

# | 达梦技术手册



# 前言

## 1.1 概述

本文档主要介绍达梦特有的 DMGEO 系统包和兼容 ORACLE 数据库的 DBMS\_ALERT、DBMS\_OUTPUT、UTL\_FILE 和 UTL\_MAIL 等系统包的功能简介、相关方法和举例说明等。

## 1.2 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

## 1.3 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。
黑体	标题、警告、注意、小窍门、说明等内容均采用黑体。
Courier new	表示代码或者屏幕显示内容。
粗体	表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。
<>	语法符号中，表示一个语法对象。
::=	语法符号中，表示定义符，用来定义一个语法对象。定义符左边为语法对象，右边为相应的语法描述。
	语法符号中，表示或者符，限定的语法选项在实际语句中只能出现一个。
{ }	语法符号中，大括号内的语法选项在实际的语句中可以出现 0...N 次(N 为大于 0 的自然数)，但是大括号本身不能出现在语句中。
[ ]	语法符号中，中括号内的语法选项在实际的语句中可以出现 0...1 次，但是中括号本身不能出现在语句中。
关键字	关键字在 DM_SQL 语言中具有特殊意义，在 SQL 语法描述中，关键字以大写形式出现。但在实际书写 SQL 语句时，关键字既可以大写也可以小写。

## 1.4 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](http://www.dameng.com) 阅读或下载 DM 的各种相关文档。

## 1.5 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：[dmttech@dameng.com](mailto:dmttech@dameng.com)

# 目录

<b>1</b>	<b>概述 .....</b>	<b>1</b>
1.1	系统包创建、删除语句 .....	1
<b>2</b>	<b>DMGEO 包 .....</b>	<b>5</b>
2.1	数据类型 .....	5
2.2	相关方法 .....	11
2.3	创建、检测、删除语句 .....	38
<b>3</b>	<b>DBMS_ADVANCED_REWRITE 包 .....</b>	<b>40</b>
3.1	相关方法 .....	40
3.2	字典信息 .....	43
3.3	使用说明 .....	43
3.4	举例说明 .....	45
<b>4</b>	<b>DBMS_ALERT 包 .....</b>	<b>47</b>
4.1	相关方法 .....	47
4.2	举例说明 .....	49
<b>5</b>	<b>DBMS_BINARY 包 .....</b>	<b>51</b>
5.1	相关方法 .....	51
5.2	错误处理 .....	55
5.3	举例说明 .....	55
<b>6</b>	<b>DBMS_JOB 包 .....</b>	<b>57</b>
6.1	相关方法 .....	57
6.2	DBMS_JOB 视图 .....	60
6.3	创建、删除语句 .....	62
6.4	举例说明 .....	62
<b>7</b>	<b>DBMS_LOB 包 .....</b>	<b>64</b>
7.1	相关方法 .....	64
7.2	举例说明 .....	79
<b>8</b>	<b>DBMS_LOCK 包 .....</b>	<b>81</b>
8.1	封锁规则 .....	81
8.2	相关方法 .....	82
8.3	举例说明 .....	85
<b>9</b>	<b>DBMS_LOGMNR 包 .....</b>	<b>88</b>
9.1	相关方法 .....	88
9.2	举例说明 .....	91

<b>10</b>	<b>DBMS_METADATA 包</b>	<b>93</b>
10.1	数据类型	93
10.2	相关方法	94
10.3	错误处理	104
10.4	举例说明	104
<b>11</b>	<b>DBMS_OBFUSCATION_TOOLKIT 包</b>	<b>109</b>
11.1	相关方法	109
11.2	使用说明	119
11.3	举例说明	119
<b>12</b>	<b>DBMS_OUTPUT 包</b>	<b>123</b>
12.1	相关方法	123
12.2	举例说明	124
<b>13</b>	<b>DBMS_PAGE 包</b>	<b>125</b>
13.1	使用前提	125
13.2	索引页	125
13.3	INODE 页	133
13.4	描述页	136
13.5	控制页	139
<b>14</b>	<b>DBMS_PIPE 包</b>	<b>147</b>
14.1	相关方法	147
14.2	动态视图	151
14.3	举例说明	151
<b>15</b>	<b>DBMS_RANDOM 包</b>	<b>152</b>
15.1	相关方法	152
15.2	举例说明	154
<b>16</b>	<b>DBMS_RLS 包</b>	<b>155</b>
16.1	策略函数	155
16.2	策略组	155
16.3	策略	156
16.4	上下文	160
16.5	举例说明	161
<b>17</b>	<b>DBMS_SESSION 包</b>	<b>168</b>
17.1	相关方法	168
17.2	举例说明	170
<b>18</b>	<b>DBMS_SPACE 包</b>	<b>174</b>
18.1	数据类型	174
18.2	相关方法	175
18.3	举例说明	181

<b>19</b>	<b>DBMS_SQL 包</b>	<b>183</b>
19.1	相关方法	183
19.2	创建语句	192
19.3	举例说明	193
<b>20</b>	<b>DBMS_TRANSACTION 包</b>	<b>195</b>
20.1	相关方法	195
20.2	举例说明	196
<b>21</b>	<b>DBMS_STATS 包</b>	<b>197</b>
21.1	数据类型	197
21.2	相关方法	197
21.3	约束	210
21.4	举例说明	210
<b>22</b>	<b>DBMS_UTILITY 包</b>	<b>213</b>
22.1	相关方法	213
<b>23</b>	<b>DBMS_WORKLOAD_REPOSITORY 包</b>	<b>215</b>
23.1	相关方法	215
23.2	创建、检测、删除语句	219
23.3	举例说明	220
<b>24</b>	<b>DBMS_XMLGEN 包</b>	<b>221</b>
24.1	使用前提	221
24.2	相关方法	221
24.3	举例说明	226
<b>25</b>	<b>UTL_ENCODE 包</b>	<b>229</b>
25.1	相关方法	229
25.2	举例说明	229
<b>26</b>	<b>UTL_FILE 包</b>	<b>230</b>
26.1	数据类型	230
26.2	相关方法	230
26.3	错误处理	238
26.4	举例说明	239
<b>27</b>	<b>UTL_INADDR 包</b>	<b>242</b>
27.1	相关方法	242
27.2	举例说明	242
<b>28</b>	<b>UTL_MAIL 包</b>	<b>243</b>
28.1	相关方法	243
28.2	举例说明	245
<b>29</b>	<b>UTL_MATCH 包</b>	<b>247</b>

---

29.1	相关方法 .....	247
29.2	举例说明 .....	247
<b>30</b>	<b>UTL_RAW 包 .....</b>	<b>250</b>
30.1	相关方法 .....	250
30.2	举例说明 .....	258
<b>31</b>	<b>UTL_TCP 包 .....</b>	<b>262</b>
31.1	相关方法 .....	262
31.2	举例说明 .....	264
<b>32</b>	<b>UTL_URL 包 .....</b>	<b>266</b>
32.1	相关方法 .....	266
32.2	举例说明 .....	267
<b>33</b>	<b>DBMS_SCHEDULER 包 .....</b>	<b>268</b>
33.1	相关方法 .....	268
33.2	创建、删除语句 .....	287
33.3	相关视图 .....	287
33.4	日历语法 .....	293
33.5	举例说明 .....	295
<b>34</b>	<b>DBMS_MVIEW 包 .....</b>	<b>298</b>
34.1	相关方法 .....	298
34.2	举例说明 .....	299
<b>35</b>	<b>UTL_SMTP 包 .....</b>	<b>301</b>
35.1	相关方法 .....	301
35.2	应用实例 .....	305
<b>36</b>	<b>UTL_HTTP 包 .....</b>	<b>307</b>
36.1	相关方法 .....	307
36.2	举例说明 .....	312
<b>37</b>	<b>UTL_I18N 包 .....</b>	<b>314</b>
1.6	37.1 相关方法 .....	314
1.7	37.2 举例说明 .....	315

# 1 概述

达梦数据库提供了达梦特有的 DMGEO 系统包和兼容 ORACLE 数据库的 DBMS\_ALERT、DBMS\_OUTPUT、UTL\_FILE 和 UTL\_MAIL 等系统包功能。

## 1.1 系统包创建、删除语句

DM 在新建库第一次启动服务器时会自动创建除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 之外的所有系统包。

用户也可以通过调用系统过程来创建、删除指定系统包：

SP\_CREATE\_SYSTEM\_PACKAGES(create\_flag, pkgname) 创建或删除指定的包；  
SP\_CREATE\_SYSTEM\_PACKAGES(create\_flag) 一次性创建或删除除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 之外的所有系统包。此外，还可以使用 SF\_CHECK\_SYSTEM\_PACKAGES() 来检测系统包启用状态。

DMGEO 包、DBMS\_JOB 包、DBMS\_WORKLOAD\_REPOSITORY 包和 DBMS\_SCHEDULER 包的创建方式比较特殊，请参考具体章节。没有特别说明的，创建和删除方式都请参考此处。



**SP\_CREATE\_SYSTEM\_PACKAGES(create\_flag, pkgname) 和**

**说明：SP\_CREATE\_SYSTEM\_PACKAGES(create\_flag) 使用限制：**

1. 不能在 MPP 全局模式下的存储过程中直接调用，在 MPP LOCAL 模式下可在存储过程中直接调用；
2. 不能在存储过程中带参数进行动态调用。

### 1.1.1 创建指定的包

SP\_CREATE\_SYSTEM\_PACKAGES 用来创建或删除指定的系统包，除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 以外。

语法如下：

---

```
void
SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG      int,
    PKGNAME           varchar(128)
```

---



---

)

---

#### 参数详解

- CREATE\_FLAG

为 1 时表示创建指定的系统包；为 0 表示删除这个系统包。

- PKGNAME

指定要创建的包名。除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 以外的系统包。

#### 返回值

无

#### 举例说明

创建 DBMS\_LOB 系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1, 'DBMS_LOB');
```

### 1.1.2 创建所有系统包

SP\_CREATE\_SYSTEM\_PACKAGES，用来创建或删除除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 以外的所有系统包。若在创建过程中某个系统包由于特定原因未能创建成功，会跳过继续创建后续的系统包。

语法如下：

---

```
void
SP_CREATE_SYSTEM_PACKAGES (
    CREATE_FLAG      int
)

```

---

#### 参数详解

- CREATE\_FLAG

为 1 时表示创建除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 以外的所有系统包；为 0 表示删除这些系统包。

#### 返回值

无

#### 举例说明

创建除了 DMGEO、DBMS\_JOB、DBMS\_WORKLOAD\_REPOSITORY 和 DBMS\_SCHEDULER 以外的所有系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1);
```

### 1.1.3 检测系统包是否启用

#### 1. SF\_CHECK\_SYSTEM\_PACKAGES

SF\_CHECK\_SYSTEM\_PACKAGES 用来检测系统包的启用状态。是否启用系统包，由是否执行过 SP\_CREATE\_SYSTEM\_PACKAGES (1); 语句决定，执行过，表示已启用。

语法如下：

---

```
int
```

```
SF_CHECK_SYSTEM_PACKAGES ( )
```

---

#### 返回值

0：未启用；1：已启用

#### 举例说明

获得系统包的启用状态。

```
SELECT SF_CHECK_SYSTEM_PACKAGES;
```

#### 2. SF\_CHECK\_SYSTEM\_PACKAGE(<PACKAGE\_NAME>)

SF\_CHECK\_SYSTEM\_PACKAGE(<PACKAGE\_NAME>) 可用来检查某个特定的系统包是否启用。

语法如下：

---

```
int
```

```
SF_CHECK_SYSTEM_PACKAGES (
```

```
    PACKAGE_NAME          VARCHAR
```

```
)
```

---

#### 返回值

0：未启用；1：已启用

#### 举例说明

获得系统包 DBMS\_SCHEDULER 的启用状态。

```
SELECT SF_CHECK_SYSTEM_PACKAGE('DBMS_SCHEDULER');
```

### 1.1.4 包间依赖关系

部分系统包之间有依赖关系。当用户创建这些系统包的时候，系统也会自动创建它们所

依赖的包。被依赖包一旦被删除，依赖包就会失效。

- DBMS\_METADATA 依赖 DBMS\_LOB, UTL\_RAW;
- DBMS\_LOB 依赖于 UTL\_RAW;
- DBMS\_ALERT 包依赖于 DBMS\_LOCK 和 DBMS\_UTILITY 两个包;
- DBMS\_WORKLOAD\_REPOSTIORY 包依赖于 UTL\_FILE;

## 2 DMGEO 包

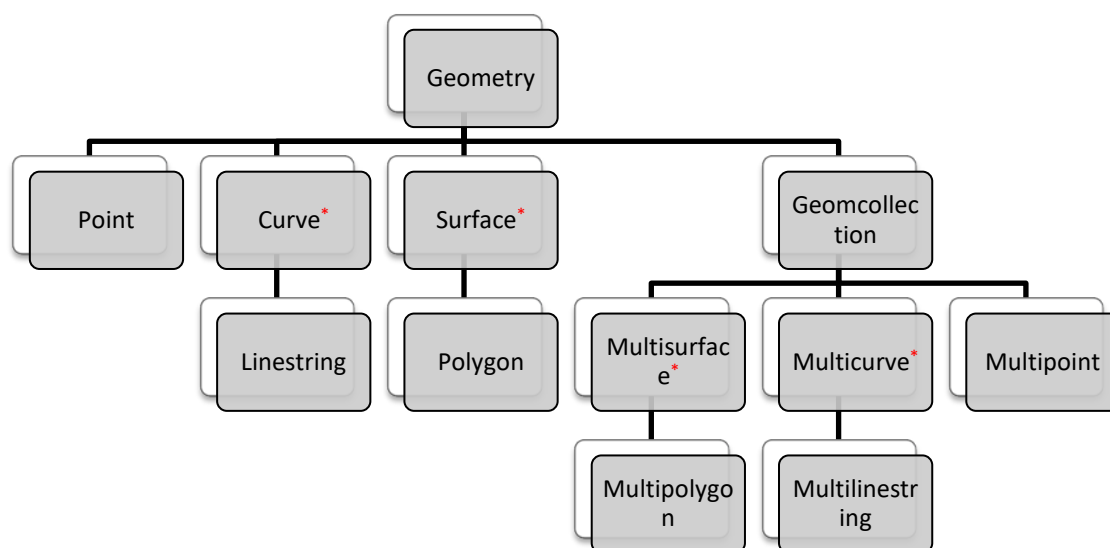
DMGEO 系统包实现了 SFA 标准（《OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option》）中规定的 SQL 预定义 schema，基于 SQL UDT（自定义数据类型）的空间数据类型和空间数据类型的初始化，以及针对空间数据类型的几何体计算函数。

### 2.1 数据类型

达梦的空间数据类型是以类的方式进行体现的，具体有：

- ST\_Geometry：最基本的几何体，是其他几何类型的基类
- ST\_Point：点几何体
- ST\_Curve, ST\_LineString：线几何体。ST\_Curve 是抽象类，ST\_LineString 是 ST\_Curve 可实例化的子类
- ST\_Surface, ST\_Polygon：面几何体。ST\_Surface 是抽象类，ST\_Polygon 是 ST\_Surface 可实例化的子类
- ST\_GeomCollection：几何体集合
- ST\_Multipoint：点集合
- ST\_Multicurve, ST\_Multilinestring：线集合。ST\_Multicurve 是抽象类，ST\_Multilinestring 是 ST\_Multicurve 可实例化的子类
- ST\_Multisurface, ST\_Multipolygon：多边形集合。ST\_Multisurface 是抽象类，ST\_Multipolygon 是 ST\_Multisurface 可实例化的子类

相关类型的继承关系如下图所示（带\*号的表示抽象父类）：



如下详细介绍各数据类型：

#### 1. ST\_Geometry

基础几何体类。

语法如下：

---

```

CREATE OR REPLACE TYPE SYSGEO.ST_GEOMETRY AS OBJECT(
    SRID                INT,
    GEO_WKB              BLOB,
    GEO_TYPEID          INT,
    CONSTRUCTOR FUNCTION ST_GEOMETRY() RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION ST_GEOMETRY(SRID INT, WKB BLOB, TYPEID INT) RETURN SELF
AS RESULT,
    FUNCTION ST_WKT()      RETURN CLOB,
    FUNCTION ST_DIM()      RETURN INT,
    FUNCTION ST_TYPE()     RETURN VARCHAR(20),
    FUNCTION ST_ISEMPTY()  RETURN INT,
    FUNCTION ST_ISSIMPLE() RETURN INT,
    FUNCTION ST_ISVALID()  RETURN INT,
    FUNCTION ST_COORDIM()  RETURN INT,
    FUNCTION ST_NUMGEOS()  RETURN INT,
    FUNCTION ST_NUMPOINTS() RETURN INT,
    FUNCTION ST_LENGTH()   RETURN DOUBLE,
    FUNCTION ST_POINT_X()  RETURN DOUBLE,
    FUNCTION ST_POINT_Y()  RETURN DOUBLE,
    FUNCTION ST_START_POINT() RETURN SYSGEO.ST_GEOMETRY,
    FUNCTION ST_END_POINT() RETURN SYSGEO.ST_GEOMETRY,
    FUNCTION ST_IS_RING()  RETURN INT,
    FUNCTION ST_IS_CLOSED() RETURN INT,
    FUNCTION ST_CENTROID() RETURN SYSGEO.ST_GEOMETRY,

```

---

---

```

FUNCTION ST_POINT_ON( )      RETURN SYSGEO.ST_GEOMETRY,
FUNCTION ST_AREA( )          RETURN DOUBLE,
FUNCTION ST_EXT_RING( )      RETURN SYSGEO.ST_GEOMETRY,
FUNCTION ST_NUM_INTER_RING( ) RETURN INT)
NOT FINAL;

```

---

**类成员说明:**

- SRID 空间参考坐标系 ID。
- geo\_wkb 二进制格式的几何体信息。
- geo\_typeid 几何体类型 ID。

**类方法说明:**

- st\_geometry() st\_geometry 类的构造函数。
- st\_wkt() 返回文本格式表示的几何体。
- st\_dim() 几何体的几何维度。
- st\_type() 几何体的类型名。
- st\_isempty() 几何体是否为空。
- st\_issimple() 是否为简单几何体。
- st\_isvalid() 几何体信息是否有效。
- st\_coordim() 几何体的坐标维度。
- st\_NumGeos() 几何体对象个数。
- ST\_NUMPOINTS() 几何体点的个数。
- ST\_LENGTH() 几何体长度。
- ST\_POINT\_X() 点几何体的 X 坐标。
- ST\_POINT\_Y() 点几何体的 Y 坐标。
- ST\_START\_POINT() 线几何体的起点。
- ST\_END\_POINT() 线几何体的终点。
- ST\_IS\_RING() 线几何体是否是环。
- ST\_IS\_CLOSED() 几何体是否是封闭的。
- ST\_CENTROID() 几何体质心。
- ST\_POINT\_ON() 几何体表面上的点。
- ST\_AREA() 几何体面积。
- ST\_EXT\_RING() 几何体外环。
- ST\_NUM\_INTER\_RING() 几何体内环数。

**2. ST\_POINT**

点类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_POINT UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION    ST_POINT() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION    ST_POINT(SRID INT, WKB BLOB, TYPEID INT) RETURN SELF  
AS RESULT) NOT FINAL;
```

---

**类说明:**

- `st_point()` 点类构造方法。

### 3. ST\_Curve

抽象线类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_CURVE UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION    ST_CURVE() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION    ST_CURVE(SRID INT, WKB BLOB, TYPEID INT) RETURN SELF  
AS RESULT) NOT FINAL;
```

---

**类说明:**

- `st_curve()` 几何线基类构造方法。

### 4. ST\_Linestring

线类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_LINESTRING UNDER SYSGEO.ST_CURVE(  
    CONSTRUCTOR FUNCTION    ST_LINESTRING() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION    ST_LINESTRING(SRID INT, WKB BLOB, TYPEID INT) RETURN  
SELF AS RESULT,  
    FUNCTION CURVE() RETURN SYSGEO.ST_CURVE) NOT FINAL;
```

---

**类说明:**

- `st_linestring()` 几何线类构造方法。

### 5. ST\_Surface

基础面类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_SURFACE UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION    ST_SURFACE() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION    ST_SURFACE(SRID INT, WKB BLOB, TYPEID INT) RETURN SELF  
AS RESULT) NOT FINAL;
```

---

**类说明:**

- `st_surface()` 几何面基类构造方法。

## 6. ST\_Polygon

面类。

语法如下：

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_POLYGON UNDER SYSGEO.ST_SURFACE(  
    CONSTRUCTOR FUNCTION ST_POLYGON() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION ST_POLYGON(SRID INT, WKB BLOB, TYPEID INT) RETURN SELF  
AS RESULT,  
    FUNCTION SURFACE() RETURN SYSGEO.ST_SURFACE) NOT FINAL;
```

---

### 类说明：

- `st_polygon()` 几何面类构造方法。

## 7. ST\_GeomCollection

基础几何体集合类。

语法如下：

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_GEOMCOLLECTION UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION ST_GEOMCOLLECTION() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION ST_GEOMCOLLECTION(SRID INT, WKB BLOB, TYPEID INT) RETURN  
SELF AS RESULT) NOT FINAL;
```

---

### 类说明：

- `st_geomcollection()` 几何体集合类构造方法。

## 8. ST\_Multipoint

多点类语法如下：

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_MULTIPPOINT UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION ST_MULTIPPOINT() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION ST_MULTIPPOINT(SRID INT, WKB BLOB, TYPEID INT) RETURN  
SELF AS RESULT) NOT FINAL;
```

---

### 类说明：

- `st_multipoint()` 多点类构造方法。

## 9. ST\_MultiCurve

抽象多线类。

语法如下：

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_MULTICURVE UNDER SYSGEO.ST_GEOMETRY(  
    CONSTRUCTOR FUNCTION ST_MULTICURVE() RETURN SELF AS RESULT,  
    CONSTRUCTOR FUNCTION ST_MULTICURVE(SRID INT, WKB BLOB, TYPEID INT) RETURN
```



---

SELF AS RESULT) NOT FINAL;

---

#### 类说明:

- `st_multicurve()` 多线基类构造方法。

10. `ST_Multilinestring`

多线类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_MULTILINESTRING UNDER SYSGEO.ST_MULTICURVE(
    CONSTRUCTOR FUNCTION    ST_MULTILINESTRING() RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION    ST_MULTILINESTRING(SRID INT, WKB BLOB, TYPEID INT)
RETURN SELF AS RESULT,
    FUNCTION ST_MCURVE() RETURN SYSGEO.ST_MULTICURVE) NOT FINAL;
```

---

#### 类成员说明:

- `st_multilinestring()` 多线类构造方法。
- `st_mcurve()` 返回 `ST_MultiCurve` 几何类对象。

11. `ST_MultiSurface`

抽象多面类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_MULTISURFACE UNDER SYSGEO.ST_GEOMETRY(
    CONSTRUCTOR FUNCTION    ST_MULTISURFACE() RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION    ST_MULTISURFACE(SRID INT, WKB BLOB, TYPEID INT)
RETURN SELF AS RESULT) NOT FINAL;
```

---

#### 类说明:

- `st_multisurface()` 多面基类构造方法。

12. `ST_Multipolygon`

多面类。

语法如下:

---

```
CREATE OR REPLACE TYPE SYSGEO.ST_MULTIPOLYGON UNDER SYSGEO.ST_MULTISURFACE(
    CONSTRUCTOR FUNCTION    ST_MULTIPOLYGON() RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION    ST_MULTIPOLYGON(SRID INT, WKB BLOB, TYPEID INT)
RETURN SELF AS RESULT,
    FUNCTION ST_MSURFACE() RETURN SYSGEO.ST_MULTISURFACE) NOT FINAL;
```

---

#### 类说明:

- `st_multipolygon()` 多面类构造方法。
- `st_msurface()` 返回 `st_msurface` 几何类对象。

## 2.2 相关方法

用户在使用 DMGEO 包之前,需要提前调用系统过程 SP\_INIT\_GEO\_SYS(1) 创建 DMGEO 包,包创建成功后就可以使用空间数据类型以及包提供的方法。

```
SP_INIT_GEO_SYS(1);
```

如果 dmgeo 包已存在,调用 SP\_INIT\_GEO\_SYS(2),系统将只重建 dmgeo 包方法,而不影响现有的空间数据类型以及数据。

```
SP_INIT_GEO_SYS(2);
```

调用系统过程 SP\_INIT\_GEO\_SYS(0) 可以删除 DMGEO 包,任何与空间数据类型相关的表、函数、过程、触发器等对象均会被级联删除。

```
SP_INIT_GEO_SYS(0);
```

创建 DMGEO 包后,系统会创建表 GEOMETRY\_COLUMNS 和 SPATIAL\_REF\_SYS,这两张表分别记录空间数据类型列创建情况和空间参考坐标系信息。GEOMETRY\_COLUMNS 内容不需要用户干预,SPATIAL\_REF\_SYS 内容可以通过 DMGEO 包提供的存储过程 ST\_ADD\_SPATIAL\_REF 和 ST\_DEL\_SPATIAL\_REF 增加或删除。

### 2.2.1 几何体构造函数

几何体构造函数大体可分为两类:通过 WKT 信息与 SRID 信息构造和通过 WKB 信息与 SRID 信息构造。通常直接构造时选择通过 WKT 信息与 SRID 信息构造,从数据库获取信息时会使用 WKB 信息与 SRID 信息构造,因此下面仅给出第一种构造方式的例子。

达梦实现的几何体函数以包中方法的形式提供给用户,包名为 dmgeo。使用如下函数时,必须加上包名 dmgeo。

#### 1. ST\_GeomFromText

根据 wkt 信息和 srid 信息构造空间数据基础类。

语法如下:

```
FUNCTION ST_GEOMFROMTEXT(
    WKT      CLOB,
    SRID     INT
)RETURN ST_GEOMETRY;
```

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

#### 2. ST\_PointFromText

根据 wkt 信息和 srid 信息构造点类。

语法如下：

---

```
FUNCTION ST_POINTFROMTEXT (  
    WKT      CLOB,  
    SRID     INT  
)RETURN ST_POINT;
```

---

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

#### 举例说明

```
dmgeo.ST_PointFromText('point empty' , 0); //空对象  
dmgeo.ST_PointFromText('point (1 1)' , 0);
```

#### 3. ST\_LineFromText

根据 wkt 信息和 srid 信息构造线类。

语法如下：

---

```
FUNCTION ST_LINEFROMTEXT (  
    WKT      CLOB,  
    SRID     INT  
)RETURN ST_LINestring;
```

---

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

#### 举例说明

```
dmgeo.ST_LineFromText('linestring empty' , 0); //空对象  
dmgeo.ST_LineFromText('linestring (1 1, 2 2)' , 0);
```

#### 4. ST\_PolyFromText

根据 wkt 信息和 srid 信息构造面类。

语法如下：

---

```
FUNCTION ST_POLYFROMTEXT (  
    WKT      CLOB,  
    SRID     INT  
)RETURN ST_POLYGON;
```

---

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

### 举例说明

```
dmgeo.ST_PolyFromText('polygon empty' , 0); //空对象
dmgeo.ST_PolyFromText('polygon ((1 1, 1 2, 2 2, 2 1, 1 1))', 0);
```

#### 5. ST\_MPointFromText

根据 wkt 信息和 srid 信息构造多点类。

语法如下：

---

```
FUNCTION ST_MPOINTFROMTEXT (
    WKT      CLOB,
    SRID     INT
)RETURN ST_MULTIPOINT;
```

---

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

#### 举例说明

```
dmgeo.ST_MPointFromText('multipoint empty' , 0 ); //空对象
dmgeo.ST_MPointFromText('multipoint (1 1, 2 2)', 0 );
```

#### 6. ST\_MLineFromText

根据 wkt 信息和 srid 信息构造多线类。

语法如下：

---

```
FUNCTION ST_MLINEFROMTEXT (
    WKT      CLOB,
    SRID     INT
)RETURN ST_MULTILINESTRING;
```

---

#### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

#### 举例说明

```
dmgeo.ST_MLineFromText('multilinestring empty' , 0); //空对象
dmgeo.ST_MLineFromText('multilinestring ((1 1,2 1),(4 4, 3 3))', 0);
```

#### 7. ST\_MPolyFromText

根据 wkt 信息和 srid 信息构造多面类。

语法如下：

---

```
FUNCTION ST_MPOLYFROMTEXT (
    WKT      CLOB,
```

---

---

```

SRID      INT
)RETURN ST_MULTIPOLYGON;

```

---

### 参数详解

- wkt 几何对象文本描述信息。
- srid 空间参考坐标系 ID。

### 举例说明

```

dmgeo.ST_MPolyFromText('multipolygon empty' , 0); //空对象
dmgeo.ST_MPolyFromText('multipolygon (((1 1, 1 2, 2 2, 2 1, 1 1)), ((3 3, 3 4,
4 4, 4 3, 3 3)))' , 0);

```

## 8. ST\_GeomFromWKB

根据 wkb 信息和 srid 信息构造空间数据基础类对象。

语法如下:

---

```

FUNCTION ST_GEOMFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_GEOMETRY;

```

---

### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

## 9. ST\_PointFromWKB

根据 wkb 信息和 srid 信息构造点类。

语法如下:

---

```

FUNCTION ST_POINTFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_POINT;

```

---

### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

## 10. ST\_LineFromWKB

根据 wkb 信息和 srid 信息构造线类。

语法如下:

---

```

FUNCTION ST_LINEFROMWKB (
    WKB      BLOB,

```

---

---

```

        SRID      INT
    )RETURN ST_LINESTRING;

```

---

#### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

#### 11. ST\_PolyFromWKB

根据 wkb 信息和 srid 信息构造面类。

语法如下：

---

```

FUNCTION ST_POLYFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_POLYGON;

```

---

#### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

#### 12. ST\_MPointFromWKB

根据 wkb 信息和 srid 信息构造多点类。

语法如下：

---

```

FUNCTION ST_MPOINTFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_MULTIPPOINT;

```

---

#### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

#### 13. ST\_MLineFromWKB

根据 wkb 信息和 srid 信息构造多线类。

语法如下：

---

```

FUNCTION ST_MLINEFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_MULTILINESTRING;

```

---

#### 参数详解

- wkb 几何对象序列化描述信息。

- srid 空间参考坐标系 ID。

#### 14. ST\_MPolyFromWKB

根据 wkb 信息和 srid 信息构造多面类。

语法如下：

---

```
FUNCTION ST_MPOLYFROMWKB (
    WKB      BLOB,
    SRID     INT
)RETURN ST_MULTIPOLYGON;
```

---

#### 参数详解

- wkb 几何对象序列化描述信息。
- srid 空间参考坐标系 ID。

#### 15. ST\_CreateCircle

根据几何体构造圆/椭圆。根据给定几何体的最左边、最上边、最右边和最下边四个边界点构建一个方形或矩形，再构建出该方形或矩形的内切圆或椭圆。

语法如下：

---

```
FUNCTION ST_CREATECIRCLE(
    GEO SYSGEO.ST_GEOMETRY,
    N INT)
RETURN SYSGEO.ST_GEOMETRY;
```

---

#### 参数详解

- GEO 几何体对象。
- N 指定生成圆（椭圆）的坐标个数，给定参数 N 将生成 N 个坐标点。N 个点均匀的排列在圆或椭圆上。考虑到 POLYGON 封闭的特征，给定参数 N 将生成 (N + 1) 个点，因为必须保证 POLYGON 首尾相连，所以会有一个重复点。

#### 返回值

一个 POLYGON 空间类型对象。

#### 举例说明

构建一个圆。

```
declare
geol st_geometry;
geo3 st_geometry;
begin
    geol = dmgeo.st_geomfromtext( 'polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4296);
```

```

geo3 = dmgeo.ST_CreateCircle(geo1, 6);

select dmgeo.st_astext(geo3, 4);

end;

/

```

结果:

```

POLYGON ((10.0000 5.0000, 7.5000 9.3301, 2.5000 9.3301, 0.0000 5.0000, 2.5000
0.6699, 7.5000 0.6699, 10.0000 5.0000))

```

#### 16. ST\_CreateArc

根据几何体构建弧线(包括圆弧和椭圆弧)。根据给定几何体的最左边、最上边、最右边和最下边四个边界点构建一个方形或矩形,构建出该方形或矩形的内切圆或椭圆。进而根据起始角度和弧线的跨度在圆或椭圆的基础上构造出弧线。

语法如下:

```

FUNCTION ST_CREATEARC(
    GEO SYSGEO.ST_GEOMETRY,
    N INT,
    STARTANG DOUBLE,
    ANGEXTENT DOUBLE
) RETURN SYSGEO.ST_GEOMETRY;

```

#### 参数详解

- GEO 几何体对象。
- N 指定生成圆弧(椭圆弧)的坐标个数,给定参数 N 将生成 N 个坐标点。
- STARTANG 表示截取圆弧、椭圆弧线的起始角度。
- ANGEXTENT 表示弧线的跨度(由此可得到结束角度),它的取值范围为  $0 \sim 2 * \Pi$ , 不在此范围的取值一律视为  $2 * \Pi$ 。

#### 返回值

一个 LINESTRING 空间类型对象。

#### 举例说明

构建一个弧线。

```

declare
geo1 st_geometry;
geo3 st_geometry;
begin
    geo1 = dmgeo.st_geomfromtext( 'polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4296);
    geo3 = dmgeo.ST_CreateArc(geo1, 6, 0, 1.772);

```



```
select dmgeo.st_astext(geo3, 4);

end;

/
```

结果:

```
LINESTRING (10.0000 5.0000, 9.6893 6.7351, 8.7957 8.2546, 7.4304 9.3696, 5.7630
9.9414, 4.0008 9.8991)
```

### 17. ST\_CreateArcPolygon

根据几何体构建扇形（包括椭圆扇形）。根据给定几何体的最左边、最上边、最右边和最下边四个边界点构建一个方形或矩形，构建出该方形或矩形的内切圆或椭圆。进而根据圆心（椭圆心）和半径，以及指定的起始角度和弧线的跨度在圆（或椭圆）的基础上构造出扇形。

语法如下:

---

```
FUNCTION ST_CREATEARCPOLYGON(
    GEO SYSGEO.ST_GEOMETRY,
    N INT,
    STARTANG DOUBLE,
    ANGEXTENT DOUBLE
) RETURN SYSGEO.ST_GEOMETRY;
```

---

#### 参数详解

- GEO 几何体对象。
- N 指定生成扇形（包括椭圆扇形）的坐标个数。由于扇形的特殊性质，给定参数 N，将生成(N + 2)个坐标点，额外的 2 个坐标点为圆心点，它作为起始和结束坐标而被重复一次。
- STARTANG 表示扇弧形的起始角度。
- ANGEXTENT 表示扇弧形的跨度，它的取值范围为  $0 \sim 2 * \Pi$ ，不在此范围的取值一律视为  $2 * \Pi$ 。

#### 返回值

一个 POLYGON 空间类型对象。

#### 举例说明

构建一个扇形。

```
declare
geol st_geometry;
geo3 st_geometry;
begin
```

```

geo1 = dmgeo.st_geomfromtext( 'polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4296);
geo3 = dmgeo.ST_CreateArcPolygon(geo1, 6, 0, 1.772);
select dmgeo.st_astext(geo3, 4);
end;
/

```

结果:

```

POLYGON ((5.0000 5.0000, 10.0000 5.0000, 9.6893 6.7351, 8.7957 8.2546, 7.4304
9.3696, 5.7630 9.9414, 4.0008 9.8991, 5.0000 5.0000))

```

### 18. ST\_CreateAnnulus

根据两个几何体构建一个圆环。首先, 根据给定几何体的最左边、最上边、最右边和最下边四个边界点构建一个方形, 两个方形的中心一定要求一致; 其次, 构建出方形的内切圆; 最后, 根据两个圆构造出一个圆环。

语法如下:

---

```

FUNCTION ST_CREATEANNULUS(
    GEO1 SYSGEO.ST_GEOMETRY,
    GEO2 SYSGEO.ST_GEOMETRY,
    N INT
) RETURN SYSGEO.ST_GEOMETRY;

```

---

#### 参数详解

- GEO1 构建外圆的几何体对象。
- GEO2 构建内圆的几何体对象。
- N 指定生成圆环的坐标个数。由于环有两个圆构造而成, 因此给定参数 N, 实际上会生成 $(2*N + 2)$ 个坐标点, 环上的内外圆上各分布 $(N + 1)$ 个点。

#### 返回值

一个 POLYGON 空间类型对象。

#### 举例说明

构建一个圆环。

```

declare
geo1 st_geometry;
geo2 st_geometry;
geo3 st_geometry;
begin

```

```

geo1 = dmgeo.st_geomfromtext( 'polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4296);
geo2 = dmgeo.st_geomfromtext( 'polygon ((3 3, 3 7, 7 7, 7 3, 3 3))', 4296);
geo3 = dmgeo.ST_CreateAnnulus(geo1,geo2, 6);

select dmgeo.st_astext(geo3, 4);

end;

/

```

结果:

```

POLYGON ((10.0000 5.0000, 7.5000 0.6699, 2.5000 0.6699, 0.0000 5.0000, 2.5000
9.3301, 7.5000 9.3301, 10.0000 5.0000), (7.0000 5.0000, 6.0000 6.7321, 4.0000
6.7321, 3.0000 5.0000, 4.0000 3.2679, 6.0000 3.2679, 7.0000 5.0000))

```

### 19. ST\_CreateAnnularSector

根据两个几何体构建一个扇环。首先，根据给定几何体构造出一个圆环；然后，根据圆心和内外圆半径，以及指定的起始角度和弧线的跨度在圆的基础上构造出扇形。

语法如下：

---

```

FUNCTION ST_CREATEANNULARSECTOR(
    GEO1 SYSGEO.ST_GEOMETRY,
    GEO2 SYSGEO.ST_GEOMETRY,
    N INT,
    STARTANG DOUBLE,
    ANGEXTENT DOUBLE
) RETURN SYSGEO.ST_GEOMETRY;

```

---

#### 参数详解

- GEO1 构建外圆的几何体对象。
- GEO2 构建内圆的几何体对象。
- N 指定生成扇环的坐标个数。由于扇环的特殊性质，给定参数 N，将生成  $(2*N + 1)$  个坐标点，额外的 1 个坐标点，它作为起始和结束坐标而被重复一次。
- STARTANG 表示扇弧形的起始角度。
- ANGEXTENT 表示扇弧形的跨度，它的取值范围为  $0 \sim 2*\Pi$ ，不在此范围的取值一律视为  $2*\Pi$ 。

#### 返回值

一个 POLYGON 空间类型对象。

#### 举例说明

构建一个扇环。

```

declare
geo1 st_geometry;
geo2 st_geometry;
geo3 st_geometry;
begin
    geo1 = dmgeo.st_geomfromtext( 'polygon ((0 0, 0 10, 10 10, 10 0, 0 0))', 4296);
    geo2 = dmgeo.st_geomfromtext( 'polygon ((3 3, 3 7, 7 7, 7 3, 3 3))', 4296);
    geo3 = dmgeo.ST_CreateAnnularSector(geo1,geo2, 6, 0, 1.772);
    select dmgeo.st_astext(geo3, 4);
end;
/

```

结果:

```

POLYGON ((10.0000 5.0000, 9.6893 6.7351, 8.7957 8.2546, 7.4304 9.3696, 5.7630
9.9414, 4.0008 9.8991, 4.6003 6.9597, 5.3052 6.9766, 5.9722 6.7478, 6.5183 6.3018,
6.8757 5.6941, 7.0000 5.0000, 10.0000 5.0000))

```

## 2.2.2 几何信息获取函数

### 1. ST\_Dimension

获取几何体对象的几何维度。

语法如下:

---

```

FUNCTION ST_DIMENSION (
    GEO ST_GEOMETRY
)RETURN INT;

```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的几何维度

### 2. ST\_CoordDim

获取几何体对象的坐标维度。

语法如下:

---

```

FUNCTION ST_COORDDIM (
    GEO ST_GEOMETRY
)RETURN INT;

```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的坐标维度

3. ST\_GeometryType

获取几何体对象的类型。

语法如下：

---

```
FUNCTION ST_GEOMETRYTYPE (
    GEO ST_GEOMETRY
)RETURN VARCHAR(20);
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的类型。

4. ST\_AsText

获取几何体对象的 wkt 格式文本描述信息。

语法如下：

---

```
FUNCTION ST_ASTEXT (
    GEO ST_GEOMETRY
)RETURN CLOB;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的文本表示形式。

5. ST\_AsText

获取指定坐标精度的几何体对象 wkt 格式文本描述信息。

语法如下：

---

```
FUNCTION ST_ASTEXT (
    GEO ST_GEOMETRY,
    V_PRECISION INT
)RETURN CLOB;
```

---

#### 参数详解

- geo 几何体对象。
- v\_precision 坐标精度，即小数位的个数，精度取值范围 1~16，缺省为 16。

#### 返回值

指定精度的几何体对象文本表示形式。

6. ST\_AsBinary

获取几何体对象的 wkb 格式序列化描述信息。

语法如下：

---

```
FUNCTION ST_ASBINARY (  
    GEO ST_GEOMETRY  
)RETURNBLOB;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的序列化表示形式。

#### 7. ST\_SRID

获取几何体对象的空间参考坐标系 ID。

语法如下：

---

```
FUNCTION ST_SRID (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的空间参考坐标系。

#### 8. ST\_IsValid

几何体对象是否是合法对象。

语法如下：

---

```
FUNCTION ST_ISVALID(  
    GEO SYSGEO.ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

0：非法；1：合法。

#### 9. ST\_IsEmpty

获取几何体对象是否为空。

语法如下：

---

```
FUNCTION ST_ISEMPY (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

0: 非空; 1: 空。

10. ST\_ISSimple

判断几何体对象是否为简单几何体。

语法如下:

---

```
FUNCTION ST_ISSIMPLE (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

0: 复杂几何体; 1: 简单几何体。

11. ST\_X

获取几何体对象的 x 坐标。

语法如下:

---

```
FUNCTION ST_X (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象, 必须是 POINT 几何类型。

#### 返回值

几何体对象的 x 坐标值。

12. ST\_Y

获取几何体对象的 y 坐标。

语法如下:

---

```
FUNCTION ST_Y (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象, 必须是 POINT 几何类型。

#### 返回值

几何体对象的 y 坐标值。

13. ST\_MinX

获取几何对象 x 坐标的最小值。

语法如下:

---

```
FUNCTION ST_MINX (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象 x 坐标的最小值。

14. ST\_MAXX

获取几何体对象的 x 坐标的最大值。

语法如下：

---

```
FUNCTION ST_MAXX (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象 x 坐标的最大值。

15. ST\_MINY

获取几何体对象 y 坐标的最小值。

语法如下：

---

```
FUNCTION ST_MINY (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象 y 坐标的最小值。

16. ST\_MAXY

获取几何体对象 y 坐标的最大值。

语法如下：

---

```
FUNCTION ST_MAXY (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象 y 坐标的最大值。



**17. ST\_StartPoint**

获取几何体对象的起始点。

语法如下：

---

```
FUNCTION ST_STARTPOINT (
    GEO ST_GEOMETRY
)RETURN ST_POINT;
```

---

**参数详解**

- geo 几何体对象，必须为 LINESTRING 几何类型。

**返回值**

几何体对象的起始点。

**18. ST\_EndPoint**

获取几何体对象的终点。

语法如下：

---

```
FUNCTION ST_ENDPOINT (
    GEO ST_GEOMETRY
)RETURN ST_POINT;
```

---

**参数详解**

- geo 几何体对象，必须为 LINESTRING 几何类型。

**返回值**

几何体对象的终点。

**19. ST\_IsRing**

获取几何体对象是否是环。

语法如下：

---

```
FUNCTION ST_ISRING (
    GEO ST_GEOMETRY
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象，必须为 LINESTRING 几何类型。

**返回值**

0：不是环；1：是环。

**20. ST\_IsClosed**

获取几何体对象是否闭合。

语法如下：

---

```
FUNCTION ST_ISCLOSED (
    GEO ST_GEOMETRY
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象，必须为 LINESTRING 或 MULTILINESTRING 几何类型。但在处理 multilinestring 时，是对内部的 linestring 逐个调用 st\_isclosed，内部的线与线不是作为一个整体进行 isclosed 判断的。

**返回值**

0：不闭合；1：闭合。

21. ST\_Length

获取几何体对象的长度。

语法如下：

---

```
FUNCTION ST_LENGTH (
    GEO ST_GEOMETRY
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象，必须为 LINESTRING 或 MULTILINESTRING 几何类型。

**返回值**

几何体对象长度

22. ST\_Perimeter

获取几何体对象的周长。

语法如下：

---

```
FUNCTION ST_PERIMETER (
    GEO ST_GEOMETRY
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象。

**返回值**

几何体对象周长。

23. ST\_NumPoints

获取几何体对象的节点数。

语法如下：

---

```
FUNCTION ST_NUMPOINTS (
    GEO ST_GEOMETRY
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象。

**返回值**

几何体对象节点数。

---

#### 24. ST\_PointN

获取几何体对象的第 n 个节点。

语法如下：

---

```
FUNCTION ST_POINTN (  
    GEO      ST_GEOMETRY,  
    N        INT  
)RETURN ST_POINT;
```

---

##### 参数详解

- geo 几何体对象。必须为 LINESTRING 几何类型。

##### 返回值

几何体对象的第 n 个节点。

#### 25. ST\_Centroid

获取几何体对象的中心点。

语法如下：

---

```
FUNCTION ST_CENTROID (  
    GEO ST_GEOMETRY  
)RETURN ST_POINT;
```

---

##### 参数详解

- geo 几何体对象，必须是 POLYGON 或 MULTIPOLYGON 几何类型。

##### 返回值

几何体对象的中心点。

#### 26. ST\_PointOnSurface

获取几何体对象表面上的一点。

语法如下：

---

```
FUNCTION ST_POINTONSURFACE (  
    GEO ST_GEOMETRY  
)RETURN ST_POINT;
```

---

##### 参数详解

- geo 几何体对象，必须是 POLYGON 或 MULTIPOLYGON 几何类型。

##### 返回值

几何体对象表面上的一点。

#### 27. ST\_Area

获取几何体对象的面积。

语法如下：

---

```
FUNCTION ST_AREA (  
    GEO ST_GEOMETRY  
)RETURN DOUBLE;
```

---

---

**参数详解**

- geo 几何体对象，必须是 POLYGON 或 MULTIPOLYGON 几何类型。

**返回值**

几何体对象的面积。

28. ST\_ExteriorRing

获取几何体对象的外环。

语法如下：

---

```
FUNCTION ST_EXTERIORRING (  
    GEO ST_GEOMETRY  
)RETURN ST_LINestring;
```

---

**参数详解**

- geo 几何体对象，必须是 POLYGON 几何类型。

**返回值**

几何体对象的外环。

29. ST\_NumInteriorRing

获取几何体对象的内环数。

语法如下：

---

```
FUNCTION ST_NUMINTERIORRING (  
    GEO ST_GEOMETRY  
)RETURN INT;
```

---

**参数详解**

- geo 几何体对象，必须是 POLYGON 几何类型。

**返回值**

几何体对象的内环数。

30. ST\_Boundary

获取几何体对象的边界。

语法如下：

---

```
FUNCTION ST_BOUNDARY (  
    GEO ST_GEOMETRY  
)RETURN ST_GEOMETRY;
```

---

**参数详解**

- geo 几何体对象。

**返回值**

几何体对象的边界。

31. ST\_Envelope

获取几何体对象的矩形范围。

语法如下：

---

```
FUNCTION ST_ENVELOPE (  
    GEO ST_GEOMETRY  
)RETURN ST_POLYGON;
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象的矩形范围。

#### 32. ST\_InteriorRingN

获取几何体对象的第 n 个内环。

语法如下：

---

```
FUNCTION ST_INTERIORRINGN (  
    GEO ST_GEOMETRY,  
    N INT  
)RETURN ST_LINestring;
```

---

#### 参数详解

- geo 几何体对象，必须是 POLYGON 几何类型。
- n 第几个内环。

#### 返回值

几何体对象的第 n 个内环。

#### 33. ST\_NumGeometries

获取几何对象个数。

语法如下：

---

```
FUNCTION ST_NUMGEOMETRIES (  
    GEO ST_GEOMETRY  
)RETURN INT
```

---

#### 参数详解

- geo 几何体对象。

#### 返回值

几何体对象个数，非 MULTI 类几何体返回 1。

#### 34. ST\_GeometryN

获取第 n 个几何对象。

语法如下：

---

```
FUNCTION ST_GEOMETRYN (  
    GEO ST_GEOMETRY,  
    N INT  
)RETURN ST_GEOMETRY
```

---

**参数详解**

- geo 几何体对象。
- n 第几个几何对象。

**返回值**

第 n 个几何对象。

### 2.2.3 空间关系判断函数

#### 1. ST\_Equals

判断两个几何对象是否相同。

语法如下：

---

```
FUNCTION ST_EQUALS (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN INT;
```

---

**参数详解**

- g1 几何对象。
- g2 几何对象。

**返回值**

0：两个几何对象不相同；1：两个几何对象相同。

#### 2. ST\_Disjoint

判断两个几何对象是否不相交。

语法如下：

---

```
FUNCTION ST_DISJOINT (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN INT;
```

---

**参数详解**

- g1 几何对象。
- g2 几何对象。

**返回值**

0：两个几何对象相交；1：两个几何对象不相交。

#### 3. ST\_Intersects

判断两个几何对象是否相交。

语法如下：

---

```
FUNCTION ST_INTERSECTS (  

```

---

---

```

G1 ST_GEOMETRY,
G2 ST_GEOMETRY
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

0: 两个几何对象不相交; 1: 两个几何对象相交。

#### 4. ST\_TOUCHES

判断两个几何对象是否接触, 即存在边界点相同, 内节点不相交。

语法如下:

---

```

FUNCTION ST_TOUCHES (
    G1 ST_GEOMETRY,
    G2 ST_GEOMETRY
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

0: 两个几何对象不接触; 1: 两个几何对象接触。

#### 5. ST\_CROSSES

判断两个几何对象是否交叉, 存在相同的点 (不是几何对象完全一致)。不能处理面与面之间的交叉情况, 可用于: 点与线, 点与面, 线与面, 线与线。

语法如下:

---

```

FUNCTION ST_CROSSES (
    G1 ST_GEOMETRY,
    G2 ST_GEOMETRY
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

0: 两个几何对象不交叉; 1: 两个几何对象交叉。

#### 举例说明

```

select dmgeo.ST_crosses(dmgeo.ST_MLineFromText('multilinestring ((0 5,4 1),(4 4,
0 0))', 0),dmgeo.ST_MLineFromText('multilinestring ((1 0,1 2),(2 0, 0 4))', 0));
```

结果:

1

#### 6. ST\_WITHIN

判断对象是否完全包含, 对象 g1 是否完全在 g2 的内部。

语法如下:

---

```
FUNCTION ST_WITHIN (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

0: g1 不完全在 g2 内部; 1: g1 完全在 g2 内部。

#### 7. ST\_CONTAINS

判断对象是否包含, g2 不存在点在 g1 的外边界, 且至少存在一个 g2 的内节点在 g1 内部。

语法如下:

---

```
FUNCTION ST_CONTAINS (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

0: g1 不包含 g2; 1: g1 包含 g2。

#### 8. ST\_OVERLAPS

判断两个几何对象是存在重叠, 但并不被对方完全包含。

语法如下:

---

```
FUNCTION ST_OVERLAPS (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。



- g2 几何对象。

#### 返回值

0: 两个几何对象不存在重叠; 1: 两个几何对象存在重叠。

#### 9. ST\_RELATE

判断两个几何对象是否满足 DE-9IM 字符串关系。

语法如下:

---

```
FUNCTION ST_RELATE (
    G1 ST_GEOMETRY,
    G2 ST_GEOMETRY,
    PATTERN CHAR(9)
) RETURN INT;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。
- pattern: DE-9IM 字符串。

#### 返回值

0: g1,g2 不满足 pattern 指定的关系; 1: g1,g2 满足 pattern 指定的关系。

## 2.2.4 几何运算函数

#### 1. ST\_Distance

获取几何对象间的最短距离。

语法如下:

---

```
FUNCTION ST_DISTANCE (
    G1 ST_GEOMETRY,
    G2 ST_GEOMETRY
) RETURN DOUBLE;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

几何对象间最短距离。

#### 2. ST\_Intersection

获取几何对象的交集。

语法如下:

---

```
FUNCTION ST_INTERSECTION (
    G1 ST_GEOMETRY,
```

---

---

```
G2 ST_GEOMETRY  
) RETURN ST_GEOMETRY;
```

---

### 参数详解

- g1 几何对象。
- g2 几何对象。

### 返回值

几何对象的交集。

### 3. ST\_Difference

获取几何对象的差集。

语法如下：

---

```
FUNCTION ST_DIFFERENCE (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN ST_GEOMETRY;
```

---

### 参数详解

- g1 几何对象。
- g2 几何对象。

### 返回值

g1 与 g2 的差集。

### 4. ST\_Union

获取几何对象的并集。

语法如下：

---

```
FUNCTION ST_UNION (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY  
) RETURN ST_GEOMETRY;
```

---

### 参数详解

- g1 几何对象。
- g2 几何对象。

### 返回值

g1 与 g2 的并集。

### 5. ST\_SymDifference

获取几何对象的差异集。

语法如下：

---

```
FUNCTION ST_SYMDIFFERENCE (  
    G1 ST_GEOMETRY,  
    G2 ST_GEOMETRY
```

---

---

```
) RETURN ST_GEOMETRY;
```

---

#### 参数详解

- g1 几何对象。
- g2 几何对象。

#### 返回值

g1 与 g2 的差异集。

#### 6. ST\_Buffer

获取代替几何对象 g1 的几何对象，其到 g1 的距离小于等于 d。

语法如下：

---

```
FUNCTION ST_BUFFER (
    G1 ST_GEOMETRY,
    D DOUBLE
) RETURN ST_GEOMETRY;
```

---

#### 参数详解

- g1 几何对象。
- d 距离。

#### 返回值

几何对象 g1 以 d 为半径的外包几何对象。

#### 7. ST\_ConvexHull

获取几何对象凸壳。

语法如下：

---

```
FUNCTION ST_CONVEXHULL (
    G1 ST_GEOMETRY
) RETURN ST_GEOMETRY;
```

---

#### 参数详解

- g1 几何对象。

#### 返回值

外包几何对象。

## 2.2.5 其他过程函数

#### 1. ST\_ADD\_SPATIAL\_REF

向 SPATIAL\_REF\_SYS 表中插入一条空间参考系信息。

语法如下：

---

```
PROCEDURE ST_ADD_SPATIAL_REF(
    V_SRID INTEGER,
    V_AUTH_NAME VARCHAR,
```

---

---

```
V_AUTH_SRID INTEGER,
V_SRTEXT    VARCHAR
);
```

---

#### 参数详解

- V\_SRID 空间参考系 ID。
- V\_AUTH\_NAME 空间参考系名称。
- V\_AUTH\_SRID 空间参考系代码。
- V\_SRTEXT 空间参考系的文本格式描述。

#### 2. ST\_DEL\_SPATIAL\_REF

向 SPATIAL\_REF\_SYS 表中删除一条空间参考系信息。

语法如下：

---

```
PROCEDURE ST_DEL_SPATIAL_REF(
    V_SRID    INTEGER
);
```

---

#### 参数详解

- V\_SR ID 空间参考系 ID。

#### 3. ST\_geo\_valid\_check

语法如下：

```
FUNCTION ST_GEO_VALID_CHECK(GEO SYSGEO.ST_GEOMETRY) RETURN INT;
```

#### 功能说明

检验空间类型对象数据是否是有效。

#### 参数详解

- geo 空间类型对象。

#### 4. ST\_UPDATE\_SRID

更新列的参考系 SRID，可在 GEOMETRY\_COLUMNS 表中查询到更新后列的 SRID。

语法如下：

---

```
PROCEDURE ST_UPDATE_SRID(
    V_SCH      VARCHAR,
    V_TAB      VARCHAR,
    V_COL      VARCHAR,
    NEW_SRID   INTEGER
);
```

---

#### 参数详解

- V\_SCH 待修改列所在表的模式名。

- V\_TAB 待修改列所在表表名。
- V\_COL 待修改列列名。
- NEW\_SRID 用以更新的新值 SRID。

## 2.3 创建、检测、删除语句

### 2.3.1 SP\_INIT\_GEO\_SYS

创建或删除 DMGEO 系统包。

语法如下：

```
void
SP_INIT_GEO_SYS(
    CREATE_FLAG    int
)
```

#### 参数详解

- CREATE\_FLAG  
为 1 时表示创建 DMGEO 包；为 0 表示删除该系统包。

#### 返回值

无

#### 举例说明

创建 DMGEO 系统包。

```
SP_INIT_GEO_SYS(1);
```



**SP\_INIT\_GEO\_SYS (create\_flag)使用限制：**

- 说明：**
1. 不能在 MPP 全局模式下的存储过程中直接调用，在 MPP LOCAL 模式下可在存储过程中直接调用；
  2. 不能在存储过程中带参数进行动态调用。

### 2.3.2 SF\_CHECK\_GEO\_SYS

系统的 GEO 系统包启用状态检测。

语法如下：

```
int
```

---

SF\_CHECK\_GEO\_SYS ( )

---

### 返回值

0：未启用；1：已启用

### 举例说明

获得 GEO 系统包的启用状态。

```
SELECT SF_CHECK_GEO_SYS;
```

## 3 DBMS\_ADVANCED\_REWRITE 包

在不改变应用程序的前提下，在服务器端将查询语句替换成其他的查询语句执行，此时就需要查询重写（QUERY REWRITE）。DM 使用 DBMS\_ADVANCED\_REWRITE 包实现该功能，不支持安全策略。DM 支持对原始语句中的某些特定词的替换，以及整个语句的替换，不支持递归和变换替换。

### 3.1 相关方法

#### 1. DECLARE\_REWRITE\_EQUIVALENCE

声明一个等价重写规则。

语法如下：

---

```
PROCEDURE DECLARE_REWRITE_EQUIVALENCE (
    NAME          VARCHAR(128),
    SOURCE_STMT    VARCHAR(8188),
    DESTINATION_STMT VARCHAR(8188),
    VALIDATE      BOOLEAN,
    REWRITE_MODE   VARCHAR(16)
);
```

---

#### 参数详解

- NAME  
重写规则的唯一标识，在会话级唯一。
- SOURCE\_STMT  
原始查询语句。
- DESTINATION\_STMT  
目标语句。
- VALIDATE  
是否校验原始语句和目标语句等价。
- REWRITE\_MODE  
包含 4 种：DISABLED，不允许重写；TEXT\_MATCH，文本匹配重写；GENERAL，变换重写；RECURSIVE，递归重写。后面两种暂不支持，可以声明成功，但是不起作用。

#### 使用说明：

(1) 如果一个 SOURCE\_STMT 对应多个 DESTINATION\_STMT，即用户创建很多相同的重写规则，只是名字不同，在重写时只有第一个才有效。也就是说只执行第一个重写规则，

后面的所有重写规则无效。

(2) VALIDATE 置为“TRUE”时，要求 SOURCE\_STMT 和 DESTINATION\_STMT 等价，例如：“SELECT C1 FROM X1 WHERE C1 != 1;”与“SELECT C1 FROM X1 WHERE C1 > 1 OR C1 < 1;”，否则报错。暂不支持为“TRUE”的情况。

(3) 不允许 SOURCE\_STMT 和 DESTINATION\_STMT 完全相同，否则报错：“源语句与目标语句相同”。

(4) SOURCE\_STMT 和 DESTINATION\_STMT 语句必须为查询语句，否则报错：“源语句与目标语句不兼容”。

(5) SOURCE\_STMT 和 DESTINATION\_STMT 语句必须经过语法和语义正确解析，否则报错。如果两个语句中都出错，则报 SOURCE\_STMT 的错，即首先解析 SOURCE\_STMT，如果出错直接报错，不再解析后面的 DESTINATION\_STMT。

(6) REWRITE\_MODE 为“TEXT\_MATCH”时，只允许替换 SOURCE\_STMT 中完全相同的语句，不区分大小写。

(7) SOURCE\_STMT 和 DESTINATION\_STMT 语句不允许包含“{}”字符。

(8) 扩展功能说明：如果 SOURCE\_STMT 为空，DESTINATION\_STMT 格式为“TABLE/[TABLE]”，则可以将当前会话语句中的“TABLE”换成“[TABLE]”，“TABLE”为正则表达式。DM 支持符合 POSIX 标准的正则表达式。如下表所示：

表正则表达式语法

语法	说明	示例
.	匹配任何除换行符之外的单个字符	DA. 匹配“DAMENG”
*	匹配前面的字符 0 次或多次	A*B 匹配“BAT”中的“B”和“ABOUT”中的“AB”
+	匹配前面的字符一次或多次	AC+ 匹配包含字母“A”和至少一个字母“C”的单词，如“RACE”和“ACE”
^	匹配行首	^CAR 仅当单词“CAR”显示为行中的第一组字符时匹配该单词
\$	匹配行尾	END\$ 仅当单词“END”显示为可能位于行尾的最后一组字符时匹配该单词
[ ]	字符集，匹配任何括号间的字符	BE[N-T] 匹配“BETWEEN”中的“BET”、“BENEATH”中的“BEN”和“BESIDE”中的“BES”，但不匹配“BELOW”中的“BEL”
[^ ]	排除字符集。匹配任何不在括号间的字符	BE[^N-T] 匹配“BEFORE”中的“BEF”、“BEHIND”中的“BEH”和“BELOW”中的“BEL”，但是不匹配“BENEATH”中的“BEN”
(表达式)	在表达式加上括号或标签在替换命令中使用	(ABC)+匹配“ABCABCABC”



	匹配 OR 符号 ( ) 之前或之后的表达式。最常用在分组中	(SPONGE MUD) BATH 匹配 "SPONGE BATH" 和 "MUD BATH"
\	按原义匹配反斜杠 (\) 之后的字符。这使您可以查找正则表达式表示法中使用的字符, 如 { 和 ^	\^ 搜索 ^ 字符
{}	匹配以带括号的表达式标记的文本	zo{1} 匹配 "ALONZO1" 和 "GONZO1" 中的 "ZO1", 但不匹配 "ZONE" 中的 "ZO"
[[:alpha:]]	表示任意字母 \w ([a-z]+)   ([A-Z]+)	
[[:digit:]]	表示任意数字 \d ([0-9]+)	
[[:lower:]]	表示任意小写字母 ([a-z]+)	
[[:alnum:]]	表示任意字母和数字 ([a-z0-9]+)	
[[:space:]]	表示任意空格	
[[:upper:]]	表示任意大写字母 ([A-Z]+)	
[[:punct:]]	表示任意标点符号	
[[:xdigit:]]	表示任意 16 进制数 ([0-9a-fA-F]+)	

## 2. ALTER\_REWRITE\_EQUIVALENCE

修改重写模式。

语法如下:

```
PROCEDURE ALTER_REWRITE_EQUIVALENCE (
    NAME          CHAR(128),
    REWRITE_MODE   VARCHAR(16)
);
```

### 参数详解

- NAME

重写规则的唯一标识。

- REWRITE\_MODE

包含 4 种, 同上。

## 3. VALIDATE\_REWRITE\_EQUIVALENCE

校验重写规则是否等效, 暂不支持。提供该函数, 但是不进行校验。

语法如下:

```
VALIDATE_REWRITE_EQUIVALENCE (
    NAME CHAR(128)
```

---

);

### 参数详解

- NAME

重写规则的唯一标识。

4. DROP\_REWRITE\_EQUIVALENCE。

删除重写规则。

语法如下：

---

```
PROCEDURE DROP_REWRITE_EQUIVALENCE (
    NAME    CHAR(128)
);
```

---

### 参数详解

- name

重写规则的唯一标识。

## 3.2 字典信息

所有的重写规则信息都保存到系统表 SYS\_REWRITE\_EQUIVALENCES 中，该系统表的结构如下表所示。所有的用户创建的语句重写信息都保存到该表中。

表系统表 SYS\_REWRITE\_EQUIVALENCES 结构

序号	列名	类型	是否允许为空	说明
1	OWNER	VARCHAR(128)	N	重写规则的拥有者
2	NAME	VARCHAR(128)	N	重写规则名
3	SOURCE_STMT	VARCHAR(3200)	Y	原始语句
4	DESTINATION_STMT	VARCHAR(3200)	Y	目标语句
5	REWRITE_MODE	VARCHAR(16)	Y	重写模式
6	RESV1	INTEGER	Y	保留字段
7	RESV2	VARCHAR(128)	Y	保留字段

主键 (OWNER, NAME)。

通过查询语句可以获得重写规则的信息：

```
SELECT * FROM SYS_REWRITE_EQUIVALENCES;
```

## 3.3 使用说明

### 1. 设置会话标记

每个会话对应一个 QUERY\_REWRITE\_INTEGRITY 标记，该标记只在会话期间起作用，不保存到字典信息中，如果会话结束，则该标记也无效。会话重建后，该标记也需要重新赋

值才有效。

QUERY\_REWRITE\_INTEGRITY 标记用于表示该会话是否可以执行查询语句重写，默认值为“ENFORCED”，另外两个值为：“TRUSTED”和“STALE\_TOLERATED”。默认不允许语句重写；DM 未对后两个参数进行严格区分，一般设置后两个参数都允许查询重写。

语法格式：

---

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
```

---

#### 参数详解

- TRUSTED 表示在保证物化视图数据是正确时才使用该视图替换。
- STALE\_TOLERATED 表示能容忍错误的数数据，即可以使用不正确的物化视图执行重写。

#### 2. 重写规则的作用域

重写规则只允许在当前会话中执行，禁止跨模式使用其他用户指定的重写规则。SYSDBA 也不可以使用其他用户的重写规则替换。

#### 3. 回滚

执行方法 DECLARE\_REWRITE\_EQUIVALENCE 后，将自动提交，不能执行回滚。在该方法之前的所有操作也自动提交。

#### 4. 自动重写

自动重写，指的是用户在执行任何查询语句时，可以自动调用重写规则执行重写，而不需要设置会话标记来控制是否执行查询重写。SYSDBA 可以设置某些用户自动重写标记，设置函数为：SP\_USER\_SET\_AUTO\_REWRITE\_FLAG。该标记和会话标记中有一个有效，则允许查询重写。

语法格式：

---

```
SP_USER_SET_AUTO_REWRITE_FLAG (
    'USER_NAME',
    VALUE
);
```

---

#### 参数详解

- USER\_NAME 用户名。
- VALUE0 表示不允许自动重写，1 表示允许自动重写。

#### 5. 查询重写时机

查询重写的时机主要发生在如下几个地方：

- (1) 一般的用户查询，例如 `SELECT COUNT(*) FROM T1;`
- (2) 非动态的查询执行时，例如存储过程中重写：`SELECT COUNT(*) FROM T1 WHERE C1 > 1;`但是以下语句不可以重写：`I INT := 1; SELECT COUNT(*) FROM T1 WHERE C1 > I;`
- (3) 视图中的查询重写，不支持递归替换。例如：`CREATE VIEW V1 AS SELECT`

COUNT(\*) FROM T1;如果“SELECT COUNT(\*) FROM T1;”存在重写规则，也可以重写。

## 6. 关闭重用执行计划功能

使用时，可以在 DM.INI 中将重用执行计划标记（USE\_PLN\_POOL）置为 0。否则，先执行过查询语句后，再设置查询规则，然后对比两次的查询结果会相同，即未被重写。如果不关闭，只能等到设置过查询规则之后，才能执行查询。

## 3.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_ADVANCED_REWRITE');
```

例 REWRITE 包的应用，注意 INI 参数 USE\_PLN\_POOL 应置为 0。

```
DROP TABLE X1;
CREATE TABLE X1(C1 INT, C2 CHAR(20));
INSERT INTO X1 VALUES(12, 'TEST12');
INSERT INTO X1 VALUES(13, 'TEST13');
INSERT INTO X1 VALUES(14, 'TEST14');
INSERT INTO X1 VALUES(15, 'TEST15');
CREATE TABLE X2(D1 INT, D2 CHAR(30));
COMMIT;
BEGIN
DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE(
'TEST_REWRITE',
'SELECT COUNT(*) FROM X1',           --原始查询语句
'SELECT COUNT(*) FROM X2',           --目标语句
FALSE,
'TEXT_MATCH'
);
END;
/
SELECT COUNT(*) FROM X1;               --执行结果 4

ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;  --默认为'ENFORCED'
SELECT COUNT(*) FROM X1;               --执行结果 0

DBMS_ADVANCED_REWRITE.DROP_REWRITE_EQUIVALENCE ('TEST_REWRITE');
--DROP REWRITE

BEGIN
  DBMS_ADVANCED_REWRITE.DECLARE_REWRITE_EQUIVALENCE(
    'TEST_REWRITE',
    NULL,
    --为空
```

```
'FRO./FROM',                                --使用正则表达式
FALSE,
'TEXT_MATCH'
);
END;
/
SELECT COUNT(*) FROK X1;                      --执行结果 4
```

## 4 DBMS\_ALERT 包

DBMS\_ALERT 系统包是为了在 DM 上兼容 oracle 的 DBMS\_ALERT 系统包。用于生成并传递数据库预警信息，当发生特定数据库事件时能够将预警信息传递给应用程序。

达梦还提供了 DBMS\_ALERT\_INFO 视图来实现跟 ORACLE 类似的功能，查看注册过的预警事件。

DM MPP 环境和 DM DSC 环境下不支持 DBMS\_ALERT 包。

### 4.1 相关方法

#### 1. DBMS\_ALERT.REGISTER

为当前会话注册一个预警事件，本操作会提交当前事务。

语法如下：

---

```
DBMS_ALERT.REGISTER (
    NAME      IN  VARCHAR(30)
)
```

---

#### 参数详解

##### ● NAME

输入参数，预警事件名。

#### 2. DBMS\_ALERT.REMOVE

删除当前会话的一个预警事件，如果该预警事件正在 DBMS\_ALERT.WAITONE 或者 DBMS\_ALERT.WAITANY 等待，则被阻塞，直到 DBMS\_ALERT.WAITONE 或者 DBMS\_ALERT.WAITANY 结束。本操作会提交当前事务。

语法如下：

---

```
DBMS_ALERT.REMOVE (
    NAME      IN  VARCHAR(30)
)
```

---

#### 参数详解

##### ● NAME

输入参数，预警事件名。

#### 3. DBMS\_ALERT.REMOVEALL

删除当前会话下所有的预警事件，如果该预警事件正在 DBMS\_ALERT.WAITONE 或者 DBMS\_ALERT.WAITANY 等待，则被阻塞，直到 DBMS\_ALERT.WAITONE 或者 DBMS\_ALERT.WAITANY 结束。本操作会提交当前事务。

语法如下：

---

```
DBMS_ALERT. REMOVEALL ( )
```

---

#### 4. DBMS\_ALERT.SIGNAL

给所有名为 name 的预警事件发送消息，当前事务提交时生效。

语法如下：

---

```
DBMS_ALERT. SIGNAL (
    NAME      IN  VARCHAR(30),
    MESSAGE   IN  VARCHAR(1800)
)
```

---

#### 参数详解

- NAME

输入参数，预警事件名。

- MESSAGE

输入参数，待发送的消息。

#### 5. DBMS\_ALERT.SET\_DEFAULTS

设置轮询时间，在 DM 中无用，只为兼容 oracle。

语法如下：

---

```
DBMS_ALERT. SET_DEFAULTS (
    SENSITIVITY IN INT
)
```

---

#### 参数详解

- SENSITIVITY

输入参数，轮询时间间隔，单位秒。参数为 NULL，则使用默认值 5s。

#### 6. DBMS\_ALERT.WAITONE

预警事件上等待预警信号，获得 DBMS\_ALERT.SIGNAL 发送的消息内容。本操作会提交当前事务。

语法如下：

---

```
DBMS_ALERT. WAITONE (
    NAME      IN  VARCHAR(30),
    MESSAGE   OUT VARCHAR(1800),
    STATUS    OUT INT,
    TIMEOUT   IN  INT DEFAULT 86400000
)
```

---

#### 参数详解

- NAME

输入参数，预警事件名。

- MESSAGE

输出参数，获得 DBMS\_ALERT.SIGNAL 发送的消息。

- STATUS

输出参数，表示 WAITONE 是否成功。如果 DBMS\_ALERT.WAITONE 传入的 NAME 是已经注册过的预警，则 0 表示预警发生；1 表示预警等待超时。

- TIMEOUT

输入参数，WAITONE 的等待超时时间，单位为秒，默认值为 86400000 秒。

7. DBMS\_ALERT.WAITANY

语法如下：

---

```
DBMS_ALERT.WAITANY (
    NAME      OUT VARCHAR(30),
    MESSAGE OUT VARCHAR(1800),
    STATUS    OUT INT,
    TIMEOUT IN  INT DEFAULT 86400000
)
```

---

### 参数详解

同 DBMS\_ALERT.WAITONE。

### 功能说明：

等待当前会话任意一个预警信号，获得 DBMS\_ALERT.SIGNAL 发送的消息内容。本操作会提交当前事务。会话上没有预警事件时，报错没有注册预警事件。

## 4.2 举例说明

使用包内的过程和函数之前，如果还未创建过例子中的系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_ALERT');
```

例 1 为当前会话注册一个名为 A1 的预警事件，给预警事件上等待预警信号，获得 DBMS\_ALERT.SIGNAL 发送的消息内容，最后删除当前会话的一个预警事件。同时，会话 1 存活期间，可以通过 DBMS\_ALERT\_INFO 视图查看到 A1 预警事件。

会话 1:

```
CREATE OR REPLACE PROCEDURE WAIT1 IS
    msg VARCHAR(1800);
    statu INTEGER;
BEGIN
    dbms_alert.register('A1');
    dbms_alert.waitone('A1', msg, statu, 39);
    print 'Msg: ' || msg || ' Statu: ' || statu;
    dbms_alert.remove('A1');
```



```
END WAIT1;  
/  
EXEC WAIT1;
```

同时会话 2 执行:

```
CREATE OR REPLACE PROCEDURE SET_ALERT1  
is  
BEGIN  
dbms_alert.signal('A1', 'this is a sig');  
COMMIT;  
END SET_ALERT1;  
/  
EXEC SET_ALERT1;
```

会话 1 存活期间, 在会话 2 上, 通过 DBMS\_ALERT\_INFO 视图查看 A1。

```
select * from DBMS_ALERT_INFO;
```

结果为:

NAME	SID	CHANGED	MESSAGE
A1	83034240	Y	this is a sig

## 5 DBMS\_BINARY 包

DBMS\_BINARY 系统包用于读写二进制流，实现从一个二进制流指定位置开始对基本数据类型的读写，包括 CHAR、VARCHAR、TINYINT、SMALLINT、INT、BIGINT、FLOAT、DOUBLE 数据类型。

### 5.1 相关方法

DBMS\_BINARY 包所支持的 8 个过程和 8 个函数，分别用来在二进制流中存取数据。这些过程和函数通过调用相应系统内部函数来实现存取数据的过程。如下详细介绍各过程和函数：

#### 1. BINARY\_GET\_CHAR

返回从二进制流 VB 中偏移 OFFSET 开始的一个 CHAR 类型数据。

语法如下：

---

```
FUNCTION BINARY_GET_CHAR(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

#### 返回值

CHAR 类型数据。

#### 2. BINARY\_GET\_VARCHAR

返回从二进制流 VB 中偏移 OFFSET 开始的一个长度为 LENGTH 的 VARCHAR 类型数据。

语法如下：

---

```
FUNCTION BINARY_GET_VARCHAR(  
    VB VARBINARY,  
    OFFSET INT,  
    LENGTH INT  
) ;
```

---

#### 返回值

VARCHAR 类型数据

#### 3. BINARY\_GET\_TINYINT

返回从二进制流 VB 中偏移 OFFSET 开始的一个 TINYINT 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_TINYINT(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

---

**返回值**

TINYINT 类型数据。

**4. BINARY\_GET\_SMALLINT**

返回从二进制流 VB 中偏移 OFFSET 开始的一个 SMALLINT 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_SMALLINT(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

**返回值**

SMALLINT 类型数据。

**5. BINARY\_GET\_INT**

返回从二进制流 VB 中偏移 OFFSET 开始的一个 INT 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_INT(  
    VB VARBINARY,  
    OFFSET INT) ;
```

---

**返回值**

INT 类型数据。

**6. BINARY\_GET\_BIGINT**

返回从二进制流 VB 中偏移 OFFSET 开始的一个 BIGINT 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_BIGINT(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

**返回值**

BIGINT 类型数据。

**7. BINARY\_GET\_FLOAT**

返回从二进制流 VB 中偏移 OFFSET 开始的一个 FLOAT 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_FLOAT(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

**返回值**

FLOAT 类型数据。

**8. BINARY\_GET\_DOUBLE**

返回从二进制流 VB 中偏移 OFFSET 开始的一个 DOUBLE 型数据。

语法如下：

---

```
FUNCTION BINARY_GET_DOUBLE(  
    VB VARBINARY,  
    OFFSET INT  
) ;
```

---

### 返回值

DOUBLE 类型数据。

### 9. BINARY\_SET\_CHAR

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 CHAR 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_CHAR(  
    VB IN OUT VARBINARY,  
    OFFSET INT, VALUE CHAR  
) ;
```

---

### 10. BINARY\_SET\_VARCHAR

从二进制流 VB 中偏移 OFFSET 开始的位置写入 VARCHAR 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_VARCHAR(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE VARCHAR  
) ;
```

---

### 11. BINARY\_SET\_TINYINT

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 TINYINT 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_TINYINT(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE TINYINT  
) ;
```

---

### 12. BINARY\_SET\_SMALLINT

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 SMALLINT 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_SMALLINT(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE SMALLINT  
) ;
```

---

### 13. BINARY\_SET\_INT

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 INT 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_INT(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE INT  
);
```

---

#### 14. BINARY\_SET\_BIGINT

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 BIGINT 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_BIGINT(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE BIGINT  
);
```

---

#### 15. BINARY\_SET\_FLOAT

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 FLOAT 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_FLOAT(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE FLOAT  
);
```

---

#### 16. BINARY\_SET\_DOUBLE

从二进制流 VB 中偏移 OFFSET 开始的位置写入一个 DOUBLE 型数据。

语法如下：

---

```
PROCEDURE BINARY_SET_DOUBLE(  
    VB IN OUT VARBINARY,  
    OFFSET INT,  
    VALUE DOUBLE  
);
```

---

### 参数详解

因为 DBMS\_BINARY 包所支持的 8 个过程和 8 个函数参数意义相同，如下统一说明：

- VB  
用户输入的二进制流。
- OFFSET  
向二进制流中写入数据或者从二进制流取出数据时的偏移量。
- LENGTH

表示从二进制流指定偏移位置开始取多少个字符。

- VALUE

待写入的数据。

## 5.2 错误处理

### 1. 空值处理

如果输入参数中存在空值，则结果返回空值。

例如：

```
BINARY_GET_CHAR (VB VARBINARY, OFFSET INT)
SELECT DBMS_BINARY.BINARY_GET_CHAR ('ABCDEF ', NULL);
```

结果：NULL

说明：偏移 OFFSET 输入空值，在函数内部处理的时候返回 NULL。

### 2. 非法参数

如果偏移量为负数或者偏移量加上所取数据类型的长度大于二进制流的长度，报非法参数错误 EC\_RN\_INVALID\_ARG\_DATA，错误码：-6803。

例如：

```
BINARY_GET_CHAR (VB VARBINARY, OFFSET INT)
SELECT DBMS_BINARY.BINARY_GET_CHAR ('ABCDEF', -11);
```

结果：输入参数非法。

说明：输入的二进制流的长度为-11，偏移量为负数不合法，提示非法参数错误。

## 5.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_BINARY');
```

例 1 使用 BINARY\_SET\_TINYINT 过程写入一个 TINYINT 类型数据，并使用 BINARY\_GET\_TINYINT 函数输出该 TINYINT 类型数据。

```
DECLARE
BIN VARBINARY(50);
VAL TINYINT;
BEGIN
BIN = 'ABCDEF87';
VAL = 127;
DBMS_BINARY.BINARY_SET_TINYINT(BIN,0,VAL);
VAL = DBMS_BINARY.BINARY_GET_TINYINT(BIN,0);
PRINT VAL;
END;
```

```
/
```

结果:

```
127
```

例 2 使用 `BINARY_SET_VARCHAR` 过程写入一个 `VARCHAR` 类型数据，并使用 `BINARY_GET_VARCHAR` 函数输出该 `VARCHAR` 类型数据。

```
DECLARE
BIN VARBINARY(50);
VAL VARCHAR;
BEGIN
BIN = 'ABCDEF87';
DBMS_BINARY.BINARY_SET_VARCHAR(BIN,0,'AAAA');
VAL = DBMS_BINARY.BINARY_GET_VARCHAR(BIN,0,4);
PRINT VAL;
END;
/
```

结果:

```
AAAA
```

## 6 DBMS\_JOB 包

为了兼容 ORACLE 定时任务的创建，按指定的时间或间隔执行用户定义的作业。达梦提供了 DBMS\_JOB 包以及 DBA\_JOBS、USER\_JOBS 视图来实现跟 ORACLE 类似的功能。

### 6.1 相关方法

#### 1. BROKEN 过程

更新一个已提交的工作的状态，典型地是用来把一个已过期工作标记为未过期工作；或者把一个未过期的工作设置为何时过期。

语法如下：

---

```
PROCEDURE    BROKEN(  
    JOB IN    INTEGER,  
    BROKEN IN  BOOLEAN,  
    NEXT_DATE IN  DATETIME :=SYSDATE  
);
```

---

#### 参数详解

- job 工作号，唯一标识一个特定工作。
- broken

指示此工作是否将标记为过期，TRUE 或 FALSE。TRUE 表明过期，而 FALSE 表明未过期。

- next\_date 指示在什么时候此工作将再次运行。此参数缺省值为当前日期和时间。

#### 2. CHANGE 过程

用来改变指定工作的设置。

语法如下：

---

```
PROCEDURE    CHANGE (  
    JOB IN    INTEGER,  
    WHAT    IN  VARCHAR2,  
    NEXT_DATE IN  DATETIME,  
    INTERVAL IN  VARCHAR2  
);
```

---

#### 参数详解

- job 工作号，唯一标识一个特定工作。
- what 是由此工作运行的一块 PL/SQL 代码块。
- next\_date 指示何时此工作将被执行。
- interval 指示一个工作重执行的频度。



**注意：**

由于 CHANGE 是保留字，调用该过程时需要在过程名上加双引号。

**3. INTERVAL 过程**

用来显式地设置重执行一个工作之间的时间间隔数。

语法如下：

---

```
PROCEDURE   INTERVAL (
    JOB IN   INTEGER,
    INTERVAL   IN   VARCHAR2
);
```

---

**参数详解**

- job 工作号，唯一标识一个特定工作。
- interval

指示一个工作重执行的频度，该频度是一个日期表达式字符串，以当天 0 点 0 分为起点和该日期表达式字符串计算出来的日期时间之差，以分钟为最小间隔单位。例如 'sysdate + 1/1440'，则表示间隔 1 分钟，'sysdate + 1/24' 则表示间隔 1 小时，'sysdate + 1' 则表示间隔 1 天，最多允许间隔 100 天。达梦的 INTERVAL 参数不支持以周，月，季等单位方式指定间隔。

**注意：**

由于 INTERVAL 是保留字，调用该过程时需要在过程名上加双引号。

**4. ISUBMIT 过程**

用来用特定的工作号提交一个工作。这个过程与 Submit() 过程的唯一区别在于此 job 参数作为 IN 型参数传递且包括一个由开发者提供的工作号。如果提供的工作号已被使用，将产生一个错误。

语法如下：

---

```
PROCEDURE   ISUBMIT (
    JOB IN   INTEGER,
    WHAT   IN   VARCHAR2,
    NEXT_DATE   IN   DATETIME,
    INTERVAL   IN   VARCHAR2,
    NO_PARSE   IN   BOOLEAN:=FALSE
);
```

---

**参数详解**

- job 是由用户指定的工作编号，不能为负数，且必须是不存在的工作编号。
- what 是将被执行的 PL/SQL 代码块。
- next\_date 指示何时将运行这个工作。
- interval 何时这个工作将被重执行。

---

- `no_parse`

指示此工作在提交时或执行时是否应进行语法分析，TRUE 指示此 PL/SQL 代码在它第一次执行时应进行语法分析，而 FALSE 指示本 PL/SQL 代码应立即进行语法分析。

- 5. NEXT\_DATE 过程

用来显式地设定一个工作的执行时间。

语法如下：

---

```
PROCEDURE NEXT_DATE(  
    JOB IN INTEGER,  
    NEXT_DATE IN DATETIME  
);
```

---

**参数详解**

- `job` 标识一个已存在的工作。
- `next_date` 指示了此工作应被执行的日期与时间。

- 6. REMOVE 过程

用来删除一个已计划运行的工作。已正在运行的工作不能由调用过程删除。

语法如下：

---

```
PROCEDURE REMOVE(  
    JOB IN INTEGER  
);
```

---

**参数详解**

- `job` 唯一地标识一个工作。这个参数的值是由为此工作调用 `Submit()` 过程返回的 `job` 参数的值。

- 7. RUN 过程

用来立即执行一个指定的工作。需要说明的是，RUN 过程的调用不影响工作的计划运行时间。

语法如下：

---

```
PROCEDURE RUN(  
    JOB IN INTEGER  
);
```

---

**参数详解**

`job` 标识将被立即执行的工作。

- 8. SUBMIT 过程

使用 `Submit()` 过程，工作被正常地计划好。

语法如下：

---

```
PROCEDURE SUBMIT (  
    JOB OUT INTEGER,
```

---

```

WHAT    IN  VARCHAR2,
NEXT_DATE  IN  DATE,
INTERVAL   IN  VARCHAR2,
NO_PARSE   IN  BOOLEAN:=FALSE
);

```

### 参数详解

- job 是由 Submit() 过程返回的工作编号。这个值用来唯一标识一个工作。
- what 是将被执行的 PL/SQL 代码块。
- next\_date 指识何时将运行这个工作。
- interval 何时这个工作将被重执行。
- no\_parse  
指示此工作在提交时或执行时是否应进行语法分析，TRUE 指示此 PL/SQL 代码在它第一次执行时应进行语法分析，而 FALSE 指示本 PL/SQL 代码应立即进行语法分析。

#### 9. WHAT 过程

允许在工作执行时重新设置此正在运行的命令。

语法如下：

```

PROCEDURE  WHAT (
    JOB IN  INTEGER,
    WHAT IN OUT VARCHAR2
);

```

### 参数详解

- job 标识一个存在的工作。
- what  
指示将被执行的新的 PL/SQL 代码，如果传入的 what 参数值为空串，那将返回该任务原来的 PL/SQL 代码。

## 6.2 DBMS\_JOB 视图

### 1. DBA\_JOBS 视图

描述数据库中所有的 JOB。

序号	列	数据类型	说明
1	JOB	INTEGER	工作的唯一标识号
2	LOG_USER	VARCHAR(8188)	提交工作的用户
3	PRIV_USER	VARCHAR(8188)	赋予工作权限的用户

4	SCHEMA_USER	VARCHAR(8188)	对工作做语法分析的用户模式
5	LAST_DATE	DATE	最后一次成功运行工作的时间
6	LAST_SEC	TIME(0)	HH:MM:SS 格式的 LAST_DATE 的时间
7	THIS_DATE	DATE	正在运行工作的开始时间
8	THIS_SEC	TIME(0)	HH:MM:SS 格式的 THIS_DATE 的时间
9	NEXT_DATE	DATE	下一次定时任务运行的时间
10	NEXT_SEC	TIME(0)	HH:MM:SS 格式的 NEXT_DATE 的时间
11	TOTAL_TIME	FLOAT	该 job 运行所需的总时间
12	BROKEN	VARCHAR(1)	是否工作中断标识参数, Y 是, N 否
13	INTERVAL	INTEGER	用于计算下一运行时间的表达式
14	FAILURES	BIGINT	工作运行连续没有成功的次数
15	WHAT	VARCHAR(1800)	执行工作的 PL/SQL 块
16	NLS_ENV	VARCHAR(1)	工作运行的 NLS 会话设置
17	MISC_ENV	VARCHAR(1)	工作运行的其他一些会话参数
18	INSTANCE	INTEGER	能够运行或正在运行工作的实例的 id 号。 单节点上默认值为 1, 如果是 mpp 或 rac 环境。 默认值为实例序号加 1。

## 2.USER\_JOBS 视图

描述当前用户所拥有的 JOB。视图结构与 DBA\_JOBS 相同。

## 3.DBA\_JOBS\_RUNNING

显示实例中所有正在执行的作业。

序号	列	数据类型	说明
1	SID	BIGINT	作业执行的 SESSION 的 ID
2	JOB	INTEGER	作业号
3	FAILURES	INTEGER	作业自上一次成功以来的失败次数, 暂不支持
4	LAST_DATE	DATE	最后一次成功运行工作的时间
5	LAST_SEC	TIME(0)	HH:MM:SS 格式的 LAST_DATE 的时间
6	THIS_DATE	DATE	本次运行的开始时间
7	THIS_SEC	TIME(0)	HH:MM:SS 格式的 THIS_DATE 的时间
8	INSTANCE	INTEGER	能够运行或正在运行作业的实例的 ID 号。 单节点上默认值为 1, 如果是 DM MPP 或 DMDS 环境, 默认值

			为实例序号加 1
--	--	--	----------

## 6.3 创建、删除语句

创建或删除 DBMS\_JOB 系统包。

语法如下：

```
void
SP_INIT_JOB_SYS(
    CREATE_FLAG    int
)
```

### 参数详解

- CREATE\_FLAG

为 1 时表示创建 DBMS\_JOB 包；为 0 表示删除该系统包。

### 返回值

无

### 举例说明

创建 DBMS\_JOB 系统包。

```
SP_INIT_JOB_SYS(1);
```

## 6.4 举例说明

用户在使用 DBMS\_JOB 包之前，需要提前调用系统过程 SP\_INIT\_JOB\_SYS(1)，创建好调试所需要的包，就可以使用 DBMS\_JOB 来调用 PL/SQL 中的过程或函数了。

```
SP_INIT_JOB_SYS(1);
```

创建测试表

```
create table a(a datetime);
```

创建一个自定义过程

```
create or replace procedure test as
begin
insert into a values(sysdate);
end;
/
```

创建 JOB

```
begin
dbms_job.isubmit(1,'test;',sysdate,'sysdate+1/1440');--每天 1440 分钟，即一分钟运行
test 过程一次
    commit;
end;
/
```

查看 JOB 运行结果

```
select  to_char(a,'yyyy/mm/dd  hh24:mi:ss')  时间  from  a;
```

结果如下：

时间

```
-----
2013/09/04  13:22:44
2013/09/04  13:23:44
2013/09/04  13:24:44
2013/09/04  13:25:44
```

删除 JOB

```
begin
    dbms_job.remove(1);
end;
/
```

PL/SQL 过程已成功完成。

## 7 DBMS\_LOB 包

兼容 ORACLE 的 DBMS\_LOB 包，用于对大对象（BLOB、CLOB）进行操作所提供的一系列方法的集合。

### 7.1 相关方法

#### 1. APPEND

追加一个源 LOB 数据到一个目标的 LOB 数据后面。

语法如下：

---

```
PROCEDURE APPEND(  
    DEST_LOB IN OUT BLOB,  
    SRC_LOB IN BLOB  
);  
  
PROCEDURE APPEND(  
    DEST_LOB IN OUT CLOB,  
    SRC_LOB IN CLOB  
);
```

---

#### 参数详解

- DEST\_LOB 目标临时大对象
- SRC\_LOB 源临时大对象

#### 2. CLOSE

关闭一个之前打开的 LOB 或 BFILE 对象。

语法如下：

---

```
PROCEDURE CLOSE(  
    LOB_LOC IN OUT BLOB  
);  
  
PROCEDURE CLOSE(  
    LOB_LOC IN OUT CLOB,  
);  
  
PROCEDURE CLOSE(  
    FILE_LOC IN OUT BFILE  
);
```

---

#### 参数详解

- LOB\_LOCB LOB 或 CLOB 类型的大对象
- FILE\_LOCB FILE 类型对象

#### 3. COMPARE

比较 2 个 LOB 的数据，返回一个结果值。

语法如下：

---

```

FUNCTION COMPARE(
    LOB_1 IN BLOB,
    LOB_2 IN BLOB,
    AMOUNT IN BIGINT:= DBMS_LOB.LOBMAXSIZE,
    OFFSET_1 IN INTEGER := 1,
    OFFSET_2 IN INTEGER := 1
) RETURN INTEGER;
FUNCTION COMPARE(
    LOB_1 IN CLOB,
    LOB_2 IN CLOB,
    AMOUNT IN BIGINT:= DBMS_LOB.LOBMAXSIZE,
    OFFSET_1 IN INTEGER := 1,
    OFFSET_2 IN INTEGER := 1
) RETURN INTEGER;
FUNCTION COMPARE(
    LOB_1 IN BFILE,
    LOB_2 IN BFILE,
    AMOUNT IN BIGINT,
    OFFSET_1 IN BIGINT:= 1,
    OFFSET_2 IN BIGINT:= 1
) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_1 第一个临时大对象
- LOB\_2 第二个临时大对象
- AMOUNT

需要比较的总的长度，BLOB 为字节长度，CLOB 为字符长度，默认为 2147483647。

LOBMAXSIZE 是包内变量，为 BIGINT 类型，默认值 9223372036854775807。

- OFFSET\_1 第一个临时大对象的起始偏移
- OFFSET\_2 第二个临时大对象的起始偏移

#### 返回值

0: 表示相等；

-1: 表示 lob\_1<lob\_2；

1: 表示 lob\_1>lob\_2

NULL: 表示 amount<1, amount>LOBMAXSIZE, offset\_1<1, offset\_2<1, offset\_1>LOBMAXSIZE, offset\_2>LOBMAXSIZE。

#### 4. COPY



拷贝指定长度的源 LOB 数据插入到目标 LOB。

语法如下：

---

```
PROCEDURE COPY(
    DEST_LOB IN OUT BLOB,
    SRC_LOB IN BLOB,
    AMOUNT IN INTEGER,
    DEST_OFFSET IN INTEGER := 1,
    SRC_OFFSET IN INTEGER := 1
);

PROCEDURE COPY(
    DEST_LOB IN OUT CLOB,
    SRC_LOB IN CLOB,
    AMOUNT IN INTEGER,
    DEST_OFFSET IN INTEGER := 1,
    SRC_OFFSET IN INTEGER := 1
);
```

---

#### 参数详解

- DEST\_LOB 目标临时大对象。
- SRC\_LOB 源临时大对象。
- AMOUNT 需要拷贝的字节或字符总数，BLOB 为字节，CLOB 为字符。
- DEST\_OFFSET  
写入的起始偏移，BLOB 为字节，CLOB 为字符，如果写入偏移大于原字段的长度，将自动填充原长度到指定偏移间的数据，BLOB 填充 0，CLOB 填充空格。
- SRC\_OFFSET 读取的起始偏移，BLOB 为字节，CLOB 为字符。

#### 5. CREATETEMPORARY

创建一个临时的 LOB 对象，并存储在临时的表空间里。

语法如下：

---

```
PROCEDURE CREATETEMPORARY(
    LOB_LOC IN OUT BLOB,
    CACHE IN BOOLEAN,
    DUR IN PLS_INTEGER := DBMS_LOB.SESSION
);

PROCEDURE CREATETEMPORARY(
    LOB_LOC IN OUT CLOB,
    CACHE IN BOOLEAN,
    DUR IN PLS_INTEGER := 10
);
```

---

#### 参数详解

- LOB\_LOC 目标临时大对象。

- CACHE 只支持 TRUE。
- DUR 存活周期，只支持 DBMS\_LOB.SESSION。

## 6. ERASE

擦除指定偏移开始的指定长度 LOB 数据。BLOB 用 0，CLOB 用空格代替原有数据。

语法如下：

---

```
PROCEDURE ERASE(  
    LOB_LOC IN OUT BLOB,  
    AMOUNT IN OUT INTEGER,  
    OFFSET IN INTEGER := 1  
);  
  
PROCEDURE ERASE(  
    LOB_LOC IN OUT CLOB,  
    AMOUNT IN OUT INTEGER,  
    OFFSET IN INTEGER := 1  
);
```

---

### 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT  
输入需要擦除数据的总数，BLOB 为字节，CLOB 为字符，输出实际擦除的总数，AMOUNT<=0 将报错，AMOUNT 大于原字段长度，将擦写所有数据。
- OFFSET 起始偏移。

## 7. FILECLOSE

关闭一个之前打开的 BFILE 对象。

语法如下：

---

```
PROCEDURE FILECLOSE(  
    FILE_LOC IN OUT BFILE  
);
```

---

### 参数详解

- FILE\_LOC BFILE 类型对象。

## 8. FILECLOSEALL

关闭所有之前打开的 BFILE 对象。

语法如下：

---

```
PROCEDURE FILECLOSEALL();
```

---

## 9. FILEEXISTS

判断指定的 BFILE 是否存在。

语法如下：

---

```
FUNCTION FILEEXISTS(  
    FILE_LOC IN BFILE  
)RETURN INTEGER;
```

---

#### 参数详解

- FILE\_LOC BFILE 类型对象。

#### 返回值

0: 不存在

1: 存在

### 10. FILEGETNAME

获取 BFILE 对象的目录名和文件名。

语法如下：

---

```
PROCEDURE FILEGETNAME(  
    FILE_LOC IN BFILE,  
    DIR_ALIAS OUT    VARHCAR,  
    FILENAME OUT    VARCHAR  
) ;
```

---

#### 参数详解

- FILE\_LOC BFILE 类型对象。
- DIR\_ALIAS 目录对象名。
- FILENAME 文件名。

### 11. FILEISOPEN

判断指定的 BFILE 是否已经打开。

语法如下：

---

```
FUNCTION FILEISOPEN(  
    FILE_LOC IN BFILE  
) RETURN INTEGER;
```

---

#### 参数详解

- FILE\_LOC BFILE 类型对象。

#### 返回值

0: 未打开

1: 打开

### 12. FILEOPEN

打开一个 BFILE 文件。

语法如下：

---

```
PROCEDURE FILEOPEN(  
    FILE_LOC IN OUT BFILE,  
    OPEN_MODE IN INTEGER:=FILE_READONLY  
);
```

---

#### 参数详解

- FILE\_LOC BFILE 类型对象。
- OPEN\_MODE 打开方式，仅支持 FILE\_READONLY。

### 13. FRAGMENT\_DELETE

删除指定偏移开始的指定长度的数据，不重写。

语法如下：

---

```
PROCEDURE FRAGMENT_DELETE(  
    LOB_LOC IN OUT BLOB,  
    AMOUNT IN INTEGER,  
    OFFSET IN INTEGER  
);  
  
PROCEDURE FRAGMENT_DELETE(  
    LOB_LOC IN OUT CLOB,  
    AMOUNT IN INTEGER,  
    OFFSET IN INTEGER  
);
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT 需要删除数据的总数，BLOB 为字节，CLOB 为字符。
- OFFSET 起始偏移。BLOB 为字节，CLOB 为字符。

#### 错误说明：

offset<1, offset 超过原字段长, amout 大于原字段长, amount + offset 大于原字段长, 返回错误。

### 14. FRAGMENT\_INSERT

插入指定长度的数据到 LOB 的指定偏移处，最大为 32K。

语法如下：

---

```
PROCEDURE FRAGMENT_INSERT(  
    LOB_LOC IN OUT BLOB,  
    AMOUNT IN INTEGER,  
    OFFSET IN INTEGER,  
    BUFFER IN VARBINARY
```

---

---

```
);
PROCEDURE FRAGMENT_INSERT(
    LOB_LOC IN OUT CLOB,
    AMOUNT IN INTEGER,
    OFFSET IN INTEGER,
    BUFFER IN VARCHAR
);
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT 需要插入数据的总数，BLOB 为字节，CLOB 为字符，最大 32k。
- OFFSET 写入起始的偏移，BLOB 为字节，CLOB 为字符。
- BUFFER 待写入数据。

#### 错误说明：

amount 大于 buffer 内数据长度，amount <= 0, offset <= 0, 返回错误。

#### 15. FRAGMENT\_MOVE

移动指定长度的数据从原始偏移到新偏移处。

语法如下：

---

```
PROCEDURE FRAGMENT_MOVE(
    LOB_LOC IN OUT BLOB,
    AMOUNT IN INTEGER,
    SRC_OFFSET IN INTEGER,
    DEST_OFFSET IN INTEGER
);

PROCEDURE FRAGMENT_MOVE(
    LOB_LOC IN OUT CLOB,
    AMOUNT IN INTEGER,
    SRC_OFFSET IN INTEGER,
    DEST_OFFSET IN INTEGER
);
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT 需要移动的数据总数，BLOB 为字节，CLOB 为字符。
- SRC\_OFFSET  
待移动数据的起始偏移。DEST\_OFFSET 不能在 (START\_OFFSET, START\_OFFSET + AMOUNT) 上取值。
- DEST\_OFFSET 目标偏移。BLOB 为字节，CLOB 为字符。

#### 错误说明：

src\_offset 或 dest\_offset 小于 1, src\_offset 或 dest\_offset 大于等于

原字段长, amount 小于 0, amount + srcoffset 大于等于原字段长度, 返回错误。

#### 16. FRAGMENT\_REPLACE

替换从指定偏移位置的指定长度的数据为新数据, 长度不超过 32K。

语法如下:

---

```
PROCEDURE FRAGMENT_REPLACE(  
    LOB_LOC IN OUT BLOB,  
    OLD_AMOUNT IN INTEGER,  
    NEW_AMOUNT IN INTEGER,  
    OFFSET IN INTEGER,  
    BUFFER IN VARBINARY  
);  
  
PROCEDURE FRAGMENT_REPLACE(  
    LOB_LOC IN OUT CLOB,  
    OLD_AMOUNT IN INTEGER,  
    NEW_AMOUNT IN INTEGER,  
    OFFSET IN INTEGER,  
    BUFFER IN VARCHAR2  
);
```

---

##### 参数详解

- LOB\_LOC 临时大对象。
- OLD\_AMOUNT 需要被替换数据的总数, BLOB 为字节, CLOB 为字符。
- NEW\_AMOUNT 替换数据的总数, BLOB 为字节, CLOB 为字符, 最大 32K。
- OFFSET 被替换数据的起始偏移。
- BUFFER 需要替换的数据。

##### 使用说明

OFFSET 小于 1, OLD\_AMOUNT 或 NEW\_AMOUNT 小于 0, OLD\_AMOUNT 或者 NEW\_AMOUNT 大于 32K, NEW\_AMOUNT 大于 BUFFER 内数据长度, 将返回错误。

#### 17. FREETEMPORARY

释放临时的 LOB 操作符。

语法如下:

---

```
PROCEDURE FREETEMPORARY(LOB_LOC IN OUT BLOB);  
PROCEDURE FREETEMPORARY(LOB_LOC IN OUT CLOB);
```

---

##### 参数详解

- LOB\_LOC 需要被释放的临时大对象

#### 18. GETLENGTH

返回 LOB 数据的字符长度。

语法如下：

---

```
FUNCTION GETLENGTH(LOB_LOC IN BLOB) RETURN INTEGER;
FUNCTION GETLENGTH(LOB_LOC IN CLOB) RETURN INTEGER;
FUNCTION GETLENGTH(FILE_LOC IN BFILE) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- FILE\_LOC BFILE 类型对象。

#### 返回值

临时大对象的数据长度。BLOB 为字节，CLOB 为字符。

### 19. GET\_STORAGE\_LIMIT

返回指定 LOB 的存储长度上限。

语法如下：

---

```
FUNCTION GET_STORAGE_LIMIT(LOB_LOC IN BLOB) RETURN INTEGER;
FUNCTION GET_STORAGE_LIMIT(LOB_LOC IN CLOB) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_LOC 大字段对象。

#### 返回值

大字段存储限制大小。DM 等同于最大长度。

### 20. INSTR

返回从指定偏移开始的第 nth 个匹配对象的位置。

语法如下：

---

```
FUNCTION INSTR(
    LOB_LOC IN BLOB,
    PATTERN IN VARBINARY,
    OFFSET IN INTEGER := 1,
    NTH IN INTEGER := 1
) RETURN INTEGER;
FUNCTION INSTR(
    LOB_LOC IN CLOB,
    PATTERN IN VARCHAR2,
    OFFSET IN INTEGER := 1,
    NTH IN INTEGER := 1
) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_LOC 临时大对象。

- PATTERN 匹配符。
- OFFSET 查找起始偏移，BLOB 为字节，CLOB 为字符。
- NTH 查找的第几个匹配符。

#### 返回值

找到的匹配符出现的偏移。BLOB 为字节，CLOB 为字符，找不到匹配串返回 0。

#### 错误说明：

如果任何一个输入参数：为 NULL 或者非法则返回 0，offset<1，offset>LOBMAXSIZE，nth<1，nth>LOBMAXSIZE 时报错。

### 21. ISOPEN

判断指定的 LOB 对象是否已经打开。

语法如下：

---

```
FUNCTION ISOPEN(LOB_LOC IN BLOB) RETURN INTEGER;
FUNCTION ISOPEN(LOB_LOC IN CLOB) RETURN INTEGER;
FUNCTION ISOPEN(FILE_LOC IN BFILE) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_LOB LOB 类型对象。
- FILE\_LOC BFILE 类型对象。

#### 返回值

0：未打开

1：打开

### 22. ISTEMPORARY

返回是否指定的 LOB 是临时操作符。

语法如下：

---

```
FUNCTION ISTEMPORARY(LOB_LOC IN BLOB) RETURN INTEGER;
FUNCTION ISTEMPORARY(LOB_LOC IN CLOB) RETURN INTEGER;
```

---

#### 参数详解

- LOB\_LOC 大对象。

#### 返回值

1：是临时大对象；

0：不是临时大对象；

NULL：输入参数为 NULL 或没有与表的大字段列进行关联的大字段对象。

### 23. LOADBLOBFROMFILE

从 BFILE 对象加载数据到 BLOB 对象。



语法如下：

---

```
PROCEDURE LOADBLOBFROMFILE(  
    DEST_LOB    IN OUT    BLOB,  
    SRC_BFILE   IN        BFILE,  
    AMOUNT      IN        INTEGER,  
    DEST_OFFSET IN OUT    INTEGER,  
    SRC_OFFSET  IN OUT    BIGINT  
);
```

---

#### 参数详解

- DEST\_LOB 目标 BLOB 对象。
- SRC\_BFILE 源 BFILE 对象。
- AMOUNT 要从 BFILE 对象加载到 BLOB 对象的字节数。
- DEST\_OFFSET 开始写入 BLOB 对象的偏移。
- SRC\_OFFSET 从 BFILE 对象开始读取的偏移。

#### 24. LOADFROMFILE

从 BFILE 对象加载数据到 BLOB 对象。

语法如下：

---

```
PROCEDURE LOADFROMFILE(  
    DEST_LOB    IN OUT    BLOB,  
    SRC_FILE    IN        BFILE,  
    AMOUNT      IN        INTEGER,  
    DEST_OFFSET IN OUT    INTEGER:=1,  
    SRC_OFFSET  IN OUT    BIGINT:=1  
);
```

---

#### 参数详解

- DEST\_LOB 目标 BLOB 对象。
- SRC\_FILE 源 BFILE 对象。
- AMOUNT 要从 BFILE 对象加载到 BLOB 对象的字节数。
- DEST\_OFFSET 开始写入 BLOB 对象的偏移。
- SRC\_OFFSET 从 BFILE 对象开始读取的偏移。

#### 25. OPEN

打开一个 LOB 对象。

语法如下：

---

```
PROCEDURE OPEN(  
    LOB_LOC IN OUT BLOB,  
    OPEN_MODE IN INTEGER  
);
```

---

---

```

PROCEDURE OPEN(
    LOB_LOC IN OUT CLOB,
    OPEN_MODE IN INTEGER
);

PROCEDURE OPEN(
    FILE_LOC IN OUT BFILE,
    OPEN_MODE IN INTEGER:=FILE_READONLY
);

```

---

### 参数详解

- LOB\_LOC LOB 对象。
- FILE\_LOC BFILE 类型对象。
- OPEN\_MODE 打开方式，BFILE 对象仅支持 FILE\_READONLY。

## 26. READ

读取指定偏移开始的指定长度数据到 buffer 中。

语法如下：

---

```

PROCEDURE READ(
    LOB_LOC IN BLOB,
    AMOUNT IN OUT INTEGER,
    OFFSET IN INTEGER,
    BUFFER OUT VARBINARY
);

PROCEDURE READ(
    LOB_LOC IN CLOB,
    AMOUNT IN OUT INTEGER,
    OFFSET IN INTEGER,
    BUFFER OUT VARCHAR2
);

PROCEDURE READ(
    FILE_LOC IN BFILE,
    AMOUNT IN OUT INTEGER,
    OFFSET IN BIGINT,
    BUFFER OUT VARBINARY
);

```

---

### 参数详解

- LOB\_LOC 临时大对象。
- FILE\_LOC BFILE 对象。
- AMOUNT 输入需要读取的数据总数，BLOB 为字节，CLOB 为字符，输出为实际读取的总数。
- OFFSET 读取数据的偏移，BLOB 为字节，CLOB 为字符。

- BUFFER 存储数据的缓冲区。

#### 错误说明：

如果对象为 CLOB 类型，offset + amount 大于原字段长度，将会自动截断，返回比要求长度少的数据。如果是 BLOB 类型，返回错误。

amount 小于等于 0，返回错误。

offset 为 null，小于等于 0，超过原字段长度，返回错误。

## 27. SUBSTR

获取指定偏移开始的指定长度的子串。

语法如下：

---

```

FUNCTION SUBSTR(
    LOB_LOC IN BLOB,
    AMOUNT IN INTEGER := 32767,
    OFFSET IN INTEGER := 1
) RETURN VARBINARY;
FUNCTION SUBSTR(
    LOB_LOC IN CLOB,
    AMOUNT IN INTEGER := 32767,
    OFFSET IN INTEGER := 1
) RETURN VARCHAR2;
FUNCTION SUBSTR(
    FILE_LOC IN BFILE,
    AMOUNT IN INTEGER := 32767,
    OFFSET IN BIGINT := 1
) RETURN VARBINARY;
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- FILE\_LOC BFILE 对象。
- AMOUNT

读取的数据长度，BLOB 为字节，CLOB 为字符，上限为 32767，实际获取为大对象长度与 AMOUNT 的小值。

- OFFSET 读取数据的起始偏移。

#### 返回值

返回指定偏移开始的指定长度的子串。若输入的 amount 或 offset 参数非法，返回 NULL。

## 28. TRIM/TRIM\_LOB

截断 LOB 数据到指定的长度。

语法如下:

---

```
PROCEDURE TRIM(  
    LOB_LOC IN OUT BLOB,  
    NEW_LEN IN INTEGER  
);  
  
PROCEDURE TRIM(  
    LOB_LOC IN OUT CLOB,  
    NEW_LEN IN INTEGER  
);  
  
PROCEDURE TRIM_LOB(  
    LOB_LOC IN OUT BLOB,  
    NEW_LEN IN INTEGER  
);  
  
PROCEDURE TRIM_LOB(  
    LOB_LOC IN OUT CLOB,  
    NEW_LEN IN INTEGER  
);
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- NEW\_LEN 截断到的新长度，BLOB 为字节，CLOB 为字符。

#### 29. WRITEAPPEND

追加指定长度字符的数据到 LOB 对象的末尾。

语法如下:

---

```
PROCEDURE WRITEAPPEND(  
    LOB_LOC IN OUT BLOB,  
    AMOUNT IN INTEGER,  
    BUFFER IN VARBINARY  
);  
  
PROCEDURE WRITEAPPEND(  
    LOB_LOC IN OUT CLOB,  
    AMOUNT IN INTEGER,  
    BUFFER IN VARCHAR2  
);
```

---

#### 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT 追加数据的总数，BLOB 为字节，CLOB 为字符。
- BUFFER 待追加的数据。

#### 错误说明:

amount 小于等于 0 或 null 或者超过 buffer 内数据长度，返回错误。

## 30. WRITE

写入指定长度字符的数据到 LOB 对象，从 LOB 对象的指定绝对偏移开始。如果指定的位置有数据，则新写入的会覆盖原来的数据。

语法如下：

---

```
PROCEDURE WRITE (
    LOB_LOC IN OUT BLOB,
    AMOUNT IN INTEGER,
    OFFSET IN INTEGER,
    BUFFER IN VARBINARY
);
PROCEDURE WRITE (
    LOB_LOC IN OUT CLOB,
    AMOUNT IN INTEGER,
    OFFSET IN INTEGER,
    BUFFER IN VARCHAR2
);
```

---

## 参数详解

- LOB\_LOC 临时大对象。
- AMOUNT 写入数据的总数，BLOB 为字节，CLOB 为字符。
- OFFSET 偏移量，起始为 1。BLOB 为字节，CLOB 为字符。
- BUFFER 待写入的数据。

## 错误说明：

amount 小于等于 0 或 null 或者超过 buffer 内数据长度，返回错误。

## 31. CONVERTTOBLOB

将指定的 lob 的指定部分按一定的编码格式完成 blob 和 clob 的互相转换。

语法如下：

---

```
PROCEDURE CONVERTTOBLOB(
    DEST_LOB          IN OUT  BLOB,
    SRC_CLOB          IN      CLOB,
    AMOUNT            IN      INTEGER,
    DEST_OFFSET       IN OUT  INTEGER,
    SRC_OFFSET        IN OUT  INTEGER,
    BLOB_CSID         IN      NUMBER,
    LANG_CONTEXT      IN OUT  INTEGER,
    WARNING           OUT  INTEGER
);
PROCEDURE CONVERTTOCLOB(
    DEST_LOB          IN OUT  CLOB,
    SRC_BLOB          IN      BLOB,
```

---

---

```

    AMOUNT      IN      INTEGER,
    DEST_OFFSET IN OUT  INTEGER,
    SRC_OFFSET   IN OUT  INTEGER,
    BLOB_CSID    IN      NUMBER,
    LANG_CONTEXT IN OUT  INTEGER,
    WARNING      OUT     INTEGER
);

```

---

### 参数详解

- DEST\_LOB

输入输出参数，转换的结果将存入到这个大字段中，如果这个大字段原来已经有数据，新转换出的数据会根据指定的目标偏移覆盖原数据。

- SRC\_LOB 待转换的数据。

- AMOUNT 转换的数据总数，BLOB 为字节，CLOB 为字符。

- DEST\_OFFSET 完成转换的数据向目的大字段存放时的偏移。

- SRC\_OFFSET 源数据开始转换的偏移。

- BLOB\_CSID, LANG\_CONTEXT, WARNING

这三个参数暂时不支持，调用时指定任意值即可，CLOB 转 BLOB 时，最后 BLOB 编码格式和 CLOB 的存入格式相同。

### 错误说明：

如果 AMOUNT 大于 SRC\_LOB 的长度、AMOUNT 小于 0、DEST\_OFFSET + AMOUNT 大于大字段限制最大长度 (DBMS\_LOB.LOBMAXSIZE)，返回错误。

## 7.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_LOB');
```

例将输入的 BLOB 类型数据转化为 CLOB 类型。

```

CREATE OR REPLACE FUNCTION BLOBTOCLOB ( BLOB_IN IN BLOB)
    RETURN CLOB
AS
    V_CLOB CLOB;
    V_VARCHAR VARCHAR2(32767);
    V_START PLS_INTEGER := 1;
    V_BUFFER PLS_INTEGER := 32767;
BEGIN
    DBMS_LOB.CREATETEMPORARY(V_CLOB, TRUE);
    FOR I IN 1 .. CEIL(DBMS_LOB.GETLENGTH(BLOB_IN) / V_BUFFER) + 1
    LOOP
        V_VARCHAR := UTL_RAW.CAST_TO_VARCHAR2(DBMS_LOB.SUBSTR(BLOB_IN,

```

```
V_BUFFER, V_START));  
        DBMS_LOB.WRITEAPPEND(V_CLOB, LENGTH(V_VARCHAR), V_VARCHAR);  
        PRINT V_CLOB;  
        V_START := V_START + V_BUFFER;  
    END LOOP;  
    RETURN V_CLOB;  
END BLOBTOCLOB;  
/
```

调用函数：

```
CALL BLOBTOCLOB('B4EFC3CECAFDBEDDBFE2D3D0CFDEB9ABCBBE');
```

查看结果：

达梦数据库有限公司

## 8 DBMS\_LOCK 包

兼容 ORACLE 的 DBMS\_LOCK 包，提供锁管理服务器的接口。该包可以完成下列功能：

1. 提供对设备的独占访问；
2. 提供程序级的读锁；
3. 判断程序之后何时释放锁；
4. 同步程序以及强制顺序执行。

DM MPP 环境和 DM DSC 环境下不支持 DBMS\_LOCK 包。

### 8.1 封锁规则

#### 1、封锁类型

DBMS\_LOCK 包过程函数对于封锁的类型定义如下表所示，其数据类型为 INTEGER。

名称	意义	值	描述
NL_MODE	NULL	1	
SS_MODE	Sub Shared	2	可以使用在聚合对象以指定在该对象的子部分上获取的共享锁
SX_MODE	Sub eXclusive Row Exclusive Mode	3	可以使用在聚合对象以指定在该对象的子部分上获取的独占锁
S_MODE	Shared Row Exclusive Mode Intended Exclusive	4	共享锁
SSX_MODE	Shared Sub eXclusive Share Row Exclusive Mode	5	该模式表名整个聚合对象占有共享锁，但其某些组成部分会上独占锁（S_MODE+SX_MODE）
X_MODE	eXclusive	6	独占锁

#### 2、兼容规则

若当前锁对象已被其他程序封锁，新封锁的类型兼容规则如下：

当前持有	新加					
	NL	SS	SX	S	SSX	X
NL	Success	Success	Success	Success	Success	Success
SS	Success	Success	Success	Success	Success	Fail
SX	Success	Success	Success	Fail	Fail	Fail
S	Success	Success	Fail	Success	Fail	Fail



<b>SSX</b>	Success	Success	Fail	Fail	Fail	Fail
<b>X</b>	Success	Fail	Fail	Fail	Fail	Fail

## 8.2 相关方法

### 1. ALLOCATE\_UNIQUE

申请一个锁对象。已经申请的锁对象可在 SYS.DBMS\_LOCK\_ALLOCATED 表查询。

语法如下：

```
PROCEDURE ALLOCATE_UNIQUE (
    LOCK_NAME          IN  VARCHAR2,
    LOCK_HANDLE        OUT VARCHAR2,
    EXPIRATION_SECS    IN  INTEGER  DEFAULT 864000
);
```

#### 参数详解

- LOCK\_NAME 输入参数，锁对象名称。
- LOCK\_HANDLE  
输出参数，锁对象句柄，根据锁名生成的唯一标识；后续过程可以使用该标识对锁对象进行操作。会话对一个新的锁名调用该过程时，会在系统中新生成一个锁对象，在 DBMS\_LOCK\_ALLOCATED 表中插入一行数据记录锁的相关信息。
- EXPIRATION\_SECS  
输入参数，锁对象的过期时间。最后一次调用 ALLOCATE\_UNIQUE 后允许从 DBMS\_LOCK\_ALLOCATED 表中删除数据的等待时间，单位为秒。

### 2. REQUEST

根据指定类型进行封锁。

语法如下：

```
FUNCTION REQUEST(
    LOCK_ID            IN  INTEGER,
    LOCK_MODE          IN  INTEGER DEFAULT X_MODE,
    TIMEOUT            IN  INTEGER DEFAULT MAXWAIT,
    RELEASE_ON_COMMIT IN  BOOLEAN DEFAULT FALSE
)RETURN INTEGER;

FUNCTION REQUEST(
    LOCK_HANDLE        IN  VARCHAR2,
    LOCK_MODE          IN  INTEGER DEFAULT X_MODE,
    TIMEOUT            IN  INTEGER DEFAULT MAXWAIT,
    RELEASE_ON_COMMIT IN  BOOLEAN DEFAULT FALSE
)RETURN INTEGER;
```

#### 参数详解

- LOCK\_ID 或 LOCK\_HANDLE 输入参数，申请进行封锁动作的锁对象 ID 或句柄。
- LOCK\_MODE 输入参数，上锁的模式。
- TIMEOUT 输入参数，尝试上锁的等待时间，单位为秒。
- RELEASE\_ON\_COMMIT 输入参数，指明在事务结束时是否释放锁。

返回值说明：

返回值	描述
0	成功
1	超时
2	死锁
3	参数错误
4	没有拥有锁
5	无效的锁 id 或句柄

使用说明：

若使用 LOCK\_ID 进行封锁，锁对象的 ID 取值范围为 0~1073741823。否则返回 3（参数错误）。

使用名称 ALLOCATE\_UNIQUE 的锁对象的 ID 范围为 1073741824~1999999999。也就是说，使用 ALLOCATE\_UNIQUE 方式创建的表对象不能通过 LOCK\_ID 的方式进行封锁转化和释放。同时可以使用 LOCK\_ID 不进行 ALLOCATE\_UNIQUE 而直接 REQUEST，但通过 LOCK\_ID 方式使用的锁对象不会插入到 DBMS\_LOCK\_ALLOCATED 表中。

### 3. CONVERT/CONVERT2

对封锁类型进行修改。

语法如下：

```
FUNCTION CONVERT(  
    LOCK_ID          IN INTEGER,  
    LOCK_MODE        IN INTEGER,  
    TIMEOUT          IN NUMBER DEFAULT MAXWAIT  
)RETURN INTEGER;  
  
FUNCTION CONVERT(  
    LOCK_HANDLE      IN VARCHAR2,  
    LOCK_MODE        IN INTEGER,  
    TIMEOUT          IN NUMBER DEFAULT MAXWAIT  
)RETURN INTEGER;  
  
FUNCTION CONVERT2(  
    LOCK_ID          IN INTEGER,  
    LOCK_MODE        IN INTEGER,  
    TIMEOUT          IN NUMBER DEFAULT MAXWAIT  
)RETURN INTEGER;
```

---

```

FUNCTION CONVERT2(
    LOCK_HANDLE IN VARCHAR2,
    LOCK_MODE   IN INTEGER,
    TIMEOUT     IN NUMBER DEFAULT MAXWAIT
)RETURN INTEGER;

```

---

### 参数详解

- LOCK\_ID 或 LOCK\_HANDLE

输入参数，要修改的锁对象 ID 或根据过程 ALLCOATE\_UNIQUE 得到的锁对象句柄；  
LOCK\_ID 的说明同 REQUEST。

- LOCK\_MODE 对锁对象新的封锁模式。
- TIMEOUT 尝试修改的等待时间，如果在该阶段内没有成功，返回 1（超时）。

### 返回值说明：

返回值	描述
0	成功
1	超时
2	死锁
3	参数错误
4	没有拥有锁
5	无效的锁的 id 或句柄

### 使用说明：

如果只有当前程序对该锁对象进行了封锁，可以成功转化到其他任意封锁模式；

如果与其他程序并发封锁，新的上锁模式是否能够成功要取决于其他程序所持锁的模式与新的封锁模式是否兼容，如果不兼容，则上锁失败；

由于 CONVERT 是 DM 数据库的保留字，调用 CONVERT 函数时需要使用“`”对其进行引用，或者使用同样功能的 CONVERT2。

### 4. RELEASE 函数

DBMS\_LOCK 包默认在会话结束时释放对锁对象的封锁，该函数用于显式地释放当前会话持有的一个锁对象的封锁。

### 语法如下：

---

```

FUNCTION RELEASE (
    LOCK_ID      IN INTEGER
)RETURN INTEGER;

FUNCTION RELEASE (
    LOCK_HANDLE IN VARCHAR2
)RETURN INTEGER;

```

---

### 参数详解

- LOCK\_ID 或 LOCK\_HANDLE

输入参数，要释放封锁的锁对象 ID 或根据过程 ALLOCATE\_UNIQUE 返回的句柄；

LOCK\_ID 的说明同 REQUEST。

返回值说明：

返回值	描述
0	成功
3	参数错误
4	没有拥有锁
5	无效的锁的 id 或句柄

### 5. SLEEP 过程

使会话休眠一段时间。

语法如下：

```
PROCEDURE SLEEP (
    SECONDS IN NUMBER
);
```

### 参数详解

- SECONDS 输入参数，休眠时间，单位为秒。

## 8.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_LOCK');
```

如果要通过名称来进行封锁，首先要通过 ALLOCATE\_UNIQUE 过程申请封锁句柄，之后通过该句柄进行封锁和释放操作。如下例以 'my\_lock' 申请句柄并上 x 锁。

```
DECLARE
HDL VARCHAR(128);
RET INT;
BEGIN
DBMS_LOCK.ALLOCATE_UNIQUE('MY_LOCK', HDL);
SELECT DBMS_LOCK.REQUEST(HDL, 6) INTO RET;
PRINT RET;
END;
/
```

执行成功后，其他会话同上例执行封锁，会因封锁不兼容而进行等待。

若已经进行了封锁，需要对封锁的类型进行修改，比如，上述例子中进行 x 封锁的级别

太高，导致其他会话无法对 'my\_lock' 进行封锁，可将其转换为 S 锁。

```
DECLARE
HDL VARCHAR(128);
RET INT;
BEGIN
DBMS_LOCK.ALLOCATE_UNIQUE('MY_LOCK', HDL);
SELECT DBMS_LOCK."CONVERT"(HDL, 4) INTO RET;
PRINT RET;
END;
/
```

转换成功后，其他会话以与 S 锁兼容的模式（比如 NL，SS，S 模式）进行封锁均可成功。

若已经完成了封锁内需要的操作，可以以如下方式释放该封锁。

```
DECLARE
HDL VARCHAR(128);
BEGIN
DBMS_LOCK.ALLOCATE_UNIQUE('MY_LOCK', HDL);
DBMS_LOCK.RELEASE(HDL);
END;
/
```

上述各例中，均通过句柄进行封锁操作，用户也可直接通过 id 进行封锁。封锁和转换时省略超时时间表示一直等待直至成功，用户可根据需要设置超时时间，也可将超时时间设置为 0 使其立刻返回。

通过名称申请句柄的锁对象，会添加在系统表 SYS.DBMS\_LOCK\_ALLOCATED 中查询得到。该系统表中的数据在锁对象过期后根据情况进行淘汰。

如上操作后，查询可以得到 'MY\_LOCK' 的相关信息，包括名称、id 及过期时间。

```
SQL>SELECT * FROM SYS.DBMS_LOCK_ALLOCATED;
```

行号	NAME	LOCKID	EXPIRATION
1	MY_LOCK	1073741824	2014-07-20 16:19:59.708000

DM 也提供动态性能视图 V\$DBMS\_LOCKS 供用户查看当前系统中 DBMS\_LOCK 封锁的情况。如两个会话并发申请 'MY\_LOCK'，其中 S 锁封锁成功，X 锁封锁等待的情况，查询可得如下结果：

```
SQL>SELECT * FROM V$DBMS_LOCKS;
```

行号	HANDLE	ID	MODE	OWN_SESS	WAIT_SESS
1	1073741824	1073741824	176	1073741824	S 515604448 NULL
2	1073741824	1073741824	176	1073741824	X 516931992 515604448

通过记录可以看出 x 封锁在等待 s 封锁。

## 9 DBMS\_LOGMNR 包

用户可以使用 DBMS\_LOGMNR 包对归档日志进行挖掘，重构出 DDL 和 DML 等操作，并通过获取的信息进行更深入的分析。

目前 DBMS\_LOGMNR 只支持对归档日志进行分析，配置归档后，还需要将 dm.ini 中的 RLOG\_APPEND\_LOGIC 选项置为 1 或 2。

DM MPP 环境下不支持 DBMS\_LOGMNR 包。

### 9.1 相关方法

#### 1. PROCEDURE ADD\_LOGFILE

增加日志文件。

语法如下：

---

```
PROCEDURE ADD_LOGFILE (
    LOGFILENAME      IN VARCHAR,
    OPTIONS          IN INT DEFAULT ADDFILE
);
```

---

#### 参数详解

- LogFileName

增加的文件的全路径。

- Options

可选配置参数，具体的可选配置包括：

- ◆ NEW 结束当前 LOGMNR（调用 LOGMNR\_END），并增加指定文件
- ◆ ADDFILE 在当前 LOGMNR 中增加日志文件
- ◆ REMOVE 从当前 LOGMNR 中去除一个日志文件（如果已经 START，则不可移除）

#### 2. COLUMN\_PRESENT

判断某列是否被包含在指定的一行逻辑记录中。

语法如下：

---

```
FUNCTION COLUMN_PRESENT (
    SQL_REDO_UNDO      IN BIGINT,
    COLUMN_NAME        IN VARCHAR DEFAULT ''
) RETURN INT;
```

---

#### 参数详解

- SQL\_REDO\_UNDO

REDO 或 UNDO 记录的唯一标识，对应 V\$LOGMNR\_CONTENTS 中的 REDO\_VALUE 或

UNDO\_VALUE 字段。

- COLUMN\_NAME

由模式名.表名.列名组成的字符串。

### 返回值

0或1，0表示该值不可以挖掘，1表示可以挖掘

3. END\_LOGMNR

语法如下：

---

```
PROCEDURE END_LOGMNR();
```

---

### 功能说明：

结束 LOGMNR。

4. MINE\_VALUE

以字符串的格式来获取某一条日志中包含的指定列的值。

语法如下：

---

```
FUNCTION MINE_VALUE(
    SQL_REDO_UNDO IN BIGINT,
    COLUMN_NAME IN VARCHAR DEFAULT '') RETURN VARCHAR;
);
```

---

### 参数详解

- SQL\_REDO\_UNDO

REDO 或 UNDO 记录的唯一标识，对应 V\$LOGMNR\_CONTENTS 中的 REDO\_VALUE 或 UNDO\_VALUE 字段。

- COLUMN\_NAME

由模式名.表名.列名组成的字符串。

### 返回值

以字符串的格式返回某一条日志中包含的指定列的值。

5. REMOVE\_LOGFILE

从日志列表中移除某个日志文件。

语法如下：

---

```
PROCEDURE REMOVE_LOGFILE (
    LOGFILENAME IN VARCHAR
);
```

---

### 参数详解

- LogFileName

待移除的日志文件名。

6. START\_LOGMNR



根据指定的模式和条件来开始某个会话上的 LOGMNR，一个会话上仅能 START 一个 LOGMNR。

语法如下：

```
PROCEDURE START_LOGMNR (
    STARTSCN          IN BIGINT DEFAULT 0,
    ENDSCN            IN BIGINT DEFAULT 0,
    STARTTIME         IN DATETIME DEFAULT '1988/1/1',
    ENDTIME           IN DATETIME DEFAULT '2110/12/31',
    DICTFILENAME       IN VARCHAR DEFAULT '',
    OPTIONS           IN INTDEFAULT 0
);
```

### 参数详解

- STARTSCN

分析时或者加载时的过滤条件，日志起始序列号，默认 0，表示无限制。

- ENDSCN

分析时或者加载时的过滤条件，日志结束序列号，默认 0，表示无限制。

- STARTTIME

分析时或者加载时的过滤条件，日志起始时间，默认 1988/1/1。

- ENDTIME

分析时或者加载时的过滤条件，日志结束时间，默认 2110/12/31。

- DICTFILENAME

离线字典的全路径名，默认为空。如果选用离线字典模式，则需要指定该选项，根据此项提供的全路径来加载离线字典文件。如果已经配置了其他模式（在线字典或者 REDO 日志抽取字典信息），此项不会生效。

- OPTIONS

提供如下表所列的可选模式，各模式可以通过 + 或者按位或来进行组合。其它位的值如 1、4、8 等目前不支持，配置后不会报错，但是没有效果。例如，组合全部模式，则取值计算方法为  $2+16+64+2048=2130$ ，那么 OPTIONS 值取就是 2130。

Options	对应值	说明
COMMITTED_DATA_ONLY	2	仅从已交的事务的日志中挖掘信息
DICT_FROM_ONLINE_CATALOG	16	使用在线字典
NO_SQL_DELIMITER	64	拼写的 SQL 语句最后不添加分隔符
NO_ROWID_IN_STMT	2048	拼写的 SQL 语句中不包含 ROWID

## 9.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_LOGMNR');
```

例展示日志分析工具使用流程。

### 第1步 配置环境

(1) 修改 dm.ini 中的参数，如下：

```
ARCH_INI          = 1
RLOG_APPEND_LOGIC = 1
```

(2) dmarch.ini 需要配置本地归档，举例如下：

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL

ARCH_DEST          = d:\dmdata\arch

ARCH_FILE_SIZE     = 128          #单位 Mb

ARCH_SPACE_LIMIT   = 0          #单位 Mb, 0 表示无限制, 范围 1024~4294967294M
```

### 第2步 添加归档日志文件

(1) 查询有哪些归档日志

```
SELECT NAME , FIRST_TIME , NEXT_TIME , FIRST_CHANGE# , NEXT_CHANGE# FROM
V$ARCHIVED_LOG;
```

设查询结果如下：

```
D:\DMDATA\ARCH\ARCHIVE_LOCAL1_20140505155106577.LOG
2014-05-05 15:51:06.000000 2014-05-05 15:51:07.000000
1                23303
```

(2) 添加一个或多个需要分析的归档日志文件

```
DBMS_LOGMNR.ADD_LOGFILE('D:\DMDATA\ARCH\ARCHIVE_LOCAL1_20140505155106577.LOG'
)
```

如要查看通过 ADD\_LOGFILE 添加的归档日志文件，可以通过动态视图 V\$LOGMNR\_LOGS 进行查询，如下：

```
SELECT LOW_SCN, NEXT_SCN, LOW_TIME, HIGH_TIME, LOG_ID, FILENAME FROM
V$LOGMNR_LOGS;
```

查询结果如下：

```
1                23303
2014-05-05 15:51:06.000000 2014-05-05 15:51:07.000000 0
D:\DMDATA\ARCH\ARCHIVE_LOCAL1_20140505155106577.LOG
```

### 第3步 启动归档日志文件分析

```
DBMS_LOGMNR.START_LOGMNR(OPTIONS=>2128 , STARTTIME=>TO_DATE('2014-03-04
17:36:00','YYYY-MM-DD HH24:MI:SS') , ENDTIME=>TO_DATE('2014-06-04
```

```
17:36:02','YYYY-MM-DD HH24:MI:SS'));
```

如要查看归档日志文件的分析结果，可以通过动态视图 V\$logmnr\_contents 进行查询，如下：

```
SELECT OPERATION_CODE , SCN, SQL_REDO , TIMESTAMP ,SEG_OWNER, TABLE_NAME FROM  
V$logmnr_contents WHERE SEG_OWNER = 'SYSDBA' AND OPERATION_CODE IN (3,1,2);
```

#### 第4步 终止归档日志文件分析

```
DBMS_LOGMNR.END_LOGMNR( );
```

## 10 DBMS\_METADATA 包

GET\_DDL 函数用于获取数据库对表、视图、索引、全文索引、存储过程、函数、包、序列、同义词、约束、触发器等的 DDL 语句。

MPP 环境下不支持使用 DBMS\_METADATA 包。

### 10.1 数据类型

DBMS\_METADATA 包中涉及到类型。如下统一说明。

#### 1. KU\$\_PARSED\_ITEM 记录类型

---

```
TYPE KU$_PARSED_ITEM IS RECORD (
    ITEM          VARCHAR(30),
    VALUE         VARCHAR(4000),
    OBJECT_ROW    SMALLINT
);
```

---

##### 参数详解

- ITEM  
被解析对象属性的名称。
- VALUE  
对象属性值，如果没有就是 NULL。
- OBJECT\_ROW  
对象所在的行数。

KU\$\_PARSED\_ITEMS 为 KU\$\_PARSED\_ITEM 类型的索引表：

---

```
TYPE KU$_PARSED_ITEMS IS TABLE OF KU$_PARSED_ITEM INDEX BY INT;
```

---

#### 2. KU\$\_DDL 记录类型

---

```
TYPE KU$_DDL IS RECORD(
    DDLTEXT      CLOB,
    PARSEDITEM   KU$_PARSED_ITEMS
);
```

---

##### 参数详解

- DDLTEXT  
DDL 语句。
- PARSEDITEM  
对象属性。

KU\$\_DDL 为 KU\$\_DDL 类型的索引表：

---

```
TYPE KU$_DDL IS TABLE OF KU$_DDL INDEX BY INT;
```

---

### 3. KU\$\_MULTI\_DDL 记录类型

---

```
TYPE KU$_MULTI_DDL IS RECORD(
    OBJECT_ROW NUMBER,
    DDLS KU$_DDL$
);
```

---

#### 参数详解

- OBJECT\_ROW  
对象所在的行数。
- DDLS  
KU\$\_DDL\$ 类型。

KU\$\_MULTI\_DDL\$ 为 KU\$\_MULTI\_DDL 类型的索引表:

---

```
TYPE KU$_MULTI_DDL$ IS TABLE OF KU$_MULTI_DDL INDEX BY INT;
```

---

## 10.2 相关方法

### 1. GET\_DDL

获取指定对象元数据中的 DDL 语句。

语法如下:

---

```
FUNCTION GET_DDL(
    OBJECT_TYPE IN VARCHAR(30),
    NAME        IN VARCHAR(128),
    SCHNAME     IN VARCHAR(128) DEFAULT NULL
) RETURN CLOB
```

---

#### 参数详解

- OBJECT\_TYPE  
对象类型。包括表、视图、物化视图、索引、全文索引、存储过程、函数、包、目录等，详情请见 OPEN 参数详解。其中，OBJECT\_TYPE 只能为大写。
- NAME  
对象名称，区分大小写。
- SCHEMA  
模式，默认是当前用户模式。

#### 返回值

以 DDL 返回对象元数据中的 DDL 语句。

#### 错误处理

INVALID\_ARGVAL: 如果输入参数中存在空值或非法值。

OBJECT\_NOT\_FOUND: 如果指定的对象在数据库中不存在。

## 2. "OPEN"

打开对象类型的句柄

语法如下：

```
FUNCTION OPEN (
    OBJECT_TYPE IN VARCHAR2
) RETURN NUMBER;
```

## 参数详解

- OBJECT\_TYPE

可返回的对象类型名称，合法的类型名称见下表。

表 OPEN 函数可返回的对象类型名称

类型名称	含义	属性	说明
CLASS	类类型	SN	默认返回类头类体
CLASS_HEAD	类型名	SN	无
CLASS_BODY	类型体	SN	无
COL_STATISTICS	列统计	D	无
COMMENT	注释	D	无
CONSTRAINT	约束	SND	不包括聚集主键和非空约束
CONTEXT	上下文	N	无
CONTEXT_INDEX	全文索引	N	无
DATABASE_EXPORT	数据库下的所有对象	H	库级导出
DB_LINK	数据库链接	SN	因此类对象具有所有者，因此将其视为模式级对象。 对于公有连接，它们的所有者是 PUBLIC；对于私有链接，它们的创建者就是它们的所有者
DIRECTORY	目录	N	无
DOMAIN	域	N	无
FUNCTION	存储函数	SN	无
INDEX	索引	SND	不包括系统内部定义的索引
INDEX	索引统计	D	无

_STATISTICS			
JOB	任务	N	无
OBJECT_GRANT	对象权限	DG	无
PACKAGE	包	SN	默认返回包头包体
PKG_SPEC	包头	SN	无
PKG_BODY	包体	SN	无
POLICY	策略	D	无
PROCEDURE	存储过程	SN	无
ROLE	角色	N	无
ROLE_GRANT	角色权限	G	无
SCHEMA_EXPORT	模式下的所有对象	H	模式级导出
SEQUENCE	序列	SN	无
SYNONYM	同义词	见说明	私有同义词为模式对象, 公有同义词为命名对象
SYSTEM_GRANT	系统权限	G	无
TABLE	表	SN	无
TABLE_STATISTICS	表统计信息	D	无
TABLE_EXPORT	表及其相关的元数据	H	表级导出
TABLESPACE	表空间	N	无
TRIGGER	触发器	SND	无
USER	用户	N	无
VIEW	视图	SN	无
TYPE	用户自定义类型	SN	无
MATERIALIZED_VIEW	物化视图	SN	无
MATERIALIZED_VIEW_LOG	物化视图日志	SN	无

注：属性列中，S 表示模式对象，N 表示命名的对象，D 表示依赖对象，G 表示被授权的对象，H 表示包含不同类型的对象。

### 返回值

不透明的句柄。该句柄将用于 SET\_FILTER, SET\_PARSE\_ITEM, FETCH\_XXX, AND CLOSE。

### 错误处理

INVALID\_ARGVAL: 参数为 NULL, 或者参数值非法。

### 3. "CLOSE"

关闭 OPEN 打开的句柄，并清理相关环境。

语法如下：

---

```
PROCEDURE CLOSE (
    HANDLE IN NUMBER
);
```

---

#### 参数详解

HANDLE

OPEN 函数返回的句柄。

#### 错误处理

INVALID\_ARGVAL: 参数为 NULL, 或者参数值非法。

#### 4. FETCH\_DDL()

该函数返回符合 OPEN, SET\_FILTER, SET\_PARSE\_ITEM 设定条件的对象元数据。

语法如下：

---

```
FUNCTION FETCH_DDL (
    HANDLE IN NUMBER
) RETURN KU$_DDL;
```

---

#### 参数详解

##### ● HANDLE

OPEN 函数返回的句柄。

#### 返回值

对象的元数据或者当所有对象都已返回后返回 NULL。

#### 使用说明：

从表 KU\$\_DDL 中返回 DDL 语句, 表 KU\$\_DDL 的每一行在 DDLTEXT 列包含一条 DDL 语句; 如果使用解析的话, 解析出的对象属性将从 PARSEDITEM 列返回。

#### 错误处理

INVALID\_ARGVAL: 参数为 NULL, 或者参数值非法。

#### 5. FETCH\_CLOB()

该函数返回符合 OPEN, SET\_FILTER 设定条件的对象元数据。

语法如下：

---

```
FUNCTION FETCH_CLOB(
    HANDLE IN INT
) RETURN CLOB;
```

---

#### 参数详解

##### ● HANDLE

OPEN 函数返回的句柄。

#### 返回值



对象的元数据或者当所有对象都已返回后返回 NULL。

**使用说明：**

将对象以 CLOB 返回。

**异常：**

INVALID\_ARGVAL：参数为 NULL，或者参数值非法。

## 6. GET\_GRANTED\_DDL ( )

该函数用于返回对象的授权语句。

语法如下：

---

```
FUNCTION GET_GRANTED_DDL (
    OBJECT_TYPE      IN VARCHAR2,
    GRANTEE          IN VARCHAR2,
    OBJECT_COUNT      IN NUMBER   DEFAULT 10000
)RETURN CLOB;
```

---

**参数详解**

- OBJECT\_TYPE  
对象类型。参数要与 OPEN 的对象类型参数相同，不能为 HETEROGENEOUS 对象类型。
- GRANTEE  
被赋予权限的对象。
- OBJECT\_COUNT  
返回对象个数的最大值。

**返回值**

以 DDL 返回对象的元数据

**异常：**

INVALID\_ARGVAL：参数值为 NULL，或者参数值非法。

OBJECT\_NOT\_FOUND：指定的对象数据库中不存在。

## 7. GET\_DEPENDENT\_DDL ( )

该函数用于返回依赖对象的 DDL 语句。

语法如下：

---

```
FUNCTION GET_DEPENDENT_DDL (
    OBJECT_TYPE      IN VARCHAR2,
    BASE_OBJECT_NAME IN VARCHAR2,
    BASE_OBJECT_SCHEMA IN VARCHAR2 DEFAULT NULL,
    OBJECT_COUNT      IN NUMBER   DEFAULT 10000
)RETURN CLOB;
```

---

**参数详解**

- OBJECT\_TYPE  
对象类型。参数要与 OPEN 的对象类型参数相同，不能为 HETEROGENEOUS 对象类型。
- BASE\_OBJECT\_NAME  
基对象名称。内部使用 BASE\_OBJECT\_NAME 过滤器过滤。
- BASE\_OBJECT\_SCHEMA  
基对象模式。内部使用 BASE\_OBJECT\_SCHEMA 过滤器过滤。
- OBJECT\_COUNT  
返回对象个数的最大值。

**返回值**

以 DDL 返回对象的元数据。

**异常：**

INVALID\_ARGVAL：参数值为 NULL，或者参数值非法。

OBJECT\_NOT\_FOUND 指定的对象数据库中不存在。

**8. SET\_FILTER()**

该过程用于过滤待返回的对象。

语法如下：

```
PROCEDURE SET_FILTER (
    HANDLE                IN  NUMBER,
    NAME                  IN  VARCHAR2,
    VALUE                 IN  VARCHAR2
);
```

**参数详解**

- HANDLE  
句柄，由 DBMS\_METADATA.OPEN 输出。
- NAME  
过滤器的名称，详见下表。
- VALUE  
过滤器的值，与 NAME 相对应，详见下表。

表 SET\_FILTER 过程中的过滤器名称

对象类型	名称	数据类型	含义
可命名对象	NAME	文本	通过对象名过滤
可命名对象	NAME_EXPR	文本表达式	通过对象名的 where 表达式过滤，满足条件的返回
可命名对象	EXCLUDE_NAME_EXPR	文本表达式	通过对象名的 where 表达式过滤，不满足条件的返回

模式对象	SCHEMA	文本	通过模式名过滤，如果是公有同义词则指定其模式为 PUBLIC
模式对象	SCHEMA_EXPR	文本表达式	通过对象名的 where 表达式过滤，默认为当前模式
表、索引、表级导出	TS	文本	通过表空间名过滤
表、索引、表级导出	TABLESPACE_EXPR	文本表达式	通过表空间名的 where 表达式过滤
依赖对象	BASE_OBJECT_NAME	文本	通过基对象名过滤。
依赖对象	BASE_OBJECT_SCHEMA	文本	通过基对象模式过滤。如果使用 BASE_OBJECT_NAME 过滤，默认是当前模式。
依赖对象	BASE_OBJECT_NAME_EXPR	文本表达式	通过基对象名的 where 表达式过滤，满足条件的返回。不适用于模式和库级触发器。
依赖对象	EXCLUDE_BASE_OBJECT_NAME_EXPR	文本表达式	通过基对象名的 where 表达式过滤，不满足条件的返回。不适用于模式和库级触发器。
依赖对象	BASE_OBJECT_SCHEMA_EXPR	文本表达式	通过基对象模式的 where 表达式过滤。
依赖对象	BASE_OBJECT_TYPE	文本	通过基对象的对象类型过滤。
依赖对象	BASE_OBJECT_TYPE_EXPR	文本表达式	通过基对象的对象类型表达式过滤。
依赖对象	BASE_OBJECT_TS	文本	通过基对象的表空间过滤。
依赖对象	BASE_OBJECT_TS_EXPR	文本表达式	通过基对象的表空间 where 表达式过滤。
被授权对象	GRANTEE	文本	被授权的用户或角色。AUDIT_obj 只能过滤跟 grantee 有关的其他 grant 过滤器不适用。
被授权对象	ROLE_NAME	文本	通过权限或角色的名称过滤

被授权对象	ROLE_NAME_EXPR	文本表达式	通过权限或角色的 where 表达式名称过滤
被授权对象	GRANTEE_EXPR	文本表达式	通过被授权者名称的 where 表达式过滤，符合 条件的返回
被授权对象	EXCLUDE_GRANTEE_EXPR	文本表达式	通过被授权者名称的 where 表达式过滤，不 符合条件的返回
对象权限 (OBJECT_GRANT )	GRANTOR	文本	通过授权者过滤，对于 系统用户 (sysdba sysauditor sysso) 赋予的权限在 权限表里面 ID 记录的 都是-1，所以设定了一个 grantor 叫 "\$SUPER", 当 filter 中设定 grantor 为 \$SUPER 时，系统用户 赋予的权限将全部返回 而不区分具体是哪个系 统用户
所有对象	CUSTOM_FILTER	文本	用户自定义过滤，要求 填写完整的 where 子 句，如 "NAME = 'T1'"
模式级导出	SCHEMA	文本	通过模式名过滤
模式级导出	SCHEMA_EXPR	文本表达式	通过模式名 where 表达 式过滤，有以下两种情 况：1. 获取模式对象； 2. 获取依赖于此模式的 对象。默认情况下，将 选中当前用户的对象。
表级导出	SCHEMA	文本	通过模式名过滤
表级导出	SCHEMA_EXPR	文本表达式	通过模式名 where 表达 式过滤，有以下两种情 况：1. 获取模式下的表； 2. 获取依赖于表的对 象。默认情况下，将选 中当前用户的对象。
表级导出	NAME	文本	通过表名过滤出表及依

			依赖表的对象
表级导出	NAME_EXPR	文本表达式	通过表名的 where 表达式过滤出表及依赖表的对象

注：约束的 BASE\_OBJECT\_TYPE 是 UTAB，注释的 BASE\_OBJECT\_TYPE 为 TABLE 或者 VIEW。索引、OBJECT\_GRANT 的 BASE\_OBJECT\_TYPE 为 UTAB，TABLE\_STATISTICS，INDEX\_STATISTICS，COL\_STATISTICS 的 BASE\_OBJECT\_TYPE 分别对应为 T、I、C。

### 错误处理

INVALID\_ARGVAL：参数为 NULL，或者参数值非法。

INVALID\_OPERATION：非法操作。在调用 FETCH\_XXX 之后，不允许再调用 SET\_FILTER。

INCONSISTENT\_ARGS：参数不一致，包括如下状况：

- 过滤器名字与 DBMS\_METADATA.OPEN 中指定的类型不匹配。
- 过滤器的名字与 OBJECT\_TYPE\_PATH 不匹配。
- OBJECT\_TYPE\_PATH 与 DBMS\_METADATA.OPEN 中指定的类型不匹配。
- 输入的过滤器的 VALUE 与期待的数据类型不匹配。

### 9. SET\_PARSE\_ITEM()

该过程用于解析对象的属性。

语法如下：

```
PROCEDURE SET_PARSE_ITEM (
    HANDLE          IN  NUMBER,
    NAME            IN  VARCHAR2,
    OBJECT_TYPE     IN  VARCHAR2 DEFAULT NULL
);
```

### 参数详解

- HANDLE  
OPEN 函数返回的句柄。
- NAME  
被解析对象属性的名称，具体的可被解析的对象属性名称见下表。
- OBJECT\_TYPE  
应用于不同对象的集合，不设置时解析所有的对象。

表 SET\_PARSE\_ITEM 可解析的对象属性

对象类型	属性名称	含义
所有对象	VERB	如果调用 FETCH_DDL，表 ku\$_ddl 每一行 ddltext 列中的动词将被返回。如果 ddlText 是 SQL DDL 语句，SQL 中的动词（如 CREATE，GRANT，AUDIT）被返回。如果 ddlText 是过程（如

		DBMS_METADATA.FETCH_DDL( ) 那么 package.procedure-name 被返回
所有对象	OBJECT_TYPE	如果调用 FETCH_DDL, ddlText 是 SQL DDL 语句且包含动词 CREATE 或者 ALTER, DDL 语句中的对象类型将被返回 (如 TABLE, PKG_BODY 等等)。否则“表 OPEN 函数可返回的对象类型名称”中的对象类型被返回
模式对象	SCHEMA	返回对象的模式, 如果不是模式对象则没有返回值
命名的对象	NAME	返回对象的名称, 如果不是命名的对象则没有返回值
表、表数据、索引	TABLESPACE	返回对象表空间名称, 如果是分区表则返回默认的表空间。对于 TABLE_DATA 对象, 总是返回行所在的表空间
触发器	ENABLE	返回触发器是禁用还是启用状态
依赖对象	BASE_OBJECT_NAME	返回基对象名称, 如果不是依赖对象则没有返回值
依赖对象	BASE_OBJECT_SCHEMA	返回基对象所属的模式, 如果不是依赖对象则没有返回值
依赖对象	BASE_OBJECT_TYPE	返回基对象所属类型, 如果不是依赖对象则没有返回值

注: 对象类型是可解析的对象属性所适用的范围; BASE\_OBJECT\_NAME, BASE\_OBJECT\_SCHEMA, BASE\_OBJECT\_TYPE 这三个属性只解析约束、索引、全文索引、表级/视图级触发器。

#### 使用说明:

FETCH\_XXX 函数可以返回对象的 DDL 语句, 使用 SET\_PARSE\_ITEM 则可以返回对象的各个属性, 可以多次调用 SET\_PARSE\_ITEM 来解析和返回多个解析项, 返回的解析项存于表 KU\$\_PARSED\_ITEMS 中。

#### 错误处理

INVALID\_ARGVAL: 参数为 NULL, 或者参数值非法。

INVALID\_OPERATION: 非法操作。SET\_PARSE\_ITEM 只能用于 FETCH\_XXX 函数之前。第一次调用 FETCH\_XXX 后, 不允许再调用 SET\_PARSE\_ITEM。

#### 10. GET\_QUERY( )

该函数用于获取查询文本。

语法如下:

```
FUNCTION GET_QUERY (
    HANDLE IN NUMBER)
RETURN VARCHAR2;
```

#### 参数详解

HANDLE: OPEN 函数返回的句柄。

#### 返回值

将被用于 FENTH\_XXX 函数的查询文本。

#### 错误处理

INVALID\_ARGVAL: 参数值为 NULL, 或非法。

11. SET\_COUNT ( )

该过程用于指定一次 FETCH\_XXX 函数调用所返回对象个数的最大值。

语法如下:

---

```
PROCEDURE SET_COUNT (
    HANDLE          IN NUMBER,
    VALUE           IN NUMBER
);
```

---

#### 参数详解

- HANDLE  
OPEN 函数返回的句柄。
- VALUE  
设定的最大值。

#### 错误处理

INVALID\_ARGVAL: 参数值为 NULL 或非法;

INVALID\_OPERATION: 非法操作。SET\_COUNT 只能用于第一次调用 FETCH\_XXX 函数之前, 第一次调用 FETCH\_XXX 后, 不允许再调用 SET\_COUNT。

## 10.3 错误处理

错误、异常情况释义:

INVALID\_ARGVAL: 非法的参数数据。

OBJECT\_NOT\_FOUND: 未找到对象。

INVALID\_OPERATION: 无效的过程/函数调用顺序。

INCONSISTENT\_ARGS: 对象类型与过滤器不匹配。

## 10.4 举例说明

使用包内的过程和函数之前, 如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_METADATA');
```

例 1 获取表和存储过程的 DDL 语句。

```
--建表
CREATE TABLE "SYSDBA"."T1"
(
    "C1" CHAR(10) NOT NULL,
```

```
"C2" CHAR(10),
CONSTRAINT "PK" PRIMARY KEY("C1")) STORAGE(ON "MAIN", CLUSTERBTR);
--获取对象的 DDL 语句
SELECT DBMS_METADATA.GET_DDL('TABLE','T1','SYSDBA');
```

执行结果:

```
CREATE TABLE "SYSDBA"."T1"
(
"C1" CHAR(10) NOT NULL,
"C2" CHAR(10),
CONSTRAINT "PK" PRIMARY KEY("C1")) STORAGE(ON "MAIN", CLUSTERBTR);
```

例 2 使用 DBMS\_METADATA 程序化接口获取元数据

```
--建表
CREATE TABLE "RESOURCES"."TIMECARDS"
(
"EMPLOYEE_ID" NUMBER(6,0),
"WEEK" NUMBER(2,0),
"JOB_ID" VARCHAR2(10),
"HOURS_WORKED" NUMBER(4,2),
FOREIGN KEY ("EMPLOYEE_ID")
REFERENCES "RESOURCES"."EMPLOYEE" ("EMPLOYEEID")
) STORAGE(ON "MAIN", CLUSTERBTR) ;

--建函数
CREATE OR REPLACE FUNCTION GET_TABLE_MD RETURN CLOB IS
-- 定义局部变量
H NUMBER; --HANDLE RETURNED BY OPEN
TH NUMBER; -- HANDLE RETURNED BY ADD_TRANSFORM
DOC CLOB;
BEGIN
-- 指定对象类型
H := DBMS_METADATA."OPEN" ('TABLE');

-- 使用过滤器返回特定的对象
DBMS_METADATA.SET_FILTER(H,'SCHEMA','RESOURCES');
DBMS_METADATA.SET_FILTER(H,'NAME','TIMECARDS');

-- 获取对象
DOC := DBMS_METADATA.FETCH_CLOB(H);

-- 释放资源
DBMS_METADATA."CLOSE" (H);
RETURN DOC;
END;
/
SELECT GET_TABLE_MD;
```



查询结果如下：

```
CREATE TABLE "RESOURCES"."TIMECARDS"
(
  "EMPLOYEE_ID" NUMBER(6,0),
  "WEEK" NUMBER(2,0),
  "JOB_ID" VARCHAR2(10),
  "HOURS_WORKED" NUMBER(4,2),
  FOREIGN KEY("EMPLOYEE_ID")
REFERENCES "RESOURCES"."EMPLOYEE"("EMPLOYEEID")
) STORAGE(ON "MAIN", CLUSTERBTR) ;
```

### 例 3 使用 DBMS\_METADATA 浏览接口获取元数据

```
SELECT DBMS_METADATA.GET_DDL('TABLE','TIMECARDS','RESOURCES');
```

查询结果同例 2。

### 例 4 获取多个对象

```
DROP TABLE MY_METADATA;
CREATE TABLE MY_METADATA (MD CLOB);
CREATE OR REPLACE PROCEDURE GET_TABLES_MD IS
-- 定义局部变量
H      NUMBER;          -- HANDLE RETURNED BY 'OPEN'
TH     NUMBER;          -- HANDLE RETURNED BY 'ADD_TRANSFORM'
DOC    CLOB;             -- METADATA IS RETURNED IN A CLOB
BEGIN
-- 指定对象类型
H := DBMS_METADATA."OPEN" ('TABLE');

-- 使用过滤器指定模式
DBMS_METADATA.SET_FILTER(H,'SCHEMA','SYSDBA') ;

-- 获取对象
LOOP
  DOC := DBMS_METADATA.FETCH_CLOB(H);
  -- 当没有对象可获取时，FETCH_CLOB 返回 NULL
  EXIT WHEN DOC IS NULL;
  -- 将元数据存入表中
  INSERT INTO MY_METADATA(MD) VALUES (DOC);
  COMMIT;
END LOOP;
-- 释放资源
DBMS_METADATA."CLOSE" (H);
END;
/
CALL GET_TABLES_MD;
```

```
SELECT * FROM MY_METADATA;
```

### 例 5 使用 PARSE ITEMS 解析特定的元数据属性

```
DROP TABLE MY_METADATA;
```

```
CREATE TABLE MY_METADATA (
```

```
    OBJECT_TYPE    VARCHAR2(30),
```

```
    NAME           VARCHAR2(30),
```

```
    MD             CLOB);
```

```
CREATE OR REPLACE PROCEDURE GET_TABLES_AND_INDEXES IS
```

```
-- 定义局部变量
```

```
H1      NUMBER;          -- 打开表返回的句柄
```

```
DOC     DBMS_METADATA.KU$_DDL;      -- KU$_DDL, 返回的元数据
```

```
DDL     CLOB;             -- 对象的 DDL 语句
```

```
PI      DBMS_METADATA.KU$_PARSED_ITEMS;  --包含在 KU$_DDL 中的对象返回的解析项
```

```
OBJNAME VARCHAR2(30);  -- 解析对象的名称
```

```
BEGIN
```

```
-- 指定对象类型: TABLE.
```

```
H1 := DBMS_METADATA."OPEN" ('TABLE');
```

```
-- 将表名作为解析项返回
```

```
DBMS_METADATA.SET_PARSE_ITEM(H1, 'NAME');
```

```
-- 设置循环: 获取 TABLE 对象
```

```
LOOP
```

```
    DOC := DBMS_METADATA.FETCH_DDL(H1);
```

```
--当没有对象可获取时, FETCH_CLOB 返回 NULL
```

```
    EXIT WHEN DOC IS NULL;
```

```
-- 循环 KU$_DDL 表中的行
```

```
FOR I IN DOC.FIRST..DOC.LAST LOOP
```

```
    DDL := DOC(I).DDLTEXT;
```

```
    PI := DOC(I).PARSEDITEM;
```

```
-- 循环返回的解析项
```

```
IF PI IS NOT NULL AND PI.COUNT > 0 THEN
```

```
    FOR J IN PI.FIRST..PI.LAST LOOP
```

```
        IF PI(J).ITEM='NAME' THEN
```

```
            OBJNAME := PI(J).VALUE;
```

```
        END IF;
```

```
    END LOOP;
```

```
END IF;
```

```
-- 将该对象的信息插入表 MY_METADATA 中
```

```
INSERT INTO MY_METADATA(OBJECT_TYPE, NAME, MD)
```

```
VALUES ('TABLE', OBJNAME, DDL);
```

```
        COMMIT;
    END LOOP;
END LOOP;
DBMS_METADATA."CLOSE" (H1);
END;
/
CALL GET_TABLES_AND_INDEXES;
SELECT * FROM MY_METADATA;
```

## 11 DBMS\_OBFUSCATION\_TOOLKIT 包

为了保护敏感数据，DM 提供一个数据加密包 DBMS\_OBFUSCATION\_TOOLKIT。利用这个包，用户可以对数据进行 DES、DES3 加密，或者对数据进行 MD5 散列。

### 11.1 相关方法

DBMS\_OBFUSCATION\_TOOLKIT 包中包含的过程和函数如下详细介绍：

#### 1. DES3DECRYPT

对 VARBINARY 数据进行 DES3 解密。

语法如下：

---

```
PROCEDURE DES3DECRYPT(
    INPUT          IN    VARBINARY,
    KEY            IN    VARBINARY,
    DECRYPTED_DATA  OUT   VARBINARY,
    WHICH          IN    INT          DEFAULT 0,
    IV             IN    VARBINARY    DEFAULT NULL
);
```

---

#### 返回值

解密后的 VARBINARY 数据。

对 VARCHAR2 数据进行 DES3 解密。

语法如下：

---

```
PROCEDURE DES3DECRYPT(
    INPUT_STRING   IN    VARCHAR2,
    KEY_STRING     IN    VARCHAR2,
    DECRYPTED_STRING OUT   VARCHAR2,
    WHICH          IN    INT          DEFAULT 0,
    IV_STRING      IN    VARCHAR2    DEFAULT NULL
);
```

---

#### 返回值

解密后的 VARCHAR2 类型数据。

对 VARBINARY 数据进行 DES3 解密。

语法如下：

---

```
FUNCTION DES3DECRYPT(
    INPUT          IN    VARBINARY,
```

---

---

```

KEY          IN  VARBINARY,
WHICH        IN  INT DEFAULT 0,
IV           IN  VARBINARY DEFAULT NULL
)RETURN VARBINARY;

```

---

### 返回值

解密后的 VARBINARY 数据。

对 VARCHAR2 数据进行 DES3 解密。

语法如下：

---

```

FUNCTION DES3DECRYPT(
    INPUT_STRING IN  VARCHAR2,
    KEY_STRING   IN  VARCHAR2,
    WHICH        IN  INT          DEFAULT 0,
    IV_STRING    IN  VARCHAR2    DEFAULT NULL
)RETURN VARCHAR2;

```

---

### 返回值

解密后的 VARCHAR2 数据。

### 参数详解

- INPUT\_STRING 输入参数，需要解密的数据。
- KEY\_STRING 输入参数，解密数据使用的密钥。
- WHICH  
输入参数，解密模式，可选值 0、1。0 表示双密钥 DES3 解密，1 表示三密钥 DES3 解密。
- IV\_STRING 输入参数，解密数据使用的初始化向量。

## 2. DES3ENCRYPT

对 VARBINARY 数据进行 DES3 加密。

语法如下：

---

```

PROCEDURE DES3ENCRYPT(
    INPUT          IN  VARBINARY,
    KEY            IN  VARBINARY,
    ENCRYPTED_DATA OUT  VARBINARY,
    WHICH          IN  INT          DEFAULT 0,
    IV             IN  VARBINARY    DEFAULT NULL
);

```

---

### 参数详解

- INPUT 输入参数，需要加密的数据。
- KEY 输入参数，加密数据使用的密钥。

- ENCRYPTED\_DATA 输出参数，被加密后的数据。
- WHICH  
输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。
- IV 输入参数，加密数据使用的初始化向量。

对 VARCHAR2 数据进行 DES3 加密。

语法如下：

---

```
PROCEDURE DES3ENCRYPT(
    INPUT_STRING      IN      VARCHAR2,
    KEY_STRING        IN      VARCHAR2,
    ENCRYPTED_STRING  OUT      VARCHAR2,
    WHICH             IN      INT          DEFAULT 0,
    IV_STRING         IN      VARCHAR2    DEFAULT NULL
);
```

---

#### 参数详解

- INPUT\_STRING 输入参数，需要加密的数据。
- KEY\_STRING 输入参数，加密数据使用的密钥。
- ENCRYPTED\_STRING 输出参数，被加密后的数据。
- WHICH  
输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。
- IV\_STRING 输入参数，加密数据使用的初始化向量。

对 VARBINARY 数据进行 DES3 加密。

语法如下：

---

```
FUNCTION DES3ENCRYPT(
    INPUT      IN  VARBINARY,
    KEY        IN  VARBINARY,
    WHICH      IN  INT          DEFAULT 0,
    IV        IN  VARBINARY    DEFAULT NULL
)RETURN VARBINARY;
```

---

#### 参数详解

- INPUT 输入参数，需要加密的数据。
- KEY 输入参数，加密数据使用的密钥。
- WHICH  
输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。

密。

- IV 输入参数，加密数据使用的初始化向量。

### 返回值

返回值加密后的 VARBINARY 数据。

对 VARCHAR2 数据进行 DES3 加密。

语法如下：

---

```
FUNCTION DES3ENCRYPT(
    INPUT_STRING IN VARCHAR2,
    KEY_STRING    IN VARCHAR2,
    WHICH         IN INT          DEFAULT 0,
    IV_STRING     IN VARCHAR2    DEFAULT NULL
)RETURN VARCHAR2;
```

---

### 参数详解

- INPUT\_STRING 输入参数，需要加密的数据。
- KEY\_STRING 输入参数，加密数据使用的密钥。
- WHICH  
输入参数，加密模式，可选值 0、1。0 表示双密钥 DES3 加密，1 表示三密钥 DES3 加密。
- IV\_STRING 输入参数，加密数据使用的初始化向量。

### 返回值

返回值加密后的 VARCHAR2 数据。

## 3. DES3GETKEY

生成 VARBINARY 类型数据的 DES3 加密密钥。

语法如下：

---

```
PROCEDURE DES3GETKEY(
    WHICH         IN INT          DEFAULT 0,
    SEED          IN  VARBINARY,
    KEY           OUT VARBINARY
);
```

---

### 参数详解

- WHICH  
输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3。
- SEED 输入参数，获取 DES3 密钥使用的种子。
- KEY 输出参数，VARBINARY 类型的 DES3 加密密钥。

生成 VARCHAR2 类型的 DES3 加密密钥。

语法如下：

---

```
PROCEDURE DES3GETKEY(
    WHICH          IN   INT          DEFAULT 0,
    SEED_STRING    IN   VARCHAR2,
    KEY            OUT  VARCHAR2
);
```

---

#### 参数详解

- WHICH  
输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3。
- SEED\_STRING 输入参数，获取 DES3 密钥使用的种子。
- KEY 输出参数，VARCHAR2 类型的 DES3 加密密钥。

生成 VARBINARY 类型的 DES3 加密密钥。

语法如下：

---

```
FUNCTION DES3GETKEY(
    WHICH IN   INT          DEFAULT 0,
    SEED  IN   VARBINARY
)RETURN VARBINARY;
```

---

#### 参数详解

- WHICH  
输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3。
- SEED 输入参数，获取 DES3 密钥使用的种子。

#### 返回值

返回值 VARBINARY 类型的 DES3 加密密钥。

生成 VARCHAR2 类型的 DES3 加密密钥。

语法如下：

---

```
FUNCTION DES3GETKEY(
    WHICH          IN   INT          DEFAULT 0,
    SEED_STRING    IN   VARCHAR2
)RETURN VARCHAR2;
```

---

#### 参数详解

- WHICH  
输入参数，DES3 工作模式，可选值 0、1。0 表示双密钥 DES3，1 表示三密钥 DES3。
- SEED\_STRING 输入参数，获取 DES3 密钥使用的种子。



**返回值**

返回值 VARCHAR2 类型的 DES3 加密密钥。

**4. DESDECRYPT**

对 VARBINARY 数据进行 DES 解密。

语法如下：

---

```
PROCEDURE DESDECRYPT(
    INPUT          IN    VARBINARY,
    KEY            IN    VARBINARY,
    DECRYPTED_DATA  OUT   VARBINARY
);
```

---

**参数详解**

- INPUT 输入参数，需要解密的数据。
- KEY 输入参数，解密数据使用的密钥。
- DECRYPTED\_DATA 输出参数，被解密后的数据。

对 VARCHAR2 数据进行 DES 解密。

语法如下：

---

```
PROCEDURE DESDECRYPT(
    INPUT_STRING    IN    VARCHAR2,
    KEY_STRING      IN    VARCHAR2,
    DECRYPTED_STRING OUT   VARCHAR2
);
```

---

**参数详解**

- INPUT\_STRING 输入参数，需要解密的数据。
- KEY\_STRING 输入参数，解密数据使用的密钥。
- DECRYPTED\_STRING 输出参数，被解密后的数据。

对 VARBINARY 数据进行 DES 解密。

语法如下：

---

```
FUNCTION DESDECRYPT(
    INPUT          IN  VARBINARY,
    KEY            IN  VARBINARY)
RETURN VARBINARY;
```

---

**参数详解**

- INPUT 输入参数，需要解密的数据。

- KEY 输入参数，解密数据使用的密钥。

### 返回值

返回值解密后的 VARBINARY 数据。

对 VARCHAR2 数据进行 DES 解密。

语法如下：

---

```
FUNCTION DESDECRYPT(
    INPUT_STRING  IN  VARCHAR2,
    KEY_STRING    IN  VARCHAR2
)RETURN VARCHAR2;
```

---

### 参数详解

- INPUT\_STRING 输入参数，需要解密的数据。
- KEY\_STRING 输入参数，解密数据使用的密钥。

### 返回值

返回值解密后的 VARCHAR2 数据。

## 5. DESENCRYPT

对 VARBINARY 数据进行 DES 加密。

语法如下：

---

```
PROCEDURE DESENCRYPT(
    INPUT          IN    VARBINARY,
    KEY            IN    VARBINARY,
    ENCRYPTED_DATA  OUT   VARBINARY
);
```

---

### 参数详解

- INPUT 输入参数，需要加密的数据。
- KEY 输入参数，加密数据使用的密钥。
- ENCRYPTED\_DATA 输出参数，被加密后的数据。

对 VARCHAR2 数据进行 DES 加密。

语法如下：

---

```
PROCEDURE DESENCRYPT(
    INPUT_STRING    IN    VARCHAR2,
    KEY_STRING      IN    VARCHAR2,
    ENCRYPTED_STRING OUT   VARCHAR2
);
```

---

**参数详解**

- INPUT\_STRING 输入参数，需要加密的数据。
- KEY\_STRING 输入参数，加密数据使用的密钥。
- ENCRYPTED\_STRING 输出参数，被加密后的数据。

对 VARBINARY 数据进行 DES 加密。

语法如下：

---

```
FUNCTION DESENCRYPT(  
    INPUT          IN  VARBINARY,  
    KEY            IN  VARBINARY  
)RETURN VARBINARY;
```

---

**参数详解**

- INPUT 输入参数，需要加密的数据。
- KEY 输入参数，加密数据使用的密钥。
- 返回值加密后的 VARBINARY 数据。

对 VARCHAR2 数据进行 DES 加密。

语法如下：

---

```
FUNCTION DESENCRYPT(  
    INPUT_STRING IN  VARCHAR2,  
    KEY_STRING   IN  VARCHAR2  
)RETURN VARCHAR2;
```

---

**参数详解**

- INPUT\_STRING 输入参数，需要加密的数据。
- KEY\_STRING 输入参数，加密数据使用的密钥。

**返回值**

返回值加密后的 VARCHAR2 数据。

**6. DESGETKEY**

生成 VARBINARY 类型的 DES 加密密钥。

语法如下：

---

```
PROCEDURE DESGETKEY(  
    SEED          IN  VARBINARY,  
    KEY           OUT VARBINARY  
) ;
```

---

**参数详解**

- SEED 输入参数，获取 DES 密钥使用的种子。
- KEY 输出参数，VARBINARY 类型的 DES 加密密钥。

生成 VARCHAR2 类型的 DES 加密密钥。

语法如下：

---

```
PROCEDURE DESGETKEY(  
    SEED_STRING IN    VARCHAR2,  
    KEY          OUT  VARCHAR2  
);
```

---

**参数详解**

- SEED\_STRING 输入参数，获取 DES 密钥使用的种子。
- KEY 输出参数，VARCHAR2 类型的 DES 加密密钥。

生成 VARBINARY 类型的 DES 加密密钥。

语法如下：

---

```
FUNCTION DESGETKEY(  
    SEED IN  VARBINARY  
)RETURN VARBINARY;
```

---

**参数详解**

- SEED 输入参数，获取 DES 密钥使用的种子。

**返回值**

返回 VARBINARY 类型的 DES 加密密钥。

生成 VARCHAR2 类型的 DES 加密密钥。

语法如下：

---

```
FUNCTION DESGETKEY(  
    SEED_STRING IN  VARCHAR2  
)RETURN VARCHAR2;
```

---

**参数详解**

- SEED\_STRING 输入参数，获取 DES 密钥使用的种子。
- 返回值 VARCHAR2 类型的 DES 加密密钥。

**7. MD5**

生成 VARBINARY 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

语法如下：

---

```
PROCEDURE MD5 (
    INPUT          IN    VARBINARY,
    CHECKSUM       OUT   VARBINARY
);
```

---

#### 参数详解

- INPUT 输入参数，需要进行散列的数据。
- CHECKSUM 输出参数，经 MD5 散列后的数据。

生成 VARCHAR2 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

语法如下：

---

```
PROCEDURE MD5 (
    INPUT_STRING    IN    VARCHAR2,
    CHECKSUM_STRING OUT   VARCHAR2
);
```

---

#### 参数详解

- INPUT\_STRING 输入参数，需要进行散列的数据。
- CHECKSUM\_STRING 输出参数，经 MD5 散列后的数据。

生成 VARBINARY 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

语法如下：

---

```
FUNCTION MD5 (
    INPUT          IN    VARBINARY
) RETURN VARBINARY;
```

---

#### 参数详解

- INPUT 输入参数，需要进行散列的数据。

返回值

返回 MD5 散列后的 VARBINARY 数据。

生成 VARCHAR2 类型数据的 MD5 散列值。同一台机器配置，每次运行的结果一致。

语法如下：

---

```
FUNCTION MD5 (
    INPUT_STRING    IN    VARCHAR2
) RETURN VARCHAR2;
```

---

#### 参数详解

- INPUT\_STRING 输入参数，需要进行散列的数据。

返回值

返回 MD5 散列后的 VARCHAR2 数据。

## 11.2 使用说明

1. 被加解密的数据不能为空。
2. DM 的 DES3 加解密目前只支持双密钥，即 WHICH 默认为 0 的情况。
3. IV 加解密数据使用的初始化向量目前不使用，默认为 NULL，此时采用系统默认的初始化向量。
4. DES3GETKEY 和 DESGETKEY 生成加密密钥的种子 SEED 目前不使用，因为 DM 用于生成密钥的环境已经配置好，只需要根据密钥的大小生成指定长度的密钥即可。
5. DES 和 DES3 加解密的块大小为 8BYTES，因此用于加解密的数据必须是 8 的整数倍。
6. DES 加解密使用的密钥长度为 8，多余 8 后的字符被忽略。
7. 双密钥 DES3 加解密使用的密钥长度为 16，多余 16 后的字符被忽略。
8. 三密钥 DES3 加解密使用的密钥长度为 24，多余 24 后的字符被忽略。
9. 十六进制在 DM 对应的是 VARBINARY 数据类型，在 ORACLE 中对应的是 RAW 类型。
10. MD5 主要用途是对数据进行散列，用于校验。不同的机器生成的散列值不同。

## 11.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_OBFUSCATION_TOOLKIT');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_OUTPUT');
SET SERVEROUTPUT ON; --DBMS_OUTPUT.PUT_LINE 需要设置这条语句，才能打印出消息
```

例 1 分别使用 DES3ENCRYPT 加密算法、DES3DECRYPT 解密算法对 VARBINARY 类型数据进行加密解密。

```
DECLARE
    RAW_INPUT      VARBINARY(128) := '74696765727469676572746967657274';
    -- 'TIGERTIGERTIGERT'
    RAW_KEY         VARBINARY(128) := '73636F747473636F747473636F747473';
    -- 'SCOTTSCOTTSCOTTS'
    ENCRYPTED_RAW    VARBINARY(2048);
    DECRYPTED_RAW    VARBINARY(2048);

BEGIN
    DBMS_OUTPUT.PUT_LINE('> ===== BEGIN TEST RAW DATA =====');
    DBMS_OUTPUT.PUT_LINE('> RAW INPUT                      : ' ||
```

```

        RAW_INPUT);

DBMS_OBFUSCATION_TOOLKIT.DES3ENCRYPT(RAW_INPUT,
        RAW_KEY, ENCRYPTED_RAW );

DBMS_OUTPUT.PUT_LINE('> ENCRYPTED HEX VALUE           : ' ||
        ENCRYPTED_RAW);

DBMS_OBFUSCATION_TOOLKIT.DES3DECRYPT(ENCRYPTED_RAW,
        RAW_KEY, DECRYPTED_RAW);

DBMS_OUTPUT.PUT_LINE('> DECRYPTED RAW OUTPUT           : ' ||
        DECRYPTED_RAW);

DBMS_OUTPUT.PUT_LINE('> ');

IF RAW_INPUT =
        DECRYPTED_RAW THEN
        DBMS_OUTPUT.PUT_LINE('> RAW DES ENCRYPTION AND DECRYPTION SUCCESSFUL');
END IF;

END;
/

```

结果:

```

> ===== BEGIN TEST RAW DATA =====
> RAW INPUT                      : 74696765727469676572746967657274
> ENCRYPTED HEX VALUE            : F77BB98AF8E7F59E329C31027CAF6C98
> DECRYPTED RAW OUTPUT           : 74696765727469676572746967657274
>
> RAW DES ENCRYPTION AND DECRYPTION SUCCESSFUL

```

例 2 分别使用 DESENCRYPT 加密算法、DESDECRYPT 解密算法对 VARCHAR2 类型的数据进行加密、解密。本示例建库时，未指定 unicode 参数 (unicode=0)。

```

DECLARE
INPUT_STRING      VARCHAR2(8)  := 'DM123456';
KEY_STRING        VARCHAR2(8)  := 'PASSWORD';

ENCRYPTED_STRING   VARCHAR2(2048);
DECRYPTED_STRING   VARCHAR2(2048);

BEGIN
    DBMS_OBFUSCATION_TOOLKIT.DESENCRYPT(
        INPUT_STRING, KEY_STRING, ENCRYPTED_STRING );

    DBMS_OUTPUT.PUT_LINE('> ENCRYPTED STRING           : ' ||
        ENCRYPTED_STRING);

    DBMS_OBFUSCATION_TOOLKIT.DESDECRYPT(
        ENCRYPTED_STRING,
        KEY_STRING,
        DECRYPTED_STRING);

    DBMS_OUTPUT.PUT_LINE('> DECRYPTED OUTPUT           : ' ||

```

```
> ENCRYPTED STRING : ㄟ 0 姆苓
> DECRYPTED OUTPUT : DM123456
>
> DES ENCRYPTION AND DECRYPTION SUCCESSFUL
```

```
DECLARE

    RETVAL VARBINARY(100);

    INPUT_STRING VARBINARY(100) := '74696765727469676572746967657274';

    BEGIN

        DBMS_OUTPUT.PUT_LINE('> ===== BEGIN TEST VARBINARY DATA =====');

        DBMS_OUTPUT.PUT_LINE('> INPUT STRING                                : '
                                || INPUT_STRING);

        RETVAL := DBMS_OBFUSCATION_TOOLKIT.MD5(INPUT_STRING);

        DBMS_OUTPUT.PUT_LINE(' > MD5 STRING OUTPUT                        : ' ||
                                RETVAL);

    END;
```

```
> ===== BEGIN TEST VARBINARY DATA =====
> INPUT STRING                               : 74696765727469676572746967657274
> MD5 STRING OUTPUT                          : 831F2AA79221A0F8F330E43A76B53323
```

```

DECLARE
    RETVAL VARCHAR2(100);
    INPUT_STRING VARCHAR2(100) :=
'012345678901234567890123456789012345678901234567890123456789012345
67890123456789';
BEGIN
    DBMS_OUTPUT.PUT_LINE('> ===== BEGIN TEST STRING DATA =====');

    DBMS_OUTPUT.PUT_LINE('> INPUT STRING                                : '
                        || INPUT_STRING);

```



```
DBMS_OBFUSCATION_TOOLKIT.DES3GETKEY(SEED_STRING => INPUT_STRING, KEY =>
RETVAL);
    DBMS_OUTPUT.PUT_LINE('> DECRYPTED STRING OUTPUT          : ' ||
        RETVAL);
END;
/
```

结果:

```
> ===== BEGIN TEST STRING DATA =====
> INPUT STRING                : 0123456789012345678901234567890123456789012
34567890123456789012345678901234567890123456789
> DECRYPTED STRING OUTPUT      : 40XDW4FKLE8386JQ  --每次生成密码均不一样
```

## 12 DBMS\_OUTPUT 包

DBMS\_OUTPUT 系统包是为了在 DM 上兼容 oracle 的 DBMS\_OUTPUT 系统包。提供将文本行写入内存、供以后提取和显示的功能。为用户从 oracle 移植应用提供方便，功能上与 oracle 基本一致。

### 12.1 相关方法

#### 1. DBMS\_OUTPUT.DISABLE

禁用 DBMS\_OUTPUT 包。

语法如下：

---

```
DBMS_OUTPUT.DISABLE
```

---

#### 举例说明

```
DBMS_OUTPUT.DISABLE(); --禁用 DBMS_OUTPUT 包
```

#### 2. DBMS\_OUTPUT.ENABLE

启用 DBMS\_OUTPUT 包。

语法如下：

---

```
DBMS_OUTPUT.ENABLE (
    BUFFER_SIZE IN INT DEFAULT 20000
)
```

---

#### 参数详解

- buffer\_size

参数被忽略，只为兼容 oracle 语法。

#### 举例说明

```
DBMS_OUTPUT.ENABLE(); --启用 DBMS_OUTPUT 包
```

#### 3. DBMS\_OUTPUT.PUT\_LINE

输出字符串到缓冲区。如果设置 (SET SERVEROUTPUT ON;) 语句，可以打印出消息到客户端。

语法如下：

---

```
DBMS_OUTPUT.PUT_LINE(
    STR IN VARCHAR
)
```

---

#### 参数详解

- STR

VARCHAR 类型。

#### 4. DBMS\_OUTPUT.GET\_LINE

从缓冲区中读取一行信息。

语法如下：

```
DBMS_OUTPUT . GET_LINE(  
    LINE OUT VARCHAR,  
    STATUS OUT INT  
)
```

#### 参数详解

- LINE

缓冲区中的内容。

- STATUS

0 或 1。读取成功返回 0，反之为 1。

## 12.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_OUTPUT');
```

```
SET SERVEROUTPUT ON; --dbms_output.put_line 需要设置这条语句，才能打印出消息
```

例放入内容到缓冲区，并从缓冲区中读出

```
--启用 DBMS_OUTPUT 包  
DBMS_OUTPUT.ENABLE();  
--使用 put_line 和 get_line  
    declare  
        buffer varchar;  
        status int;  
    begin  
        dbms_output.put_line('达梦数据库有限公司');  
        dbms_output.get_line(buffer,status);  
        dbms_output.put_line('BUFFER: ' || buffer || ' status: ' || status);  
        commit;  
    end;  
--打印结果: BUFFER:达梦数据库有限公司 status:0
```

## 13 DBMS\_PAGE 包

### 13.1 使用前提

DBMS\_PAGE 包的使用依赖 DBMS\_BINARY 包。所以，在创建 DBMS\_PAGE 包之前，请先成功创建 DBMS\_BINARY 包。

### 13.2 索引页

DBMS\_PAGE 系统包的索引页信息。

#### 13.2.1 数据类型

DBMS\_PAGE 包的索引页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
DPH_T 类型	用于承载页类型及数据页头的所有信息包括：记录数、用户记录数、堆栈底部偏移、堆栈顶部偏移、分支号、碎片空间起始偏移、最小记录偏移、最大记录偏移、外部记录信息、已用空间
DPIH_T 类型	用于承载索引数据头的所有信息，包括：在 B 树中的层次、索引号、B 树中叶节点所在的段的组号、B 树中叶节点所在的段文件号、B 树中叶节点所在的段的页号、B 树中叶节点所在的段的偏移、B 树中内节点所在段的组号、B 树中内节点所在段的文件号、B 树中内节点所在段的页号、B 树中内节点所在段的偏移
DPICH_T 类型	用于承载索引控制头的所有信息，包括：下一个 ROWID、索引的已用空间、B 树中所有记录数、目前 IDENTITY 值
DPS_T 类型	用于承载页空间使用状况的信息，包括：堆底偏移、堆顶偏移、已用字节数、空闲字节数、使用率、堆顶剩余空间、碎片空间大小、碎片空间占总剩余比率、最大可插入记录长度
REC_T 类型	用于承载记录相关信息，包括：记录序号、记录偏移、记录长度、是否被删除、ROWID 或聚簇索引 ID、回滚地址-文件号、回滚地址-页号、回滚地址-偏移
FREE_LIST_T 类型	用于承载单个碎片空间的相关信息，包括：碎片偏移、碎片长度
CTL_BIT_ARR_T 类型	用于承载控制位的 TINYINT 类型的数组
DPH_ARR_T 类型	DPH_T 类型的数组
DPIH_ARR_T 类型	DPIH_T 类型的数组
DPICH_ARR_T 类型	DPICH_T 类型的数组

DPS_ARR_T 类型	DPS_T 类型的数组
REC_ARR_T 类型	REC_T 类型的数组
FREE_LIST_ARR_T 类型	FREE_LIST_T 类型的数组

### 13.2.2 相关方法

DBMS\_PAGE 包中的索引页部分所包含的过程和函数详细介绍如下：

#### 1. DATA\_PAGE\_HEAD\_LOAD/ DATA\_PAGE\_HEAD\_GET

获取数据页头信息。过程和函数功能相同。

语法如下：

```
PROCEDURE DATA_PAGE_HEAD_LOAD(
    TS_ID IN SMALLINT,
    FILE_ID IN SMALLINT,
    PAGE_NO IN INT,
    INFO OUT DPH_T
);
```

语法如下：

```
FUNCTION DATA_PAGE_HEAD_GET(
    TS_ID IN SMALLINT,
    FILE_ID IN SMALLINT,
    PAGE_NO IN INT
)RETURN DPH_T;
```

#### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。

#### 2. DATA\_PAGE\_IND\_HEAD\_LOAD/ DATA\_PAGE\_IND\_HEAD\_GET

获取索引数据页头信息。过程和函数功能相同。

语法如下：

```
PROCEDURE DATA_PAGE_IND_HEAD_LOAD(
    TS_ID IN SMALLINT,
    FILE_ID IN SMALLINT,
    PAGE_NO IN INT,
    INFO OUT DPIH_T
);
```

---

```
);
```

---

语法如下:

---

```
FUNCTION DATA_PAGE_IND_HEAD_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT  
)RETURN DPIH_T;
```

---

#### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。

3. DATA\_PAGE\_IND\_CTL\_HEAD\_LOAD/ DATA\_PAGE\_IND\_CTL\_HEAD\_GET

获取索引控制页头信息。过程和函数功能相同。

语法如下:

---

```
PROCEDURE DATA_PAGE_IND_CTL_HEAD_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    INFO OUT DPICH_T  
)
```

---

语法如下:

---

```
FUNCTION DATA_PAGE_IND_CTL_HEAD_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT  
)RETURN DPICH_T;
```

---

#### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。

---

#### 4. DATA\_PAGE\_SPACE\_INFO\_LOAD/ DATA\_PAGE\_SPACE\_INFO\_GET

获取页空间使用状况信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE DATA_PAGE_SPACE_INFO_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    INFO OUT DPS_T  
);
```

---

语法如下：

---

```
FUNCTION DATA_PAGE_SPACE_INFO_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT  
)RETURN DPS_T;
```

---

#### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。

---

#### 5. DATA\_PAGE\_REC\_BY\_SLOT\_NO\_LOAD/ DATA\_PAGE\_REC\_BY\_SLOT\_NO\_GET

获取指定序号的记录。过程和函数功能相同。

语法如下：

---

```
PROCEDURE DATA_PAGE_REC_BY_SLOT_NO_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    SLOT_NO IN SMALLINT,  
    REC OUT REC_T  
);
```

---

语法如下：

---

```
FUNCTION DATA_PAGE_REC_BY_SLOT_NO_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    SLOT_NO IN SMALLINT
```

---

---

```
) RETURN REC_T;
```

---

### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。
- slot\_no  
记录序号。

6. DATA\_PAGE\_CTL\_BIT\_BY\_SLOT\_NO\_LOAD/

DATA\_PAGE\_CTL\_BIT\_BY\_SLOT\_NO\_GET

获取指定序号的记录的控制位。过程和函数功能相同。

语法如下：

---

```
PROCEDURE DATA_PAGE_CTL_BIT_BY_SLOT_NO_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    SLOT_NO IN SMALLINT,  
    COL_NUM IN SMALLINT,  
    CTL_BIT OUT CTL_BIT_ARR_T  
);
```

---

语法如下：

---

```
FUNCTION DATA_PAGE_CTL_BIT_BY_SLOT_NO_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    SLOT_NO IN SMALLINT,  
    COL_NUM IN SMALLINT  
)  
RETURN CTL_BIT_ARR_T;
```

---

### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no



页号。

- slot\_no

记录序号。

- col\_num

要取得控制位的个数。

#### 7. DATA\_PAGE\_LOAD\_FREE\_LIST/ DATA\_PAGE\_GET\_FREE\_LIST

获取所有碎片空间大小及偏移。过程和函数功能相同。

语法如下：

---

```
PROCEDURE DATA_PAGE_LOAD_FREE_LIST(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    NO IN SMALLINT,  
    FREE_LIST OUT FREE_LIST_ARR_T  
);
```

---

语法如下：

---

```
FUNCTION DATA_PAGE_GET_FREE_LIST(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    NO IN SMALLINT  
)RETURN FREE_LIST_ARR_T;
```

---

#### 参数详解

- ts\_id

表空间 id。

- file\_id

文件号。

- page\_no

页号。

- no

要取到第几个空闲页。

#### 8. DATA\_PAGE\_TID\_LOAD/ DATA\_PAGE\_TID\_GET

输入表空间号、文件号及页号，获得表 ID。过程和函数功能相同。

语法如下：

---

```
PROCEDURE DATA_PAGE_TID_LOAD(  
    TS_ID IN SMALLINT,
```

---

---

```
FILE_ID IN SMALLINT,  
PAGE_NO IN INT,  
TID OUT BIGINT  
);
```

---

语法如下:

---

```
FUNCTION DATA_PAGE_TID_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT  
)RETURN INT;
```

---

### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。
- page\_no  
页号。

### 9. DATA\_PAGE\_TNAME\_LOAD/ DATA\_PAGE\_TNAME\_GET

输入表空间号、文件号及页号，输出表名。过程和函数功能相同。

语法如下:

---

```
PROCEDURE DATA_PAGE_TNAME_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT,  
    TNAME OUT VARCHAR(35)  
);
```

---

语法如下:

---

```
FUNCTION DATA_PAGE_TNAME_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    PAGE_NO IN INT  
)RETURN VARCHAR;
```

---

### 参数详解

- ts\_id  
表空间 id。
- file\_id  
文件号。

- page\_no

页号。

### 13.2.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_BINARY');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_PAGE');
```

例 1 获取数据页头信息的，利用包内类型 DPH\_ARR\_T 以及过程

DATA\_PAGE\_HEAD\_LOAD，这里假设 0 号表空间的 0 号文件 32 号页是一个数据页：

```
DECLARE
INFO DBMS_PAGE.DPH_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.DPH_T[1];
DBMS_PAGE.DATA_PAGE_HEAD_LOAD(0,0,32,INFO[1]);
SELECT * FROM ARRAY INFO;
END;
/
```

输出结果(具体数值取决于实际状况)：

N_SLOT	N_REC	HEAP_LOW	HEAP_HIGH	BRANCH_NO	FREE_LIST_OFF	REC_MIN_OFF	REC_MAX_OFF	EXTERNAL	USED_BYTE
4	2	155	0	0	98	82	90	<NULL>	152

例 2 获取页内所有用户记录的信息，利用包内类型 DPH\_T、REC\_T、REC\_ARR\_T 以及过程 DATA\_PAGE\_HEAD\_LOAD、DATA\_PAGE\_REC\_BY\_SLOT\_NO\_LOAD，这里假设 4 号表空间的 0 号文件 32 号页是一个数据页，并存储了两行记录：

```
DECLARE
HEAD DBMS_PAGE.DPH_T;
INFO DBMS_PAGE.REC_ARR_T;
BEGIN
DBMS_PAGE.DATA_PAGE_HEAD_LOAD(4,0,32,HEAD);
INFO = NEW DBMS_PAGE.REC_T[HEAD.N_REC];
FOR I IN 1..HEAD.N_REC LOOP
DBMS_PAGE.DATA_PAGE_REC_BY_SLOT_NO_LOAD(4,0,32,I,INFO[I]);
END LOOP;
SELECT * FROM ARRAY INFO;
END;
/
```

输出结果(具体数值取决于实际状况)：

SLOT_NO	OFFSET	LEN	IS_DEL	TRX_ID	CLU_ROWID	ROLL_ADDR_FILE	ROLL_ADDR_PAGE	ROLL_ADDR_OFF
1	117	19	0	2705	33559296	0	2	0

2	136	19	0	2706	50336512	0	3	0
---	-----	----	---	------	----------	---	---	---

### 13.3 INODE 页

INODE 页用于存储 INODE 控制块信息，INODE 控制块用于存储段的属性信息，每个 INODE 控制块对应一个段。INODE 页结构包括通用页头、INODE 页链接、以及 INODE 控制块信息。INODE 控制块结构包括段号、半满簇中使用页的个数、空闲簇链表、半满簇链表、满簇链表。

#### 13.3.1 数据类型

DBMS\_PAGE 包的 INODE 页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
INODE_PH_T 类型	用于承载 INODE 页头的所有信息，包括：所属表空间 ID、当前页文件 ID、当前页页号、前一个页的文件 ID、前一个页页号、后一个页文件 ID、后一个页页号、页类型、校验和、LSN
INODE_PAGE_LINK_T 类型	用于承载 INODE 页的链接信息，包括：前一个 INODE 页的文件 ID、页号与偏移量，后一个 INODE 页的文件 ID、页号与偏移量
INODE_T 类型	用于承载 INODE 项的所有信息，包括：段 ID、半满簇中使用页数、空闲簇个数、第一个空闲簇描述项的文件 ID、第一个空闲簇描述项的页号、到下一个空闲簇描述项的偏移量、最后一个空闲簇描述项的文件 ID、最后一个空闲簇描述项的页号、到下一个空闲簇描述项的偏移量、半满簇个数、第一个半满簇描述项的文件 ID、第一个半满簇描述项的页号、到下一个半满簇描述项的偏移量、最后一个半满簇描述项的文件 ID、最后一个半满簇描述项的页号、到下一个半满簇描述项的偏移量、满簇个数、第一个满簇描述项的文件 ID、第一个满簇描述项的页号、到下一个满簇描述项的偏移量、最后一个满簇描述项的文件 ID、最后一个满簇描述项的页号、到下一个满簇描述项的偏移量
INODE_PH_ARR_T 类型	INODE_PH_T 类型的数组
INODE_ARR_T 类型	INODE_T 类型的数组

### 13.3.2 相关方法

DBMS\_PAGE 包中的 INODE 页部分所包含的过程和函数详细介绍如下：

#### 1. DBMS\_PAGE. IPAGE\_HEAD\_LOAD/ IPAGE\_HEAD\_GET

获得 INODE 页中页头信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE IPAGE_HEAD_LOAD(  
    PAGE IN VARBINARY,  
    INFO OUT INODE_PH_T  
);  
  
FUNCTION IPAGE_HEAD_GET(  
    PAGE IN VARBINARY  
)RETURN INODE_PH_T;
```

---

#### 参数详解

- PAGE

页内容。

#### 2. DBMS\_PAGE. IPAGE\_NTH\_INODE\_LOAD/ IPAGE\_NTH\_INODE\_GET

获得 INODE 页中第 N 个 INODE 项的信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE IPAGE_NTH_INODE_LOAD(  
    PAGE IN VARBINARY,  
    N IN SMALLINT,  
    INFO OUT INODE_T  
);  
  
FUNCTION IPAGE_NTH_INODE_GET(  
    PAGE IN VARBINARY,  
    N IN SMALLINT  
)RETURN INODE_T;
```

---

#### 参数详解

- page

页内容。

- n

第几个 inode 项。

#### 3. DBMS\_PAGE. IPAGE\_ALL\_INODES\_LOAD/ IPAGE\_ALL\_INODES\_GET

获得 INODE 页中所有 INODE 项的信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE IPAGE_ALL_INODES_LOAD(  

```

---

```

    PAGE IN VARBINARY,
    INFO OUT INODE_ARR_T
);
FUNCTION IPAGE_ALL_INODES_GET(
    PAGE IN VARBINARY
)RETURN INODE_ARR_T;

```

### 参数详解

- PAGE  
页内容。

### 13.3.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```

SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_BINARY');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_PAGE');

```

#### 例 1 获取某 INODE 页的页头信息

```

DECLARE
PAGE8 VARBINARY;
INFO DBMS_PAGE.INODE_PH_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.INODE_PH_T[1];
DBMS_PAGE.PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_HEAD_LOAD(PAGE8,INFO[1]);
SELECT * FROM ARRAY INFO;
END;
/

```

结果如下（具体数值取决于实际状况）：

TSID	SELF_FILE_ID			SELF_PAGE_NO			PREV_FILE_ID		PREV_PAGE_NO	
	NEXT_FILE_ID			NEXT_PAGE_NO			PAGE_TYPE	CHECKSUM	LSN	
0	0	8	-1	-1	0	9	17	0	15270	

#### 例 2 获取某 INODE 项的信息

```

DECLARE
PAGE8 VARBINARY;
INFO DBMS_PAGE.INODE_ARR_T;
BEGIN
INFO = NEW DBMS_PAGE.INODE_T[10];
DBMS_PAGE.PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_NTH_INODE_LOAD(PAGE8,1,INFO[1]);
SELECT * FROM ARRAY INFO;
END;
/

```

执行结果，显示 INFO 中的内容是一个 INODE\_T 记录类型的查询结果， INODE\_T 中

的信息是获取的某 INODE 页中的某个 INODE 项的信息 (具体数值取决于实际状况)

例 3 获取某 INODE 页中所有 INODE 项信息

```
DECLARE
PAGE8  VARBINARY;
INFO  DBMS_PAGE.INODE_ARR_T;
BEGIN
INFO  = NEW DBMS_PAGE.INODE_T[150];
DBMS_PAGE.PAGE_LOAD(0,0,8,PAGE8);
DBMS_PAGE.IPAGE_ALL_INODES_LOAD(PAGE8,INFO);
SELECT * FROM ARRAY INFO;
END;
/
```

执行结果, 显示 INFO 中的内容也就是一个 INODE\_ARR\_T 数组的查询结果, INODE\_ARR\_T 数组中的信息是获取的某 INODE 页中所有 INODE 项的信息 (具体数值取决于实际状况)

## 13.4 描述页

描述页是用来显示簇信息的页。主要包括簇的上一个链接簇描述项的文件 ID、上一个链接簇描述项的页号、上一个链接簇描述项的偏移、下一个链接簇描述项的文件 ID、下一个链接簇描述项的页号、下一个簇链接描述项的偏移、簇使用情况、本簇所属段 ID 和页使用情况等信息。

### 13.4.1 数据类型

DBMS\_PAGE 包的描述页部分所用到的变量和记录类型, 如下统一说明:

包内变量和记录类型	解释
DESC_ITEM_T 类型	用于记录描述项信息。包括: 上一个链接簇描述项的文件 ID、 上一个链接簇描述项的页号、 上一个链接簇描述项的偏移、 下一个链接簇描述项的文件 ID、 下一个链接簇描述项的页号、 下一个簇链接描述项的偏移、 簇使用情况、 本簇所属段 ID、

	页使用情况
DESC_ITEM_ARR_T 类型	DESC_ITEM_T 类型数组

### 13.4.2 相关方法

DBMS\_PAGE 包中的描述页部分所包含的过程和函数详细介绍如下：

#### 1. SPAGE\_NTH\_DESC\_LOAD/SPAGE\_NTH\_DESC\_GET

获得描述页的第 NTH 个描述项信息。

语法如下：

```
PROCEDURE SPAGE_NTH_DESC_LOAD(
    PAGE IN VARBINARY,
    NTH IN SMALLINT,
    DESC_ITEM OUT DESC_ITEM_T
);
FUNCTION SPAGE_NTH_DESC_GET(
    PAGE VARBINARY,
    NTH SMALLINT
)RETURN DESC_ITEM_T;
```

#### 参数详解

- page

调用接口获取，要判定是否是描述页。

- nth

用户输入的第 nth 个描述项。

#### 2. SPAGE\_ALL\_DESC\_LOAD/SPAGE\_ALL\_DESC\_GET

获得描述页的所有描述项信息，一次最多可以查看 256 个描述页信息。

语法如下：

```
PROCEDURE SPAGE_ALL_DESC_LOAD(
    PAGE IN VARBINARY,
    DESC_ITEM OUT DESC_ITEM_ARR_T
);
FUNCTION XDESC_ALL_DESC_GET(
    PAGE VARBINARY
)RETURN DESC_ITEM_ARR_T;
```

#### 参数详解

- PAGE



调用接口获取，要判定是否是描述页。

### 13.4.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_BINARY');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_PAGE');
```

例 1 获取 0 号描述页的第 0 个描述项信息

```
DECLARE
    PAGE    VARBINARY;
    DESC_ITEM DBMS_PAGE.DESCR_ITEM_T;
    DESC_ITEM_ARR DBMS_PAGE.DESCR_ITEM_ARR_T;
BEGIN
    DBMS_PAGE.PAGE_LOAD(0, 0, 0, PAGE); --0,0,0 分别表示表空间号、文件号、页号
    DESC_ITEM_ARR = NEW DBMS_PAGE.DESCR_ITEM_T[1];
    DBMS_PAGE.SPACE_NTH_DESC_LOAD(PAGE,0,DESC_ITEM_ARR[1]);
    SELECT * FROM ARRAY DESC_ITEM_ARR;
END;
/
```

将打印如下信息(具体数值取决于实际状况):

```
PREV_FILE_ID PREV_PAGE_NO PREV_OFFSET NEXT_FILE_ID NEXT_PAGE_NO NEXT_OFFSET STATE
SEGID DES_BITMAP
-1 -1 -1 0 2048 148 1202 0 00000000000000111
```

例 2 获取 0 号描述页的第 0 到第 5 个描述项

```
DECLARE
    PAGE    VARBINARY;
    I        SMALLINT;
    DESC_ITEM_ARR DBMS_PAGE.DESCR_ITEM_ARR_T;
BEGIN
    DBMS_PAGE.PAGE_LOAD(0, 0, 0, PAGE);
    DESC_ITEM_ARR = NEW DBMS_PAGE.DESCR_ITEM_T[6];
    FOR I IN 1..6 LOOP
        DBMS_PAGE.SPACE_NTH_DESC_LOAD(PAGE,I-1,DESC_ITEM_ARR[I]);
    END LOOP;
    SELECT * FROM ARRAY DESC_ITEM_ARR;
END;
/
```

将打印如下信息(具体数值取决于实际状况):

```
PREV_FILE_ID PREV_PAGE_NO PREV_OFFSET NEXT_FILE_ID NEXT_PAGE_NO NEXT_OFFSET STATE
SEGID DES_BITMAP
-1 -1 -1 0 2048 148 1202 0 00000000000000111
-1 -1 -1 -1 -1 -1 1202 1 00111111111111111
-1 -1 -1 -1 -1 -1 1202 3 01111111111111111
```

```

0 2048 340 -1 -1 -1 1202 4 00101000000000001
-1 -1 -1 -1 -1 -1 1202 5 00111111111111111
-1 -1 -1 -1 -1 -1 1202 7 00111111111111111

```

例 3 获取 0 号页所有的描述项

```

DECLARE
    PAGE          VARBINARY;
    DESC_ITEM_ARR DBMS_PAGE.DESC_ITEM_ARR_T;
BEGIN
    DBMS_PAGE.PAGE_LOAD(0, 0, 0, PAGE);
    DBMS_PAGE.SPAGE_ALL_DESC_LOAD(PAGE, DESC_ITEM_ARR);
    SELECT * FROM ARRAY DESC_ITEM_ARR;
END;
/

```

将打印 0 号页所有的描述项的信息，记录格式同上图。

## 13.5 控制页

让用户更直观的查看数据库表空间的控制页的信息。

### 13.5.1 数据类型

DBMS\_PAGE 包的控制页部分所用到的变量和记录类型，如下统一说明：

包内变量和记录类型	解释
PH_T 类型	页头记录类型，包括：表空间 ID、本文件 ID、本文件的 PAGE NO、前一个文件的 FILE ID、前一个文件的 PAGE NO、下一个文件的 FILE ID、下一个文件的 PAGE NO、文件类型、CHECKSUM 检查点、LSN 该页实际刷盘的位置信息。当表空间 ID，文件 ID 均为 0，那 PAGE NO 从 0-6 均是控制页，分别为 0 号控制页，1 号 ID 页，2 号 SEQ 页，3 号提交事务的 LAST_LCN 页，4 号系统信息页，5 号复制关系已处理的 LCN 页，6 号保留页。PAGE NO 是从 0 开始，若 PAGE NO 为-1 代表该页为空。若 FILE ID 为-1 则代表该项所代表的 FILE 为空

P0_TSH_T 类型	0 号控制页的表空间头记录类型，包括：LSN、空闲的 INODE 页的数量、INODE FREE 链表中第一个文件号、INODE FREE 链表中第一个页号、INODE FREE 链表中第一个页偏移、INODE FREE 链表中最后一个文件号、INODE FREE 链表中最后一个页号、INODE FREE 链表中最后一个页偏移、全满的 INODE 页的数量、INODE FULL 链表中第一个文件号、INODE FULL 链表中第一个页号、INODE FULL 链表中第一个页偏移、INODE FULL 链表中最后一个文件号、INODE FULL 链表中最后一个页号、INODE FULL 链表中最后一个页偏移、空闲簇的数量、空闲簇链表中第一个文件号、空闲簇链表中第一个页号、空闲簇链表中第一个页偏移、空闲簇链表中最后一个文件号、空闲簇链表中最后一个页号、空闲簇链表中最后一个页偏移、半满簇的数量、半满簇链表中第一个页 ID、半满簇链表中第一个 PAGE NO、半满簇链表中第一个页的偏移、半满簇链表中最后一个页 ID、半满簇链表中最后一个 PAGE NO、半满簇链表中最后一个页的偏移、SYSINDEX 的根页的位置、SYSOBJ 的根页、页大小、一个簇里含有页个数、LOG 日志文件大小
P0_FH_T 类型	0 号控制页的文件头记录类型，包括：一个文件中页的数量、空闲页的页号
P1_INFO_T 类型	1 号控制页的描述项记录类型，包括：下一个段 ID、下一个表 ID、下一个视图 ID、下一个用户 ID、下一个索引 ID、下一个存储过程 ID、下一个函数 ID、下一个角色 ID、下一个文件 ID、下一个触发器 ID、下一个约束 ID、下一个模式 ID、下一个序列 ID、下一个 POLICY ID、下一个 DBLINK ID、下一个事务 ID、下一个跟踪 ID、下一个事件 ID、下一个 OPERATOR ID、下一个警告 ID、下一个作业 ID、下一个作业步骤 ID、下一个作业调度 ID、下一个异步复制 ID、下一个包 ID、下一个作业步骤 ID、下一个密码引擎 ID、下一个 OBJECTTYPE ID、下一个 SYNONY ID、下一个 BLOB ID、下一个 MAL ID、下一个 TAG ID、下一个 GROUP ID、下一个 INST ID、下一个 REP ID、下一个未被使用的组 ID、下一个 NEW TRX ID
SEQ_ITEM_T 类型	2 号控制页的 SEQ 记录类型，包括：该页的使用情况、SEQ 的值
CPAGE_SEQ_HEAD_T 类型	2 号控制页的页头信息类型，包括：申请下一页所在的表空间 ID、申请下一页所在的文件 ID、申请下一页的 PAGENO、页中已使用的 SEQ 项的数量
CPAGE_SEQ_INFO_T 类型	每个页中使用多少个 SEQ 项的数量类型，包括：该页所在的表空间 ID、该页所在的文件 ID、该页的页号、该页中已使用的 SEQ 项的数量

P3_HEAD_T 类型	3 号控制页页头记录类型，包括：LAST_LCN 页的表空间 ID、LAST_LCN 页的文件 ID、LAST_LCN 页的文件号、LAST LCN 的数量
P3_ITEM_T 类型	3 号控制页的描述项记录类型，包括：该页的使用情况、前一个 LCN 的 ID、前一个 TRX 的 ID、前一个 LCN
P4_DES_T 类型	4 号控制页的描述项记录类型，包括：版本号、大小写敏感、编码方式、空串是否等同于 NULL、SITE MAGIC 站点魔数、PRIKEY 解密服务主密钥的 RSA 私钥文件路径、SVR KEY 服务器主密钥、SYSTEM DB KEY 系统数据库的 KEY、数据库模式、LOG REP TYPE 复制实例类型日志、COORDINATOR ADDR 协调器地址、COORDINATOR PORT 协调器端口、时区
P5_HEAD_T 类型	5 号控制页页头记录类型，包括：LOG FILE1 的长度以 KB 为单位、LOG FILE2 的长度以 KB 为单位、LOG WRITE FILE 的 SEQ、LOG READ FILE 的 SEQ、LOG READ FILE 的 OFFSET、复制关系已处理的 LAST LCN 的数量
P5_ITEM_T 类型	5 号控制页的描述项记录类型，包括：该项是否已被使用、REP ID2、上一个 LCN 项
PH_ARR_T 类型	PH_T 类型的数组
P0_FH_ARR_T 类型	P0_FH_T 类型的数组
P1_INFO_ARR_T 类型	P1_INFO_T 类型的数组
CPAGE_SEQ_HEAD_ARR_T 类型	CPAGE_SEQ_HEAD_T 类型的数组
CPAGE_SEQ_INFO_ARR_T 类型	CPAGE_SEQ_INFO_T 类型的数组
SEQ_ITEM_ARR_T 类型	SEQ_ITEM_T 类型的数组
P3_HEAD_ARR_T 类型	P3_HEAD_T 类型的数组
LAST_LCN_PAGE_HEAD_ARR_T 类型	P3_HEAD_T 类型的数组
P3_ITEM_ARR_T 类型	P3_ITEM_T 类型的数组
P4_DES_ARR_T 类型	P4_DES_T 类型的数组
P5_HEAD_ARR_T 类型	P5_HEAD_T 类型的数组
P5_ITEM_ARR_T 类型	P5_ITEM_T 类型的数组

### 13.5.2 相关方法

DBMS\_PAGE 包中的控制页部分所包含的过程和函数详细介绍如下：

#### 1. PAGE\_LOAD

加载数据页。

语法如下：

```
PROCEDURE PAGE_LOAD(
    TS_ID    IN INT,
    FI_ID    IN INT,
```

---

```
PAGE_NO IN INT,
PAGE OUT VARBINARY
);
```

---

### 参数详解

- TS\_ID  
表空间 ID。
- FILE\_ID  
文件号。
- PAGE\_NO  
页号。
- PAGE  
页内容。

### 2. PAGE\_HEAD\_INFO

获得通用页头信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE PAGE_HEAD_LOAD (
    PAGE IN VARBINARY,
    PH_INFO OUT PH_T
);
FUNCTION PAGE_HEAD_GET (
    PAGE IN VARBINARY
) RETURN PH_T;
```

---

### 参数详解

- PAGE  
页内容。

### 3. CTL\_PAGE0\_TS\_HEAD\_LOAD/ CTL\_PAGE0\_TS\_HEAD\_GET

获得 0 号页表空间头的信息。

语法如下：

---

```
PROCEDURE CTL_PAGE0_TS_HEAD_LOAD (
    GHEAD OUT P0_TSH_T
);
FUNCTION CTL_PAGE0_TS_HEAD_GET RETURN P0_TSH_T;
```

---

### 4. CTL\_PAGE0\_FH\_LOAD/ CTL\_PAGE0\_FH\_GET

获得 0 号页文件头的信息。

语法如下：

---

```
PROCEDURE CTL_PAGE0_FH_LOAD (
    FHEAD OUT P0_FH_T
```

---

---

);

---

FUNCTION CTL\_PAGE0\_FH\_GETRETURN P0\_FH\_T;

---

#### 5. CTL\_PAGE1\_DES\_LOAD/ CTL\_PAGE1\_DES\_GET

获得 1 号控制页的描述信息。

语法如下：

---

PROCEDURE CTL\_PAGE1\_DES\_LOAD (
 P1DES OUT P1\_INFO\_T
);

---

FUNCTION CTL\_PAGE1\_DES\_GETRETURN P1\_INFO\_T;

---

#### 6. CPAGE\_SEQ\_HEAD\_LOAD/ CPAGE\_SEQ\_HEAD\_GET

获得 SEQ 页的 SEQ 页头信息。

语法如下：

---

PROCEDURE CPAGE\_SEQ\_HEAD\_LOAD (
 PAGE2 IN VARBINARY,
 P2DES OUT CPAGE\_SEQ\_HEAD\_T
);

---

FUNCTION CPAGE\_SEQ\_HEAD\_GET (
 PAGE2 IN VARBINARY
) RETURN CPAGE\_SEQ\_HEAD\_T;

---

#### 参数详解

- PAGE2

SEQ 页内容。

#### 7. SEQ\_ITEM\_ALL\_LOAD/ SEQ\_ITEM\_ALL\_GET

获得该页的所有 SEQ 项的信息。

语法如下：

---

PROCEDURE SEQ\_ITEM\_ALL\_LOAD (
 PAGE2 IN VARBINARY,
 P2DES OUT SEQ\_ITEM\_ARR\_T
);

---

FUNCTION SEQ\_ITEM\_ALL\_GET (
 PAGE2 IN VARBINARY
) RETURN SEQ\_ITEM\_ARR\_T;

---

#### 参数详解

- PAGE2

SEQ 页内容。

#### 8. CTL\_PAGE\_SEQ\_COUNT\_N\_LOAD/ CTL\_PAGE\_SEQ\_COUNT\_N\_GET

获得系统中从 OFFSET\_N 开始往后 N 页的 SEQ 页信息并返回（最多 100 页）。

语法如下：

---

PROCEDURE CTL\_PAGE\_SEQ\_COUNT\_N\_LOAD (

---

---

```

    OFFSET_N    IN    SMALLINT DEFAULT 0,
    N           IN    SMALLINT DEFAULT 100,
    SEQCOUNT   OUT   CPAGE_SEQ_INFO_ARR_T
);
FUNCTION CTL_PAGE_SEQ_COUNT_N_GET (
    OFFSET_N    IN SMALLINT DEFAULT 0,
    N           IN SMALLINT DEFAULT 100
) RETURN CPAGE_SEQ_INFO_ARR_T;

```

---

### 参数详解

- OFFSET\_N

页偏移量。

- N

从 OFFSET\_N 开始往后 N 页。

### 9. CPAGE3\_HEAD\_GET/CPAGE3\_HEAD\_LOAD

获得 3 号控制页的页头格式信息并返回。

语法如下：

---

```

PROCEDURE CPAGE3_HEAD_LOAD(
    P3_HEAD OUT P3_HEAD_T
);
FUNCTION  CPAGE3_HEAD_GET RETURN P3_HEAD_T;

```

---

### 10. LAST\_LCN\_PAGE\_HEAD\_LOAD /LAST\_LCN\_PAGE\_HEAD\_GET

获得系统中从 OFFSET\_N 开始往后 N 页的 SEQ 页信息（最多 100 页）。

语法如下：

---

```

FUNCTION LAST_LCN_PAGE_HEAD_GET (
    OFFSET INT DEFAULT 0,
    NUM     SMALLINT DEFAULT 100
) RETURN LAST_LCN_PAGE_HEAD_ARR_T;
PROCEDURE LAST_LCN_PAGE_HEAD_LOAD (
    OFFSET          INT DEFAULT 0,
    NUM             INT DEFAULT 100,
    P_HEAD_ARR OUT  LAST_LCN_PAGE_HEAD_ARR_T
);

```

---

### 参数详解

- OFFSET

页偏移量。

- NUM

从 OFFSET\_N 开始往后 N 页。

### 11. LAST\_LCN\_PAGE\_ITEM\_LOAD /LAST\_LCN\_PAGE\_ITEM\_GET

获得相应页的 ITEM 信息。

语法如下：

---

```

FUNCTION LAST_LCN_PAGE_ITEM_GET (
    TS_ID SMALLINT,
    FILE_ID SMALLINT,
    PAGE_NO INT
) RETURN P3_ITEM_ARR_T;
PROCEDURE LAST_LCN_PAGE_ITEM_LOAD (
    TS_ID SMALLINT,
    FILE_ID SMALLINT,
    PAGE_NO INT,
    P3_ITEM_ARR OUT P3_ITEM_ARR_T
);

```

---

### 参数详解

- TS\_ID  
表空间号。
- FILE\_ID  
文件号。
- PAGE\_NO  
页号。

#### 12.CPAGE3\_LCN\_MAX\_CNT\_GET

返回一个 LAST LCN 页能存的最大 LAST LCN ITEM 项的个数。

语法如下：

---

```

FUNCTION CPAGE3_LCN_MAX_CNT_GET RETURN INT;

```

---

#### 13.CPAGE4\_DES\_LOAD/CPAGE4\_DES\_GET

获得 4 号控制页的描述信息。

语法如下：

---

```

PROCEDURE CPAGE4_DES_LOAD(
    P4_INFO OUT P4_DES_T
);

```

---

```

FUNCTION CPAGE4_DES_GET RETURN P4_DES_T;

```

---

#### 14.CPAGE5\_HEAD\_LOAD/CPAGE5\_HEAD\_GET

获得 5 号控制页的页头格式信息。

语法如下：

---

```

PROCEDURE CPAGE5_HEAD_LOAD(
    P5_HEAD OUT P5_HEAD_T
);

```

---

```

FUNCTION CPAGE5_HEAD_GET RETURN P5_HEAD_T;

```

---

#### 15.CPAGE5\_ITEM\_LOAD/ CPAGE5\_ITEM\_GET

获得 5 号控制页的描述信息。



语法如下：

```
PROCEDURE CPAGE5_ITEM_LOAD (
    P5DES OUT P5_ITEM_ARR_T
);
FUNCTION      CPAGE5_ITEM_GETRETURN P5_ITEM_ARR_T;
```

### 13.5.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_BINARY');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_PAGE');
```

例取得通用页头信息 0-3

```
DECLARE
PAGE0      VARBINARY;
PAGE1      VARBINARY;
PHINFO     DBMS_PAGE.PH_T;
PHINFO_ARR DBMS_PAGE.PH_ARR_T;
BEGIN

PHINFO_ARR = NEW DBMS_PAGE.PH_T[4];
FOR I IN 1..4 LOOP
    DBMS_PAGE.PAGE_LOAD(0,0,I-1,PAGE0);
    --DBMS_PAGE.PAGE_HEAD_LOAD(PAGE0,PHINFO);
    PHINFO_ARR[I] = DBMS_PAGE.PAGE_HEAD_GET(PAGE0);
END LOOP;
SELECT * FROM ARRAY PHINFO_ARR;
END;
/
```

结果如下：

```
TS_ID SELF_FILE_ID SELF_PAGE_NO PREV_FILE_ID PREV_PAGE_NO NEXT_FILE_ID
NEXT_PAGE_NO PAGE_TYPE CHECKSUM LSN
0 0 0 -1 -1 -1 -1 19 0 13971
0 0 1 -1 -1 -1 -1 -1 0 14681
0 0 2 -1 -1 -1 -1 -1 0 1
0 0 3 -1 -1 -1 -1 99 0 1
```

## 14 DBMS\_PIPE 包

DBMS\_PIPE 包用于同一实例的不同会话之间通过管道进行通信。这里的管道(PIPE)类似于 UNIX 系统的管道，但它不是采用操作系统机制实现的。管道信息被存储在本地消息缓冲区中，当关闭实例时会丢失管道信息。

管道的发送端和接收端都有相应的本地消息缓冲区进行接收和发送消息。DBMS\_PIPE 包中函数的使用流程大致为：首先，发送端创建(CREATE\_PIPE)管道、打包(PACK\_MESSAGE)管道消息、发送(SEND\_MESSAGE)消息；其次，接收端接收(RECEIVE\_MESSAGE)消息、取出(UNPACK\_MESSAGE)消息。

### 14.1 相关方法

#### 1. CREATE\_PIPE

显式创建一个公用或者私有管道。

语法如下：

---

```
DBMS_PIPE.CREATE_PIPE(
    PIPENAME IN VARCHAR2,
    MAXPIPESIZE IN INTEGER DEFAULT 8192,
    IS_PRIVATE IN BOOLEAN DEFAULT TRUE
)RETURN INTEGER;
```

---

#### 参数详解

- PIPENAME 指定管道的名称。
- MAXPIPESIZE 指定管道消息的最大尺寸。取值范围：8K~2G。
- IS\_PRIVATE  
指定管道类型。TRUE 表示建立私有管道，FALSE 建立公有管道。公用管道是指所有数据库用户都可以访问的管道，而私有管道只有建立管道的数据库用户才能访问。

#### 返回值

如果该函数返回 0，则表示建立管道成功，否则表示建立管道失败。

#### 使用说明：

通过 CREATE\_PIPE 创建的管道的方法为显式创建；在向指定的管道发送消息的过程中，如果指定的管道不存在，则系统会隐式创建管道，这种方法称为隐式创建。

#### 2. NEXT\_ITEM\_TYPE

获取本地消息缓冲区中下一条消息的数据类型，在调用了 RECEIVE\_MESSAGE 之后调

用该函数。

语法如下：

---

```
DBMS_PIPE.NEXT_ITEM_TYPE RETURN INTEGER;
```

---

### 返回值

返回 0，则表示管道没有任何消息。

返回 6，则表示下一项的数据类型为 NUMBER。

返回 9，则表示下一项的数据类型为 VARCHAR2。

返回 12，则表示下一项的数据类型为 DATE。

返回 23，则表示下一项的数据类型为 BINARY。

### 3. REMOVE\_PIPE

删除指定的管道。

语法如下：

---

```
DBMS_PIPE.REMOVE_PIPE (
    PIPENAME IN VARCHAR2
)RETURN INTEGER;
```

---

### 参数详解

- PIPENAME 管道名称。

### 使用说明：

1) 对于隐式创建的管道来说，管道为空时自动被删除，不需要调用该函数。

2) 对于显式创建的管道来说，只能通过调用 REMOVE\_PIPE 或者关闭数据库实例来删除管道。

### 返回值

0：删除成功

其他错误码：删除失败

### 4. RESET\_BUFFER

清空本地发送消息缓冲区。因为一个会话中所有的管道都共享一个本地管道缓冲区，所以为了防止将本地缓存中原有的数据发送到管道中，在发送消息之前应该复位管道缓冲区。

语法如下：

---

```
DBMS_PIPE.RESET_BUFFER;
```

---

### 5. PURGE

清空指定管道中的消息。

语法如下：

---

```
DBMS_PIPE.PURGE (  
    PIPENAME IN VARCHAR2  
) ;
```

---

#### 参数详解

- PIPENAME 管道名称。

### 6. PACK\_MESSAGE

在本地消息缓冲区中指定消息打包。

为了给管道发送信息，首先需要使用过程 PACK\_MESSAGE 将本地消息缓冲区中的指定消息打包好，然后使用过程 SEND\_MESSAGE 将打包好的消息发送到管道。

语法如下：

---

```
DBMS_PIPE.PACK_MESSAGE (ITEM IN VARCHAR) ;  
DBMS_PIPE.PACK_MESSAGE (ITEM IN INT) ;  
DBMS_PIPE.PACK_MESSAGE (ITEM IN BIGINT) ;  
DBMS_PIPE.PACK_MESSAGE (ITEM IN DOUBLE) ;  
DBMS_PIPE.PACK_MESSAGE (ITEM IN NUMBER) ;  
DBMS_PIPE.PACK_MESSAGE (ITEM IN DATE) ;  
DBMS_PIPE.PACK_MESSAGE_ROWID (ITEM IN BIGINT) ;
```

---

#### 参数详解

- ITEM 指定管道消息。其输入值可以是字符，数字，日期等多种数据类型。

### 7. RECEIVE\_MESSAGE

接收指定管道中的消息到本地消息缓冲区。

语法如下：

---

```
DBMS_PIPE.RECEIVE_MESSAGE (  
    PIPENAME IN VARCHAR2,  
    TIMEOUT IN INTEGER DEFAULT MAXWAIT  
) RETURN INTEGER ;
```

---

#### 参数详解

- PIPENAME 管道名称。
- TIMEOUT 最大等待时间。取值范围：0~86400000 单位：秒。

#### 返回值

返回 0，表示接收消息成功。

返回 1，则表示出现超时。

返回 2，则表示本地缓冲区不能容纳管道消息。

返回 3，则表示发生中断。

**使用说明：**

- 1) 接收之后，消息从管道删除。
- 2) 一条消息只能被接收一次。
- 3) 对于隐式创建的管道来说，管道中最后一条消息被删除后，该管道被自动删除。

## 8. SEND\_MESSAGE

向指定管道上发送一条消息。

语法如下：

---

```
DBMS_PIPE.SEND_MESSAGE (
    PIPENAME      IN VARCHAR2,
    TIMEOUT        IN INTEGER DEFAULT MAXWAIT,
    MAXPIPESIZE    IN INTEGER DEFAULT 8192
) RETURN INTEGER;
```

---

**参数详解**

- PIPENAME 管道名称。
- TIMEOUT 最大等待时间。取值范围：0~86400000 单位：秒。
- MAXPIPESIZE 指定管道消息的最大尺寸。取值范围：8K~2G。

**使用说明：**

- 1) 消息是通过调用 PACK\_MESSAGE 生成并存储在本地缓冲区内。
- 2) 如果指定的管道不存在，则自动隐式创建。

**返回值**

返回 0，表示接收消息成功。

返回 1，表示出现超时。

返回 2，则表示本地缓冲区不能容纳管道消息。

返回 3，则表示发生中断。

## 9. UNIQUE\_SESSION\_NAME

返回一个会话的唯一标示名称。对于同一会话来说，其值不会改变。

语法如下：

---

```
DBMS_PIPE.UNIQUE_SESSION_NAME
RETURN VARCHAR2;
```

---

## 10. UNPACK\_MESSAGE

用于取出消息缓冲区中的内容。在使用函数 RECEIVE\_MESSAGE 将管道消息接收到本地消息缓冲区中之后，应该使用过程 UNPACK\_MESSAGE 取得本地消息缓冲区的消息。

当使用过程 UNPACK\_MESSAGE 取出消息缓冲区的消息时，每次只能取出一条消息。如果要取出多条消息，则需要多次调用过程 UNPACK\_MESSAGE。

语法如下：

---

```
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT VARCHAR2);
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT INT);
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT BIGINT);
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT DOUBLE);
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT NUMBER);
DBMS_PIPE.UNPACK_MESSAGE (ITEM OUT DATE);
DBMS_PIPE.UNPACK_MESSAGE_ROWID (ITEM OUT BIGINT);
```

---

## 参数详解

- ITEM 指定管道消息，其输入值可以是字符，数字，日期等多种数据类型。

## 14.2 动态视图

## V\$DB\_PIPES

记录管道的相关信息。

列名	类型	说明
NAME	VARCHAR(30) NOT NULL	管道的名字
OWNERID	BIGINT NOT NULL	管道拥有者 ID
TYPE	VARCHAR(7)	管道类型:PUBLIC/PRIVATE
PIPE_SIZE	BIGINT	管道的长度

## 14.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_PIPE');
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_OUTPUT');
SET SERVEROUTPUT ON; --dbms_output.put_line 需要设置这条语句，才能打印出消息
```

举一个两个会话之间通信的实例。

第一步，第一个会话发送消息。该会话连接的用户名和密码为 SYSDBA/SYSDBA。

```
declare
v_pipeName varchar2(30):='pipe2';
```

```

v_status integer;
begin
DBMS_PIPE.PURGE(v_pipeName);
DBMS_PIPE.RESET_BUFFER;
v_status:=DBMS_PIPE.create_pipe(v_pipeName,8192,FALSE);
if v_status !=0 then
raise_application_error(-20099, 'create_pipe error status = '|| v_status);
end if;
dbms_pipe.pack_message(' hello,this is sending process!');
v_status:=dbms_pipe.send_message(v_pipeName);
if v_status !=0 then
dbms_output.put_line('error!');
end if;
end;
/

```

第二步，第二个会话接收消息。该会话连接的用户名和密码为 USER01/USER012345。

```

declare
v_pipeName varchar2(30):='pipe2';
v_status integer;
v_msg varchar2(20);
begin
v_status:=dbms_pipe.receive_message(v_pipeName);
if v_status !=0 then
dbms_output.put_line('error');
end if;
dbms_pipe.unpack_message(v_msg);
print v_msg;
end;
/

```

结果打印出: hello,this is sending process!

## 15 DBMS\_RANDOM 包

为提高 ORACLE 向 DM 移植的兼容性。实现随机产生 INT 类型、NUMBER 类型数，随机字符串，以及符合正态分布的随机数。支持 ORACLE 的 DBMS\_RANDOM 系统包。

### 15.1 相关方法

#### 1. INITIALIZE

初始化随机种子。接收 INT 类型参数，将 RAND() 种子设为输入的参数。

语法如下：

---

```
PROCEDURE INITIALIZE(
    VAL IN INT
);
```

---

## 2. SEED

重置随机种子。接收 INT 类型参数，将 RAND( ) 种子设为输入的参数。

语法如下：

---

```
PROCEDURE SEED(
    VAL IN INT
);
```

---

重置随机种子。接收 VARCHAR 类型参数，通过转换成 INT 类型再将 RAND( ) 种子设为输入的参数转换好的值。

语法如下：

---

```
PROCEDURE SEED(
    VAL IN VARCHAR2
);
```

---

## 3. TERMINATE

ORACLE 中不开放的功能，在此只做语法支持，无意义。

语法如下：

---

```
PROCEDURE TERMINATE;
```

---

## 4. RANDOM\_NORMAL/ RANDOM/ RANDOM\_STRING 或 STRING

产生符合正态分布的随机数。语法如下：

---

```
FUNCTION RANDOM_NORMAL RETURN NUMBER;
```

---

产生 INT 类型随机数。语法如下：

---

```
FUNCTION RANDOM RETURN INTEGER;
```

---

生成随机字符串。语法如下：

---

```
FUNCTION RANDOM_STRING (
    OPT IN CHAR,
    LEN IN NUMBER
) RETURN VARCHAR2;
```

---

```
FUNCTION STRING (
    OPT IN CHAR,
    LEN IN NUMBER
) RETURN VARCHAR2;
```

---

## 参数详解

### ● OPT

CHAR 类型，表示字符串模式，模式解释如下：

'u', 'U': 只产生随机的大写字母字符串；



- 'l', 'L': 只产生随机的小写字母字符串;
- 'a', 'A': 产生大小写混合的字母字符串;
- 'x', 'X': 返回大写字母和数字随机字符串;
- 'p', 'P': 返回随机可打印字符串。

当模式字符不是上述给定的字符时，系统会为按'U'的方式处理。

- LEN 表示生成字符串的长度，最大为 32767。

## 5. VALUE

生成大于 LOW，小于 HIGH 的 NUMBER 行随机数。

语法如下：

---

```
FUNCTION VALUE (
    LOW IN NUMBER DEFAULT 0,
    HIGH IN NUMBER DEFAULT 1
)RETURN NUMBER;
```

---

### 参数详解

- LOW、HIGH

都为 NUMBER 类型，LOW 小于 HIGH 时，产生大于等于 LOW 小于等于 HIGH 的随机数；

当 LOW 大于 HIGH 时，大于 HIGH 小于等于 LOW 的随机数。

## 15.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_RANDOM');
```

例 1 初始化随机种子。

```
CALL DBMS_RANDOM.INITIALIZE(15);
```

例 2 返回大于等于 10 小于 100 的 NUMBER 类型随机数。

```
SELECT DBMS_RANDOM.VALUE(10,100);
```

结果如下：

```
10.112613
```

## 16 DBMS\_RLS 包

在某些系统中，权限控制必须定义到数据行访问权限的控制，此需求一般出现在同一系统中，不同的相对独立机构使用的情况。（如：集团下属多个子公司，所有子公司使用同一套数据表，但不同子公司的数据相对隔离），绝大多数人会选择在表或视图加上 WHERE 子句来进行数据隔离。此方法编码工作量大、系统适应用户管理体系的弹性空间较小，一旦权限逻辑发生变动，就可能需要修改权限体系，导致所有的表或视图都必须修改。

为了解决上述问题，达梦数据库管理系统提供了 DBMS\_RLS 包，DBMS\_RLS 包通过策略 (POLICY) 管理方法来实现数据行的隔离。

用户只要提前调用系统过程 SP\_CREATE\_SYSTEM\_PACKAGES(1)，创建策略管理所需要的系统包 DBMS\_RLS，就可以使用 DBMS\_RLS 包中的函数或过程来实现数据访问控制。

### 16.1 策略函数

策略函数的模板如下所示：

---

```
FUNCTION POLICY_FUNCTION (
    OBJECT_SCHEMA IN VARCHAR2,  -- 模式名。如果为空则表示当前模式。
    OBJECT_NAME VARCHAR2       -- 对象名。仅支持表或视图
) RETURN VARCHAR2
```

---

注：如果不是策略函数模板形式的函数，执行时会报错“无法执行策略函数”。

策略函数中两个输入参数，一个是用户输入参数，一个是对象输入参数。两个输入参数不能不写，尽管可以在函数中没有用到，否则报错返回。

### 16.2 策略组

#### 16.2.1 创建策略组

创建一个新的策略组。

语法如下：

---

```
PROCEDURE CREATE_POLICY_GROUP (
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    POLICY_GROUP  IN VARCHAR
);
```

---

#### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。

- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_GROUP 添加到该对象的策略组名称。

**使用说明：**

1. 如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。
2. 对于受保护的每个对象（表或视图），策略组必须唯一。

## 16.2.2 删除策略组

删除一个策略组。

语法如下：

---

```
PROCEDURE DELETE_POLICY_GROUP(
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    POLICY_GROUP  IN VARCHAR
);
```

---

**参数详解**

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_GROUP 要删除该对象的策略组名称。

**使用说明：**

如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。

## 16.3 策略

### 16.3.1 创建策略

创建一个新的策略。

语法如下：

---

```
PROCEDURE ADD_POLICY(
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    POLICY_NAME   IN VARCHAR,
    FUNCTION_SCHEMA IN VARCHAR := NULL,
    POLICY_FUNCTION IN VARCHAR,
    STATEMENT_TYPES IN VARCHAR := NULL,
    UPDATE_CHECK   IN INT := 0,
    ENABLE         IN INT := 1,
```

---

---

```

STATIC_POLICY    IN INT    := 0,
POLICY_TYPE      IN INT    := NULL,
LONG_PREDICATE   INT      := 0,
SEC_RELEVANT_COLS IN VARCHAR := NULL,
SEC_RELEVANT_COLS_OPT IN INT := NULL
);

```

---

### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_NAME 添加到该对象的策略名称。
- FUNCTION\_SCHEMA 拥有策略函数的模式。
- POLICY\_FUNCTION 策略函数名称。
- STATEMENT\_TYPES 应用策略的 DML 语句类型。
- UPDATE\_CHECK 该参数对于 INSERT 或 UPDATE 类型是可选项，默认值为 0。
- ENABLE 表示添加该策略时是否启用它，默认值为 1。
- STATIC\_POLICY  
表策略是静态的，即对任何访问该对象的人产生相同的谓词字符串，该参数默认值为 0。
- POLICY\_TYPE  
表示策略静态或动态类型，默认为 NULL。如果 POLICY\_TYPE 不是 NULL，则覆盖 STATIC\_POLICY。可允许的值是 STATIC、SHARED\_STATIC、CONTEXT\_SENSITIVE、SHARED\_CONTEXT\_SENSITIVE 和 DYNAMIC。
- LONG\_PREDICATE 表示谓词字符串是否是长谓词，该参数默认为 0。
- SEC\_RELEVANT\_COLS  
只应用于表和视图，在列表中指定受保护的列，使用逗号或空格作为分隔符。
- SEC\_RELEVANT\_COLS\_OPT  
该参数的默认值为 NULL；如果不是默认值，则必须指定 DBMS\_RLS.ALL\_ROWS，用于显示敏感列为 NULL 的所有列。

### 使用说明：

1. 如果 OBJECT\_SCHEMA 和 FUNCTION\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。
2. 对于受保护的每个对象（表或视图），策略名必须唯一。
3. POLICY\_FUNCTION 为针对 OBJECT\_NAME 的策略生成谓词。如果函数是程序包的一部分，则 POLICY\_FUNCTION 必须指定程序包名，用于限定策略函数名。
4. STATEMENT\_TYPES 允许的值以逗号或空格分隔，可以是 SELECT、INSERT、UPDATE、DELETE 和 INDEX 的任意组合。默认情况下，除了 INDEX 之外的所有类型都适用。

5. 如果 UPDATE\_CHECK 参数为 1, 则会检查 INSERT 或 UPDATE 对应的数据是否满足策略函数。不满足, 则报错“策略违反检验选项”。

6. 不管 LONG\_PREDICATE 是 0, 还是 1, 谓词字符串最多可为 4000 字节。

7. STATIC\_POLICY 和 POLICY\_TYPE 暂不支持, 所有谓词都是动态的。

8. SEC\_RELEVANT\_COLS 和 SEC\_RELEVANT\_COLS\_OPT 暂不支持。

9. ADD\_POLICY 没有指定策略组, 则使用系统默认的策略组 SYS\_DEFAULT。

创建一个新的策略。用法与 ADD\_POLICY 基本相同, 只是指定了策略所属的策略组 POLICY\_GROUP, POLICY\_GROUP 默认值为系统策略组 SYS\_DEFAULT。

语法如下:

---

```
PROCEDURE ADD_GROUPED_POLICY(
    OBJECT_SCHEMA    IN VARCHAR := NULL,
    OBJECT_NAME      IN VARCHAR,
    POLICY_GROUP     IN VARCHAR := 'SYS_DEFAULT',
    POLICY_NAME      IN VARCHAR,
    FUNCTION_SCHEMA  IN VARCHAR := NULL,
    POLICY_FUNCTION  IN VARCHAR,
    STATEMENT_TYPES  IN VARCHAR := NULL,
    UPDATE_CHECK     IN INT := 0,
    ENABLE           IN INT := 1,
    STATIC_POLICY    IN INT := 0,
    POLICY_TYPE      IN INT := NULL,
    LONG_PREDICATE   INT := 0,
    SEC_RELEVANT_COLS IN VARCHAR := NULL,
    SEC_RELEVANT_COLS_OPT IN INT := NULL
);
```

---

### 16.3.2 启用策略

启用一个策略。

语法如下:

---

```
PROCEDURE ENABLE_POLICY(
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    POLICY_NAME   IN VARCHAR,
    ENABLE        IN INT := 1
);
```

---

#### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。

- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_NAME 启用该对象的策略名称。
- ENABLE 表示是否启用该策略，默认值为 1。0 表示禁用，1 表示启用。

**使用说明：**

1. 如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。
2. ENABLE\_POLICY 没有指定策略组，则使用系统默认的策略组 SYS\_DEFAULT。

启用一个策略。用法与 ENABLE\_POLICY 基本相同，只是指定了策略所属的策略组 GROUP\_NAME。

**语法如下：**


---

```
PROCEDURE ENABLE_GROUPED_POLICY(
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    GROUP_NAME    IN VARCHAR,
    POLICY_NAME   IN VARCHAR,
    ENABLE        IN INT := 1
);
```

---

### 16.3.3 禁用策略

禁用一个策略。

**语法如下：**


---

```
PROCEDURE DISABLE_GROUPED_POLICY(
    OBJECT_SCHEMA IN VARCHAR := NULL,
    OBJECT_NAME   IN VARCHAR,
    GROUP_NAME    IN VARCHAR,
    POLICY_NAME   IN VARCHAR
);
```

---

**参数**

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_NAME 禁用该对象的策略名称。

**使用说明：**

1. 如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。

### 16.3.4 删除策略

删除一个策略。

语法如下：

---

```
PROCEDURE DROP_POLICY(  
    OBJECT_SCHEMA IN VARCHAR := NULL,  
    OBJECT_NAME   IN VARCHAR,  
    POLICY_NAME   IN VARCHAR  
);
```

---

#### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- POLICY\_NAME 要删除该对象的策略名称。

#### 使用说明：

1. 如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户的当前模式。
2. DROP\_POLICY 没有指定策略组，则使用系统默认的策略组 SYS\_DEFAULT。

删除一个策略。用法与 DROP\_POLICY 基本相同，只是指定了策略所属的策略组 POLICY\_GROUP，POLICY\_GROUP 默认值为系统策略组 SYS\_DEFAULT。

语法如下：

---

```
PROCEDURE DROP_GROUPED_POLICY(  
    OBJECT_SCHEMA IN VARCHAR := NULL,  
    OBJECT_NAME   IN VARCHAR,  
    POLICY_GROUP  IN VARCHAR := 'SYS_DEFAULT',  
    POLICY_NAME   IN VARCHAR  
);
```

---

## 16.4 上下文

### 16.4.1 创建上下文策略

创建一个上下文策略。

语法如下：

---

```
PROCEDURE ADD_POLICY_CONTEXT(  
    OBJECT_SCHEMA IN VARCHAR := NULL,  
    OBJECT_NAME   IN VARCHAR,  
    NAMESPACE    IN VARCHAR,
```

---

---

```

    ATTRIBUTE      IN VARCHAR
);

```

---

#### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- NAMESPACE 添加到该对象的上下文名称。
- ATTRIBUTE 策略对应上下文的属性名称。

#### 使用说明：

1. 如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。
2. 对于受保护的每个对象（表或视图），同一上下文的属性名必须唯一。

### 16.4.2 删除上下文策略

删除一个上下文策略。

语法如下：

---

```

PROCEDURE DROP_POLICY_CONTEXT(
    OBJECT_SCHEMA  IN VARCHAR := NULL,
    OBJECT_NAME    IN VARCHAR,
    NAMESPACE     IN VARCHAR,
    ATTRIBUTE      IN VARCHAR
);

```

---

#### 参数详解

- OBJECT\_SCHEMA 包含由策略保护的表或视图的模式。
- OBJECT\_NAME 由策略保护的表或视图的名称。
- NAMESPACE 添加到该对象的上下文名称。
- ATTRIBUTE 策略对应上下文的属性名称。

#### 使用说明：

如果 OBJECT\_SCHEMA 为 NULL，则使用调用过程的用户当前模式。

## 16.5 举例说明

### 例 1 通过策略过滤用户对表数据的访问控制

数据准备：

```

CONN SYSDBA/SYSDBA

SP_CREATE_SYSTEM_PACKAGES(1);

```



```
CREATE USER USER01 IDENTIFIED BY USER012345;

GRANT DBA TO USER01;

CONN USER01/USER012345
```

第一步：建立测试数据表(T\_POLICY)：

```
CREATE TABLE T_POLICY( T1 VARCHAR2(10), T2 NUMBER(10));

INSERT INTO T_POLICY VALUES('A',10);

INSERT INTO T_POLICY VALUES('B',20);

INSERT INTO T_POLICY VALUES('C',30);

COMMIT;
```

第二步：建立测试 POLICY 的函数：

```
CREATE OR REPLACE FUNCTION FN_GETPOLICY(

    P_SCHEMA IN VARCHAR2,

    P_OBJECT IN VARCHAR2

) RETURN VARCHAR2 IS

    RESULT VARCHAR2(1000);

BEGIN

    RESULT:='T2 NOT IN (10)';

    RETURN(RESULT);

END;

/
```

第三步：加入 POLICY：

```
DBMS_RLS.ADD_POLICY(

    OBJECT_SCHEMA =>'USER01', --数据表(或视图)所在的 SCHEMA 名称

    OBJECT_NAME =>'T_POLICY', --数据表(或视图)的名称

    POLICY_NAME =>'T_TESTPOLICY', --POLICY 的名称，主要用于将来对 POLICY 的管理

    FUNCTION_SCHEMA =>'USER01', --返回 WHERE 子句的函数所在 SCHEMA 名称

    POLICY_FUNCTION =>'FN_GETPOLICY', --返回 WHERE 子句的函数名称

    STATEMENT_TYPES =>'SELECT,INSERT,UPDATE,DELETE', --要使用该 POLICY 的 DML 类型

    UPDATE_CHECK =>1, --仅适用于 STATEMENT_TYPE 为 'INSERT,UPDATE'，值为 1 或 0

    ENABLE =>1 --是否启用，值为 1 或 0

);
```

第四步：查询数据表(T\_POLICY)：

```
SELECT * FROM T_POLICY;
```

查询结果:

行号	T1	T2
1	B	20
2	C	30

从查询结果中可以看出少了 T2=10 这行数据。

第五步：检查更新数据

```
UPDATE T_POLICY SET T2 = 10;
```

运行结果:

[-6614]:违反策略检查约束.

注：如果 UPDATE\_CHECK 设为 1，则用户插入的值不符合 POLICY\_FUNCTION 返回条件时，该 DML 执行返回错误信息。

第六步：禁用 POLICY

```
DBMS_RLS.ENABLE_POLICY('USER01','T_POLICY','T_TESTPOLICY',0);
```

```
SELECT * FROM T_POLICY;
```

查询结果:

行号	T1	T2
1	A	10
2	B	20
3	C	30

从查询结果中可以看到 T2=10 这行数据。

第七步：删除 POLICY

```
DBMS_RLS.DROP_POLICY('USER01','T_POLICY','T_TESTPOLICY');
```

## 例 2 上下文在数据访问控制中的使用

数据准备:

```
CONN SYSDBA/SYSDBA
```

```
CREATE USER MYKGIS IDENTIFIED BY MYKGIS123;
```

```
GRANT DBA TO MYKGIS;
```

```
CONN MYKGIS/MYKGIS123
```

```
DROP TABLE EMPLOYEE;
```

```
CREATE TABLE EMPLOYEE (
```

```

    EMPID INT, FIRSTNAME VARCHAR(20),

    LASTNAME VARCHAR(20),

    LOCATION VARCHAR(20) ,

    DEPARTMENT VARCHAR(20)

);

INSERT INTO EMPLOYEE VALUES(1, 'STEVE', 'MILLER', 'GA', 'IT');

INSERT INTO EMPLOYEE VALUES(2, 'SCOTT', 'TIGER', 'GA', 'HR');

INSERT INTO EMPLOYEE VALUES(3, 'TOM', 'LUTZ', 'FL', 'HR');

INSERT INTO EMPLOYEE VALUES(4, 'HARRY', 'MYKGIS', 'FL', 'IT');

COMMIT;


DROP TABLE ORDERS;

CREATE TABLE ORDERS(

    ORDER_ID INT, AMOUNT  FLOAT,

    LOCATION VARCHAR(20),

    DEPARTMENT VARCHAR(20)

);

INSERT INTO ORDERS VALUES(1, 3454.45, 'GA', 'IT');

INSERT INTO ORDERS VALUES(2, 324893.34, 'FL', 'IT');

INSERT INTO ORDERS VALUES(3, 34545.11, 'FL', 'HR');

INSERT INTO ORDERS VALUES(4, 234.99, 'GA', 'HR');

COMMIT;

```

#### 第一步：创建上下文

```

CREATE CONTEXT DRIVCTX USING MYKGIS.APPSEC;


--创建上下文所使用的包

CREATE OR REPLACE PACKAGE MYKGIS.APPSEC AS

V_DEPARTMENT VARCHAR2(2);

V_LOCATION VARCHAR2(2);

PROCEDURE SETCONT(POLICY_GROUP VARCHAR2);

END;

```

```

/

--创建上下文所使用的包体

CREATE OR REPLACE PACKAGE BODY MYKGIS.APPSEC AS

PROCEDURE SETCONT(POLICY_GROUP VARCHAR2) AS

V_DEPARTMENT VARCHAR2(2);

V_LOCATION VARCHAR2(2);

BEGIN

DBMS_SESSION.SET_CONTEXT('DRIVCTX','ACTIVE_APP',POLICY_GROUP);

SELECT DEPARTMENT INTO V_DEPARTMENT

FROM MYKGIS.EMPLOYEE WHERE LASTNAME = SYS_CONTEXT('USERENV','SESSION_USER');

DBMS_SESSION.SET_CONTEXT('DRIVCTX','DEPARTMENT',V_DEPARTMENT);

SELECT LOCATION INTO V_LOCATION

FROM MYKGIS.EMPLOYEE WHERE LASTNAME = SYS_CONTEXT('USERENV','SESSION_USER');

DBMS_SESSION.SET_CONTEXT('DRIVCTX','LOCATION',V_LOCATION);

END;

END;

/

```

第二步：为表 ORDERS 定义上下文

```
DBMS_RLS.ADD_POLICY_CONTEXT('MYKGIS','ORDERS','DRIVCTX','ACTIVE_APP');
```

第三步：创建策略函数

```

CREATE OR REPLACE FUNCTION MYKGIS.DEP_POLICY(D1 VARCHAR2, D2 VARCHAR2)

RETURN VARCHAR2 AS

BEGIN

RETURN 'SYS_CONTEXT(''DRIVCTX'', ''DEPARTMENT'') = DEPARTMENT';

END;

/

CREATE OR REPLACE FUNCTION MYKGIS.LOC_POLICY(D1 VARCHAR2, D2 VARCHAR2)

RETURN VARCHAR2 AS

BEGIN

```

```

RETURN 'SYS_CONTEXT(''DRIVCTX'', 'LOCATION') = LOCATION';

END;

/

```

#### 第四步：创建策略组

```

DBMS_RLS.CREATE_POLICY_GROUP('MYKGIS','ORDERS','DEPARTMENT');

DBMS_RLS.CREATE_POLICY_GROUP('MYKGIS','ORDERS','LOCATION');

```

#### 第五步：创建策略

```

DBMS_RLS.ADD_GROUPED_POLICY('MYKGIS','ORDERS','DEPARTMENT','DEP_SECURITY'
,'MYKGIS','DEP_POLICY');

DBMS_RLS.ADD_GROUPED_POLICY('MYKGIS','ORDERS','LOCATION','LOC_SECURITY','
MYKGIS','LOC_POLICY');

```

#### 第六步：设置上下文中的策略组

```

MYKGIS.APPSEC.SETCONT('DEPARTMENT');

--查询策略函数返回结果:

SELECT MYKGIS.DEP_POLICY('MYKGIS','EMPLOYEE') FROM DUAL;

--查询结果:

SYS_CONTEXT('DRIVCTX','DEPARTMENT') = DEPARTMENT

--查询上下文的属性值

SELECT * FROM V$CONTEXT WHERE NAMESPACE = 'DRIVCTX';

NAMESPACE      ATTRIBUTE      VALUE
DRIVCTX         DEPARTMENT     IT
DRIVCTX         ACTIVE_APP     DEPARTMENT
DRIVCTX         LOCATION       FL

```

#### 第七步：查询表

```

SELECT * FROM ORDERS;

ORDER_ID      AMOUNT  LOCATION      DEPARTMENT
1             3454.45   GA             IT
2             324893.34  FL             IT

```

从查询结果中可以看出过滤掉了 DEPARTMENT 不是 IT 的数据。

第八步：重新设置上下文中的策略组

```
MYKGIS.APPSEC.SETCONT('LOCATION');
```

--查询策略函数返回结果：

```
SELECT MYKGIS.LOC_POLICY('MYKGIS', 'EMPLOYEE') FROM DUAL;
```

--查询结果：

```
SYS_CONTEXT('DRIVCTX ', 'LOCATION') = LOCATION
```

```
SELECT * FROM ORDERS;
```

查询结果如下：

ORDER_ID	AMOUNT	LOCATION	DEPARTMENT
2	324893.34	FL	IT
3	34545.11	FL	HR

从查询结果中可以看出过滤掉了 LOCATION 不是 FL 的数据。

## 17 DBMS\_SESSION 包

DBMS\_SESSION 包用来访问或者设置会话信息。

### 17.1 相关方法

#### 1. CLEAR\_ALL\_CONTEXT 过程

清理当前会话的所有上下文。该过程需通过 namespace 关联的 package 调用。

语法如下：

---

```
DBMS_SESSION.CLEAR_ALL_CONTEXT(
    NAMESPACE    VARCHAR(30)
);
```

---

#### 参数详解

- NAMESPACE 不区分大小写。

#### 2. CLEAR\_CONTEXT 过程

清理 namespace 中的 attribute 值。该过程需通过上下文 namespace 关联的 package 调用。

语法如下：

---

```
DBMS_SESSION.CLEAR_CONTEXT
    NAMESPACE          VARCHAR(30),
    CLIENT_IDENTIFIER   VARCHAR(30)
    ATTRIBUTE           VARCHAR(30)
);
```

---

#### 参数详解

- NAMESPACE 和 ATTRIBUTE 不区分大小写。
- CLIENT\_IDENTIFIER 用于全局访问的 CLIENT 标识，不起作用，如果指定，则忽略。

#### 3. IS\_SESSION\_ALIVE 函数

会话是否为活动会话，如果是，则返回 TRUE，否则返回 FALSE。

语法如下：

---

```
DBMS_SESSION.IS_SESSION_ALIVE (
    UNIQUEID BIGINT
)RETURN VARCHAR;
```

---

#### 参数详解

- UNIQUEID 为会话的唯一标识。

#### 4. LIST\_CONTEXT 过程

返回当前会话所有上下文属性和值。

语法如下：

---

```

TYPE APPCTXRECTYP IS RECORD(
    NAMESPACE VARCHAR2(30),
    ATTRIBUTE   VARCHAR2(30),
    VALUE       VARCHAR2(256)
);

TYPE APPCTXTABTYP IS TABLE OF APPCTXRECTYP INDEX BY BINARY_INTEGER;

DBMS_SESSION.LIST_CONTEXT(
    LIST OUT APPCTXTABTYP,
    SIZE OUT NUMBER
);

```

---

#### 参数详解

- LIST 为输出参数，类型为索引表。
- SIZE 为输出参数，该索引表中元素的总数。

#### 5. RESET\_PACKAGE 过程

清除当前会话上的所有 package 对象，即可以释放所有 package 对象占用的内存空间。

语法如下：

---

```

DBMS_SESSION.RESET_PACKAGE;

```

---

#### 参数详解

无。

#### 6. SET\_CONTEXT 过程

设置上下文 namespace 的属性和值。

语法如下：

---

```

DBMS_SESSION.SET_CONTEXT (
    NAMESPACE VARCHAR(30),
    ATTRIBUTE   VARCHAR(30),
    VALUE       VARCHAR(4000),
    USERNAME    VARCHAR(128),
    CLIENT_ID    VARCHAR(64)
);

```

---

#### 参数详解

- NAMESPACE 和 ATTRIBUTE 不区分大小写。
- USERNAME 和 CLIENT\_ID 参数不起作用，如果指定，则忽略。

#### 7. UNIQUE\_SESSION\_ID 函数

返回唯一会话 id。



语法如下：

---

```
DBMS_SESSION.UNIQUE_SESSION_ID
```

```
RETURN VARCHAR;
```

---

#### 8. CLOSE\_DATABASE\_LINK 过程

关闭一个指定的打开的外部链接，若该外部链接正在使用中（可通过 V\$DBLINK 进行查询），则报错。

语法如下：

---

```
DBMS_SESSION.CLOSE_DATABASE_LINK (
    DBLINK VARCHAR(128)
);
```

---

#### 参数详解

- DBLINK 为指定的要关闭的外部链接名。

## 17.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_SESSION');
```

初始化环境：

--创建上下文关联的包

```
CREATE OR REPLACE PACKAGE TEST_PK AS
    PROCEDURE SET_CONTEXT(TS_NAME VARCHAR, KEY VARCHAR, VALUE VARCHAR);
    PROCEDURE CLEAR_CONTEXT(TS_NAME VARCHAR, CLIENTID VARCHAR, KEY VARCHAR);
    PROCEDURE CLEAR_ALL_CONTEXT(TS_NAME VARCHAR);
END TEST_PK;

/

CREATE OR REPLACE PACKAGE BODY TEST_PK AS
    PROCEDURE SET_CONTEXT(TS_NAME VARCHAR, KEY VARCHAR, VALUE VARCHAR) AS
    BEGIN
        DBMS_SESSION.SET_CONTEXT(TS_NAME, KEY, VALUE);
    END;

    PROCEDURE CLEAR_CONTEXT(TS_NAME VARCHAR, CLIENTID VARCHAR, KEY VARCHAR) AS
    BEGIN
        DBMS_SESSION.CLEAR_CONTEXT(TS_NAME, CLIENTID, KEY);
    END;

    PROCEDURE CLEAR_ALL_CONTEXT(TS_NAME VARCHAR) AS
    BEGIN
        DBMS_SESSION.CLEAR_ALL_CONTEXT(TS_NAME);
    END;
END TEST_PK;
```

```

/

--创建上下文 C_TEST
CREATE OR REPLACE CONTEXT C_TEST USING TEST_PK;

```

例 1 设置上下文 C\_TEST 的属性和值。

```

CALL TEST_PK.SET_CONTEXT('C_TEST', 'K1', 'V1');
CALL TEST_PK.SET_CONTEXT('C_TEST', 'K2', 'V2');

```

例 2 查询上下文 C\_TEST 的属性 K1 的值。

```

SELECT SYS_CONTEXT('C_TEST', 'K1') FROM DUAL;

```

结果如下：

```

V1

```

例 3 清除上下文 C\_TEST 属性 K1 的值。

```

CALL TEST_PK.CLEAR_CONTEXT('C_TEST', NULL, 'K1');

```

再次查询名字空间 C\_TEST 的属性 K1 的值。

```

SELECT SYS_CONTEXT('C_TEST', 'K1') FROM DUAL;

```

结果如下：

```

NULL

```

例 4 当前会话是否为活动会话

```

SELECT DBMS_SESSION.IS_SESSION_ALIVE(DBMS_SESSION.UNIQUE_SESSION_ID) FROM DUAL;

```

结果如下：

```

TRUE

```

例 5 打印当前会话所有的上下文属性和值

```

DECLARE
TYPE APPCTXRECTYP IS RECORD (
    NAMESPACE VARCHAR2(30),
    ATTRIBUTE VARCHAR2(30),
    VALUE      VARCHAR2(256));

TYPE APPCTXTABTYP IS TABLE OF APPCTXRECTYP INDEX BY BINARY_INTEGER;

RES APPCTXTABTYP;
NUM INT;
BEGIN
    DBMS_SESSION.LIST_CONTEXT (RES, NUM);
    FOR I IN 0..NUM-1 LOOP
        PRINT RES(I).NAMESPACE;
        PRINT RES(I).ATTRIBUTE;
        PRINT RES(I).VALUE;
    END LOOP;
END;
/

```

结果如下：

```
C_TEST  
K1  
NULL  
C_TEST  
K2  
V2
```

例 6 删除上下文 C\_TEST

```
DROP CONTEXT C_TEST;
```

例 7 清理当前会话的 package 对象

```
CREATE OR REPLACE PACKAGE TEST_P IS  
CNT NUMBER := 0;  
PROCEDURE PRINT_STATUS;  
END TEST_P;  
  
/  
  
CREATE OR REPLACE PACKAGE BODY TEST_P IS  
PROCEDURE PRINT_STATUS IS  
BEGIN  
    PRINT 'TEST_P.CNT = ' || CNT;  
END;  
END;  
  
/
```

初始化并打印值：

```
BEGIN  
TEST_P.CNT := 5236;  
TEST_P.PRINT_STATUS;  
END;  
  
/
```

输出结果：

```
TEST_P.CNT = 5236
```

打印 CNT 值：

```
BEGIN  
    TEST_P.PRINT_STATUS;  
END;  
  
/
```

输出结果：

```
TEST_P.CNT = 5236
```

清理包对象：

```
CALL DBMS_SESSION.RESET_PACKAGE;
```

再打印 CNT 值：

```
BEGIN  
    TEST_P.PRINT_STATUS;  
END;
```

/

输出结果:

```
TEST_P.CNT = 0
```

## 18 DBMS\_SPACE 包

为了展示所有物理对象（文件、页）和逻辑对象（表空间、簇、段）的存储空间信息。  
通过 DBMS\_SPACE 包来获取表空间（不包含 HUGE 表空间）、文件、页、簇、段的内容。

### 18.1 数据类型

DBMS\_SPACE 包中涉及的变量和记录类型。如下统一说明：

包内变量和记录类型	解释
TS_T	表空间记录类型，用于记录表空间的信息，包括：表空间 ID、表空间名、表空间类型：1 DB 类型，2 临时文件组、表空间状态、表空间的最大空间、表空间的总大小（页）、包含文件的个数
TS_ARR_T	表空间记录类型数组
TS_ALL_ARR_T	表空间 ID 数组
FILE_T	文件记录类型，用于记录文件的信息，包括：文件路径、文件创建时间、文件读写状态 1 读，2 写文件修改的时间、修改的事务 ID、文件的总大小（M）、文件的空闲大小（M）、数据文件中连续空白页的起始页号、读页个数、写页个数、页大小（K）、读请求个数、写请求个数、文件可扩展标记、文件最大大小（M）、文件每次扩展大小（M）、文件包含的总描述页的数目
FILE_ARR_T	文件记录类型数组
FILE_ALL_ARR_T	文件的 ID 数组
SEG_T	段记录类型，用于记录段的信息，包括：表空间 ID、段 INODE 项的文件 ID、段 INODE 项的页号、段 INODE 项的页偏移、全满簇的个数、半满簇的个数、空闲簇的个数
SEG_ARR_T	段记录类型数组
SEG_ID_ARR_T	段 ID 数组
EXTENT_T	簇记录类型，用于记录簇的信息，包括：表空间号、段 ID、簇状态、簇的页标记位图、簇描述项的文件 ID、簇描述项所在页号、簇描述项的页偏移、簇的起始页号、簇的终止页号、下一个簇描述项的文件号 ID、下一个簇描述项的页号、下一个簇描述项的页偏移
EXTENT_ARR_T	簇记录类型数组
EXTENT_SIZE	簇的大小（页为单位）
PAGE_ADDR_T	页地址记录类型，用于记录页地址的信息，包括：表空间

	ID, 文件 ID, 页号
PAGE_ADDR_ARR_T	页地址记录类型数组
PAGE_N	描述页能描述的页数

## 18.2 相关方法

DBMS\_SPACE 包中包含的过程和函数如下详细介绍：

### 1. TS\_LOAD/ TS\_GET

根据输入的表空间 ID，获得表空间信息。过程和函数功能相同。

语法如下：

```
PROCEDURE TS_LOAD(
    TSID      IN  SMALLINT,
    TS_ARR OUT TS_ARR_T
);
```

语法如下：

```
FUNCTION TS_GET (
    TSID  IN  SMALLINT
)RETURN TS_ARR_T;
```

#### 参数详解

- TSID 表空间 ID。
- TS\_ARR\_T 表空间记录类型数组。

### 2. TS\_N\_LOAD / TS\_N\_GET

获得数据库中表空间的个数。过程和函数功能相同。

语法如下：

```
PROCEDURE TS_N_LOAD(
    NUMBER OUT INT
);
```

语法如下：

```
FUNCTION TS_N_GET
RETURN INT;
```

#### 参数详解

- NUMBER 表空间的个数。

### 3. TS\_ALL\_LOAD/ TS\_ALL\_GET

获得所有表空间的 ID。过程和函数功能相同。

语法如下：

```
PROCEDURE TS_ALL_LOAD(
```

---

```
TS_ALL_ARR OUT TS_ALL_ARR_T,  
OFFSET IN INT DEFAULT 1,  
NUM IN INT DEFAULT 100  
);
```

---

语法如下:

---

```
FUNCTION TS_ALL_GET(  
    OFFSET IN INT DEFAULT 0,  
    NUM IN INT DEFAULT 100  
) RETURN TS_ALL_ARR_T;
```

---

#### 参数详解

- OFFSET 数组起始位置。
- NUM 数组长度
- TS\_ALL\_ARR 表空间 ID

#### 4. FILE\_LOAD/ FILE\_GET

根据输入的表空间 ID、文件 ID，获得文件信息。过程和函数功能相同。

语法如下:

---

```
PROCEDURE FILE_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT,  
    FILE_ARR OUT FILE_ARR_T  
);
```

---

语法如下:

---

```
FUNCTION FILE_GET(  
    TS_ID IN SMALLINT,  
    FILE_ID IN SMALLINT  
) RETURN FILE_ARR_T;
```

---

#### 参数详解

- TSID 表空间 ID。
- FILE\_ID 文件号。
- FILE\_ARR\_T 文件记录类型数组。

#### 5. FILE\_ALL\_LOAD/ FILE\_ALL\_GET

获得所有文件的 ID。过程和函数功能相同。

语法如下:

---

```
PROCEDURE FILE_ALL_LOAD(  
    TS_ID IN SMALLINT,  
    FILE_ALL_ARR OUT FILE_ALL_ARR_T,  
    OFFSET IN INT DEFAULT 1,  
    NUM IN INT DEFAULT 100
```

---

---

);

---

语法如下:

---

```
FUNCTION FILE_ALL_GET(
    TS_ID IN SMALLINT,
    OFFSET IN INT DEFAULT 1,
    NUM IN INT DEFAULT 100
)RETURN FILE_ALL_ARR_T;
```

---

#### 参数详解

- TSID 表空间 ID。
- FILE\_ALL\_ARR 文件的 ID。
- OFFSET 数组起始位置。
- NUM 数组的长度。

#### 6. SEG\_LOAD\_BY\_ID/SEG\_GET\_BY\_ID

根据输入的表空间 ID、段号获得段信息。过程和函数功能相同。

语法如下:

---

```
PROCEDURE SEG_LOAD_BY_ID(
    TS_ID IN SMALLINT,
    SEGID IN INT,
    SEG_INFO OUT SEG_T
);
```

---

语法如下:

---

```
FUNCTION SEG_GET_BY_ID(
    TS_ID IN SMALLINT,
    SEGID IN INT
)RETURN SEG_T;
```

---

#### 参数详解

- TSID 表空间 ID。
- SEGID 段号。
- SEG\_INFO 段的信息。
- SEG\_T 段记录类型。

#### 7. SEG\_LOAD\_BY\_HEADER/ SEG\_GET\_BY\_HEADER

根据输入的表空间 ID、段头的文件号、页号、页内偏移，获得段信息，其中页内偏移取值只能为 62 或 72。过程和函数功能相同。

语法如下:

---

```
PROCEDURE SEG_LOAD_BY_HEADER(
    TS_ID IN SMALLINT,
    HEADER_FILE_ID IN SMALLINT,
    HEADER_PAGE_NO IN INT,
```

---



---

```
HEADER_OFFSET IN SMALLINT,  
SEG_INFO OUT SEG_T  
);
```

---

语法如下:

---

```
FUNCTION SEG_GET_BY_HEADER(  
    TS_ID IN SMALLINT,  
    HEADER_FILE_ID IN SMALLINT,  
    HEADER_PAGE_NO IN INT,  
    HEADER_OFFSET IN SMALLINT  
)RETURN SEG_T;
```

---

#### 参数详解

- TSID 表空间 ID。
- FILE\_ID 文件号。
- HEADER\_FILE\_ID 段头文件号。
- HEADER\_PAGE\_NO 段头页号。
- HEADER\_OFFSET 段头偏移量。
- SEG\_INFO 段的信息。

#### 8. SEG\_ALL\_LOAD/ SEG\_ALL\_GET

根据表空间 ID、数组相对起始位置、数组长度，获得数据库中表空间中所有段的 ID 信息。过程和函数功能相同。

语法如下:

---

```
PROCEDURE SEG_ALL_LOAD(  
    TS_ID IN SMALLINT,  
    SEG_ID_ARR OUT SEG_ID_ARR_T,  
    OFFSET IN INT DEFAULT 1,  
    NUM IN INT DEFAULT 100  
);
```

---

语法如下:

---

```
FUNCTION SEG_ALL_GET(  
    TS_ID IN SMALLINT,  
    OFFSET IN INT DEFAULT 1,  
    NUM IN INT DEFAULT 100  
)RETURN SEG_ID_ARR_T;
```

---

#### 参数详解

- TSID 表空间 ID。
- OFFSET 数组起始位置。
- NUM 数组长度。
- SEG\_ID\_ARR\_T 段 ID 信息。

## 9. SEG\_EXTENT\_LST\_LOAD/ SEG\_EXTENT\_LST\_GET

根据输入的表空间 ID、段号、链表类型、链表方向、链表相对起始位置、指定链表长度，获得 FULL 或 FREE 链表的信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE SEG_EXTENT_LST_LOAD(
    TS_ID IN SMALLINT,
    SEGID IN INT,
    EXTENT_ARR OUT EXTENT_ARR_T,
    LINKTYPE IN VARCHAR(5) DEFAULT 'FULL',
    DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
    OFFSET IN INT DEFAULT 1,
    NUM IN INT DEFAULT 100
);
```

---

语法如下：

---

```
FUNCTION SEG_EXTENT_LST_GET(
    TSID IN SMALLINT,
    SEGID IN SMALLINT,
    LINKTYPE IN VARCHAR(5) DEFAULT 'FULL',
    DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
    OFFSET IN INT DEFAULT 1,
    NUM IN INT DEFAULT 100,
) RETURN EXTENT_ARR_T;
```

---

### 参数详解

- TSID 表空间 ID。
- SEGID 段 ID。
- LINKTYPE 链表类型。
- DIRECT 链表方向。
- OFFSET 链表相对起始位置。
- NUM 指定链表长度。
- EXTENT\_ARR\_T 簇信息结构体链表

## 10. EXTENT\_SIZE\_LOAD/ EXTENT\_SIZE\_GET

获得簇的大小（页为单位）。过程和函数功能相同。

语法如下：

---

```
PROCEDURE EXTENT_SIZE_LOAD(
    EXTENT_SIZE OUT INT
);
```

---

语法如下：

---

```
FUNCTION EXTENT_SIZE_GET RETURN INT;
```

---

## 11. EXTENT\_XDESC\_PAGE\_NO\_LOAD/EXTENT\_XDESC\_PAGE\_NO\_GET

根据输入的表空间 ID、文件 ID，获得所有簇描述页的信息。过程和函数功能相同。

语法如下：

---

```
PROCEDURE EXTENT_XDESC_PAGE_NO_LOAD(
    TS_ID SMALLINT,
    FILE_ID SMALLINT,
    XPAGE_ADDR_ARR OUT PAGE_ADDR_ARR_T,
    OFFSET INT DEFAULT 1,
    NUM INT DEFAULT 100
);
```

---

语法如下：

---

```
FUNCTION EXTENT_XDESC_PAGE_NO_GET(
    TSID IN SMALLINT,
    FILE_ID IN SMALLINT,
    OFFSET IN INT DEFAULT 0,
    NUM IN INT DEFAULT 100,
)RETURN PAGE_ADDR_ARR_T;
```

---

### 参数详解

- TSID 表空间 ID。
- FILE\_ID 文件 ID。
- OFFSET 数组的起始位置。
- NUM 指定输出的长度。
- PAGE\_ADDR\_ARR\_T 页地址记录类型数组。

## 12. PAGE\_N\_LOAD/ PAGE\_N\_GET

获得一个描述页能描述的页数。过程和函数功能相同。

语法如下：

---

```
PROCEDURE PAGE_N_LOAD(
    PAGE_N OUT INT
);
```

---

语法如下：

---

```
FUNCTION PAGE_N_GET
RETURN INT;
```

---

### 参数详解

- PAGE\_N 描述页能描述的页数。

## 13. IPAGE\_NO\_ALL\_LOAD/ IPAGE\_NO\_ALL\_GET

输入表空间 ID、INODE 页链表类型 FULL 或 FREE 链表方向、链表相对起始位置、指定链表长度，输出所有 INODE 页的页号。过程和函数功能相同。

语法如下：

```
PROCEDURE IPAGE_NO_ALL_LOAD(
    TS_ID IN SMALLINT,
    PAGE_ADDR_ARR OUT PAGE_ADDR_ARR_T ,
    LINKTYPE IN VARCHAR(5) DEFAULT 'FULL',
    DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
    OFFSET IN INT DEFAULT 1,
    NUM IN INT DEFAULT 100
);
```

语法如下：

```
FUNCTION IPAGE_NO_ALL_GET(
    TS_ID IN SMALLINT,
    LINKTYPE IN VARCHAR(5) DEFAULT 'FULL',
    DIRECT IN VARCHAR(10) DEFAULT 'FORWARD',
    OFFSET IN INT DEFAULT 1,
    NUM IN INT DEFAULT 100
)RETURN PAGE_ADDR_ARR_T;
```

#### 参数详解

- TSID 表空间 ID。
- PAGE\_ADDR\_ARR\_T 页地址记录类型数组。
- LINKTYPEINODE 页链表类型：FULL 或 FREE。
- DIRECT 链表的方向。
- OFFSET 链表的相对起始位置。
- NUM 指定链表长度。

## 18.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程

SP\_CREATE\_SYSTEM\_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 获取表空间信息（表空间 ID、表空间名、表空间类型：1 DB 类型，2 临时文件组、表空间状态、表空间的总大小（页）、包含文件的个数）。

```
DECLARE
    TS_INFO DBMS_SPACE.TS_ARR_T;
BEGIN
    TS_INFO = DBMS_SPACE.TS_GET(4);
    IF TS_INFO IS NOT NULL
    THEN
        SELECT * FROM ARRAY TS_INFO;
    ELSE PRINT 'TABLESPACE IS NULL';
```

```
END IF;
END;
/
```

结果:

行号	ID	NAME	CACHE	TYPE	STATUS	MAX_SIZE	TOTAL_SIZE	FILE_NUM
1	4	MAIN	1		0	0	16384	1

例 2 获得数据库中表空间的个数

```
DECLARE
TS_NUM INT;
BEGIN
TS_NUM = DBMS_SPACE.TS_N_GET;
PRINT 'TS_NUM: ';
PRINT TS_NUM;
END;
/
```

结果:

```
TS_NUM:
5
```

--个数根据实际表空间个数不同而不同, 不包括 HUGE 表空间。

例 3 获得文件信息 (文件路径、文件创建时间、文件读写状态 1 读, 2 写文件修改的时间、修改的事务 ID、文件的总大小 (M)、文件的空闲大小 (M)、数据文件中连续空白页的起始页号、读页个数、写页个数、页大小 (K)、读请求个数、写请求个数、文件可扩展标记、文件最大大小 (M)、文件每次扩展大小 (M)、文件包含的总描述页的数目)

```
DECLARE
FILE_INFO DBMS_SPACE.FILE_ARR_T;
BEGIN
FILE_INFO = DBMS_SPACE.FILE_GET(4,0);
IF FILE_INFO IS NOT NULL
THEN
SELECT * FROM ARRAY FILE_INFO;
ELSE PRINT 'FILE IS NULL';
END IF;
END;
/
```

结果:

行号	PATH	CREATE_TIME	MODIFY_TIME	TOTAL_SIZE	FREE_SIZE	FREE_PAGE_NO	PAGE_SIZE	AUTOEXTEND	MAX_SIZE	NEXT_SIZE
1	D:\DATABASE0718\DAMENG\MAIN.DBF	2012-07-19 09:47:55.000000	2012-08-07 09:43:33.000000	16384	16348	368	8192	1	0	0

## 19 DBMS\_SQL 包

支持在 PL/SQL 中使用动态 SQL 语句。

### 19.1 相关方法

DBMS\_SQL 包中所用到的结构定义如下：

```
TYPE BINARY_FLOAT_TABLE IS TABLE OF FLOAT INDEX BY INTEGER;
TYPE BLOB_TABLE IS TABLE OF BLOB INDEX BY INTEGER;
TYPE CLOB_TABLE IS TABLE OF CLOB INDEX BY INTEGER;
TYPE DATE_TABLE IS TABLE OF DATETIME INDEX BY INTEGER;
TYPE DESC_TAB IS TABLE OF DESC_REC INDEX BY INTEGER;
TYPE DESC_TAB2 IS TABLE OF DESC_REC2 INDEX BY INTEGER;
TYPE DESC_TAB3 IS TABLE OF DESC_REC3 INDEX BY INTEGER;
TYPE INTERVAL_DAY_TO_SECOND_TABLE IS TABLE OF INTERVAL DAY TO SECOND INDEX BY
INTEGER;
TYPE INTERVAL_YEAR_TO_MONTH_TABLE IS TABLE OF INTERVAL YEAR TO MONTH INDEX BY
INTEGER;
TYPE INT_TABLE IS TABLE OF INT INDEX BY INTEGER;
TYPE NUMBER_TABLE IS TABLE OF NUMBER INDEX BY INTEGER;
TYPE TIME_TABLE IS TABLE OF TIME INDEX BY INTEGER;
TYPE TIME_WITH_TIME_ZONE_TABLE IS TABLE OF TIME WITH TIME ZONE INDEX BY INTEGER;
TYPE TIMESTAMP_TABLE IS TABLE OF DATETIME INDEX BY INTEGER;
TYPE TIMESTAMP_WITH_LTZ_TABLE IS TABLE OF TIMESTAMP WITH LOCAL TIME ZONE INDEX
BY INTEGER;
TYPE TIMESTAMP_WITH_TIME_ZONE_TABLE IS TABLE OF TIMESTAMP WITH TIME ZONE INDEX
BY INTEGER;
TYPE VARCHAR2_TABLE IS TABLE OF VARCHAR2(2000) INDEX BY INTEGER;
TYPE VARCHAR2A IS TABLE OF VARCHAR2(32767) INDEX BY INTEGER;
```

DESC\_REC 的结构如下：

```
TYPE DESC_REC IS RECORD(
    COL_TYPE          INTEGER := 0,
    COL_MAX_LEN       INTEGER := 0,
    COL_NAME          VARCHAR2(128) := '',
    COL_NAME_LEN      INTEGER := 0,
    COL_SCHEMA_NAME   VARCHAR2(128) := '',
    COL_SCHEMA_NAME_LEN INTEGER := 0,
    COL_PRECISION     INTEGER := 0,
    COL_SCALE         INTEGER := 0,
    COL_CHARSETID     INTEGER := 0,
    COL_CHARSETFORM   INTEGER := 0,
```

---

```
COL_NULL_OK          BOOLEAN          := TRUE);
```

使用 DESCRIBE\_COLUMNS 函数可以将查询项的数据类型填充到 DESC\_REC 中,其中服务器部分数据类型和 DBMS\_SQL 中的数据类型的映射关系如下:

TYPECODE_VARCHAR	INTEGER := 1;
TYPECODE_NUMBER	INTEGER := 2;
TYPECODE_INT	INTEGER := 2;
TYPECODE_SMALLINT	INTEGER := 2;
TYPECODE_TINYINT	INTEGER := 2;
TYPECODE_BIGINT	INTEGER := 2;
TYPECODE_FLOAT	INTEGER := 2;
TYPECODE_VARCHAR2	INTEGER := 9;
TYPECODE_DATE	INTEGER := 12;
TYPECODE_VARBINARY	INTEGER := 23;
TYPECODE_RAW	INTEGER := 95;
TYPECODE_CHAR	INTEGER := 96;
TYPECODE_BLOB	INTEGER := 113;
TYPECODE_BFILE	INTEGER := 114;
TYPECODE_CLOB	INTEGER := 112;
TYPECODE_TIME	INTEGER := 170;
TYPECODE_TIME_TZ	INTEGER := 171;
TYPECODE_TIMESTAMP	INTEGER := 180;
TYPECODE_TIMESTAMP_TZ	INTEGER := 181;
TYPECODE_TIMESTAMP_LTZ	INTEGER := 232;
TYPECODE_INTERVAL_YM	INTEGER := 182;
TYPECODE_INTERVAL_DS	INTEGER := 183;
TYPECODE_REF	INTEGER := 110;
TYPECODE_VARRAY	INTEGER := 247;

DBMS\_SQL 包中包含的过程和函数如下详细介绍:

#### 1. BIND\_ARRAY

按照 SQL 语句中变量名将变量值绑定到游标的索引表上。

语法如下:

---

```
PROCEDURE BIND_ARRAY (
    CURID          IN INTEGER,
    NAME           IN VARCHAR2,
    <TABLE_VARIABLE> IN <DATATYPE>
    [, INDEX1      IN INTEGER,
    INDEX2         IN INTEGER] )
;
```

---

#### 参数详解

- <DATATYPE>

是 DBMS\_SQL 包中索引表类型。如下所示：

- ◆ BINARY\_FLOAT\_TABLE
- ◆ BLOB\_TABLE
- ◆ CLOB\_TABLE
- ◆ DATE\_TABLE
- ◆ INTERVAL\_DAY\_TO\_SECOND\_TABLE
- ◆ INTERVAL\_YEAR\_TO\_MONTH\_TABLE
- ◆ INT\_TABLE
- ◆ NUMBER\_TABLE
- ◆ TIME\_TABLE
- ◆ TIME\_WITH\_TIME\_ZONE\_TABLE
- ◆ TIMESTAMP\_TABLE
- ◆ TIMESTAMP\_WITH\_LTZ\_TABLE
- ◆ TIMESTAMP\_WITH\_TIME\_ZONE\_TABLE
- ◆ VARCHAR2\_TABLE
- ◆ VARCHAR2A

## 2. BIND\_VARIABLE

按照 SQL 语句中变量名将变量值绑定到游标的基本数据类型变量上。

语法如下：

---

```
PROCEDURE BIND_VARIABLE (
    C           IN INTEGER,
    NAME        IN VARCHAR2,
    VALUE       IN <DATATYPE>
);
```

---

### 参数详解

- C 游标。
- NAME 变量名称。
- VALUE 变量值。可以为任何类型，类型<DATATYPE>是 DBMS\_SQL 包中基本数据类型，如下所示：

- ◆ FLOAT
- ◆ CLOB
- ◆ DATETIME
- ◆ INTERVAL DAY TO SECOND



- ◆ NUMBER
- ◆ TIME
- ◆ TIME WITH TIME ZONE
- ◆ TIMESTAMP WITH TIME ZONE
- ◆ VARCHAR
- ◆ INTERVAL YEAR TO MONTH
- ◆ INT
- ◆ SMALLINT
- ◆ TINYINT
- ◆ BIGINT
- ◆ VARBINARY
- ◆ BLOB

### 3. BIND\_VARIABLE\_RAW

绑定二进制数据类型变量，可通过 OUT\_VALUE\_SIZE 指定 BINARY 长度。

语法如下：

---

```
BIND_VARIABLE_RAW (  
    C           IN INTEGER,  
    NAME        IN VARCHAR2,  
    VALUE       IN BINARY[,OUT_VALUE_SIZE IN INTEGER]  
);
```

---

### 4. BIND\_VARIABLE\_CHAR

绑定 CHAR 数据类型变量，可通过 OUT\_VALUE\_SIZE 指定 CHAR 长度。

语法如下：

---

```
BIND_VARIABLE_CHAR (  
    C           IN INTEGER,  
    NAME        IN VARCHAR2,  
    VALUE       IN CHAR[,OUT_VALUE_SIZE IN INTEGER]  
);
```

---

### 5. BIND\_VARIABLE\_ROWID

绑定 ROWID 到游标的 BIGINT 数据类型变量上。

语法如下：

---

```
BIND_VARIABLE_ROWID (  
    C           IN INTEGER,  
    NAME        IN VARCHAR2,  
    VALUE       IN BIGINT  
);
```

---

## 6. CLOSE\_CURSOR

关闭所给定的游标。

语法如下：

---

```
DBMS_SQL.CLOSE_CURSOR (
    C      IN OUT INTEGER
);
```

---

## 7. COLUMN\_VALUE

根据列的位置，返回游标中的列值，通常在 FETCH\_ROWS 后被调用。<DATATYPE>可以是 DBMS\_SQL 中的基本数据类型和索引表类型。

语法如下：

---

```
DBMS_SQL.COLUMN_VALUE (
    C              IN  INTEGER,
    POSITION        IN  INTEGER,
    VALUE          OUT <DATATYPE>
    [, COLUMN_ERROR OUT NUMBER]
    [, ACTUAL_LENGTH OUT INTEGER]
);
```

---

### 参数详解

- C 游标。
- POSITION 为对应动态 SQL 中列的位置。
- VALUE 列值。

## 8. COLUMN\_VALUE\_CHAR

根据列的位置，返回数据类型为 CHAR 的列值。

语法如下：

---

```
COLUMN_VALUE_CHAR (
    C              IN  INTEGER,
    POSITION        IN  INTEGER,
    VALUE          OUT CHAR,
    [, COLUMN_ERROR OUT NUMBER]
    [, ACTUAL_LENGTH OUT INTEGER]
);
```

---

## 9. COLUMN\_VALUE\_RAW

根据列的位置，返回数据类型为 BINARY 的列值。

语法如下：

---

```
COLUMN_VALUE_RAW (
    C              IN  INTEGER,
    POSITION        IN  INTEGER,
    VALUE          OUT BINARY,
```

---

---

```
[ ,COLUMN_ERROR    OUT NUMBER]
[ ,ACTUAL_LENGTH    OUT INTEGER]);
```

---

#### 10. COLUMN\_VALUE\_ROWID

根据列的位置，返回数据类型为 BIGINT 的列值。

语法如下：

---

```
COLUMN_VALUE_ROWID (
    C                IN  INTEGER,
    POSITION          IN  INTEGER,
    VALUE            OUT ROWID,
    [ ,COLUMN_ERROR    OUT NUMBER]
    [ ,ACTUAL_LENGTH    OUT INTEGER]);
```

---

#### 11. DEFINE\_ARRAY

定义索引表类型的接收列，用于接收 FETCH 结果集，一次可获取多行数据。

语法如下：

---

```
DBMS_SQL.DEFINE_ARRAY (
    C                IN  INTEGER,
    POSITION          IN  INTEGER,
    <TABLE_VARIABLE> IN <DATATYPE>
    CNT              IN  INTEGER,
    LOWER_BND        IN  INTEGER
);
```

---

#### 12. DEFINE\_COLUMN

定义基本数据类型接收列，用于接收 FETCH 结果集，一次获取一行数据。

语法如下：

---

```
DBMS_SQL.DEFINE_COLUMN (
    C                IN  INTEGER,
    POSITION          IN  INTEGER,
    COLUMN           IN  <DATATYPE>
);
```

---

#### 参数详解

- C 游标。
- POSITION 为对应动态 SQL 中的位置(从 1 开始)。
- COLUMN 该值所对应的变量,可以为任何类型。

#### 13. EXECUTE

执行给定游标内的 SQL 语句。

语法如下：

---

```
DBMS_SQL.EXECUTE (
    C    IN  INTEGER
```

---

---

```
) RETURN INTEGER;
```

---

### 参数详解

- C 游标。

### 返回值

1 表示成功, 0 表示失败。处理结果只对 INSERT,DELETE,UPDATE 操作才有意义, 而对 SELECT 语句来说可以忽略。

#### 14. EXECUTE\_AND\_FETCH FUNCTION

执行给定游标内的 SQL 语句, 同时将 FETCH 数据。

语法如下:

---

```
DBMS_SQL.EXECUTE_AND_FETCH (  
    C          IN INTEGER,  
    EXACT      IN BOOLEAN DEFAULT FALSE  
)RETURN INTEGER;
```

---

#### 15. FETCH\_ROWS

FETCH 指定游标中的数据, 同时返回 FETCH 的行数。为 0 时表示已经取到游标末端。

语法如下:

---

```
DBMS_SQL.FETCH_ROWS (  
    C IN INTEGER  
)RETURN INTEGER;
```

---

#### 16. IS\_OPEN

判断游标是否打开, 若打开则返回 TRUE, 否则返回 FALSE。

语法如下:

---

```
DBMS_SQL.IS_OPEN (  
    C          IN INTEGER  
)RETURN BOOLEAN;
```

---

#### 17. OPEN\_CURSOR

打开一个游标, 返回游标号。

语法如下:

---

```
DBMS_SQL.OPEN_CURSOR  
RETURN INTEGER;
```

---

#### 18. PARSE

解析 SQL 语句。

语法如下:

---

```
DBMS_SQL.PARSE (  
    C          IN INTEGER,  
    STATEMENT   IN VARCHAR2,
```

---

---

```
LANGUAGE_FLAG      IN   INTEGER
);
```

---

### 参数详解

- C 游标。
- STATEMENT 动态游标所提供的 SQL 语句。
- LANGUAGE\_FLAG 该参数只为兼容 ORACLE 数据库，没有实际意义。

### 19. TO\_CURSOR\_NUMBER

将一个外部游标转化为 DBMS\_SQL 包内部的游标，返回游标号。

语法如下：

---

```
DBMS_SQL.TO_CURSOR_NUMBER (
    RC IN OUT SYS_REFCURSOR
)RETURN INTEGER;
```

---

### 20. TO\_REFCURSOR

将 DBMS\_SQL 包内的游标转换为 PL/SQL 游标。

语法如下：

---

```
DBMS_SQL.TO_REFCURSOR (
    CURSOR_NUMBER IN OUT INTEGER
)RETURN SYS_REFCURSOR;
```

---

### 21. DESCRIBE\_COLUMNS

获取查询项的描述信息，需要一个执行过查询操作的游标作为输入参数。

语法如下：

---

```
DBMS_SQL.DESCRIBE_COLUMNS (
    C              IN   INTEGER,
    COL_CNT        OUT  INTEGER,
    DESC_T         OUT  DESC_TAB
);
```

---

### 22. VARIABLE\_VALUE

获取变量的值，通常需要先执行 BIND\_VARIABLE。

语法如下：

---

```
DBMS_SQL.VARIABLE_VALUE (
    C              IN   INTEGER,
    NAME           IN   VARCHAR2,
    VALUE          OUT  NOCOPY <DATATYPE>
);
```

---

### 23. VARIABLE\_VALUE\_CHAR

根据返回数据类型为 CHAR 的变量值。

语法如下：

---

```
VARIABLE_VALUE_CHAR (
    C          IN  INTEGER,
    NAME       IN  VARCHAR2,
    VALUE      OUT CHAR
);
```

---

#### 24. VARIABLE\_VALUE\_RAW

根据返回数据类型为 RAW 的变量值。

语法如下：

---

```
VARIABLE_VALUE_RAW (
    C          IN  INTEGER,
    NAME       IN  VARCHAR2,
    VALUE      OUT BINARY
);
```

---

#### 25. VARIABLE\_VALUE\_ROWID

根据返回数据类型为 BIGINT 的变量值。

语法如下：

---

```
VARIABLE_VALUE_ROWID (
    C          IN  INTEGER,
    NAME       IN  VARCHAR2,
    VALUE      OUT BIGINT
);
```

---

#### 26. DEFINE\_COLUMN\_CHAR

定义 VARCHAR 数据类型接收列，用于接收 FETCH 结果集，一次获取一行数据。

语法如下：

---

```
DBMS_SQL.DEFINE_COLUMN_CHAR(
    C          IN  INTEGER,
    POSITION    IN  INTEGER,
    COLUMN     IN  VARCHAR,
    COLUMN_SIZE IN  INTEGER
);
```

---

#### 参数详解

- C 游标。
- POSITION 为对应动态 SQL 中列的位置。
- COLUMN 列值。
- COLUMN\_SIZE 可接收的最大列值。

#### 27. DEFINE\_COLUMN\_RAW

定义 RAW 数据类型接收列，用于接收 FETCH 结果集，一次获取一行数据。

语法如下：

---

```
DBMS_SQL.DEFINE_COLUMN_RAW(
    C                IN INTEGER,
    POSITION          IN INTEGER,
    COLUMN           IN RAW,
    COLUMN_SIZE      IN INTEGER
);
```

---

#### 参数详解

- C 游标。
- POSITION 为对应动态 SQL 中列的位置。
- COLUMN 列值。
- COLUMN\_SIZE 可接收的最大列值。

#### 28. DEFINE\_COLUMN\_LONG

定义 CLOB 数据类型接收列，用于接收 FETCH 结果集，一次获取一行数据。

语法如下：

---

```
DBMS_SQL.DEFINE_COLUMN_LONG(
    C                IN INTEGER,
    POSITION          IN INTEGER
);
```

---

#### 参数详解

- C 游标。
- POSITION 为对应动态 SQL 中列的位置。

## 19.2 创建语句

创建 DBMS\_SQL 系统包。可以使用如下语法或者 [1.1 系统包创建、删除语句](#) 中的语法。

语法如下：

---

```
void
SP_UPDATE_DBMS_SQL_PACKAGES ( )
```

---

#### 返回值

无

#### 举例说明

创建 DBMS\_SQL 系统包。

```
SP_UPDATE_DBMS_SQL_PACKAGES();
```

## 19.3 举例说明

使用包内的过程和函数之前，如果还未创建过例子中的系统包，请先调用系统过程创建系统包。

```
SP_UPDATE_DBMS_SQL_PACKAGES();

SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_OUTPUT');
```

例下面是利用 DBMS\_SQL 动态执行语句的例子。

### 例 1.

--一次获取一行数据

```
CREATE TABLE T1(N1 NUMBER);
INSERT INTO T1 VALUES(1.1),(2.1),(3.2),(4.3);
--非 COLLECTION 类型
DECLARE
    C        NUMBER;
    D1       NUMBER;
BEGIN
    C := DBMS_SQL.OPEN_CURSOR;--打开游标
    DBMS_SQL.PARSE(C, 'SELECT N1 FROM SYSDBA.T1', '1');--解析 SQL 语句
    DBMS_SQL.DEFINE_COLUMN(C, 1, D1);--定义列，将来可用 COLUMN_VALUE 获取列值
    DBMS_SQL.EXECUTE(C);--执行语句
    DBMS_SQL.FETCH_ROWS(C);--获取结果集
    DBMS_SQL.COLUMN_VALUE(C, 1, D1);--获取列值
    DBMS_SQL.CLOSE_CURSOR(C);--关闭游标
    DBMS_OUTPUT.PUT_LINE(D1);
END;
```

输出 1.1

### 例 2.

--一次获取多行数据

```
DECLARE
    C        NUMBER;
    D1       DBMS_SQL.NUMBER_TABLE;
BEGIN
    C := DBMS_SQL.OPEN_CURSOR;--打开游标
    DBMS_SQL.PARSE(C, 'SELECT N1 FROM SYSDBA.T1', '1');--解析 SQL 语句
    DBMS_SQL.DEFINE_ARRAY(C, 1, D1, 3, 1);--定义列，将来可用 COLUMN_VALUE 获取列值
    DBMS_SQL.EXECUTE(C);--执行语句
    DBMS_SQL.FETCH_ROWS(C);--获取结果集
    DBMS_SQL.COLUMN_VALUE(C, 1, D1);--获取列值
```



```

DBMS_SQL.CLOSE_CURSOR(C);--关闭游标

DBMS_OUTPUT.PUT_LINE(D1(2));--输出 D1 中第二条列值

END;

```

输出 2.1

### 例 3.

--绑定参数

```

DECLARE
    C          NUMBER;
    D1         NUMBER;
BEGIN
    D1 := 9.12;
    C := DBMS_SQL.OPEN_CURSOR;--打开游标
    DBMS_SQL.PARSE(C, 'INSERT INTO SYSDBA.T1 VALUES(:C1)', '1');--解析 SQL 语句
    DBMS_SQL.BIND_VARIABLE(C, 'C1', D1);
    DBMS_SQL.EXECUTE(C);--执行语句
    DBMS_SQL.CLOSE_CURSOR(C);--关闭游标
END;

SELECT * FROM T1;

```

输出:

行号	N1
1	1.1000000000000000E+000
2	2.1000000000000000E+000
3	3.2000000000000000E+000
4	4.3000000000000000E+000
5	9.1199999999999999E+000

### 例 4.

--批量绑定

```

DECLARE
    C          NUMBER;
    D1         DBMS_SQL.NUMBER_TABLE;
BEGIN
    D1(1) := 12.3;
    D1(2) := 6.12;
    D1(3) := 8.12;
    D1(4) := 342.12;
    C := DBMS_SQL.OPEN_CURSOR;--打开游标
    DBMS_SQL.PARSE(C, 'INSERT INTO SYSDBA.T1 VALUES(:C1)', '1');--解析 SQL 语句

```

```
DBMS_SQL.BIND_ARRAY(C, 'C1', D1);
DBMS_SQL.EXECUTE(C); -- 执行语句
DBMS_SQL.CLOSE_CURSOR(C); -- 关闭游标
END;
```

```
SELECT * FROM T1;
```

输出:

行号	N1
1	1.100000000000000E+000
2	2.100000000000000E+000
3	3.200000000000000E+000
4	4.300000000000000E+000
5	9.119999999999999E+000
6	1.230000000000000E+001
7	6.120000000000000E+000
8	8.119999999999999E+000
9	3.421200000000000E+002

9 ROWS GOT

## 20 DBMS\_TRANSACTION 包

DBMS\_TRANSACTION 包提供获得当前活动事务号的功能。

### 20.1 相关方法

返回当前活动事务号。

语法如下:

---

```
FUNCTION LOCAL_TRANSACTION_ID (
    CREATE_TRANSACTION BOOLEAN := FALSE
) RETURN VARCHAR2;
```

---

#### 参数详解

- CREATE\_TRANSACTION

表示如果当前没有活动事务，是否创建一个新的事务。TRUE 创建；FALSE 不创建，返回空。默认为 FALSE。

## 20.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_TRANSACTION');
```

例返回当前事务号，若当前没有活动事务，则返回一个新的事务的事务号

```
SELECT DBMS_TRANSACTION.LOCAL_TRANSACTION_ID(TRUE);
```

结果如下：

```
1871
```

## 21 DBMS\_STATS 包

优化统计信息描述了数据库中的对象细节。查询优化使用这些信息选择最合适的执行计划。使用 DBMS\_STATS 包来收集统计、删除信息，将收集的统计信息记录在数据字典中。

### 21.1 数据类型

DBMS\_STATS 包中涉及到类型。如下统一说明。

OBJECTELEM 类型是 DBMS\_STATS 专有类型。用户不能引用和改变该记录的内容。

OBJECTELEM 记录类型定义如下：

---

```
TYPE OBJECTELEM IS RECORD (
    OWNNAME      VARCHAR(128) ,
    OBJTYPE      VARCHAR(6) ,
    OBJNAME      VARCHAR(128) ,
    PARTNAME     VARCHAR(128) ,
    SUBPARTNAME  VARCHAR(128)
);
```

---

#### 参数详解

- OWNNAME 模式名。
- OBJTYPE 对象类型，TABLE 或 INDEX。
- OBJNAME 对象（表或索引）名称，区分大小写。
- PARTNAME 分区名称，区分大小写。
- SUBPARTNAME 子分区名称，区分大小写。

OBJECTTAB 为 OBJECTELEM 类型的索引表：

```
TYPE OBJECTTAB IS TABLE OF OBJECTELEM INDEX BY INT;
```

---

### 21.2 相关方法

DBMS\_STATS 包中包含的过程和函数如下详细介绍：

#### 1. COLUMN\_STATS\_SHOW

根据模式名，表名和列名获得该列的统计信息。返回两个结果集：一个是列的统计信息；另一个是直方图的统计信息。

语法如下：

---

```
PROCEDURE COLUMN_STATS_SHOW (
    OWNNAME      IN  VARCHAR(128),
    TABNAME      IN  VARCHAR(128),
```

---

```
COLNAME      IN  VARCHAR(128)
);
```

列的统计信息和直方图的统计信息，格式分别如下：

表列统计信息

名称	解释
NUM_DISTINCT	不同列值的个数
LOW_VALUE	列最小值
HIGH_VALUE	列最大值
NUM_NULLS	空值的个数
NUM_BUCKETS	直方图桶的个数
SAMPLE_SIZE	样本容量
HISTOGRAM	直方图的类型

表直方图的统计信息

名称	解释
OWNER	模式名
TABLE_NAME	表名
COLUMN_NAME	列名
HISTOGRAM	直方图类型
ENDPOINT_VALUE	样本值
ENDPOINT_HEIGHT	对于频率直方图，样本值的个数；对于等高直方图，小于样本值大于前一个样本值的个数。
ENDPOINT_KEYGHT	对于频率直方图无效；对于等高直方图，样本值的个数。
ENDPOINT_DISTINCT	对于频率直方图无效；对于等高直方图，小于样本值大于前一个样本值之间不同样本的个数。

### 参数详解

- OWNNAME 模式名，区分大小写。
- TABNAME 表名，区分大小写。
- COLNAME 列名，区分大小写。

## 2. TABLE\_STATS\_SHOW

根据模式名，表名获得该表的统计信息。

语法如下：

```
PROCEDURE TABLE_STATS_SHOW (
    OWNNAME      IN  VARCHAR(128),
    TABNAME      IN  VARCHAR(128)
);
```

表的统计信息，格式如下：

表统计信息

名称	解释
NUM_ROWS	表的总行数
LEAF_BLOCKS	总的页数
LEAF_USED_BLOCKS	已经使用的页数

#### 参数详解

- OWNNAME 模式名，区分大小写。
- TABNAME 表名，区分大小写。

### 3. INDEX\_STATS\_SHOW

根据模式名，索引名获得该索引的统计信息。返回两个结果集：一个是索引的统计信息；另一个是直方图的统计信息。

语法如下：

```
PROCEDURE INDEX_STATS_SHOW (
    OWNNAME      IN  VARCHAR(128),
    INDEXNAME    IN  VARCHAR(128)
);
```

索引的统计信息和直方图的统计信息，格式分别如下：

表索引统计信息

名称	解释
BLEVEL	B_Tree 的层次
LEAF_BLOCKS	页数
DISTINCT_KEYS	不同样本的个数
CLUSTERING_FACTOR	聚集因子，表示索引的数据分布与物理分布之间的关系
NUM_ROWS	行数
SAMPLE_SIZE	样本容量

表直方图的统计信息

名称	解释
OWNER	模式名
NAME	索引名
COLUMN_NAME	列名
HISTOGRAM	直方图类型
ENDPOINT_VALUE	样本值
ENDPOINT_HEIGHT	对于频率直方图，样本值的个数； 对于等高直方图，小于样本值大于前一个样本值的个数。
ENDPOINT_KEYGHT	对于频率直方图无效；对于等高直方图，样本值的个数。
ENDPOINT_DISTINCT	对于频率直方图无效；对于等高直方图，小于样本值大于前一个样本值之间不同样本的个数。

**参数详解**

- OWNNAME 模式名，区分大小写。
- INDEXNAME 索引名，区分大小写。

**4. GATHER\_TABLE\_STATS**

根据设定的参数，收集表、表中的列和表上的索引的统计信息。其中，对于表，只搜集表的总行数、总的页数、已经使用的页数等基本信息。

语法如下：

```
PROCEDURE GATHER_TABLE_STATS (
    OWNNAME          VARCHAR(128),
    TABNAME          VARCHAR(128),
    PARTNAME         VARCHAR(128) DEFAULT NULL,
    ESTIMATE_PERCENT  DOUBLE   DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    BLOCK_SAMPLE     BOOLEAN  DEFAULT FALSE,
    METHOD_OPT        VARCHAR  DEFAULT GET_PREFS('METHOD_OPT'),
    DEGREE           INT      DEFAULT TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY      VARCHAR  DEFAULT GET_PREFS('GRANULARITY'),
    CASCADE          BOOLEAN  DEFAULT TO_CASCADE_TYPE(GET_PREFS('CASCADE')),
    STATTAB          VARCHAR  DEFAULT NULL,
    STATID           VARCHAR  DEFAULT NULL,
    STATOWN          VARCHAR  DEFAULT NULL,
```

---

```

NO_INVALIDATE          BOOLEAN          DEFAULT          TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
FORCE                  BOOLEAN DEFAULT FALSE
);

```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- TABNAME 表名，区分大小写。
- PARTNAME 分区表名，默认为 NULL，区分大小写。
- ESTIMATE\_PERCENT 收集的百分比，范围为 0.000001~100，默认系统自定。
- BLOCK\_SAMPLE 保留参数，是否使用随机块代替随机行，默认为 TRUE。
- METHOD\_OPT  
控制列的统计信息集合和直方图的创建的格式，默认为 FOR ALL COULMNS SIZE AUTO。  
其中 BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、BOOLEAN 类型不能被收集。格式选项如下：

```

FOR ALL [INDEXED | HIDDEN] COLUMNS [<size_clause>]

<size_clause>:: = SIZE {INTEGER | REPEAT | AUTO | SKEWONLY}

或者

FOR COLUMNS[<size clause>] <<column_name>| [<size_clause>]>{,<column_name |
[<size_clause>]>}

```

各参数解释如下：

- ◆ INDEXED | HIDDEN 表示只统计索引或者隐藏的列，缺省为都统计
- ◆ INTEGER 直方图的桶数，范围 1~254
- ◆ REPEAT 只统计已经有直方图的列
- ◆ AUTO 根据数据分布和工作量自动决定统计直方图的列
- ◆ SKEWONLY 根据数据分布决定统计直方图的列
- DEGREE 保留参数，收集的并行度，默认为 1。
- GRANULARITY

保留参数，收集的粒度，默认为 AUTO；GRANULARITY 可选参数如下：AUTO | DEFAULT | ALL | PARTITION | SUBPARTITION | GLOBAL | GLOBAL AND PARTITION。各参数解释如下：

- ◆ ALL 收集全部的统计信息（SUBPARTITION，PARTITION 和 GLOBAL）
- ◆ AUTO 默认值，根据分区的类型来决定如何收集
- ◆ DEFAULT 和 AUTO 功能相同
- ◆ GLOBAL 收集 GLOBAL 表的统计信息



- ◆ GLOBAL AND PARTITION 收集 GLOBAL 和 PARTITION 表的统计信息
- ◆ PARTITION 收集 PARTITION 表的统计信息
- ◆ SUBPARTITION 收集 SUBPARTITION 表的统计信息。当子分区表个数超过 50 时，因为采用跳跃采样，所以统计信息会有误差。即使采样率为 100，也还是会有误差。可以通过减少统计层次的方式降低这种误差，即将 GRANULARITY=>'SUBPARTITION' 修改成 GRANULARITY=>'GLOBAL AND PARTITION' 或 GRANULARITY=>'GLOBAL'。
- CASCADE  
是否收集索引信息，TRUE 或 FALSE，默认为 TRUE。当表对象类型为 HUGE 表时，CASCADE 参数应置为 FALSE。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。

#### 5. GATHER\_INDEX\_STATS

根据设定的参数，收集索引的统计信息。

语法如下：

---

```

PROCEDURE GATHER_INDEX_STATS (
    OWNNAME          VARCHAR(128),
    INDNAME          VARCHAR(128),
    PARTNAME         VARCHAR(128) DEFAULT NULL,
    ESTIMATE_PERCENT  DOUBLE                                DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    STATTAB          VARCHAR DEFAULT NULL,
    STATID           VARCHAR DEFAULT NULL,
    STATOWN          VARCHAR DEFAULT NULL,
    DEGREE           INT          DEFAULT
TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY      VARCHAR DEFAULT GET_PREFS('GRANULARITY'),
    NO_INVALIDATE     BOOLEAN                                DEFAULT
TO_NO_INVALIDATE_TYPE(GET_PREFS('NO_INVALIDATE')),
    FORCE             BOOLEAN DEFAULT FALSE
);

```

---

#### 参数详解

- OWNNAME 模式名，区分大小写。
- INDNAME 索引名，区分大小写。

- PARTNAME 分区索引名，默认为 NULL，区分大小写。
- ESTIMATE\_PERCENT 收集的百分比，范围为 0.000001~100，默认系统自定。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- DEGREE 保留参数，收集的并行度，默认为 1。
- GRANULARITY 保留参数，收集的粒度，默认为 ALL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。

## 6. GATHER\_SCHEMA\_STATS

收集模式下对象的统计信息。

语法如下：

---

```
PROCEDURE GATHER_SCHEMA_STATS (
    OWNNAME          VARCHAR(128),
    ESTIMATE_PERCENT  DOUBLE   DEFAULT
TO_ESTIMATE_PERCENT_TYPE(GET_PREFS('ESTIMATE_PERCENT')),
    BLOCK_SAMPLE     BOOLEAN  DEFAULT FALSE,
    METHOD_OPT        VARCHAR  DEFAULT GET_PREFS('METHOD_OPT'),
    DEGREE            INT     DEFAULT TO_DEGREE_TYPE(GET_PREFS('DEGREE')),
    GRANULARITY       VARCHAR  DEFAULT GET_PREFS('GRANULARITY'),
    CASCADE           BOOLEAN  DEFAULT TO_CASCADE_TYPE(GET_PREFS('CASCADE')),
    STATTAB           VARCHAR  DEFAULT NULL,
    STATID            VARCHAR  DEFAULT NULL,
    OPTIONS           VARCHAR  DEFAULT 'GATHER',
    OBJLIST           OUT  OBJECTTAB DEFAULT NULL,
    STATOWN           VARCHAR  DEFAULT NULL,
    NO_INVALIDATE     BOOLEAN  DEFAULT TO_NO_INVALIDATE_TYPE
(GET_PREFS('NO_INVALIDATE')),
    FORCE              BOOLEAN  DEFAULT FALSE,
    OBJ_FILTER_LIST   OBJECTTAB DEFAULT NULL
);
```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- ESTIMATE\_PERCENT 收集的百分比，范围为 0.000001~100，默认系统自定。
- BLOCK\_SAMPLE 保留参数，是否使用随机块代替随机行，默认为 TRUE。
- METHOD\_OPT  
控制列的统计信息集合和直方图的创建：默认为 FOR ALL COULMNS SIZE AUTO；只支持其中一种格式：

---

```
FOR ALL [INDEXED | HIDDEN] COLUMNS [<size_clause>]
```

```
<size_clause>:= SIZE {integer | REPEAT | AUTO | SKEWONLY}
```

- DEGREE 保留参数，收集的并行度，默认为 1。
- GRANULARITY 保留参数，收集的粒度，默认为 ALL。
- CASCADE 是否收集索引信息，TRUE 或 FALSE。默认为 TRUE。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- OPTIONS

控制收集的列，默认为 NULL；选项如下：GATHER|GATHER AUTO|GATHER STALE|GATHER EMPTY|LIST AUTO|LIST STALE|LIST EMPTY。各选项解释如下：

- ◆ GATHER：收集模式下所有对象的统计信息。
- ◆ GATHER AUTO：自动收集需要的统计信息。系统隐含的决定哪些对象需要新的统计信息，以及怎样收集这些统计信息。此时，只有 OWNNAME, STATTAB, STATID, OBJLIST AND STATOWN有效，返回收集统计信息的对象。
- ◆ GATHER STALE：对旧的对象收集统计信息。返回找到的旧的对象。
- ◆ GATHER EMPTY：收集没有统计信息对象的统计信息。返回这些对象。
- ◆ LIST AUTO：返回GATHER AUTO方式处理的对象。
- ◆ LIST STALE：返回旧的对象信息。
- ◆ LIST EMPTY：返回没有统计信息的对象。
- OBJLIST 返回 OPTION 选项对应的链表，默认为 NULL。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。
- OBJ\_FILTER\_LIST 存放过滤条件的模式名、表名和子表名，默认为 NULL。

#### 7. DELETE\_TABLE\_STATS

根据设定参数，删除与表相关对象的统计信息。

语法如下：

---

```
PROCEDURE DELETE_TABLE_STATS (
    OWNNAME          VARCHAR(128),
    TABNAME           VARCHAR(128),
    PARTNAME          VARCHAR(128) DEFAULT NULL,
    STATTAB           VARCHAR DEFAULT NULL,
    STATID            VARCHAR DEFAULT NULL,
    CASCADE_PARTS     BOOLEAN  DEFAULT TRUE,
    CASCADE_COLUMNS   BOOLEAN  DEFAULT TRUE,
```

---

---

```

    CASCADE_INDEXES BOOLEAN DEFAULT TRUE,
    STATOWN          VARCHAR DEFAULT NULL,
    NO_INVALIDATE    BOOLEAN          DEFAULT          TO_NO_INVALIDATE_TYPE
    (GET_PREFS('NO_INVALIDATE')),
    FORCE             BOOLEAN DEFAULT FALSE
);

```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- TABNAME 表名，区分大小写。
- PARTNAME 分区表名，默认为 NULL，区分大小写。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- CASCADE\_PARTS 是否级联删除分区表信息，默认为 TRUE。
- CASCADE\_COLUMNS 是否级联删除表中列的信息，TRUE 或 FALSE。默认为 TRUE。
- CASCADE\_INDEXES 是否级联删除表的索引信息，TRUE 或 FALSE。默认为 TRUE。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。

### 8. DELETE\_SCHEMA\_STATS

根据设定参数，删除模式下对象的统计信息。

语法如下：

---

```

PROCEDURE DELETE_SCHEMA_STATS (
    OWNNAME          VARCHAR(128),
    STATTAB          VARCHAR DEFAULT NULL,
    STATID           VARCHAR DEFAULT NULL,
    STATOWN          VARCHAR DEFAULT NULL,
    NO_INVALIDATE    BOOLEAN          DEFAULT          TO_NO_INVALIDATE_TYPE
    (GET_PREFS('NO_INVALIDATE')),
    FORCE             BOOLEAN DEFAULT FALSE
);

```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。

## 9. DELETE\_INDEX\_STATS

根据设定参数，删除索引的统计信息。

语法如下：

---

```
PROCEDURE DELETE_INDEX_STATS (
    OWNNAME          VARCHAR(128),
    INDNAME           VARCHAR(128),
    PARTNAME          VARCHAR(128) DEFAULT NULL,
    STATTAB           VARCHAR DEFAULT NULL,
    STATID            VARCHAR DEFAULT NULL,
    CASCADE_PARTS     BOOLEAN  DEFAULT TRUE,
    STATOWN           VARCHAR DEFAULT NULL,
    NO_INVALIDATE     BOOLEAN                                DEFAULT      TO_NO_INVALIDATE_TYPE
    (GET_PREFS('NO_INVALIDATE')),
    FORCE              BOOLEAN  DEFAULT FALSE
);
```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- INDNAME 索引名，区分大小写。
- PARTNAME 分区表名，默认为 NULL，区分大小写。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- CASCADE\_PARTS 是否级联删除分区表信息，TRUE 或 FALSE。默认为 TRUE。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。

## 10. DELETE\_COLUMN\_STATS

根据设定参数，删除列的统计信息。

语法如下：

---

```
PROCEDURE DELETE_COLUMN_STATS (
    OWNNAME          VARCHAR(128),
    TABNAME          VARCHAR(128),
    COLNAME           VARCHAR(128),
    PARTNAME          VARCHAR(128) DEFAULT NULL,
    STATTAB           VARCHAR DEFAULT NULL,
    STATID            VARCHAR DEFAULT NULL,
    CASCADE_PARTS     BOOLEAN  DEFAULT TRUE,
    STATOWN           VARCHAR DEFAULT NULL,
    NO_INVALIDATE     BOOLEAN                                DEFAULT
    TO_NO_INVALIDATE_TYPE(GET_PREFS('NO_INVALIDATE')),
);
```

---

---

```

FORCE          BOOLEAN DEFAULT FALSE,
COL_STAT_TYPE  VARCHAR DEFAULT 'ALL'
);

```

---

### 参数详解

- OWNNAME 模式名，区分大小写。
- TABNAME 表名，区分大小写。
- COLNAME 列名，区分大小写。
- PARTNAME 分区表名，默认为 NULL，区分大小写。
- STATTAB 保留参数，统计信息存放的表，默认为 NULL。
- STATID 保留参数，统计信息的 ID，默认为 NULL。
- CASCADE\_PARTS 是否级联删除分区表信息，TRUE 或 FALSE。默认为 TRUE。
- STATOWN 保留参数，统计信息的模式，默认为 NULL。
- NO\_INVALIDATE 保留参数，是否让依赖游标失效，默认为 TRUE。
- FORCE 保留参数，是否强制收集统计信息，默认为 FALSE。
- COL\_STAT\_TYPE 保留参数，是否只删除直方图信息，默认为 ALL。

### 11. UPDATE\_ALL\_STATS

更新已有的统计信息。

语法如下：

---

```

PROCEDURE UPDATE_ALL_STATS ();

```

---

### 12. CONVERT\_RAW\_VALUE

根据设定参数，删除列的统计信息。

语法如下：

---

```

PROCEDURE CONVERT_RAW_VALUE (
    I_RAW          VARBINARY(8188),
    M_N            OUT NUMBER
);

```

---

```

PROCEDURE CONVERT_RAW_VALUE (
    I_RAW          VARBINARY(8188),
    M_N            OUT DATETIME
);

```

---

```

PROCEDURE CONVERT_RAW_VALUE (
    I_RAW          VARBINARY(8188),
    M_N            OUT VARCHAR(8188)
);

```

---

```

PROCEDURE CONVERT_RAW_VALUE (
    I_RAW          VARBINARY(8188),
    M_N            OUT DOUBLE
);

```

---

**参数详解**

- I\_RAW 输入参数, VARBINARY 类型的最大或最小值, 期望来源于 SYSSTATS 表的 V\_MAX 或 V\_MIN 列。
- M\_N 输出参数, 由 I\_RAW 转换而来, 支持 NUMBER、DATETIME、VARCHAR(8188)、DOUBLE 类型。

**13. CREATE\_STAT\_TABLE**

创建一个统计信息表, 用于保存待导出的统计信息。统计信息保存到该表中后, 可以使用 DM 的数据导入导出工具进行跨实例导入导出。

语法如下:

---

```
PROCEDURE CREATE_STAT_TABLE (
    STATOWN          VARCHAR(128),
    STATTAB          VARCHAR(128),
    TABLESPACE      VARCHAR(128) DEFAULT NULL,
    GLOBAL_TEMPORARY BOOLEAN DEFAULT FALSE
);
```

---

**参数详解**

- STATOWN 统计信息表的模式名, 区分大小写。
- STATTAB 统计信息表名, 区分大小写。
- TABLESPACE 表空间名, 默认为 NULL, 区分大小写。
- GLOBAL\_TEMPORARY 是否创建为会话级的全局临时表。

**14. DROP\_STAT\_TABLE**

删除统计信息表。

语法如下:

---

```
PROCEDURE DROP_STAT_TABLE (
    STATOWN          VARCHAR(128),
    STATTAB          VARCHAR(128)
);
```

---

**参数详解**

- STATOWN 统计信息表的模式名, 区分大小写。
- STATTAB 统计信息表名, 区分大小写。

**15. EXPORT\_TABLE\_STATS**

把目标表的统计信息导出到指定统计信息表中。

语法如下:

---

```
PROCEDURE EXPORT_TABLE_STATS(
    OWNNAME          VARCHAR(128),
    TABNAME          VARCHAR(128),
```

---

---

```

PARTNAME      VARCHAR(128) DEFAULT NULL,
STATTAB       VARCHAR(128),
STATID        VARCHAR(128) DEFAULT '',
CASCADE       BOOLEAN   DEFAULT TRUE,
STATOWN       VARCHAR(128) DEFAULT NULL,
STAT_CATEGORY VARCHAR(128) DEFAULT NULL
);

```

---

#### 参数详解

- OWNNAME 目标表的模式名，区分大小写。
- TABNAME 目标表名，区分大小写。
- PARTNAME 目标表分区名，如果不指定分区，则一起导出所有子表的统计信息，默认为 NULL。
- STATTAB 统计信息表名，区分大小写。
- STATID 由用户指定的统计信息标识名，默认为空字符串。
- CASCADE 是否连列和索引的统计信息一起导出，默认为 TRUE
- STATOWN 统计信息表的模式名，区分大小写，默认为 NULL。
- STAT\_CATEGORY 仅保留参数以兼容 ORACLE，功能暂未实现。

#### 16. IMPORT\_TABLE\_STATS

把统计信息表中的统计信息导入到目标表中。

语法如下：

---

```

OWNNAME      VARCHAR(128),
TABNAME      VARCHAR(128),
PARTNAME     VARCHAR(128) DEFAULT NULL,
STATTAB      VARCHAR(128),
STATID       VARCHAR(128) DEFAULT '',
CASCADE      BOOLEAN   DEFAULT TRUE,
STATOWN      VARCHAR(128) DEFAULT NULL,
NO_INVALIDATE BOOLEAN   DEFAULT
TO_NO_INVALIDATE_TYPE(GET_PREFS('NO_INVALIDATE')),
FORCE        BOOLEAN   DEFAULT FALSE,
STAT_CATEGORY VARCHAR(128) DEFAULT NULL
);

```

---

#### 参数详解

- OWNNAME 目标表的模式名，区分大小写。
- TABNAME 目标表名，区分大小写。
- PARTNAME 目标表分区名，如果不指定分区，则一起导入所有子表的统计信息，默认为 NULL。
- STATTAB 统计信息表名，区分大小写。



- STATID 由用户指定的统计信息标识名，默认为空字符串。
- CASCADE 是否连列和索引的统计信息一起导出，默认为 TRUE
- STATOWN 统计信息表的模式名，区分大小写，默认为 NULL。
- NO\_INVALIDATE 仅保留参数以兼容 ORACLE，功能暂未实现。
- FORCE 仅保留参数以兼容 ORACLE，功能暂未实现。
- STAT\_CATEGORY 仅保留参数以兼容 ORACLE，功能暂未实现。

## 21.3 约束

以下对象不支持统计信息：

1. 外部表、DBLINK 远程表、动态视图表、记录类型数组所用的临时表。
2. 所在表空间为 OFFLINE 的对象。
3. 位图索引，位图连接索引、虚索引、全文索引、空间索引、数组索引、无效的索引。
4. BLOB、IMAGE、LONGVARBINARY、CLOB、TEXT、LONGVARCHAR、自定义类型列和空间类型列等列类型。

## 21.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_STATS');
```

例 1 收集模式 PERSON 下表 ADDRESS 的统计信息，并打印收集的表信息。

```
BEGIN
  DECLARE
    OBJTAB DBMS_STATS.OBJECTTAB;
    OBJ_FILTER_LIST DBMS_STATS.OBJECTTAB;
  BEGIN
    OBJ_FILTER_LIST(0).OWNNAME = 'PERSON';
    OBJ_FILTER_LIST(0).OBJTYPE = 'TABLE';
    OBJ_FILTER_LIST(0).OBJNAME = 'ADDRESS';
    DBMS_STATS.GATHER_SCHEMA_STATS(
      'PERSON',
      1.0,
      FALSE,
      'FOR ALL COLUMNS SIZE AUTO',
      1,
      'AUTO',
      TRUE,
      NULL,
      NULL,
```

```

        'GATHER',
        OBJTAB,
        NULL,
        TRUE,
        TRUE,
        OBJ_FILTER_LIST
    );
    PRINT OBJTAB.COUNT;
    FOR I IN 0..OBJTAB.COUNT-1 LOOP
        PRINT OBJTAB(I).OWNNAME;
        PRINT OBJTAB(I).OBJTYPE;
        PRINT OBJTAB(I).OBJNAME;
        PRINT OBJTAB(I).PARTNAME;
        PRINT OBJTAB(I).SUBPARTNAME;
        PRINT '-----';
    END LOOP;
END;
/

```

打印结果

```

1
PERSON
TABLE
ADDRESS
NULL
NULL

```

例 2 获得 PERSON 模式下表 ADDRESS 中列 ADDRESSID 的统计信息。

```
DBMS_STATS.COLUMN_STATS_SHOW('PERSON','ADDRESS','ADDRESSID');
```

返回结果集 1:

NUM_DISTINCT	LOW_VALUE	HIGH_VALUE	NUM_NULLS	NUM_BUCKETS	SAMPLE_SIZE	HISTOGRAM
16	1	16	0	16	16	FREQUENCY

结果集 2

OWNER	TABLE_NAME	COLUMN_NAME	HISTOGRAM	ENDPOINT_VALUE	ENDPOINT_HEIGHT	ENDPOINT_KEYHEIGHT	ENDPOINT_DISTINCT
PERSON	ADDRESS	ADDRESSID	FREQUENCY	1	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	2	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	3	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	4	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	5	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	6	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	7	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	8	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	9	1	NULL	NULL
PERSON	ADDRESS	ADDRESSID	FREQUENCY	10	1	NULL	NULL

```
PERSON ADDRESS ADDRESSID FREQUENCY 11 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 12 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 13 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 14 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 15 1 NULL NULL
PERSON ADDRESS ADDRESSID FREQUENCY 16 1 NULL NULL
```

例 3 删除 PERSON 模式下表 ADDRESS 的统计信息。

```
BEGIN
    DBMS_STATS.DELETE_TABLE_STATS ( 'PERSON' , 'ADDRESS' );
END;
/
```

## 22 DBMS\_UTILITY 包

部分兼容 ORACLE 的 DBMS\_UTILITY 包，提供一些实用的方法。

### 22.1 相关方法

#### 1) FORMAT\_ERROR\_STACK

获取 PL/SQL 异常的堆栈信息。

语法如下：

---

```
FUNCTION  FORMAT_ERROR_STACK (
)RETURN VARCHAR2;
```

---

#### 2) GET\_HASH\_VALUE

对于指定的字符串，返回范围在[base, base+hase\_size-1]的散列值。

语法如下：

---

```
FUNCTION  GET_HASH_VALUE(
    NAME      VARCHAR,
    BASE      INT,
    HASH_SIZE  INT
)RETURN INT;
```

---

#### 参数详解

- NAME 指定的要被 HASH 的字符串
- BASE 返回的 HASH 值的基准值，即返回值的最小值
- HASH\_SIZE 指定的 HASH 表的大小

#### 3) GET\_TIME

返回一个代表当前时间的以 1/100 秒为精度的值，该值并不真正表示当前的时间值，此方法常用来计算两个时间点的间隔。

语法如下：

---

```
FUNCTION  GET_TIME() RETURN NUMBER;
```

---

#### 参数详解

无

#### 4) FORMAT\_CALL\_STACK

返回 PL/SQL 当前调用的堆栈，包括每层的栈帧方法名。

语法如下：

---

```
FUNCTION  FORMAT_CALL_STACK() RETURN VARCHAR2;
```

---

5) FORMAT\_ERROR\_BACKTRACE

返回 PL/SQL 发生异常时的堆栈信息，包括每层的栈帧地址和方法名。

语法如下：

---

```
FUNCTION  FORMAT_ERROR_BACKTRACE() RETURN VARCHAR2;
```

---

## 23 DBMS\_WORKLOAD\_REPOSITORY 包

数据库快照是一个只读的静态的数据库。DM 快照功能是基于数据库实现的，每个快照是基于数据库的只读镜像。通过检索快照，可以获取源数据库在快照创建时间点的相关数据信息。

为了方便管理自动工作集负载信息库 AWR (Automatic Workload Repository) 的信息，系统为其所有重要统计信息和负载信息执行一次快照，并将这些快照存储在 AWR 中。AWR 功能默认是关闭的，如果需要开启，则调用 DBMS\_WORKLOAD\_REPOSITORY.AWR\_SET\_INTERVAL 过程设置快照的间隔时间。DBMS\_WORKLOAD\_REPOSITORY 包还负责 snapshot (快照) 的管理。

DM 数据库在创建该包时，默认创建一个名为 SYSAUX 的表空间，对应的数据文件为 SYSAUX.DBF，该表空间用于存储该包生成快照的数据。如果该包被删除，那么 SYSAUX 表空间也对应地被删除。

DM MPP 环境下不支持 DBMS\_WORKLOAD\_REPOSITORY 包。

### 23.1 相关方法

1. AWR\_CLEAR\_HISTORY();

清理之前的所有 snapshot 记录。

语法如下：

---

```
PROCEDURE AWR_CLEAR_HISTORY();
```

---

2. AWR\_SET\_INTERVAL();

设置生成 snapshot 的时间间隔。

语法如下：

---

```
PROCEDURE AWR_SET_INTERVAL(
    AWR_INTERVAL    IN INT DEFAULT 60
);
```

---

#### 参数详解

##### ● AWR\_INTERVAL

时间间隔，单位分钟。有效范围为 [10, 525600]，默认值为 60。参数为 0 时，关闭快照（关闭时参数值为 40150 分钟(110 年)，是一个无效的值）。

3. AWR\_REPORT\_HTML

生成 html 格式的报告

定义 1:

---

```

FUNCTION AWR_REPORT_HTML(
    START_SNAP_ID          IN INT,
    END_SNAP_ID            IN INT
) RETURN AWRRPT_ROW_TYPE PIPELINED;

```

---

### 参数详解

- START\_SNAP\_ID 为起始 SNAPSHOT\_ID。
- END\_SNAP\_ID 为终止 SNAPSHOT\_ID。

### 返回值

返回包含报告的全部 html 脚本信息的嵌套表类型 AWRRPT\_ROW\_TYPE。

把 awr 数据报表生成到指定路径的 html 文件。

### 定义 2:

---

```

PROCEDURE SYS.AWR_REPORT_HTML(
    START_ID IN INT,
    END_ID IN INT,
    DEST_DIR IN VARCHAR(128),
    DEST_FILE IN VARCHAR(128)
);

```

---

### 参数详解

- START\_ID 为起始 SNAPSHOT\_ID。
- END\_ID 为终止 SNAPSHOT\_ID。
- DEST\_DIR 为指定生成报告的目标路径。
- DEST\_FILE 为指定生成报告的目标文件名，文件名需要以 .HTM 和 .HTML 结尾。

### 4. AWR\_REPORT\_TEXT

生成 text 格式的报告

### 定义 1:

---

```

FUNCTION AWR_REPORT_TEXT(
    START_SNAP_ID          IN INT,
    END_SNAP_ID            IN INT
) RETURN AWRRPT_ROW_TYPE PIPELINED;

```

---

### 参数详解

- START\_SNAP\_ID 为起始 SNAPSHOT\_ID
- END\_SNAP\_ID 为终止 SNAPSHOT\_ID。

### 返回值

返回包含报告的全部 text 脚本信息的嵌套表类型 AWRRPT\_ROW\_TYPE。

把 awr 数据报表生成到指定路径的 text 文件。

**定义 2:**


---

```
PROCEDURE SYS.AWR_REPORT_TEXT(
    START_ID IN INT,
    END_ID IN INT,
    DEST_DIR IN VARCHAR(128),
    DEST_FILE IN VARCHAR(128)
);
```

---

**参数详解**

- START\_ID 为起始 SNAPSHOT\_ID。
- END\_ID 为终止 SNAPSHOT\_ID。
- DEST\_DIR 为指定生成报告的目标路径。
- DEST\_FILE 为指定生成报告的目标文件名，文件名需要以 .TXT 结尾。

**5. CREATE\_SNAPSHOT**

创建一次快照 snapshot

语法如下：

---

```
FUNCTION CREATE_SNAPSHOT(
    FLUSH_LEVEL IN VARCHAR2 DEFAULT 'TYPICAL'
) RETURN INT;
```

---

**参数详解**

- FLUSH\_LEVEL  
'TYPICAL' OR 'ALL'; 如果为空，则默认为 'TYPICAL'，该值会影响快照生成数据的大小，如果是 'ALL'，则将全部历史数据保存，如果是 'TYPICAL' 则会刷部分数据，具体在后续会涉及到。

**返回值**

返回创建的快照 ID 值。

**6. DROP\_SNAPSHOT\_RANGE**

删除 snapshot。

语法如下：

---

```
PROCEDURE DROP_SNAPSHOT_RANGE(
    LOW_SNAP_ID IN INT,
    HIGH_SNAP_ID IN INT,
    DBID IN INT DEFAULT NULL
);
```

---

**参数详解**

- LOW\_SNAP\_ID SNAPSHOT\_ID 范围的起始值。
- HIGH\_SNAP\_ID SNAPSHOT\_ID 范围的结束值。
- DBID 表示 SNAPSHOT 所在的 DB 唯一标识，默认为 NULL，表示当前 DB，目前该



参数不起作用。

## 7. MODIFY\_SNAPSHOT\_SETTINGS

设置 snapshot 的属性值。

语法如下：

---

```
PROCEDURE MODIFY_SNAPSHOT_SETTINGS(
    RETENTION      IN  INT      DEFAULT NULL,
    AWR_INTERVAL   IN  INT      DEFAULT NULL,
    TOPNSQL        IN  INT      DEFAULT NULL,
    DBID           IN  INT      DEFAULT NULL
);
```

---

```
PROCEDURE MODIFY_SNAPSHOT_SETTINGS(
    RETENTION      IN  INT      DEFAULT NULL,
    AWR_INTERVAL   IN  INT      DEFAULT NULL,
    TOPNSQL        IN  VARCHAR2,
    DBID           IN  INT      DEFAULT NULL
);
```

---

### 参数详解

#### ● RETENTION

表示 SNAPSHOT 在数据库中保留的时间，以分钟为单位，最小值为 1 天，最大值为 100 年；如果值为 0，则表示永久保留；如果值为 NULL，则表示本次设置的该值无效，保留以前的旧值。

#### ● AWR\_INTERVAL

表示每次生成 SNAPSHOT 的间隔时间，以分钟为单位，最小值为 10 分钟，最大值为 1 年；如果值为 0，则 SNAPSHOT 会失效。如果值为 NULL，则表示本次设置的该值无效，保留以前的旧值。

#### ● TOPNSQL

如果为 NULL，则保留当前设置的值。

如果为 INT 类型，则表示按照 SQL 的衡量标准（执行时间，CPU 时间，消耗内存等）获取保存的 SQL 个数，最小值为 30，最大值为 50000。

如果为 VARCHAR2 类型，则可以设置如下 3 个值：DEFAULT、MAXIMUM 和 N。DEFAULT 对应值为 100；MAXIMUM 对应值为 50000；值 N 是数字串，但要求转化为 INT 类型之后，值必须在 30 至 50000 之间。

目前该参数不起作用。

#### ● DBID

表示 SNAPSHOT 所在的 DB 唯一标识，默认为 NULL，表示当前 DB。目前该参数不起作用。

## 23.2 创建、检测、删除语句

### 23.2.1 SP\_INIT\_AWR\_SYS

创建或删除 DBMS\_WORKLOAD\_REPOSITORY 系统包。

语法如下：

---

```
void
SP_INIT_AWR_SYS(
    CREATE_FLAG      int
)

```

---

#### 参数详解

- CREATE\_FLAG

为 1 时表示创建 DBMS\_WORKLOAD\_REPOSITORY 包；为 0 表示删除该系统包。

#### 返回值

无

#### 举例说明

创建 DBMS\_WORKLOAD\_REPOSITORY 系统包。

```
SP_INIT_AWR_SYS(1);
```

### 23.2.2 SF\_CHECK\_AWR\_SYS

系统的 DBMS\_WORKLOAD\_REPOSITORY 系统包启用状态检测。

语法如下：

---

```
int
SF_CHECK_AWR_SYS ()

```

---

#### 返回值

0：未启用；1：已启用

#### 举例说明

获得 DBMS\_WORKLOAD\_REPOSITORY 系统包的启用状态。

```
SELECT SF_CHECK_AWR_SYS;
```

## 23.3 举例说明

用户在使用 DBMS\_WORKLOAD\_REPOSITORY 包之前，需要提前调用系统过程并设置间隔时间：

```
SP_INIT_AWR_SYS(1);
```

下面语句设置间隔为 10 分钟，也可以是其他值：

```
CALL DBMS_WORKLOAD_REPOSITORY.AWR_SET_INTERVAL(10);
```

设置成功后，可以使用 CREATE\_SNAPSHOT 手动创建快照，也可以等待设置的间隔时间后系统自动创建快照，快照 id 从 1 开始递增：

手动创建快照：

```
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

查看创建的快照信息，包括快照 id：

```
SELECT * FROM SYS.WRM$_SNAPSHOT;
```

查看 snapshot 的 id 在 1~2 范围内的 AWR 分析报告的带 html 格式的内容。然后复制到文本文件中，保存成 html 格式即可查看。

```
SELECT * FROM TABLE (DBMS_WORKLOAD_REPOSITORY.AWR_REPORT_HTML(1,2));
```

把 snapshot 的 id 在 1~2 范围内的 AWR 分析报告生成到 c 盘 awr1.html 文件。

```
SYS.AWR_REPORT_HTML(1,2,'C:\','AWR1.HTML');
```

通过 DMBS\_WORKLOAD\_REPOSITORY 包还可以对快照本身做增删改操作。

例 1 删除 id 在 22~32 之间的 snapshot。

```
CALL DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE(22,32);
```

例 2 修改 snapshot 的间隔时间为 30 分钟、保留时间为 1 天。

```
CALL DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(1440,30);
```

查询设置后快照参数。

```
SELECT * FROM SYS.WRM$_WR_CONTROL;
```

例 3 创建一次 snapshot。

```
CALL DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT();
```

例 4 清理全部 snapshot。

```
CALL DBMS_WORKLOAD_REPOSITORY.AWR_CLEAR_HISTORY();
```

例 5 设置 snapshot 的间隔为 10 分钟。

```
CALL DBMS_WORKLOAD_REPOSITORY.AWR_SET_INTERVAL(10);
```

## 24 DBMS\_XMLGEN 包

兼容 Oracle 的 DBMS\_XMLGEN 系统包，将 SQL 查询结果转换成 XML 文档。

### 24.1 使用前提

DBMS\_XMLGEN 包的使用依赖 DBMS\_LOB 包。所以，在创建 DBMS\_XMLGEN 包之前，请先成功创建 DBMS\_LOB 包。

### 24.2 相关方法

#### 1. NEWCONTEXT()

申请上下文句柄。

语法如下：

---

```
FUNCTION NEWCONTEXT (
    QUERYSTRING    IN VARCHAR(32767))
RETURN CTXHANDLE;
```

或

```
FUNCTION NEWCONTEXT (
    QUERY          IN CURSOR)
RETURN CTXHANDLE;
```

---

#### 参数详解

- QUERYSTRING 查询语句，字符串类型。
- QUERY 查询语句的引用游标。

#### 返回值

ctxHandle: 上下文句柄。

#### 2. GETXML()

GETXML() 方法用来获取 XML 数据，有以下三种调用方式。

- 1) 根据指定的最大获取行数，将获取的数据添加在传入的 clob 的内容之后。使用该函数可以重用 clob，避免额外的 clob 复制。

语法如下：

---

```
FUNCTION GETXML(
    CTX              IN INTEGER,
    TMPCLOB          INOUT CLOB,
    DTDORSHEMA      IN INTEGER
) RETURN INTEGER;
```

---

---

**参数详解**

- CXT 之前根据查询语句获取的上下文句柄。
- TMPCLOB 写入 XML 数据的 CLOB。
- DTDORSHEMA 仅支持 0，默认为 0，可以不指定。

**返回值**

总是为 0。

2) 此方法会将获取的 XML 数据放在一个临时生成的 clob 中，并返回这个临时 clob，该临时 clob 必须通过 DBMS\_LOB.FREETEMPORARY 进行释放。

语法如下：

---

```
FUNCTION GETXML(
    CXT          IN  INTEGER,
    DTDORSHEMA IN  INTEGER
) RETURN CLOB;
```

---

**参数详解**

- CXT 之前根据查询语句获取的上下文句柄。
- DTDORSHEMA 仅支持 0，默认为 0，可以不指定。

**返回值**

包含获取的 XML 数据的 clob。

3) 此方法根据指定的 SQL 语句，生成内部临时上下文句柄，获取 XML 数据并返回，并释放临时上下文句柄。

语法如下：

---

```
FUNCTION GETXML(
    SQLQUERY      IN  VARCHAR(32767),
    DTDORSHEMA IN  INTEGER
) RETURN CLOB;
```

---

**参数详解**

- SQLQUERY 查询语句。
- DTDORSHEMA 仅支持 0，默认为 0，可以不指定。

**返回值**

包含获取的 XML 数据的 clob。

**3. GETNUMROWSPROCESSED()**

调用 GETXML 函数之后，使用该函数获取实际获取的数据行数，该函数的返回值不包含跳过的数据。

语法如下：

---

```
FUNCTION GETNUMROWSPROCESSED(
```

---

---

```

    CTX      IN  INTEGER
)RETURN BIGINT;

```

---

#### 参数详解

- CTX 操作的上下文句柄。

#### 返回值

GETXML 实际获取的行数。

#### 4. CLOSECONTEXT()

关闭上下文句柄并释放相关资源。

语法如下：

---

```

PROCEDURE CLOSECONTEXT(
    CTX      IN  INTEGER
)RETURN BIGINT;

```

---

#### 参数详解

- CTX 待关闭的上下文句柄。

#### 5. CONVERT()

将 XML 数据在是否需要保留特殊字符之间进行转换。

语法如下：

---

```

FUNCTION CONVERT(
    XMLDATA      IN  VARCHAR(32767),
    FLAG         IN  INTEGER
)RETURN VARCHAR(32767);

```

或

```

FUNCTION CONVERT(
    XMLDATA      IN  CLOB),
    FLAG         IN  INTEGER
)RETURN CLOB;

```

---

#### 参数详解

- XMLDATA 需要转换的 XML 数据。
- FLAG  
转换标记，取值 ENTITY\_ENCODE 和 ENTITY\_DECODE，默认为 ENTITY\_ENCODE。

#### 返回值

转换之后的数据。

#### 6. RESTARTQUERY()

重新查询。

语法如下：

---

```
PROCEDURE RESTARTQUERY(  
    CTX      IN  INTEGER  
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。

#### 7. SETCONVERTSPECIALCHARS( )

设置获取结果集 XML 时特殊字符是否转化。

语法如下：

---

```
PROCEDURE SETCONVERTSPECIALCHARS(  
    CTX      IN  INTEGER,  
    CONV     IN  BOOLEAN  
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。
- CONV 是否要转化特殊字符。

#### 8. SETMAXROWS( )

设置每次调用 GETXML 时的最大获取行数。

语法如下：

---

```
PROCEDURE SETMAXROWS(  
    CTX      IN  INTEGER,  
    MAXROWS IN  BIGINT  
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。
- MAXROWS 设置的最大获取行数。

#### 9. SETNULLHANDLING( )

设置 XML 中对 NULL 值的表示方法。

语法如下：

---

```
PROCEDURE SETNULLHANDLING(  
    CTX      IN  INTEGER,  
    FLAG     IN  INTEGER  
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。
- FLAG

NULL 值的表示方法，可有如下三种取值。

- ◆ DROP\_NULLS0，也是默认值，直接跳过值为 NULL 的列。
- ◆ NULL\_ATTR1，在值为 NULL 的列的列名标签中显示 `xsi:nil="true"`。
- ◆ EMPTY\_TAG2，值为 NULL 的列仅显示列名标签。

#### 10. SETROWSETTAG( )

设置 XML 文档中 ROWSET 标签的名称，默认的标签名为 ROWSET。

语法如下：

---

```
PROCEDURE SETROWSETTAG(  
    CTX      IN  INTEGER,  
    ROWSETTAGNAME  IN  VARCHAR(32767)  
);
```

---

##### 参数详解

- CTX 操作的上下文句柄。
- ROWSETTAGNAME 设置的标签名。

#### 11. SETROWTAG( )

设置 XML 文档中 ROW 标签的名称，默认的标签名为 ROW。

语法如下：

---

```
PROCEDURE SETROWTAG(  
    CTX      IN  INTEGER,  
    ROWTAGNAME  IN  VARCHAR(32767)  
);
```

---

##### 参数详解

- CTX 操作的上下文句柄。
- ROWTAGNAME 设置的标签名。

#### 12. SETSKIPROWS( )

设置每次调用 GETXML 函数时跳过的行数，可以用在 XML 或 HTML 数据获取时各页数据的生成，第一页时，设置为 0，后面各页时，设置为前面已经获取的数据行数。

语法如下：

---

```
PROCEDURE SETSKIPROWS(  
    CTX      IN  INTEGER,  
    SKIPROWS  IN  BIGINT  
);
```

---

##### 参数详解

- CTX 操作的上下文句柄。



- SKIPROWS 跳过的行数。

### 13. USEITEMTAGSFORCOLL()

设置生成的 XML 中，集合类型元素的标签名。DM 目前仅仅语法支持，调用后无实际效果。

语法如下：

---

```
PROCEDURE USEITEMTAGSFORCOLL(
    CTX      IN  INTEGER
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。

### 14. USENULLATTRIBUTEINDICATOR()

另一种设置 NULL 值表现方式的过程，设置是否用一个 XML 属性来表示 NULL，或者忽略 NULL。

语法如下：

---

```
PROCEDURE USENULLATTRIBUTEINDICATOR(
    CTX      IN  INTEGER,
    ATTRIND  IN  BOOLEAN
);
```

---

#### 参数详解

- CTX 操作的上下文句柄。
- ATTRIND

TRUE 对应 SETNULLHANDLING 的 NULL\_ATTR，为默认值；FALSE 对应 SETNULLHANDLING 的 DROP\_NULLS。

## 24.3 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_LOB');
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_XMLGEN');
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_OUTPUT');
SET SERVEROUTPUT ON; --dbms_output.put_line 需要设置这条语句，才能打印出消息
```

#### 1. 数据准备

```
CREATE TABLE TEST_XML(A INT, B VARCHAR(30));

INSERT INTO TEST_XML VALUES(0, NULL);

DECLARE I INT; BEGIN FOR I IN 1..100 LOOP INSERT INTO TEST_XML VALUES(I, '<XML>$');
```

```
END LOOP; END;
/
COMMIT;
```

## 2. 最简单的获取 XML 数据的例子

```
DECLARE
    DTA CLOB;
BEGIN
    DTA := DBMS_XMLGEN.GETXML('SELECT * FROM TEST_XML WHERE ROWNUM < 4');
    DBMS_OUTPUT.PUT_LINE(DTA);
    DBMS_LOB.FREETEMPORARY(DTA);
END;
/
```

在这个例子中，直接指定查询语句调用 GETXML 方法，在 GETXML 中会生成临时的上下文句柄，这个句柄的所有属性都会使用缺省值。GETXML 返回一个临时生成的 CLOB，之后由程序员负责调用 DBMS\_LOB.FREETEMPORARY 来释放它。

上述 PL/SQL 执行后将打印出以下的 XML 数据。

```
<?XML VERSION="1.0"?>
<ROWSET>
<ROW>
<A>0</A>
</ROW>
<ROW>
<A>1</A>
<B>&LT;XML&GT;$</B>
</ROW>
<ROW>
<A>2</A>
<B>&LT;XML&GT;$</B>
</ROW>
</ROWSET>
```

## 3. 典型的的获取 XML 数据的例子

```
DECLARE
    CTX DBMS_XMLGEN.CTXHANDLE;
    DTA CLOB;
BEGIN
    CTX := DBMS_XMLGEN.NEWCONTEXT('SELECT * FROM TEST_XML');
    DBMS_XMLGEN.SETMAXROWS(CTX, 2);
    DTA := DBMS_XMLGEN.GETXML(CTX);
    DBMS_OUTPUT.PUT_LINE(DTA);
    DBMS_LOB.FREETEMPORARY(DTA);
    DBMS_XMLGEN.SETSKIPROWS(CTX, 5);
    DTA := DBMS_XMLGEN.GETXML(CTX);
    DBMS_OUTPUT.PUT_LINE(DTA);
```

```
DBMS_LOB.FREETEMPORARY(DTA);  
DBMS_XMLGEN.CLOSECONTEXT(CTX);  
END;  
/
```

这是一个较为典型的获取 XML 数据的例子，首先申请一个上下文句柄，然后使用这个上下文句柄进行一些属性设置，再调用 GETXML 获取 XML 数据，最后关闭上下文句柄。

上述 PL/SQL 执行后将打印出以下的 XML 数据。

```
<?XML VERSION="1.0"?>  
<ROWSET>  
<ROW>  
<A>0</A>  
</ROW>  
<ROW>  
<A>1</A>  
<B>&LT;XML&GT;$</B>  
</ROW>  
</ROWSET>  
  
<?XML VERSION="1.0"?>  
<ROWSET>  
<ROW>  
<A>7</A>  
<B>&LT;XML&GT;$</B>  
</ROW>  
<ROW>  
<A>8</A>  
<B>&LT;XML&GT;$</B>  
</ROW>  
</ROWSET>
```

## 25 UTL\_ENCODE 包

UTL\_ENCODE 包提供了将 varbinary 类型数据进行编码和解码的功能。

### 25.1 相关方法

#### 1. BASE64\_ENCODE

将 VARBINARY 格式的数据 R 编码成 base64 字符集格式,再以 varbinary 类型返回。

语法如下:

---

```
FUNCTION UTL_ENCODE.BASE64_ENCODE (  
    R IN VARBINARY  
)RETURN VARBINARY;
```

---

#### 2. BASE64\_DECODE

将 base64 字符集的编码 R, 解码成原始的 varbinary 类型数据。

语法如下:

---

```
FUNCTION UTL_ENCODE.BASE64_DECODE (  
    R IN VARBINARY  
)RETURN VARBINARY;
```

---

### 25.2 举例说明

使用包内的过程和函数之前, 如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_ENCODE');
```

例使用 BASE64\_ENCODE 和 BASE64\_deCODE 对数据进行编码和解码。

--数据准备:

```
DROP TABLE T;  
CREATE TABLE T (I VARBINARY (10));  
INSERT INTO T VALUES ('420921197908051523');
```

--使用 BASE64\_ENCODE 和 BASE64\_DECODE:

```
SELECT UTL_ENCODE.BASE64_ENCODE (I) FROM T;
```

查询结果: 0X51676B6847586B494252556A

```
SELECT UTL_ENCODE.BASE64_DECODE (UTL_ENCODE.BASE64_ENCODE (I)) FROM T;
```

查询结果: 0X420921197908051523

## 26 UTL\_FILE 包

UTL\_FILE 包为 PL/SQL 程序提供读和写操作系统数据文件的功能。它提供一套严格的使用标准操作系统文件 I/O 方式：OPEN、PUT、GET 和 CLOSE 操作。

当用户想读取或写一个数据文件的时候，可以使用 FOPEN 来返回的文件句柄。这个文件句柄将用于随后在文件上的所有操作。例如，过程 PUT\_LINE 写 TEXT 字符串和行终止符到一个已打开的文件句柄，以及使用 GET\_LINE 来读取指定文件句柄的一行到提供的缓存。

### 26.1 数据类型

UTL\_FILE 定义了一种 FILE\_TYPE 记录类型。FILE\_TYPE 类型是 UTL\_FILE 专有类型。用户不能引用和改变该记录的内容。FILE\_TYPE 记录类型定义如下：

语法如下：

---

```
TYPE FILE_TYPE IS RECORD (
    ID          INTEGER,
    DATATYPE    INTEGER,
    BYTE_MODE   BOOLEAN
);
```

---

#### 参数详解

- ID 需要处理的外部文件句柄。
- DATATYPE 文件的类型：1 表示 CHAR、2 表示 NCHAR、3 表示 BINARY。
- BYTE\_MODE 文件打开后的类型：TRUE 表示二进制模式；FALSE 表示文本模式。

### 26.2 相关方法

UTL\_FILE 包中的 18 个过程和函数详细介绍如下：



**注意：** 对于打开模式为 "w"、"a"、"wb"、"ab"、"r+"、"w+"、"a+"、"rb+"、"wb+"、"ab+" 的 FOPEN、FREMOVE 及 FRENAME 操作，有如下限制：

1. 如果是非 WINDOWS 环境，且运行达梦服务器的进程有效用户为 root，则禁止文件操作；
2. 执行操作的数据库用户必须有 DBA 权限；

3. 只能创建、修改在达梦系统目录（含子目录）下的文件。用户可以通

以下详细介绍各过程和函数：

#### 1. FCLOSE

关闭一个已打开文件。

语法如下：

---

```
PROCEDURE FCLOSE(  
    FILE      IN OUT FILE_TYPE  
);
```

---

### 参数详解

- FILE 通过 FOPEN 或 FOPEN\_NCHAR 调用，返回的活动文件句柄。

#### 2. FCLOSE\_ALL

关闭在这个会话里打开的所有文件。用作紧急情况下清理程序，比如，当 PL/SQL 存在异常时。FCLOSE\_ALL 不会改变用户打开的文件句柄的状态。即，在调用 FCLOSE\_ALL 后用 IS\_OPEN 去测试一个文件句柄，返回值仍为 TRUE，即使该文件已经关闭了。在 FCLOSE\_ALL 之前打开的所有文件都无法进一步读或写。

语法如下：

---

```
PROCEDURE FCLOSE_ALL;
```

---

#### 3. FCOPY

把一个文本文件中指定的连续的行数据拷贝到另外一个文件中。源文件需要以可读模式打开。

语法如下：

---

```
PROCEDURE FCOPY(  
    SRC_LOCATION IN VARCHAR(128),  
    SRC_FILENAME IN VARCHAR(128),  
    DEST_LOCATION IN VARCHAR(128),  
    DEST_FILENAME IN VARCHAR(128),  
    START_LINE   IN INTEGER DEFAULT 1,  
    END_LINE     IN INTEGER DEFAULT NULL  
);
```

---

### 参数详解

- SRC\_LOCATION 源文件目录。
- SRC\_FILENAME 被拷贝的源文件名称。
- DEST\_LOCATION 新创建的目标文件的目录。
- DEST\_FILENAME 源文件的新名称。
- START\_LINE 拷贝的起始行，默认为文件的第 1 行。
- END\_LINE 拷贝的终止行，默认为 NULL，即为文件的最后一行。

#### 4. FFLUSH

以物理写入的方式将待定的数据写入到文件句柄所指定的文件中。通常地，待写入文件的数据是存放在缓冲区中的。FFLUSH 程序强制将缓冲区中的数据写入到文件中。数据必须以换行符结束。

语法如下：

---

```
PROCEDURE FFLUSH(  

```

---

---

```
FILE IN FILE_TYPE
) ;
```

---

### 参数详解

- FILE 由 FOPEN 或 FOPEN\_NCHAR 调用，返回的文件句柄。

#### 5. FGETATTR

读取并返回磁盘文件的属性。

语法如下：

---

```
PROCEDURE FGETATTR(
    LOCATION    IN  VARCHAR(128),
    FILENAME    IN  VARCHAR(128),
    FEXISTS     OUT BOOLEAN,
    FILE_LENGTH OUT NUMBER,
    BLOCKSIZE   OUT NUMBER
);
```

---

### 参数详解

- LOCATION 源文件路径。
- FILENAME 被检查的文件。
- FEXISTS 文件是否存在，存在为 TRUE，不存在为 FALSE。
- FILE\_LENGTH 文件的长度（字节），若文件不存在，则长度为 NULL。
- BLOCKSIZE 文件系统中数据块的大小（字节），若文件不存在，则长度为 NULL。

#### 6. FGETPOS

指定文件中当前相对位置偏移量（字节）。

语法如下：

---

```
FUNCTION FGETPOS(
    FILE    IN FILE_TYPE
)RETURN INTEGER;
```

---

### 参数详解

- FILE 通过 FOPEN 或 FOPEN\_NCHAR 调用，返回的活动文件句柄。

### 返回值

返回值 FGETPOS 以字节的方式返回一个打开文件相对位置偏移量。如果文件未打开则发生异常。如果当前位置在文件的最开始，则返回 0。

### 注意事项：

如果文件以 BYTE 的模式打开，那么会发生 INVALID OPERATION。

#### 7. FOPEN

打开指定文件并返回一个文件句柄。用户指定文件每行最大的字符数，并且同时可以打开文件最多 50 个。

语法如下：

---

```

FUNCTION FOPEN(
    LOCATION      IN VARCHAR(128),
    FILENAME      IN VARCHAR(128),
    OPEN_MODE     IN VARCHAR(128),
    MAX_LINESIZE  IN INTEGER DEFAULT 1024
)RETURN FILE_TYPE;

```

---

### 参数详解

- **LOCATION** 源文件路径。
- **FILENAME**  
文件名称，包括文件类型，但不包含文件路径。如果文件名称中包含路径，则 **FOPEN** 忽略此处的路径。在 **UNIX** 系统中，文件名不能包含转义符：“/”。
- **OPEN\_MODE**  
文件打开模式。包括：**R** 只读模式；**W** 写模式；**A** 附加模式；**RB** 只读打开一个二进制文件，只允许读；**WB** 只写打开或建立一个二进制文件，只允许写数据；**AB** 追加打开一个二进制文件，并在文件末尾写数据。当以“**A**”或“**AB**”的方式打开文件时，若该文件不存在，则以“**W**”的方式创建该文件。
- **FILE** 通过 **FOPEN** 或 **FOPEN\_NCHAR** 调用，返回的活动文件句柄。
- **MAX\_LINESIZE** 文件每行最大的字符数，包括换行符。最小为 1，最大为 32767。

### 注意事项：

文件的路径和文件名必须加上引号，以便于和相近的路径名区分开来。

### 异常：

**INVALID\_PATH**：文件路径无效；

**INVALID\_MODE**：参数 **OPEN\_MODE** 字符串无效；

**INVALID\_OPERATION**：文件无法按照请求打开；

**INVALID\_MAXLINESIZE**：指定的最大行字符数值太大或太小。

### 8. FREMOVE

在有足够权限的情况下，从系统中删除一个文件。

语法如下：

---

```

PROCEDURE FREMOVE(
    LOCATION IN VARCHAR(128),
    FILENAME IN VARCHAR(128)
);

```

---

### 参数详解

- **LOCATION** 文件路径。
- **FILENAME** 被删除文件名称。

### 注意事项：



FREMOVE 存储过程不会在删除文件之前验证权限,但是操作系统会验证文件和路径的正确性。没有权限,或者文件和路径不正确,则返回删除失败信息。

#### 9. FRENAME

修改一个文件的名称。

语法如下:

---

```
PROCEDURE FRENAME(  
    SRC_LOCATION IN VARCHAR(128),  
    SRC_FILENAME IN VARCHAR(128),  
    DEST_LOCATION IN VARCHAR(128),  
    DEST_FILENAME IN VARCHAR(128),  
    OVERWRITE IN BOOLEAN DEFAULT FALSE  
);
```

---

#### 参数详解

- SRC\_LOCATION 要改名文件的存放目录。
- SRC\_FILENAME 要改名的源文件名称。
- DEST\_LOCATION 改名后文件存放的路径。
- DEST\_FILENAME 修改后的新名称。
- OVERWRITE  
设置是否能够重写。如果设置为“TRUE”,则在 SRC\_LOCATION 目录中覆盖任何名为 DEST\_FILENAME 的文件。若设置为“FALSE”,就会产生异常。缺省为 FLASE。

#### 10. FSEEK

根据指定的字节数,调整文件指针前进或后退。

语法如下:

---

```
PROCEDURE FSEEK(  
    FILE IN FILE_TYPE,  
    ABSOLUTE_OFFSET IN INTEGER DEFAULT NULL,  
    RELATIVE_OFFSET IN INTEGER DEFAULT NULL  
);
```

---

#### 参数详解

- FILE 通过 FOPEN 或 FOPEN\_NCHAR 调用,返回的活动文件句柄。
- ABSOLUTE\_OFFSET 要寻找的字节的绝对路径,默认为 NULL。
- RELATIVE\_OFFSET  
指针前进或后退的字节数。用正整数表示向前查找,负整数表示向后查找。

#### 注意事项:

使用 FSEEK,可以读取先前的行而不用关闭文件再重新找开。但是你必须知道要跳过的字符数。

该存储过程根据 RELATIVE\_OFFSET 参数指定的字节数进行查找。

如果在指定字节数之前就已经到达了文件的开头，那么文件指针指定在文件开头。如果在指定字节数之前就已经到达了文件的尾部，那么 `INVALID_OFFSET` 错误就会发生。

如果文件是以 `BYTE` 的模式打开的，那么 `INVALID OPERATION` 异常就会发生。

### 11. GET\_LINE

读取指定文件的一行到提供的缓存。读取的文本不包括该行的终结符，或者直到文件的末尾，或者到指定的长度。并且不能超过 `FOPEN` 时指定的 `MAX_LINESIZE`。

语法如下：

---

```
PROCEDURE GET_LINE(
    FILE          IN  FILE_TYPE,
    BUFFER        OUT VARCHAR,
    LEN           IN  INTEGER DEFAULT NULL
);
```

---

#### 参数详解

- `FILE` 通过 `FOPEN` 调用，返回的活动文件句柄。
- `BUFFER` 接收数据文件中行读的数据缓冲区。
- `LEN`

从文件中读取的字节数。默认情况为 `NULL`，`NULL` 表示读取的字节数为 `MAX_LINESIZE`。

#### 注意事项：

如果该行不匹配缓冲区，则发生 `READ_ERROR` 异常。如果到文件结束了仍没有文本读取，发生 `NO_DATA_FOUND` 异常。如果 `FILE` 是以 `BYTE` 模式打开的，则产生 `INVALID_OPERATION` 异常。

由于行的结束符不被读取，读取空行则返回空字符串。

缓冲区最大大小为 32767 字节，除非 `FOPEN` 指定了更小的值。如果未指定，默认值为 1024。

### 12. GET\_RAW

从文件中读取原始字符串值，并通过读取的字节数调整文件指针的头部。`GET_RAW` 函数忽略掉行结束符，并返回通过 `GET_RAW` 的 `LEN` 参数请求的实际字节数。

语法如下：

---

```
FUNCTION GET_RAW(
    FILE          IN  FILE_TYPE,
    BUFFER        OUT BLOB,
    LEN           IN  INTEGER DEFAULT NULL
)RETURN INTEGER;
```

---

#### 参数详解

- `FILE` 通过 `FOPEN` 或 `FOPEN_NCHAR` 调用，返回的活动文件句柄。

- BUFFER 接收从文件中读取的数据行的缓冲区。
- LEN

从文件中读取的字节数。默认情况为 NULL, NULL 表示读取的字节数为 MAX\_LINESIZE。

#### 注意事项:

如果子程序读取超过文件末尾, 那么将产生 DATA\_NO\_FOUND 的异常。程序在执行循环时应该允许捕获这个异常。

#### 13. IS\_OPEN

判断文件句柄指定的文件是否已打开, 如果已打开则返回 TRUE, 否则 FALSE。不保证用户在尝试使用该文件句柄时没有操作系统的权限。

语法如下:

---

```
FUNCTION IS_OPEN(
    FILE      IN FILE_TYPE
)RETURN BOOLEAN;
```

---

#### 参数详解

- FILE 通过 FOPEN 或 FOPEN\_NCHAR 调用, 返回的活动文件句柄。

#### 14. NEW\_LINE

在当前位置输出一个或多个行终止符。

语法如下:

---

```
PROCEDURE NEW_LINE(
    FILE      IN FILE_TYPE,
    LINES     IN INTEGER DEFAULT 1
);
```

---

#### 参数详解

- FILE 通过 FOPEN 或 FOPEN\_NCHAR 调用, 返回的活动文件句柄。
- LINES 写入文件的行终结符数量。

#### 异常:

INVALID\_FILEHANDLE  
INVALID\_OPERATION  
WRITE\_ERROR

#### 15. PUT

把缓冲区中的文本字符写入到数据文件。文件必须是以写模式打开的。UTL\_FILE.PUT 输出数据时不会附加行终止符。使用 NEW\_LINE 来添加行结束符或者使用 PUT\_LINE 来写一行带有结束符的数据。

语法如下:

---

```
PROCEDURE PUT(
    FILE      IN FILE_TYPE,
    BUFFER    IN VARCHAR
```

---

---

);

### 参数详解

- FILE 由 FOPEN\_NCHAR 返回的文件句柄。
- BUFFER 包含要写入文件的数据缓存。

### 注意事项:

如果 FOPEN 没有指定一个更小的值, BUFFER 参数的最大值将是 32767。如果未指定, FOPEN 默认的行最大值为 1024。连续调用 PUT 的总和在没有缓冲区刷页的情况下不能超过 32767。

### 异常:

INVALID\_FILEHANDLE

INVALID\_OPERATION

WRITE\_ERROR

### 16. PUT\_LINE

把缓冲区中的文本字符串写入数据文件, 同时输出一个与系统有关的行终止符。

语法如下:

---

```
PROCEDURE PUT_LINE(
    FILE      IN FILE_TYPE,
    BUFFER     IN VARCHAR,
    AUTOFLUSH IN BOOLEAN DEFAULT FALSE
);
```

---

### 参数详解

- FILE 由 FOPEN 返回的文件句柄。
- BUFFER 包含要写入文件的数据缓存。
- AUTOFLUSH 数据写完后, 自动清理缓冲区。

### 注意事项:

缓冲区大小最大为 32767。

如果文件有 BYTE 模式打开, 则产生 INVALID\_OPERATION 异常。

### 异常:

INVALID\_FILEHANDLE

INVALID\_OPERATION

WRITE\_ERROR

### 17. PUT\_RAW

接收输入的原始数据并将其写入缓存。

语法如下:

---

```
PROCEDURE PUT_RAW(
    FILE      IN FILE_TYPE,
    BUFFER     IN BLOB,
```

---

---

```
AUTOFLUSH IN BOOLEAN DEFAULT FALSE
);
```

---

### 参数详解

- FILE 由 FOPEN 返回的文件句柄。
- BUFFER 包含要写入文件的数据缓存；
- AUTOFLUSH 数据写完后，自动清理缓冲区。

### 注意事项：

通过设置第三个参数可以使缓冲区自动冲刷。

BUFFER 最大值为 32767。

### 18. PUTF

以一个模版样式输出至多 5 个字符串，类似 C 中的 PRINTF()。

语法如下：

---

```
PROCEDURE PUTF(
    FILE      IN FILE_TYPE,
    FORMAT     IN VARCHAR,
    ARG1      IN VARCHAR DEFAULT NULL,
    ARG2      IN VARCHAR DEFAULT NULL,
    ARG3      IN VARCHAR DEFAULT NULL,
    ARG4      IN VARCHAR DEFAULT NULL,
    ARG5      IN VARCHAR DEFAULT NULL
);
```

---

### 参数详解

- FILE 由 FOPEN 返回的文件句柄。
- FORMAT 决定格式的格式串。格式串可使用以下样式：1) %S 在格式串中可以使用最多 5 个 %S，与后面的 5 个参数一一对应。%S 会被后面的参数依次填充，如果没有足够的参数，%S 会被忽视，不被写入文件；2) \N 换行符。在格式串中没有个数限制。
- ARGN 可选的 5 个参数，最多 5 个。

### 注意事项：

如果文件以 BYTE 模式打开，返回 INVALID\_OPERATION 错误。

%s, \N 与 C 语言中的类似。

## 26.3 错误处理

错误、异常情况释义：

- 1、EC\_OPEN\_FILE\_FAILED：打开文件失败
- 2、EC\_RN\_INVALID\_MODE\_NAME：无效的文件模式名
- 3、EC\_WRITE\_FILE\_FAILED：写入文件失败

- 4、EC\_FILE\_CLOSE\_FAILED: 文件关闭失败
- 5、EC\_FILE\_REMOVE\_FAILED: 文件删除失败
- 6、EC\_FILE\_NAME\_TOO\_LONG: 文件名过长
- 7、EC\_FILE\_PATH\_TOO\_LONG: 文件路径名过长
- 8、EC\_CANNOT\_FIND\_DATA: 无法获取数据
- 9、EC\_INVALID\_OPERATION: 无效的操作
- 10、EC\_INVALID\_OFFSET: 无效的偏移
- 11、EC\_RN\_INVALID\_STRINGFORMAT: 无效的字符串格式
- 12、EC\_RN\_TOO\_MANY\_FILES: 打开文件数目过多
- 13、EC\_FILE\_EXIST: 文件已存在

## 26.4 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_FILE');
```

在成功调用下面 3 个例子中的过程和函数之前，先要在达梦库文件所在的系统目录

D:\DMDBMS\DATA\DAMENG 下创建文件 UTL\_FILE\_TEMP\U12345.TMP。文件的内容为：

```
THIS IS LINE 1.
THIS IS LINE 2.
THIS IS LINE 3.
THIS IS LINE 4.
THIS IS LINE 5.
```

### 例 1 使用 GET\_LINE 过程读文件

```
DECLARE
    V1 VARCHAR2(8186);
    F1 UTL_FILE.FILE_TYPE;
BEGIN
    -- 本例子中 MAX_LINESIZE 小于 GET_LINE 的长度要求，所以返回的字节数小于等于 256。
    F1 :=
    UTL_FILE.FOPEN('D:\DMDBMS\DATA\DAMENG\UTL_FILE_TEMP', 'u12345.tmp', 'R', 256);
    UTL_FILE.GET_LINE(F1, V1, 32767);
    PRINT 'GET_LINE: ' || V1;
    UTL_FILE.FCLOSE(F1);

    -- FOPEN 的 MAX_LINESIZE 是 NULL，NULL 即为默认的 1024，所以返回的字节数小于等于 1024。
    F1 := UTL_FILE.FOPEN('D:\DMDBMS\DATA\DAMENG\UTL_FILE_TEMP', 'u12345.tmp', 'R');
    UTL_FILE.GET_LINE(F1, V1, 32767);
    PRINT 'GET_LINE: ' || V1;
    UTL_FILE.FCLOSE(F1);

    -- GET_LINE 未指定字节数，所以默认为 NULL1024，NULL 即为 1024。
```

```

F1 := UTL_FILE.FOPEN('D:\DMDBMS\DATA\DAMENG\UTL_FILE_TEMP','u12345.tmp','R');
UTL_FILE.GET_LINE(F1,V1);
PRINT 'GET_LINE:' || V1;
UTL_FILE.FCLOSE(F1);
END;
/

```

每个过程都是输出第一行，这个例子的结果如下所示：

```

GET LINE: THIS IS LINE 1.
GET LINE: THIS IS LINE 1.
GET LINE: THIS IS LINE 1.

```

**例 2** 使用 PUTF 过程向文件尾部添加内容。

```

DECLARE
    HANDLE UTL_FILE.FILE_TYPE;
    MY_WORLD VARCHAR2(4) := 'ZORK';
BEGIN
    HANDLE :=
UTL_FILE.FOPEN('d:\dmdbms\data\DAMENG\UTL_FILE_TEMP','u12345.tmp','A');
    UTL_FILE.PUTF(HANDLE,'\NHELLO, WORLD!\NI COME FROM %s
WITH %s.\N',MY_WORLD,'GREETINGS FOR ALL EARTHLINGS');
    UTL_FILE.FFLUSH(HANDLE);
    UTL_FILE.FCLOSE(HANDLE);
END;
/

```

该例子在 U12345.TMP 文件的末尾添加了如下内容：

```

HELLO, WORLD!
I COME FROM ZORK WITH GREETINGS FOR ALL EARTHLINGS.

```

**例 3** 使用 GET\_RAW 过程从 UTL\_FILE\_TEMP 路径下的 U12345.TMP 文件中读取原始信息。

首先，将 U12345.TMP 文件中的内容写为：HELLO WORLD！

```

CREATE OR REPLACE PROCEDURE GETRAW(N IN VARCHAR2) IS
    H    UTL_FILE.FILE_TYPE;
    BUF   BLOB;
    AMNT  CONSTANT BINARY_INTEGER := 32767;
BEGIN
    H := UTL_FILE.FOPEN('d:\dmdbms\data\DAMENG\UTL_FILE_TEMP', N, 'R', 32767);
    UTL_FILE.GET_RAW(H, BUF, AMNT);
    PRINT 'THIS IS THE RAW DATA: ';
    SELECT BUF;
    UTL_FILE.FCLOSE (H);
END;
/

```

过程建好后，运行如下语句：

```

begin
    getraw('u12345.tmp');

```

```
end;  
/
```

输出结果为:

```
THIS IS THE RAW DATA:
```

```
行号      BUF
```

```
-----  
1
```

```
0X54484953204953204C494E4520312E0A54484953204953204C494E4520322E0A54484953204  
953204C494E4520332E0A54484953204953204C494E4520342E0A54484953204953204C494E45  
20352E5C4E48454C4C4F2C20574F524C44215C4E4920434F4D452046524F4D205A4F524B20574  
95448204752454554494E475320464F5220414C4C2045415254484C494E47532E5C4E
```



## 27 UTL\_INADDR 包

为了在 PL/SQL 中，提供网络地址转换的支持，开发 UTL\_INADDR 包，提供一组 API，实现主库的 IP 地址和主库名之间的转换。

### 27.1 相关方法

#### 1. GET\_HOST\_ADDRESS

依据给定的主库名返回其 IP 地址。返回给定 HOST 主库的 IP 地址（以 CHAR 的形式展现）。

语法如下：

---

```
FUNCTION GET_HOST_ADDRESS (
    HOST IN VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

---

#### 参数详解

##### ● HOST

主库名，VARCHAR2 类型。默认值为 NULL，用本机的主库名代替，表示获取本机的 IP 地址；

#### 2. GET\_HOST\_NAME

依据给定的 IP 地址返回主库名。

语法如下：

---

```
FUNCTION GET_HOST_NAME(
    IP IN VARCHAR2 DEFAULT NULL
) RETURN VARCHAR2;
```

---

#### 参数详解

##### ● IP

IP 地址，VARCHAR2 类型。默认值为 NULL，用本机的 IP 地址代替，表示获取本机的主库名。

### 27.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_INADDR');
```

例获得 192.168.0.30 机器的名称，以及名称为 TEST 机器的 IP 地址。

```
SELECT UTL_INADDR.GET_HOST_NAME('192.168.0.30');
SELECT UTL_INADDR.GET_HOST_ADDRESS('TEST');
```

## 28 UTL\_MAIL 包

为了在 PL/SQL 中，提供 EMAIL 发送的支持，开发 UTL\_MAIL 包，提供一组 API，实现发送简单的邮件和包含附件的邮件。

### 28.1 相关方法

#### 1. INIT

包需要的配置信息。

语法如下：

---

```
PROCEDURE INIT(  
    SMTP_SERVER VARCHAR2(128),  
    SMTP_PORT    INT DEFAULT 25,  
    SENDER       VARCHAR2(128),  
    PASSWORD     VARCHAR2(128)  
);
```

---

#### 参数详解

- SMTP\_SERVER SMTP 服务器的地址，如 SMTP.163.COM。
- SMTP\_PORT SMTP 的端口号，一般都是 25。
- SENDER 设置发送的邮箱，也就是要认证的邮箱。
- PASSWORD 设置发送邮箱的密码。

#### 2. SEND

发送 MESSAGE 的信息。

语法如下：

---

```
PROCEDURE SEND (  
    SENDER IN VARCHAR2,  
    RECIPIENTS IN VARCHAR2,  
    CC IN VARCHAR2 DEFAULT NULL,  
    BCC IN VARCHAR2 DEFAULT NULL,  
    SUBJECT IN VARCHAR2 DEFAULT NULL,  
    MESSAGE IN VARCHAR2 ,  
    MIME_TYPE IN VARCHAR2 DEFAULT 'TEXT/PLAIN; CHARSET=US-ASCII',  
    PRIORITY IN INT DEFAULT NULL  
);
```

---

#### 参数详解

- SENDER 发送的邮箱地址。
- RECIPIENTS 接收邮件的地址。
- CC 抄送的邮件的地址。

- BCC 秘密抄送的邮件的地址。
- SUBJECT 邮件的主题。
- MESSAGE 邮件的内容，正文部分。
- MIME\_TYPE 邮件的内容格式，DEFAULT IS 'TEXT/PLAIN; CHARSET=US-ASCII'。
- PRIORITY 优先级（不用设置）。

### 3. SEND\_ATTACH\_RAW

发送带附件的邮件。

语法如下：

---

```
PROCEDURE SEND_ATTACH_RAW (  
    SENDER IN VARCHAR2,  
    RECIPIENTS IN VARCHAR2,  
    CC IN VARCHAR2 DEFAULT NULL,  
    BCC IN VARCHAR2 DEFAULT NULL,  
    SUBJECT IN VARCHAR2 DEFAULT NULL,  
    MESSAGE IN VARCHAR2 DEFAULT NULL,  
    MIME_TYPE IN VARCHAR2 DEFAULT 'TEXT/PLAIN; CHARSET=US-ASCII',  
    PRIORITY IN INT DEFAULT NULL,  
    ATTACHMENT IN BLOB,  
    ATT_INLINE IN BOOLEAN DEFAULT TRUE,  
    ATT_MIME_TYPE IN VARCHAR2 DEFAULT 'APPLICATION/OCTET-STREAM',  
    ATT_FILENAME IN VARCHAR2 DEFAULT NULL  
);
```

---

### 参数详解

- SENDER 发送的邮箱地址。
- RECIPIENTS 接收邮件的地址。
- CC 抄送的邮件的地址。
- BCC 秘密抄送的邮件的地址。
- SUBJECT 邮件的主题。
- MESSAGE 邮件的内容，正文部分。
- MIME\_TYPE  
邮件的内容格式：文本、HTML 等。默认为 'TEXT/PLAIN;CHARSET=US-ASCII'。
- PRIORITY 优先级（不用设置）。
- ATTACHMENT 附件。
- ATT\_INLINE 附件在文件中显示是否使用下划线。
- ATT\_MIME\_TYPE 邮件附件的类型，如文本。
- ATT\_FILENAME 附件的文件名。

## 28.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_MAIL');
```

例在使用该包前，需要先初始化该包，调用 INIT 函数，设置使用 SMTP 服务器，端口（一般默认 25，设置用于认证的邮箱，认证的密码。设置完后，就可以使用包的函数：SEND 和 SEND\_ATTACH\_RAW。

```
/*初始化包的函数*/
UTL_MAIL.INIT(
    'SMTP.163.COM',      //SMTP 服务器
    25,                  //端口
    'UTL_MAIL@163.COM', //认证的邮箱
    'DM_UTL_MAIL'        //认证邮箱的密码
);
/*发送不带附件的邮件*/
UTL_MAIL.SEND(
    'UTL_MAIL@163.COM',      //发送的邮箱地址
    'RECIPIENT@DAMENG.COM',  //接受的邮箱地址
    '',
    '',
    'TEST',                  //主题
    'THIS IS A MAIL FOR TEST .', //正文
    'TEXT/PLAIN; CHARSET=US-ASCII', //类型
    3
);
/* 发送带附件的邮件 */
UTL_MAIL.SEND_ATTACH_RAW(
    'UTL_MAIL@163.COM',
    'RECIPIENT@DAMENG.COM',
    '',
    '',
    'TEST',
    'THIS IS A MAIL FOR TEST .',
    'TEXT/PLAIN; CHARSET=US-ASCII',
    3,
    HEXTORAW('123456789056453181316943419643165139832084312646546546546546156
4'), //附件二进制的内容
    FALSE,
    'APPLICATION/OCTET-STREAM', //附件的类型
    'ATTACH.ZIP'                //附件的名字
);
```



## 29 UTL\_MATCH 包

为了在 DM 上兼容 oracle 的 utl\_match 包，提供功能上与 oracle 基本一致的 utl\_match 包。利用这个包提供计算两个字符串差异字符个数和相似度的函数。

### 29.1 相关方法

utl\_match 包所支持的 2 个函数，分别计算源字符串和目标字符串的差异字符个数和相似度。

#### 1. edit\_distance

输出字符串 s1 转化为 s2 所需要被插入、删除或替换字符的个数。

语法如下：

---

```
FUNCTION EDIT_DISTANCE(
    S1 IN VARCHAR,
    S2 IN VARCHAR
)RETURN INTEGER;
```

---

#### 参数详解

- S1 输入参数，VARCHAR 数据。
- S2 输入参数，VARCHAR 数据。

#### 2. edit\_distance\_similarity

输出两个字符串的相似度。通过字符串 s1 转化为 s2 所需要被插入、删除或替换字符的个数，计算出字符串 s1 和 s2 的相似度。返回 1~100 之间的整数，0 代表完全不匹配，100 代表完全匹配。

语法如下：

---

```
FUNCTION EDIT_DISTANCE_SIMILARITY (
    S1 IN VARCHAR,
    S2 IN VARCHAR
)RETURN INTEGER;
```

---

#### 参数详解

- S1 输入参数，VARCHAR 数据。
- S2 输入参数，VARCHAR 数据。

### 29.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_MATCH');
```

例 UTL\_MATCH 包的应用。

例1 EDIT\_DISTANCE

```
SELECT UTL_MATCH.EDIT_DISTANCE('dameng','meng') FROM DUAL;
```

查询结果:

```
2
```

例2 EDIT\_DISTANCE\_SIMILARITY

```
SELECT UTL_MATCH.EDIT_DISTANCE_SIMILARITY('china','chin') FROM DUAL;
```

查询结果:

```
80
```





## 30 UTL\_RAW 包

为了在 DM 上兼容 oracle 的 utl\_raw 包,提供功能上与 oracle 基本一致的 utl\_raw 包。利用这个包提供将十六进制类型转化为其它数据类型的函数和一系列十六进制类型相关函数。

### 30.1 相关方法

utl\_raw 包所支持的 20 个函数,分别实现十六进制的相关转化函数。这些函数通过调用相应系统内部函数来实现存取数据的过程。如下详细介绍各函数:

#### 1. X RANGE

输出包含 start~end 在内的,十六进制单字节区间内的全部字节。

语法如下:

---

```
FUNCTION X RANGE(  
    START_BYTE IN VARBINARY DEFAULT NULL,  
    END_BYTE   IN VARBINARY DEFAULT NULL  
) RETURN VARBINARY;
```

---

#### 参数详解

- START 输入参数,单字节 VARBINARY 数据。
- END 输入参数,单字节 VARBINARY 数据。

注意: 1) 要求 start 和 end 都是单字节。若超过了一个字节,则只从左取第一个高位字节; 2) 如果 start 值>end 值,则返回以 start 十六进制字节开始的,通过 FF 至 00,再到 end 的全部十六进制字节; 3) 如果 start 为 null 默认为 00, end 为 null 默认为 FF,二者均为 null 则 start 默认为 00, end 默认为 FF。

#### 2. BIT\_AND

输出 r1 与 r2 的按位运算的逻辑与。

语法如下:

---

```
FUNCTION BIT_AND(  
    R1 IN VARBINARY,  
    R2 IN VARBINARY  
) RETURN VARBINARY;
```

---

#### 参数详解

- R1 输入参数,VARBINARY 数据。
- R2 输入参数,VARBINARY 数据。

注意: 如果 r1、r2 长度不等,则对长度较小的参数低位补 0xFF 字节至与另一个参数

对齐后再做按位运算。

### 3. BIT\_COMPLEMENT

输出 *r* 的按位运算的反码。

语法如下：

---

```
FUNCTION BIT_COMPLEMENT(  
    R IN VARBINARY  
) RETURN VARBINARY;
```

---

#### 参数详解

- *R* 输入参数，varbinary 数据。

### 4. BIT\_OR

输出 *r1* 与 *r2* 的按位运算的逻辑或。

语法如下：

---

```
FUNCTION BIT_OR(  
    R1 IN VARBINARY,  
    R2 IN VARBINARY  
) RETURN VARBINARY;
```

---

#### 参数详解

- *R1* 输入参数，VARBINARY 数据。
- *R2* 输入参数，VARBINARY 数据。

注意：如果 *r1*、*r2* 长度不等，则对长度较小的参数低位补 0x00 字节至与另一个参数对齐后再做按位运算。

### 5. BIT\_XOR

输出 *r1* 与 *r2* 的按位运算的逻辑异或。

语法如下：

---

```
FUNCTION BIT_XOR(  
    R1 IN VARBINARY,  
    R2 IN VARBINARY  
) RETURN VARBINARY;
```

---

#### 参数详解

- *R1* 输入参数，VARBINARY 数据。
- *R2* 输入参数，VARBINARY 数据。

注意：如果 *r1*、*r2* 长度不等，则对长度较小的参数低位补 0x00 字节至与另一个参数对齐后再做按位运算。

### 6. COMPARE

比较十六进制串数据 *r1* 和 *r2*。如果 *r1* 串长度 < *r2* 串长度，*r1* 串用 pad 填充至二者

长度相等后再作比较。返回结果：数字，表示从几个数字开始不相等。0 表示相等。

语法如下：

---

```
FUNCTION COMPARE(
    R1 IN VARBINARY,
    R2 IN VARBINARY,
    PAD IN VARBINARY,
    LSCHEMA_NAME VARCHAR,
    LTABLE_NAME VARCHAR,
    LCOL_NAME VARCHAR,
    RSCHEMA_NAME VARCHAR,
    RTABLE_NAME VARCHAR,
    RCOL_NAME VARCHAR
) RETURN NUMBER;
```

---

#### 参数详解

- R1 输入参数，VARBINARY 数据。
- R2 输入参数，VARBINARY 数据。
- PAD 输入参数，VARBINARY 数据。
- LSCHEMA\_NAME 左数据源串所在模式名。
- LTABLE\_NAME 左数据源串所在表名。
- LCOL\_NAME 左数据源串所在列名。
- RSCHEMA\_NAME 右数据源串所在模式名。
- RTABLE\_NAME 右数据源串所在表名。
- RCOL\_NAME 右数据源串所在列名。

#### 7. CONCAT

按参数顺序分别连接各个十六进制串数。最多可连接 12 个。返回连接后的十六进制串数值。

语法如下：

---

```
FUNCTION CONCAT(
    R1 IN VARBINARY DEFAULT NULL,
    R2 IN VARBINARY DEFAULT NULL,
    R3 IN VARBINARY DEFAULT NULL,
    R4 IN VARBINARY DEFAULT NULL,
    R5 IN VARBINARY DEFAULT NULL,
    R6 IN VARBINARY DEFAULT NULL,
    R7 IN VARBINARY DEFAULT NULL,
    R8 IN VARBINARY DEFAULT NULL,
    R9 IN VARBINARY DEFAULT NULL,
    R10 IN VARBINARY DEFAULT NULL,
    R11 IN VARBINARY DEFAULT NULL,
```

---

---

```
R12 IN VARBINARY DEFAULT NULL
) RETURN VARBINARY;
```

---

#### 8. CONVERT\_RAW

将十六进制串数值从后指定的字符集到前指定的字符集转换。

语法如下:

---

```
FUNCTION CONVERT_RAW (
    R          IN VARBINARY,
    TO_CHARSET IN VARCHAR2,
    FROM_CHARSET IN VARCHAR2
) RETURN VARBINARY;
```

---

#### 参数详解

支持 utf8 与其它字符集(GBK、BIG5、ISO\_8859\_9、EUC\_JP、EUC\_KR、KOI8R、GB18030、SQL\_ASCII、ISO\_8859\_1)的相互转换。

#### 9. COPIES

将十六进制串数值拷贝 n 次, 依次排在前一个数的后面。返回新的十六进制数值。

语法如下:

---

```
FUNCTION COPIES(
    R IN VARBINARY,
    N IN NUMBER
) RETURN VARBINARY;
```

---

#### 10. LENGTH

返回十六进制串的字节长度。

语法如下:

---

```
FUNCTION LENGTH(
    R IN VARBINARY
) RETURN NUMBER;
```

---

#### 11. overlay\_raw

用十六进制串 overlay\_str 覆盖源十六进制串 target 中指定的十六进制子串。

语法如下:

---

```
FUNCTION OVERLAY_RAW(
    OVERLAY_STR IN VARBINARY,
    TARGET      IN VARBINARY,
    POS         IN INTEGER DEFAULT 1,
    LEN         IN INTEGER DEFAULT NULL,
    PAD         IN VARBINARY  DEFAULT NULL
) RETURN VARBINARY;
```

---

#### 参数详解

- OVERLAY\_STR 十六进制串。

- TARGET 源十六进制串。
- POS 子串在源串 TARGET 中的起始位置。
- LEN 子串的长度。
- PAD

OVERLAY\_STR 串长度若小于 LEN 或 POS 超过了 TARGET 的长度，用 PAD 值代替。

## 12. REVERSE\_RAW

将输入十六进制串的字符顺序反转后返回。

语法如下：

---

```
FUNCTION REVERSE_RAW (  
    R IN VARBINARY  
) RETURN VARBINARY;
```

---

## 13. SUBSTR

返回十六进制串中从字节位置 POS 开始的 LEN 个字节。

语法如下：

---

```
FUNCTION SUBSTR(  
    R    IN VARBINARY,  
    POS IN INTEGER,  
    LEN IN INTEGER DEFAULT NULL  
) RETURN VARBINARY;
```

---

## 14. TRANSLATE

十六进制替换函数。将 R 字符串中的 from\_set 串，全部替换成 to\_set 串。

语法如下：

---

```
FUNCTION TRANSLATE(  
    R          IN VARBINARY,  
    FROM_SET IN VARBINARY,  
    TO_SET    IN VARBINARY  
) RETURN VARBINARY;
```

---

### 参数详解

- R 源字符串。
- FROM\_SET 被替换的字符串。
- TO\_SET 替换后的字符串。

## 15. TRANSLITERATE

功能和 TRANSLATE 一样。

语法如下：

---

```
FUNCTION TRANSLITERATE(  
    R          IN VARBINARY,  
    TO_SET    IN VARBINARY DEFAULT NULL,
```

---

---

```
FROM_SET IN VARBINARY DEFAULT NULL,
PAD      IN VARBINARY DEFAULT NULL
) RETURN VARBINARY;
```

---

### 参数详解

- PAD

如果 TO\_SET 小于 FROM\_SET，少的部分用 PAD 值代替。如果 PAD 为 NULL，则转换后的结果为 NULL。

#### 16. CAST\_TO\_RAW

ASCII 字符转化成十六进制数字。

语法如下：

---

```
FUNCTION CAST_TO_RAW(
    C IN VARCHAR2
) RETURN VARBINARY;
```

---

#### 17. CAST\_TO\_VARCHAR2

十六进制数字转化成对应的 ASCII 字符。

语法如下：

---

```
FUNCTION CAST_TO_VARCHAR2(
    R IN VARBINARY
) RETURN VARCHAR2;
```

---

#### 18. CAST\_TO\_INT

十六进制数字转化成整型十进制。

语法如下：

---

```
FUNCTION CAST_TO_INT(
    R IN VARBINARY,
    ENDIANESS IN INTEGER DEFAULT 1
) RETURN INTEGER;
```

---

### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示依次从高位向低位读取。2 和 3 表示依次从低位向高位读取。

#### 19. CAST\_FROM\_INT

整型十进制转化成十六进制数字。

语法如下：

---

```
FUNCTION CAST_FROM_INT(
    N IN INTEGER,
    ENDIANESS IN INTEGER DEFAULT 1
) RETURN VARBINARY;
```

---

## 20. CAST\_TO\_BINARY\_INTEGER

十六进制数字转化成整型十进制。

语法如下：

---

```
FUNCTION CAST_TO_BINARY_INTEGER (  
    R IN VARBINARY,  
    ENDIANESS IN INTEGER DEFAULT 1  
)RETURN INTEGER;
```

---

### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

## 21. CAST\_FROM\_BINARY\_INTEGER

整型十进制转化成十六进制数字。

语法如下：

---

```
FUNCTION CAST_FROM_BINARY_INTEGER (  
    N IN INTEGER,  
    ENDIANESS IN INTEGER DEFAULT 1  
) RETURN VARBINARY;
```

---

### 参数详解

- ENDIANESS:

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

## 22. CAST\_TO\_BINARY\_DOUBLE

十六进制数字转化成 DOUBLE。

语法如下：

---

```
FUNCTION CAST_TO_BINARY_DOUBLE (  
    R IN VARBINARY,  
    ENDIANESS IN INTEGER DEFAULT 1  
)RETURN DOUBLE;
```

---

### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

## 23. CAST\_FROM\_BINARY\_DOUBLE

DOUBLE 转化成十六进制数字。

语法如下：

---

```
FUNCTION CAST_BINARY_DOUBLE (
    N IN DOUBLE,
    ENDIANESS IN INTEGER DEFAULT 1
) RETURN VARBINARY;
```

---

#### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

#### 24. CAST\_TO\_BINARY\_FLOAT

十六进制数字转化成 FLOAT。

语法如下：

---

```
FUNCTION CAST_TO_FLOAT(
    R IN VARBINARY,
    ENDIANESS IN INTEGER DEFAULT 1
)RETURN FLOAT;
```

---

#### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

#### 25. CAST\_FROM\_BINARY\_FLOAT

FLOAT 转化成十六进制数字。

语法如下：

---

```
FUNCTION CAST_FROM_BINARY_FLOAT(
    N IN DOUBLE,
    ENDIANESS IN INTEGER DEFAULT 1
) RETURN VARBINARY;
```

---

#### 参数详解

- ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

#### 26. CAST\_TO\_NUMBER

十六进制数字转化成 NUMBER。

语法如下：

---

```
FUNCTION CAST_TO_NUMBER(
    R IN VARBINARY,
    ENDIANESS IN INTEGER DEFAULT 1
```

---



---

```
)RETURN NUMBER;
```

---

### 参数详解

#### ● ENDIANESS

读取 R 数据的顺序，取值范围：1、2、3，默认为 1。1 表示使用大端方式。2 表示使用小端方式，3 表示使用机器的大小端方式。

#### 27. CAST\_FROM\_NUMBER

将 number 类型（dec、decimal）转化成十六进制数据。

语法如下：

---

```
FUNCTION CAST_FROM_NUMBER(  
    N IN NUMBER  
) RETURN VARBINARY;
```

---

## 30.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'UTL_RAW');
```

例 UTL\_FILE 包的应用。

#### 例1 xrange

```
select utl_raw.xrange(utl_raw.cast_to_raw('A'),utl_raw.cast_to_raw('Z')) from  
dual;
```

查询结果：

```
0x4142434445464748494A4B4C4D4E4F505152535455565758595A
```

#### 例2 bit\_and

```
select utl_raw.bit_and('0102F3','F30201') from dual;
```

查询结果：

```
0x010201
```

#### 例3 bit\_complement

```
select utl_raw.bit_complement('0102F3') from dual;
```

查询结果：

```
0xFEFD0C
```

#### 例4 bit\_or

```
select utl_raw.bit_or('0102F3','F30201') from dual;
```

查询结果：

```
0XF302F3
```

#### 例5 bit\_xor

```
SELECT utl_raw.bit_xor('0102F3','F30201') from dual;
```

查询结果：

```
0XF200F2
```

#### 例6 compare

```
select utl_raw.compare(utl_raw.cast_to_raw('ABC'),utl_raw.cast_to_raw('ABCD'))
from dual;
```

查询结果:

```
4
```

#### 例7 concat

```
select utl_raw.concat('A','41','B','42') from dual;
```

查询结果:

```
0x0A410B42
```

#### 例8 convert\_raw

```
select UTL_RAW.CONVERT_raw('E4B8ADE69687', 'GBK', 'UTF8') from dual;
```

查询结果:

```
0xD6D0CEC4
```

```
select UTL_RAW.CONVERT_raw('D6D0CEC4','UTF8', 'GBK') from dual;
```

查询结果:

```
0xE4B8ADE69687
```

#### 例9 copies

```
select utl_raw.copies('A',6) from dual;
```

查询结果:

```
0x0A0A0A0A0A0A
```

#### 例10 length

```
select utl_raw.Length(UTL_RAW.CAST_TO_RAW('ABCD')) from dual;
```

查询结果:

```
4
```

#### 例11 overlay\_raw

```
select utl_raw.overlay_raw
(utl_raw.cast_to_raw('D'),utl_raw.cast_to_raw('AAAC'),1,1) from dual;
```

查询结果:

```
0x44414143
```

```
select
utl_raw.overlay_raw(utl_raw.cast_to_raw('D'),utl_raw.cast_to_raw('AAAC'),1,2,
utl_raw.cast_to_raw(' ')) from dual;
```

查询结果:

```
0X44204143
```

#### 例12 reverse\_raw

```
select utl_raw.reverse_raw(utl_raw.cast_to_raw('123')) from dual;
```

查询结果:

```
0x333231
```

例13 substr

```
select utl_raw.substr(0x414243444541424344454142434445,6,2) from dual;
```

查询结果:

```
0x4142
```

例14 translate

```
select
UTL_RAW.translate(utl_raw.cast_to_raw('aDe'),utl_raw.cast_to_raw('AD'),utl_ra
w.cast_to_raw('ad')) from dual;
```

查询结果:

```
616465 --即 ade
```

例15 translITERATE

```
select
UTL_RAW.translITERATE(utl_raw.cast_to_raw('FaDe'),utl_raw.cast_to_raw('aDef')
,utl_raw.cast_to_raw('ADEF'),utl_raw.cast_to_raw(' ')) from dual;
```

查询结果:

```
66616465 --即 fade
```

例16 CAST\_TO\_RAW

```
select UTL_RAW.CAST_TO_RAW(10) from dual;
```

查询结果:

```
0x3130
```

例17 cast\_to\_varchar2

```
select utl_raw.cast_to_varchar2('40') from dual;
```

查询结果:

```
@
```

例18 cast\_to\_INT

```
select utl_raw.cast_to_int('00000064',1) from dual;
```

查询结果:

```
100
```

```
select utl_raw.cast_to_int('00000064',2) from dual;
```

查询结果:

```
1677721600
```

例19 cast\_from\_INT

```
select utl_raw.cast_from_int(10) from dual;
```

查询结果:

```
0x0000000A
```

```
select utl_raw.cast_FROM_int(94311,2) from dual;
```

查询结果:

```
0x67700100
```

例20 cast\_from\_number

```
select UTL_RAW.cast_FROM_number(10) from dual;
```

查询结果:

```
0xC10B
```

## 31 UTL\_TCP 包

许多的应用程序都是基于 TCP/IP 协议的，UTL\_TCP 包的功能是提供一个可以与外部的 TCP/IP 服务器通讯，并能够进行基本的收发数据的功能。

### 31.1 相关方法

#### 1. AVAILABLE

当前能够读取的连接字符数(最大不超过 96 字节)。

语法如下：

---

```
FUNCTION AVAILABLE(
    C IN OUT CONNECTION,
    TIMEOUT IN INT DEFAULT 0
)RETURN INT;
```

---

#### 参数详解

- C 一个 OPEN\_CONNECTION 打开的、有效的 CONNECTION。
- TIMEOUT 如果没有数据，则在制定的 TIMEOUT 时间返回。

#### 返回值

当前获取的一个 TCP/IP 连接字节数。

#### 2. CLOSE\_ALL\_CONNECTIONS

关闭所有的连接。

语法如下：

---

```
UTL_TCP.CLOSE_ALL_CONNECTIONS;
```

---

#### 3. CLOSE\_CONNECTION

关闭指定的连接。

语法如下：

---

```
UTL_TCP.CLOSE_CONNECTION (
    C IN OUT CONNECTION
);
```

---

#### 4. GET\_RAW

读取指定长度（二进制的数据）的数据。

语法如下：

---

```
FUNCTION GET_RAW (
    C IN OUT CONNECTION,
    LEN IN INT DEFAULT 1,
    PEEK IN BOOLEAN DEFAULT FALSE
)RETURN VARBINARY;
```

---

### 参数详解

- C 一个 OPEN\_CONNECTION 打开的、有效的 CONNECTION。
- LEN 要收到的数据的长度，默认值 1。
- PEEK

该选项为 TRUE，则指只是查看队列的数据，不会从队列中删除，下次还能够读到；为 FALSE，则会删除队列中的数据。默认参数，FALSE。

### 返回值

读取到的二进制数据。

#### 5. OPEN\_CONNECTION

打开一个连接到外部服务器的连接。

语法如下：

---

```
FUNCTION OPEN_CONNECTION(  
    REMOTE_HOST IN VARCHAR2,  
    REMOTE_PORT IN INT,  
    LOCAL_HOST IN VARCHAR2 DEFAULT NULL,  
    LOCAL_PORT IN INT DEFAULT NULL,  
    TX_TIMEOUT IN INT DEFAULT NULL  
) RETURN CONNECTION;
```

---

### 参数详解

- REMOTE\_HOST 远程主库的名字，或者 IP 地址。
- REMOTE\_PORT 远程主库的端口。
- LOCAL\_HOST 本地主库的名字（一般不用设置）。
- LOCAL\_PORT 本地主库的端口（一般不用设置）。
- TX\_TIMEOUT 尝试链接到远程主库的等待时间。

### 返回值

一个连接到指定外部服务器的链接。

#### 6. READ\_RAW

以二进制形式读取一行数据。

语法如下：

---

```
FUNCTION READ_RAW (  
    C IN OUT CONNECTION,  
    DATA OUT VARBINARY,  
    LEN IN INT DEFAULT 1,  
    PEEK IN BOOLEAN DEFAULT FALSE  
) RETURN INT;
```

---

### 参数详解

- C 一个 OPEN\_CONNECTION 打开的、有效的 CONNECTION。

- DATA 用于存放读到的数据。
- LEN 要收到的数据的长度，默认值 1。
- PEEK

该选项为 TRUE，则指只是查看队列的数据，不会从队列中删除，下次还能够读到；为 FALSE，则会删除队列中的数据。默认参数，FALSE。

### 返回值

收到的数据的字节长度。

### 7. WRITE\_RAW

发送一行二进制数据。

语法如下：

---

```
FUNCTION WRITE_RAW(
    C IN OUT CONNECTION,
    DATA IN VARBINARY,
    LEN IN INT DEFAULT NULL
) RETURN INT;
```

---

### 参数详解

- C 一个 OPEN\_CONNECTION 打开的、有效的 CONNECTION。
- DATA 存放要发送的数据。
- LEN 要发送数据的长度。

### 返回值

发送的二进制数据的字节长度。只有成功发送才会有返回值。

## 31.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_TCP');
```

例 UTL\_TCP 包示例。

```
DECLARE
    HOST VARCHAR(128);
    PORT INT;
    RET    INT;
    LEN    INT;
    DATA VARBINARY(512);
    SHOW   VARBINARY(128);
    C UTL_TCP.CONNECTION;
BEGIN
    HOST := '192.168.0.212';
    PORT := 25;
    C := UTL_TCP.OPEN_CONNECTION(HOST, PORT);
```

```
RET := UTL_TCP.WRITE_RAW(C, RAWTOHEX('EHLO WXH@DAMENG.SHANGHAI'),50);  
PRINT RET;  
LEN := 48;  
RET := UTL_TCP.AVAILABLE(C,0);  
PRINT RET;  
SHOW := UTL_TCP.GET_RAW(C,LEN);  
PRINT SHOW;    -- ' 220 192.168.0.212 ESMTP '的二进制流  
RET := UTL_TCP.READ_RAW(C,DATA,LEN);  
UTL_TCP.CLOSE_CONNECTION(C);  
END;  
/
```

结果如下:

```
24  
25  
323230203139322E3136382E302E3231322045534D54500D0A
```



## 32 UTL\_URL 包

为了兼容 Oracle 的 UTL\_URL 系统包，对 URL 进行转码与解码操作。URL 是一个指向网页、图片等网络资源的字符串。访问网络资源使用的是 HTTP 协议。例如，DM 网站的 URL 为：<http://www.dameng.com>。通常，一个正确的 URL 字符串包含字母、数字、标点符号三种规范字符。这些字符都是非保留字。如果 URL 中含有非规范字符，那么就需要转码；如果 URL 中含有保留字符，则根据用户需要决定是否转码。

URL 中可以使用的规范字符和保留字符。如下所示：

■ 规范字符包括：

[A-Z]、[a-z]、[0-9]；

减号 (-)、下划线 (\_)、句点 (.)、感叹号 (!)、波浪号 (~)、星号 (\*)、重音符号(')、左圆括号(())、右圆括号 ( ) )。

■ 保留字符包括：

分号 (;)、斜线 (/)、问号 (?)、冒号 (:)、(@)、地址符 (&)、等号 (=)、加号 (+)、美元符号 (\$)、逗号 (,)。

### 32.1 相关方法

#### 1. UTL\_URL.ESCAPE

对 URL 中的非法字符（及保留字符）进行转码。

语法如下：

---

```
UTL_URL.ESCAPE (
    URL                IN VARCHAR,
    ESCAPE_RESERVED_CHARS IN BOOLEAN DEFAULT FALSE,
    URL_CHARSET        IN VARCHAR2 DEFAULT 'GBK')
RETURN VARCHAR2;
```

---

#### 参数详解

- URL 待转码的 URL。
- ESCAPE\_RESERVED\_CHARS 指明是否转义 URL 的保留字符。
- URL\_CHARSET

指定 URL 的转码格式使用的字符集。字符集包含：UTF8、GBK、BIG5、ISO\_8859\_9、EUC\_JP、EUC\_KR、KOI8R、ISO\_8859\_1、SQL\_ASCII、GB18030、ISO\_8859\_11。

#### 返回值

返回转码后的 URL。

**使用说明：**

此存储函数转义基于 RFC2396 标准的 URL 中包含的非规范字符。

**2. UTL\_URL.UNESCAPE**

对 URL 进行解码。

语法如下：

---

```
UTL_URL.UNESCAPE (
    URL          IN VARCHAR,
    URL_CHARSET  IN VARCHAR2 DEFAULT 'GBK'
)RETURN VARCHAR2;
```

---

**参数详解**

- URL 待解码 URL。
- URL\_CHARSET

待解码 URL 的编码格式使用的字符集。字符集包含：UTF8、GBK、BIG5、ISO\_8859\_9、EUC\_JP、EUC\_KR、KOI8R、ISO\_8859\_1、SQL\_ASCII、GB18030、ISO\_8859\_11。

**返回值**

返回解码后的 URL。

## 32.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'UTL_URL');
```

第一步，对 URL (<http://www.dameng.com/a url with space.html>) 使用 UTF8 字符集进行转码：

```
SELECT UTL_URL.ESCAPE ('http://www.dameng.com/a url with space.html', FALSE,
'UTF8') FROM DUAL;
```

查询结果：

```
http://www.dameng.com/a%20url%20with%20space.html
```

第二步，对上面得到的 URL 进行解码：

```
SELECT UTL_URL.UNESCAPE ('http://www.dameng.com/a%20url%20with%20space.html',
'UTF8');
```

查询结果：

```
http://www.dameng.com/a url with space.html
```

比较可以发现，经过转码再解码后的 URL 与原来相同。

## 33 DBMS\_SCHEDULER 包

DBMS\_SCHEDULER 包提供一系列调度相关的存储过程及方法供 PL/SQL 调用，目前主要兼容 ORACLE 的 DBMS\_SCHEDULER 包中几个常用的方法。

### 33.1 相关方法

#### 33.1.1 SCHEDULE 对象

##### 1. CREATE\_SCHEDULE 过程

创建一个调度。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.CREATE_SCHEDULE(
    SCHEDULE_NAME      IN VARCHAR,
    START_DATE          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    REPEAT_INTERVAL     IN VARCHAR,
    END_DATE            IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    COMMENTS            IN VARCHAR DEFAULT NULL);
```

---

##### 参数详解

- SCHEDULE\_NAME  
调度名称。要求在包模式下名称唯一。不能为空，若为设置，则报错。
- START\_DATE  
调度第一次有效日期或者时间。对于重复的调度，START\_DATE 是个参照值，这种情况 START\_DATE 不是调度的开始时间，调用的开始决定于 REPEATE\_INTERVAL 的设置。START\_DATE 用于决定调度的第一个实例。
- REPEATE\_INTERVAL  
调度重复间隔，用于指定调用隔多久重复一次。它是基于日历语法的表达式。详见 [附录：日历语法](#)。对于命名的调度，REPEATE\_INTERVAL 不支持 PL/SQL 表达式。
- END\_DATE  
作业停止运行时间。如果为未设置，作业则一直有效。
- COMMENTS  
调度相关的评论或注释。默认为空。

##### 2. DROP\_SCHEDULE 过程

删除一个调度。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.DROP_SCHEDULE(
    SCHEDULE_NAME      IN VARCHAR,
    FORCE                IN BOOLEAN DEFAULT FALSE);
```

---

#### 参数详解

- SCHEDULE\_NAME

调度名称。仅支持单独一个调度名称。

- FORCE

设为 FALSE，说明调度删除前不能被任何作业引用，否则报错。

设为 TRUE，所有引用这个调度的作业在调度删除之前都将调为无效。

使用该调度，但正在运行的作业不受影响。

如果为 NULL，则取值为 FALSE。

## 33.1.2 PROGRAM 对象

### 1. CREATE\_PROGRAM 过程

创建一个程序。通过 CREATE\_PROGRAM 方法创建的 PROGRAM 对象，在视图 DBA\_SCHEDULER\_PROGRAMS 或 USER\_SCHEDULER\_PROGRAMS 中可以查看到。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.CREATE_PROGRAM(
    PROGRAM_NAME      IN VARCHAR,
    PROGRAM_TYPE      IN VARCHAR,
    PROGRAM_ACTION     IN VARCHAR,
    NUMBER_OF_ARGUMENTS IN INT DEFAULT 0,
    ENABLED            IN BOOLEAN DEFAULT FALSE,
    COMMENTS          IN VARCHAR DEFAULT NULL);
```

---

#### 参数详解

- PROGRAM\_NAME

程序名称。要求在包模式下名称中名称唯一。若未设置，则报错。

- PROGRAM\_TYPE

程序类型。未设置，则报错。支持类型如下：

PLSQL\_BLOCK：说明程序是一个 PL/SQL 语句块，即 DM 的 SQL 程序设计里的语句块。

此类型的程序或者作业不允许设置参数个数，也就是说设置的参数个数必须为 0；

STORED\_PROCEURE：说明程序是一个 PL/SQL 存储过程。支持存储过程，不支持带有

返回值的函数。包含输入输出或者输出参数的 PL/SQL 存储过程不支持。

- PROGRAM\_ACTION

定义程序的动作。可能的动作如下：

对于 PL/SQL 语句块，程序动作就是去执行 PL/SQL 代码。

对于存储过程，程序动作是存储过程的名称。如果存储过程与作业不属于同一个模式，则需要指定存储过程名称的时候，指定模式名。

若未指定程序动作，则报错。

- NUMBER\_OF\_ARGUMENTS

定义程序包含的参数的个数。若未设置，则默认为 0。一个程序最多可以指定 255 个数。

- ENABLED

指定程序是否以激活的方式创建。若设置为 TRUE，则执行合法性检测，检测成功，则创建程序并置 ENABLED 状态。默认设置为 FALSE，那么程序不以激活的方式创建，可以在程序使用之前通过调用 ENABLE 过程来激活。

如果为 NULL，则取值为 FALSE。

- COMMENTS

程序的相关评论或者注释。默认为空。

## 2. DEFINE\_PROGRAM\_ARGUMENT 过程

该过程为重载过程。通过 DEFINE\_PROGRAM\_ARGUMENT 方法定义的 PROGRAM 参数信息，可以在 DBA\_SCHEDULER\_PROGRAM\_ARGS 或 USER\_SCHEDULER\_PROGRAM\_ARGS 视图中查看得到。

1) 定义一个程序参数，不带默认值。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
    PROGRAM_NAME          IN VARCHAR,
    ARGUMENT_POSITION     IN INT,
    ARGUMENT_NAME         IN VARCHAR DEFAULT NULL,
    ARGUMENT_TYPE         IN VARCHAR,
    OUT_ARGUMENT          IN BOOLEAN DEFAULT FALSE
);
```

---

2) 定义一个程序参数，带默认值。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT(
    PROGRAM_NAME          IN VARCHAR,
    ARGUMENT_POSITION     IN INT,
    ARGUMENT_NAME         IN VARCHAR DEFAULT NULL,
    ARGUMENT_TYPE         IN VARCHAR,
    DEFAULT_VALUE         IN VARCHAR,
    OUT_ARGUMENT          IN BOOLEAN DEFAULT FALSE
);
```

---

### 参数详解

- PROGRAM\_NAME  
待定义参数的程序名。该程序必须已经存在。
- ARGUMENT\_POSITION  
待定义参数的位置，可选值，从 1 开始到 NUMBER\_OF\_ARGUMENTS。若指定位置已经定义过，则将被覆盖。
- ARGUMENT\_NAME  
定义参数名称，可选项。若设置，则要确保在程序所有参数名中唯一。若设置，则可能被其他包过程使用，包括 SET\_JOB\_ARGUMENT\_VALUE 过程。
- ARGUMENT\_TYPE  
定义参数使用数据的类型。调度不证实也不使用这个类型。只有在程序用户决定赋值给参数时才使用。所有可用的 SQL 数据类型均允许，不支持用户自定义类型。
- DEFAULT\_VALUE  
参数默认值。若作业未给参数设值，则将默认值赋值给对应参数。
- OUT\_ARGUMENT  
预留参数，以后使用。必须设置为 FALSE。如果为 NULL，则取值为 FALSE。

### 3. DROP\_PROGRAM 过程

删除一个程序。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.DROP_PROGRAM(
    PROGRAM_NAME          IN VARCHAR,
    FORCE                  IN BOOLEAN DEFAULT FALSE
);
```

---

### 参数详解

- PROGRAM\_NAME

待删除的程序名。

- **FORCE**

设为 FALSE，说明程序删除前不能被任何作业引用，否则报错。

设为 TRUE，所有引用这个程序的作业在调度删除之前都将调为无效。

使用该程序，且正在运行的作业不受影响，并允许继续运行。

如果为 NULL，则取值为 FALSE。

### 33.1.3 JOB 对象

#### 1. CREATE\_JOB 过程

创建作业。通过 CREATE\_JOB 创建的 JOB 对象，可以在 DBA\_SCHEDULER\_JOBS 或 USER\_SCHEDULER\_JOBS 视图中查看得到。

1) 不使用已经存在的程序（PROGRAM）或者调度（SCHEDULE）创建作业。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.CREATE_JOB(
    JOB_NAME          IN VARCHAR,
    JOB_TYPE          IN VARCHAR,
    JOB_ACTION        IN VARCHAR,
    NUMBER_OF_ARGUMENTS IN INT DEFAULT 0,
    START_DATE        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    REPEAT_INTERVAL   IN VARCHAR DEFAULT NULL,
    END_DATE          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    JOB_CLASS         IN VARCHAR DEFAULT 'DEFAULT_JOB_CLASS',
    ENABLED           IN BOOLEAN DEFAULT FALSE,
    AUTO_DROP         IN BOOLEAN DEFAULT TRUE,
    COMMENTS          IN VARCHAR DEFAULT NULL,
    CREDENTIAL_NAME   IN VARCHAR DEFAULT NULL,
    DESTINATION_NAME  IN VARCHAR DEFAULT NULL);
```

---

2) 使用命名程序（PROGRAM）和命名调度（SCHEDULE）创建作业。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.CREATE_JOB(
    JOB_NAME          IN VARCHAR,
    PROGRAM_NAME      IN VARCHAR,
    SCHEDULE_NAME     IN VARCHAR,
    JOB_CLASS         IN VARCHAR DEFAULT 'DEFAULT_JOB_CLASS',
```

---

---

ENABLED	IN BOOLEAN DEFAULT FALSE,
AUTO_DROP	IN BOOLEAN DEFAULT TRUE,
COMMENTS	IN VARCHAR DEFAULT NULL,
JOB_STYLE	IN VARCHAR DEFAULT 'REGULAR',
CREDENTIAL_NAME	IN VARCHAR DEFAULT NULL,
DESTINATION_NAME	IN VARCHAR DEFAULT NULL);

---

3) 使用命名程序 (PROGRAM) 和内置调度 (SCHEDULE) 创建作业。

语法如下:

---

```
PROCEDURE DBMS_SCHEDULER.CREATE_JOB(
    JOB_NAME          IN VARCHAR,
    PROGRAM_NAME      IN VARCHAR,
    START_DATE        IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    REPEAT_INTERVAL   IN VARCHAR DEFAULT NULL,
    END_DATE          IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
    JOB_CLASS         IN VARCHAR DEFAULT 'DEFAULT_JOB_CLASS',
    ENABLED           IN BOOLEAN DEFAULT FALSE,
    AUTO_DROP         IN BOOLEAN DEFAULT TRUE,
    COMMENTS          IN VARCHAR DEFAULT NULL,
    JOB_STYLE         IN VARCHAR DEFAULT 'REGULAR',
    CREDENTIAL_NAME   IN VARCHAR DEFAULT NULL,
    DESTINATION_NAME  IN VARCHAR DEFAULT NULL);
```

---

### 参数详解

- JOB\_NAME

作业名称。要求在包模式下名称中名称唯一,再其他某事下使用时,必须加模式名前缀。

若未设置,则报错。使用 GENERATE\_JOB\_NAME 过程可以自动生成一个作业名。详见 [GENERATE\\_JOB\\_NAME 过程](#)。

- JOB\_TYPE

创建作业类型。未设置,则报错。支持类型如下:

PLSQL\_BLOCK: 指定作业是一个匿名的 PL/SQL 语句块。此类型的程序或者作业不允许设置参数个数,也就是说设置的参数个数必须为 0;

STORED\_PROCEURE: 指定作业是一个 PL/SQL 存储过程。支持存储过程,不支持带有返回值的函数。

- JOB\_ACTION

定义作业的动作。可能的动作如下:

对于 PL/SQL 语句块,程序动作就是去执行 PL/SQL 代码。



对于存储过程，程序动作是存储过程的名称。如果存储过程与作业不属于同一个模式，则需要指定存储过程名称的时候，指定模式名。

若未指定作业动作，则报错。

- NUMBER\_OF\_ARGUMENTS

定义作业预期的参数个数。若未设置，则默认为 0。一个程序最多可以指定 255 个参数。

- PROGRAM\_NAME

和作业关联的 program 名称。

- SCHEDULE\_NAME

作业使用的调度名称。

- START\_DATE

作业起始日期。如果 START\_DATE 和 REPEAT\_INTERVAL 都为 NULL，表示作业在 ENABLE 后立即执行。

- REPEAT\_INTERVAL

作业重复执行及间隔时长。仅支持 [日历语法格式](#)，详见附录。

- END\_DATE

作业结束日期。

- JOB\_CLASS

不支持，仅作兼容参数用。

- ENABLED

指定作业创建时是否启用。默认为 FALSE。作业如果未启用，则系统不会调度执行。作业创建时设置为 TRUE，会去检查作业是否有效。作业 ENABLE 时，如果作业所使用的 PROGRAM 无效，作业创建失败。

如果为 NULL，则取值为 FALSE。

- AUTO\_DROP

如果为 TRUE，作业自动删除。默认为 TRUE。自动删除的条件如下：

- 作业的结束日期已经过期
- 作业不是重复执行作业，并且只执行一次

如果为 NULL，则取值为 FALSE，作业不会自动删除。

- COMMENTS

程序的相关评论或者注释。默认为空。

- JOB\_STYLE

不支持，只作兼容参数用。

- CREDENTIAL\_NAME

不支持，只作兼容参数用。

- DESTINATION\_NAME

不支持，只作兼容参数用。

## 2. SET\_JOB\_ARGUMENT\_VALUE 过程

重载过程。通过 SET\_JOB\_ARGUMENT\_VALUE 设置的 JOB 参数值，可以在 DBA\_SCHEDULER\_JOB\_ARGS 或 USER\_SCHEDULER\_JOB\_ARGS 视图中查询得到。

### 1) 通过指定位置设置参数值。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE(
    JOB_NAME          IN VARCHAR,
    ARGUMENT_POSITION IN INT,
    ARGUMENT_VALUE    IN VARCHAR
);
```

---

2) 通过指定参数名称，设置相应参数值。仅适用于作业使用了已经存在的程序对象，并且程序参数通过调用 DEFINE\_PROGRAM\_ARGUMENT 过程定义了参数名称。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE(
    JOB_NAME          IN VARCHAR,
    ARGUMENT_NAME     IN VARCHAR,
    ARGUMENT_VALUE    IN VARCHAR
);
```

---

### 参数详解

- JOB\_NAME

待设置参数值的作业名称。

- ARGUMENT\_NAME

待设置参数值的程序参数名称。

- ARGUMENT\_POSITION

待设置参数值的程序参数位置，从 1 开始。

- ARGUMENT\_VALUE

设置到程序参数上的新值，暂仅支持字符串类型的值。

### 3. DROP\_JOB 过程

删除一个作业过程。作业删除时，作业上所有设置参数值也都删除。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.DROP_JOB(
    JOB_NAME          IN VARCHAR,
    FORCE              IN BOOLEAN DEFAULT FALSE,
    DEFER             IN BOOLEAN DEFAULT FALSE,
    COMMIT_SEMANTICS  IN VARCHAR2 DEFAULT 'STOP_ON_FIRST_ERROR'
);
```

---

#### 参数详解

- **JOB\_NAME**  
待删除的作业名。
- **FORCE**  
设为 TRUE，调度首先尝试中止正在运行的作业实例（通过调用 STOP\_JOB 过程，参数 force 设置 FALSE）。如果为 NULL，则取值为 FALSE。
- **DEFER**  
设为 TRUE，调度运行正在运行的作业完成，然后删除。如果为 NULL，则取值为 FALSE。
- **COMMIT\_SEMANTICS**  
提交语义，暂不处理。

### 4. RUN\_JOB 过程

立即执行一次作业。

语法如下：

---

```
PROCEDURE RUN_JOB (JOB_NAME          IN VARCHAR,
                   USE_CURRENT_SESSION IN BOOLEAN DEFAULT TRUE);
```

---

#### 参数详解

- **JOB\_NAME**  
待运行的作业名。
- **USE\_CURRENT\_SESSION**  
作业是否运行在和存储过程被调用的同一个会话中。不支持此功能，仅作兼容。如果为 NULL，则取值为 FALSE。

### 5. STOP\_JOB 过程

停止正在运行的作业。

语法如下：

```
PROCEDURE STOP_JOB (
    JOB_NAME          IN VARCHAR,
    FORCE              IN BOOLEAN DEFAULT FALSE,
    COMMIT_SEMANTICS  IN VARCHAR DEFAULT 'STOP_ON_FIRST_ERROR');
```

#### 参数详解

- JOB\_NAME

待停止的作业名。

- FORCE

FALSE 表示中断作业，DM 会用中断会话的方式停止作业，VM 里面会定期检测会话是否中断；

TRUE 表示立即停止作业，目前不支持这种方式。

如果为 NULL，则取值为 FALSE。

- COMMIT\_SEMANTICS

仅语法支持。

### 33.1.4 全局调度属性

全局调度属性是作用于整个 SCHEDULER 中所有作业上。

表全局属性列表

属性名称	取值	备注
Default_timezone	Start_date 未指定时,用于重复作业和窗口应用日历语法格式串时获取时区信息	不支持, 仅作兼容使用
Event_expiry_time	以秒为单位, 作业从调度事件队列中产生的过期事件; 若为 NULL, 则过期时间为 24 小时。	不支持, 仅作兼容使用
Log_history	作业日志和窗口日志的保存天数, 默认 30 天。	不支持, 仅作兼容使用
Max_job_slave_processes	特殊系统配置和加载的从进程的最大个数。默认是 NULL, 有效值范围为 1-999。	不支持, 仅作兼容使用
Email_server	调度用于发送作业状态事件邮件	支持

	通知使用的 SMTP 服务器的地址。 如果为 NULL，则不能发送。	
Email_port	调度用于发送作业状态事件邮件通知使用的 SMTP 服务器的端口号，默认 25。	扩展支持，DM 特有的选项
Email_password	邮件发送连接 SMTP 服务器的密码。	扩展支持，加密保存。
Email_sender	发送作业状态邮件通知的默认邮件地址，默认为 NULL	支持
Email_mimetype	邮件的 MIMETYPE，设置邮件使用的内容和编码方式，默认 TEXT/PLAIN； CHARSET=US-ASCII； 中文可调整为 TEXT/PLAIN； CHARSET=GB18030	扩展支持，DM 特有的选项

### 1. SET\_SCHEDULER\_ATTRIBUTE 过程

设置全局调度的一个属性。设置属性值会立即有效，但结果不一定立即出现。全局调度属性可以在 DBA\_SCHEDULER\_GLOBAL\_ATTRIBUTE 视图中查询得到。

语法如下：

```
PROCEDURE SET_SCHEDULER_ATTRIBUTE(
    ATTRIBUTE          IN VARCHAR,
    VALUE              IN VARCHAR);
```

#### 参数详解

- ATTRIBUTE

属性名称，详见上表。

- VALUE

属性值，详见上表说明，根据实际情况设置。

### 2. GET\_SCHEDULER\_ATTRIBUTE 过程

获取全局调度的一个属性。

语法如下：

```
PROCEDURE GET_SCHEDULER_ATTRIBUTE(
    ATTRIBUTE          IN VARCHAR,
```

VALUE

OUT VARCHAR);

**参数详解**

## ● ATTRIBUTE

属性名称，详见上表。

## ● VALUE

属性值，详见上表说明，根据实际情况设置。

**33.1.5 邮件通知**

编辑作业事件的邮件通知。

表在邮件中 SUBJECT 和 BODY 参数中可使用的变量

变量名称	释义
%job_owner%	创建的作业的模式
%job_name%	-
%job_subname%	用于基于事件的作业，设置了 parallel_instances 属性或者用于 CHAIN 步骤 (STEPS)
%event_type%	见表 2
%event_timestamp%	事件时间戳
%log_id%	指向视图 *_SCHEULER_JOB_LOG 和 *_SCHEDULER_JOB_RUN_DETAILS 中的 LOG_ID 列。
%error_code%	错误码
%error_message%	错误信息
%run_count%	运行次数
%failure_count%	失败次数
%retry_count%	重试次数

表调度抛出的事件类型

事件类型	释义
Job_all_events	不是一个事件，是一个常量，用于提供一种简单的方法激活所有事件
Job_broken	作业必须已经 disabled 并且调整到 BROKEN 状态，因为执行失败次数超过了作业属性 max_failures 设置值。
Job_chain_stalled	运行 CHAIN 的作业进入 CHAIN_STALLED 状态。A running chain

	becomes stalled if there are no steps running or scheduled to run and the chain evaluation_interval is set to NULL. No progress will be made in the chain unless there is manual intervention.
Job_completed	作业成功完成，达到 max_runs 或者 end_date
Job_disabled	调度本身调整作业 disabled，或者调用 SET_ATTRIBUTE。
Job_failed	作业失败，可以是抛出错误，也可以是异常中断。
Job_over_max_dur	作业执行超过了 max_run_duration 属性设置的最大运行周期。（注意：没必要使用作业 raise_events 属性来激活这个事件；总是激活状态）
Job_run_completed	作业运行或者失败、或者成功或者停止
Job_sch_lim_reached	达到作业调用 limit。作业未开始，因为启动作业延迟超过了作业属性 schedule_limit 定义值。
Job_started	作业启动了。
Job_stopped	调用 STOP_JOB 停止了作业。
Job_succeeded	作业成功完成。

### 1. ADD\_JOB\_EMAIL\_NOTIFICATION 过程

为作业添加一个邮件通知。无论哪个指定的作业状态事件触发，邮件会发送到指定链表中的接收端。

语法如下：

```

PROCEDURE ADD_JOB_EMAIL_NOTIFICATION(
    JOB_NAME      IN VARCHAR,
    RECIPIENTS    IN VARCHAR,
    SENDER        IN VARCHAR DEFAULT NULL,
    SUBJECT       IN VARCHAR DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_SUBJECT,
    BODY          IN VARCHAR DEFAULT DBMS_SCHEDULER.DEFAULT_NOTIFICATION_BODY,
    EVENTS        IN                                VARCHAR                                DEFAULT
'JOB_FAILED,JOB_BROKEN,JOB_SCH_LIM_REACHED,JOB_CHAIN_STALED,JOB_OVER_MAX_DUR'
,
    FILTER_CONDITION IN VARCHAR DEFAULT NULL);

```

### 参数详解

- **JOB\_NAME**  
添加邮件通知的目标作业名称，不能为 NULL。
- **Recipients**

逗号隔开的接收端的 EMAIL 地址链表。列出的所有邮件通知都会发送到所有的接收端地址。不能为 NULL。

- Sender

发送端的 EMAIL 地址。若为 NULL 或者忽略,则使用全局调度属性中的 email\_sender。

- Subject

邮件主题。目前仅支持默认主题:

```
DamengScheduler Job Notification
- %job_owner%.%job_name% %event_type%
```

其中,%中间代表是要插入的变量值。

- Body

邮件内容。目前仅支持默认邮件内容格式:

```
'Job: %job_owner%.%job_name%
Event: %event_type%
Date: %event_timestamp%
Log id: %log_id%
Run count: %run_count%
Failure count: %failure_count%
Retry count: %retry_count%
Error code: %error_code%
Error message:%error_message%'
```

其中,%中间代表是要插入的变量值。

- Events

以逗号隔开的发送邮件通知的作业状态事件。不能为 NULL。列表中任何一个事件触发都会向接收端发送通知。如果忽略,则以下默认事件发送通知:

```
JOB_FAILED、JOB_BROKEN、JOB_SCH_LIM_REACHED、JOB_CHAIN_STALLED、
JOB_OVER_MAX_DUR。
```

- Filter\_condition

用于过程要发送邮件通知的事件。如果为 NULL,所有指定事件发生时都会产生一个邮件通知并发送。Filter\_condition 必须是一个布尔类型 SQL WHERE 表达式,引用:event 绑定变量。绑定变量自动绑定到代表抛出事件的对象上(实现中并未支持,仅作兼容处理)。

例如,过滤条件为:



:event.error\_code=600 或者 :error.error\_code=700

## 2. REMOVE\_JOB\_EMAIL\_NOTIFICATION 过程

移除作业的邮件通知。可以移除所有的邮件通知，或者仅移除指定接收者或指定事件的邮件通知。执行作业删除时，会将作业上所有的邮件通知移除。

语法如下：

---

```
PROCEDURE REMOVE_JOB_EMAIL_NOTIFICATION(
    JOB_NAME IN VARCHAR,
    RECIPIENTS IN VARCHAR DEFAULT NULL,
    EVENTS IN VARCHAR DEFAULT NULL);
```

---

### 参数详解

- JOB\_NAME

移除邮件通知的目标作业。不能为 NULL。

- RECIPIENTS

逗号隔开的要移除满足配置指定接收者的邮件通知。若为 NULL，则移除邮件通知时，忽略接口端的过滤。

- EVENTS

逗号隔开的要移除满足配置直指定事件的邮件通知。若为 NULL，则移除邮件通知时，忽略指定事件的过滤。

## 33.1.6 其他

### 1. GENERATE\_JOB\_NAME 函数

返回一个唯一的作业名称。名称格式为{前缀}N，其中 N 是从一个序列获取的数字。若未指定前缀，则默认产生名字为 JOB\$\_1、JOB\$\_2、JOB\$\_3 等。若设置前缀为 'DM'，则生成 JOB 名称为 DM1、DM2、DM3 等。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.GENERATE_JOB_NAME(
    PREFIX IN VARCHAR DEFAULT 'JOB$_'
)RETURN VARCHAR;
```

---

### 参数详解

- PREFIX

生成作业名的前缀

### 2. EVALUATE\_CALENDAR\_STRING 过程

计算日历字符串。

语法如下：

---

```
PROCEDURE DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (
    CALENDAR_STRING          IN  VARCHAR,
    START_DATE               IN  TIMESTAMP WITH TIME ZONE,
    RETURN_DATE_AFTER        IN  TIMESTAMP WITH TIME ZONE,
    NEXT_RUN_DATE            OUT TIMESTAMP WITH TIME ZONE
);
```

---

### 参数详解

- CALENDAR\_STRING

待计算的日历格式字符串。必须是按照日历语法。

- START\_DATE

日历字符串有效的起始日期和时间。若指定为 NULL，则起始时间为当前时间。

- RETURN\_DATE\_AFTER

使用 START\_DATE 和 CALENDAR\_STRING，调度就有足够的信息来决定所有有效执行日期。设置这个参数，调度知道匹配哪个可能的时间退出。若未设置，则系统直接使用系统时间戳。

- NEXT\_RUN\_DATE

匹配 CALENDAR\_STRING 和 START\_DATE，并在 RETURN\_DATE\_AFTER 参数值之后发生的第一个时间戳。

### 3. SET\_ATTRIBUTE 过程

设置 SCHEDULER 对象的一个属性，目前仅支持 JOB 对象的属性设置。

语法如下：

---

```
PROCEDURE SET_ATTRIBUTE(
    NAME          IN VARCHAR,
    ATTRIBUTE     IN VARCHAR,
    VALUE         IN {BOOLEAN | VARCHAR | TIMESTAMP |
                     TIMESTAMP WITH TIME ZONE |
                     INTERVAL DAY TO SECOND});
```

---

### 参数详解

- NAME

设置的对象名称。目前仅支持对 JOB 对象属性设置。

- ATTRIBUTE

属性名称。

➤ JOB 对象支持的属性有以下属性：

➤ auto\_drop: 作业是否自动删除

如果为 TRUE，作业自动删除。默认为 TRUE。自动删除的条件如下：

- 1) 作业的结束日期已经过期
  - 2) 作业不是重复执行作业，并且只执行一次
- **program\_name**: 作业执行的程序名称，若 JOB 对象设置了 JOB\_TYPE、JOB\_ACTION、NUMBER\_OF\_ARGUMENTS，则不允许再设置 PROGRAM\_NAME 属性。
  - **schedule\_name**: 作业所使用的调度名称，若 JOB 对象设置了 START\_DATE、END\_DATE、REPEAT\_INTERVAL 属性，则不允许再设置 SCHEDULE\_NAME 属性。
  - **repeat\_interval**: 作业执行的重复/间隔规则，仅支持日历语法，详见 [日历语法](#) 章节。若 JOB 对象设置了 SCHEDULE\_NAME 属性，则不允许设置该属性。
  - **end\_date**: 作业截止日期。若 JOB 对象设置了 SCHEDULE\_NAME 属性，则不允许设置该属性。
  - **start\_date**: 作业起始日期。若 JOB 对象设置了 SCHEDULE\_NAME 属性，则不允许设置该属性。



说明：

**Repeat\_interval、end\_date、start\_date** 属性设置的前提是 JOB 对象未设置 **SCHEDULE\_NAME** 属性。若事先设置了 **SCHEDULE\_NAME** 属性，但后来这个 **SCHEDULE** 对象被删除了，那么 JOB 对象对应的 **SCHEDULE\_NAME** 属性会被置空，用户可以通过组合设置 **repeat\_interval、end\_date、start\_date** 属性或者重新设置 **SCHEDULE\_NAME** 属性，完成调度相关内容的补充，确保 JOB 对象可以正常执行调度。

- **max\_run\_duration**: 作业允许运行的最大时间长度。参数数据类型为 INTERVAL DAY TO SECOND，必须为有效的间隔类型串。
- **comments**: 作业的相关评论或注释。
- **number\_of\_arguments**: 作业的参数个数，等同 create\_job 中的 number\_of\_arguments 参数。若 JOB 对象设置了 PROGRAM\_NAME 属性，则不允许设置该属性。
- **job\_action**: 作业的动作内容，等同 create\_job 中的 job\_action 参数。若 JOB 对象设置了 PROGRAM\_NAME 属性，则不允许设置该属性。
- **job\_type**: 作业的动作类型，等同 create\_job 中 job\_type 参数。若 JOB 对象设置 PROGRAM\_NAME 属性，则不允许设置该属性。



说明:

`Number_of_arguments`、`job_action`、`job_type` 属性设置的前提是 JOB 对象未设置 `PROGRAM_NAME` 属性。若事先设置了 `PROGRAM_NAME` 属性，但后来这个 `PROGRAM` 对象被删除了，那么 JOB 对象对应的 `PROGRAM_NAME` 属性会被置空，用户可以通过组合设置 `number_of_arguments`、`job_action`、`job_type` 属性或者重新设置 `PROGRAM_NAME` 属性，完成程序相关内容的补充，确保 JOB 对象可以正常执行工作。

➤ 以下属性仅作兼容，未实现功能：

```
restartable
allow_runs_in_restricted_mode
follow_default_timezone
instance_stickiness
parallel_instances
stop_on_windows_close
instance_id
max_failures
max_runs
job_priority
logging_level
credential_name
database_role
destination
destination_name
event_spec
job_class
job_weight
raise_events
schedule_limit
```

● VALUE

设置的属性值。

#### 4. PURGE\_LOG 过程

清理日志。

语法如下：

---

```
PROCEDURE PURGE_LOG (
    LOG_HISTORY IN INT          DEFAULT 0,
    WHICH_LOG IN VARCHAR        DEFAULT 'JOB_AND_WINDOW_LOG',
    JOB_NAME     IN VARCHAR     DEFAULT NULL)
```

---

#### 参数详解

- LOG\_HISTORY

历史记录保留天数。取值范围 0-999，如果为 0，则清空历史记录。

- WHICH\_LOG

指定清理的日志类型，不支持，仅作兼容参数用。注：DBMS\_SCHEDULER 日志只支持 JOB\_LOG 类型日志，WHICH\_LOG 指定与否，PURGE\_LOG 均删除作业日志。

- JOB\_NAME

指定清理日志历史记录的作业名称。

#### 5. DISABLE 过程

使指定的对象失效。目前仅支持 PROGRAM 和 JOB。

语法如下：

---

```
PROCEDURE DISABLE (
    NAME          IN VARCHAR,
    FORCE          IN BOOLEAN DEFAULT FALSE,
    COMMIT_SEMANTICS IN VARCHAR DEFAULT 'STOP_ON_FIRST_ERROR')
```

---

#### 参数详解

- NAME

DISABLE 对象名称。目前仅支持 JOB 和 PROGRAM

- FORCE

是否强制 DISABLE 对象。参数不支持，仅作兼容用，目前处理都为强制 DISABLE

- COMMIT\_SEMANTICS

不支持，仅作兼容用。

#### 6. ENABLE 过程

使指定的对象生效。目前仅支持 PROGRAM 和 JOB。

语法如下：

---

```
PROCEDURE ENABLE(
    NAME          IN VARCHAR,
    COMMIT_SEMANTICS IN VARCHAR DEFAULT 'STOP_ON_FIRST_ERROR')
```

---

#### 参数详解

- NAME

ENABLE 对象名称。目前仅支持 JOB 和 PROGRAM。

作业 ENABLE 时，如果作业所使用的 PROGRAM 无效，作业创建失败。

- COMMIT\_SEMANTICS

不支持，仅作兼容用。

## 33.2 创建、删除语句

创建或删除 DBMS\_SCHEDULER 系统包。

语法如下：

```
void  
  
SP_INIT_DBMS_SCHEDULER_SYS (  
    CREATE_FLAG    int  
)
```

### 参数详解

- CREATE\_FLAG

为 1 时表示创建 DBMS\_SCHEDULER 包；为 0 表示删除该系统包。

### 返回值

无

### 举例说明

创建 DBMS\_JOB 系统包。

```
SP_INIT_DBMS_SCHEDULER_SYS (1);
```

## 33.3 相关视图

MPP 环境下，作业只在节点号最小的节点上执行。所有视图，都只显示本站点的相关信息。

### 1. DBA\_SCHEDULER\_PROGRAMS

记录 PROGRAM 对象信息。

序号	列	说明
1	OWNER	PROGRAM 所有者
2	PROGRAM_NAME	PROGRAM 名称

3	PROGRAM_TYPE	PROGRAM 类型，仅支持 PLSQL_BLOCK 和 STORED_PROCEDURE 类型
4	PROGRAM_ACTION	PROGRAM 执行的动作
5	NUMBER_OF_ARGUMENTS	参数个数
6	ENABLED	是否启用
7	DETACHED	不支持，仅作兼容用
8	SCHEDULE_LIMIT	不支持，仅作兼容用
9	PRIORITY	不支持，仅作兼容用
10	WEIGHT	不支持，仅作兼容用
11	MAX_RUNS	不支持，仅作兼容用
12	MAX_FAILURES	不支持，仅作兼容用
13	MAX_RUN_DURATION	不支持，仅作兼容用
14	NLS_ENV	不支持，仅作兼容用
15	COMMENTS	PROGRAM 评论或注释

## 2. DBA\_SCHEDULER\_PROGRAM\_ARGS

记录 PROGRAM 对象的参数定义信息。

序号	列	说明
1	OWNER	PROGRAM 所有者
2	PROGRAM_NAME	PROGRAM 名称
3	ARGUMENT_NAME	参数名称
4	ARGUMENT_POSITION	当前参数在参数列表中所在位置
5	ARGUMENT_TYPE	参数的数据类型
6	METADATA_ATTRIBUTE	不支持，仅作兼容用
7	DEFAULT_VALUE	参数默认值，字符串格式
8	DEFAULT_ANYDATA_VALUE	不支持，仅作兼容用
9	OUT_ARGUMENT	是否为输出参数

## 3. DBA\_SCHEDULER\_SCHEDULES

记录 SCHEDULE 对象信息。

序号	列	说明
1	OWNER	调度的所有者
2	SCHEDULE_NAME	调度名称
3	SCHEDULE_TYPE	调度类型，取值 ONCE, CALENDAR

4	START_DATE	起始时间
5	REPEAT_INTERVAL	调度的重复/间隔规则
6	EVENT_QUEUE_OWNER	不支持，仅作兼容用
7	EVENT_QUEUE_NAME	不支持，仅作兼容用
8	EVENT_QUEUE_AGENT	不支持，仅作兼容用
9	EVENT_CONDITION	不支持，仅作兼容用
10	FILE_WATCHER_OWNER	不支持，仅作兼容用
11	FILE_WATCHER_NAME	不支持，仅作兼容用
12	END_DATE	结束时间
13	COMMENTS	调度的评论或注释

#### 4. DBA\_SCHEDULER\_JOBS

记录 JOB 对象信息。

序号	列	说明
1	OWNER	作业的所有者
2	JOB_NAME	作业名称
3	JOB_SUBNAME	作业子名称，不支持，仅作兼容用
4	JOB_STYLE	作业风格，仅支持 REGULAR
5	JOB_CREATOR	作业原始的创建者
6	CLIENT_ID	不支持，仅作兼容用
7	GLOBAL_UID	不支持，仅作兼容用
8	PROGRAM_OWNER	和作业关联的 PROGRAM 的所有者
9	PROGRAM_NAME	和作业关联的 PROGRAM 的名称
10	JOB_TYPE	作业类型，取值 PLSQL_BLOCK, STORED_PROCEDURE， 不支持 EXECUTABLE 和 CHAIN
11	JOB_ACTION	作业执行的动作
12	NUMBER_OF_ARGUMENTS	作业参数个数
13	SCHEDULE_OWNER	调度的所有者
14	SCHEDULE_NAME	和作业相关联的调度名称
15	SCHEDULE_TYPE	调度的类型
16	START_DATE	作业起始时间



17	REPEAT_INTERVAL	作业的重复/间隔规则
18	EVENT_QUEUE_OWNER	不支持，仅作参考用
19	EVENT_QUEUE_NAME	不支持，仅作参考用
20	EVENT_QUEUE_AGENT	不支持，仅作参考用
21	EVENT_CONDITION	不支持，仅作参考用
22	EVENT_RULE	不支持，仅作参考用
23	FILE_WATCHER_OWNER	不支持，仅作参考用
24	FILE_WATCHER_NAME	不支持，仅作参考用
25	END_DATE	作业对束时间
26	JOB_CLASS	不支持，仅作参考用
27	ENABLED	作业是否启用
28	AUTO_DROP	作业结束时是否自动删除
29	RESTARTABLE	作业执行失败是否重新启动。不支持，仅作参考用
30	STATE	当前作业的状态
31	JOB_PRIORITY	不支持，仅作参考用
32	RUN_COUNT	不支持，仅作参考用
33	MAX_RUNS	不支持，仅作参考用
34	FAILURE_COUNT	不支持，仅作参考用
35	MAX_FAILURES	不支持，仅作参考用
36	RETRY_COUNT	不支持，仅作参考用
37	LAST_START_DATE	不支持，仅作参考用
38	LAST_RUN_DURATION	不支持，仅作参考用
39	NEXT_RUN_DATE	作业下一次执行时间
40	SCHEDULE_LIMIT	不支持，仅作参考用
41	MAX_RUN_DURATION	作业执行的最大持续时间
42	LOGGING_LEVEL	不支持，仅作参考用
43	STOP_ON_WINDOW_CLOSE	不支持，仅作参考用
44	INSTANCE_STICKINESS	不支持，仅作参考用
45	RAISE_EVENTS	不支持，仅作参考用
46	SYSTEM	不支持，仅作参考用
47	JOB_WEIGHT	不支持，仅作参考用

48	NLS_ENV	不支持，仅作参考用
49	SOURCE	不支持，仅作参考用
50	NUMBER_OF_DESTINATIONS	不支持，仅作参考用
51	DESTINATION_OWNER	不支持，仅作参考用
52	DESTINATION	不支持，仅作参考用
53	CREDENTIAL_OWNER	不支持，仅作参考用
54	CREDENTIAL_NAME	不支持，仅作参考用
55	INSTANCE_ID	不支持，仅作参考用
56	DEFERRED_DROP	不支持，仅作参考用
57	ALLOW_RUNS_IN_RESTRICTED_MODE	不支持，仅作参考用
58	COMMENTS	作业的评论或注释
59	FLAGS	内部使用字段

#### 5. DBA\_SCHEDULER\_JOB\_ARGS

记录 JOB 对象的参数值信息。

序号	列	说明
1	OWNER	参数所属作业的所有者
2	JOB_NAME	参数所属作业名称
3	ARGUMENT_NAME	参数名称
4	ARGUMENT_POSITION	当前参数在参数列表中所在位置
5	ARGUMENT_TYPE	参数的数据类型
6	VALUE	参数值，字符串格式
7	ANYDATA_VALUE	其他格式的参数值
8	OUT_ARGUMENT	是否为输出参数

#### 6. DBA\_SCHEDULER\_JOB\_RUN\_DETAILS

记录 JOB 运行过程中的详细信息。

序号	列	说明
1	LOG_ID	日志唯一 ID
2	LOG_DATE	日志记录的日期
3	OWNER	作业的所有者
4	JOB_NAME	作业名称
5	JOB_SUBNAME	作业子名称，不支持，仅作参考用

6	STATUS	作业运行状态
7	ERROR#	作业出错时的错误码
8	REQ_START_DATE	作业请求运行时间，不支持，仅作参考用
9	ACTUAL_START_DATE	作业实际执行起始时间
10	RUN_DURATION	不支持，仅作参考用
11	INSTANCE_ID	不支持，仅作参考用
12	SESSION_ID	作业执行所在会话 ID
13	SLAVE_PID	不支持，仅作参考用
14	CPU_USED	不支持，仅作参考用
15	CREDENTIAL_OWNER	不支持，仅作参考用
16	CREDENTIAL_NAME	不支持，仅作参考用
17	DESTINATION_OWNER	不支持，仅作参考用
18	DESTINATION	不支持，仅作参考用
19	ADDITIONAL_INFO	作业运行过程中相关信息

#### 7. DBA\_SCHEDULER\_JOB\_LOG

记录作业日志信息。

序号	列	说明
1	LOG_ID	日志唯一 ID
2	LOG_DATE	日志记录的日期
3	OWNER	作业的所有者
4	JOB_NAME	作业名称
5	JOB_SUBNAME	作业子名称，不支持，仅作参考用
6	JOB_CLASS	不支持，仅作参考用
7	OPERATION	不支持，仅作参考用
8	STATUS	作业运行状态
9	USER_NAME	用户名
10	CLIENT_ID	不支持，仅作参考用
11	GLOBAL_UID	不支持，仅作参考用
12	CREDENTIAL_OWNER	不支持，仅作参考用
13	CREDENTIAL_NAME	不支持，仅作参考用
14	DESTINATION_OWNER	不支持，仅作参考用

14	DESTINATION	不支持，仅作兼容用
16	ADDITIONAL_INFO	作业运行过程中相关信息

#### 8. DBA\_SCHEDULER\_GLOBAL\_ATTRIBUTE

记录 SCHEDULER 包的全局属性。

序号	列	说明
1	ATTRIBUTE_NAME	属性名称，不包括 EMAIL_PASSWORD
2	VALUE	属性值

## 33.4 日历语法

DM 支持的日历格式语法：

```
repeat_interval:: =
    <frequency 子句>[ ;<interval 子句>] [ ;<bymonth 子句>] [ ;<byweekno 子句>] [ ;<byyearmonth 子句>] [ ;<byhour 子句>] [ ;<byminute 子句>] [ ;<bysecond 子句>]

    <frequency 子句>::= FREQ = <predefined_frequency>

    <predefined_frequency>::= YEARLY | MONTHLY | WEEKLY | DAILY | HOURLY | MINUTELY | SECONDLY

    <interval 子句> ::= INTERVAL = <intervalnum>

    <intervalnum> ::= <取值 1 到 99>

    <bymonth 子句> ::= BYMONTH = <monthlist>

    <monthlist> ::= <month>{,<month>}

    <month> ::= <numeric_month> | <char_month>

    <numeric_month> ::= 1 | 2 | 3 ... |12

    <char_month> ::= JAN | FEB | MAR | APR | MAY | JUN | JUL | AUG | SEP | OCT | NOV | DEC

    <byweekno 子句>::= BYWEEKNO = <weeknumber_list>

    <weeknumber_list> ::= <weeknumber>{,<weeknumber>}

    <weeknumber> ::= [<minus>] <weekno>
```

```
<weekno> ::= <取值 1 到 53>

<byyear day 子句> ::= BYYEAR DAY = <year day_list>
<year day_list> ::= <year day>{, <year day>}
<year day> ::= [<minus>] <year daynum>
<year daynum> ::= <取值 1 到 366>

<bymonth day 子句> ::= BYMONTH DAY = <month day_list>
<month day_list> ::= <month day>{, <month day>}
<month day> ::= [<minus>] <month daynum>
<month daynum> ::= <取值 1 到 31>

<byday 子句> ::= BYDAY = <byday_list>
<byday_list> ::= <byday>{, <byday>}
<byday> ::= [<weekdaynum>] <day>
<weekdaynum> ::= [<minus>] <daynum>
<daynum> ::= <取值 1 到 53> /* 当 frequency 为 yearly */
<daynum> ::= <取值 1 到 5> /* 当 frequency 为 monthly */
<day> ::= MON | TUE | WED | THU | FRI | SAT | SUN

<byhour 子句> ::= BYHOUR = <hour_list>
<hour_list> ::= <hour>{, <hour>}
<hour> ::= <取值 0 到 23>

<byminute 子句> ::= BYMINUTE = <minute_list>
<minute_list> ::= <minute>{, <minute>}
<minute> ::= <取值 0 到 59>

<bysecond 子句> ::= BYSECOND = <second_list>
<second_list> ::= <second>{, <second>}
<second> ::= <取值 0 到 59>
```

```
<minus> ::= -
```

### 参数详解

- <MINUS>为负数 (-)，表示倒数。
- <BYWEEKNO>只有在 FREQ=YEARLY 时有效。
- <DAYNUM>FREQ=YEARLY 时，<BYDAY>里面<DAYNUM>范围为 1-53；FREQ=MONTHLY 时，<BYDAY>里面<DAYNUM>范围为 1-5。
- SECONDLY，<BYSECOND 子句>秒级设置目前仅语法支持，实际不起作用。

举几个简单的日历写法。例如：

- 1) 每年的第 1 天执行：freq=yearly;byyearday=1;
- 2) 每月的最后一天执行一次：freq=monthly;bymonthday=-1;
- 3) 每月 1, 3, 5 号执行：freq=monthly;bymonthday=1,3,5;
- 4) 每月第一周，周 1 执行：freq=monthly;byday=1MON;
- 5) 每天执行一次，12 点执行：freq=daily;byhour=12。

## 33.5 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_INIT_DBMS_SCHEDULER_SYS(1);
```

准备工作：

```
drop table test cascade;
create table test(c1 varchar, c2 varchar);
create or replace procedure procl(c1 varchar, c2 varchar)
as
begin
insert into test values(c1, c2);
commit;
end;
/
```

创建 PROGRAM：

```
begin
dbms_scheduler.create_program(
program_name=>'PROG1',
program_type=>'STORED_PROCEDURE',
program_action=>'PROC1',
number_of_arguments=>2);
end;
/
```

定义 program 参数：

```
begin
dbms_scheduler.define_program_argument(
program_name=>'PROG1',
argument_position=>1,
argument_name=>'C1',
argument_type=>'VARCHAR'
);
end;
/

begin
dbms_scheduler.define_program_argument(
program_name=>'PROG1',
argument_position=>2,
argument_name=>'C2',
argument_type=>'VARCHAR'
);
end;
/
```

设置 PROGRAM 为 ENABLE 状态:

```
begin
dbms_scheduler.enable('PROG1');
end;
/
```

创建作业:

```
begin
dbms_scheduler.create_job(
job_name=>'test_job',
program_name=>'PROG1',
start_date=>'2018-05-18 8:55:00',
repeat_interval=>'FREQ=MINUTELY;INTERVAL=1',
end_date=>'2018-05-20 9:15:00',
enabled=>FALSE,
auto_drop=>FALSE,
comments=>'test_job',
job_style=>'REGULAR'
);
end;
/
```

设置作业参数值:

```
begin
dbms_scheduler.set_job_argument_value(
job_name=>'test_job',
argument_position=>1,
```

```
argument_value=>'abc'
);
end;
/
begin
dbms_scheduler.set_job_argument_value(
job_name=>'test_job',
argument_name=>'C2',
argument_value=>'abc'
);
end;
/
```

配置全局调度属性:

```
begin
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    'email_server',
    '192.168.0.212');
end;
/

begin
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    'email_sender',
    'shenning@dameng.shanghai');
end;
/

begin
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    'email_port',
    '25');
end;
/

begin
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    'email_password',
    '888888');
end;
/

begin
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE(
    'email_mimetype',
    'TEXT/PLAIN; CHARSET=GB18030');
end;
```



```
end;
/
```

添加作业邮件通知:

```
BEGIN
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION(
    Job_name => 'test_job',
    Recipients => 'shenning@dameng.shanghai',
    Sender    => 'shenning@dameng.shanghai',
    events    => 'JOB_ALL_EVENTS');
END;
/

--设置作业为 ENABLE
begin
dbms_scheduler.enable(name='test_job');
end;
/
```

删除作业邮件通知:

```
BEGIN
DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION(
    Job_name => 'test_job');
END;
/
```

## 34 DBMS\_MVIEW 包

DBMS\_MVIEW 包提供了一个可以一次性刷新多个物化视图的方法。部分兼容 ORACLE 的 DBMS\_MVIEW 包的功能。

### 34.1 相关方法

#### 1) REFRESH

刷新物化视图。

语法如下:

---

```
PROCEDURE REFRESH(
    LST                      IN  VARCHAR,
    METHOD                    IN  VARCHAR          DEFAULT NULL,
    ROLLBACK_SEG             IN  VARCHAR          DEFAULT NULL,
    PUSH_DEFERRED_RPC        IN  BOOL             DEFAULT TRUE,
    REFRESH_AFTER_ERRORS     IN  BOOL             DEFAULT FALSE,
    PURGE_OPTION              IN  BINARY_INTEGER  DEFAULT 1,
```

---

---

```

PARALLELISM          IN  BINARY_INTEGER  DEFAULT 0,
HEAP_SIZE             IN  BINARY_INTEGER  DEFAULT 0,
ATOMIC_REFRESH        IN  BOOL            DEFAULT TRUE,
NESTED                IN  BOOL            DEFAULT FALSE
);

```

---

### 参数详解

- **LST**  
指定的要刷新的物化视图名称，多个物化视图则用逗号隔开。
- **METHOD**  
对应 LST 中的物化视图的刷新方式，'F'表示快速刷新，'C'表示完全刷新，空串表示默认刷新方式。另外，为兼容 ORACLE，还支持'A','P'刷新，但只是语法支持，功能没有实现。标记不区分大小写。
- **ATOMIC\_REFRESH**  
指明 LST 中的所有物化视图如果有失败的话，是提交已成功刷新的记录还是全部不提交。TRUE 表示全部不提交，FALSE 表示提交已成功刷新的记录。缺省为 TRUE。
- **ROLLBACK\_SEG/PUSH\_DEFERRED\_RPC/REFRESH\_FATER\_ERRORS/PURGE/OPTION/PARALLELISM/HEAP\_SIZE/NESTED** 仅语法支持。

## 34.2 举例说明

使用包内的过程和函数之前，如果还未创建过 DBMS\_MVIEW 包。请先调用系统过程创建 DBMS\_MVIEW 包。

```
SP_CREATE_SYSTEM_PACKAGES (1,'DBMS_MVIEW');
```

### 数据准备

--创建模式和表

```

CREATE SCHEMA PURCHASING1 AUTHORIZATION SYSDBA;

CREATE TABLE PURCHASING1.VENDOR

(VENDORID INT IDENTITY(1,1) PRIMARY KEY,

ACCOUNTNO VARCHAR(15) NOT NULL,

NAME VARCHAR(50) NOT NULL,

ACTIVEFLAG BIT NOT NULL,

WEBURL VARCHAR(1024),

CREDIT INT NOT NULL CHECK(CREDIT IN(1,2,3,4,5)));

```

--创建物化视图日志

```
CREATE MATERIALIZED VIEW LOG ON PURCHASING1.VENDOR WITH
```

```
ROWID(VENDORID,ACCOUNTNO,NAME,ACTIVEFLAG,WEBURL,CREDIT) PURGE    START WITH  
SYSDATE    REPEAT INTERVAL '1' DAY;
```

```
--创建物化视图 PURCHASING1.MV_VENDOR_EXCELLENT1
```

```
CREATE MATERIALIZED VIEW  PURCHASING1.MV_VENDOR_EXCELLENT1  
REFRESH WITH ROWID START WITH SYSDATE+1 NEXT SYSDATE + 2 AS  
SELECT ROWID A, VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT  
FROM  PURCHASING1.VENDOR  
WHERE  CREDIT = 1;
```

```
--创建物化视图 SYSDBA.MV_VENDOR_EXCELLENT2
```

```
CREATE MATERIALIZED VIEW  SYSDBA.MV_VENDOR_EXCELLENT2  
REFRESH WITH ROWID START WITH SYSDATE+1 NEXT SYSDATE + 2 AS  
SELECT ROWID A, VENDORID, ACCOUNTNO, NAME, ACTIVEFLAG, CREDIT  
FROM  PURCHASING1.VENDOR  
WHERE  CREDIT = 1;
```

一次性刷新模式 PURCHASING1 模式 MV\_VENDOR\_EXCELLENT1 物化视图和 SYSDBA 模式 MV\_VENDOR\_EXCELLENT2 物化视图，前者使用快速刷新，后者使用完全刷新。

```
CALL DBMS_MVIEW.REFRESH('PURCHASING1.MV_VENDOR_EXCELLENT1',  
SYSDBA.MV_VENDOR_EXCELLENT2', 'FC');
```

## 35 UTL\_SMTP 包

UTL\_SMTP 包的功能是提供对 smtp 服务器的基本访问请求以及通过 SMTP 服务器发送邮件的功能。

使用 UTL\_SMTP 包发送邮件的大致流程为：首先，声明连接，并开启连接；其次，遵照 SMTP 协议与服务器进行握手，并根据 RFC821 简单邮件传输协议进行命令和邮件内容等的发送；最后，关闭会话，关闭连接。

### 35.1 相关方法

下面对各个函数和过程进行详细说明。

#### 1. connection

定义 connection 为 record 类型，为下面的过程或函数使用。

语法如下：

---

```
TYPE CONNECTION IS RECORD (
    REMOTE_HOST      VARCHAR2(255),
    REMOTE_PORT      INT,
    TX_TIMEOUT       INT,
    PRIVATE_HNDL     BIGINT
);
```

---

#### 参数详解

- Remote\_host, remote\_port  
为 smtp 服务器的地址及端口号，由 open\_connection 过程传入。
- tx\_timeout  
为等待响应时间，同样由 open\_connection 过程传入。
- private\_hndl  
为内部标识 connection 的 id

#### 2. OPEN\_CONNECTION

用于开启并返回与 SMTP 服务器的连接。

语法如下：

---

```
FUNCTION OPEN_CONNECTION (
    HOST      IN  VARCHAR2,
    PORT      IN  PLS_INTEGER DEFAULT 25,
    TX_TIMEOUT IN  PLS_INTEGER DEFAULT NULL)
RETURN CONNECTION;
```

---

#### 参数详解

- `host`  
SMTP 服务器地址。
- `port`  
SMTP 服务器端口号。
- `tx_timeout`  
等待时间，`NULL` 代表一直等待，`0` 代表不等待。

### 3. HELO/EHLO

用于向 SMTP 服务器打招呼。EHLO 指令表示需要 SMTP 认证，HELO 指令表示不需要 SMTP 认证。

语法如下：

---

```
PROCEDURE HELO (
    C          IN OUT NOCOPY   CONNECTION,
    R_DOMAIN  IN              VARCHAR2);
```

---

或者

---

```
PROCEDURE EHLO (
    C          IN OUT NOCOPY   CONNECTION,
    R_DOMAIN  IN              VARCHAR2);
```

---

### 参数详解

- `c`  
使用的 SMTP 连接。

### 4. COMMAND

用于向 smtp 服务器发送命令。命令不由此程序判断正确性，发送错误命令，程序会报错。

语法如下：

---

```
PROCEDURE COMMAND (
    C          IN OUT NOCOPY   CONNECTION,
    CMD       IN              VARCHAR2,
    ARG       IN              VARCHAR2 DEFAULT NULL);
```

---

### 参数详解

- `c`  
使用的连接。
- `cmd` 命令名  
准备登陆为 AUTH LOGIN(AUTH LOGIN 命令使用后，要分别再使用 command 发送用户名和密码)；握手为 HELO；更多的命令详见 RFC821 简单邮件传输协议。
- `arg` 命令参数  
可选参数，不同的 cmd 使用不同的 arg。例如：AUTH LOGIN 无参数，但必须接下来继续使用 command 发送用户名和密码；而 HELO 后面的参数为服务器地址。

## 5. MAIL

用于告知服务器发件人。

语法如下：

---

```
PROCEDURE MAIL (  
    C          IN OUT NOCOPY   CONNECTION,  
    SENDER     IN              VARCHAR2,  
    M_PARAMETERS IN           VARCHAR2 DEFAULT NULL);
```

---

### 参数详解

- **c**  
使用的 SMTP 连接。
- **sender**  
发件人邮箱。
- **m\_parameters**  
可选参数。公认的几种 SMTP 的扩展将会用到 MAIL FROM 和 RCPT TO 的额外参数，详见 RFC1869 第 6 部分，格式要求必须符合 "XXX=XXX (XXX=XXX ....)"。

## 6. RCPT

用于告知 SMTP 服务器收件人信息。

语法如下：

---

```
PROCEDURE RCPT (  
    C          IN OUT NOCOPY   CONNECTION,  
    RECIPIENTS IN              VARCHAR2,  
    R_PARAMETERS IN           VARCHAR2 DEFAULT NULL);
```

---

### 参数详解

- **c**  
使用的 SMTP 连接。
- **recipients**  
收件人邮箱。
- **r\_parameters**  
可选参数。公认的几种 SMTP 的扩展将会用到 MAIL FROM 和 RCPT TO 的额外参数，详见 RFC1869 第 6 部分，格式要求必须符合 "XXX=XXX (XXX=XXX ....)"。

## 7. OPEN\_DATA

用于告知 SMTP 服务器开始发送数据。只有使用过该过程才可以发送数据。

语法如下：

---

```
PROCEDURE OPEN_DATA (  
    C          IN OUT NOCOPY CONNECTION);
```

---

### 参数详解

- **c**

使用的 SMTP 连接。

#### 8. WRITE\_DATA

用于发送邮件内容 (varchar 类型)。

语法如下：

---

```
PROCEDURE WRITE_DATA (  
    C      IN OUT NOCOPY CONNECTION,  
    DATA  IN VARCHAR2);
```

---

##### 参数详解

- C  
使用的 SMTP 连接。
- data  
发送的内容，varchar 类型。

#### 9. WRITE\_RAW\_DATA

用来发送邮件内容 (varbinary 类型)。

语法如下：

---

```
PROCEDURE WRITE_RAW_DATA (  
    C      IN OUT NOCOPY CONNECTION,  
    DATA  IN VARBINARY);
```

---

##### 参数详解

- C  
使用的 SMTP 连接。
- data  
邮件内容，varbinary 类型。

#### 10. CLOSE\_DATA

关闭数据流，告知服务器数据发送完毕。

语法如下：

---

```
PROCEDURE CLOSE_DATA (  
    C      IN OUT NOCOPY CONNECTION);
```

---

##### 参数详解

- C  
所使用 SMTP 连接。

#### 11. QUIT

用来结束会话。

语法如下：

---

```
PROCEDURE QUIT (  
    C IN OUT NOCOPY CONNECTION);
```

---

##### 参数详解

- c  
使用的 SMTP 连接。

## 12. CLOSE\_CONNECTION

关闭 SMTP 连接。

语法如下：

---

```
PROCEDURE CLOSE_CONNECTION (
    C      IN OUT NOCOPY CONNECTION);
```

---

### 参数详解

- c  
所需要关闭的 SMTP 连接。

## 35.2 应用实例

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1, 'UTL_SMTP');
SP_CREATE_SYSTEM_PACKAGES(1, 'UTL_TCP');
SP_CREATE_SYSTEM_PACKAGES(1, 'UTL_RAW');
```

举一个使用 UTL\_SMTP 包发邮件的例子。发件人 yhx@dameng.shanghai，收件人 gyf@dameng.shanghai，发送内容为邮件头部信息(收件人、发件人、发送时间、主题、编码等)，重复 10 次的“达梦数据库第 i 行”。

```
DECLARE
    C UTL_SMTP.CONNECTION;      --声明连接
    MSG VARCHAR2(4000);
BEGIN
    C := UTL_SMTP.OPEN_CONNECTION('192.168.0.212');  --开启连接
    UTL_SMTP.HELO(C, '192.168.0.212');  --握手
    UTL_SMTP.COMMAND(C, 'AUTH LOGIN');  --准备登陆

    UTL_SMTP.COMMAND(C, UTL_RAW.CAST_TO_VARCHAR2(UTL_ENCODE.BASE64_ENCODE(UTL_RAW.
    CAST_TO_RAW('yhx@DAMENG.SHANGHAI'))));  --用户名

    UTL_SMTP.COMMAND(C, UTL_RAW.CAST_TO_VARCHAR2(UTL_ENCODE.BASE64_ENCODE(UTL_RAW.
    CAST_TO_RAW('888888'))));  --密码

    UTL_SMTP.MAIL(C, 'yhx@DAMENG.SHANGHAI');  --发件人
    UTL_SMTP.RCPT(C, 'GYF@DAMENG.SHANGHAI');  --收件人
    UTL_SMTP.OPEN_DATA(C);  --开始发送邮件内容

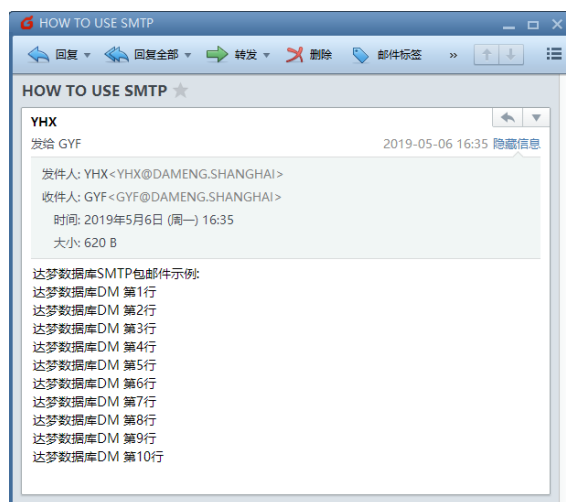
    MSG := 'CONTENT-TYPE: TEXT/PLAIN; CHARSET=GB2312' || UTL_TCP.CRLF || 'DATE:' ||
    TO_CHAR(SYSDATE, 'MM DD YYYY HH24:MI:SS') || UTL_TCP.CRLF || 'FROM:' ||
    'YHX@DAMENG.SHANGHAI' || UTL_TCP.CRLF || 'SUBJECT: HOW TO USE SMTP' ||
    UTL_TCP.CRLF ||
```



```

'TO: ' || 'GYF@DAMENG.SHANGHAI' || UTL_TCP.CRLF ||
'CONTENT-TYPE: TEXT/PLAIN; CHARSET=GB2312' || UTL_TCP.CRLF ;
--MSG 为邮件头信息，邮件头信息需要与正文信息，通过 UTL_TCP.CRLF 来分隔开
UTL_SMTP.WRITE_RAW_DATA(C, UTL_RAW.CAST_TO_RAW(MSG)); --发送邮件头信息
UTL_SMTP.WRITE_DATA(C, UTL_TCP.CRLF || '达梦数据库 SMTP 包邮件示例:' ||
UTL_TCP.CRLF); --发送邮件正文信息
FOR I IN 1 .. 10 LOOP
    UTL_SMTP.WRITE_DATA(C, '达梦数据库 DM 第' || i || '行' || UTL_TCP.CRLF);
END LOOP;
UTL_SMTP.CLOSE_DATA(C); --邮件内容发送完毕
UTL_SMTP.QUIT(C); --结束会话
UTL_SMTP.CLOSE_CONNECTION(C); --关闭连接
END;
```

所收到的邮件内容 (包含 msg 中所写的收件人，发件人，时间信息):



## 36 UTL\_HTTP 包

UTL\_HTTP 包提供了通过 HTTP 协议获取网页内容的功能。

UTL\_HTTP 包中各函数的调用顺序如下图所示。另外，REQUEST 函数是独立于流程图之外的，相当于使用流程图中各函数的缺省配置，最终返回指定网页的前 2000 个字节。

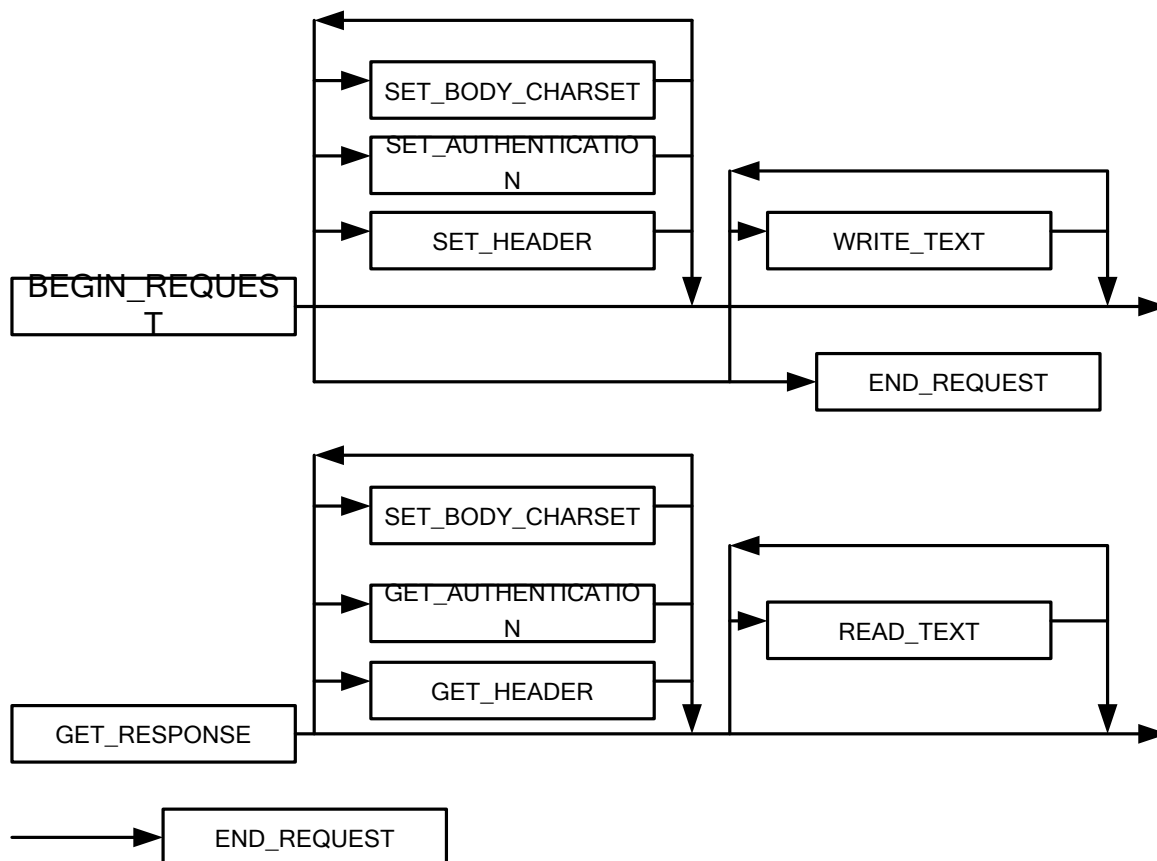


图 UTL\_HTTP 包中函数使用顺序图

### 36.1 相关方法

下面对各个函数和过程进行详细说明。

#### 1. REQ

定义 REQ 为 RECORD 类型，用来存放请求消息。

语法如下：

```

TYPE REQ IS RECORD (
    URL           VARCHAR2(32767),
    METHOD         VARCHAR2(64),
    HTTP_VERSION  VARCHAR2(64)
);

```

---

);

---

### 参数详解

- URL WEB 服务器的 URL 地址。URL 中可以包含用户名密码。
- METHOD 对指定 URL 发起的操作命令。常见的命令有 GET、POST，默认为 GET。
- HTTP\_VERSION HTTP 协议版本号,UTL\_HTTP 包提供 1.0 和 1.1 两个版本宏定义，若此参数设为 NULL，则默认使用最新的 HTTP 协议版本。此参数默认为 NULL。

#### 2. RESP

定义 RESP 为 RECORD 类型，用来存放响应消息。

语法如下：

---

```
TYPE RESP IS RECORD (
    STATUS_CODE    PLS_INTEGER,
    REASON_PHRASE  VARCHAR2(256),
    HTTP_VERSION   VARCHAR2(64)
);
```

---

### 参数详解

- STATUS\_CODE WEB 服务器返回的状态码。
- REASON\_PHRASE 关于状态码的简单文本描述。
- HTTP\_VERSION HTTP 协议版本号,UTL\_HTTP 包提供 1.0 和 1.1 两个版本宏定义，若此参数设为 NULL，则默认使用最新的 HTTP 协议版本。此参数默认为 NULL。

#### 3. BEGIN\_REQUEST

用于发起一个新的 http 请求。

语法如下：

---

```
FUNCTION BEGIN_REQUEST (
    URL              IN VARCHAR2,
    METHOD            IN VARCHAR2 DEFAULT 'GET',
    HTTP_VERSION     IN VARCHAR2 DEFAULT NULL,
    REQUEST_CONTEXT  IN REQUEST_CONTEXT_KEY DEFAULT NULL)
RETURN REQ;
```

---

### 参数详解

- URL WEB 服务器的 URL 地址。URL 中可以包含用户名密码。
- METHOD 对指定 URL 发起的操作命令。
- HTTP\_VERSION HTTP 协议版本号,UTL\_HTTP 包提供 1.0 和 1.1 两个版本宏定义，若此参数设为 NULL，则默认使用最新的 HTTP 协议版本。此参数默认为 NULL。
- REQUEST\_CONTEXT 目前达梦不支持此参数，仅用作兼容参数。

### 返回值

HTTP 请求消息。

#### 4. END\_REQUEST

用于结束 http 请求。

语法如下：

---

```
PROCEDURE END_REQUEST (
    R IN OUT NOCOPY REQ
);
```

---

#### 参数详解

- R HTTP 请求句柄。

5. GET\_RESPONSE

用于读取 http 响应消息。

语法如下：

---

```
FUNCTION GET_RESPONSE (
    R IN OUT NOCOPY REQ)
RETURN RESP;
```

---

#### 参数详解

- R HTTP 请求消息。

#### 返回值

HTTP 请求句柄。

6. END\_RESPONSE

用于结束 http 响应操作。

语法如下：

---

```
PROCEDURE END_RESPONSE (
    R IN OUT NOCOPY RESP
);
```

---

#### 参数详解

- R HTTP 响应句柄。

7. WRITE\_TEXT

用于向 http 请求 body 中填写数据。

语法如下：

---

```
PROCEDURE WRITE_TEXT(
    R IN OUT NOCOPY REQ,
    DATA IN          VARCHAR2 CHARACTER SET ANY_CS
);
```

---

#### 参数详解

- R HTTP 请求消息。
- DATA 准备向 BODY 写入的数据。

8. READ\_TEXT

用于读取 http 响应消息中 body 的内容并将内容输出到指定的缓冲区中。body 中的数

据将自动转换成数据库所指定的字符集。

语法如下：

---

```
PROCEDURE READ_TEXT(
    R      IN OUT NOCOPY RESP,
    DATA  OUT NOCOPY VARCHAR2 CHARACTER SET ANY_CS,
    LEN    IN PLS_INTEGER DEFAULT NULL
);
```

---

#### 参数详解

- RHTTP 请求消息。
- DATA 读取 BODY 中的数据。
- LEN 请求读取的数据长度

#### 9. SET\_HEADER

用于设置 http 请求的头信息。设置头信息的请求将被立即发送到 web 服务器。

语法如下：

---

```
PROCEDURE SET_HEADER (
    R      IN OUT NOCOPY REQ,
    NAME   IN VARCHAR2,
    VALUE  IN VARCHAR2
);
```

---

#### 参数详解

- R HTTP 请求消息。
- NAME HEADER 中请求设置的属性名。
- VALUE HEADER 中请求设置的属性值。

#### 10. GET\_HEADER

用于获取 http 响应头信息。过程将响应头中的第 n 个属性名及属性值返回。

语法如下：

---

```
PROCEDURE GET_HEADER (
    R      IN OUT NOCOPY RESP,
    N      IN PLS_INTEGER,
    NAME   OUT NOCOPY VARCHAR2,
    VALUE  OUT NOCOPY VARCHAR2
);
```

---

#### 参数详解

- R HTTP 响应消息。
- N HEADER 中的第 N 个属性。
- NAME 属性名。
- VALUE 属性值。

#### 11. SET\_BODY\_CHARSET

用于设置 body 的字符集，至于是请求还是响应消息中的 body，则需要看上下文。  
语法如下：

---

```
PROCEDURE SET_BODY_CHARSET (
    CHARSET IN VARCHAR2 DEFAULT NULL
);
```

---

或者

---

```
PROCEDURE SET_BODY_CHARSET(
    R          IN OUT NOCOPY REQ,
    CHARSET IN VARCHAR2 DEFAULT NULL
);
```

---

或者

---

```
PROCEDURE SET_BODY_CHARSET(
    R          IN OUT NOCOPY RESP,
    CHARSET IN VARCHAR2 DEFAULT NULL
);
```

---

- R HTTP 请求（或响应）句柄。
- CHARSET 字符集名称。

## 12. SET\_AUTHENTICATION

用于设置 http 请求消息中的权限。WEB 服务器需要这些权限信息用于授权访问。  
语法如下：

---

```
PROCEDURE SET_AUTHENTICATION(
    R          IN OUT NOCOPY REQ,
    USERNAME IN VARCHAR2,
    PASSWORD IN VARCHAR2,
    SCHEME    IN VARCHAR2 DEFAULT 'BASIC',
    FOR_PROXY IN BOOLEAN  DEFAULT FALSE
);
```

---

### 参数详解

- R HTTP 请求消息。
- USERNAME WEB 服务器端所需的用户名。
- PASSWORD WEB 服务器端所需的密码。
- SCHEME 访问模式，仅支持"BASIC"。
- FOR\_PROXY 是否通过代理来访问 WEB，默认为 FALSE。

## 13. GET\_AUTHENTICATION

从 http 响应消息中获取 WEB 服务器所需要的授权信息。  
语法如下：

---

```
PROCEDURE GET_AUTHENTICATION(
    R          IN OUT NOCOPY RESP,
```

---

```

SCHEME      OUT VARCHAR2,
REALM       OUT VARCHAR2,
FOR_PROXY   IN BOOLEAN  DEFAULT FALSE
);

```

### 参数详解

- R HTTP 响应消息。
- SCHEME WEB 服务器的授权模式。
- REALM WEB 服务器需要授权的区域。
- FOR\_PROXY 是否返回访问代理所需的权限而不是 WEB 服务器的权限，默认 FALSE。

### 14. REQUEST

返回网页的前 2000 个字节。该函数可以直接用在 SQL 查询中。

语法如下：

```

FUNCTION      REQUEST (
    URL              IN  VARCHAR2
)RETURN REQ;

```

### 参数详解

URL WEB 服务器的 URL 地址。URL 中可以包含用户名密码。

### 返回值

返回网页的前 2000 个字节。

## 36.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程创建系统包。

```

SP_CREATE_SYSTEM_PACKAGES (1, 'UTL_HTTP');
SP_CREATE_SYSTEM_PACKAGES (1, 'DBMS_OUTPUT');
SET SERVEROUTPUT ON;  --DBMS_OUTPUT.PUT_LINE 需要设置这条语句，才能打印出消息

```

举一个使用 UTL\_HTTP 包获取网页信息的例子。

```

declare
req utl_http.REQ;
resp utl_http.resp;
data varchar2(32563);
receiveDate varchar;
begin
req :=
utl_http.begin_request('http://192.168.0.104/blog/2015_06_05/CSDN_public_1433
381803730.html', 'POST');
data := ' ';
utl_http.write_text(req, data);
resp := utl_http.get_response(req);

```

```
utl_http.set_body_charset(resp, 'utf-8');  
utl_http.read_text(resp, receiveDate, 1024);  
dbms_output.put_line(receiveDate);  
utl_http.end_response(resp);  
end;
```

输出的网页内容为:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html xmlns="www.w3.org/1999/xhtml">  
  <head>  
    <script type="text/javascript" src="c.csdnimg.cn/pubfooter/js/tracking.js"  
charset="utf-8"></script>  
    <script type="text/javascript">  
      var protocol = window.location.protocol;  
      document.write('<script type="text/javascript"  
src="csdnimg.cn/pubfooter/js/repoAddr2.js?v=' + Math.random() + '"></' +  
'script>');  
    </script>  
    <script id="allmobilize" charset="utf-8"  
src="a.yunshiwei.com/46aae4d1e2371e4aa769798941cef698/allmobilize.min.js"></s  
cript>  
    <meta http-equiv="Cache-Control" content="no-siteapp">  
    <link rel="alternate" media="handheld" href="#">  
    <title>使用 UTL_HTTP 包获取网页内容 - IndexMan 的专栏  
      - 博客频道 - CSDN.NET</title>  
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  
    <meta name="description" content="UTL_HTTP 包提供了容易的方式通过 HTTP 协议获取网页  
内容
```



## 37 UTL\_I18N 包

为了在 DM 上兼容 oracle 的 utl\_i18n 包，提供功能上与 oracle 基本一致的 utl\_i18n 包。UTL\_I18N 包提供字符串与十六进制编码的相互转换功能。

UTL\_I18N 依赖于 utl\_raw。创建 UTL\_I18N 之前，确保 utl\_raw 已成功创建。

### 37.1 相关方法

下面对各个函数和过程进行详细说明。

#### 1. STRING\_TO\_RAW

将字符串转换成指定字符集下的十六进制串数值。

语法如下：

---

```
FUNCTION STRING_TO_RAW(  
    DATA IN VARCHAR,  
    TO_CHARSET IN VARCHAR DEFAULT NULL)  
RETURN VARBINARY
```

---

#### 参数详解

- DATA 输入参数，字符串。
- TO\_CHARSET 输入参数，字符集。指定转换后的字符集。

#### 返回值

转换后的十六进制串数值。字符集输错了，转换失败了，返回 NULL；单字节 0~255 用任何字符集编码都可以转成 ASCII 值；三种字符集库（GBK 库、UTF8 库和韩文库）下都支持转换到 UTF8 编码，只有 UTF8 库支持转到字符集（GBK、BIG5、ISO\_8859\_9、EUC\_JP、EUC\_KR、KOI8R、GB18030、SQL\_ASCII、ISO\_8859\_1）。字符串输入 null，返回当前字符集下的编码。

#### 2. RAW\_TO\_CHAR

将从指定的字符集下的十六进制串数值转到字符串。

语法如下：

---

```
FUNCTION RAW_TO_CHAR(  
    DATA IN VARBINARY ,  
    FROM_CHARSET IN VARCHAR DEFAULT NULL)  
RETURN VARCHAR
```

---

#### 参数详解

- DATA 输入参数，VARBINARY 数据。
- FROM\_CHARSET 输入参数，字符集。转换前的字符集。

## 返回值

转换后的字符串。注意：字符集输错了，转换失败了，返回 NULL；ASCII 值用任何字符集编码都可以转成单字节；只有 UTF8 库支持指定字符集(GBK、BIG5、ISO\_8859\_9、EUC\_JP、EUC\_KR、KOI8R、GB18030、SQL\_ASCII、ISO\_8859\_1)。十六进制串输入 NULL，返回在当前字符集下的字符串。

## 37.2 举例说明

使用包内的过程和函数之前，如果还未创建过系统包。请先调用系统过程

SP\_CREATE\_SYSTEM\_PACKAGES (1)创建系统包。

```
SP_CREATE_SYSTEM_PACKAGES(1);
```

例 1 STRING\_TO\_RAW 函数的使用。

一 将 GBK 库中，GBK 编码的字符串转换成指定字符集下的十六进制串数值。

```
SQL> select utl_i18n.string_to_raw('中国','utf8') from dual;
```

查询结果为：

```
E4B8ADE59BBD
```

```
SQL> select utl_i18n.string_to_raw('中国','GBK') from dual;
```

查询结果为：

```
D6D0B9FA
```

```
SQL> select utl_i18n.string_to_raw('abcdef','utf8') from dual;
```

查询结果为：

```
616263646566
```

```
SQL> select utl_i18n.string_to_raw('abcdef','GBK') from dual;
```

查询结果为：

```
616263646566
```

二 UTF8 库中，将 UTF8 编码的字符串转换成指定字符集下的十六进制串数值。

```
SQL> select utl_i18n.string_to_raw('備註','BIG5');
```

查询结果为：

```
0xB3C6B5F9
```

三 韩文库中，将韩文编码的字符串转换成指定字符集下的十六进制串数值。

```
SQL>select utl_i18n.string_to_raw(' ','EUC_KR') from dual;
```

查询结果为：

```
0x20C3B6BCF6
```

例 2 RAW\_TO\_CHAR 函数的使用。

一 从指定字符集下的十六进制串数值转为 GBK 编码（例如，本机为 GBK 库）字符串。

```
select UTL_I18N.RAW_TO_CHAR('61', 'utf8') from dual;
```

查询结果为:

a

```
select UTL_I18N.RAW_TO_CHAR('E4B8ADE59BBD', 'UTF8') from dual;
```

查询结果为:

中国

```
select UTL_I18N.RAW_TO_CHAR('D6D0B9FA', 'GBK') from dual;
```

查询结果为:

中国

```
SQL> select UTL_I18N.RAW_TO_CHAR('616263', 'GBK') from dual;
```

查询结果为:

abc

```
select UTL_I18N.RAW_TO_CHAR('616263', 'UTF8') from dual;
```

查询结果为:

abc

二 从指定字符集下的十六进制串数值转为 UTF8 编码(例如,本机为 UTF8 库)字符串。

```
select UTL_I18N.RAW_TO_CHAR(utl_i18n.string_to_raw('備註','BIG5'),'BIG5') from dual;
```

查询结果为:

備註

三 从指定字符集下的十六进制串数值转为韩文编码(例如,本机为韩文库)字符串。

```
select UTL_I18N.RAW_TO_CHAR(0x20C3B6BCF6,'EUC_KR') from dual;
```

查询结果为:

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

---