DUAL;

输入 A 的值:3

行号          3

-----

1              3

已用时间: 0.558(毫秒). 执行号:733.

SQL> SET VERIFY ON

SQL> SELECT &A FROM DUAL;

输入 A 的值:3

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(默认值) |n>
```

3.3.13 LINESIZE

设置屏幕上一行显示宽度。

语法如下：

```
SET LINESIZE <screen_length(默认值, 屏幕宽度) |n>
```

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(默认值)> [SIZE <20000(默认值) |n>]  
[FOR[MAT] <WRA[PPED] | WOR[D_WRAPPED](默认值) | 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。

```
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;

/

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.

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      /

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。

语法如下:

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

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(默认值, 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 (默认值)| 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(默认值) | NL | INDEX | ON|TRACE>
```

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。

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(默认值) | n | ALL>] [LINE[NUM] <ON | OFF(默认值)>]
```

```
[INDENT <ON | OFF(默认值)>]
```

DEPTH: 结构信息显示至第 N 层, 默认只显示第 1 层。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：

```
SETTRIMS[POOL] <OFF(默认值) | 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(默认值) | 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(默认值)>
```

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(默认值) | 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:

```
SETNEST_COMMENT <ON|OFF(默认值)>
```

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 SETNEST_COMMENT OFF 时, 只支持一行注释, 程序只要找到一组 /* 和 */ 就会结束。

```
SQL>SETNEST_COMMENT OFF

/*****          abcdffdddef

/*****

*/

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：

```
SETNULL_ASNULL<ON|OFF(默认值)>
```

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;
```

```
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.

```
SETNULL_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:

```
SETCMD_EXEC <ON (默认值)|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;
```

缺省 CMD_EXEC 为 ON。

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

```
SQL> select 1;
```

```
行号      1
```

```
-----
```

```
1          1
```

已用时间: 0.842(毫秒). 执行号:1664.

```
SQL> /
```

```
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.

修改 CMD_EXEC 为 OFF。

```
SQL> Set cmd_exec off

SQL> `d:\test.sql

SQL> select 1;

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.
```

3.4 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 DIsq1 环境的当前环境变量设置。

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。

语法如下：

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

<system_variable>:环境变量。

示例如下，显示 HEADING 和 TIMING 两个变量：

```
SQL> SHOW HEADING TIMING

HEADING ON.

TIMING ON
```

4DIsq1 常用命令

4.1 帮助 HELP

DIsq1 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所有命令用法。

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

将屏幕显示的内容输出到指定文件。

语法如下：

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

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

<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

OFF: 关闭 SPOOL 输出



注意: 只有 SPOOL OFF 之后, 才能在输出文件中看到输出的内容。

示例如下:

```
SQL>spool d:\b.sql
SQL>select top 5* from sysobjects;
SQL>spool off
```

先执行上述语句, 然后, 查看 d:\b.sql 文件。

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] <table>|<view>|<proc>|<fun>|<pkg>|<record>|<class> ;

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。

举例说明：

例 1 获取表 `sysgrants` 的结构描述。

```
SQL> desc sysgrants;

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
```

```
已用时间: 22.679(毫秒). 执行号:757.
```

例 3 获取包的结构描述

创建实例数据库:

```
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 PersonPackageAS
    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
        INSERTINTO 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 PersonListAS

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;

```

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

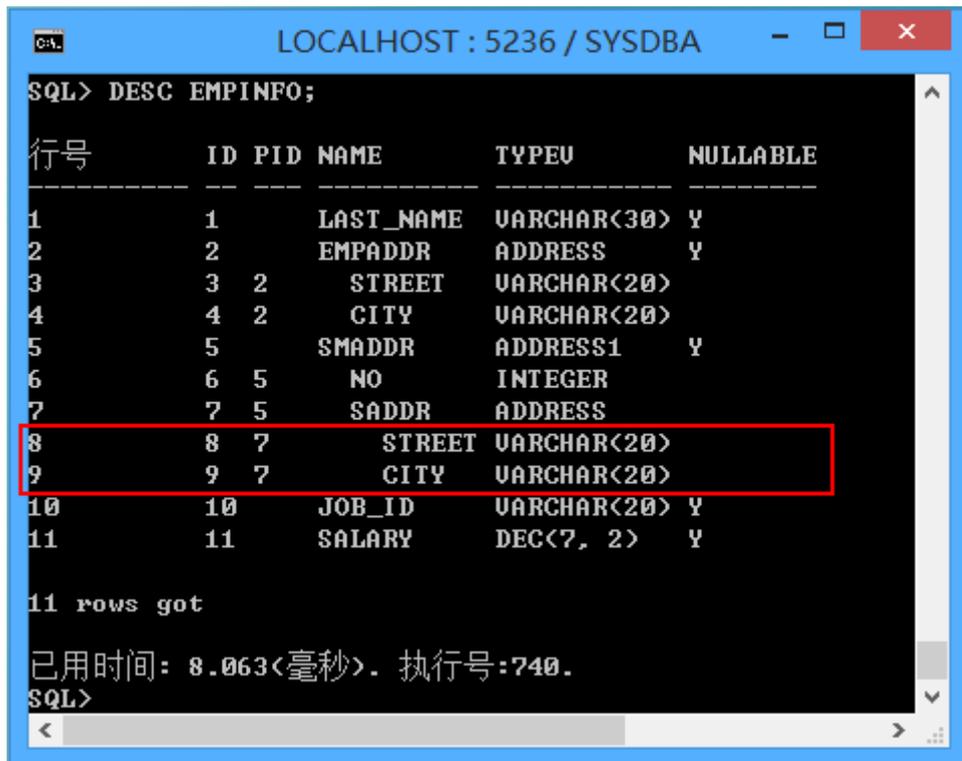
已用时间: 18.325(毫秒). 执行号:768.

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：



```
LOCALHOST : 5236 / SYSDBA  
SQL> DESC EMPINFO;  
行号      ID PID NAME          TYPEU          NULLABLE  
-----  
1         1      LAST_NAME    UARCHAR(30)  Y  
2         2      EMPADDR     ADDRESS       Y  
3         3  2      STREET      UARCHAR(20)  
4         4  2      CITY       UARCHAR(20)  
5         5      SMADDR     ADDRESS1     Y  
6         6  5      NO         INTEGER  
7         7  5      SADDR     ADDRESS  
8         8  7      STREET    UARCHAR(20)  
9         9  7      CITY     UARCHAR(20)  
10        10     JOB_ID     UARCHAR(20)  Y  
11        11     SALARY    DEC(7, 2)    Y  
  
11 rows got  
已用时间: 8.063<毫秒>. 执行号:740.  
SQL>
```

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。默认的变量前缀是&。示例如下：

```
SQL>DEF VAR=666;
SQL>select * from dual 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
```

Define 变量与其他字符之间的连接字符是点号'.'。示例如下：

```
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;
```

```
行号          C1
```

```
-----
```

```
1              1
```

```
已用时间: 0.428(毫秒). 执行号:702.
```

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

```
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.
```

DEFINE 变量定义为整型。示例如下:

```
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;
```

```
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);

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;

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.

```

DEFINE 变量定义为字符型。示例如下:

```

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;

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

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;

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);

行号          F1(0)
-----
1              777          -- 返回值 OG=777

已用时间: 0.460(毫秒). 执行号:713.

```

DEFINE 变量定义为日期类型。示例如下:

```

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

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

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

```

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, 表示该 column|expr 的值, 同时作为变量存在。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本地变量。

示例如下:

```

SQL> COLUMN CVAR NEW_VALUE DVAR
SQL> SELECT CUSTOMERID CVAR FROM SALES.CUSTOMER;
行号          CVAR
-----
1          1

```

```

2          2
3          3
4          4
5          5
6          6

```

```
6 rows got
```

已用时间: 1.105(毫秒). 执行号:1053.

```
SQL> DEFINE DVAR
```

```
DEFINE DVAR      = "6"      (INT)
```

通过如下方式设置将变量 VARIABLE 与列名 column 之间的关联, 默认值为 ON, 表示已关联。

语法如下:

```
COL[UMN] column <OFF|ON(默认值)>
```

示例如下:

```
SQL>COLUMN CVAR 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《DM7_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 SQLEERROR
```

```
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;  
  
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
```

例如，当执行如下语句时，用户想查看更多结果集，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。

```
LIST 或者;
```

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《DM7_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 testvalues(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 DIsq1 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 DIsq1 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。

<`运行脚本>中的符号`位于键盘第二排左起第一个。



注意:

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:

```

LOCALHOST : 5236 / SYSDBA
D:\dmdbms\bin>disql SYSDBA/SYSDBA `D:\test.sql
服务器[LOCALHOST:5236]:处于普通打开状态
使用普通用户登录
登录使用时间: 25.107<毫秒>
disql V7.1.5.10-Build(2015.10.09-61560trunc)
Connected to: DM 7.1.5.10
SQL> drop table t01;
操作已执行
已用时间: 15.922<毫秒>. 执行号:140.
SQL> create table t01(c1 varchar(100), c2 varchar(100));
操作已执行
已用时间: 31.899<毫秒>. 执行号:141.
SQL> begin
for i in 1..10 loop
insert into t01 values('a' || i, 'b' || i);
end loop;
end;
PL/SQL 过程已成功完成
已用时间: 0.504<毫秒>. 执行号:142.
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:

```

LOCALHOST : 5236 / SYSDBA
SQL> start D:\test.sql
SQL> drop table t01;
操作已执行
已用时间: 9.600<毫秒>. 执行号:143.
SQL> create table t01(c1 varchar(100), c2 varchar(100));
操作已执行
已用时间: 1.110<毫秒>. 执行号:144.
SQL> begin
for i in 1..10 loop
insert into t01 values('a' || i, 'b' || i);
end loop;
end;
PL/SQL 过程已成功完成
已用时间: 0.311<毫秒>. 执行号:145.
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。如脚本 D:\test.sql:

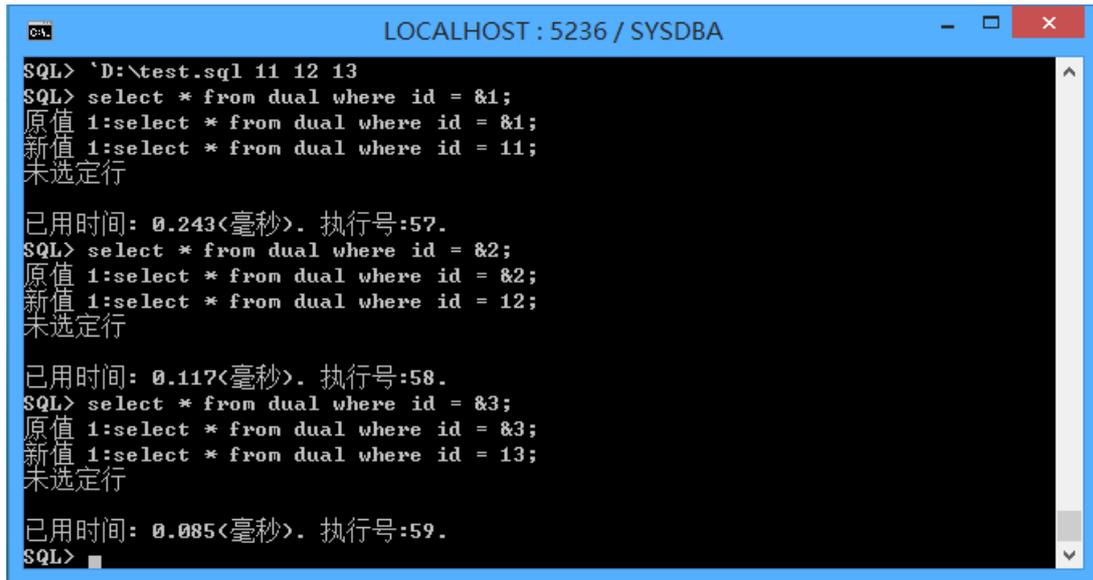
```
select * from dual where id = &1;
select * from dual where id = &2;
select * from dual where id = &3;
```

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。

示例，输入三个参数值 11、12、13：

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

执行结果如下：



```

LOCALHOST : 5236 / SYSDBA
SQL> 'D:\test.sql 11 12 13
SQL> select * from dual where id = &1;
原值 1:select * from dual where id = &1;
新值 1:select * from dual where id = 11;
未选定行

已用时间: 0.243<毫秒>. 执行号:57.
SQL> select * from dual where id = &2;
原值 1:select * from dual where id = &2;
新值 1:select * from dual where id = 12;
未选定行

已用时间: 0.117<毫秒>. 执行号:58.
SQL> select * from dual where id = &3;
原值 1:select * from dual where id = &3;
新值 1:select * from dual where id = 13;
未选定行

已用时间: 0.085<毫秒>. 执行号:59.
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');

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：

```

LOCALHOST : 5236 / SYSDBA
SQL> 'D:\test.sql' 'hello' ''hello world''
SQL> create table test(a varchar);
操作已执行
已用时间: 1.551<毫秒>. 执行号:244.
SQL> insert into test values('hello');
影响行数 1
已用时间: 0.226<毫秒>. 执行号:245.
SQL> insert into test values('hello world');
影响行数 1
已用时间: 0.135<毫秒>. 执行号:246.
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: 0.315<毫秒>. 执行号:247.
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: 0.237<毫秒>. 执行号:248.
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=DIsq1

select &n from dual;

select '&s' from dual;

```

如果变量没有定义，那么在通过&引用时，DIsq1 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:

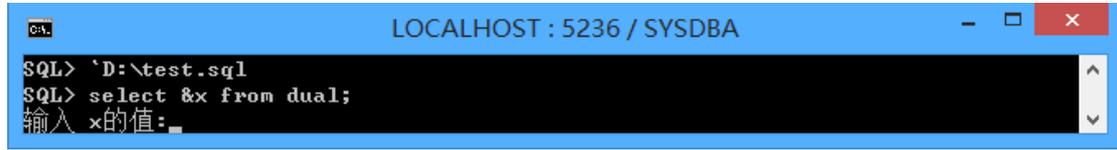


图 5.5 交互式输入参数值用法

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递信息。

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 10 * from v$dm_ini;
```

执行脚本:

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

结果如下:

```
SQL> `f:\prompt.sql
SQL> prompt 所有 ini 参数和 dminit 建库参数信息 (系统值、最小值和最大值);
所有 ini 参数和 dminit 建库参数信息 (系统值、最小值和最大值)
SQL> select top 10 * from v$dm_ini;
```

行号	PARA_NAME	PARA_VALUE	MIN_VALUE	MAX_VALUE
1	CTL_PATH	D:\dmdbms\data\DAMENG\dm.ctl	NULL	NULL
2	SYSTEM_PATH	D:\dmdbms\data\DAMENG	NULL	NULL

3	TEMP_PATH	D:\dmdbms\data\DAMENG	NULL	NULL
4	BAK_PATH	D:\dmdbms\data\DAMENG\bak	NULL	NULL
5	BAK_POLICY	0	0	2
6	AUD_PATH	NULL	NULL	NULL
7	INSTANCE_NAME	DMSERVER	NULL	NULL
8	INSTANCE_ADDR	NULL	NULL	NULL
9	MAX_OS_MEMORY	90	40	100
10	MEMORY_POOL	161	5	67108864

10 rows got

已用时间：0.538(毫秒)。执行号：740。