Db2 11.1 for Linux, UNIX, and Windows

Built-In Modules

Db2 11.1 for Linux, UNIX, and Windows

Built-In Modules

Notice regarding this document

This document in PDF form is provided as a courtesy to customers who have requested documentation in this format. It is provided As-Is without warranty or maintenance commitment.

Contents

Notice regarding this document iii

Figures ix

Tables xi

Built-in modules 1

DBMS_ALERT module 1
REGISTER procedure - Register to receive a specified alert 2
REMOVE procedure - Remove registration for a specified alert 3
REMOVEALL procedure - Remove registration for all alerts 4
SET_DEFAULTS - Set the polling interval for WAITONE and WAITANY. 4
SIGNAL procedure - Signal occurrence of a specified alert 5
WAITANY procedure - Wait for any registered alerts 6
WAITONE procedure - Wait for a specified alert . 7
DBMS_DDL module 9
WRAP function - Obfuscate a DDL statement . . 9
CREATE_WRAPPED procedure - Deploy an obfuscated object 10
DBMS_JOB module 12
BROKEN procedure - Set the state of a job to either broken or not broken 14
CHANGE procedure - Modify job attributes . . 15
INTERVAL procedure - Set run frequency . . . 15
NEXT_DATE procedure - Set the date and time when a job is run 16
REMOVE procedure - Delete the job definition from the database 17
RUN procedure - Force a broken job to run. . . 18
SUBMIT procedure - Create a job definition and store it in the database. 18
WHAT procedure - Change the SQL statement run by a job 19
DBMS_LOB module 20
APPEND procedures - Append one large object to another 21
CLOSE procedures - Close an open large object 22
COMPARE function - Compare two large objects 22
CONVERTTOBLOB procedure - Convert character data to binary 23
CONVERTTOCLOB procedure - Convert binary data to character. 24
COPY procedures - Copy one large object to another 25
ERASE procedures - Erase a portion of a large object 26
GET_STORAGE_LIMIT function - Return the limit on the largest allowable large object . . . 26
GETLENGTH function - Return the length of the large object 27
INSTR function - Return the location of the *n*th occurrence of a given pattern 27
ISOPEN function - Test if the large object is open 28
OPEN procedures - Open a large object 28
READ procedures - Read a portion of a large object 29
SUBSTR function - Return a portion of a large object 29
TRIM procedures - Truncate a large object to the specified length 30
WRITE procedures - Write data to a large object 30
WRITEAPPEND procedures - Append data to the end of a large object 31
DBMS_OUTPUT module 32
DISABLE procedure - Disable the message buffer 33
ENABLE procedure - Enable the message buffer 33
GET_LINE procedure - Get a line from the message buffer 34
GET_LINES procedure - Get multiple lines from the message buffer 35
NEW_LINE procedure - Put an end-of-line character sequence in the message buffer . . . 37
PUT procedure - Put a partial line in the message buffer 37
PUT_LINE procedure - Put a complete line in the message buffer 38
DBMS_PIPE module 39
CREATE_PIPE function - Create a pipe 41
NEXT_ITEM_TYPE function - Return the data type code of the next item 42
PACK_MESSAGE function - Put a data item in the local message buffer 44
PACK_MESSAGE_RAW procedure - Put a data item of type RAW in the local message buffer . . 45
PURGE procedure - Remove unreceived messages from a pipe 46
RECEIVE_MESSAGE function - Get a message from a specified pipe 47
REMOVE_PIPE function - Delete a pipe 48
RESET_BUFFER procedure - Reset the local message buffer 50
SEND_MESSAGE procedure - Send a message to a specified pipe 51
UNIQUE_SESSION_NAME function - Return a unique session name 52
UNPACK_MESSAGE procedures - Get a data item from the local message buffer 53
DBMS_RANDOM module 54
SEED procedure 55
SEED_STRING procedure 55
INITIALIZE procedure 56
TERMINATE procedure 56
RANDOM function. 56
VALUE function 56

STRING function 57
NORMAL function 58
DBMS_SQL module 58
BIND_VARIABLE_BLOB procedure - Bind a BLOB value to a variable 61
BIND_VARIABLE_CHAR procedure - Bind a CHAR value to a variable 62
BIND_VARIABLE_CLOB procedure - Bind a CLOB value to a variable 62
BIND_VARIABLE_DATE procedure - Bind a DATE value to a variable 63
BIND_VARIABLE_DOUBLE procedure - Bind a DOUBLE value to a variable. 63
BIND_VARIABLE_INT procedure - Bind an INTEGER value to a variable 64
BIND_VARIABLE_NUMBER procedure - Bind a NUMBER value to a variable 64
BIND_VARIABLE_RAW procedure - Bind a RAW value to a variable 64
BIND_VARIABLE_TIMESTAMP procedure - Bind a TIMESTAMP value to a variable 65
BIND_VARIABLE_VARCHAR procedure - Bind a VARCHAR value to a variable 66
CLOSE_CURSOR procedure - Close a cursor . . 66
COLUMN_VALUE_BLOB procedure - Return a BLOB column value into a variable 67
COLUMN_VALUE_CHAR procedure - Return a CHAR column value into a variable 67
COLUMN_VALUE_CLOB procedure - Return a CLOB column value into a variable 68
COLUMN_VALUE_DATE procedure - Return a DATE column value into a variable 68
COLUMN_VALUE_DOUBLE procedure - Return a DOUBLE column value into a variable . . . 69
COLUMN_VALUE_INT procedure - Return an INTEGER column value into a variable 70
COLUMN_VALUE_LONG procedure - Return a LONG column value into a variable 70
COLUMN_VALUE_NUMBER procedure - Return a DECFLOAT column value into a variable. . . 71
COLUMN_VALUE_RAW procedure - Return a RAW column value into a variable 72
COLUMN_VALUE_TIMESTAMP procedure - Return a TIMESTAMP column value into a variable. 72
COLUMN_VALUE_VARCHAR procedure - Return a VARCHAR column value into a variable. 73
DEFINE_COLUMN_BLOB- Define a BLOB column in the SELECT list 74
DEFINE_COLUMN_CHAR procedure - Define a CHAR column in the SELECT list 74
DEFINE_COLUMN_CLOB - Define a CLOB column in the SELECT list 75
DEFINE_COLUMN_DATE - Define a DATE column in the SELECT list 75
DEFINE_COLUMN_DOUBLE - Define a DOUBLE column in the SELECT list 76
DEFINE_COLUMN_INT- Define an INTEGER column in the SELECT list 76
DEFINE_COLUMN_LONG procedure - Define a LONG column in the SELECT list 77
DEFINE_COLUMN_NUMBER procedure - Define a DECFLOAT column in the SELECT list . 77
DEFINE_COLUMN_RAW procedure - Define a RAW column or expression in the SELECT list . 78
DEFINE_COLUMN_TIMESTAMP - Define a TIMESTAMP column in the SELECT list. . . . 78
DEFINE_COLUMN_VARCHAR procedure - Define a VARCHAR column in the SELECT list . 79
DESCRIBE_COLUMNS procedure - Retrieve a description of the columns in a SELECT list . . 79
DESCRIBE_COLUMNS2 procedure - Retrieve a description of column names in a SELECT list. . 82
EXECUTE procedure - Run a parsed SQL statement 84
EXECUTE_AND_FETCH procedure - Run a parsed SELECT command and fetch one row . . 85
FETCH_ROWS procedure - Retrieve a row from a cursor 88
IS_OPEN function - Check whether a cursor is open. 90
IS_OPEN procedure - Check whether a cursor is open. 91
LAST_ROW_COUNT procedure - return the cumulative number of rows fetched 92
OPEN_CURSOR procedure - Open a cursor . . 94
PARSE procedure - Parse an SQL statement . . 95
VARIABLE_VALUE_BLOB procedure - Return the value of a BLOB INOUT or OUT parameter . 97
VARIABLE_VALUE_CHAR procedure - Return the value of a CHAR INOUT or OUT parameter . 98
VARIABLE_VALUE_CLOB procedure - Return the value of a CLOB INOUT or OUT parameter . 98
VARIABLE_VALUE_DATE procedure - Return the value of a DATE INOUT or OUT parameter . 99
VARIABLE_VALUE_DOUBLE procedure - Return the value of a DOUBLE INOUT or OUT parameter 99
VARIABLE_VALUE_INT procedure - Return the value of an INTEGER INOUT or OUT parameter 100
VARIABLE_VALUE_NUMBER procedure - Return the value of a DECFLOAT INOUT or OUT parameter. 100
VARIABLE_VALUE_RAW procedure - Return the value of a BLOB(32767) INOUT or OUT parameter 100
VARIABLE_VALUE_TIMESTAMP procedure - Return the value of a TIMESTAMP INOUT or OUT parameter. 101
VARIABLE_VALUE_VARCHAR procedure - Return the value of a VARCHAR INOUT or OUT parameter. 101
DBMS_UTILITY module. 102
ANALYZE_DATABASE procedure - Gather statistics on tables, clusters, and indexes . . . 103
ANALYZE_PART_OBJECT procedure - Gather statistics on a partitioned table or partitioned index 104

ANALYZE_SCHEMA procedure - Gather statistics on schema tables, clusters, and indexes. 105
CANONICALIZE procedure - Canonicalize a string 106
COMMA_TO_TABLE procedures - Convert a comma-delimited list of names into a table of names 108
COMPILE_SCHEMA procedure - Compile all functions, procedures, triggers, and packages in a schema 110
DB_VERSION procedure - Retrieve the database version 110
EXEC_DDL_STATEMENT procedure - Run a DDL statement 111
FORMAT_CALL_STACK 112
FORMAT_ERROR_BACKTRACE 113
GET_CPU_TIME function - Retrieve the current CPU time. 115
GET_DEPENDENCY procedure - List objects dependent on the given object 116
GET_HASH_VALUE function - Compute a hash value for a given string 117
GET_TIME function - Return the current time 118
NAME_RESOLVE procedure - Obtain the schema and other membership information for a database object 119
NAME_TOKENIZE procedure - Parse the given name into its component parts 123
TABLE_TO_COMMA procedures - Convert a table of names into a comma-delimited list of names 126
VALIDATE procedure - Change an invalid routine into a valid routine 128
MONREPORT module 128
CONNECTION procedure - Generate a report on connection metrics 130
CURRENTAPPS procedure - Generate a report of point-in-time application processing metrics . 141
CURRENTSQL procedure - Generate a report that summarizes activities 142
DBSUMMARY procedure - Generate a summary report of system and application performance metrics 142
LOCKWAIT procedure - Generate a report of current lock waits 147
PKGCACHE procedure - Generate a summary report of package cache metrics 150
UTL_DIR module 151
CREATE_DIRECTORY procedure - Create a directory alias 151
CREATE_OR_REPLACE_DIRECTORY procedure - Create or replace a directory alias . 152
DROP_DIRECTORY procedure - Drop a directory alias 153
GET_DIRECTORY_PATH procedure - Get the path for a directory alias 153
UTL_FILE module. 154
FCLOSE procedure - Close an open file . . . 155
FCLOSE_ALL procedure - Close all open files 156
FCOPY procedure - Copy text from one file to another 157
FFLUSH procedure - Flush unwritten data to a file 158
FOPEN function - Open a file 159
FREMOVE procedure - Remove a file 161
FRENAME procedure - Rename a file 161
GET_LINE procedure - Get a line from a file 162
IS_OPEN function - Determine whether a specified file is open 164
NEW_LINE procedure - Write an end-of-line character sequence to a file 165
PUT procedure - Write a string to a file . . . 166
PUT_LINE procedure - Write a line of text to a file 167
PUTF procedure - Write a formatted string to a file 169
UTL_FILE.FILE_TYPE 170
UTL_MAIL module 170
SEND procedure - Send an e-mail to an SMTP server 171
SEND_ATTACH_RAW procedure - Send an e-mail with a BLOB attachment to an SMTP server 172
SEND_ATTACH_VARCHAR2 procedure - Send an e-mail with a VARCHAR attachment to an SMTP server. 173
UTL_SMTP module 174
CLOSE_DATA procedure - End an e-mail message 177
COMMAND procedure - Run an SMTP command 177
COMMAND_REPLIES procedure - Run an SMTP command where multiple reply lines are expected 178
DATA procedure - Specify the body of an e-mail message 178
EHLO procedure - Perform initial handshaking with an SMTP server and return extended information 179
HELO procedure - Perform initial handshaking with an SMTP server 180
HELP procedure - Send the HELP command 180
MAIL procedure - Start a mail transaction . . . 181
NOOP procedure - Send the null command . . 181
OPEN_CONNECTION function - Return a connection handle to an SMTP server 182
OPEN_CONNECTION procedure - Open a connection to an SMTP server 183
OPEN_DATA procedure - Send the DATA command to the SMTP server 183
QUIT procedure - Close the session with the SMTP server. 184
RCPT procedure - Provide the e-mail address of the recipient 184
RSET procedure - End the current mail transaction 185
VRFY procedure - Validate and verify the recipient's e-mail address 185
WRITE_DATA procedure - Write a portion of an e-mail message 186
WRITE_RAW_DATA procedure - Add RAW data to an e-mail message 186

Index 189

Figures

1. Sample MONREPORT.LOCKWAIT output - summary section 148
2. Sample MONREPORT.LOCKWAIT output - details section 149

Tables

1. Built-in routines available in the DBMS_ALERT module 1
2. Built-in routines available in the DBMS_DDL module 9
3. Built-in routines available in the DBMS_JOB module 13
4. Built-in constants available in the DBMS_JOB module 13
5. Built-in routines available in the DBMS_LOB module 20
6. DBMS_LOB public variables 21
7. Built-in routines available in the DBMS_OUTPUT module 32
8. Built-in routines available in the DBMS_PIPE module 39
9. NEXT_ITEM_TYPE data type codes 43
10. RECEIVE_MESSAGE status codes 47
11. REMOVE_PIPE status codes 48
12. SEND_MESSAGE status codes 51
13. Built-in routines available in the DBMS_RANDOM module 54
14. Built-in routines available in the DBMS_SQL module 58
15. DBMS_SQL built-in types and constants 61
16. DESC_TAB definition through DESC_REC records 80
17. DESC_TAB2 definition through DESC_REC2 records 83
18. Built-in routines available in the DBMS_UTILITY module 102
19. DBMS_UTILITY public variables 103
20. Built-in routines available in the MONREPORT module 128
21. Built-in routines available in the UTL_DIR module 151
22. Built-in routines available in the UTL_FILE module 154
23. Named conditions for an application 155
24. Built-in routines available in the UTL_MAIL module 170
25. Built-in routines available in the UTL_SMTP module 174
26. Built-in types available in the UTL_SMTP module 175

Built-in modules

The built-in modules provide an easy-to-use programmatic interface for performing a variety of useful operations.

For example, you can use built-in modules to perform the following functions:

- Send and receive messages and alerts across connections.
- Write to and read from files and directories on the operating system's file system.
- Generate reports containing a variety of monitor information.

Built-in modules can be invoked from an SQL-based application, theDb2® command line, or a command script.

Built-in modules transform string data according to the database code page setting.

Built-in modules are not supported for the following product editions:

- Db2 Express-C

DBMS_ALERT module

The DBMS_ALERT module provides a set of procedures for registering for alerts, sending alerts, and receiving alerts.

Alerts are stored in SYSTOOLS.DBMS_ALERT_INFO, which is created in the SYSTOOLSPACE when you first reference this module for each database.

The schema for this module is SYSIBMADM.

The DBMS_ALERT module includes the following built-in routines.

Table 1. Built-in routines available in the DBMS_ALERT module

Routine name	Description
REGISTER procedure	Registers the current session to receive a specified alert.
REMOVE procedure	Removes registration for a specified alert.
REMOVEALL procedure	Removes registration for all alerts.
SIGNAL procedure	Signals the occurrence of a specified alert.
SET_DEFAULTS procedure	Sets the polling interval for the WAITONE and WAITANY procedures.
WAITANY procedure	Waits for any registered alert to occur.
WAITONE procedure	Waits for a specified alert to occur.

Usage notes

The procedures in the DBMS_ALERT module are useful when you want to send an alert for a specific event. For example, you might want to send an alert when a trigger is activated as the result of changes to one or more tables.

The DBMS_ALERT module requires that the database configuration parameter CUR_COMMIT is set to ON

For the SYSTOOLS.DBMS_ALERT_INFO table to be created successfully in the SYSTOOLSPACE table space, ensure that you have CREATETAB authority if you are running the DBMS_ALERT module for the first time.

Example

When a trigger, TRIG1, is activated, send an alert from connection 1 to connection 2 . First, create the table and the trigger.

```
CREATE TABLE T1 (C1 INT)@

CREATE TRIGGER TRIG1
AFTER INSERT ON T1
REFERENCING NEW AS NEW
FOR EACH ROW
BEGIN ATOMIC
CALL DBMS_ALERT.SIGNAL( 'trig1', NEW.C1 );
END@
```

From connection 1, issue an INSERT statement.

```
INSERT INTO T1 values (10)@
-- Commit to send messages to the listeners (required in early program)
CALL DBMS_ALERT.COMMIT()@
```

From connection 2, register to receive the alert called trig1 and wait for the alert.

```
CALL DBMS_ALERT.REGISTER('trig1')@
CALL DBMS_ALERT.WAITONE('trig1', ?, ?, 5)@
```

This example results in the following output:

```
  Value of output parameters
  --------------------------
  Parameter Name  : MESSAGE
  Parameter Value : -

  Parameter Name  : STATUS
  Parameter Value : 1

  Return Status = 0
```

REGISTER procedure - Register to receive a specified alert

The REGISTER procedure registers the current session to receive a specified alert.

Syntax

Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of the alert.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

Use the REGISTER procedure to register for an alert named alert_test, and then wait for the signal.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_name    VARCHAR(30) DEFAULT 'alert_test';
  DECLARE v_msg     VARCHAR(80);
  DECLARE v_status  INTEGER;
  DECLARE v_timeout INTEGER DEFAULT 5;
  CALL DBMS_ALERT.REGISTER(v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  CALL DBMS_ALERT.WAITONE(v_name , v_msg , v_status , v_timeout);
  CALL DBMS_OUTPUT.PUT_LINE('Alert name   : ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
  CALL DBMS_ALERT.REMOVE(v_name);
END@

CALL proc1@
```

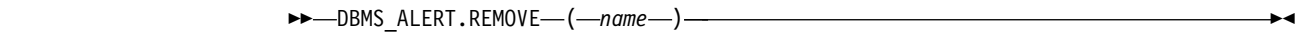
This example results in the following output:

```
Waiting for signal...
Alert name   : alert_test
Alert status : 1
```

REMOVE procedure - Remove registration for a specified alert

The REMOVE procedure removes registration from the current session for a specified alert.

Syntax



Procedure parameters

name

An input argument of type VARCHAR(128) that specifies the name of the alert.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

Use the REMOVE procedure to remove an alert named alert_test.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_name    VARCHAR(30) DEFAULT 'alert_test';
  DECLARE v_msg     VARCHAR(80);
  DECLARE v_status  INTEGER;
  DECLARE v_timeout INTEGER DEFAULT 5;
  CALL DBMS_ALERT.REGISTER(v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  CALL DBMS_ALERT.WAITONE(v_name , v_msg , v_status , v_timeout);
  CALL DBMS_OUTPUT.PUT_LINE('Alert name   : ' || v_name);
```

```
  CALL DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
  CALL DBMS_ALERT.REMOVE(v_name);
END@

CALL proc1@
```

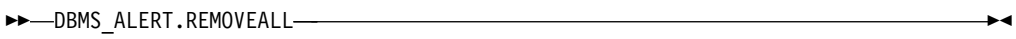
This example results in the following output:

```
Waiting for signal...
Alert name   : alert_test
Alert status : 1
```

REMOVEALL procedure - Remove registration for all alerts

The REMOVEALL procedure removes registration from the current session for all alerts.

Syntax



Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

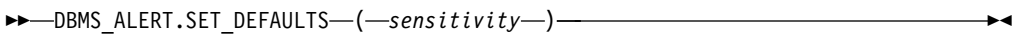
Use the REMOVEALL procedure to remove registration for all alerts.

```
CALL DBMS_ALERT.REMOVEALL@
```

SET_DEFAULTS - Set the polling interval for WAITONE and WAITANY

The SET_DEFAULTS procedure sets the polling interval that is used by the WAITONE and WAITANY procedures.

Syntax



Procedure parameters

sensitivity
An input argument of type INTEGER that specifies an interval in seconds for the WAITONE and WAITANY procedures to check for signals. If a value is not specified, then the interval is 1 second by default.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

Use the SET_DEFAULTS procedure to specify the polling interval for the WAITONE and WAITANY procedures.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_name    VARCHAR(30) DEFAULT 'alert_test';
  DECLARE v_msg     VARCHAR(80);
  DECLARE v_status  INTEGER;
  DECLARE v_timeout INTEGER DEFAULT 20;
  DECLARE v_polling INTEGER DEFAULT 3;
  CALL DBMS_ALERT.REGISTER(v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  CALL DBMS_ALERT.SET_DEFAULTS(v_polling);
  CALL DBMS_OUTPUT.PUT_LINE('Polling interval: ' || v_polling);
  CALL DBMS_ALERT.WAITONE(v_name , v_msg , v_status , v_timeout);
  CALL DBMS_ALERT.REMOVE(v_name);
END@

CALL proc1@
```

This example results in the following output:

```
Polling interval : 3
```

SIGNAL procedure - Signal occurrence of a specified alert

The SIGNAL procedure signals the occurrence of a specified alert. The signal includes a message that is passed with the alert. The message is distributed to the listeners (processes that have registered for the alert) when the SIGNAL call is issued.

Syntax

Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of the alert.

message
An input argument of type VARCHAR(32672) that specifies the information to pass with this alert. This message can be returned by the WAITANY or WAITONE procedures when an alert occurs.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

Use the SIGNAL procedure to signal the occurrence of an alert named alert_test.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_name   VARCHAR(30) DEFAULT 'alert_test';
  CALL DBMS_ALERT.SIGNAL(v_name,'This is the message from ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END@

CALL proc1@
```

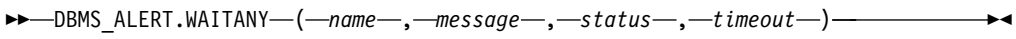
This example results in the following output:

```
Issued alert for alert_test
```

WAITANY procedure - Wait for any registered alerts

The WAITANY procedure waits for any registered alerts to occur.

Syntax



Procedure parameters

name
An output argument of type VARCHAR(128) that contains the name of the alert.

message
An output argument of type VARCHAR(32672) that contains the message sent by the SIGNAL procedure.

status
An output argument of type INTEGER that contains the status code returned by the procedure. The following values are possible

0 An alert occurred.

1 A timeout occurred.

timeout
An input argument of type INTEGER that specifies the amount of time in seconds to wait for an alert.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

From one connection, run a CLP script called `waitany.clp` to receive any registered alerts.

`waitany.clp`:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_name    VARCHAR(30);
  DECLARE v_msg     VARCHAR(80);
  DECLARE v_status  INTEGER;
  DECLARE v_timeout INTEGER DEFAULT 20;
  CALL DBMS_ALERT.REGISTER('alert_test');
  CALL DBMS_ALERT.REGISTER('any_alert');
  CALL DBMS_OUTPUT.PUT_LINE('Registered for alert alert_test and any_alert');
  CALL DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  CALL DBMS_ALERT.WAITANY(v_name , v_msg , v_status , v_timeout);
  CALL DBMS_OUTPUT.PUT_LINE('Alert name : ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Alert msg : ' || v_msg);
  CALL DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Alert timeout: ' || v_timeout || ' seconds');
```

```
  CALL DBMS_ALERT.REMOVEALL;
END@

call proc1@
```

From another connection, run a script called `signal.clp` to issue a signal for an alert named any_alert.

`signal.clp`:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc2
  BEGIN
  DECLARE v_name VARCHAR(30) DEFAULT 'any_alert';
  CALL DBMS_ALERT.SIGNAL(v_name,'This is the message from ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END@

CALL proc2@
```

The script `signal.clp` results in the following output:

```
Issued alert for any_alert
```

The script `waitany.clp` results in the following output:

```
Registered for alert alert_test and any_alert
Waiting for signal...
Alert name : any_alert
Alert msg : This is the message from any_alert
Alert status : 0
Alert timeout: 20 seconds
```

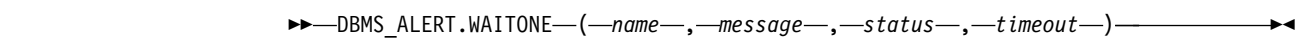
Usage notes

If no alerts are registered when the WAITANY procedure is called, the procedure returns SQL0443N.

WAITONE procedure - Wait for a specified alert

The WAITONE procedure waits for a specified alert to occur.

Syntax



Procedure parameters

name
An input argument of type VARCHAR(128) that specifies the name of the alert.

message
An output argument of type VARCHAR(32672) that contains the message sent by the SIGNAL procedure.

status
An output argument of type INTEGER that contains the status code returned by the procedure. The following values are possible

0 An alert occurred.

1 A timeout occurred.

timeout
An input argument of type INTEGER that specifies the amount of time in seconds to wait for the specified alert.

Authorization

EXECUTE privilege on the DBMS_ALERT module.

Example

Run a CLP script named `waitone.clp` to receive an alert named alert_test.

`waitone.clp`:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_name    VARCHAR(30) DEFAULT 'alert_test';
  DECLARE v_msg     VARCHAR(80);
  DECLARE v_status  INTEGER;
  DECLARE v_timeout INTEGER DEFAULT 20;
  CALL DBMS_ALERT.REGISTER(v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Waiting for signal...');
  CALL DBMS_ALERT.WAITONE(v_name , v_msg , v_status , v_timeout);
  CALL DBMS_OUTPUT.PUT_LINE('Alert name : ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Alert msg : ' || v_msg);
  CALL DBMS_OUTPUT.PUT_LINE('Alert status : ' || v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Alert timeout: ' || v_timeout || ' seconds');
  CALL DBMS_ALERT.REMOVE(v_name);
END@

CALL proc1@
```

From a different connection, run a script named `signalalert.clp` to issue a signal for an alert named alert_test.

`signalalert.clp`:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc2
BEGIN
  DECLARE v_name VARCHAR(30) DEFAULT 'alert_test';
  CALL DBMS_ALERT.SIGNAL(v_name,'This is the message from ' || v_name);
  CALL DBMS_OUTPUT.PUT_LINE('Issued alert for ' || v_name);
END@

CALL proc2@
```

The script `signalalert.clp` results in the following output:

```
Issued alert for alert_test
```

The script `waitone.clp` results in the following output:

```
Waiting for signal...
Alert name : alert_test
Alert msg : This is the message from alert_test
Alert status : 0
Alert timeout: 20 seconds
```

DBMS_DDL module

The DBMS_DDL module provides the capability to obfuscate DDL objects such as routines, triggers, views or PL/SQL packages. Obfuscation allows the deployment of SQL objects to a database without exposing the procedural logic.

The DDL statements for these objects are obfuscated both in vendor-provided install scripts as well as in the Db2 catalogs.

The schema for this module is SYSIBMADM.

The DBMS_DDL module includes the following routines.

Table 2. Built-in routines available in the DBMS_DDL module

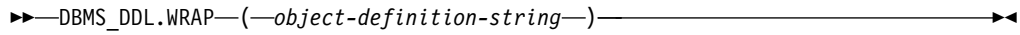
Routine name	Description
WRAP function	Produces an obfuscated version of the DDL statement provided as argument.
CREATE_WRAPPED procedure	Deploys a DDL statement in the database in an obfuscated format.

WRAP function - Obfuscate a DDL statement

The **WRAP** function transforms a readable DDL statement into an obfuscated DDL statement.

Syntax

In an obfuscated DDL statement, the procedural logic and embedded SQL statements are scrambled in such a way that any intellectual property in the logic cannot be easily extracted. If the DDL statement corresponds to an external routine definition, the portion following the parameter list is encoded.



Parameters

object-definition-string

A string of type CLOB(2M) containing a DDL statement text which can be one of the following (SQLSTATE 5UA0O):

- create procedure
- create function
- create package (PL/SQL)
- create package body (PL/SQL)
- create trigger
- create view
- alter module add function
- alter module publish function
- alter module add procedure
- alter module publish procedure

The result is a string of type CLOB(2M) which contains an encoded version of the input statement. The encoding consists of a prefix of the original statement

up to and including the routine signature or the trigger, view or package name, followed by the keyword WRAPPED. This keyword is followed by information about the application server that executes the function. The information has the form *pppvvrrm*, where:

- *ppp* identifies the product as Db2 using the letters SQL
- *vv* is a two-digit version identifier, such as '09'
- *rr* is a two-digit release identifier, such as '07'
- *m* is a one-character modification level identifier, such as '0'.

For example, Fixpack 2 of Version 9.7 is identified as 'SQL09072'. This application server information is followed by a string of letters (a-z, and A-Z), digits (0-9), underscores and colons. No syntax checking is done on the input statement beyond the prefix that remains readable after obfuscation.

The encoded DDL statement is typically longer than the plain text form of the statement. If the result exceeds the maximum length for SQL statements an error is raised (SQLSTATE 54001).

Note: The encoding of the statement is meant to obfuscate the content and should not be considered as a form of strong encryption.

Authorization

EXECUTE privilege on the DBMS_DDL module

Example

- Produce an obfuscated version of a function that computes a yearly salary from an hourly wage given a 40 hour workweek

```
VALUES(DBMS_DDL.WRAP('CREATE FUNCTION ' ||
                     'salary(wage DECFLOAT) ' ||
                     'RETURNS DECFLOAT ' ||
                     'RETURN wage * 40 * 52'))
The result of the previous statement would be something of the form:

CREATE FUNCTION salary(wage DECFLOAT) WRAPPED SQL09072 obfuscated-text
```

- Produce an obfuscated form of a trigger setting a complex default

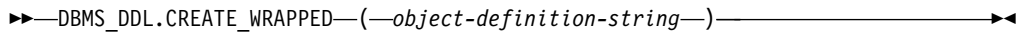
```
VALUES(DBMS_DDL.WRAP('CREATE OR REPLACE TRIGGER ' ||
                      'trg1 BEFORE INSERT ON emp ' ||
                      'REFERENCING NEW AS n ' ||
                      'FOR EACH ROW ' ||
                      'WHEN (n.bonus IS NULL) ' ||
                      'SET n.bonus = n.salary * .04'))

The result of the previous statement would be something of the form:

CREATE OR REPLACE TRIGGER trg1 WRAPPED SQL09072 obfuscated-text
```

CREATE_WRAPPED procedure - Deploy an obfuscated object

The **CREATE_WRAPPED** procedure transforms a plain text DDL object definition into an obfuscated DDL object definition and then deploys the object in the database.

Syntax

In an obfuscated DDL statement, the procedural logic and embedded SQL statements are encoded in such a way that any intellectual property in the logic cannot be easily extracted.



Parameters

object-definition-string
A string of type CLOB(2M) containing a DDL statement text which can be one of the following (SQLSTATE 5UA0O):

- create procedure
- create function
- create package (PL/SQL)
- create package body (PL/SQL)
- create trigger
- create view
- alter module add function
- alter module publish function
- alter module add procedure
- alter module publish procedure

The procedure transforms the input into an obfuscated DDL statement string and then dynamically executes that DDL statement. Special register values such as PATH and CURRENT SCHEMA in effect at invocation as well as the current invoker's rights are being used.

The encoding consists of a prefix of the original statement up to and including the routine signature or the trigger, view or package name, followed by the keyword **WRAPPED**. This keyword is followed by information about the application server that executes the procedure. The information has the form "*pppvvrrm*," where:

- *ppp* identifies the product as Db2 using the letters SQL
- *vv* is a two-digit version identifier, such as '09'
- *rr* is a two-digit release identifier, such as '07'
- *m* is a one-character modification level identifier, such as '0'.

For example, Fixpack 2 of Version 9.7 is identified as 'SQL09072'. This application server information is followed by a string of letters (a-z, and A-Z), digits (0-9), underscores and colons. No syntax checking is done on the input statement beyond the prefix that remains readable after obfuscation.

The encoded DDL statement is typically longer than the plain text form of the statement. If the result exceeds the maximum length for SQL statements an error is raised (SQLSTATE 54001).

Note: The encoding of the statement is meant to obfuscate the content and should not be considered as a form of strong encryption.

Authorization

EXECUTE privilege on the DBMS_DDL module.

Example

- Create an obfuscated function computing a yearly salary from an hourly wage given a 40 hour workweek

```
CALL DBMS_DDL.CREATE_WRAPPED('CREATE FUNCTION ' ||
                   'salary(wage DECFLOAT) ' ||
                   'RETURNS DECFLOAT ' ||
                   'RETURN wage * 40 * 52');
SELECT text FROM SYSCAT.ROUTINES
 WHERE routinename = 'SALARY'
    AND routineschema = CURRENT SCHEMA;
```

Upon successful execution of the CALL statement, The SYSCAT.ROUTINES.TEXT column for the row corresponding to routine 'SALARY' would be something of the form:

```
CREATE FUNCTION salary(wage DECFLOAT) WRAPPED SQL09072 obfuscated-text
```

- Create an obfuscated trigger setting a complex default

```
CALL DBMS_DDL.CREATE_WRAPPED('CREATE OR REPLACE TRIGGER ' ||
                   'trg1 BEFORE INSERT ON emp ' ||
                   'REFERENCING NEW AS n ' ||
                   'FOR EACH ROW ' ||
                   'WHEN (n.bonus IS NULL) ' ||
                   'SET n.bonus = n.salary * .04');
SELECT text FROM SYSCAT.TRIGGERS
 WHERE trigname = 'TRG1'
   AND trigschema = CURRENT SCHEMA;
```

Upon successful execution of the CALL statement, The SYSCAT.TRIGGERS.TEXT column for the row corresponding to trigger 'TRG1' would be something of the form:

```
CREATE OR REPLACE TRIGGER trg1 WRAPPED SQL09072  obfuscated-text
```

DBMS_JOB module

The DBMS_JOB module provides procedures for the creation, scheduling, and managing of jobs.

The DBMS_JOB module provides an alternate interface for the Administrative Task Scheduler (ATS). A job is created by adding a task to the ATS. The actual task name is constructed by concatenating the DBMS_JOB.TASK_NAME_PREFIX procedure name with the assigned job identifier, such as SAMPLE_JOB_TASK_1 where 1 is the job identifier.

A job runs a stored procedure which has been previously stored in the database. The SUBMIT procedure is used to create and store a job definition. A job identifier is assigned to every job, along with its associated stored procedure and the attributes describing when and how often the job is run.

On first run of the SUBMIT procedure in a database, the SYSTOOLSPACE table space is created if necessary.

To enable job scheduling for the DBMS_JOB routines, run:

```
db2set DB2_ATS_ENABLE=1
```

When and how often a job runs depends upon two interacting parameters - **next_date** and **interval**. The **next_date** parameter is a datetime value that specifies the next date and time when the job is to be executed. The **interval** parameter is a string that contains a date function that evaluates to a datetime value. Just prior to any execution of the job, the expression in the **interval** parameter is evaluated, and the resulting value replaces the **next_date** value stored with the job. The job is then executed. In this manner, the expression in **interval** is re-evaluated prior to each job execution, supplying the **next_date** date and time for the next execution.

The first run of a scheduled job, as specified by the **next_date** parameter, should be set at least 5 minutes after the current time, and the interval between running each job should also be at least 5 minutes.

The schema for this module is SYSIBMADM.

The DBMS_JOB module includes the following built-in routines.

Table 3. Built-in routines available in the DBMS_JOB module

Routine name	Description
BROKEN procedure	Specify that a given job is either broken or not broken.
CHANGE procedure	Change the parameters of the job.
INTERVAL procedure	Set the execution frequency by means of a date function that is recalculated each time the job runs. This value becomes the next date and time for execution.
NEXT_DATE procedure	Set the next date and time when the job is to be run.
REMOVE procedure	Delete the job definition from the database.
RUN procedure	Force execution of a job even if it is marked as broken.
SUBMIT procedure	Create a job and store the job definition in the database.
WHAT procedure	Change the stored procedure run by a job.

Table 4. Built-in constants available in the DBMS_JOB module

Constant name	Description
ANY_INSTANCE	The only supported value for the instance argument for the DBMS_JOB routines.
TASK_NAME_PREFIX	This constant contains the string that is used as the prefix for constructing the task name for the administrative task scheduler.

Usage notes

When the first job is submitted through the DBMS_JOB module for each database, the Administrative Task Scheduler setup is performed:

1. Create the SYSTOOLSPACE table space if it does not exist;
2. Create the ATS table and views, such as SYSTOOLS.ADMIN_TASK_LIST.

To list the scheduled jobs, run:

```
db2 SELECT * FROM systools.admin_task_list
      WHERE name LIKE DBMS_JOB.TASK_NAME_PREFIX || '_%'
```

To view the status of the job execution, run:

```
db2 SELECT * FROM systools.admin_task_status
      WHERE name LIKE DBMS_JOB.TASK_NAME_PREFIX || '_%'
```

Note: The expected value for job identifier is not the value of TASKID that is returned by SYSTOOLS.ADMIN_TASK_LIST. For example, you have the following job list:

```
NAME                    TASKID
---------------         ------
DBMS_JOB_TASK_2              3
DBMS_JOB_TASK_3              4
```

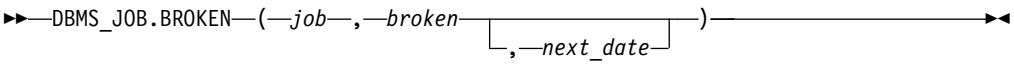
If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

BROKEN procedure - Set the state of a job to either broken or not broken

The BROKEN procedure sets the state of a job to either broken or not broken.

A broken job cannot be executed except by using the RUN procedure.

Syntax



Parameters

job
An input argument of type DECIMAL(20) that specifies the identifier of the job to be set as broken or not broken.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                    TASKID
---------------         ------
DBMS_JOB_TASK_2              3
DBMS_JOB_TASK_3              4
```

If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

broken
An input argument of type BOOLEAN that specifies the job status. If set to "true", the job state is set to `broken`. If set to "false", the job state is set to `not broken`. Broken jobs cannot be run except through the RUN procedure.

next_date
An optional input argument of type DATE that specifies the date and time when the job runs. The default is `SYSDATE`.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: Set the state of a job with job identifier 104 to broken:

```
CALL DBMS_JOB.BROKEN(104,true);
```

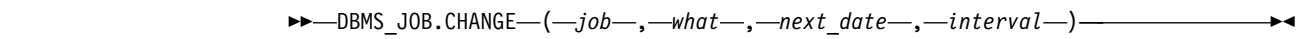
Example 2: Change the state back to not broken:

```
CALL DBMS_JOB.BROKEN(104,false);
```

CHANGE procedure - Modify job attributes

The CHANGE procedure modifies certain job attributes, including the executable SQL statement, the next date and time the job is run, and how often it is run.

Syntax



Parameters

job

An input argument of type DECIMAL(20) that specifies the identifier of the job with the attributes to modify.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                    TASKID
---------------         ------
DBMS_JOB_TASK_2              3
DBMS_JOB_TASK_3              4
```

If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

what

An input argument of type VARCHAR(1024) that specifies the executable SQL statement. Set this argument to NULL if the existing value is to remain unchanged.

next_date

An input argument of type TIMESTAMP(0) that specifies the next date and time when the job is to run. Set this argument to NULL if the existing value is to remain unchanged.

interval

An input argument of type VARCHAR(1024) that specifies the date function that, when evaluated, provides the next date and time the job is to run. Set this argument to NULL if the existing value is to remain unchanged.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

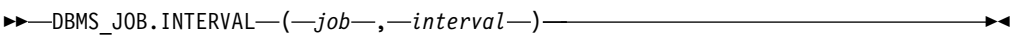
Example 1: Change the job to run next on December 13, 2009. Leave other parameters unchanged.

```
CALL DBMS_JOB.CHANGE(104,NULL,TO_DATE('13-DEC-09','DD-MON-YY'),NULL);
```

INTERVAL procedure - Set run frequency

The INTERVAL procedure sets the frequency of how often a job is run.

Syntax



Parameters

job
An input argument of type DECIMAL(20) that specifies the identifier of the job whose frequency is being changed.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                TASKID
---------------     ------
DBMS_JOB_TASK_2          3
DBMS_JOB_TASK_3          4
```

If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

interval
An input argument of type VARCHAR(1024) that specifies the date function that, when evaluated, provides the next date and time the job is to run.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

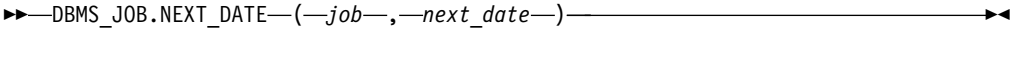
Example 1: Change the job to run once a week:

```
CALL DBMS_JOB.INTERVAL(104,'SYSDATE + 7');
```

NEXT_DATE procedure - Set the date and time when a job is run

The NEXT_DATE procedure sets the next date and time of when the job is to run.

Syntax



Parameters

job
An input argument of type DECIMAL(20) that specifies the identifier of the job whose next run date is to be modified.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                TASKID
---------------     ------
DBMS_JOB_TASK_2          3
DBMS_JOB_TASK_3          4
```

If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

next_date
An input argument of type TIMESTAMP(0) that specifies the date and time when the job is to be run next.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: Change the job to run next on December 14, 2009:

```
CALL DBMS_JOB.NEXT_DATE(104, TO_DATE('14-DEC-09','DD-MON-YY'));
```

REMOVE procedure - Delete the job definition from the database

The REMOVE procedure deletes the specified job from the database.

In order to have it executed again in the future, the job must be resubmitted using the SUBMIT procedure.

Note: The stored procedure associated with the job is not deleted when the job is removed.

Syntax

Parameters

job
An input argument of type DECIMAL(20) that specifies the identifier of the job to be removed from the database.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                 TASKID
---------------      ------
DBMS_JOB_TASK_2           3
DBMS_JOB_TASK_3           4
```

If you want to remove DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: Remove a job from the database:

```
CALL DBMS_JOB.REMOVE(104);
```

RUN procedure - Force a broken job to run

The RUN procedure forces a job to run, even if it has a broken state.

Syntax

Parameters

job

An input argument of type DECIMAL(20) that specifies the identifier of the job to run.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                TASKID
---------------     ------
DBMS_JOB_TASK_2          3
DBMS_JOB_TASK_3          4
```

If you want to run DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: Force a job to run.

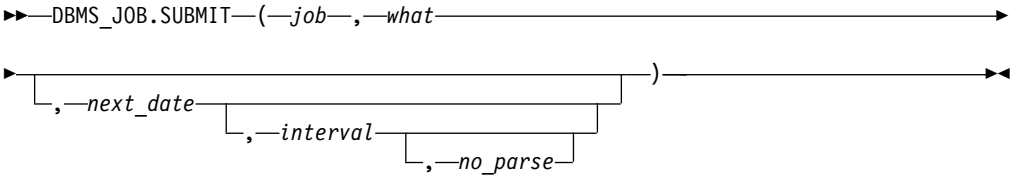
```
CALL DBMS_JOB.RUN(104);
```

SUBMIT procedure - Create a job definition and store it in the database

The SUBMIT procedure creates a job definition and stores it in the database.

A job consists of a job identifier, the stored procedure to be executed, when the job is first executed, and a date function that calculates the next date and time for the job to be run.

Syntax



Parameters

job

An output argument of type DECIMAL(20) that specifies the identifier assigned to the job.

what
An input argument of type VARCHAR(1024) that specifies the name of the dynamically executable SQL statement.

next_date
An optional input argument of type TIMESTAMP(0) that specifies the next date and time when the job is to be run. The default is SYSDATE.

interval
An optional input argument of type VARCHAR(1024) that specifies a date function that, when evaluated, provides the date and time of the execution after the next execution. If *interval* is set to NULL, then the job is run only once. NULL is the default.

no_parse
An optional input argument of type BOOLEAN. If set to true, do not syntax-check the SQL statement at job creation; instead, perform syntax checking only when the job first executes. If set to false, syntax check the SQL statement at job creation. The default is false.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: The following example creates a job using the stored procedure, job_proc. The job will first execute in about 5 minutes, and runs once a day thereafter as set by the *interval* argument, SYSDATE + 1.

```
SET SERVEROUTPUT ON@

BEGIN
  DECLARE jobid           INTEGER;
  CALL DBMS_JOB.SUBMIT(jobid,'CALL job_proc();',SYSDATE + 5 minutes, 'SYSDATE + 1');
  CALL DBMS_OUTPUT.PUT_LINE('jobid: ' || jobid);
END@
```

The output from this command is as follows:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

BEGIN
  DECLARE jobid           INTEGER;
  CALL DBMS_JOB.SUBMIT(jobid,'CALL job_proc();',SYSDATE + 5 minutes, 'SYSDATE + 1');
  CALL DBMS_OUTPUT.PUT_LINE('jobid: ' || jobid);
END
DB20000I  The SQL command completed successfully.

jobid: 1
```

WHAT procedure - Change the SQL statement run by a job

The WHAT procedure changes the SQL statement run by a specified job.

Syntax

Parameters

job
An input argument of type DECIMAL(20) that specifies the job identifier for which the dynamically executable SQL statement is to be changed.

Note: The expected value for job identifier is not the value of TASKID in the SYSTOOLS.ADMIN_TASK_LIST view. For example, you have the following job list:

```
NAME                    TASKID
---------------         ------
DBMS_JOB_TASK_2              3
DBMS_JOB_TASK_3              4
```

If you want to modify DBMS_JOB_TASK_2, you must pass 2 as the job identifier.

what
An input argument of type VARCHAR(1024) that specifies the dynamically executed SQL statement.

Authorization

EXECUTE privilege on the DBMS_JOB module.

Examples

Example 1: Change the job to run the `list_emp` procedure:

```
CALL DBMS_JOB.WHAT(104,'list_emp;');
```

DBMS_LOB module

The DBMS_LOB module provides the capability to operate on large objects.

In the following sections describing the individual procedures and functions, lengths and offsets are measured in bytes if the large objects are BLOBs. Lengths and offsets are measured in characters if the large objects are CLOBs.

The DBMS_LOB module supports LOB data up to 10M bytes.

The schema for this module is SYSIBMADM.

The DBMS_LOB module includes the following routines which can contain BLOB and CLOB versions (for example, the OPEN procedure has an OPEN_BLOB and OPEN_CLOB implementation).

Table 5. Built-in routines available in the DBMS_LOB module

Routine Name	Description
APPEND procedure	Appends one large object to another.
CLOSE procedure	Close an open large object.
COMPARE function	Compares two large objects.
CONVERTTOBLOB procedure	Converts character data to binary.
CONVERTTOCLOB procedure	Converts binary data to character.
COPY procedure	Copies one large object to another.

Table 5. Built-in routines available in the DBMS_LOB module (continued)

Routine Name	Description
ERASE procedure	Erase a large object.
GET_STORAGE_LIMIT function	Get the storage limit for large objects.
GETLENGTH function	Get the length of the large object.
INSTR function	Get the position of the nth occurrence of a pattern in the large object starting at offset.
ISOPEN function	Check if the large object is open.
OPEN procedure	Open a large object.
READ procedure	Read a large object.
SUBSTR function	Get part of a large object.
TRIM procedure	Trim a large object to the specified length.
WRITE procedure	Write data to a large object.
WRITEAPPEND procedure	Write data from the buffer to the end of a large object.

Note: In partitioned database environments, you will receive an error if you execute any of the following routines inside a WHERE clause of a SELECT statement:

- dbms_lob.compare
- dbms_lob.get_storage_limit
- dbms_lob.get_length
- dbms_lob.instr
- dbms_lob.isopen
- dbms_lob.substr

The following table lists the public variables available in the module.

Table 6. DBMS_LOB public variables

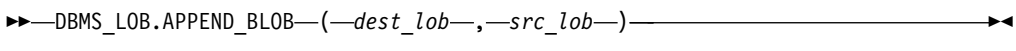
Public variables	Data type	Value
lob_readonly	INTEGER	0
lob_readwrite	INTEGER	1

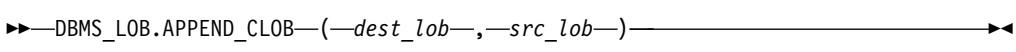
APPEND procedures - Append one large object to another

The APPEND procedures provide the capability to append one large object to another.

Note: Both large objects must be of the same type.

Syntax





Parameters

dest_lob
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator for the destination object. Must be the same data type as *src_lob*.

src_lob
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator for the source object. Must be the same data type as *dest_lob*.

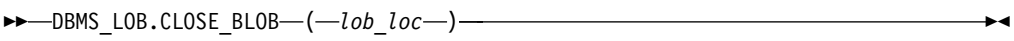
Authorization

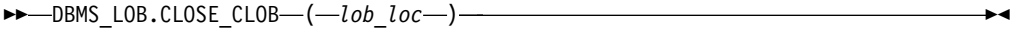
EXECUTE privilege on the DBMS_LOB module.

CLOSE procedures - Close an open large object

The CLOSE procedures are a no-op.

Syntax





Parameters

lob_loc
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be closed.

Authorization

EXECUTE privilege on the DBMS_LOB module.

COMPARE function - Compare two large objects

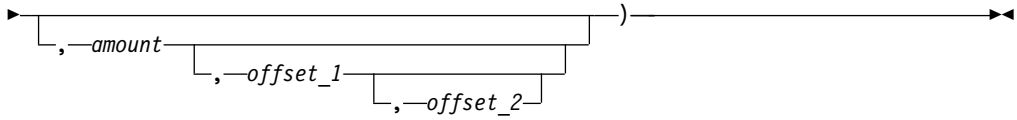
The COMPARE function performs an exact byte-by-byte comparison of two large objects for a given length at given offsets.

The function returns:

- Zero if both large objects are exactly the same for the specified length for the specified offsets
- Non-zero if the objects are not the same
- Null if *amount*, *offset_1*, or *offset_2* are less than zero.

Note: The large objects being compared must be the same data type.

Syntax



Parameters

lob_1
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the first large object to be compared. Must be the same data type as *lob_2*.

lob_2
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the second large object to be compared. Must be the same data type as *lob_1*.

amount
An optional input argument of type INTEGER. If the data type of the large objects is BLOB, then the comparison is made for *amount* bytes. If the data type of the large objects is CLOB, then the comparison is made for *amount* characters. The default is the maximum size of a large object.

offset_1
An optional input argument of type INTEGER that specifies the position within the first large object to begin the comparison. The first byte (or character) is offset 1. The default is 1.

offset_2
An optional input argument of type INTEGER that specifies the position within the second large object to begin the comparison. The first byte (or character) is offset 1. The default is 1.

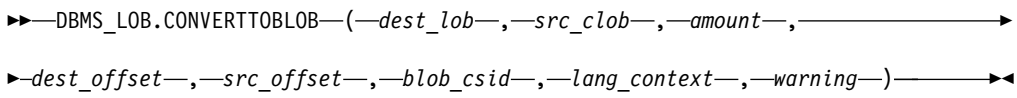
Authorization

EXECUTE privilege on the DBMS_LOB module.

CONVERTTOBLOB procedure - Convert character data to binary

The CONVERTTOBLOB procedure provides the capability to convert character data to binary.

Syntax



Parameters

dest_lob
An input or output argument of type BLOB(10M) that specifies the large object locator into which the character data is to be converted.

src_clob
An input argument of type CLOB(10M) that specifies the large object locator of the character data to be converted.

amount
An input argument of type INTEGER that specifies the number of characters of *src_clob* to be converted.

dest_offset
An input or output argument of type INTEGER that specifies the position (in bytes) in the destination BLOB where writing of the source CLOB should begin. The first byte is offset 1.

src_offset
An input or output argument of type INTEGER that specifies the position (in characters) in the source CLOB where conversion to the destination BLOB should begin. The first character is offset 1.

blob_csid
An input argument of type INTEGER that specifies the character set ID of the destination BLOB. This value must match the database codepage.

lang_context
An input argument of type INTEGER that specifies the language context for the conversion. This value must be 0.

warning
An output argument of type INTEGER that always returns 0.

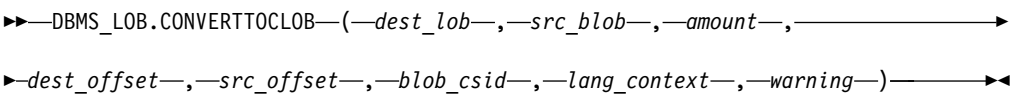
Authorization

EXECUTE privilege on the DBMS_LOB module.

CONVERTTOCLOB procedure - Convert binary data to character

The CONVERTTOCLOB procedure provides the capability to convert binary data to character.

Syntax



Parameters

dest_lob
An input or output argument of type CLOB(10M) that specifies the large object locator into which the binary data is to be converted.

src_clob
An input argument of type BLOB(10M) that specifies the large object locator of the binary data to be converted.

amount
An input argument of type INTEGER that specifies the number of characters of *src_blob* to be converted.

dest_offset
An input or output argument of type INTEGER that specifies the position (in characters) in the destination CLOB where writing of the source BLOB should begin. The first byte is offset 1.

src_offset
An input or output argument of type INTEGER that specifies the position (in bytes) in the source BLOB where conversion to the destination CLOB should begin. The first character is offset 1.

blob_csid
An input argument of type INTEGER that specifies the character set ID of the source BLOB. This value must match the database codepage.

lang_context
An input argument of type INTEGER that specifies the language context for the conversion. This value must be 0.

warning
An output argument of type INTEGER that always returns 0.

Authorization

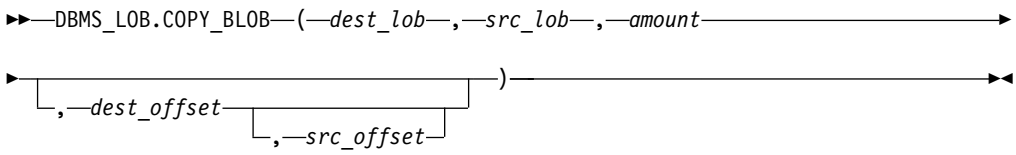
EXECUTE privilege on the DBMS_LOB module.

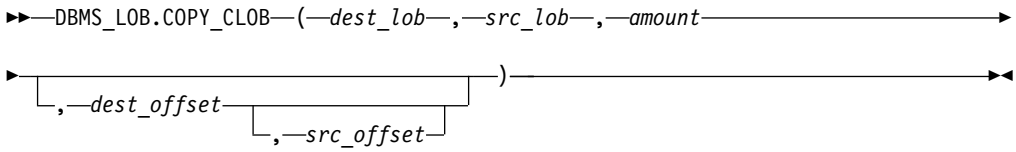
COPY procedures - Copy one large object to another

The COPY procedures provide the capability to copy one large object to another.

Note: The source and destination large objects must be the same data type.

Syntax





Parameters

dest_lob
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to which *src_lob* is to be copied. Must be the same data type as *src_lob*.

src_lob
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object from which *dest_lob* is to be copied. Must be the same data type as *dest_lob*.

amount
An input argument of type INTEGER that specifies the number of bytes or characters of *src_lob* to be copied.

dest_offset
An optional input argument of type INTEGER that specifies the position in the

destination large object where writing of the source large object should begin. The first position is offset 1. The default is 1.

src_offset
An optional input argument of type INTEGER that specifies the position in the source large object where copying to the destination large object should begin. The first position is offset 1. The default is 1.

Authorization

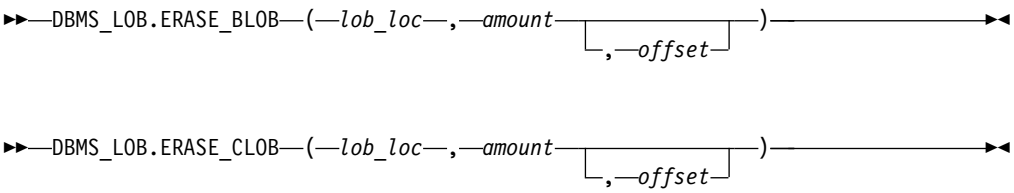
EXECUTE privilege on the DBMS_LOB module.

ERASE procedures - Erase a portion of a large object

The ERASE procedures provide the capability to erase a portion of a large object.

To erase a large object means to replace the specified portion with zero-byte fillers for BLOBs or with spaces for CLOBs. The actual size of the large object is not altered.

Syntax



Parameters

lob_loc
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be erased.

amount
An input or output argument of type INTEGER that specifies the number of bytes or characters to be erased.

offset
An optional input argument of type INTEGER that specifies the position in the large object where erasing is to begin. The first byte or character is at position 1. The default is 1.

Authorization

EXECUTE privilege on the DBMS_LOB module.

GET_STORAGE_LIMIT function - Return the limit on the largest allowable large object

The GET_STORAGE_LIMIT function returns the limit on the largest allowable large object.

The function returns an INTEGER value that reflects the maximum allowable size of a large object in this database.

Syntax

Authorization

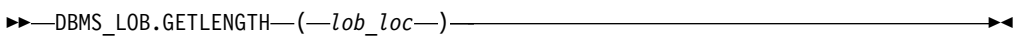
EXECUTE privilege on the DBMS_LOB module.

GETLENGTH function - Return the length of the large object

The GETLENGTH function returns the length of a large object.

The function returns an INTEGER value that reflects the length of the large object in bytes (for a BLOB) or characters (for a CLOB).

Syntax



Parameters

lob_loc
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object whose length is to be obtained.

Authorization

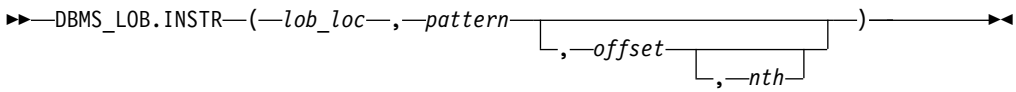
EXECUTE privilege on the DBMS_LOB module.

INSTR function - Return the location of the *n*th occurrence of a given pattern

The INSTR function returns the location of the *n*th occurrence of a given pattern within a large object.

The function returns an INTEGER value of the position within the large object where the pattern appears for the nth time, as specified by *nth*. This value starts from the position given by *offset*.

Syntax



Parameters

lob_loc
An input argument of type BLOB or CLOB that specifies the large object locator of the large object in which to search for the *pattern*.

pattern
An input argument of type BLOB(32767) or VARCHAR(32672) that specifies

the pattern of bytes or characters to match against the large object. Note that *pattern* must be BLOB if *lob_loc* is a BLOB; and *pattern* must be VARCHAR if *lob_loc* is a CLOB.

offset
An optional input argument of type INTEGER that specifies the position within *lob_loc* to start searching for the *pattern*. The first byte or character is at position 1. The default value is 1.

nth
An optional argument of type INTEGER that specifies the number of times to search for the *pattern*, starting at the position given by *offset*. The default value is 1.

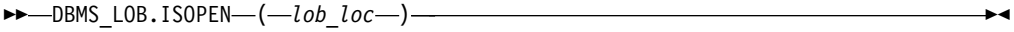
Authorization

EXECUTE privilege on the DBMS_LOB module.

ISOPEN function - Test if the large object is open

The ISOPEN function always returns an INTEGER value of 1..

Syntax



Parameters

lob_loc
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be tested by the function.

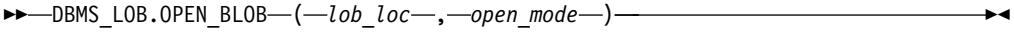
Authorization

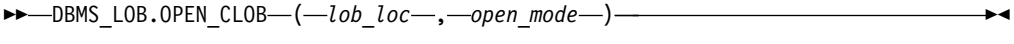
EXECUTE privilege on the DBMS_LOB module.

OPEN procedures - Open a large object

The OPEN procedures are a no-op.

Syntax





Parameters

lob_loc
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be opened.

open_mode
An input argument of type INTEGER that specifies the mode in which to open the large object. Set to 0 (`lob_readonly`) for read-only mode. Set to 1 (`lob_readwrite`) for read-write mode.

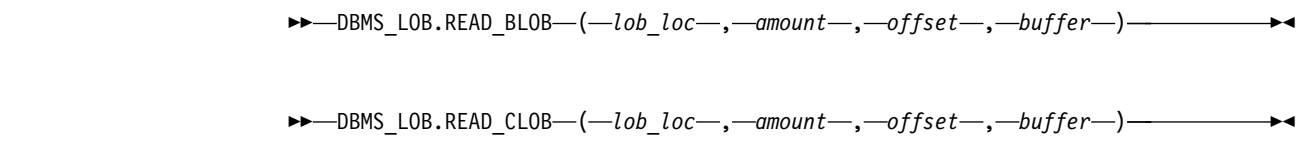
Authorization

EXECUTE privilege on the DBMS_LOB module.

READ procedures - Read a portion of a large object

The READ procedures provide the capability to read a portion of a large object into a buffer.

Syntax



Parameters

lob_loc
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be read.

amount
An input or output argument of type INTEGER that specifies the number of bytes or characters to read.

offset
An input argument of type INTEGER that specifies the position to begin reading. The first byte or character is at position 1.

buffer
An output argument of type BLOB(32762) or VARCHAR(32672) that specifies the variable to receive the large object. If *lob_loc* is a BLOB, then *buffer* must be BLOB. If *lob_loc* is a CLOB, then *buffer* must be VARCHAR.

Authorization

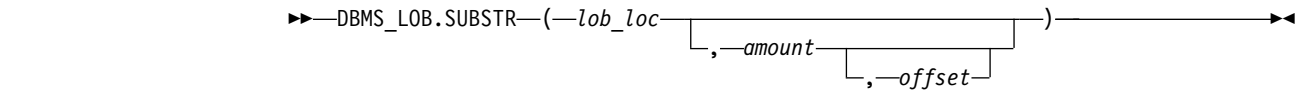
EXECUTE privilege on the DBMS_LOB module.

SUBSTR function - Return a portion of a large object

The SUBSTR function provides the capability to return a portion of a large object.

The function returns a BLOB(32767) (for a BLOB) or VARCHAR (for a CLOB) value for the returned portion of the large object read by the function.

Syntax



Parameters

lob_loc
An input argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be read.

amount
An optional input argument of type INTEGER that specifies the number of bytes or characters to be returned. The default value is 32,767.

offset
An optional input argument of type INTEGER that specifies the position within the large object to begin returning data. The first byte or character is at position 1. The default value is 1.

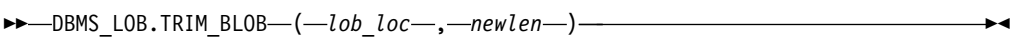
Authorization

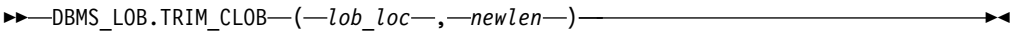
EXECUTE privilege on the DBMS_LOB module.

TRIM procedures - Truncate a large object to the specified length

The TRIM procedures provide the capability to truncate a large object to the specified length.

Syntax





Parameters

lob_loc
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be trimmed.

newlen
An input argument of type INTEGER that specifies the new number of bytes or characters to which the large object is to be trimmed.

Authorization

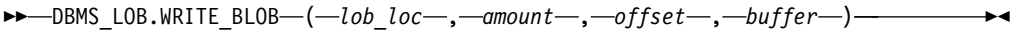
EXECUTE privilege on the DBMS_LOB module.

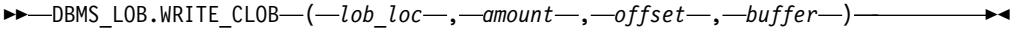
WRITE procedures - Write data to a large object

The WRITE procedures provide the capability to write data into a large object.

Any existing data in the large object at the specified offset for the given length is overwritten by data given in the buffer.

Syntax





Parameters

lob_loc
An input or output argument of type BLOB(10M) or CLOB(10M) that specifies the large object locator of the large object to be written.

amount
An input argument of type INTEGER that specifies the number of bytes or characters in *buffer* to be written to the large object.

offset
An input argument of type INTEGER that specifies the offset in bytes or characters from the beginning of the large object for the write operation to begin. The start value of the large object is 1.

buffer
An input argument of type BLOB(32767) or VARCHAR(32672) that contains the data to be written to the large object. If *lob_loc* is a BLOB, then *buffer* must be BLOB. If *lob_loc* is a CLOB, then *buffer* must be VARCHAR.

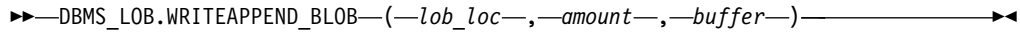
Authorization

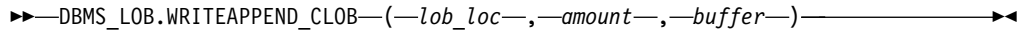
EXECUTE privilege on the DBMS_LOB module.

WRITEAPPEND procedures - Append data to the end of a large object

The WRITEAPPEND procedures provide the capability to add data to the end of a large object.

Syntax





Parameters

lob_loc
An input or output argument of type BLOB or CLOB that specifies the large object locator of the large object to which data is to appended.

amount
An input argument of type INTEGER that specifies the number of bytes or characters from *buffer* to be appended to the large object.

buffer
An input argument of type BLOB(32767) or VARCHAR(32672) that contains the data to be appended to the large object. If *lob_loc* is a BLOB, then *buffer* must be BLOB. If *lob_loc* is a CLOB, then *buffer* must be VARCHAR.

Authorization

EXECUTE privilege on the DBMS_LOB module.

DBMS_OUTPUT module

The DBMS_OUTPUT module provides a set of procedures for putting messages (lines of text) in a message buffer and getting messages from the message buffer. These procedures are useful during application debugging when you need to write messages to standard output.

The schema for this module is SYSIBMADM.

The DBMS_OUTPUT module includes the following built-in routines.

Table 7. Built-in routines available in the DBMS_OUTPUT module

Routine name	Description
DISABLE procedure	Disables the message buffer.
ENABLE procedure	Enables the message buffer
GET_LINE procedure	Gets a line of text from the message buffer.
GET_LINES procedure	Gets one or more lines of text from the message buffer and places the text into a collection
NEW_LINE procedure	Puts an end-of-line character sequence in the message buffer.
PUT procedure	Puts a string that includes no end-of-line character sequence in the message buffer.
PUT_LINE procedure	Puts a single line that includes an end-of-line character sequence in the message buffer.

The procedures in this module allow you to work with the message buffer. Use the command line processor (CLP) command SET SERVEROUTPUT ON to redirect the output to standard output.

DISABLE and ENABLE procedures are not supported inside autonomous procedures.

An autonomous procedure is a procedure that, when called, executes inside a new transaction independent of the original transaction.

Example

In proc1 use the PUT and PUT_LINE procedures to put a line of text in the message buffer. When proc1 runs for the first time, SET SERVEROUTPUT ON is specified, and the line in the message buffer is printed to the CLP window. When proc1 runs a second time, SET SERVEROUTPUT OFF is specified, and no lines from the message buffer are printed to the CLP window.

```
CREATE PROCEDURE proc1( P1 VARCHAR(10) )
BEGIN
  CALL DBMS_OUTPUT.PUT( 'P1 = ' );
  CALL DBMS_OUTPUT.PUT_LINE( P1 );
END@

SET SERVEROUTPUT ON@

CALL proc1( '10' )@
```

```
SET SERVEROUTPUT OFF@

CALL proc1( '20' )@
```

The example results in the following output:

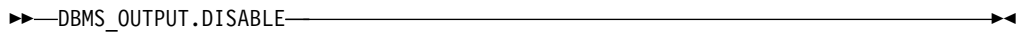
```
CALL proc1( '10' )

  Return Status = 0

P1 = 10

SET SERVEROUTPUT OFF
DB20000I  The SET SERVEROUTPUT command completed successfully.

CALL proc1( '20' )

  Return Status = 0
```

DISABLE procedure - Disable the message buffer

The DISABLE procedure disables the message buffer.

After this procedure runs, any messages that are in the message buffer are discarded. Calls to the PUT, PUT_LINE, or NEW_LINE procedures are ignored, and no error is returned to the sender.

Syntax



Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

The following example disables the message buffer for the current session:

```
CALL DBMS_OUTPUT.DISABLE@
```

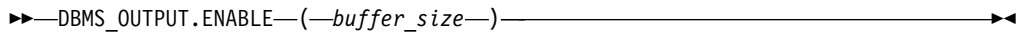
Usage notes

To send and receive messages after the message buffer has been disabled, use the ENABLE procedure.

ENABLE procedure - Enable the message buffer

The ENABLE procedure enables the message buffer. During a single session, applications can put messages in the message buffer and get messages from the message buffer.

Syntax



Procedure parameters

buffer_size
An input argument of type INTEGER that specifies the maximum length of the message buffer in bytes. If you specify a value of less than 2000 for *buffer_size*, the buffer size is set to 2000. If the value is NULL, then the default buffer size is 20000.

Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

The following example enables the message buffer:

```
CALL DBMS_OUTPUT.ENABLE( NULL )@
```

Usage notes

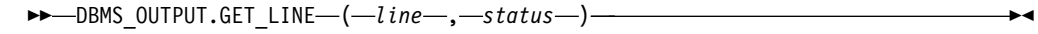
You can call the ENABLE procedure to increase the size of an existing message buffer. Any messages in the old buffer are copied to the enlarged buffer.

GET_LINE procedure - Get a line from the message buffer

The GET_LINE procedure gets a line of text from the message buffer. The text must be terminated by an end-of-line character sequence.

Tip: To add an end-of-line character sequence to the message buffer, use the PUT_LINE procedure, or, after a series of calls to the PUT procedure, use the NEW_LINE procedure.

Syntax



Procedure parameters

line
An output argument of type VARCHAR(32672) that returns a line of text from the message buffer.

status
An output argument of type INTEGER that indicates whether a line was returned from the message buffer:

- 0 indicates that a line was returned
- 1 indicates that there was no line to return

Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

Use the GET_LINE procedure to get a line of text from the message buffer. In this example, proc1 puts a line of text in the message buffer. proc3 gets the text from the message buffer and inserts it into a table named messages. proc2 then runs, but

because the message buffer is disabled, no text is added to the message buffer. When the select statement runs, it returns only the text added by proc1.

```
CALL DBMS_OUTPUT.ENABLE( NULL )@

CREATE PROCEDURE proc1()
BEGIN
  CALL DBMS_OUTPUT.PUT_LINE( 'PROC1 put this line in the message buffer.' );
END@

CREATE PROCEDURE proc2()
BEGIN
  CALL DBMS_OUTPUT.PUT_LINE( 'PROC2 put this line in the message buffer.' );
END@

CREATE TABLE messages ( msg VARCHAR(100) )@

CREATE PROCEDURE proc3()
BEGIN
  DECLARE line VARCHAR(32672);
  DECLARE status INT;

  CALL DBMS_OUTPUT.GET_LINE( line, status );
  while status = 0 do
    INSERT INTO messages VALUES ( line );
    CALL DBMS_OUTPUT.GET_LINE( line, status );
  end while;
END@

CALL proc1@

CALL proc3@

CALL DBMS_OUTPUT.DISABLE@

CALL proc2@

CALL proc3@

SELECT * FROM messages@
```

This example results in the following output:

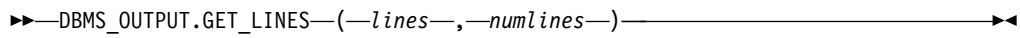
```
MSG
-----------------------------------------------
PROC1 put this line in the message buffer.

  1 record(s) selected.
```

GET_LINES procedure - Get multiple lines from the message buffer

The GET_LINES procedure gets one or more lines of text from the message buffer and stores the text in a collection. Each line of text must be terminated by an end-of-line character sequence.

Tip: To add an end-of-line character sequence to the message buffer, use the PUT_LINE procedure, or, after a series of calls to the PUT procedure, use the NEW_LINE procedure.

Syntax



Procedure parameters

lines
An output argument of type DBMS_OUTPUT.CHARARR that returns the lines of text from the message buffer. The type DBMS_OUTPUT.CHARARR is internally defined as a VARCHAR(32672) ARRAY[2147483647] array.

numlines
An input and output argument of type INTEGER. When used as input, specifies the number of lines to retrieve from the message buffer. When used as output, indicates the actual number of lines that were retrieved from the message buffer. If the output value of *numlines* is less than the input value, then there are no more lines remaining in the message buffer.

Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

Use the GET_LINES procedure to get lines of text from the message buffer and store the text in an array. The text in the array can be inserted into a table and queried.

```
CALL DBMS_OUTPUT.ENABLE( NULL )@

CREATE PROCEDURE proc1()
BEGIN
  CALL DBMS_OUTPUT.PUT_LINE( 'PROC1 put this line in the message buffer.' );
  CALL DBMS_OUTPUT.PUT_LINE( 'PROC1 put this line in the message buffer.' );
END@

CREATE PROCEDURE proc2()
BEGIN
  CALL DBMS_OUTPUT.PUT_LINE( 'PROC2 put this line in the message buffer.' );
END@

CREATE TABLE messages ( msg VARCHAR(100) )@

CREATE PROCEDURE proc3()
BEGIN
  DECLARE lines DBMS_OUTPUT.CHARARR;
  DECLARE numlines INT;
  DECLARE i INT;

  CALL DBMS_OUTPUT.GET_LINES( lines, numlines );
  SET i = 1;
  WHILE i <= numlines DO
    INSERT INTO messages VALUES ( lines[i] );
    SET i = i + 1;
  END WHILE;
END@

CALL proc1@

CALL proc3@

CALL DBMS_OUTPUT.DISABLE@

CALL proc2@

CALL proc3@

SELECT * FROM messages@
```

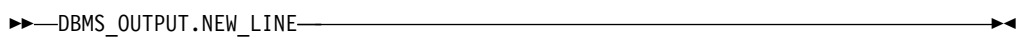
This example results in the following output:

```
MSG
----------------------------------------
PROC1 put this line in the message buffer.
PROC1 put this line in the message buffer

  2 record(s) selected.
```

NEW_LINE procedure - Put an end-of-line character sequence in the message buffer

The NEW_LINE procedure puts an end-of-line character sequence in the message buffer.

Syntax



Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

Use the NEW_LINE procedure to write an end-of-line character sequence to the message buffer. In this example, the text that is followed by an end-of-line character sequence displays as output because SET SERVEROUTPUT ON is specified. However, the text that is in the message buffer, but is not followed by an end-of-line character, does not display.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  CALL DBMS_OUTPUT.PUT( 'T' );
  CALL DBMS_OUTPUT.PUT( 'h' );
  CALL DBMS_OUTPUT.PUT( 'i' );
  CALL DBMS_OUTPUT.PUT( 's' );
  CALL DBMS_OUTPUT.NEW_LINE;
  CALL DBMS_OUTPUT.PUT( 'T' );
  CALL DBMS_OUTPUT.PUT( 'h' );
  CALL DBMS_OUTPUT.PUT( 'a' );
  CALL DBMS_OUTPUT.PUT( 't' );
END@

CALL proc1@

SET SERVEROUTPUT OFF@
```

This example results in the following output:

```
This
```

PUT procedure - Put a partial line in the message buffer

The PUT procedure puts a string in the message buffer. No end-of-line character sequence is written at the end of the string.

Syntax

Procedure parameters

item
An input argument of type VARCHAR(32672) that specifies the text to write to the message buffer.

Authorization

EXECUTE privilege on the DBMS_OUTPUT module.

Example

Use the PUT procedure to put a partial line in the message buffer. In this example, the NEW_LINE procedure adds an end-of-line character sequence to the message buffer. When proc1 runs, because SET SERVEROUTPUT ON is specified, a line of text is returned.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  CALL DBMS_OUTPUT.PUT( 'H' );
  CALL DBMS_OUTPUT.PUT( 'e' );
  CALL DBMS_OUTPUT.PUT( 'l' );
  CALL DBMS_OUTPUT.PUT( 'l' );
  CALL DBMS_OUTPUT.PUT( 'o' );
  CALL DBMS_OUTPUT.PUT( '.' );
  CALL DBMS_OUTPUT.NEW_LINE;
END@

CALL proc1@

SET SERVEROUTPUT OFF@
```

This example results in the following output:

```
Hello.
```

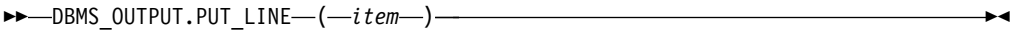
Usage notes

After using the PUT procedure to add text to the message buffer, use the NEW_LINE procedure to add an end-of-line character sequence to the message buffer. Otherwise, the text is not returned by the GET_LINE and GET_LINES procedures because it is not a complete line.

PUT_LINE procedure - Put a complete line in the message buffer

The PUT_LINE procedure puts a single line that includes an end-of-line character sequence in the message buffer.

Syntax



Procedure parameters

item
An input argument of type VARCHAR(32672) that specifies the text to write to the message buffer.

Authorization

EXECUTE privilege on the PUT_LINE procedure.

Example

Use the PUT_LINE procedure to write a line that includes an end-of-line character sequence to the message buffer.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE PROC1()
BEGIN
  CALL DBMS_OUTPUT.PUT( 'a' );
  CALL DBMS_OUTPUT.NEW_LINE;
  CALL DBMS_OUTPUT.PUT_LINE( 'b' );
END@

CALL PROC1@

SET SERVEROUTPUT OFF@
```

This example results in the following output:

```
a
b
```

DBMS_PIPE module

The DBMS_PIPE module provides a set of routines for sending messages through a pipe within or between sessions that are connected to databases within the same Db2 instance.

The schema for this module is SYSIBMADM.

The DBMS_PIPE module includes the following built-in routines.

Table 8. Built-in routines available in the DBMS_PIPE module

Routine name	Description
CREATE_PIPE function	Explicitly creates a private or public pipe.
NEXT_ITEM_TYPE function	Determines the data type of the next item in a received message.
PACK_MESSAGE function	Puts an item in the session's local message buffer.
PACK_MESSAGE_RAW procedure	Puts an item of type RAW in the session's local message buffer.
PURGE procedure	Removes unreceived messages in the specified pipe.
RECEIVE_MESSAGE function	Gets a message from the specified pipe.
REMOVE_PIPE function	Deletes an explicitly created pipe.
RESET_BUFFER procedure	Resets the local message buffer.

Table 8. Built-in routines available in the DBMS_PIPE module (continued)

Routine name	Description
SEND_MESSAGE procedure	Sends a message on the specified pipe.
UNIQUE_SESSION_NAME function	Returns a unique session name.
UNPACK_MESSAGE procedures	Retrieves the next data item from a message and assigns it to a variable.

Usage notes

Pipes are created either implicitly or explicitly during procedure calls. An *implicit pipe* is created when a procedure call contains a reference to a pipe name that does not exist. For example, if a pipe named "mailbox" is passed to the SEND_MESSAGE procedure and that pipe does not already exist, a new pipe named "mailbox" is created. An *explicit pipe* is created by calling the CREATE_PIPE function and specifying the name of the pipe.

Pipes can be private or public. A *private pipe* can only be accessed by the user who created the pipe. Even an administrator cannot access a private pipe that was created by another user. A *public pipe* can be accessed by any user who has access to the DBMS_PIPE module. To specify the access level for a pipe, use the CREATE_PIPE function and specify a value for the *private* parameter: "false" specifies that the pipe is public; "true" specifies that the pipe is private. If no value is specified, the default is to create a private pipe. All implicit pipes are public.

To send a message through a pipe, call the PACK_MESSAGE function to put individual data items (lines) in a local message buffer that is unique to the current session. Then, call the SEND_MESSAGE procedure to send the message through the pipe.

To receive a message, call the RECEIVE_MESSAGE function to get a message from the specified pipe. The message is written to the receiving session's local message buffer. Then, call the UNPACK_MESSAGE procedure to retrieve the next data item from the local message buffer and assign it to a specified program variable. If a pipe contains multiple messages, the RECEIVE_MESSAGE function gets the messages in FIFO (first-in-first-out) order.

Each session maintains separate message buffers for messages that are created by the PACK_MESSAGE function and messages that are retrieved by the RECEIVE_MESSAGE function. The separate message buffers allow you to build and receive messages in the same session. However, when consecutive calls are made to the RECEIVE_MESSAGE function, only the message from the last RECEIVE_MESSAGE call is preserved in the local message buffer.

Example

In connection 1, create a pipe that is named pipe1. Put a message in the session's local message buffer, and send the message through pipe1.

```
BEGIN
  DECLARE status INT;
  SET status = DBMS_PIPE.CREATE_PIPE( 'pipe1' );
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
  SET status = DBMS_PIPE.SEND_MESSAGE( 'pipe1' );
END@
```

In connection 2, receive the message, unpack it, and display it to standard output.

```
SET SERVEROUTPUT ON@

BEGIN
  DECLARE status   INT;
  DECLARE int1     INTEGER;
  DECLARE date1    DATE;
  DECLARE raw1     BLOB(100);
  DECLARE varchar1 VARCHAR(100);
  DECLARE itemType INTEGER;

  SET status = DBMS_PIPE.RECEIVE_MESSAGE( 'pipe1' );
  IF( status = 0 ) THEN
    SET itemType = DBMS_PIPE.NEXT_ITEM_TYPE();
    CASE itemType
      WHEN 6 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_INT( int1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'int1: ' || int1 );
      WHEN 9 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_CHAR( varchar1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'varchar1: ' || varchar1 );
      WHEN 12 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_DATE( date1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'date1:' || date1 );
      WHEN 23 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_RAW( raw1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'raw1: ' || VARCHAR(raw1) );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'Unexpected value' );
    END CASE;
  END IF;
  SET status = DBMS_PIPE.REMOVE_PIPE( 'pipe1' );
END@
```

This example results in the following output:

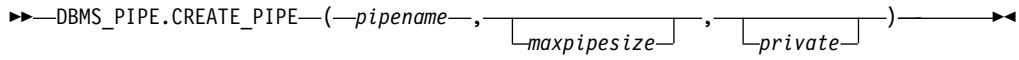
```
  varchar1: message1
```

CREATE_PIPE function - Create a pipe

The CREATE_PIPE function explicitly creates a public or private pipe with the specified name.

For more information about explicit public and private pipes, see the topic about the DBMS_PIPE module.

Syntax



Return value

This function returns the status code 0 if the pipe is created successfully.

Function parameters

pipename
An input argument of type VARCHAR(128) that specifies the name of the pipe. For more information about pipes, see "DBMS_PIPE module" on page 39.

maxpipesize
An optional input argument of type INTEGER that specifies the maximum capacity of the pipe in bytes. The default is 8192 bytes.

private
An optional input argument that specifies the access level of the pipe:

For non-partitioned database environments
A value of "0" or "FALSE" creates a public pipe.

A value of "1" or "TRUE creates a private pipe. This is the default.

In a partitioned database environment
A value of "0" creates a public pipe.

A value of "1" creates a private pipe. This is the default.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

Example 1: Create a private pipe that is named messages:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_status        INTEGER;
  SET v_status = DBMS_PIPE.CREATE_PIPE('messages');
  DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status: ' || v_status);
END@

CALL proc1@
```

This example results in the following output:

```
CREATE_PIPE status: 0
```

Example 2: Create a public pipe that is named mailbox:

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE v_status INTEGER;
  SET v_status = DBMS_PIPE.CREATE_PIPE('mailbox',0);
  DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status: ' || v_status);
END@

CALL proc2@
```

This example results in the following output:

```
CREATE_PIPE status: 0
```

NEXT_ITEM_TYPE function - Return the data type code of the next item

The NEXT_ITEM_TYPE function returns an integer code that identifies the data type of the next data item in a received message.

The received message is stored in the session's local message buffer. Use the UNPACK_MESSAGE procedure to move each item off of the local message buffer,

and then use the NEXT_ITEM_TYPE function to return the data type code for the next available item. A code of 0 is returned when there are no more items left in the message.

Syntax

Return value

This function returns one of the following codes that represents a data type.

Table 9. NEXT_ITEM_TYPE data type codes

Type code	Data type
0	No more data items
6	INTEGER
9	VARCHAR
12	DATE
23	BLOB

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1, pack and send a message. In proc2, receive the message and then unpack it by using the NEXT_ITEM_TYPE function to determine its type.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE status INT;
  SET status = DBMS_PIPE.CREATE_PIPE( 'pipe1' );
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
  SET status = DBMS_PIPE.SEND_MESSAGE( 'pipe1' );
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE status   INT;
  DECLARE num1     DECFLOAT;
  DECLARE date1    DATE;
  DECLARE raw1     BLOB(100);
  DECLARE varchar1 VARCHAR(100);
  DECLARE itemType INTEGER;

  SET status = DBMS_PIPE.RECEIVE_MESSAGE( 'pipe1' );
  IF( status = 0 ) THEN
    SET itemType = DBMS_PIPE.NEXT_ITEM_TYPE();
    CASE itemType
      WHEN 6 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_NUMBER( num1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'num1: ' || num1 );
      WHEN 9 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_CHAR( varchar1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'varchar1: ' || varchar1 );
```

```
      WHEN 12 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_DATE( date1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'date1:' || date1 );
      WHEN 23 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_RAW( raw1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'raw1: ' || VARCHAR(raw1) );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'Unexpected value' );
    END CASE;
  END IF;
  SET status = DBMS_PIPE.REMOVE_PIPE( 'pipe1' );
END@

CALL proc1@

CALL proc2@
```

This example results in the following output:

```
varchar1: message1
```

PACK_MESSAGE function - Put a data item in the local message buffer

The PACK_MESSAGE function puts a data item in the session's local message buffer.

Syntax

Procedure parameters

item
An input argument of type VARCHAR(4096), DATE, or DECFLOAT that contains an expression. The value returned by this expression is added to the local message buffer of the session.

Tip: To put data items of type RAW in the local message buffer, use the PACK_MESSAGE_RAW procedure.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

Use the PACK_MESSAGE function to put a message for Sujata in the local message buffer, and then use the SEND_MESSAGE procedure to send the message on a pipe.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_status    INTEGER;
  DECLARE    status      INTEGER;
  SET status = DBMS_PIPE.PACK_MESSAGE('Hi, Sujata');
  SET status = DBMS_PIPE.PACK_MESSAGE('Can you attend a meeting at 3:00, today?');
  SET status = DBMS_PIPE.PACK_MESSAGE('If not, is tomorrow at 8:30 ok with you?');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe');
```

```
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END@

CALL proc1@
```

This example results in the following output:

```
  SEND_MESSAGE status: 0
```

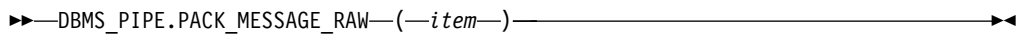
Usage notes

The PACK_MESSAGE function or PACK_MESSAGE_RAW procedure must be called at least once before issuing a SEND_MESSAGE call.

PACK_MESSAGE_RAW procedure - Put a data item of type RAW in the local message buffer

The PACK_MESSAGE_RAW procedure puts a data item of type RAW in the session's local message buffer.

Syntax



Procedure parameters

item

An input argument of type BLOB(4096) that specifies an expression. The value returned by this expression is added to the session's local message buffer.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

Use the PACK_MESSAGE_RAW procedure to put a data item of type RAW in the local message buffer.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_raw           BLOB(100);
  DECLARE v_raw2   BLOB(100);
  DECLARE v_status        INTEGER;
  SET v_raw = BLOB('21222324');
  SET v_raw2 = BLOB('30000392');
  CALL DBMS_PIPE.PACK_MESSAGE_RAW(v_raw);
  CALL DBMS_PIPE.PACK_MESSAGE_RAW(v_raw2);
  SET v_status = DBMS_PIPE.SEND_MESSAGE('datatypes');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END@

CALL proc1@
```

This example results in the following output:

```
SEND_MESSAGE status: 0
```

Usage notes

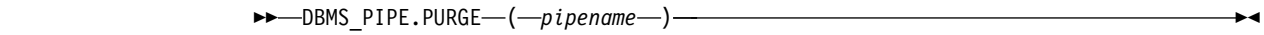
The PACK_MESSAGE function or PACK_MESSAGE_RAW procedure must be called at least once before issuing a SEND_MESSAGE call.

PURGE procedure - Remove unreceived messages from a pipe

The PURGE procedure removes unreceived messages in the specified implicit pipe.

Tip: Use the REMOVE_PIPE function to delete an explicit pipe.

Syntax



Procedure parameters

pipename
An input argument of type VARCHAR(128) that specifies the name of the implicit pipe.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1 send two messages on a pipe: Message #1 and Message #2. In proc2, receive the first message, unpack it, and then purge the pipe. When proc3 runs, the call to the RECEIVE_MESSAGE function times out and returns the status code 1 because no message is available.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE   v_status        INTEGER;
  DECLARE   status          INTEGER;
  SET status = DBMS_PIPE.PACK_MESSAGE('Message #1');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
  SET status = DBMS_PIPE.PACK_MESSAGE('Message #2');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE   v_item          VARCHAR(80);
  DECLARE   v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
  CALL DBMS_PIPE.UNPACK_MESSAGE_CHAR(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
  CALL DBMS_PIPE.PURGE('pipe');
END@

CREATE PROCEDURE proc3()
BEGIN
  DECLARE    v_item          VARCHAR(80);
```

```
  DECLARE    v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
END@

CALL proc1@

CALL proc2@

CALL proc3@
```

This example results in the following output.

From proc1:

```
SEND_MESSAGE status: 0
SEND_MESSAGE status: 0
```

From proc2:

```
RECEIVE_MESSAGE status: 0
Item: Hi, Sujata
```

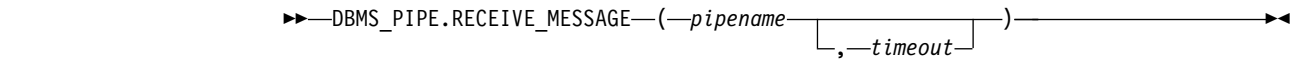
From proc3:

```
RECEIVE_MESSAGE status: 1
```

RECEIVE_MESSAGE function - Get a message from a specified pipe

The RECEIVE_MESSAGE function gets a message from a specified pipe.

Syntax



Return value

The RECEIVE_MESSAGE function returns one of the following status codes of type INTEGER.

Table 10. RECEIVE_MESSAGE status codes

Status code	Description
0	Success
1	Time out

Function parameters

pipename
An input argument of type VARCHAR(128) that specifies the name of the pipe. If the specified pipe does not exist, the pipe is created implicitly. For more information about pipes, see "DBMS_PIPE module" on page 39.

timeout
An optional input argument of type INTEGER that specifies the wait time in seconds. The default is 86400000 (1000 days).

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1, send a message. In proc2, receive and unpack the message. Timeout if the message is not received within 1 second.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE status INTEGER;
  SET status = DBMS_PIPE.CREATE_PIPE( 'pipe1' );
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
  SET status = DBMS_PIPE.SEND_MESSAGE( 'pipe1' );
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE    v_item          VARCHAR(80);
  DECLARE    v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe1',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
  CALL DBMS_PIPE.UNPACK_MESSAGE(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END@

CALL proc1@
CALL proc2@
```

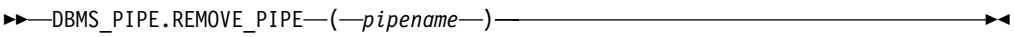
This example results in the following output:

```
RECEIVE_MESSAGE status: 0
Item: message1
```

REMOVE_PIPE function - Delete a pipe

The REMOVE_PIPE function deletes an explicitly created pipe. Use this function to delete any public or private pipe that was created by the CREATE_PIPE function.

Syntax



Return value

This function returns one of the following status codes of type INTEGER.

Table 11. REMOVE_PIPE status codes

Status code	Description
0	Pipe successfully removed or does not exist
NULL	An exception is thrown

Function parameters

pipename
An input argument of type VARCHAR(128) that specifies the name of the pipe.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1 send two messages on a pipe: Message #1 and Message #2. In proc2, receive the first message, unpack it, and then delete the pipe. When proc3 runs, the call to the RECEIVE_MESSAGE function times out and returns the status code 1 because the pipe no longer exists.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_status        INTEGER;
  DECLARE    status          INTEGER;
  SET  v_status = DBMS_PIPE.CREATE_PIPE('pipe1');
  CALL DBMS_OUTPUT.PUT_LINE('CREATE_PIPE status : ' || v_status);

  SET status = DBMS_PIPE.PACK_MESSAGE('Message #1');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe1');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);

  SET status = DBMS_PIPE.PACK_MESSAGE('Message #2');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe1');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE    v_item          VARCHAR(80);
  DECLARE    v_status        INTEGER;
  DECLARE    status          INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe1',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
  CALL DBMS_PIPE.UNPACK_MESSAGE(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
  SET status = DBMS_PIPE.REMOVE_PIPE('pipe1');
END@

CREATE PROCEDURE proc3()
BEGIN
  DECLARE    v_item          VARCHAR(80);
  DECLARE    v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe1',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
END@

CALL proc1@

CALL proc2@

CALL proc3@
```

This example results in the following output.

From proc1:

```
CREATE_PIPE status : 0
SEND_MESSAGE status: 0
SEND_MESSAGE status: 0
```

From proc2:

```
RECEIVE_MESSAGE status: 0
Item: Message #1
```

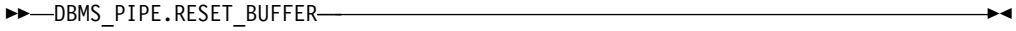
From proc3:

```
RECEIVE_MESSAGE status: 1
```

RESET_BUFFER procedure - Reset the local message buffer

The RESET_BUFFER procedure resets a pointer to the session's local message buffer back to the beginning of the buffer. Resetting the buffer causes subsequent PACK_MESSAGE calls to overwrite any data items that existed in the message buffer prior to the RESET_BUFFER call.

Syntax



Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1, use the PACK_MESSAGE function to put a message for an employee named Sujata in the local message buffer. Call the RESET_BUFFER procedure to replace the message with a message for Bing, and then send the message on a pipe. In proc2, receive and unpack the message for Bing.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_status        INTEGER;
  DECLARE    status          INTEGER;
  SET status = DBMS_PIPE.PACK_MESSAGE('Hi, Sujata');
  SET status = DBMS_PIPE.PACK_MESSAGE('Can you attend a meeting at 3:00, today?');
  SET status = DBMS_PIPE.PACK_MESSAGE('If not, is tomorrow at 8:30 ok with you?');
  CALL DBMS_PIPE.RESET_BUFFER;
  SET status = DBMS_PIPE.PACK_MESSAGE('Hi, Bing');
  SET status = DBMS_PIPE.PACK_MESSAGE('Can you attend a meeting at 9:30, tomorrow?');
  SET v_status = DBMS_PIPE.SEND_MESSAGE('pipe');
  CALL DBMS_OUTPUT.PUT_LINE('SEND_MESSAGE status: ' || v_status);
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE    v_item          VARCHAR(80);
  DECLARE    v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
  CALL DBMS_PIPE.UNPACK_MESSAGE_CHAR(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
  CALL DBMS_PIPE.UNPACK_MESSAGE(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END@

CALL proc1@

CALL proc2@
```

This example results in the following output:

From proc1:

```
SEND_MESSAGE status: 0
```

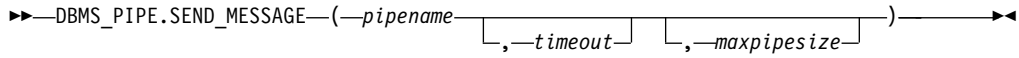
From proc2:

```
RECEIVE_MESSAGE status: 0
Item: Hi, Bing
Item: Can you attend a meeting at 9:30, tomorrow?
```

SEND_MESSAGE procedure - Send a message to a specified pipe

The SEND_MESSAGE procedure sends a message from the session's local message buffer to a specified pipe.

Syntax



Return value

This procedure returns one of the following status codes of type INTEGER.

Table 12. SEND_MESSAGE status codes

Status code	Description
0	Success
1	Time out

Procedure parameters

pipename
An input argument of type VARCHAR(128) that specifies the name of the pipe. If the specified pipe does not exist, the pipe is created implicitly. For more information about pipes, see "DBMS_PIPE module" on page 39.

timeout
An optional input argument of type INTEGER that specifies the wait time in seconds. The default is 86400000 (1000 days).

maxpipesize
An optional input argument of type INTEGER that specifies the maximum capacity of the pipe in bytes. The default is 8192 bytes.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1, send a message. In proc2, receive and unpack the message. Timeout if the message is not received within 1 second.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE status INTEGER;
  SET status = DBMS_PIPE.CREATE_PIPE( 'pipe1' );
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
```

```
  SET status = DBMS_PIPE.SEND_MESSAGE( 'pipe1' );
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE    v_item          VARCHAR(80);
  DECLARE    v_status        INTEGER;
  SET v_status = DBMS_PIPE.RECEIVE_MESSAGE('pipe1',1);
  CALL DBMS_OUTPUT.PUT_LINE('RECEIVE_MESSAGE status: ' || v_status);
  CALL DBMS_PIPE.UNPACK_MESSAGE(v_item);
  CALL DBMS_OUTPUT.PUT_LINE('Item: ' || v_item);
END@

CALL proc1@
CALL proc2@
```

This example results in the following output:

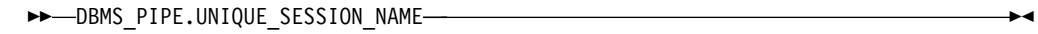
```
RECEIVE_MESSAGE status: 0
Item: message1
```

UNIQUE_SESSION_NAME function - Return a unique session name

The UNIQUE_SESSION_NAME function returns a unique name for the current session.

You can use this function to create a pipe that has the same name as the current session. To create this pipe, pass the value returned by the UNIQUE_SESSION_NAME function to the SEND_MESSAGE procedure as the pipe name. An implicit pipe is created that has the same name as the current session.

Syntax



Return value

This function returns a value of type VARCHAR(128) that represents the unique name for the current session.

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

Create a pipe that has the same name as the current session.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    status          INTEGER;
  DECLARE    v_session       VARCHAR(30);
  SET v_session = DBMS_PIPE.UNIQUE_SESSION_NAME;
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
  SET status = DBMS_PIPE.SEND_MESSAGE(v_session);
```

```
  CALL DBMS_OUTPUT.PUT_LINE('Sent message on pipe ' || v_session);
END@

CALL proc1@
```

This example results in the following output:

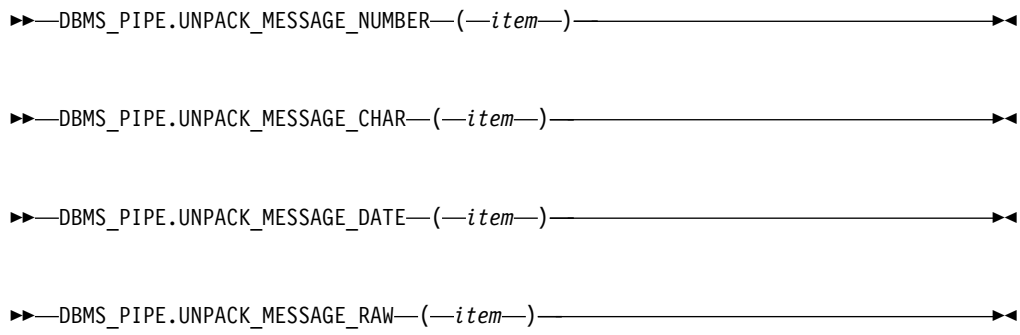
```
Sent message on pipe *LOCAL.myschema.080522010048
```

UNPACK_MESSAGE procedures - Get a data item from the local message buffer

The UNPACK_MESSAGE procedures retrieve the next data item from a message and assign it to a variable.

Before calling one of the UNPACK_MESSAGE procedures, use the RECEIVE_MESSAGE procedure to place the message in the local message buffer.

Syntax



Procedure parameters

item

An output argument of one of the following types that specifies a variable to receive data items from the local message buffer.

Routine	Data type
UNPACK_MESSAGE_NUMBER	DECFLOAT
UNPACK_MESSAGE_CHAR	VARCHAR(4096)
UNPACK_MESSAGE_DATE	DATE
UNPACK_MESSAGE_RAW	BLOB(4096)

Authorization

EXECUTE privilege on the DBMS_PIPE module.

Example

In proc1, pack and send a message. In proc2, receive the message, unpack it using the appropriate procedure based on the item's type, and display the message to standard output.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
```

```
BEGIN
  DECLARE status INT;
  SET status = DBMS_PIPE.CREATE_PIPE( 'pipe1' );
  SET status = DBMS_PIPE.PACK_MESSAGE('message1');
  SET status = DBMS_PIPE.SEND_MESSAGE( 'pipe1' );
END@

CREATE PROCEDURE proc2()
BEGIN
  DECLARE status   INT;
  DECLARE num1     DECFLOAT;
  DECLARE date1    DATE;
  DECLARE raw1     BLOB(100);
  DECLARE varchar1 VARCHAR(100);
  DECLARE itemType INTEGER;

  SET status = DBMS_PIPE.RECEIVE_MESSAGE( 'pipe1' );
  IF( status = 0 ) THEN
    SET itemType = DBMS_PIPE.NEXT_ITEM_TYPE();
    CASE itemType
      WHEN 6 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_NUMBER( num1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'num1: ' || num1 );
      WHEN 9 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_CHAR( varchar1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'varchar1: ' || varchar1 );
      WHEN 12 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_DATE( date1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'date1:' || date1 );
      WHEN 23 THEN
        CALL DBMS_PIPE.UNPACK_MESSAGE_RAW( raw1 );
        CALL DBMS_OUTPUT.PUT_LINE( 'raw1: ' || VARCHAR(raw1) );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'Unexpected value' );
    END CASE;
  END IF;
  SET status = DBMS_PIPE.REMOVE_PIPE( 'pipe1' );
END@

CALL proc1@

CALL proc2@
```

This example results in the following output:

```
varchar1: message1
```

DBMS_RANDOM module

The DBMS_RANDOM module provides the capability to produce random numbers. It provides functions and procedures to seed a random number generator and then return random numbers or strings.

The schema for all procedures and functions in this module is SYSIBMADM.

The DBMS_RANDOM module includes the following built-in routines:

Table 13. Built-in routines available in the DBMS_RANDOM module

Routine name	Description
SEED procedure	The SEED procedure seeds the random number generator with a number.
SEED_STRING procedure	The SEED_STRING procedure seeds the random number generator with a string.

Table 13. Built-in routines available in the DBMS_RANDOM module (continued)

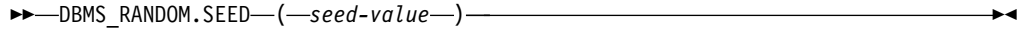
Routine name	Description
INITIALIZE procedure	The INITIALIZE procedure seeds the random number generator with a number.
TERMINATE procedure	TERMINATE is a no-op procedure which provides no operation.
RANDOM function	The RANDOM function returns a random number in the range of -2^{31} and 2^{31}.
VALUE function	The VALUE function returns a random number in a specified range.
STRING function	The STRING function returns a random string.
NORMAL function	The NORMAL function returns a random number in a standard normal distribution.

The DBMS_RANDOM module relies on the random number facilities of the host operating system. The random number facility of each host might vary in factors such as the number of potential distinct values and the quality of the randomness. For these reasons, the output of the functions and procedures in the DBMS_RANDOM module are not suitable as a source of randomness in a cryptographic system.

SEED procedure

The SEED procedure seeds the random number generator with a number.

Syntax



Description

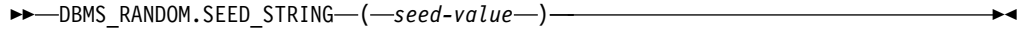
seed-value
An expression that returns a VARCHAR(4000) or INTEGER that seeds the random number generator.

If *seed-value* is NULL, the random number generator is seeded with 0.

SEED_STRING procedure

The SEED_STRING procedure seeds the random number generator with a string.

Syntax



Description

seed-value
An expression that returns a VARCHAR(4000) that seeds the random number generator.

If *seed-value* is NULL, the random number generator is seeded with an empty string.

INITIALIZE procedure

The INITIALIZE procedure seeds the random number generator with a number.

Syntax

Description

seed-value
An expression that returns an INTEGER that seeds the random number generator.

If *seed-value* is NULL, the random number generator is seeded with 0.

TERMINATE procedure

TERMINATE is a no-op procedure which provides no operation.

Syntax

RANDOM function

The RANDOM function returns a random number in the range of -2^{31} and 2^{31}.

Syntax

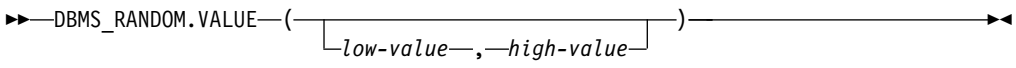
Note: Calling RANDOM before seeding the random number generator explicitly by calling SEED, SEED_STRING, or INITIALIZE defaults to using a seed that is based on the system current timestamp.

VALUE function

The VALUE function returns a random number in a specified range.

The function returns a DECFLOAT value.

Syntax



Description

low-value
An expression that returns an integer that specifies the upper bound on the random number.

high-value
An expression that returns an integer that specifies the lower bound on the random number.

Calling VALUE with no parameters defaults to returning a random number between 0 and 1.

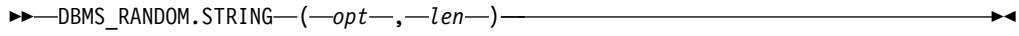
If *low-value* or *high-value* are NULL, then the result is NULL.

Note: Calling VALUE before seeding the random number generator explicitly by calling SEED, SEED_STRING, or INITIALIZE defaults to using a seed that is based on the system current timestamp.

STRING function

The STRING function returns a random string.

Syntax



Description

opt
An expression that returns CHAR(1). Specifies the operating mode and determines what the result should contain. The following values are valid:

'L' or 'l'
Lowercase ASCII characters only

'U' or 'u'
Uppercase ASCII characters only

'P' or 'p'
Printable ASCII characters only

'A' or 'a'
Combination of uppercase and lowercase ASCII characters

'X' or 'x'
Combination of uppercase ASCII characters and numbers

Any other input will default to uppercase ASCII characters, similar to 'U'.

len
An expression that returns an INTEGER. Specifies the length of the resulting random string, up to a maximum of 4000.

If *opt* is NULL, then the result will default to uppercase ASCII characters, similar to 'U'.

If *len* is NULL, then a zero length string will be returned.

Note: Calling STRING before seeding the random number generator explicitly by calling SEED, SEED_STRING, or INITIALIZE defaults to using a seed that is based on the system current timestamp.

NORMAL function

The NORMAL function returns a random number in a standard normal distribution.

The function returns a DECFLOAT value.

Syntax

Note: Calling NORMAL before seeding the random number generator explicitly by calling SEED, SEED_STRING, or INITIALIZE defaults to using a seed that is based on the system current timestamp.

DBMS_SQL module

The DBMS_SQL module provides a set of procedures for executing dynamic SQL, and therefore supports various data manipulation language (DML) or data definition language (DDL) statement.

The schema for this module is SYSIBMADM.

The DBMS_SQL module includes the following built-in routines.

Table 14. Built-in routines available in the DBMS_SQL module

Procedure name	Description
BIND_VARIABLE_BLOB procedure	Provides the input BLOB value for the IN or INOUT parameter; and defines the data type of the output value to be BLOB for the INOUT or OUT parameter.
BIND_VARIABLE_CHAR procedure	Provides the input CHAR value for the IN or INOUT parameter; and defines the data type of the output value to be CHAR for the INOUT or OUT parameter.
BIND_VARIABLE_CLOB procedure	Provides the input CLOB value for the IN or INOUT parameter; and defines the data type of the output value to be CLOB for the INOUT or OUT parameter.
BIND_VARIABLE_DATE procedure	Provides the input DATE value for the IN or INOUT parameter; and defines the data type of the output value to be DATE for the INOUT or OUT parameter.
BIND_VARIABLE_DOUBLE procedure	Provides the input DOUBLE value for the IN or INOUT parameter; and defines the data type of the output value to be DOUBLE for the INOUT or OUT parameter.
BIND_VARIABLE_INT procedure	Provides the input INTEGER value for the IN or INOUT parameter; and defines the data type of the output value to be INTEGER for the INOUT or OUT parameter.

Table 14. Built-in routines available in the DBMS_SQL module (continued)

Procedure name	Description
BIND_VARIABLE_NUMBER procedure	Provides the input DECFLOAT value for the IN or INOUT parameter; and defines the data type of the output value to be DECFLOAT for the INOUT or OUT parameter.
BIND_VARIABLE_RAW procedure	Provides the input BLOB(32767) value for the IN or INOUT parameter; and defines the data type of the output value to be BLOB(32767) for the INOUT or OUT parameter.
BIND_VARIABLE_TIMESTAMP procedure	Provides the input TIMESTAMP value for the IN or INOUT parameter; and defines the data type of the output value to be TIMESTAMP for the INOUT or OUT parameter.
BIND_VARIABLE_VARCHAR procedure	Provides the input VARCHAR value for the IN or INOUT parameter; and defines the data type of the output value to be VARCHAR for the INOUT or OUT parameter.
CLOSE_CURSOR procedure	Closes a cursor.
COLUMN_VALUE_BLOB procedure	Retrieves the value of column of type BLOB.
COLUMN_VALUE_CHAR procedure	Retrieves the value of column of type CHAR.
COLUMN_VALUE_CLOB procedure	Retrieves the value of column of type CLOB.
COLUMN_VALUE_DATE procedure	Retrieves the value of column of type DATE.
COLUMN_VALUE_DOUBLE procedure	Retrieves the value of column of type DOUBLE.
COLUMN_VALUE_INT procedure	Retrieves the value of column of type INTEGER.
COLUMN_VALUE_LONG procedure	Retrieves the value of column of type CLOB(32767).
COLUMN_VALUE_NUMBER procedure	Retrieves the value of column of type DECFLOAT.
COLUMN_VALUE_RAW procedure	Retrieves the value of column of type BLOB(32767).
COLUMN_VALUE_TIMESTAMP procedure	Retrieves the value of column of type TIMESTAMP
COLUMN_VALUE_VARCHAR procedure	Retrieves the value of column of type VARCHAR.
DEFINE_COLUMN_BLOB procedure	Defines the data type of the column to be BLOB.
DEFINE_COLUMN_CHAR procedure	Defines the data type of the column to be CHAR.
DEFINE_COLUMN_CLOB procedure	Defines the data type of the column to be CLOB.
DEFINE_COLUMN_DATE procedure	Defines the data type of the column to be DATE.

Table 14. Built-in routines available in the DBMS_SQL module (continued)

Procedure name	Description
DEFINE_COLUMN_DOUBLE procedure	Defines the data type of the column to be DOUBLE.
DEFINE_COLUMN_INT procedure	Defines the data type of the column to be INTEGER.
DEFINE_COLUMN_LONG procedure	Defines the data type of the column to be CLOB(32767).
DEFINE_COLUMN_NUMBER procedure	Defines the data type of the column to be DECFLOAT.
DEFINE_COLUMN_RAW procedure	Defines the data type of the column to be BLOB(32767).
DEFINE_COLUMN_TIMESTAMP procedure	Defines the data type of the column to be TIMESTAMP.
DEFINE_COLUMN_VARCHAR procedure	Defines the data type of the column to be VARCHAR.
DESCRIBE_COLUMNS procedure	Return a description of the columns retrieved by a cursor.
DESCRIBE_COLUMNS2 procedure	Identical to DESCRIBE_COLUMNS, but allows for column names greater than 32 characters.
EXECUTE procedure	Executes a cursor.
EXECUTE_AND_FETCH procedure	Executes a cursor and fetch one row.
FETCH_ROWS procedure	Fetches rows from a cursor.
IS_OPEN function	Checks if a cursor is open.
IS_OPEN procedure	Checks if a cursor is open.
LAST_ROW_COUNT procedure	Returns the total number of rows fetched.
OPEN_CURSOR procedure	Opens a cursor.
PARSE procedure	Parses a DDL statement.
VARIABLE_VALUE_BLOB procedure	Retrieves the value of INOUT or OUT parameters as BLOB.
VARIABLE_VALUE_CHAR procedure	Retrieves the value of INOUT or OUT parameters as CHAR.
VARIABLE_VALUE_CLOB procedure	Retrieves the value of INOUT or OUT parameters as CLOB.
VARIABLE_VALUE_DATE procedure	Retrieves the value of INOUT or OUT parameters as DATE.
VARIABLE_VALUE_DOUBLE procedure	Retrieves the value of INOUT or OUT parameters as DOUBLE.
VARIABLE_VALUE_INT procedure	Retrieves the value of INOUT or OUT parameters as INTEGER.
VARIABLE_VALUE_NUMBER procedure	Retrieves the value of INOUT or OUT parameters as DECFLOAT.
VARIABLE_VALUE_RAW procedure	Retrieves the value of INOUT or OUT parameters as BLOB(32767).
VARIABLE_VALUE_TIMESTAMP procedure	Retrieves the value of INOUT or OUT parameters as TIMESTAMP.

Table 14. Built-in routines available in the DBMS_SQL module (continued)

Procedure name	Description
VARIABLE_VALUE_VARCHAR procedure	Retrieves the value of INOUT or OUT parameters as VARCHAR.

The following table lists the built-in types and constants available in the DBMS_SQL module.

Table 15. DBMS_SQL built-in types and constants

Name	Type or constant	Description
DESC_REC	Type	A record of column information.
DESC_REC2	Type	A record of column information.
DESC_TAB	Type	An array of records of type DESC_REC.
DESC_TAB2	Type	An array of records of type DESC_REC2.
NATIVE	Constant	The only value supported for language_flag parameter of the PARSE procedure.

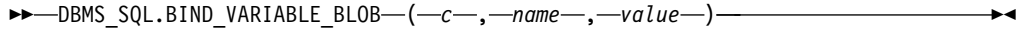
Usage notes

The routines in the DBMS_SQL module are useful when you want to construct and run dynamic SQL statements. For example, you might want execute DDL or DML statements such as "ALTER TABLE" or "DROP TABLE", construct and execute SQL statements on the fly, or call a function which uses dynamic SQL from within a SQL statement.

BIND_VARIABLE_BLOB procedure - Bind a BLOB value to a variable

The BIND_VARIABLE_BLOB procedure provides the capability to associate a BLOB value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type BLOB(2G) that specifies the value to be assigned.

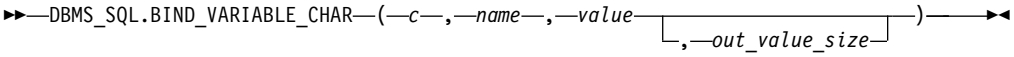
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_CHAR procedure - Bind a CHAR value to a variable

The BIND_VARIABLE_CHAR procedure provides the capability to associate a CHAR value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type CHAR(254) that specifies the value to be assigned.

out_value_size
An optional input argument of type INTEGER that specifies the length limit for the IN or INOUT argument, and the maximum length of the output value for the INOUT or OUT argument. If it is not specified, the length of *value* is assumed.

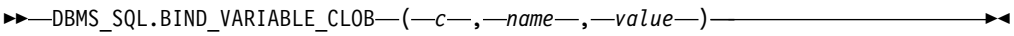
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_CLOB procedure - Bind a CLOB value to a variable

The BIND_VARIABLE_CLOB procedure provides the capability to associate a CLOB value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type CLOB(2G) that specifies the value to be assigned.

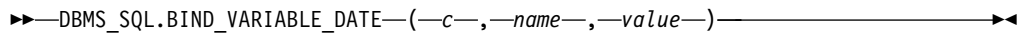
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_DATE procedure - Bind a DATE value to a variable

The BIND_VARIABLE_DATE procedure provides the capability to associate a DATE value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type DATE that specifies the value to be assigned.

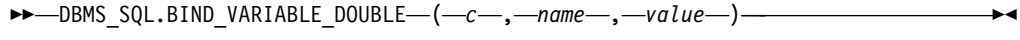
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_DOUBLE procedure - Bind a DOUBLE value to a variable

The BIND_VARIABLE_DOUBLE procedure provides the capability to associate a DOUBLE value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type DOUBLE that specifies the value to be assigned.

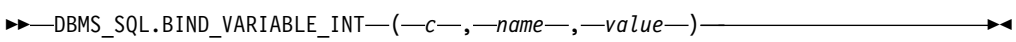
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_INT procedure - Bind an INTEGER value to a variable

The BIND_VARIABLE_INT procedure provides the capability to associate an INTEGER value with an IN or INOUT bind variable in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type INTEGER that specifies the value to be assigned.

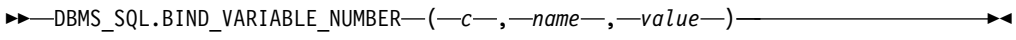
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_NUMBER procedure - Bind a NUMBER value to a variable

The BIND_VARIABLE_NUMBER procedure provides the capability to associate a NUMBER value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type DECFLOAT that specifies the value to be assigned.

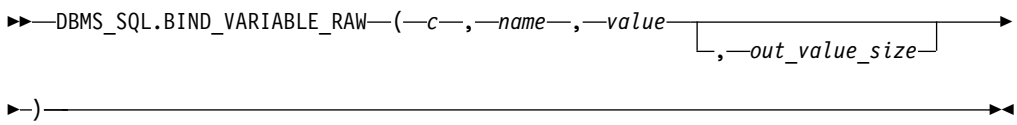
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_RAW procedure - Bind a RAW value to a variable

The BIND_VARIABLE_RAW procedure provides the capability to associate a RAW value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type BLOB(32767) that specifies the value to be assigned.

out_value_size
An optional input argument of type INTEGER that specifies the length limit for the IN or INOUT argument, and the maximum length of the output value for the INOUT or OUT argument. If it is not specified, the length of *value* is assumed.

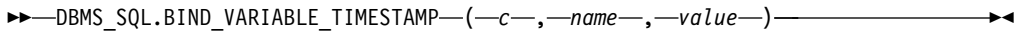
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_TIMESTAMP procedure - Bind a TIMESTAMP value to a variable

The BIND_VARIABLE_TIMESTAMP procedure provides the capability to associate a TIMESTAMP value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type TIMESTAMP that specifies the value to be assigned.

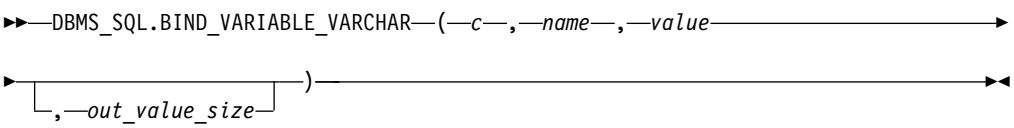
Authorization

EXECUTE privilege on the DBMS_SQL module.

BIND_VARIABLE_VARCHAR procedure - Bind a VARCHAR value to a variable

The BIND_VARIABLE_VARCHAR procedure provides the capability to associate a VARCHAR value with an IN, INOUT, or OUT argument in an SQL command.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID for the SQL command with bind variables.

name
An input argument of type VARCHAR(128) that specifies the name of the bind variable in the SQL command.

value
An input argument of type VARCHAR(32672) that specifies the value to be assigned.

out_value_size
An input argument of type INTEGER that specifies the length limit for the IN or INOUT argument, and the maximum length of the output value for the INOUT or OUT argument. If it is not specified, the length of *value* is assumed.

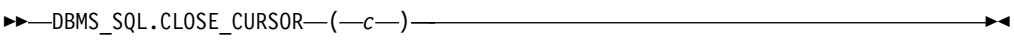
Authorization

EXECUTE privilege on the DBMS_SQL module.

CLOSE_CURSOR procedure - Close a cursor

The CLOSE_CURSOR procedure closes an open cursor. The resources allocated to the cursor are released and it cannot no longer be used.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor to be closed.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Examples

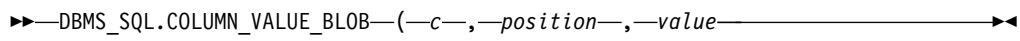
Example 1: This example illustrates closing a previously opened cursor.

```
DECLARE
    curid           INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
             .
             .
             .
    DBMS_SQL.CLOSE_CURSOR(curid);
END;
```

COLUMN_VALUE_BLOB procedure - Return a BLOB column value into a variable

The COLUMN_VALUE_BLOB procedure defines a variable that will receive a BLOB value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type BLOB(2G) that specifies the variable receiving the data returned by the cursor in a prior fetch call.

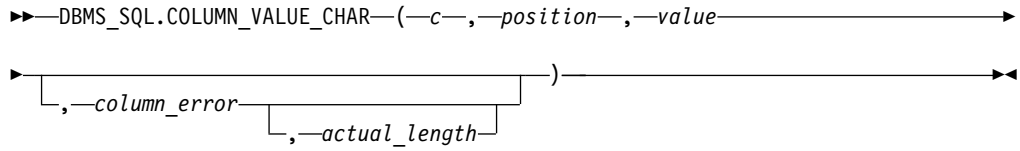
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_CHAR procedure - Return a CHAR column value into a variable

The COLUMN_VALUE_CHAR procedure defines a variable to receive a CHAR value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type CHAR that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An optional output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An optional output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

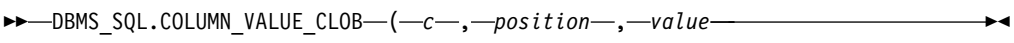
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_CLOB procedure - Return a CLOB column value into a variable

The COLUMN_VALUE_CLOB procedure defines a variable that will receive a CLOB value from a cursor.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type CLOB(2G) that specifies the variable receiving the data returned by the cursor in a prior fetch call.

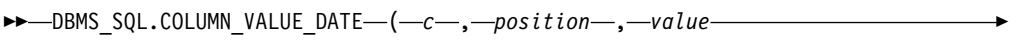
Authorization

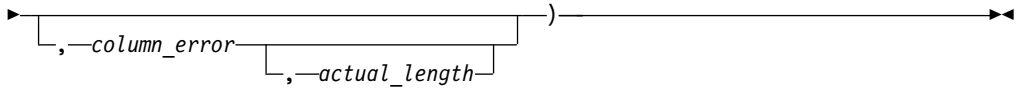
EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_DATE procedure - Return a DATE column value into a variable

The COLUMN_VALUE_DATE procedure defines a variable that will receive a DATE value from a cursor.

Syntax





Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type DATE that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

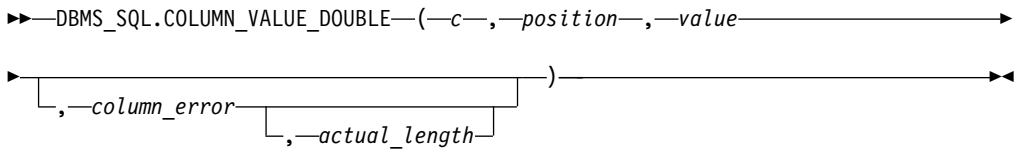
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_DOUBLE procedure - Return a DOUBLE column value into a variable

The COLUMN_VALUE_DOUBLE procedure defines a variable that will receive a DOUBLE value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type DOUBLE that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

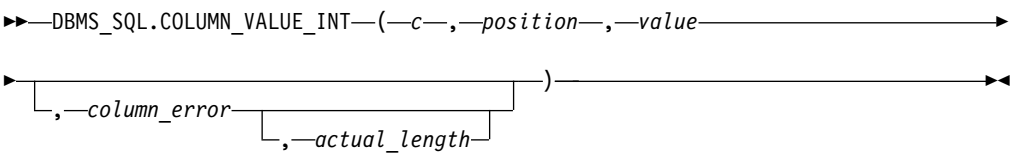
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_INT procedure - Return an INTEGER column value into a variable

The COLUMN_VALUE_INT procedure defines a variable that will receive a INTEGER value from a cursor.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type INTEGER that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

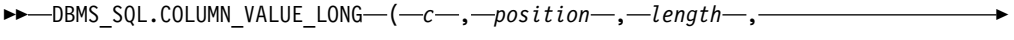
Authorization

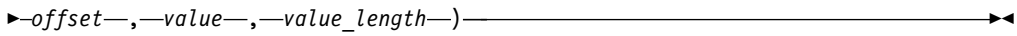
EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_LONG procedure - Return a LONG column value into a variable

The COLUMN_VALUE_LONG procedure defines a variable that will receive a portion of a LONG value from a cursor.

Syntax





Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

length
An input argument of type INTEGER that specifies the desired number of bytes of the LONG data to retrieve beginning at *offset*.

offset
An input argument of type INTEGER that specifies the position within the LONG value to start data retrieval.

value
An output argument of type CLOB(32760) that specifies the variable receiving the data returned by the cursor in a prior fetch call.

value_length
An output argument of type INTEGER that returns the actual length of the data returned.

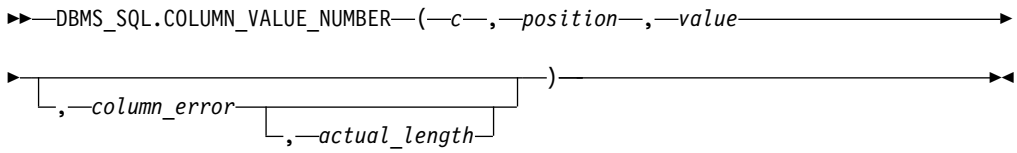
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_NUMBER procedure - Return a DECFLOAT column value into a variable

The COLUMN_VALUE_NUMBER procedure defines a variable that will receive a DECFLOAT value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type DECFLOAT that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An optional output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An optional output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

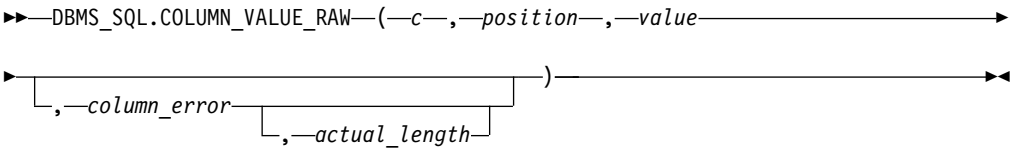
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_RAW procedure - Return a RAW column value into a variable

The COLUMN_VALUE_RAW procedure defines a variable that will receive a RAW value from a cursor.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type BLOB(32767) that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An optional output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An optional output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

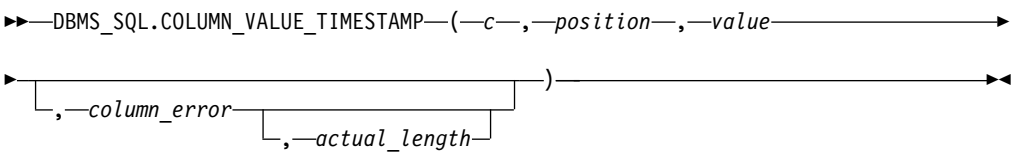
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_TIMESTAMP procedure - Return a TIMESTAMP column value into a variable

The COLUMN_VALUE_TIMESTAMP procedure defines a variable that will receive a TIMESTAMP value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type TIMESTAMP that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

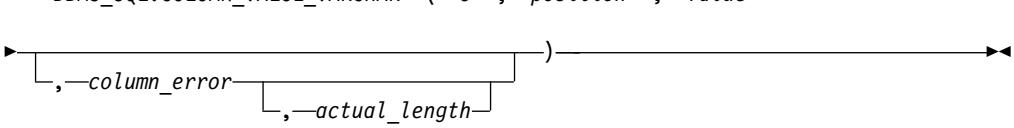
Authorization

EXECUTE privilege on the DBMS_SQL module.

COLUMN_VALUE_VARCHAR procedure - Return a VARCHAR column value into a variable

The COLUMN_VALUE_VARCHAR procedure defines a variable that will receive a VARCHAR value from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor that is returning data to the variable being defined.

position
An input argument of type INTEGER that specifies the position of the returned data within the cursor. The first value in the cursor is position 1.

value
An output argument of type VARCHAR(32672) that specifies the variable receiving the data returned by the cursor in a prior fetch call.

column_error
An output argument of type INTEGER that returns the SQLCODE, if any, associated with the column.

actual_length
An output argument of type INTEGER that returns the actual length of the data, prior to any truncation.

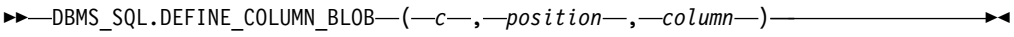
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_BLOB- Define a BLOB column in the SELECT list

The DEFINE_COLUMN_BLOB procedure defines a BLOB column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type BLOB(2G).

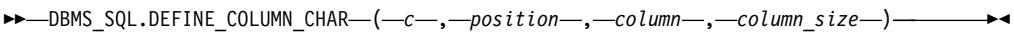
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_CHAR procedure - Define a CHAR column in the SELECT list

The DEFINE_COLUMN_CHAR procedure defines a CHAR column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type CHAR(254).

column_size
An input argument of type INTEGER that specifies the maximum length of the returned data. Returned data exceeding *column_size* is truncated to *column_size* characters.

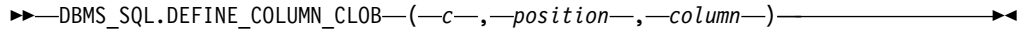
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_CLOB - Define a CLOB column in the SELECT list

The DEFINE_COLUMN_CLOB procedure defines a CLOB column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type CLOB(2G).

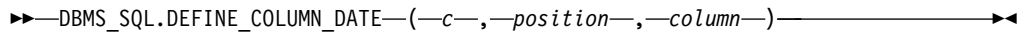
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_DATE - Define a DATE column in the SELECT list

The DEFINE_COLUMN_DATE procedure defines a DATE column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type DATE.

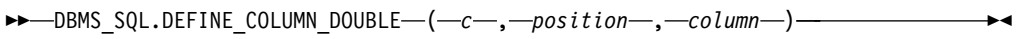
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_DOUBLE - Define a DOUBLE column in the SELECT list

The DEFINE_COLUMN_DOUBLE procedure defines a DOUBLE column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type DOUBLE.

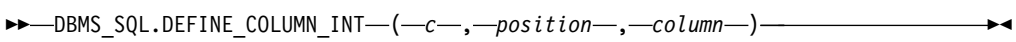
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_INT- Define an INTEGER column in the SELECT list

The DEFINE_COLUMN_INT procedure defines an INTEGER column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type INTEGER.

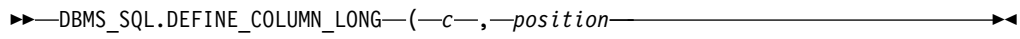
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_LONG procedure - Define a LONG column in the SELECT list

The DEFINE_COLUMN_LONG procedure defines a LONG column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

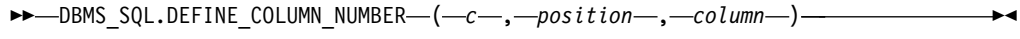
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_NUMBER procedure - Define a DECFLOAT column in the SELECT list

The DEFINE_COLUMN_NUMBER procedure defines a DECFLOAT column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type DECFLOAT.

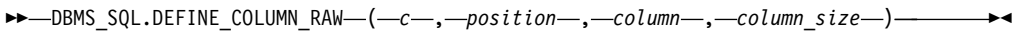
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_RAW procedure - Define a RAW column or expression in the SELECT list

The DEFINE_COLUMN_RAW procedure defines a RAW column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type BLOB(32767).

column_size
An input argument of type INTEGER that specifies the maximum length of the returned data. Returned data exceeding *column_size* is truncated to *column_size* characters.

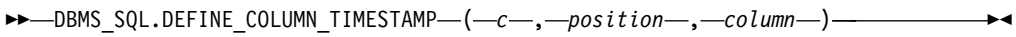
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_TIMESTAMP - Define a TIMESTAMP column in the SELECT list

The DEFINE_COLUMN_TIMESTAMP procedure defines a TIMESTAMP column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type TIMESTAMP.

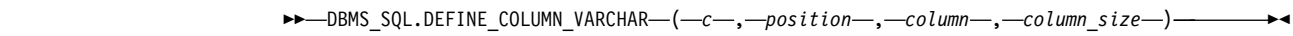
Authorization

EXECUTE privilege on the DBMS_SQL module.

DEFINE_COLUMN_VARCHAR procedure - Define a VARCHAR column in the SELECT list

The DEFINE_COLUMN_VARCHAR procedure defines a VARCHAR column or expression in the SELECT list that is to be returned and retrieved in a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor handle associated with the SELECT command.

position
An input argument of type INTEGER that specifies the position of the column or expression in the SELECT list that is being defined.

column
An input argument of type VARCHAR(32672).

column_size
An input argument of type INTEGER that specifies the maximum length of the returned data. Returned data exceeding *column_size* is truncated to *column_size* characters.

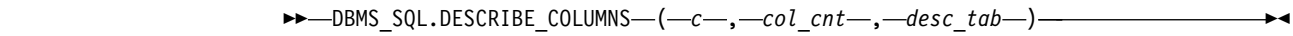
Authorization

EXECUTE privilege on the DBMS_SQL module.

DESCRIBE_COLUMNS procedure - Retrieve a description of the columns in a SELECT list

The DESCRIBE_COLUMNS procedure provides the capability to retrieve a description of the columns in a SELECT list from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor whose columns are to be described.

col_cnt
An output argument of type INTEGER that returns the number of columns in the SELECT list of the cursor.

desc_tab
An output argument of type DESC_TAB that describes the column metadata. The DESC_TAB array provides information on each column in the specified cursor.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure requires a user temporary table space with a page size of 4K; otherwise it returns an SQL0286N error. You can create the user temporary table space with this command:

```
CREATE USER TEMPORARY TABLESPACE DBMS_SQL_TEMP_TBS
```

DESC_TAB is an array of DESC_REC records of column information:

Table 16. DESC_TAB definition through DESC_REC records

Record name	Description
col_type	SQL data type as defined in Supported SQL data types in C and C++ embedded SQL applications.
col_max_len	Maximum length of the column.
col_name	Column name.
col_name_len	Length of the column name.
col_schema	Always NULL.
col_schema_name_len	Always NULL.
col_precision	Precision of the column as defined in the database. If col_type denotes a graphic or DBCLOB SQL data type, then this variable indicates the maximum number of double-byte characters the column can hold.
col_scale	Scale of the column as defined in the database (only applies to DECIMAL, NUMERIC, TIMESTAMP).
col_charsetid	Always NULL.
col_charsetform	Always NULL.
col_null_ok	Nullable indicator. This has a value of 1 if the column is nullable, otherwise, 0.

```
ALTER MODULE SYSIBMADM.DBMS_SQL PUBLISH TYPE DESC_REC AS ROW
(
  col_type INTEGER,
  col_max_len INTEGER,
  col_name VARCHAR(128),
  col_name_len INTEGER,
  col_schema_name VARCHAR(128),
  col_schema_name_len INTEGER,
  col_precision INTEGER,
  col_scale INTEGER,
  col_charsetid INTEGER,
  col_charsetform INTEGER,
  col_null_ok INTEGER
);

ALTER MODULE SYSIBMADM.DBMS_SQL PUBLISH TYPE DESC_TAB AS DESC_REC ARRAY[INTEGER];
```

Examples

Example 1: The following example describes the `empno`, `ename`, `hiredate`, and `sal` columns from the "EMP" table.

```
SET SERVEROUTPUT ON@

BEGIN
```

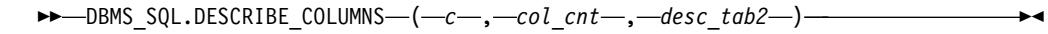
```
  DECLARE handle INTEGER;
  DECLARE col_cnt INTEGER;
  DECLARE col DBMS_SQL.DESC_TAB;
  DECLARE i INTEGER DEFAULT 1;
  DECLARE CUR1 CURSOR FOR S1;

  CALL DBMS_SQL.OPEN_CURSOR( handle );
  CALL DBMS_SQL.PARSE( handle,
      'SELECT empno, firstnme, lastname, salary
        FROM employee', DBMS_SQL.NATIVE );
  CALL DBMS_SQL.DESCRIBE_COLUMNS( handle, col_cnt, col );

  IF col_cnt > 0 THEN
    CALL DBMS_OUTPUT.PUT_LINE( 'col_cnt = ' || col_cnt );
    CALL DBMS_OUTPUT.NEW_LINE();
    fetchLoop: LOOP
      IF i > col_cnt THEN
        LEAVE fetchLoop;
      END IF;

      CALL DBMS_OUTPUT.PUT_LINE( 'i = ' || i );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_name = ' || col[i].col_name );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_name_len = ' ||
          NVL(col[i].col_name_len, 'NULL') );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_schema_name = ' ||
          NVL( col[i].col_schema_name, 'NULL' ) );

      IF col[i].col_schema_name_len IS NULL THEN
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_schema_name_len = NULL' );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_schema_name_len = ' ||
            col[i].col_schema_name_len);
      END IF;

      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_type = ' || col[i].col_type );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_max_len = ' || col[i].col_max_len );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_precision = ' || col[i].col_precision );
      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_scale = ' || col[i].col_scale );

      IF col[i].col_charsetid IS NULL THEN
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_charsetid = NULL' );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_charsetid = ' || col[i].col_charsetid );
      END IF;

      IF col[i].col_charsetform IS NULL THEN
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_charsetform = NULL' );
      ELSE
        CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_charsetform = ' || col[i].col_charsetform );
      END IF;

      CALL DBMS_OUTPUT.PUT_LINE( 'col[i].col_null_ok = ' || col[i].col_null_ok );
      CALL DBMS_OUTPUT.NEW_LINE();
      SET i = i + 1;
    END LOOP;
  END IF;
END@

Output:
col_cnt = 4

i = 1
col[i].col_name = EMPNO
col[i].col_name_len = 5
col[i].col_schema_name = NULL
col[i].col_schema_name_len = NULL
col[i].col_type = 452
```

```
col[i].col_max_len = 6
col[i].col_precision = 6
col[i].col_scale = 0
col[i].col_charsetid = NULL
col[i].col_charsetform = NULL
col[i].col_null_ok = 0

i = 2
col[i].col_name = FIRSTNME
col[i].col_name_len = 8
col[i].col_schema_name = NULL
col[i].col_schema_name_len = NULL
col[i].col_type = 448
col[i].col_max_len = 12
col[i].col_precision = 12
col[i].col_scale = 0
col[i].col_charsetid = NULL
col[i].col_charsetform = NULL
col[i].col_null_ok = 0

i = 3
col[i].col_name = LASTNAME
col[i].col_name_len = 8
col[i].col_schema_name = NULL
col[i].col_schema_name_len = NULL
col[i].col_type = 448
col[i].col_max_len = 15
col[i].col_precision = 15
col[i].col_scale = 0
col[i].col_charsetid = NULL
col[i].col_charsetform = NULL
col[i].col_null_ok = 0

i = 4
col[i].col_name = SALARY
col[i].col_name_len = 6
col[i].col_schema_name = NULL
col[i].col_schema_name_len = NULL
col[i].col_type = 484
col[i].col_max_len = 5
col[i].col_precision = 9
col[i].col_scale = 2
col[i].col_charsetid = NULL
col[i].col_charsetform = NULL
col[i].col_null_ok = 1
```

DESCRIBE_COLUMNS2 procedure - Retrieve a description of column names in a SELECT list

The DESCRIBE_COLUMNS2 procedure provides the capability to retrieve a description of the columns in a SELECT list from a cursor.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor whose columns are to be described.

col_cnt
An output argument of type INTEGER that returns the number of columns in the SELECT list of the cursor.

desc_tab
An output argument of type DESC_TAB2 that describes the column metadata. The DESC_TAB2 array provides information on each column in the specified cursor

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure requires a user temporary table space with a page size of 4K; otherwise it returns an SQL0286N error. You can create the user temporary table space with this command:

```
CREATE USER TEMPORARY TABLESPACE DBMS_SQL_TEMP_TBS
```

DESC_TAB2 is an array of DESC_REC2 records of column information:

Table 17. DESC_TAB2 definition through DESC_REC2 records

Record name	Description
col_type	SQL data type as defined in Supported SQL data types in C and C++ embedded SQL applications.
col_max_len	Maximum length of the column.
col_name	Column name.
col_name_len	Length of the column name.
col_schema	Always NULL.
col_schema_name_len	Always NULL.
col_precision	Precision of the column as defined in the database. If col_type denotes a graphic or DBCLOB SQL data type, then this variable indicates the maximum number of double-byte characters the column can hold.
col_scale	Scale of the column as defined in the database (only applies to DECIMAL, NUMERIC, TIMESTAMP).
col_charsetid	Always NULL.
col_charsetform	Always NULL.
col_null_ok	Nullable indicator. This has a value of 1 if the column is nullable, otherwise, 0.

```
ALTER MODULE SYSIBMADM.DBMS_SQL PUBLISH TYPE DESC_REC2 AS ROW
(
col_type INTEGER,
col_max_len INTEGER,
col_name VARCHAR(128),
col_name_len INTEGER,
col_schema_name VARCHAR(128),
col_schema_name_len INTEGER,
col_precision INTEGER,
col_scale INTEGER,
```

```
col_charsetid INTEGER,
col_charsetform INTEGER,
col_null_ok INTEGER
);

ALTER MODULE SYSIBMADM.DBMS_SQL PUBLISH TYPE DESC_TAB2 AS DESC_REC2 ARRAY[INTEGER];
```

EXECUTE procedure - Run a parsed SQL statement

The EXECUTE function executes a parsed SQL statement.

Syntax

Parameters

c An input argument of type INTEGER that specifies the cursor ID of the parsed SQL statement to be executed.

ret
An output argument of type INTEGER that returns the number of rows processed if the SQL command is DELETE, INSERT, or UPDATE; otherwise it returns 0.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following anonymous block inserts a row into the "DEPT" table.

```
SET SERVEROUTPUT ON@

CREATE TABLE dept (
  deptno DECIMAL(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
  dname  VARCHAR(14) NOT NULL,
  loc    VARCHAR(13),
  CONSTRAINT dept_dname_uq UNIQUE( deptno, dname )
)@

BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'INSERT INTO dept VALUES (50, ''HR'', ''LOS ANGELES'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

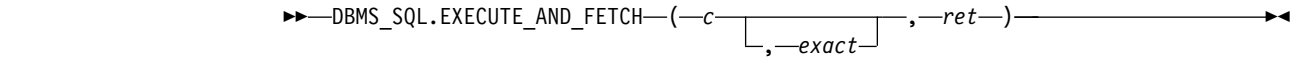
This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE TABLE dept
( deptno DECIMAL(2) NOT NULL CONSTRAINT dept_pk PRIMARY KEY,
  dname  VARCHAR(14) NOT NULL,
  loc    VARCHAR(13),
  CONSTRAINT dept_dname_uq UNIQUE( deptno, dname ) )
DB20000I  The SQL command completed successfully.

BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'INSERT INTO dept VALUES (50, ''HR'', ''LOS ANGELES'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.

Number of rows processed: 1
```

EXECUTE_AND_FETCH procedure - Run a parsed SELECT command and fetch one row

The EXECUTE_AND_FETCH procedure executes a parsed SELECT command and fetches one row.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor id of the cursor for the SELECT command to be executed.

exact
An optional argument of type INTEGER. If set to 1, an exception is thrown if the number of rows in the result set is not exactly equal to 1. If set to 0, no exception is thrown. The default is 0. A NO_DATA_FOUND (SQL0100W) exception is thrown if *exact* is set to 1 and there are no rows in the result set. A TOO_MANY_ROWS (SQL0811N) exception is thrown if *exact* is set to 1 and there is more than one row in the result set.

ret
An output argument of type INTEGER that returns 1 if a row was fetched successfully, 0 if there are no rows to fetch.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following stored procedure uses the EXECUTE_AND_FETCH function to retrieve one employee using the employee's name. An exception will be thrown if the employee is not found, or there is more than one employee with the same name.

```
SET SERVEROUTPUT ON@

CREATE TABLE emp (
  empno    DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
  ename    VARCHAR(10),
  job      VARCHAR(9),
  mgr      DECIMAL(4),
  hiredate TIMESTAMP(0),
  sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
  comm     DECIMAL(7,2) )@

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)@
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)@
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)@
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)@
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)@

CREATE OR REPLACE PROCEDURE select_by_name(
IN p_ename ANCHOR TO emp.ename)
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_empno ANCHOR TO emp.empno;
  DECLARE v_hiredate ANCHOR TO emp.hiredate;
  DECLARE v_sal ANCHOR TO emp.sal;
  DECLARE v_comm ANCHOR TO emp.comm;
  DECLARE v_disp_date VARCHAR(10);
  DECLARE v_sql VARCHAR(120);
  DECLARE v_status INTEGER;
  SET v_sql = 'SELECT empno, hiredate, sal, NVL(comm, 0)
      FROM emp e WHERE ename = :p_ename ';
  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.BIND_VARIABLE_VARCHAR(curid, ':p_ename', UPPER(p_ename));
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 2, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 3, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_comm);
  CALL DBMS_SQL.EXECUTE_AND_FETCH(curid, 1 /*True*/, v_status);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 2, v_hiredate);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 3, v_sal);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_comm);
  SET v_disp_date = TO_CHAR(v_hiredate, 'MM/DD/YYYY');
  CALL DBMS_OUTPUT.PUT_LINE('Number    : ' || v_empno);
  CALL DBMS_OUTPUT.PUT_LINE('Name      : ' || UPPER(p_ename));
  CALL DBMS_OUTPUT.PUT_LINE('Hire Date : ' || v_disp_date);
  CALL DBMS_OUTPUT.PUT_LINE('Salary    : ' || v_sal);
  CALL DBMS_OUTPUT.PUT_LINE('Commission: ' || v_comm);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@

CALL select_by_name( 'MARTIN' )@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE TABLE emp
 ( empno    DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
   ename    VARCHAR(10),
   job      VARCHAR(9),
   mgr      DECIMAL(4),
   hiredate TIMESTAMP(0),
   sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
   comm     DECIMAL(7,2) )
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)
DB20000I  The SQL command completed successfully.

CREATE OR REPLACE PROCEDURE select_by_name(
IN p_ename ANCHOR TO emp.ename)
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_empno ANCHOR TO emp.empno;
  DECLARE v_hiredate ANCHOR TO emp.hiredate;
  DECLARE v_sal ANCHOR TO emp.sal;
  DECLARE v_comm ANCHOR TO emp.comm;
  DECLARE v_disp_date VARCHAR(10);
  DECLARE v_sql VARCHAR(120);
  DECLARE v_status INTEGER;
  SET v_sql = 'SELECT empno, hiredate, sal, NVL(comm, 0)
      FROM emp e WHERE ename = :p_ename ';
  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.BIND_VARIABLE_VARCHAR(curid, ':p_ename', UPPER(p_ename));
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 2, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 3, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_comm);
  CALL DBMS_SQL.EXECUTE_AND_FETCH(curid, 1 /*True*/, v_status);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 2, v_hiredate);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 3, v_sal);
  CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_comm);
  SET v_disp_date = TO_CHAR(v_hiredate, 'MM/DD/YYYY');
  CALL DBMS_OUTPUT.PUT_LINE('Number    : ' || v_empno);
  CALL DBMS_OUTPUT.PUT_LINE('Name      : ' || UPPER(p_ename));
  CALL DBMS_OUTPUT.PUT_LINE('Hire Date : ' || v_disp_date);
  CALL DBMS_OUTPUT.PUT_LINE('Salary    : ' || v_sal);
  CALL DBMS_OUTPUT.PUT_LINE('Commission: ' || v_comm);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.

CALL select_by_name( 'MARTIN' )

  Return Status = 0
```

```
Number    : 7654
Name      : MARTIN
Hire Date : 09/28/1981
Salary    : 1250.00
Commission: 1400.00
```

FETCH_ROWS procedure - Retrieve a row from a cursor

The FETCH_ROWS function retrieves a row from a cursor

Syntax

Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor from which to fetch a row.

ret
An output argument of type INTEGER that returns 1 if a row was fetched successfully, 0 if there are no rows to fetch.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following examples fetches the rows from the "EMP" table and displays the results.

```
SET SERVEROUTPUT ON@

CREATE TABLE emp (
  empno    DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
  ename    VARCHAR(10),
  job      VARCHAR(9),
  mgr      DECIMAL(4),
  hiredate TIMESTAMP(0),
  sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
  comm     DECIMAL(7,2) )@

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)@
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)@
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)@
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)@
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)@

BEGIN
  DECLARE curid INTEGER;
  DECLARE v_empno DECIMAL(4);
  DECLARE v_ename VARCHAR(10);
  DECLARE v_hiredate DATE;
  DECLARE v_sal DECIMAL(7, 2);
  DECLARE v_comm DECIMAL(7, 2);
```

```
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;
  DECLARE v_rowcount INTEGER;

  SET v_sql = 'SELECT empno, ename, hiredate, sal, ' || 'comm FROM emp';

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_VARCHAR(curid, 2, v_ename, 10);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 3, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 5, v_comm);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME       HIREDATE    SAL
        COMM');
  CALL DBMS_OUTPUT.PUT_LINE('-----  ----------  ----------  --------
       ' || '--------');

  FETCH_LOOP: LOOP
    CALL DBMS_SQL.FETCH_ROWS(curid, v_status);

    IF v_status = 0 THEN
      LEAVE FETCH_LOOP;
    END IF;

    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
    CALL DBMS_SQL.COLUMN_VALUE_VARCHAR(curid, 2, v_ename);
    CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 3, v_hiredate);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_sal);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 5, v_comm);
    CALL DBMS_OUTPUT.PUT_LINE(v_empno || '   ' ||
            RPAD(v_ename, 10) || '  ' || TO_CHAR(v_hiredate,
            'yyyy-mm-dd') || ' ' || TO_CHAR(v_sal,
            '9,999.99') || ' ' || TO_CHAR(NVL(v_comm, 0),
            '9,999.99'));
  END LOOP FETCH_LOOP;

  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

This example results in the following output:

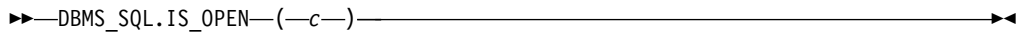
```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE TABLE emp (empno  DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
    ename    VARCHAR(10), job      VARCHAR(9), mgr      DECIMAL(4),
    hiredate TIMESTAMP(0),
    sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
    comm     DECIMAL(7,2) )
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)
DB20000I  The SQL command completed successfully.
```

```
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_empno DECIMAL(4);
  DECLARE v_ename VARCHAR(10);
  DECLARE v_hiredate DATE;
  DECLARE v_sal DECIMAL(7, 2);
  DECLARE v_comm DECIMAL(7, 2);
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;
  DECLARE v_rowcount INTEGER;

  SET v_sql = 'SELECT empno, ename, hiredate, sal, ' || 'comm FROM emp';

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_VARCHAR(curid, 2, v_ename, 10);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 3, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 5, v_comm);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME       HIREDATE    SAL
      COMM');
  CALL DBMS_OUTPUT.PUT_LINE('-----  ----------  ----------  --------
      ' || '--------');

  FETCH_LOOP: LOOP
    CALL DBMS_SQL.FETCH_ROWS(curid, v_status);

    IF v_status = 0 THEN
      LEAVE FETCH_LOOP;
    END IF;

    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
    CALL DBMS_SQL.COLUMN_VALUE_VARCHAR(curid, 2, v_ename);
    CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 3, v_hiredate);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_sal);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 5, v_comm);
    CALL DBMS_OUTPUT.PUT_LINE(v_empno || '   ' || RPAD(v_ename,
        10) || '  ' || TO_CHAR(v_hiredate,
        'yyyy-mm-dd') || ' ' || TO_CHAR(v_sal,
        '9,999.99') || ' ' || TO_CHAR(NVL(v_comm,
         0), '9,999.99'));
  END LOOP FETCH_LOOP;

  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.

EMPNO  ENAME       HIREDATE    SAL       COMM
-----  ----------  ----------  --------  --------
7369   SMITH       1980-12-17    800.00      0.00
7499   ALLEN       1981-02-20  1,600.00    300.00
7521   WARD        1981-02-22  1,250.00    500.00
7566   JONES       1981-04-02  2,975.00      0.00
7654   MARTIN      1981-09-28  1,250.00  1,400.00
```

IS_OPEN function - Check whether a cursor is open

The IS_OPEN function tests whether a specified cursor is open.

Syntax



Parameters

c An input argument of type INTEGER that specifies the ID of the cursor to be tested.

ret
An output argument of type BOOLEAN that returns a value of `TRUE` if the specified cursor is open and `FALSE` if the cursor is closed.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

You can call this function with the function invocation syntax in a PL/SQL assignment statement.

Examples

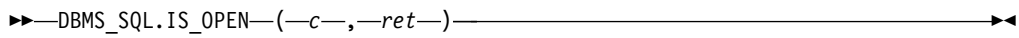
In the following example, the DBMS_SQL.IS_OPEN function is called to determine whether the cursor that is specified in the *cur* argument is open:

```
DECLARE rc boolean;

SET rc = DBMS_SQL.IS_OPEN(cur);
```

IS_OPEN procedure - Check whether a cursor is open

The IS_OPEN procedure tests whether a specified cursor is open.

Syntax



Parameters

c An input argument of type INTEGER that specifies the ID of the cursor to be tested.

ret
An output argument of type INTEGER that returns a value of `1` if the specified cursor is open and `0` if the cursor is closed.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Examples

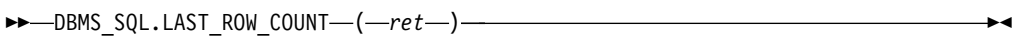
In the following example, the DBMS_SQL.IS_OPEN procedure is called to determine whether the cursor that is specified in the *cur* argument is open:

```
DECLARE rc integer;

CALL DBMS_SQL.IS_OPEN(cur, rc);
```

LAST_ROW_COUNT procedure - return the cumulative number of rows fetched

The LAST_ROW_COUNT procedure returns the number of rows that have been fetched.

Syntax



Parameters

ret
An output argument of type INTEGER that returns the number of rows that have been fetched so far in the current session. A call to DBMS_SQL.PARSE resets the counter.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following example uses the LAST_ROW_COUNT procedure to display the total number of rows fetched in the query.

```
SET SERVEROUTPUT ON@

CREATE TABLE emp (
  empno    DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
  ename    VARCHAR(10),
  job      VARCHAR(9),
  mgr      DECIMAL(4),
  hiredate TIMESTAMP(0),
  sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
  comm     DECIMAL(7,2) )@

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)@
INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)@
INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)@
INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)@
INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)@

BEGIN
  DECLARE curid INTEGER;
  DECLARE v_empno DECIMAL(4);
  DECLARE v_ename VARCHAR(10);
  DECLARE v_hiredate DATE;
  DECLARE v_sal DECIMAL(7, 2);
  DECLARE v_comm DECIMAL(7, 2);
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;
  DECLARE v_rowcount INTEGER;

  SET v_sql = 'SELECT empno, ename, hiredate, sal, ' || 'comm FROM emp';
```

```
  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_VARCHAR(curid, 2, v_ename, 10);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 3, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 5, v_comm);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME       HIREDATE    SAL
     COMM');
  CALL DBMS_OUTPUT.PUT_LINE('-----  ----------  ----------  --------
     ' || '--------');

  FETCH_LOOP: LOOP
    CALL DBMS_SQL.FETCH_ROWS(curid, v_status);

    IF v_status = 0 THEN
      LEAVE FETCH_LOOP;
    END IF;

    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
    CALL DBMS_SQL.COLUMN_VALUE_VARCHAR(curid, 2, v_ename);
    CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 3, v_hiredate);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_sal);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 5, v_comm);
    CALL DBMS_OUTPUT.PUT_LINE(v_empno || '   ' || RPAD(v_ename,
       10) || '  ' || TO_CHAR(v_hiredate,
       'yyyy-mm-dd') || ' ' || TO_CHAR(v_sal,
       '9,999.99') || ' ' || TO_CHAR(NVL(v_comm,
       0), '9,999.99'));
  END LOOP FETCH_LOOP;

  CALL DBMS_SQL.LAST_ROW_COUNT( v_rowcount );
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows: ' || v_rowcount);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE TABLE emp ( empno    DECIMAL(4) NOT NULL CONSTRAINT emp_pk PRIMARY KEY,
   ename    VARCHAR(10), job      VARCHAR(9),
   mgr      DECIMAL(4),
   hiredate TIMESTAMP(0),
   sal      DECIMAL(7,2) CONSTRAINT emp_sal_ck CHECK (sal > 0),
   comm     DECIMAL(7,2) )
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7369,'SMITH','CLERK',7902,'1980-12-17',800,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7499,'ALLEN','SALESMAN',7698,'1981-02-20',1600,300)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7521,'WARD','SALESMAN',7698,'1981-02-22',1250,500)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7566,'JONES','MANAGER',7839,'1981-04-02',2975,NULL)
DB20000I  The SQL command completed successfully.

INSERT INTO emp VALUES (7654,'MARTIN','SALESMAN',7698,'1981-09-28',1250,1400)
DB20000I  The SQL command completed successfully.

BEGIN
  DECLARE curid INTEGER;
```

```
  DECLARE v_empno DECIMAL(4);
  DECLARE v_ename VARCHAR(10);
  DECLARE v_hiredate DATE;
  DECLARE v_sal DECIMAL(7, 2);
  DECLARE v_comm DECIMAL(7, 2);
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;
  DECLARE v_rowcount INTEGER;

  SET v_sql = 'SELECT empno, ename, hiredate, sal, ' || 'comm FROM emp';

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 1, v_empno);
  CALL DBMS_SQL.DEFINE_COLUMN_VARCHAR(curid, 2, v_ename, 10);
  CALL DBMS_SQL.DEFINE_COLUMN_DATE(curid, 3, v_hiredate);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 4, v_sal);
  CALL DBMS_SQL.DEFINE_COLUMN_NUMBER(curid, 5, v_comm);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('EMPNO  ENAME       HIREDATE    SAL
     COMM');
  CALL DBMS_OUTPUT.PUT_LINE('-----  ----------  ----------  --------
     ' || '--------');

  FETCH_LOOP: LOOP
    CALL DBMS_SQL.FETCH_ROWS(curid, v_status);

    IF v_status = 0 THEN
      LEAVE FETCH_LOOP;
    END IF;

    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 1, v_empno);
    CALL DBMS_SQL.COLUMN_VALUE_VARCHAR(curid, 2, v_ename);
    CALL DBMS_SQL.COLUMN_VALUE_DATE(curid, 3, v_hiredate);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 4, v_sal);
    CALL DBMS_SQL.COLUMN_VALUE_NUMBER(curid, 5, v_comm);
    CALL DBMS_OUTPUT.PUT_LINE(
      v_empno || '   ' || RPAD(v_ename, 10) || '  ' || TO_CHAR(v_hiredate,
      'yyyy-mm-dd') || ' ' || TO_CHAR(v_sal,
      '9,999.99') || ' ' || TO_CHAR(NVL(v_comm,
      0), '9,999.99'));
  END LOOP FETCH_LOOP;

  CALL DBMS_SQL.LAST_ROW_COUNT( v_rowcount );
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows: ' || v_rowcount);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.

EMPNO  ENAME       HIREDATE    SAL       COMM
-----  ----------  ----------  --------  --------
7369   SMITH       1980-12-17    800.00      0.00
7499   ALLEN       1981-02-20  1,600.00    300.00
7521   WARD        1981-02-22  1,250.00    500.00
7566   JONES       1981-04-02  2,975.00      0.00
7654   MARTIN      1981-09-28  1,250.00  1,400.00
Number of rows: 5
```

OPEN_CURSOR procedure - Open a cursor

The OPEN_CURSOR procedure creates a new cursor.

A cursor must be used to parse and execute any dynamic SQL statement. Once a cursor has been opened, it can be used again with the same or different SQL statements. The cursor does not have to be closed and reopened in order to be used again.

Syntax

Parameters

c An output argument of type INTEGER that specifies the cursor ID of the newly created cursor.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following example creates a new cursor:

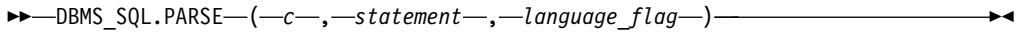
```
DECLARE
    curid           INTEGER;
BEGIN
    curid := DBMS_SQL.OPEN_CURSOR;
             .
             .
             .
END;
```

PARSE procedure - Parse an SQL statement

The PARSE procedure parses an SQL statement.

If the SQL command is a DDL command, it is immediately executed and does not require running the EXECUTE procedure.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of an open cursor.

statement
The SQL statement to be parsed.

language_flag
This argument is provided for Oracle syntax compatibility. Use a value of 1 or `DBMS_SQL.native`.

Authorization

EXECUTE privilege on the DBMS_SQL module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

Examples

Example 1: The following anonymous block creates a table named job. Note that DDL statements are executed immediately by the PARSE procedure and do not require a separate EXECUTE step.

```
SET SERVEROUTPUT ON@

BEGIN
  DECLARE curid INTEGER;
  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, 'CREATE TABLE job (jobno DECIMAL(3),
     ' || 'jname VARCHAR(9))', DBMS_SQL.native);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

BEGIN
  DECLARE curid INTEGER;
  CALL DBMS_SQL.OPEN_CURSOR(curid);
  CALL DBMS_SQL.PARSE(curid, 'CREATE TABLE job (jobno DECIMAL(3), ' ||
    'jname VARCHAR(9))', DBMS_SQL.native);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.
```

Example 2: The following inserts two rows into the job table.

```
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'INSERT INTO job VALUES (100, ''ANALYST'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
  SET v_sql = 'INSERT INTO job VALUES (200, ''CLERK'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

This example results in the following output:

```
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(50);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'INSERT INTO job VALUES (100, ''ANALYST'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
```

```
  SET v_sql = 'INSERT INTO job VALUES (200, ''CLERK'')';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.EXECUTE(curid, v_status);
  CALL DBMS_OUTPUT.PUT_LINE('Number of rows processed: ' || v_status);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.

Number of rows processed: 1
Number of rows processed: 1
```

Example 3: The following anonymous block uses the DBMS_SQL module to execute a block containing two INSERT statements. Note that the end of the block contains a terminating semicolon, whereas in the prior examples, the individual INSERT statements did not have a terminating semicolon.

```
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(100);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'BEGIN ' || 'INSERT INTO job VALUES (300, ''MANAGER''); '
       || 'INSERT INTO job VALUES (400, ''SALESMAN''); ' || 'END;';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END@
```

This example results in the following output:

```
BEGIN
  DECLARE curid INTEGER;
  DECLARE v_sql VARCHAR(100);
  DECLARE v_status INTEGER;

  CALL DBMS_SQL.OPEN_CURSOR(curid);
  SET v_sql = 'BEGIN ' || 'INSERT INTO job VALUES (300, ''MANAGER''); ' ||
    'INSERT INTO job VALUES (400, ''SALESMAN''); ' || 'END;';
  CALL DBMS_SQL.PARSE(curid, v_sql, DBMS_SQL.native);
  CALL DBMS_SQL.CLOSE_CURSOR(curid);
END
DB20000I  The SQL command completed successfully.
```

VARIABLE_VALUE_BLOB procedure - Return the value of a BLOB INOUT or OUT parameter

The VARIABLE_VALUE_BLOB procedure provides the capability to return the value of a BLOB INOUT or OUT parameter.

Syntax

Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type BLOB(2G) that specifies the variable receiving the value.

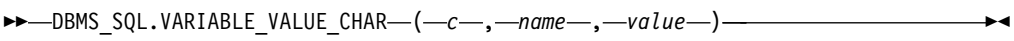
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_CHAR procedure - Return the value of a CHAR INOUT or OUT parameter

The VARIABLE_VALUE_CHAR procedure provides the capability to return the value of a CHAR INOUT or OUT parameter.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type CHAR(254) that specifies the variable receiving the value.

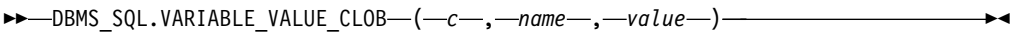
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_CLOB procedure - Return the value of a CLOB INOUT or OUT parameter

The VARIABLE_VALUE_CLOB procedure provides the capability to return the value of a CLOB INOUT or OUT parameter.

Syntax



Parameters

c
An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type CLOB(2G) that specifies the variable receiving the value.

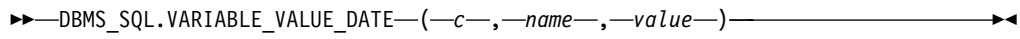
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_DATE procedure - Return the value of a DATE INOUT or OUT parameter

The VARIABLE_VALUE_DATE procedure provides the capability to return the value of a DATE INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type DATE that specifies the variable receiving the value.

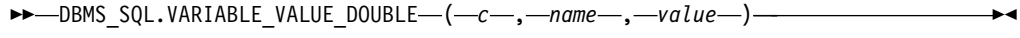
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_DOUBLE procedure - Return the value of a DOUBLE INOUT or OUT parameter

The VARIABLE_VALUE_DOUBLE procedure provides the capability to return the value of a DOUBLE INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type DOUBLE that specifies the variable receiving the value.

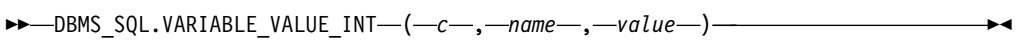
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_INT procedure - Return the value of an INTEGER INOUT or OUT parameter

The VARIABLE_VALUE_INT procedure provides the capability to return the value of a INTEGER INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type INTEGER that specifies the variable receiving the value.

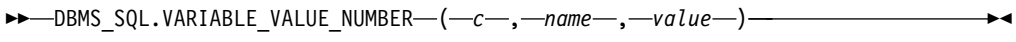
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_NUMBER procedure - Return the value of a DECFLOAT INOUT or OUT parameter

The VARIABLE_VALUE_NUMBER procedure provides the capability to return the value of a DECFLOAT INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type DECFLOAT that specifies the variable receiving the value.

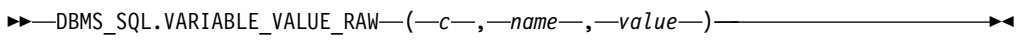
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_RAW procedure - Return the value of a BLOB(32767) INOUT or OUT parameter

The VARIABLE_VALUE_RAW procedure provides the capability to return the value of a BLOB(32767) INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type BLOB(32767) that specifies the variable receiving the value.

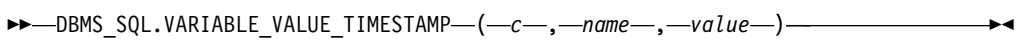
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_TIMESTAMP procedure - Return the value of a TIMESTAMP INOUT or OUT parameter

The VARIABLE_VALUE_TIMESTAMP procedure provides the capability to return the value of a TIMESTAMP INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type TIMESTAMP that specifies the variable receiving the value.

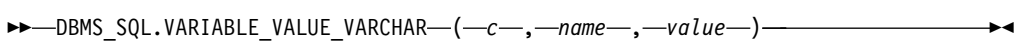
Authorization

EXECUTE privilege on the DBMS_SQL module.

VARIABLE_VALUE_VARCHAR procedure - Return the value of a VARCHAR INOUT or OUT parameter

The VARIABLE_VALUE_VARCHAR procedure provides the capability to return the value of a VARCHAR INOUT or OUT parameter.

Syntax



Parameters

c An input argument of type INTEGER that specifies the cursor ID of the cursor returning a bind variable.

name
An input argument that specifies the name of the bind variable.

value
An output argument of type VARCHAR(32672) that specifies the variable receiving the value.

Authorization

EXECUTE privilege on the DBMS_SQL module.

DBMS_UTILITY module

The DBMS_UTILITY module provides various utility programs.

The schema for this module is SYSIBMADM.

The DBMS_UTILITY module includes the following routines.

Table 18. Built-in routines available in the DBMS_UTILITY module

Routine Name	Description
ANALYZE_DATABASE procedure	Analyze database tables, clusters, and indexes.
ANALYZE_PART_OBJECT procedure	Analyze a partitioned table or partitioned index.
ANALYZE_SCHEMA procedure	Analyze schema tables, clusters, and indexes.
CANONICALIZE procedure	Canonicalizes a string (for example, strips off white space).
COMMA_TO_TABLE procedure	Convert a comma-delimited list of names to a table of names.
COMPILE_SCHEMA procedure	Compile programs in a schema.
DB_VERSION procedure	Get the database version.
EXEC_DDL_STATEMENT procedure	Execute a DDL statement.
FORMAT_CALL_STACK function	Get a description of the current call stack.
FORMAT_ERROR_BACKTRACE function	Get a description of the call stack that existed at the time of the most recent error within a compiled SQL routine.
GET_CPU_TIME function	Get the current CPU time.
GET_DEPENDENCY procedure	Get objects that depend on the given object.
GET_HASH_VALUE function	Compute a hash value.
GET_TIME function	Get the current time.
NAME_RESOLVE procedure	Resolve the given name.
NAME_TOKENIZE procedure	Parse the given name into its component parts.
TABLE_TO_COMMA procedure	Convert a table of names to a comma-delimited list.

Table 18. Built-in routines available in the DBMS_UTILITY module (continued)

Routine Name	Description
VALIDATE procedure	Make an invalid database object valid.

The following table lists the built-in variables and types available in the DBMS_UTILITY module.

Table 19. DBMS_UTILITY public variables

Public variables	Data type	Description
lname_array	TABLE	For lists of long names.
uncl_array	TABLE	For lists of users and names.

The LNAME_ARRAY is for storing lists of long names including fully-qualified names.

```
ALTER MODULE SYSIBMADM.DBMS_UTILITY PUBLISH TYPE LNAME_ARRAY AS VARCHAR(4000) ARRAY[];
```

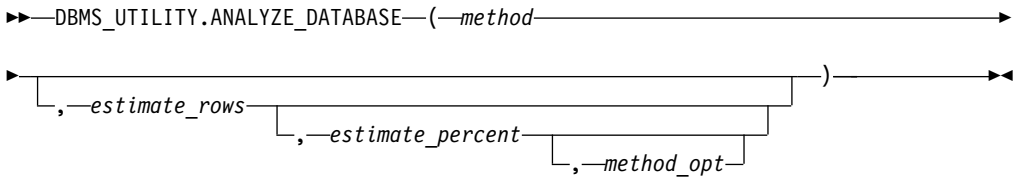
The UNCL_ARRAY is for storing lists of users and names.

```
ALTER MODULE SYSIBMADM.DBMS_UTILITY PUBLISH TYPE UNCL_ARRAY  AS VARCHAR(227)  ARRAY[];
```

ANALYZE_DATABASE procedure - Gather statistics on tables, clusters, and indexes

The ANALYZE_DATABASE procedure provides the capability to gather statistics on tables, clusters, and indexes in the database.

Syntax



Parameters

method

An input argument of type VARCHAR(128) that specifies the type of analyze functionality to perform. Valid values are:

- ESTIMATE - gather estimated statistics based upon on either a specified number of rows in *estimate_rows* or a percentage of rows in *estimate_percent*;
- COMPUTE - compute exact statistics; or
- DELETE - delete statistics from the data dictionary.

estimate_rows

An optional input argument of type INTEGER that specifies the number of rows on which to base estimated statistics. One of *estimate_rows* or *estimate_percent* must be specified if the *method* is ESTIMATE. The default value is NULL.

estimate_percent

An optional input argument of type INTEGER that specifies the percentage of

rows upon which to base estimated statistics. One of *estimate_rows* or *estimate_percent* must be specified if the *method* is ESTIMATE. The default value is NULL.

method_opt
An optional input argument of type VARCHAR(1024) that specifies the object types to be analyzed. Any combination of the following keywords are valid:

- [FOR TABLE]
- [FOR ALL [INDEXED] COLUMNS] [SIZE n]
- [FOR ALL INDEXES]

The default is NULL.

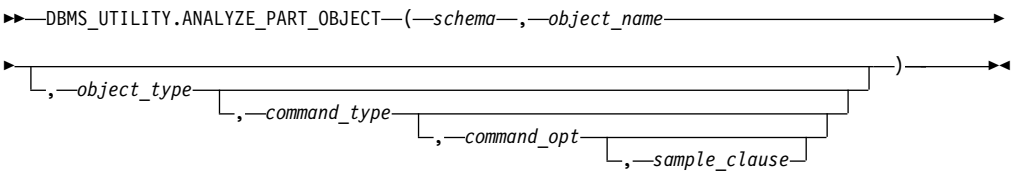
Authorization

EXECUTE privilege on the DBMS_UTILITY module.

ANALYZE_PART_OBJECT procedure - Gather statistics on a partitioned table or partitioned index

The ANALYZE_PART_OBJECT procedure provides the capability to gather statistics on a partitioned table or index.

Syntax



Parameters

schema
An input argument of type VARCHAR(128) that specifies the schema name of the schema whose objects are to be analyzed.

object_name
An input argument of type VARCHAR(128) that specifies the name of the partitioned object to be analyzed.

object_type
An optional input argument of type CHAR that specifies the type of object to be analyzed. Valid values are:

- T - table;
- I - index.

The default is T.

command_type
An optional input argument of type CHAR that specifies the type of analyze functionality to perform. Valid values are:

- E - gather estimated statistics based upon on a specified number of rows or a percentage of rows in the *sample_clause* clause;
- C - compute exact statistics; or
- V - validate the structure and integrity of the partitions.

The default value is E.

command_opt
An optional input argument of type VARCHAR(1024) that specifies the options for the statistics calculation. For *command_type* E or C, this argument can be any combination of:

- [FOR TABLE]
- [FOR ALL COLUMNS]
- [FOR ALL LOCAL INDEXES]

For *command_type* V, this argument can be CASCADE if *object_type* is T. The default value is NULL.

sample_clause
An optional input argument of type VARCHAR(128). If *command_type* is E, this argument contains the following clause to specify the number of rows or percentage of rows on which to base the estimate.

```
SAMPLE n { ROWS | PERCENT }
```

The default value is `SAMPLE 5 PERCENT`.

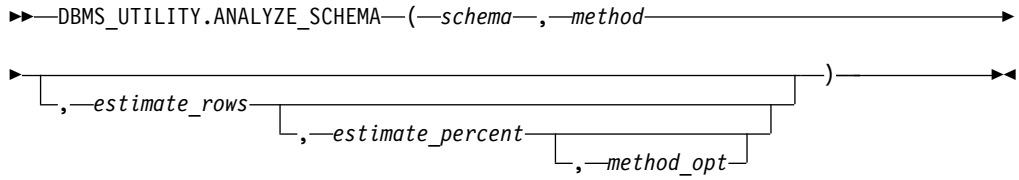
Authorization

EXECUTE privilege on the DBMS_UTILITY module.

ANALYZE_SCHEMA procedure - Gather statistics on schema tables, clusters, and indexes

The ANALYZE_SCHEMA procedure provides the capability to gather statistics on tables, clusters, and indexes in the specified schema.

Syntax



Parameters

schema
An input argument of type VARCHAR(128) that specifies the schema name of the schema whose objects are to be analyzed.

method
An input argument of type VARCHAR(128) that specifies the type of analyze functionality to perform. Valid values are:

- ESTIMATE - gather estimated statistics based upon on either a specified number of rows in *estimate_rows* or a percentage of rows in *estimate_percent*;
- COMPUTE - compute exact statistics; or
- DELETE - delete statistics from the data dictionary.

estimate_rows
An optional input argument of type INTEGER that specifies the number of

rows on which to base estimated statistics. One of *estimate_rows* or *estimate_percent* must be specified if the *method* is ESTIMATE. The default value is NULL.

estimate_percent
An optional input argument of type INTEGER that specifies the percentage of rows upon which to base estimated statistics. One of *estimate_rows* or *estimate_percent* must be specified if the *method* is ESTIMATE. The default value is NULL.

method_opt
An optional input argument of type VARCHAR(1024) that specifies the object types to be analyzed. Any combination of the following keywords are valid:

- [FOR TABLE]
- [FOR ALL [INDEXED] COLUMNS] [SIZE n]
- [FOR ALL INDEXES]

The default is NULL.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

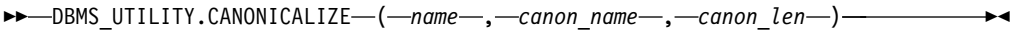
CANONICALIZE procedure - Canonicalize a string

The CANONICALIZE procedure performs various operations on an input string.

The CANONICALIZE procedure performs the following operations on an input string:

- If the string is not double-quoted, verifies that it uses the characters of a legal identifier. If not, an exception is thrown. If the string is double-quoted, all characters are allowed.
- If the string is not double-quoted and does not contain periods, puts all alphabetic characters into uppercase and eliminates leading and trailing spaces.
- If the string is double-quoted and does not contain periods, strips off the double quotes.
- If the string contains periods and no portion of the string is double-quoted, puts each portion of the string into uppercase and encloses each portion in double quotes.
- If the string contains periods and portions of the string are double-quoted, returns the double-quoted portions unchanged, including the double quotes, and returns the non-double-quoted portions in uppercase and enclosed in double quotes.

Syntax



Parameters

name
An input argument of type VARCHAR(1024) that specifies the string to be canonicalized.

canon_name
An output argument of type VARCHAR(1024) that returns the canonicalized string.

canon_len
An input argument of type INTEGER that specifies the number of bytes in *name* to canonicalize starting from the first character.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following procedure applies the CANONICALIZE procedure on its input parameter and displays the results.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE canonicalize(
  IN p_name VARCHAR(4096),
  IN p_length INTEGER DEFAULT 30)
BEGIN
  DECLARE v_canon VARCHAR(100);

  CALL DBMS_UTILITY.CANONICALIZE(p_name, v_canon, p_length);
  CALL DBMS_OUTPUT.PUT_LINE('Canonicalized name ==>' || v_canon || '<==');
  CALL DBMS_OUTPUT.PUT_LINE('Length: ' || LENGTH(v_canon));
END@

CALL canonicalize('Identifier')@
CALL canonicalize('"Identifier"')@
CALL canonicalize('"_+142%"')@
CALL canonicalize('abc.def.ghi')@
CALL canonicalize('"abc.def.ghi"')@
CALL canonicalize('"abc".def."ghi"')@
CALL canonicalize('"abc.def".ghi')@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE OR REPLACE PROCEDURE canonicalize(
  IN p_name VARCHAR(4096),
  IN p_length INTEGER DEFAULT 30)
BEGIN
  DECLARE v_canon VARCHAR(100);

  CALL DBMS_UTILITY.CANONICALIZE(p_name, v_canon, p_length);
  CALL DBMS_OUTPUT.PUT_LINE('Canonicalized name ==>' || v_canon || '<==');
  CALL DBMS_OUTPUT.PUT_LINE('Length: ' || LENGTH(v_canon));
END
DB20000I  The SQL command completed successfully.

CALL canonicalize('Identifier')

  Return Status = 0

Canonicalized name ==>IDENTIFIER<==
Length: 10

CALL canonicalize('"Identifier"')

  Return Status = 0
```

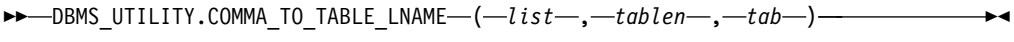
```
Canonicalized name ==>Identifier<==
Length: 10

CALL canonicalize('"_+142%"')

  Return Status = 0

Canonicalized name ==>_+142%<==
Length: 6

CALL canonicalize('abc.def.ghi')

  Return Status = 0

Canonicalized name ==>"ABC"."DEF"."GHI"<==
Length: 17

CALL canonicalize('"abc.def.ghi"')

  Return Status = 0

Canonicalized name ==>abc.def.ghi<==
Length: 11

CALL canonicalize('"abc".def."ghi"')

  Return Status = 0

Canonicalized name ==>"abc"."DEF"."ghi"<==
Length: 17

CALL canonicalize('"abc.def".ghi')

  Return Status = 0

Canonicalized name ==>"abc.def"."GHI"<==
Length: 15
```

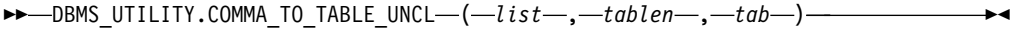
COMMA_TO_TABLE procedures - Convert a comma-delimited list of names into a table of names

The COMMA_TO_TABLE procedure converts a comma-delimited list of names into an array of names. Each entry in the list becomes an element in the array.

Note: The names must be formatted as valid identifiers.

Syntax





Parameters

list
An input argument of type VARCHAR(32672) that specifies a comma-delimited list of names.

tablen
An output argument of type INTEGER that specifies the number of entries in *tab*.

tab
An output argument of type LNAME_ARRAY or UNCL_ARRAY that contains a table of the individual names in *list*. See LNAME_ARRAY or UNCL_ARRAY for a description of *tab*.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following procedure uses the COMMA_TO_TABLE_LNAME procedure to convert a list of names to a table. The table entries are then displayed.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE comma_to_table(
  IN p_list VARCHAR(4096))
BEGIN
  DECLARE r_lname DBMS_UTILITY.LNAME_ARRAY;
  DECLARE v_length INTEGER;
  CALL DBMS_UTILITY.COMMA_TO_TABLE_LNAME(p_list, v_length, r_lname);
  BEGIN
    DECLARE i INTEGER DEFAULT 1;
    DECLARE loop_limit INTEGER;

    SET loop_limit = v_length;
    WHILE i <= loop_limit DO
      CALL DBMS_OUTPUT.PUT_LINE(r_lname[i]);
      SET i = i + 1;
    END WHILE;
  END;
END@

CALL comma_to_table('sample_schema.dept,sample_schema.emp,sample_schema.jobhist')@
```

This example results in the following output:

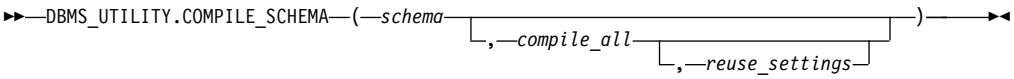
```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE OR REPLACE PROCEDURE comma_to_table(
  IN p_list VARCHAR(4096))
BEGIN
  DECLARE r_lname DBMS_UTILITY.LNAME_ARRAY;
  DECLARE v_length INTEGER;
  CALL DBMS_UTILITY.COMMA_TO_TABLE_LNAME(p_list, v_length, r_lname);
  BEGIN
    DECLARE i INTEGER DEFAULT 1;
    DECLARE loop_limit INTEGER;

    SET loop_limit = v_length;
    WHILE i <= loop_limit DO
      CALL DBMS_OUTPUT.PUT_LINE(r_lname[i]);
      SET i = i + 1;
    END WHILE;
  END;
END
DB20000I  The SQL command completed successfully.
```

```
CALL comma_to_table('sample_schema.dept,sample_schema.emp,sample_schema.jobhist')

  Return Status = 0

sample_schema.dept
sample_schema.emp
sample_schema.jobhist
```

COMPILE_SCHEMA procedure - Compile all functions, procedures, triggers, and packages in a schema

The COMPILE_SCHEMA procedure provides the capability to recompile all functions, procedures, triggers, and packages in a schema.

Syntax



Parameters

schema
An input argument of type VARCHAR(128) that specifies the schema in which the programs are to be recompiled.

compile_all
An optional input argument of type BOOLEAN that must be set to `false`, meaning that the procedure only recompiles programs currently in invalid state.

reuse_settings
An optional input argument of type BOOLEAN that must be set to `false`, meaning that the procedure uses the current session settings.

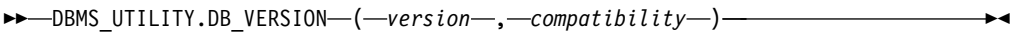
Authorization

EXECUTE privilege on the DBMS_UTILITY module.

DB_VERSION procedure - Retrieve the database version

The DB_VERSION procedure returns the version number of the database.

Syntax



Parameters

version
An output argument of type VARCHAR(1024) that returns the database version number.

compatibility
An output argument of type VARCHAR(1024) that returns the compatibility setting of the database.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following anonymous block displays the database version information.

```
SET SERVEROUTPUT ON@

BEGIN
  DECLARE v_version VARCHAR(80);
  DECLARE v_compat VARCHAR(80);

  CALL DBMS_UTILITY.DB_VERSION(v_version, v_compat);
  CALL DBMS_OUTPUT.PUT_LINE('Version: ' || v_version);
  CALL DBMS_OUTPUT.PUT_LINE('Compatibility: ' || v_compat);
END@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

BEGIN
  DECLARE v_version VARCHAR(80);
  DECLARE v_compat VARCHAR(80);

  CALL DBMS_UTILITY.DB_VERSION(v_version, v_compat);
  CALL DBMS_OUTPUT.PUT_LINE('Version: ' || v_version);
  CALL DBMS_OUTPUT.PUT_LINE('Compatibility: ' || v_compat);
END
DB20000I  The SQL command completed successfully.

Version: Db2 v9.7.0.0
Compatibility: Db2 v9.7.0.0
```

EXEC_DDL_STATEMENT procedure - Run a DDL statement

The EXEC_DDL_STATEMENT procedure provides the capability to execute a DDL command.

Syntax

Parameters

parse_string
An input argument of type VARCHAR(1024) that specifies the DDL command to execute.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

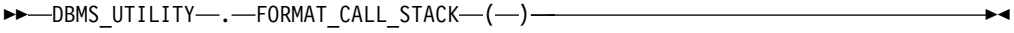
Examples

Example 1: The following anonymous block creates the job table.

```
BEGIN
  CALL DBMS_UTILITY.EXEC_DDL_STATEMENT(
    'CREATE TABLE job (' ||
    'jobno DECIMAL(3),' ||
    'jname VARCHAR(9))' );
END@
```

FORMAT_CALL_STACK

The FORMAT_CALL_STACK function returns a returns a string of type VARCHAR(32672) that reflects the state of the call stack in the current session at the time that FORMAT_CALL_STACK is invoked.



Authorization

Each line of the result describes an active routine on the call stack. The most recently invoked routine appears as the first line, followed by less recently invoked routines. So, the result describes the current call chain, beginning with the most deeply nested routine, followed by its caller, the caller's caller, and so on.

The invocation of the function itself is not included in the reported call stack.

If no other routines are active in the current session at the time **FORMAT_CALL_STACK()** function is invoked, the function returns NULL.

Each line of the call stack begins with the line number, relative to the first line of the routine, which is line 1. If line number information is unavailable for that routine, the line is displayed as a hyphen ("-").

After the line number, the routine type is displayed as a procedure, function, trigger, or anonymous block (compound statement). External routines are also displayed. However, no distinction is made between external procedures or external functions. Line number information is unavailable for external routines.

The routine type is followed by the fully-qualified routine name: Routine schema, module name (if applicable), routine name, and then the subroutine name (if applicable). The specific name of the routine is reported after its fully-qualified name. The schema of the specific name is not reported, and is assumed to be the same as in the fully-qualified routine name.

EXECUTE privilege on the DBMS_UTILITY module.

Example

In this example, **FORMAT_CALL_STACK()** function is used in a condition handler to record an error in a logging table.

```
create global temporary table error_log(ts timestamp, message varchar(4096))
on commit preserve rows
not logged on rollback preserve rows
in error_ts %

-- ...

create procedure C(in N integer, in D integer)
language SQL
begin
```

```
declare X double;

declare continue handler for sqlexception
begin
declare message varchar(255);
declare NL char(1) default x'0a' ;

get diagnostics exception 1 message = message_text;

insert into error_log values(
current timestamp,
'N = ' || N || '; D = ' || D || NL ||
message || NL || NL ||
'Line Routine' || NL ||
'-------- ----------------' || NL ||
dbms_utility.format_call_stack());
end;

set X = cast(N as double) / cast(D as double);
end %
```

If routine C is called with D=0, we obtain a record in the ERROR_LOG table as shown below:

```
select * from error_log order by ts asc

TS MESSAGE
-------------------------- ------------------------------------------//--
2015-01-27-14.34.30.972622 N = 10; D = 0
SQL0801N Division by zero was attempted. SQLSTATE=22012

Line Routine
-------- ----------------
13 procedure MYSCHEMA.C (specific SQL150217153125821)
7 procedure MYSCHEMA.B (specific SQL150217153126322)
4 procedure MYSCHEMA.A (specific SQL150217153126423)

1 record(s) selected.
```

Here procedure C was called from line 7 of some procedure MYSCHEMA.B, which in turn was called from line 4 of procedure MYSCHEMA.A.

FORMAT_ERROR_BACKTRACE

The FORMAT_ERROR_BACKTRACE function returns a string of type VARCHAR(32672) that reflects the state of the call stack at the time of the most recent error to occur in an SQL routine during the current session.

Authorization

Each line of the result describes an active routine on the call stack at the time of the error; the most recently invoked routine appears as the first line, followed by less recently invoked routines. In other words, the result describes the call chain at the time of the error, beginning with the most deeply nested routine, followed by its caller, the caller's caller, and so on.

If no SQL routine has encountered an error during the current session, the function returns NULL.

Each line of the call stack begins with the line number, relative to the first line of the routine, which is line 1. If line number information is unavailable for that routine, the line is displayed as a hyphen ("-").

After the line number, the routine type is displayed as a procedure, function, trigger, or anonymous block (compound statement). External routines are also displayed; however, no distinction is made between external procedures or external functions. Line number information is unavailable for external routines.

The routine type is followed by the fully qualified routine name, routine schema, module name (if applicable), routine name, and then the subroutine name (if applicable). The specific name of the routine is reported after its fully-qualified name. The schema of the specific name is not reported, and is assumed to be the same as in the fully-qualified routine name.

EXECUTE privilege on the DBMS_UTILITY module.

Example

The following nested call scenario returns an error.

```
create table T1(C1 integer, C2 integer, tag varchar(32)) @
insert into T1 values (1, 6, 'VI'), (2, 10, 'X'), (3, 11, 'XI'), (4, 48, 'XLVIII') @

create or replace procedure B(
in colname varchar(128),
in value integer,
in tag varchar(32))
language SQL
begin
declare stmt_text varchar(256);
declare S1 statement;

set stmt_text = 'update T1 set tag = ? where ' || colname || ' = ?';
prepare S1 from stmt_text;
execute S1 using tag, value;
end @

create or replace procedure A()
begin
call B('C1', 1, 'six');
call B('C2', 11, 'eleven');
call B('C3', 0, 'zero'); -- will produce an error
call B('C2', 48, 'forty-eight');
end @

begin
call A;
end @
DB21034E The command was processed as an SQL statement because it was not a valid
Command Line Processor command. During SQL processing it returned:
SQL0206N "C3" is not valid in the context where it is used. SQLSTATE=42703
```

After the error occurs, FORMAT_ERROR_BACKTRACE returns information similar to the following:

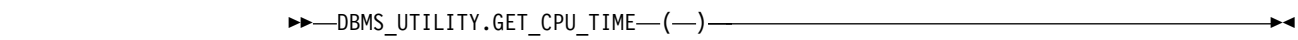
```
values dbms_utility.format_error_backtrace()

1
--------------------------------------------------------------------------//--
11 procedure MYSCHEMA.B (specific SQL150217153622825); SQLCODE=-206
```

```
5 procedure MYSCHEMA.A (specific SQL150217153623026)
2 anonymous block (specific SQL150217153624330)

1 record(s) selected.
```

GET_CPU_TIME function - Retrieve the current CPU time

The GET_CPU_TIME function returns the CPU time in hundredths of a second from some arbitrary point in time.

Syntax



Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following SELECT command retrieves the current CPU time.

```
SELECT DBMS_UTILITY.GET_CPU_TIME FROM DUAL;

get_cpu_time
------------
         603
```

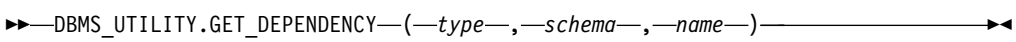
Example 2: Calculate the elapsed time by obtaining difference between two CPU time values.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()

  BEGIN
     DECLARE cpuTime1 BIGINT;
     DECLARE cpuTime2 BIGINT;
     DECLARE cpuTimeDelta BIGINT;
     DECLARE i INTEGER;

     SET cpuTime1 = DBMS_UTILITY.GET_CPU_TIME();

     SET i = 0;
     loop1: LOOP
        IF i > 10000 THEN
           LEAVE loop1;
        END IF;
     SET i = i + 1;
     END LOOP;

     SET cpuTime2 = DBMS_UTILITY.GET_CPU_TIME();

     SET cpuTimeDelta = cpuTime2 - cpuTime1;

     CALL DBMS_OUTPUT.PUT_LINE( 'cpuTimeDelta = ' || cpuTimeDelta );
     END
     @

  CALL proc1@
```

GET_DEPENDENCY procedure - List objects dependent on the given object

The GET_DEPENDENCY procedure provides the capability to list all objects that are dependent upon the given object.

Syntax



Parameters

type
An input argument of type VARCHAR(128) that specifies the object type of *name*. Valid values are FUNCTION, INDEX, LOB, PACKAGE, PACKAGE BODY, PROCEDURE, SEQUENCE, TABLE, TRIGGER, and VIEW.

schema
An input argument of type VARCHAR(128) that specifies the name of the schema in which *name* exists.

name
An input argument of type VARCHAR(128) that specifies the name of the object for which dependencies are to be obtained.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following anonymous block finds dependencies on the table T1, and the function FUNC1.

```
SET SERVEROUTPUT ON@

CREATE TABLE SCHEMA1.T1 (C1 INTEGER)@

CREATE OR REPLACE FUNCTION SCHEMA2.FUNC1( parm1 INTEGER )
SPECIFIC FUNC1
RETURNS INTEGER
BEGIN
  RETURN parm1;
END@

CREATE OR REPLACE FUNCTION SCHEMA3.FUNC2()
SPECIFIC FUNC2
RETURNS INTEGER
BEGIN
  DECLARE retVal INTEGER;
  SELECT SCHEMA2.FUNC1(1) INTO retVal FROM SCHEMA1.T1;
END@

CALL DBMS_UTILITY.GET_DEPENDENCY('FUNCTION', 'SCHEMA2', 'FUNC1')@
CALL DBMS_UTILITY.GET_DEPENDENCY('TABLE', 'SCHEMA1', 'T1')@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE TABLE SCHEMA1.T1 (C1 INTEGER)
```

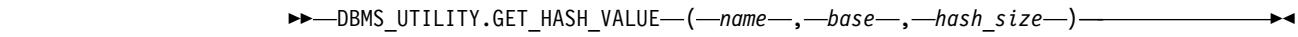
```
DB20000I  The SQL command completed successfully.

CREATE OR REPLACE FUNCTION SCHEMA2.FUNC1( parm1 INTEGER )
SPECIFIC FUNC1
RETURNS INTEGER
BEGIN
  RETURN parm1;
END
DB20000I  The SQL command completed successfully.

CREATE OR REPLACE FUNCTION SCHEMA3.FUNC2()
SPECIFIC FUNC2
RETURNS INTEGER
BEGIN
  DECLARE retVal INTEGER;
  SELECT SCHEMA2.FUNC1(1) INTO retVal FROM SCHEMA1.T1;
END
DB20000I  The SQL command completed successfully.

CALL DBMS_UTILITY.GET_DEPENDENCY('FUNCTION', 'SCHEMA2', 'FUNC1')

  Return Status = 0

DEPENDENCIES ON SCHEMA2.FUNC1
------------------------------------------------------------------
*FUNCTION SCHEMA2.FUNC1()
*   FUNCTION SCHEMA3 .FUNC2()

CALL DBMS_UTILITY.GET_DEPENDENCY('TABLE', 'SCHEMA1', 'T1')

  Return Status = 0

DEPENDENCIES ON SCHEMA1.T1
------------------------------------------------------------------
*TABLE SCHEMA1.T1()
*   FUNCTION SCHEMA3 .FUNC2()
```

GET_HASH_VALUE function - Compute a hash value for a given string

The GET_HASH_VALUE function provides the capability to compute a hash value for a given string.

The function returns a generated hash value of type INTEGER, and the value is platform-dependent.

Syntax



Parameters

name
An input argument of type VARCHAR(32672) that specifies the string for which a hash value is to be computed.

base
An input argument of type INTEGER that specifies the starting value at which hash values are to be generated.

hash_size
An input argument of type INTEGER that specifies the number of hash values for the desired hash table.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following example returns hash values for two strings. The starting value for the hash values is 100, with a maximum of 1024 distinct values.

```
SELECT DBMS_UTILITY.GET_HASH_VALUE('Peter',100,1024) AS HASH_VALUE
     FROM SYSIBM.SYSDUMMY1@
SELECT DBMS_UTILITY.GET_HASH_VALUE('Mary',100,1024) AS HASH_VALUE
     FROM SYSIBM.SYSDUMMY1@
```

This example results in the following output:

```
SELECT DBMS_UTILITY.GET_HASH_VALUE('Peter',100,1024) AS HASH_VALUE
     FROM SYSIBM.SYSDUMMY1

HASH_VALUE
--------------------
                 343

  1 record(s) selected.


SELECT DBMS_UTILITY.GET_HASH_VALUE('Mary',100,1024) AS HASH_VALUE
     FROM SYSIBM.SYSDUMMY1

HASH_VALUE
--------------------
                 760

  1 record(s) selected.
```

GET_TIME function - Return the current time

The GET_TIME function provides the capability to return the current time in hundredths of a second.

Syntax

The value represents the number of hundredths of second since 1970-01-01-00:00:00.00000000000.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following example shows calls to the GET_TIME function.

```
SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

 get_time
```

```
---------
  1555860

SELECT DBMS_UTILITY.GET_TIME FROM DUAL;

 get_time
---------
  1556037
```

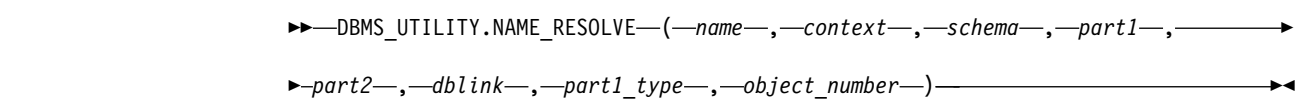
Example 2: The following example converts the value returned by the GET_TIME function into hours, minutes, and seconds (with no adjustment for timezone).

```
VALUES TIMESTAMP('1-1-1970') + (DBMS_UTILITY.GET_TIME() / 100) SECONDS

1
--------------------------
2012-11-22-19.23.14.000000
```

NAME_RESOLVE procedure - Obtain the schema and other membership information for a database object

The NAME_RESOLVE procedure provides the capability to obtain the schema and other membership information of a database object. Synonyms are resolved to their base objects.

Syntax



Parameters

name
An input argument of type VARCHAR(1024) that specifies the name of the database object to resolve. Can be specified in the format:

```
[[ a.]b.]c[@dblink ]
```

context
An input argument of type INTEGER. Set to the following values:

- 1 - to resolve a function, procedure, or module name;
- 2 - to resolve a table, view, sequence, or synonym name; or
- 3 - to resolve a trigger name.

schema
An output argument of type VARCHAR(128) that specifies the name of the schema containing the object specified by *name*.

part1
An output argument of type VARCHAR(128) that specifies the name of the resolved table, view, sequence, trigger, or module.

part2
An output argument of type VARCHAR(128) that specifies the name of the resolved function or procedure (including functions and procedures within a module).

dblink
An output argument of type VARCHAR(128) that specifies name of the database link (if `@dblink` is specified in *name*).

part1_type
An output argument of type INTEGER. Returns the following values:

- 2 - resolved object is a table;
- 4 - resolved object is a view;
- 6 - resolved object is a sequence;
- 7 - resolved object is a stored procedure;
- 8 - resolved object is a stored function;
- 9 - resolved object is a module or a function or procedure within a module; or
- 12 - resolved object is a trigger.

object_number
An output argument of type INTEGER that specifies the object identifier of the resolved database object.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following stored procedure is used to display the returned values of the NAME_RESOLVE procedure for various database objects.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE name_resolve(
  IN p_name VARCHAR(4096),
  IN p_context DECFLOAT )
BEGIN
  DECLARE v_schema VARCHAR(30);
  DECLARE v_part1 VARCHAR(30);
  DECLARE v_part2 VARCHAR(30);
  DECLARE v_dblink VARCHAR(30);
  DECLARE v_part1_type DECFLOAT;
  DECLARE v_objectid DECFLOAT;

  CALL DBMS_UTILITY.NAME_RESOLVE(p_name, p_context, v_schema, v_part1, v_part2,
    v_dblink, v_part1_type, v_objectid);
  CALL DBMS_OUTPUT.PUT_LINE('name      : ' || p_name);
  CALL DBMS_OUTPUT.PUT_LINE('context   : ' || p_context);
  CALL DBMS_OUTPUT.PUT_LINE('schema    : ' || v_schema);
  IF v_part1 IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('part1     : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('part1     : ' || v_part1);
  END IF;
  IF v_part2 IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('part2     : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('part2     : ' || v_part2);
  END IF;
  IF v_dblink IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('dblink    : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('dblink    : ' || v_dblink);
  END IF;
  CALL DBMS_OUTPUT.PUT_LINE('part1 type: ' || v_part1_type);
  CALL DBMS_OUTPUT.PUT_LINE('object id : ' || v_objectid);
END@

DROP TABLE S1.T1@
```

```
CREATE TABLE S1.T1 (C1 INT)@

CREATE OR REPLACE PROCEDURE S2.PROC1
BEGIN
END@

CREATE OR REPLACE MODULE S3.M1@
ALTER MODULE S3.M1 PUBLISH FUNCTION F1() RETURNS BOOLEAN
BEGIN
  RETURN TRUE;
END@

CALL NAME_RESOLVE( 'S1.T1', 2 )@
CALL NAME_RESOLVE( 'S2.PROC1', 2 )@
CALL NAME_RESOLVE( 'S2.PROC1', 1 )@
CALL NAME_RESOLVE( 'PROC1', 1 )@
CALL NAME_RESOLVE( 'M1', 1 )@
CALL NAME_RESOLVE( 'S3.M1.F1', 1 )@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE OR REPLACE PROCEDURE name_resolve(
  IN p_name VARCHAR(4096),
  IN p_context DECFLOAT )
BEGIN
  DECLARE v_schema VARCHAR(30);
  DECLARE v_part1 VARCHAR(30);
  DECLARE v_part2 VARCHAR(30);
  DECLARE v_dblink VARCHAR(30);
  DECLARE v_part1_type DECFLOAT;
  DECLARE v_objectid DECFLOAT;

  CALL DBMS_UTILITY.NAME_RESOLVE(p_name, p_context, v_schema, v_part1, v_part2,
    v_dblink, v_part1_type, v_objectid);
  CALL DBMS_OUTPUT.PUT_LINE('name      : ' || p_name);
  CALL DBMS_OUTPUT.PUT_LINE('context   : ' || p_context);
  CALL DBMS_OUTPUT.PUT_LINE('schema    : ' || v_schema);
  IF v_part1 IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('part1     : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('part1     : ' || v_part1);
  END IF;
  IF v_part2 IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('part2     : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('part2     : ' || v_part2);
  END IF;
  IF v_dblink IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('dblink    : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('dblink    : ' || v_dblink);
  END IF;
  CALL DBMS_OUTPUT.PUT_LINE('part1 type: ' || v_part1_type);
  CALL DBMS_OUTPUT.PUT_LINE('object id : ' || v_objectid);
END
DB20000I  The SQL command completed successfully.

DROP TABLE S1.T1
DB20000I  The SQL command completed successfully.

CREATE TABLE S1.T1 (C1 INT)
DB20000I  The SQL command completed successfully.

CREATE OR REPLACE PROCEDURE S2.PROC1
```

```
BEGIN
END
DB20000I  The SQL command completed successfully.

CREATE OR REPLACE MODULE S3.M1
DB20000I  The SQL command completed successfully.

ALTER MODULE S3.M1 PUBLISH FUNCTION F1() RETURNS BOOLEAN
BEGIN
  RETURN TRUE;
END
DB20000I  The SQL command completed successfully.

CALL NAME_RESOLVE( 'S1.T1', 2 )

  Return Status = 0

name      : S1.T1
context   : 2
schema    : S1
part1     : T1
part2     : NULL
dblink    : NULL
part1 type: 2
object id : 8

CALL NAME_RESOLVE( 'S2.PROC1', 2 )
SQL0204N  "S2.PROC1" is an undefined name.  SQLSTATE=42704

CALL NAME_RESOLVE( 'S2.PROC1', 1 )

  Return Status = 0

name      : S2.PROC1
context   : 1
schema    : S2
part1     : PROC1
part2     : NULL
dblink    : NULL
part1 type: 7
object id : 66611

CALL NAME_RESOLVE( 'PROC1', 1 )

  Return Status = 0

name      : PROC1
context   : 1
schema    : S2
part1     : NULL
part2     : PROC1
dblink    : NULL
part1 type: 7
object id : 66611

CALL NAME_RESOLVE( 'M1', 1 )

  Return Status = 0

name      : M1
context   : 1
schema    : S3
part1     : NULL
part2     : M1
dblink    : NULL
part1 type: 9
object id : 16
```

```
CALL NAME_RESOLVE( 'S3.M1.F1', 1 )

  Return Status = 0

name      : S3.M1.F1
context   : 1
schema    : S3
part1     : M1
part2     : F1
dblink    : NULL
part1 type: 9
object id : 16
```

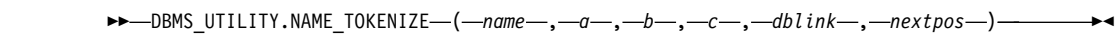
Example 2: Resolve a table accessed by a database link. Note that NAME_RESOLVE does not check the validity of the database object on the remote database. It merely echoes back the components specified in the *name* argument.

```
BEGIN
    name_resolve('sample_schema.emp@sample_schema_link',2);
END;

name      : sample_schema.emp@sample_schema_link
context   : 2
schema    : SAMPLE_SCHEMA
part1     : EMP
part2     :
dblink    : SAMPLE_SCHEMA_LINK
part1 type: 0
object id : 0
```

NAME_TOKENIZE procedure - Parse the given name into its component parts

The NAME_TOKENIZE procedure parses a name into its component parts. Names without double quotes are put into uppercase, and double quotes are stripped from names with double quotes.

Syntax



Parameters

name

An input argument of type VARCHAR(1024) that specifies the string containing a name in the following format:

```
a[.b[.c]][@dblink ]
```

a

An output argument of type VARCHAR(128) that returns the leftmost component.

b

An output argument of type VARCHAR(128) that returns the second component, if any.

c

An output argument of type VARCHAR(128) that returns the third component, if any.

dblink

An output argument of type VARCHAR(32672) that returns the database link name.

nextpos
An output argument of type INTEGER that specifies the position of the last character parsed in *name*.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following stored procedure is used to display the returned values of the NAME_TOKENIZE procedure for various names.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE name_tokenize(
  IN p_name VARCHAR(100) )
BEGIN
  DECLARE v_a VARCHAR(30);
  DECLARE v_b VARCHAR(30);
  DECLARE v_c VARCHAR(30);
  DECLARE v_dblink VARCHAR(30);
  DECLARE v_nextpos INTEGER;

  CALL DBMS_UTILITY.NAME_TOKENIZE(p_name, v_a, v_b, v_c, v_dblink, v_nextpos);
  CALL DBMS_OUTPUT.PUT_LINE('name   : ' || p_name);
  IF v_a IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('a      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('a      : ' || v_a);
  END IF;
  IF v_b IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('b      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('b      : ' || v_b);
  END IF;
  IF v_c IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('c      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('c      : ' || v_c);
  END IF;
  IF v_dblink IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('dblink : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('dblink : ' || v_dblink);
  END IF;
  IF v_nextpos IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('nextpos: NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('nextpos: ' || v_nextpos);
  END IF;
END@

CALL name_tokenize( 'b' )@
CALL name_tokenize( 'a.b' )@
CALL name_tokenize( '"a".b.c' )@
CALL name_tokenize( 'a.b.c@d' )@
CALL name_tokenize( 'a.b."c"@"d"' )@
```

This example results in the following output:

```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE OR REPLACE PROCEDURE name_tokenize(
  IN p_name VARCHAR(100) )
```

```
BEGIN
  DECLARE v_a VARCHAR(30);
  DECLARE v_b VARCHAR(30);
  DECLARE v_c VARCHAR(30);
  DECLARE v_dblink VARCHAR(30);
  DECLARE v_nextpos INTEGER;

  CALL DBMS_UTILITY.NAME_TOKENIZE(p_name, v_a, v_b, v_c, v_dblink, v_nextpos);
  CALL DBMS_OUTPUT.PUT_LINE('name   : ' || p_name);
  IF v_a IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('a      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('a      : ' || v_a);
  END IF;
  IF v_b IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('b      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('b      : ' || v_b);
  END IF;
  IF v_c IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('c      : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('c      : ' || v_c);
  END IF;
  IF v_dblink IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('dblink : NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('dblink : ' || v_dblink);
  END IF;
  IF v_nextpos IS NULL THEN
    CALL DBMS_OUTPUT.PUT_LINE('nextpos: NULL');
  ELSE
    CALL DBMS_OUTPUT.PUT_LINE('nextpos: ' || v_nextpos);
  END IF;
END
DB20000I  The SQL command completed successfully.

CALL name_tokenize( 'b' )

  Return Status = 0

name   : b
a      : B
b      : NULL
c      : NULL
dblink : NULL
nextpos: 1

CALL name_tokenize( 'a.b' )

  Return Status = 0

name   : a.b
a      : A
b      : B
c      : NULL
dblink : NULL
nextpos: 3

CALL name_tokenize( '"a".b.c' )

  Return Status = 0

name   : "a".b.c
a      : a
b      : B
c      : C
```

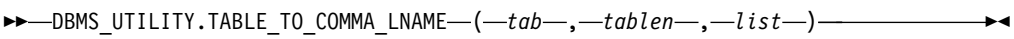
```
dblink : NULL
nextpos: 7

CALL name_tokenize( 'a.b.c@d' )

  Return Status = 0

name   : a.b.c@d
a      : A
b      : B
c      : C
dblink : D
nextpos: 7

CALL name_tokenize( 'a.b."c"@"d"' )

  Return Status = 0

name   : a.b."c"@"d"
a      : A
b      : B
c      : c
dblink : d
nextpos: 11
```

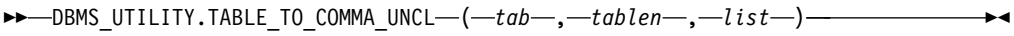
TABLE_TO_COMMA procedures - Convert a table of names into a comma-delimited list of names

The TABLE_TO_COMMA procedures convert an array of names into a comma-delimited list of names. Each array element becomes a list entry.

Note: The names must be formatted as valid identifiers.

Syntax





Parameters

tab
An input argument of type LNAME_ARRAY or UNCL_ARRAY that specifies the array containing names. See LNAME_ARRAY or UNCL_ARRAY for a description of *tab*.

tablen
An output argument of type INTEGER that returns the number of entries in *list*.

list
An output argument of type VARCHAR(32672) that returns the comma-delimited list of names from *tab*.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

Examples

Example 1: The following example first uses the COMMA_TO_TABLE_LNAME procedure to convert a comma-delimited list to a table. The TABLE_TO_COMMA_LNAME procedure then converts the table back to a comma-delimited list which is displayed.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE table_to_comma(
  IN p_list VARCHAR(100))
BEGIN
  DECLARE r_lname DBMS_UTILITY.LNAME_ARRAY;
  DECLARE v_length INTEGER;
  DECLARE v_listlen INTEGER;
  DECLARE v_list VARCHAR(80);

  CALL DBMS_UTILITY.COMMA_TO_TABLE_LNAME(p_list, v_length, r_lname);
  CALL DBMS_OUTPUT.PUT_LINE('Table Entries');
  CALL DBMS_OUTPUT.PUT_LINE('-------------');
  BEGIN
    DECLARE i INTEGER DEFAULT 1;
    DECLARE LOOP_LIMIT INTEGER;
    SET LOOP_LIMIT = v_length;

    WHILE i <= LOOP_LIMIT DO
      CALL DBMS_OUTPUT.PUT_LINE(r_lname(i));
      SET i = i + 1;
    END WHILE;
  END;
  CALL DBMS_OUTPUT.PUT_LINE('-------------');
  CALL DBMS_UTILITY.TABLE_TO_COMMA_LNAME(r_lname, v_listlen, v_list);
  CALL DBMS_OUTPUT.PUT_LINE('Comma-Delimited List: ' || v_list);
END@

CALL table_to_comma('sample_schema.dept,sample_schema.emp,sample_schema.jobhist')@
```

This example results in the following output:

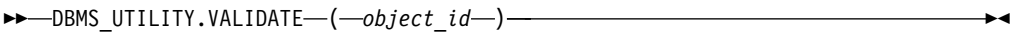
```
SET SERVEROUTPUT ON
DB20000I  The SET SERVEROUTPUT command completed successfully.

CREATE OR REPLACE PROCEDURE table_to_comma(
  IN p_list VARCHAR(100))
BEGIN
  DECLARE r_lname DBMS_UTILITY.LNAME_ARRAY;
  DECLARE v_length INTEGER;
  DECLARE v_listlen INTEGER;
  DECLARE v_list VARCHAR(80);

  CALL DBMS_UTILITY.COMMA_TO_TABLE_LNAME(p_list, v_length, r_lname);
  CALL DBMS_OUTPUT.PUT_LINE('Table Entries');
  CALL DBMS_OUTPUT.PUT_LINE('-------------');
  BEGIN
    DECLARE i INTEGER DEFAULT 1;
    DECLARE LOOP_LIMIT INTEGER;
    SET LOOP_LIMIT = v_length;

    WHILE i <= LOOP_LIMIT DO
      CALL DBMS_OUTPUT.PUT_LINE(r_lname(i));
      SET i = i + 1;
    END WHILE;
  END;
  CALL DBMS_OUTPUT.PUT_LINE('-------------');
  CALL DBMS_UTILITY.TABLE_TO_COMMA_LNAME(r_lname, v_listlen, v_list);
  CALL DBMS_OUTPUT.PUT_LINE('Comma-Delimited List: ' || v_list);
END
```

```
DB20000I  The SQL command completed successfully.

CALL table_to_comma('sample_schema.dept,sample_schema.emp,sample_schema.jobhist')

  Return Status = 0

Table Entries
-------------
sample_schema.dept
sample_schema.emp
sample_schema.jobhist
-------------
Comma-Delimited List: sample_schema.dept,sample_schema.emp,sample_schema.jobhist
```

VALIDATE procedure - Change an invalid routine into a valid routine

The VALIDATE procedure provides the capability to change the state of an invalid routine to valid.

Syntax



Parameters

object_id
An input argument of type INTEGER that specifies the identifier of the routine to be changed to a valid state. The ROUTINEID column of the SYSCAT.ROUTINES view contains all the routine identifiers.

Authorization

EXECUTE privilege on the DBMS_UTILITY module.

MONREPORT module

The MONREPORT module provides a set of procedures for retrieving a variety of monitoring data and generating text reports.

The schema for this module is SYSIBMADM.

The MONREPORT module includes the following built-in routines.

Table 20. Built-in routines available in the MONREPORT module

Routine name	Description
CONNECTION procedure	The Connection report presents monitor data for each connection.
CURRENTAPPS procedure	The Current Applications report presents the current instantaneous state of processing of units of work, agents, and activities for each connection. The report starts with state information summed across connections, followed by a section for details for each connection.

Table 20. Built-in routines available in the MONREPORT module (continued)

Routine name	Description
CURRENTSQL procedure	The Current SQL report lists the top activities currently running, as measured by various metrics.
DBSUMMARY procedure	The Summary report contains in-depth monitor data for the entire database, as well as key performance indicators for each connection, workload, service class, and database member.
LOCKWAIT procedure	The Lock Waits report contains information about each lock wait currently in progress. Details include lock holder and requestor details, plus characteristics of the lock held and the lock requested.
PKGCACHE procedure	The Package Cache report lists the top statements accumulated in the package cache as measured by various metrics.

Usage notes

Monitor element names are displayed in upper case (for example, TOTAL_CPU_TIME). To find out more information about a monitor element, search the product documentation for the monitor name.

For reports with a *monitoring_interval* input, negative values in a report are inaccurate. This may occur during a rollover of source data counters. To determine accurate values, re-run the report after the rollover is complete.

Note: The reports are implemented using SQL procedures within modules, and as such can be impacted by the package cache configuration. If you observe slow performance when running the reports, inspect your package cache configuration to ensure it is sufficient for your workload. For further information, see "pckcachesz - Package cache size configuration parameter".

The following examples demonstrate various ways to call the MONREPORT routines. The examples show the MONREPORT.CONNECTION(*monitoring_interval*, *application_handle*) procedure. You can handle optional parameters for which you do not want to enter a value in the following ways:

- You can always specify null or DEFAULT.
- For character inputs, you can specify an empty string (' ').
- If it is the last parameter, you can omit it.

To generate a report that includes a section for each connection, with the default monitoring interval of 10 seconds, make the following call to the MONREPORT.CONNECTION procedure:

```
call monreport.connection()
```

To generate a report that includes a section only for the connection with application handle 32, with the default monitoring interval of 10 seconds, you can make either of the following calls to the MONREPORT.CONNECTION procedure:

```
call monreport.connection(DEFAULT, 32)
call monreport.connection(10, 32)
```

To generate a report that includes a section for each connection, with a monitoring interval of 60 seconds, you can make either of the following calls to the MONREPORT.CONNECTION procedure:

```
call monreport.connection(60)
call monreport.connection(60, null)
```

By default, the reports in this module are generated in English. To change the language in which the reports are generated, change the CURRENT LOCALE LC_MESSAGES special register. For example, to generate the CONNECTION report in French, issue the following commands:

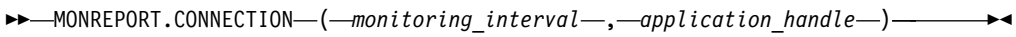
```
SET CURRENT LOCALE LC_MESSAGES = 'CLDR 1.5:fr_FR'
CALL MONREPORT.CONNECTION
```

Ensure that the language in which the reports are generated is supported by the database code page. If the database code page is Unicode, you can generate the reports in any language.

CONNECTION procedure - Generate a report on connection metrics

The CONNECTION procedure gathers monitor data for each connection and produces a text-formatted report.

Syntax



Parameters

monitoring_interval
An optional input argument of type INTEGER that specifies the duration in seconds that monitoring data is collected before it is reported. For example, if you specify a monitoring interval of `30`, the routine calls table functions, waits 30 seconds and calls the table functions again. The routine then calculates the difference, which reflects changes during the interval. If the *monitoring_interval* argument is not specified (or if null is specified), the default value is `10`. The range of valid inputs are the integer values `0`-`3600` (that is, up to 1 hour).

application_handle
An optional input argument of type BIGINT that specifies an application handle that identifies a connection. If the *application_handle* argument is not specified (or if null is specified), the report includes a section for each connection. The default is null.

Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

Examples

The following examples demonstrate various ways to call the CONNECTION procedure.

This example produces a report for all connections, with data displayed corresponding to an interval of 30 seconds:

```
call monreport.connection(30);
```

This example produces a report for a connection with an application handle of 34. Data is displayed based on absolute totals accumulated in the source table functions (rather than based on the current interval):

```
call monreport.connection(0, 34);
```

This next example produces a report for a connection with an application handle of 34. Data is displayed corresponding to an interval of 10 seconds.

```
call monreport.connection(DEFAULT, 34);
```

The final example produces the default report: for all connections, with data displayed corresponding to an interval of 10 seconds:

```
call monreport.connection;
```

Here is an example of the report output for the default procedure call (all connections, 10 second interval):

```
Result set 1
--------------

TEXT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Monitoring report - connection
--------------------------------------------------------------------------------
Database:               SAMPLE
Generated:              04/06/2010 13:36:52
Interval monitored:     10
-- Command options --
APPLICATION_HANDLE:     All

================================================================================
Part 1 - Summary of connections

--------------------------------------------------------------------------------
     APPLICATION  TOTAL_        TOTAL_    ACT_COMPLETED  TOTAL_WAIT  CLIENT_IDLE
#    _HANDLE      CPU_TIME      ACT_TIME  _TOTAL         _TIME       _WAIT_TIME
---  -----------  ------------  --------  -------------  ----------  -----------
1    180          0             0         0              0           0
2    65711        116307        675       1              410         9884
3    131323       116624        679       1              717         12895

================================================================================
Part 2 - Details for each connection

connection #:1
--------------------------------------------------------------------------------

--Connection identifiers--
Application identifiers
  APPLICATION_HANDLE                = 180
  APPLICATION_NAME                  = db2bp
  APPLICATION_ID                    = *N0.jwr.100406173420
Authorization IDs
  SYSTEM_AUTHID                     = JWR
  SESSION_AUTHID                    = JWR
Client attributes
  CLIENT_ACCTNG                     =
  CLIENT_USERID                     =
  CLIENT_APPLNAME                   =
```

```
  CLIENT_WRKSTNNAME                =
  CLIENT_PID                       = 29987
  CLIENT_PRDID                     = SQL09081
  CLIENT_PLATFORM                  = LINUXX8664
  CLIENT_PROTOCOL                  = LOCAL
-- Other connection details --
CONNECTION_START_TIME              = 2010-04-06-13.34.20.635181
NUM_LOCKS_HELD                     = 9

Work volume and throughput
-------------------------------------------------------------------------------
                                 Per second             Total
                                 ---------------------  ----------------------
TOTAL_APP_COMMITS                0                      0
ACT_COMPLETED_TOTAL              0                      0
APP_RQSTS_COMPLETED_TOTAL        0                      0

TOTAL_CPU_TIME                   = 0
TOTAL_CPU_TIME per request       = 0

Row processing
  ROWS_READ/ROWS_RETURNED        = 0 (0/0)
  ROWS_MODIFIED                  = 0

Wait times
-------------------------------------------------------------------------------

-- Wait time as a percentage of elapsed time --

                                       %    Wait time/Total time
                                       ---  ----------------------------------
For requests                           0    0/0
For activities                         0    0/0

-- Time waiting for next client request --

CLIENT_IDLE_WAIT_TIME               = 0
CLIENT_IDLE_WAIT_TIME per second    = 0

-- Detailed breakdown of TOTAL_WAIT_TIME --

                             %    Total
                             ---  -----------------------------------------
TOTAL_WAIT_TIME              100  3434

I/O wait time
  POOL_READ_TIME             23   805
  POOL_WRITE_TIME            8    280
  DIRECT_READ_TIME           3    131
  DIRECT_WRITE_TIME          3    104
  LOG_DISK_WAIT_TIME         10   344
LOCK_WAIT_TIME               0    18
AGENT_WAIT_TIME              0    0
Network and FCM
  TCPIP_SEND_WAIT_TIME       0    0
  TCPIP_RECV_WAIT_TIME       0    0
  IPC_SEND_WAIT_TIME         0    0
  IPC_RECV_WAIT_TIME         0    0
  FCM_SEND_WAIT_TIME         0    0
  FCM_RECV_WAIT_TIME         6    212
WLM_QUEUE_TIME_TOTAL         0    0
CF_WAIT_TIME                 32   1101
RECLAIM_WAIT_TIME            2    98
SMP_RECLAIM_WAIT_TIME        3    118

Component times
-------------------------------------------------------------------------------
```

```
-- Detailed breakdown of processing time --

                                   %                Total
                                   ---------------  -------------------------
Total processing                   100              0

Section execution
  TOTAL_SECTION_PROC_TIME          0                0
    TOTAL_SECTION_SORT_PROC_TIME   0                0
Compile
  TOTAL_COMPILE_PROC_TIME          0                0
  TOTAL_IMPLICIT_COMPILE_PROC_TIME 0                0
Transaction end processing
  TOTAL_COMMIT_PROC_TIME           0                0
  TOTAL_ROLLBACK_PROC_TIME         0                0
Utilities
  TOTAL_RUNSTATS_PROC_TIME         0                0
  TOTAL_REORGS_PROC_TIME           0                0
  TOTAL_LOAD_PROC_TIME             0                0

Buffer pool
------------------------------------------------------------------------------
  Buffer pool hit ratios

Type             Ratio            Formula
---------------  ---------------  ---------------------------------------------
Data             100              (1-(0+0-0)/(27+0))
Index            100              (1-(0+0-0)/(24+0))
XDA              0                (1-(0+0-0)/(0+0))
COL              0                (1-(0+0-0)/(0+0))
LBP Data         100              (27-0)/(27+0)
LBP Index        0                (0-0)/(24+0)
LBP XDA          0                (0-0)/(0+0)
LBP COL          0                (0-0)/(0+0)
GBP Data         0                (0 - 0)/0
GBP Index        0                (0 - 0)/0
GBP XDA          0                (0 - 0)/0
GBP COL          0                (0 - 0)/0

I/O
------------------------------------------------------------------------------
Buffer pool reads
  POOL_DATA_L_READS                = 27
  POOL_TEMP_DATA_L_READS           = 0
  POOL_DATA_P_READS                = 0
  POOL_TEMP_DATA_P_READS           = 0
  POOL_ASYNC_DATA_READS            = 0
  POOL_INDEX_L_READS               = 24
  POOL_TEMP_INDEX_L_READS          = 0
  POOL_INDEX_P_READS               = 0
  POOL_TEMP_INDEX_P_READS          = 0
  POOL_ASYNC_INDEX_READS           = 0
  POOL_XDA_L_READS                 = 0
  POOL_TEMP_XDA_L_READS            = 0
  POOL_XDA_P_READS                 = 0
  POOL_TEMP_XDA_P_READS            = 0
  POOL_ASYNC_XDA_READS             = 0
  POOL_COL_L_READS                 = 0
  POOL_TEMP_COL_L_READS            = 0
  POOL_COL_P_READS                 = 0
  POOL_TEMP_COL_P_READS            = 0
  POOL_ASYNC_COL_READS             = 0
Buffer pool pages found
  POOL_DATA_LBP_PAGES_FOUND        = 27
  POOL_ASYNC_DATA_LBP_PAGES_FOUND  = 0
  POOL_INDEX_LBP_PAGES_FOUND       = 0
  POOL_ASYNC_INDEX_LBP_PAGES_FOUND = 0
```

```
  POOL_XDA_LBP_PAGES_FOUND          = 0
  POOL_ASYNC_XDA_LBP_PAGES_FOUND    = 0
  POOL_COL_LBP_PAGES_FOUND          = 0
  POOL_ASYNC_COL_LBP_PAGES_FOUND    = 0
Buffer pool writes
  POOL_DATA_WRITES                  = 0
  POOL_XDA_WRITES                   = 0
  POOL_INDEX_WRITES                 = 0
  POOL_COL_WRITES                   = 0
Direct I/O
  DIRECT_READS                      = 620
  DIRECT_READ_REQS                  = 15
  DIRECT_WRITES                     = 0
  DIRECT_WRITE_REQS                 = 0
Log I/O
  LOG_DISK_WAITS_TOTAL              = 0

Locking
------------------------------------------------------------------------------
                        Per activity                    Total
                        ------------------------------  ----------------------
LOCK_WAIT_TIME          0                               0
LOCK_WAITS              0                               0
LOCK_TIMEOUTS           0                               0
DEADLOCKS               0                               0
LOCK_ESCALS             0                               0

Routines
------------------------------------------------------------------------------
                             Per activity               Total
                             -------------------------  ------------------------
TOTAL_ROUTINE_INVOCATIONS    0                          0
TOTAL_ROUTINE_TIME           0                          0

TOTAL_ROUTINE_TIME per invocation   = 0

Sort
------------------------------------------------------------------------------
TOTAL_SORTS                         = 0
SORT_OVERFLOWS                      = 0
POST_THRESHOLD_SORTS                = 0
POST_SHRTHRESHOLD_SORTS             = 0

Network
------------------------------------------------------------------------------
Communications with remote clients
TCPIP_SEND_VOLUME per send          = 0          (0/0)
TCPIP_RECV_VOLUME per receive       = 0          (0/0)

Communications with local clients
IPC_SEND_VOLUME per send            = 0          (0/0)
IPC_RECV_VOLUME per receive         = 0          (0/0)

Fast communications manager
FCM_SEND_VOLUME per send            = 0          (0/0)
FCM_RECV_VOLUME per receive         = 0          (0/0)

Other
------------------------------------------------------------------------------
Compilation
  TOTAL_COMPILATIONS                = 0
  PKG_CACHE_INSERTS                 = 0
  PKG_CACHE_LOOKUPS                 = 0
Catalog cache
  CAT_CACHE_INSERTS                 = 0
  CAT_CACHE_LOOKUPS                 = 0
Transaction processing
```

```
  TOTAL_APP_COMMITS                = 0
  INT_COMMITS                      = 0
  TOTAL_APP_ROLLBACKS              = 0
  INT_ROLLBACKS                    = 0
Log buffer
  NUM_LOG_BUFFER_FULL              = 0
Activities aborted/rejected
  ACT_ABORTED_TOTAL                = 0
  ACT_REJECTED_TOTAL               = 0
Workload management controls
  WLM_QUEUE_ASSIGNMENTS_TOTAL      = 0
  WLM_QUEUE_TIME_TOTAL             = 0

Db2 utility operations
------------------------------------------------------------------------------
  TOTAL_RUNSTATS                   = 0
  TOTAL_REORGS                     = 0
  TOTAL_LOADS                      = 0

connection #:2
------------------------------------------------------------------------------

--Connection identifiers--
Application identifiers
  APPLICATION_HANDLE               = 65711
  APPLICATION_NAME                 = db2bp
  APPLICATION_ID                   = *N1.jwr.100406173430
Authorization IDs
  SYSTEM_AUTHID                    = JWR
  SESSION_AUTHID                   = JWR
Client attributes
  CLIENT_ACCTNG                    =
  CLIENT_USERID                    =
  CLIENT_APPLNAME                  =
  CLIENT_WRKSTNNAME                =
  CLIENT_PID                       = 30044
  CLIENT_PRDID                     = SQL09081
  CLIENT_PLATFORM                  = LINUXX8664
  CLIENT_PROTOCOL                  = LOCAL
-- Other connection details --
CONNECTION_START_TIME              = 2010-04-06-13.34.31.058344
NUM_LOCKS_HELD                     = 0

Work volume and throughput
------------------------------------------------------------------------------
                                 Per second            Total
                                 --------------------  ----------------------
TOTAL_APP_COMMITS                0                     1
ACT_COMPLETED_TOTAL              0                     1
APP_RQSTS_COMPLETED_TOTAL        0                     2

TOTAL_CPU_TIME                   = 116307
TOTAL_CPU_TIME per request       = 58153

Row processing
  ROWS_READ/ROWS_RETURNED        = 0 (8/0)
  ROWS_MODIFIED                  = 5

Wait times
------------------------------------------------------------------------------

-- Wait time as a percentage of elapsed time --

                                       %    Wait time/Total time
                                       ---  ---------------------------------
For requests                           58   410/696
For activities                         58   398/675
```

```
-- Time waiting for next client request --

CLIENT_IDLE_WAIT_TIME              = 9884
CLIENT_IDLE_WAIT_TIME per second   = 988

-- Detailed breakdown of TOTAL_WAIT_TIME --

                             %    Total
                             ---  -----------------------------------------
TOTAL_WAIT_TIME              100  410

I/O wait time
  POOL_READ_TIME             5    23
  POOL_WRITE_TIME            28   116
  DIRECT_READ_TIME           0    1
  DIRECT_WRITE_TIME          0    4
  LOG_DISK_WAIT_TIME         11   48
LOCK_WAIT_TIME               2    11
AGENT_WAIT_TIME              0    0
Network and FCM
  TCPIP_SEND_WAIT_TIME       0    0
  TCPIP_RECV_WAIT_TIME       0    0
  IPC_SEND_WAIT_TIME         0    1
  IPC_RECV_WAIT_TIME         0    0
  FCM_SEND_WAIT_TIME         0    0
  FCM_RECV_WAIT_TIME         1    5
WLM_QUEUE_TIME_TOTAL         0    0
CF_WAIT_TIME                 17   73
RECLAIM_WAIT_TIME            23   96
SMP_RECLAIM_WAIT_TIME        4    20

Component times
------------------------------------------------------------------------------
-- Detailed breakdown of processing time --

                                   %                Total
                                   ---------------  --------------------------
Total processing                   100              286

Section execution
  TOTAL_SECTION_PROC_TIME          96               276
    TOTAL_SECTION_SORT_PROC_TIME   0                0
Compile
  TOTAL_COMPILE_PROC_TIME          0                2
  TOTAL_IMPLICIT_COMPILE_PROC_TIME 0                0
Transaction end processing
  TOTAL_COMMIT_PROC_TIME           1                4
  TOTAL_ROLLBACK_PROC_TIME         0                0
Utilities
  TOTAL_RUNSTATS_PROC_TIME         0                0
  TOTAL_REORGS_PROC_TIME           0                0
  TOTAL_LOAD_PROC_TIME             0                0

Buffer pool
------------------------------------------------------------------------------
Buffer pool hit ratios

Type             Ratio            Reads (Logical/Physical)
---------------  ---------------  -----------------------------------------
Data             91               72/6
Index            100              46/0
XDA              0                0/0
COL              0                0/0
Temp data        0                0/0
Temp index       0                0/0
Temp XDA         0                0/0
```

```
Temp COL         0               0/0
GBP Data         60              (10 - 6)/10
GBP Index        0               (8 - 0)/8
GBP COL          0               (0 - 0)/0
LBP Data         52              (34 - 0)/72
LBP Index        0               (46 - 0)/46
LBP COL          0               (0 - 0)/(0 + 0)

I/O
------------------------------------------------------------------------------
Buffer pool writes
  POOL_DATA_WRITES      = 36
  POOL_XDA_WRITES       = 0
  POOL_INDEX_WRITES     = 0
  POOL_COL_WRITES       = 0

Direct I/O
  DIRECT_READS          = 1
  DIRECT_READ_REQS      = 1
  DIRECT_WRITES         = 4
  DIRECT_WRITE_REQS     = 1
Log I/O
  LOG_DISK_WAITS_TOTAL  = 13

Locking
------------------------------------------------------------------------------
                       Per activity                  Total
                       ----------------------------  ----------------------
LOCK_WAIT_TIME         11                            11
LOCK_WAITS             100                           1
LOCK_TIMEOUTS          0                             0
DEADLOCKS              0                             0
LOCK_ESCALS            0                             0

Routines
------------------------------------------------------------------------------
                              Per activity             Total
                              -----------------------  -----------------------
TOTAL_ROUTINE_INVOCATIONS     0                        0
TOTAL_ROUTINE_TIME            0                        0

TOTAL_ROUTINE_TIME per invocation   = 0

Sort
------------------------------------------------------------------------------
TOTAL_SORTS                         = 0
SORT_OVERFLOWS                      = 0
POST_THRESHOLD_SORTS                = 0
POST_SHRTHRESHOLD_SORTS             = 0

Network
------------------------------------------------------------------------------
Communications with remote clients
TCPIP_SEND_VOLUME per send          = 0          (0/0)
TCPIP_RECV_VOLUME per receive       = 0          (0/0)

Communications with local clients
IPC_SEND_VOLUME per send            = 54         (108/2)
IPC_RECV_VOLUME per receive         = 69         (138/2)

Fast communications manager
FCM_SEND_VOLUME per send            = 0          (0/0)
FCM_RECV_VOLUME per receive         = 432        (2592/6)

Other
------------------------------------------------------------------------------
Compilation
```

```
  TOTAL_COMPILATIONS                = 1
  PKG_CACHE_INSERTS                 = 2
  PKG_CACHE_LOOKUPS                 = 2
Catalog cache
  CAT_CACHE_INSERTS                 = 3
  CAT_CACHE_LOOKUPS                 = 8
Transaction processing
  TOTAL_APP_COMMITS                 = 1
  INT_COMMITS                       = 0
  TOTAL_APP_ROLLBACKS               = 0
  INT_ROLLBACKS                     = 0
Log buffer
  NUM_LOG_BUFFER_FULL               = 0
Activities aborted/rejected
  ACT_ABORTED_TOTAL                 = 0
  ACT_REJECTED_TOTAL                = 0
Workload management controls
  WLM_QUEUE_ASSIGNMENTS_TOTAL       = 0
  WLM_QUEUE_TIME_TOTAL              = 0

Db2 utility operations
------------------------------------------------------------------------------
  TOTAL_RUNSTATS                    = 0
  TOTAL_REORGS                      = 0
  TOTAL_LOADS                       = 0

connection #:3
------------------------------------------------------------------------------

--Connection identifiers--
Application identifiers
  APPLICATION_HANDLE                = 131323
  APPLICATION_NAME                  = db2bp
  APPLICATION_ID                    = *N2.jwr.100406173452
Authorization IDs
  SYSTEM_AUTHID                     = JWR
  SESSION_AUTHID                    = JWR
Client attributes
  CLIENT_ACCTNG                     =
  CLIENT_USERID                     =
  CLIENT_APPLNAME                   =
  CLIENT_WRKSTNNAME                 =
  CLIENT_PID                        = 30510
  CLIENT_PRDID                      = SQL09081
  CLIENT_PLATFORM                   = LINUXX8664
  CLIENT_PROTOCOL                   = LOCAL
-- Other connection details --
CONNECTION_START_TIME               = 2010-04-06-13.34.52.398427
NUM_LOCKS_HELD                      = 0

Work volume and throughput
------------------------------------------------------------------------------
                                  Per second             Total
                                  ---------------------  ----------------------
TOTAL_APP_COMMITS                 0                      1
ACT_COMPLETED_TOTAL               0                      1
APP_RQSTS_COMPLETED_TOTAL         0                      2

TOTAL_CPU_TIME                    = 116624
TOTAL_CPU_TIME per request        = 58312

Row processing
  ROWS_READ/ROWS_RETURNED         = 0 (18/0)
  ROWS_MODIFIED                   = 4

Wait times
------------------------------------------------------------------------------
```

```
-- Wait time as a percentage of elapsed time --

                                        %    Wait time/Total time
                                        ---  --------------------------------
For requests                            82   717/864
For activities                          80   549/679

-- Time waiting for next client request --

CLIENT_IDLE_WAIT_TIME               = 12895
CLIENT_IDLE_WAIT_TIME per second    = 1289

-- Detailed breakdown of TOTAL_WAIT_TIME --

                              %    Total
                              ---  ------------------------------------------
TOTAL_WAIT_TIME               100  717

I/O wait time
  POOL_READ_TIME              2    16
  POOL_WRITE_TIME             18   136
  DIRECT_READ_TIME            0    3
  DIRECT_WRITE_TIME           0    2
  LOG_DISK_WAIT_TIME          10   77
LOCK_WAIT_TIME                3    27
AGENT_WAIT_TIME               0    0
Network and FCM
  TCPIP_SEND_WAIT_TIME        0    0
  TCPIP_RECV_WAIT_TIME        0    0
  IPC_SEND_WAIT_TIME          0    0
  IPC_RECV_WAIT_TIME          0    0
  FCM_SEND_WAIT_TIME          0    0
  FCM_RECV_WAIT_TIME          21   157
WLM_QUEUE_TIME_TOTAL          0    0
CF_WAIT_TIME                  9    66
RECLAIM_WAIT_TIME             12   92
SMP_RECLAIM_WAIT_TIME         16   119

Component times
------------------------------------------------------------------------------
-- Detailed breakdown of processing time --

                                    %                Total
                                    ---------------  --------------------------
Total processing                    100              147

Section execution
  TOTAL_SECTION_PROC_TIME           89               131
    TOTAL_SECTION_SORT_PROC_TIME    0                0
Compile
  TOTAL_COMPILE_PROC_TIME           4                6
  TOTAL_IMPLICIT_COMPILE_PROC_TIME  0                0
Transaction end processing
  TOTAL_COMMIT_PROC_TIME            1                2
  TOTAL_ROLLBACK_PROC_TIME          0                0
Utilities
  TOTAL_RUNSTATS_PROC_TIME          0                0
  TOTAL_REORGS_PROC_TIME            0                0
  TOTAL_LOAD_PROC_TIME              0                0

Buffer pool
------------------------------------------------------------------------------
Buffer pool hit ratios

Type             Ratio            Reads (Logical/Physical)
---------------  ---------------  -------------------------------------------
```

```
Data             91              47/4
Index            100             78/0
XDA              0               0/0
COL              0               0/0
Temp data        0               0/0
Temp index       0               0/0
Temp XDA         0               0/0
Temp COL         0               0/0
GBP Data         26              (15 - 4)/15
GBP Index        0               (9 - 0)/9
GBP COL          0               (0 - 0)/0
LBP Data         6               (44 - 0)/47
LBP Index        48              (40 - 0)/78
LBP COL          0               (0 - 0)/(0 + 0)

I/O
-------------------------------------------------------------------------------
Buffer pool writes
  POOL_DATA_WRITES      = 3
  POOL_XDA_WRITES       = 0
  POOL_INDEX_WRITES     = 35
  POOL_COL_WRITES       = 0

Direct I/O
  DIRECT_READS          = 15
  DIRECT_READ_REQS      = 4
  DIRECT_WRITES         = 6
  DIRECT_WRITE_REQS     = 1
Log I/O
  LOG_DISK_WAITS_TOTAL  = 18

Locking
-------------------------------------------------------------------------------
                        Per activity                  Total
                        ----------------------------  ----------------------
LOCK_WAIT_TIME          27                            27
LOCK_WAITS              200                           2
LOCK_TIMEOUTS           0                             0
DEADLOCKS               0                             0
LOCK_ESCALS             0                             0

Routines
-------------------------------------------------------------------------------
                             Per activity             Total
                             -----------------------  ------------------------
TOTAL_ROUTINE_INVOCATIONS    0                        0
TOTAL_ROUTINE_TIME           0                        0

TOTAL_ROUTINE_TIME per invocation   = 0

Sort
-------------------------------------------------------------------------------
TOTAL_SORTS                         = 1
SORT_OVERFLOWS                      = 0
POST_THRESHOLD_SORTS                = 0
POST_SHRTHRESHOLD_SORTS             = 0

Network
-------------------------------------------------------------------------------
Communications with remote clients
TCPIP_SEND_VOLUME per send          = 0          (0/0)
TCPIP_RECV_VOLUME per receive       = 0          (0/0)

Communications with local clients
IPC_SEND_VOLUME per send            = 54         (108/2)
IPC_RECV_VOLUME per receive         = 73         (146/2)
```

```
Fast communications manager
FCM_SEND_VOLUME per send            = 0          (0/0)
FCM_RECV_VOLUME per receive         = 1086       (10864/10)

Other
--------------------------------------------------------------------------------
Compilation
  TOTAL_COMPILATIONS                = 1
  PKG_CACHE_INSERTS                 = 2
  PKG_CACHE_LOOKUPS                 = 2
Catalog cache
  CAT_CACHE_INSERTS                 = 0
  CAT_CACHE_LOOKUPS                 = 9
Transaction processing
  TOTAL_APP_COMMITS                 = 1
  INT_COMMITS                       = 0
  TOTAL_APP_ROLLBACKS               = 0
  INT_ROLLBACKS                     = 0
Log buffer
  NUM_LOG_BUFFER_FULL               = 0
Activities aborted/rejected
  ACT_ABORTED_TOTAL                 = 0
  ACT_REJECTED_TOTAL                = 0
Workload management controls
  WLM_QUEUE_ASSIGNMENTS_TOTAL       = 0
  WLM_QUEUE_TIME_TOTAL              = 0

Db2 utility operations
--------------------------------------------------------------------------------
  TOTAL_RUNSTATS                    = 0
  TOTAL_REORGS                      = 0
  TOTAL_LOADS                       = 0

628 record(s) selected.

Return Status = 0
```

CURRENTAPPS procedure - Generate a report of point-in-time application processing metrics

The CURRENTAPPS procedure gathers information about the current instantaneous state of processing of units or work, agents, and activities for each connection.

Syntax

Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

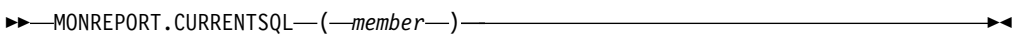
The following examples demonstrate ways to call the CURRENTAPPS procedure:

```
call monreport.currentapps;
call monreport.currentapps();
```

CURRENTSQL procedure - Generate a report that summarizes activities

The CURRENTSQL procedure generates a text-formatted report that summarizes currently running activities.

Syntax



Parameters

member
An input argument of type SMALLINT that determines whether to show data for a particular member or partition, or to show data summed across all members. If this argument is not specified (or if null is specified), the report shows values summed across all members. If a valid member number is specified, the report shows values for that member.

Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

The following examples demonstrate various ways to call the CURRENTSQL procedure. The first example produces a report that shows activity metrics aggregated across all members:

```
call monreport.currentsql;
```

The next example produces a report that shows activity metrics specific to the activity performance on member number 4.

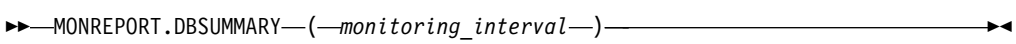
```
call monreport.currentsql(4);
```

DBSUMMARY procedure - Generate a summary report of system and application performance metrics

The DBSUMMARY procedure generates a text-formatted monitoring report that summarizes system and application performance metrics.

The DB Summary report contains in-depth monitor data for the entire database as well as key performance indicators for each connection, workload, service class, and database member.

Syntax



Parameters

monitoring_interval
An optional input argument of type INTEGER that specifies the duration in seconds that monitoring data is collected before it is reported. For example, if you specify a monitoring interval of 30, the routine calls the table functions, waits 30 seconds and then calls the table functions again. The DBSUMMARY procedure then calculates the difference, which reflects changes during the

interval. If the *monitoring_interval* argument is not specified (or if null is specified), the default value is 10. The range of valid inputs are the integer values 0-3600 (that is, up to 1 hour).

Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

Examples

The following examples demonstrate various ways to call the DBSUMMARY procedure.

The first example produces a report that displays data corresponding to an interval of 30 seconds.

```
call monreport.dbsummary(30);
```

The next example produces a report that displays data corresponding to an interval of 10 seconds (the default value):

```
call monreport.dbsummary;
```

This procedure call returns the following output:

```
  Result set 1
  --------------

  TEXT
  ------------------------------------------------------------------------------
  ------------------------------------------------------------------------------
  Monitoring report - database summary
  ------------------------------------------------------------------------------
  Database:                                SAMPLE
  Generated:                               04/06/2010 13:35:24
  Interval monitored:                      10

  ==============================================================================
  Part 1 - System performance

  Work volume and throughput
  ------------------------------------------------------------------------------
                                   Per second             Total
                                   ---------------------  ----------------------
  TOTAL_APP_COMMITS                0                      2
  ACT_COMPLETED_TOTAL              0                      9
  APP_RQSTS_COMPLETED_TOTAL        0                      6

  TOTAL_CPU_TIME                   = 2649800
  TOTAL_CPU_TIME per request       = 441633

  Row processing
    ROWS_READ/ROWS_RETURNED        = 97 (685/7)
    ROWS_MODIFIED                  = 117

  Wait times
  ------------------------------------------------------------------------------

  -- Wait time as a percentage of elapsed time --

                                        %    Wait time/Total time
                                        ---  ---------------------------------
  For requests                          19   3434/17674
  For activities                        10   1203/11613
```

```
-- Time waiting for next client request --

CLIENT_IDLE_WAIT_TIME              = 70566
CLIENT_IDLE_WAIT_TIME per second   = 7056

-- Detailed breakdown of TOTAL_WAIT_TIME --

                             %    Total
                             ---  ------------------------------------------
TOTAL_WAIT_TIME              100  3434

I/O wait time
  POOL_READ_TIME             23   805
  POOL_WRITE_TIME            8    280
  DIRECT_READ_TIME           3    131
  DIRECT_WRITE_TIME          3    104
  LOG_DISK_WAIT_TIME         10   344
LOCK_WAIT_TIME               0    18
AGENT_WAIT_TIME              0    0
Network and FCM
  TCPIP_SEND_WAIT_TIME       0    0
  TCPIP_RECV_WAIT_TIME       0    0
  IPC_SEND_WAIT_TIME         0    0
  IPC_RECV_WAIT_TIME         0    0
  FCM_SEND_WAIT_TIME         0    0
  FCM_RECV_WAIT_TIME         6    212
WLM_QUEUE_TIME_TOTAL         0    0
CF_WAIT_TIME                 32   1101
RECLAIM_WAIT_TIME            2    98
SMP_RECLAIM_WAIT_TIME        3    118

Component times
------------------------------------------------------------------------------
-- Detailed breakdown of processing time --

                                   %                Total
                                   ---------------  --------------------------
Total processing                   100              14240

Section execution
  TOTAL_SECTION_PROC_TIME          2                365
    TOTAL_SECTION_SORT_PROC_TIME   0                0
Compile
  TOTAL_COMPILE_PROC_TIME          0                17
  TOTAL_IMPLICIT_COMPILE_PROC_TIME 2                294
Transaction end processing
  TOTAL_COMMIT_PROC_TIME           0                36
  TOTAL_ROLLBACK_PROC_TIME         0                0
Utilities
  TOTAL_RUNSTATS_PROC_TIME         0                0
  TOTAL_REORGS_PROC_TIME           0                0
  TOTAL_LOAD_PROC_TIME             0                0

Buffer pool
------------------------------------------------------------------------------
Buffer pool hit ratios

Type             Ratio            Formula
---------------  ---------------  ------------------------------------------
Data             100              (1-(0+0-0)/(27+0))
Index            100              (1-(0+0-0)/(24+0))
XDA              0                (1-(0+0-0)/(0+0))
COL              0                (1-(0+0-0)/(0+0))
LBP Data         100              (27-0)/(27+0)
LBP Index        0                (0-0)/(24+0)
LBP XDA          0                (0-0)/(0+0)
```

```
LBP COL          0               (0-0)/(0+0)
GBP Data         0               (0 - 0)/0
GBP Index        0               (0 - 0)/0
GBP XDA          0               (0 - 0)/0
GBP COL          0               (0 - 0)/0

I/O
-------------------------------------------------------------------------------
Buffer pool reads
  POOL_DATA_L_READS                = 27
  POOL_TEMP_DATA_L_READS           = 0
  POOL_DATA_P_READS                = 0
  POOL_TEMP_DATA_P_READS           = 0
  POOL_ASYNC_DATA_READS            = 0
  POOL_INDEX_L_READS               = 24
  POOL_TEMP_INDEX_L_READS          = 0
  POOL_INDEX_P_READS               = 0
  POOL_TEMP_INDEX_P_READS          = 0
  POOL_ASYNC_INDEX_READS           = 0
  POOL_XDA_L_READS                 = 0
  POOL_TEMP_XDA_L_READS            = 0
  POOL_XDA_P_READS                 = 0
  POOL_TEMP_XDA_P_READS            = 0
  POOL_ASYNC_XDA_READS             = 0
  POOL_COL_L_READS                 = 0
  POOL_TEMP_COL_L_READS            = 0
  POOL_COL_P_READS                 = 0
  POOL_TEMP_COL_P_READS            = 0
  POOL_ASYNC_COL_READS             = 0
Buffer pool pages found
  POOL_DATA_LBP_PAGES_FOUND        = 27
  POOL_ASYNC_DATA_LBP_PAGES_FOUND  = 0
  POOL_INDEX_LBP_PAGES_FOUND       = 0
  POOL_ASYNC_INDEX_LBP_PAGES_FOUND = 0
  POOL_XDA_LBP_PAGES_FOUND         = 0
  POOL_ASYNC_XDA_LBP_PAGES_FOUND   = 0
  POOL_COL_LBP_PAGES_FOUND         = 0
  POOL_ASYNC_COL_LBP_PAGES_FOUND   = 0
Buffer pool writes
  POOL_DATA_WRITES                 = 0
  POOL_XDA_WRITES                  = 0
  POOL_INDEX_WRITES                = 0
  POOL_COL_WRITES                  = 0
Direct I/O
  DIRECT_READS                     = 620
  DIRECT_READ_REQS                 = 15
  DIRECT_WRITES                    = 0
  DIRECT_WRITE_REQS                = 0
Log I/O
  LOG_DISK_WAITS_TOTAL             = 0

Locking
-------------------------------------------------------------------------------
                        Per activity                  Total
                        ----------------------------  ----------------------
LOCK_WAIT_TIME          2                             18
LOCK_WAITS              22                            2
LOCK_TIMEOUTS           0                             0
DEADLOCKS               0                             0
LOCK_ESCALS             0                             0

Routines
-------------------------------------------------------------------------------
                              Per activity            Total
                              ----------------------  ----------------------
TOTAL_ROUTINE_INVOCATIONS     0                       1
TOTAL_ROUTINE_TIME            1117                    10058
```

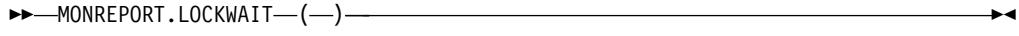
```
TOTAL_ROUTINE_TIME per invocation   = 10058

Sort
--------------------------------------------------------------------------------
TOTAL_SORTS                        = 5
SORT_OVERFLOWS                     = 0
POST_THRESHOLD_SORTS               = 0
POST_SHRTHRESHOLD_SORTS            = 0

Network
--------------------------------------------------------------------------------
Communications with remote clients
TCPIP_SEND_VOLUME per send         = 0          (0/0)
TCPIP_RECV_VOLUME per receive      = 0          (0/0)

Communications with local clients
IPC_SEND_VOLUME per send           = 137        (1101/8)
IPC_RECV_VOLUME per receive        = 184        (1106/6)

Fast communications manager
FCM_SEND_VOLUME per send           = 3475       (31277/9)
FCM_RECV_VOLUME per receive        = 2433       (131409/54)

Other
--------------------------------------------------------------------------------
Compilation
  TOTAL_COMPILATIONS               = 4
  PKG_CACHE_INSERTS                = 11
  PKG_CACHE_LOOKUPS                = 13
Catalog cache
  CAT_CACHE_INSERTS                = 74
  CAT_CACHE_LOOKUPS                = 112
Transaction processing
  TOTAL_APP_COMMITS                = 2
  INT_COMMITS                      = 2
  TOTAL_APP_ROLLBACKS              = 0
  INT_ROLLBACKS                    = 0
Log buffer
  NUM_LOG_BUFFER_FULL              = 0
Activities aborted/rejected
  ACT_ABORTED_TOTAL                = 0
  ACT_REJECTED_TOTAL               = 0
Workload management controls
  WLM_QUEUE_ASSIGNMENTS_TOTAL      = 0
  WLM_QUEUE_TIME_TOTAL             = 0

Db2 utility operations
--------------------------------------------------------------------------------
  TOTAL_RUNSTATS                   = 0
  TOTAL_REORGS                     = 0
  TOTAL_LOADS                      = 0

================================================================================
Part 2 - Application performance drill down

Application performance database-wide
--------------------------------------------------------------------------------
TOTAL_CPU_TIME          TOTAL_       TOTAL_APP_     ROWS_READ +
per request             WAIT_TIME %  COMMITS        ROWS_MODIFIED
----------------------  -----------  -------------  ----------------------------
441633                  19           2              802

Application performance by connection
--------------------------------------------------------------------------------
APPLICATION_   TOTAL_CPU_TIME        TOTAL_          TOTAL_APP_     ROWS_READ +
HANDLE         per request           WAIT_TIME %     COMMITS        ROWS_MODIFIED
```

```
-------------  -------------------  -----------  -------------  -------------
180            0                    0            0              0
65711          495970               46           1              566
131323         324379               43           1              222

Application performance by service class
-------------------------------------------------------------------------------
SERVICE_       TOTAL_CPU_TIME       TOTAL_       TOTAL_APP_     ROWS_READ +
CLASS_ID       per request          WAIT_TIME %  COMMITS        ROWS_MODIFIED
--------       -------------------  -----------  -------------  -------------
11             0                    0            0              0
12             0                    0            0              0
13             440427               19           2              802

Application performance by workload
-------------------------------------------------------------------------------
WORKLOAD_      TOTAL_CPU_TIME         TOTAL_       TOTAL_APP_     ROWS_READ +
NAME           per request            WAIT_TIME %  COMMITS        ROWS_MODIFIED
-------------  ---------------------  -----------  -------------  -------------
SYSDEFAULTADM  0                      0            0              0
SYSDEFAULTUSE  410174                 45           2              788

===============================================================================
Part 3 - Member level information

- I/O wait time is
  (POOL_READ_TIME + POOL_WRITE_TIME + DIRECT_READ_TIME + DIRECT_WRITE_TIME).

        TOTAL_CPU_TIME         TOTAL_       RQSTS_COMPLETED_  I/O
MEMBER  per request            WAIT_TIME %  TOTAL             wait time
------  ---------------------  -----------  ----------------  ----------------
0       17804                  0            9                 10
1       108455                 47           14                866
2       74762                  41           13                441


267 record(s) selected.

Return Status = 0
```

LOCKWAIT procedure - Generate a report of current lock waits

The Lock Waits report contains information about each lock wait currently in progress. Details include information about the lock holder and requestor and characteristics of the lock held and the lock requested.

Syntax



Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

The following examples demonstrate various ways to call the LOCKWAIT procedure:

```
call monreport.lockwait;
call monreport.lockwait();
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Monitoring report - current lock waits
--------------------------------------------------------------------------------
Database:               SAMPLE
Generated:              08/28/2009 07:16:26

================================================================================
Part 1 - Summary of current lock waits

--------------------------------------------------------------------------------

      REQ_APPLICATION  LOCK_MODE  HLD_APPLICATION  LOCK_  LOCK_OBJECT_TYPE
#     HANDLE           REQUESTED  _HANDLE          MODE
----  ---------------  ---------  ---------------  -----  ----------------------
1     26               U          21               U      ROW
2     25               U          21               U      ROW
3     24               U          21               U      ROW
4     23               U          21               U      ROW
5     22               U          21               U      ROW
6     27               U          21               U      ROW

================================================================================

...


390 record(s) selected.

Return Status = 0
```

Figure 1. Sample MONREPORT.LOCKWAIT output - summary section

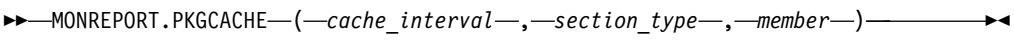
```
================================================================================
Part 2: Details for each current lock wait

lock wait #:1
--------------------------------------------------------------------------------

-- Lock details --

LOCK_NAME            = 04000500040000000000000052
LOCK_WAIT_START_TIME = 2009-08-28-07.15.31.013802
LOCK_OBJECT_TYPE     = ROW
TABSCHEMA            = TRIPATHY
TABNAME              = INVENTORY
ROWID                = 4
LOCK_STATUS          = W
LOCK_ATTRIBUTES      = 0000000000000000
ESCALATION           = N

-- Requestor and holder application details --

Attributes           Requestor                      Holder
-------------------  -----------------------------  ----------------------------
APPLICATION_HANDLE   26                             21
APPLICATION_ID       *LOCAL.tripathy.090828111531   *LOCAL.tripathy.090828111435
APPLICATION_NAME     java                           java
SESSION_AUTHID       TRIPATHY                       TRIPATHY
MEMBER               0                              0
LOCK_MODE            -                              U
LOCK_MODE_REQUESTED  U                              -

-- Lock holder current agents --

AGENT_TID            = 41
REQUEST_TYPE         = FETCH
EVENT_STATE          = IDLE
EVENT_OBJECT         = REQUEST
EVENT_TYPE           = WAIT
ACTIVITY_ID          =
UOW_ID               =

-- Lock holder current activities --

ACTIVITY_ID          = 1
UOW_ID               = 1
LOCAL_START_TIME     = 2009-08-28-07.14.31.079757
ACTIVITY_TYPE        = READ_DML
ACTIVITY_STATE       = IDLE

STMT_TEXT            =
select * from inventory for update

-- Lock requestor waiting agent and activity --

AGENT_TID            = 39
REQUEST_TYPE         = FETCH
ACTIVITY_ID          = 1
UOW_ID               = 1
LOCAL_START_TIME     = 2009-08-28-07.15.31.012935
ACTIVITY_TYPE        = READ_DML
ACTIVITY_STATE       = EXECUTING

STMT_TEXT            =
select * from inventory for update
```

Figure 2. Sample MONREPORT.LOCKWAIT output - details section

PKGCACHE procedure - Generate a summary report of package cache metrics

The Package Cache Summary report lists the top statements accumulated in the package cache as measured by various metrics.

Syntax



Parameters

cache_interval
An optional input argument of type INTEGER that specifies the report should only include data for package cache entries that have been updated in the past number of minutes specified by the *cache_interval* value. For example a *cache_interval* value of 60 produces a report based on package cache entries that have been updated in the past 60 minutes. Valid values are integers between `0` and `10080`, which supports an interval of up to 7 days. If the argument is not specified (or if null is specified), the report includes data for package cache entries regardless of when they were added or updated.

section_type
An optional input argument of type CHAR(1) that specifies whether the report should include data for static SQL, dynamic SQL, or both. If the argument is not specified (or if null is specified), the report includes data for both types of SQL. Valid values are: `d` or `D` (for dynamic) and `s` or `S` (for static).

member
An optional input argument of type SMALLINT that determines whether to show data for a particular member or partition, or to show data summed across all members. If this argument is not specified (or if null is specified), the report shows values summed across all members. If a valid member number is specified, the report shows values for that member.

Authorization

The following privilege is required:

- EXECUTE privilege on the MONREPORT module

The following examples demonstrate various ways to call the PKGCACHE procedure. The first example produces a report based on all statements in the package cache, with data aggregated across all members:

```
call monreport.pkgcache;
```

The next example produces a report based on both dynamic and static statements in the package cache for which metrics have been updated within the last 30 minutes, with data aggregated across all members:

```
call monreport.pkgcache(30);
```

The next example produces a report based on all dynamic statements in the package cache, with data aggregated across all members:

```
call monreport.pkgcache(DEFAULT, 'd');
```

The next example produces a report based on both dynamic and static statements in the package cache for which metrics have been updated within the last 30 minutes, with data specific to a member number 4:

```
call db2monreport.pkgcache(30, DEFAULT, 4);
```

UTL_DIR module

The UTL_DIR module provides a set of routines for maintaining directory aliases that are used with the UTL_FILE module.

Note: The UTL_DIR module does not issue any direct operating system calls, for example, the **mkdir** or **rmdir** commands. Maintenance of the physical directories is outside the scope of this module.

The schema for this module is SYSIBMADM.

For the SYSTOOLS.DIRECTORIES table to be created successfully in the SYSTOOLSPACE table space, ensure that you have CREATETAB authority if you are running the UTL_DIR module for the first time.

The UTL_DIR module includes the following built-in routines.

Table 21. Built-in routines available in the UTL_DIR module

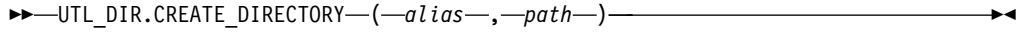
Routine name	Description
CREATE_DIRECTORY procedure	Creates a directory alias for the specified path.
CREATE_OR_REPLACE_DIRECTORY procedure	Creates or replaces a directory alias for the specified path.
DROP_DIRECTORY procedure	Drops the specified directory alias.
GET_DIRECTORY_PATH procedure	Gets the corresponding path for the specified directory alias.

CREATE_DIRECTORY procedure - Create a directory alias

The CREATE_DIRECTORY procedure creates a directory alias for the specified path.

Directory information is stored in SYSTOOLS.DIRECTORIES, which is created in the SYSTOOLSPACE when you first reference this module for each database.

Syntax



Procedure parameters

alias
An input argument of type VARCHAR(128) that specifies the directory alias.

path
An input argument of type VARCHAR(1024) that specifies the path.

Authorization

EXECUTE privilege on the UTL_DIR module.

Example

Create a directory alias, and use it in a call to the UTL_FILE.FOPEN function.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()
BEGIN
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE  isOpen          BOOLEAN;
  DECLARE  v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  CALL UTL_DIR.CREATE_DIRECTORY('mydir', '/home/user/temp/mydir');
  SET v_filehandle = UTL_FILE.FOPEN('mydir',v_filename,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL DBMS_OUTPUT.PUT_LINE('Opened file: ' || v_filename);
  CALL UTL_FILE.FCLOSE(v_filehandle);
END@

CALL proc1@
```

This example results in the following output:

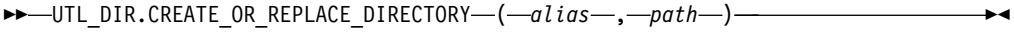
```
Opened file: myfile.csv
```

CREATE_OR_REPLACE_DIRECTORY procedure - Create or replace a directory alias

The CREATE_OR_REPLACE_DIRECTORY procedure creates or replaces a directory alias for the specified path.

Directory information is stored in SYSTOOLS.DIRECTORIES, which is created in the SYSTOOLSPACE when you first reference this module for each database.

Syntax



Procedure parameters

alias
An input argument of type VARCHAR(128) that specifies the directory alias.

path
An input argument of type VARCHAR(1024) that specifies the path.

Authorization

EXECUTE privilege on the UTL_DIR module.

Example

Example 1: Create a directory alias. Because the directory already exists, an error occurs.

```
CALL UTL_DIR.CREATE_DIRECTORY('mydir', 'home/user/temp/empdir')@
```

This example results in the following output:

```
SQL0438N  Application raised error or warning with diagnostic text: "directory
alias already defined".  SQLSTATE=23505
```

Example 2: Create or replace a directory alias.

```
CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', 'home/user/temp/empdir')@
```

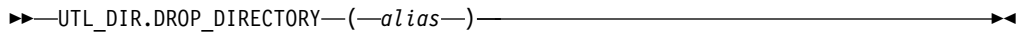
This example results in the following output:

```
  Return Status = 0
```

DROP_DIRECTORY procedure - Drop a directory alias

The DROP_DIRECTORY procedure drops the specified directory alias.

Syntax



Procedure parameters

alias
An input argument of type VARCHAR(128) that specifies the directory alias.

Authorization

EXECUTE privilege on the UTL_DIR module.

Example

Drop the specified directory alias.

```
CALL UTL_DIR.DROP_DIRECTORY('mydir')@
```

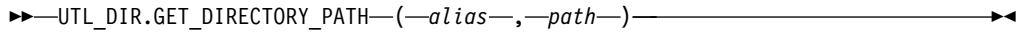
This example results in the following output:

```
  Return Status = 0
```

GET_DIRECTORY_PATH procedure - Get the path for a directory alias

The GET_DIRECTORY_PATH procedure returns the corresponding path for a directory alias.

Syntax



Procedure parameters

alias
An input argument of type VARCHAR(128) that specifies the directory alias.

path
An output argument of type VARCHAR(1024) that specifies the path that is defined for a directory alias.

Authorization

EXECUTE privilege on the UTL_DIR module.

Example

Get the path that is defined for a directory alias.

```
CALL UTL_DIR.GET_DIRECTORY_PATH('mydir', ? )@
```

This example results in the following output:

```
Value of output parameters
--------------------------
Parameter Name  : PATH
Parameter Value : home/rhoda/temp/mydir

Return Status = 0
```

UTL_FILE module

The UTL_FILE module provides a set of routines for reading from and writing to files on the database server's file system.

The schema for this module is SYSIBMADM.

The UTL_FILE module includes the following built-in routines and types.

Table 22. Built-in routines available in the UTL_FILE module

Routine name	Description
FCLOSE procedure	Closes a specified file.
FCLOSE_ALL procedure	Closes all open files.
FCOPY procedure	Copies text from one file to another.
FFLUSH procedure	Flushes unwritten data to a file
FOPEN function	Opens a file.
FREMOVE procedure	Removes a file.
FRENAME procedure	Renames a file.
GET_LINE procedure	Gets a line from a file.
IS_OPEN function	Determines whether a specified file is open.
NEW_LINE procedure	Writes an end-of-line character sequence to a file.
PUT procedure	Writes a string to a file.
PUT_LINE procedure	Writes a single line to a file.
PUTF procedure	Writes a formatted string to a file.
UTL_FILE.FILE_TYPE	Stores a file handle.

The following is a list of named conditions (these are called

```
exceptions
```

by Oracle) that an application can receive.

Table 23. Named conditions for an application

Condition Name	Description
access_denied	Access to the file is denied by the operating system.
charsetmismatch	A file was opened using FOPEN_NCHAR, but later I/O operations used non-CHAR functions such as PUTF or GET_LINE.
delete_failed	Unable to delete file.
file_open	File is already open.
internal_error	Unhandled internal error in the UTL_FILE module.
invalid_filehandle	File handle does not exist.
invalid_filename	A file with the specified name does not exist in the path.
invalid_maxlinesize	The MAX_LINESIZE value for FOPEN is invalid. It must be between 1 and 32672.
invalid_mode	The open_mode argument in FOPEN is invalid.
invalid_offset	The ABSOLUTE_OFFSET argument for FSEEK is invalid. It must be greater than 0 and less than the total number of bytes in the file.
invalid_operation	File could not be opened or operated on as requested.
invalid_path	The specified path does not exist or is not visible to the database
read_error	Unable to read the file.
rename_failed	Unable to rename the file.
write_error	Unable to write to the file.

Usage notes

To reference directories on the file system, use a directory alias. You can create a directory alias by calling the UTL_DIR.CREATE_DIRECTORY or UTL_DIR.CREATE_OR_REPLACE_DIRECTORY procedures. For example, `CALL UTL_DIR.CREATE_DIRECTORY('mydir', 'home/user/temp/mydir')@`.

The UTL_FILE module executes file operations by using the Db2 instance ID. Therefore, if you are opening a file, verify that the Db2 instance ID has the appropriate operating system permissions.

FCLOSE procedure - Close an open file

The FCLOSE procedure closes a specified file.

Syntax

Procedure parameters

file
An input or output argument of type UTL_FILE.FILE_TYPE that contains the file handle. When the file is closed, this value is set to 0.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Open a file, write some text to the file, and then close the file.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()
BEGIN
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE  isOpen          BOOLEAN;
  DECLARE  v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE  v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', '/tmp');
  SET v_filehandle = UTL_FILE.FOPEN(v_dirAlias,v_filename,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL UTL_FILE.PUT_LINE(v_filehandle,'Some text to write to the file.');
  CALL UTL_FILE.FCLOSE(v_filehandle);
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      CALL DBMS_OUTPUT.PUT_LINE('Closed file: ' || v_filename);
    END IF;
END@

CALL proc1@
```

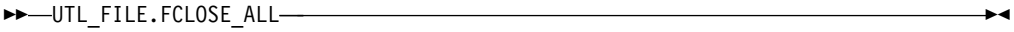
This example results in the following output:

```
Closed file: myfile.csv
```

FCLOSE_ALL procedure - Close all open files

The FCLOSE_ALL procedure closes all open files. The procedure runs successfully even if there are no open files to close.

Syntax



Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Open a couple of files, write some text to the files, and then close all open files.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()
BEGIN
```

```
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE  v_filehandle2   UTL_FILE.FILE_TYPE;
  DECLARE  isOpen          BOOLEAN;
  DECLARE  v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE  v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  DECLARE  v_filename2     VARCHAR(20) DEFAULT 'myfile2.csv';
  CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', '/tmp');
  SET v_filehandle = UTL_FILE.FOPEN(v_dirAlias,v_filename,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL UTL_FILE.PUT_LINE(v_filehandle,'Some text to write to a file.');
  SET v_filehandle2 = UTL_FILE.FOPEN(v_dirAlias,v_filename2,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle2 );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL UTL_FILE.PUT_LINE(v_filehandle2,'Some text to write to another file.');
  CALL UTL_FILE.FCLOSE_ALL;
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      CALL DBMS_OUTPUT.PUT_LINE(v_filename || ' is now closed.');
    END IF;
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle2 );
    IF isOpen != TRUE THEN
      CALL DBMS_OUTPUT.PUT_LINE(v_filename2 || ' is now closed.');
    END IF;
END@

CALL proc1@
```

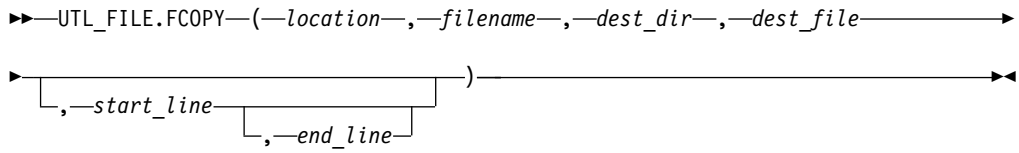
This example results in the following output:

```
myfile.csv is now closed.
myfile2.csv is now closed.
```

FCOPY procedure - Copy text from one file to another

The FCOPY procedure copies text from one file to another.

Syntax



Procedure parameters

location
An input argument of type VARCHAR(128) that specifies the alias of the directory that contains the source file.

filename
An input argument of type VARCHAR(255) that specifies the name of the source file.

dest_dir
An input argument of type VARCHAR(128) that specifies the alias of the destination directory.

dest_file
An input argument of type VARCHAR(255) that specifies the name of the destination file.

start_line
An optional input argument of type INTEGER that specifies the line number of the first line of text to copy in the source file. The default is 1.

end_line
An optional input argument of type INTEGER that specifies the line number of the last line of text to copy in the source file. If this argument is omitted or null, the procedure continues copying all text through the end of the file.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Make a copy of a file, empfile.csv, that contains a comma-delimited list of employees from the emp table.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile       UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias      VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file      VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_dest_file     VARCHAR(20) DEFAULT 'empcopy.csv';
  DECLARE    v_empline       VARCHAR(200);
  CALL UTL_FILE.FCOPY(v_dirAlias,v_src_file,v_dirAlias,v_dest_file);
END@

CALL proc1@
```

This example results in the following output:

```
  Return Status = 0
```

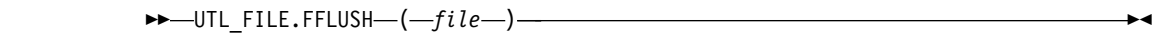
The file copy, `empcopy.csv`, contains the following data:

```
10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220
20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300
30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060
50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214
60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580
70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893
90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380
100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092
```

FFLUSH procedure - Flush unwritten data to a file

The FFLUSH procedure forces unwritten data in the write buffer to be written to a file.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Flush each line after calling the NEW_LINE procedure.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile_src    UTL_FILE.FILE_TYPE;
  DECLARE    v_empfile_tgt    UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias       VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file       VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_dest_file      VARCHAR(20) DEFAULT 'empfilenew.csv';
  DECLARE    v_empline        VARCHAR(200);
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLSTATE1 CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'SET SQLSTATE1 = SQLSTATE;

  SET v_empfile_src = UTL_FILE.FOPEN(v_dirAlias,v_src_file,'r');
  SET v_empfile_tgt = UTL_FILE.FOPEN(v_dirAlias,v_dest_file,'w');

  loop1: LOOP
    CALL UTL_FILE.GET_LINE(v_empfile_src,v_empline);
    IF SQLSTATE1 = '02000' THEN -- NO DATA FOUND
      LEAVE loop1;
    END IF;
    CALL UTL_FILE.PUT(v_empfile_tgt,v_empline);
    CALL UTL_FILE.NEW_LINE(v_empfile_tgt);
    CALL UTL_FILE.FFLUSH(v_empfile_tgt);
  END LOOP;
  CALL DBMS_OUTPUT.PUT_LINE('Updated file: ' || v_dest_file);
  CALL UTL_FILE.FCLOSE_ALL;
END@

CALL proc1@
```

This example results in the following output:

```
Updated file: empfilenew.csv
```

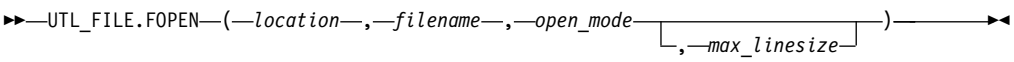
The updated file, `empfilenew.csv`, contains the following data:

```
10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220

20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300

30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060

50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214

60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580

70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893

90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380

100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092
```

FOPEN function - Open a file

The FOPEN function opens a file for I/O.

Syntax



Return value

This function returns a value of type UTL_FILE.FILE_TYPE that indicates the file handle of the opened file.

Function parameters

location
An input argument of type VARCHAR(128) that specifies the alias of the directory that contains the file.

filename
An input argument of type VARCHAR(255) that specifies the name of the file.

open_mode
An input argument of type VARCHAR(10) that specifies the mode in which the file is opened:

a append to file

r read from file

w write to file

max_linesize
An optional input argument of type INTEGER that specifies the maximum size of a line in characters. The default value is 1024 bytes. In read mode, an exception is thrown if an attempt is made to read a line that exceeds *max_linesize*. In write and append modes, an exception is thrown if an attempt is made to write a line that exceeds *max_linesize*. End-of-line character(s) do not count towards the line size.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Open a file, write some text to the file, and then close the file.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()
BEGIN
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE  isOpen          BOOLEAN;
  DECLARE  v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE  v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', '/tmp');
  SET v_filehandle = UTL_FILE.FOPEN(v_dirAlias,v_filename,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL DBMS_OUTPUT.PUT_LINE('Opened file: ' || v_filename);
  CALL UTL_FILE.PUT_LINE(v_filehandle,'Some text to write to the file.');
```

```
  CALL UTL_FILE.FCLOSE(v_filehandle);
END@

CALL proc1@
```

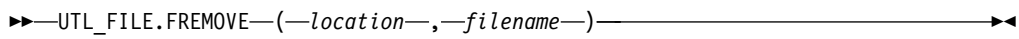
This example results in the following output.

```
Opened file: myfile.csv
```

FREMOVE procedure - Remove a file

The FREMOVE procedure removes a specified file from the system. If the file does not exist, this procedure throws an exception.

Syntax



Procedure parameters

location
An input argument of type VARCHAR(128) that specifies the alias of the directory that contains the file.

filename
An input argument of type VARCHAR(255) that specifies the name of the file.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Remove the file `myfile.csv` from the system.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  CALL UTL_FILE.FREMOVE(v_dirAlias,v_filename);
  CALL DBMS_OUTPUT.PUT_LINE('Removed file: ' || v_filename);
END@

CALL proc1@
```

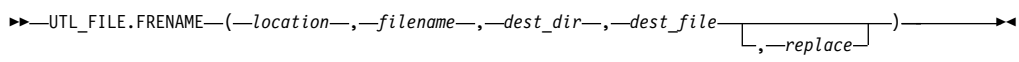
This example results in the following output:

```
Removed file: myfile.csv
```

FRENAME procedure - Rename a file

The FRENAME procedure renames a specified file. Renaming a file effectively moves a file from one location to another.

Syntax



Procedure parameters

location
An input argument of type VARCHAR(128) that specifies the alias of the directory that contains the file that you want to rename.

filename
An input argument of type VARCHAR(255) that specifies the name of the file that you want to rename.

dest_dir
An input argument of type VARCHAR(128) that specifies the alias of the destination directory.

dest_file
An input argument of type VARCHAR(255) that specifies the new name of the file.

replace
An optional input argument of type INTEGER that specifies whether to replace the file *dest_file* in the directory *dest_dir* if the file already exists:

1 Replaces existing file.

0 Throws an exception if the file already exists. This is the default if no value is specified for *replace*.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Rename a file, empfile.csv, that contains a comma-delimited list of employees from the emp table.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_dirAlias      VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file      VARCHAR(20) DEFAULT 'oldemp.csv';
  DECLARE    v_dest_file     VARCHAR(20) DEFAULT 'newemp.csv';
  DECLARE    v_replace       INTEGER DEFAULT 1;
  CALL UTL_FILE.FRENAME(v_dirAlias,v_src_file,v_dirAlias,
        v_dest_file,v_replace);
  CALL DBMS_OUTPUT.PUT_LINE('The file ' || v_src_file ||
    ' has been renamed to ' || v_dest_file);
END@

CALL proc1@
```

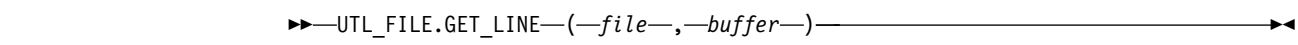
This example results in the following output:

```
The file oldemp.csv has been renamed to newemp.csv
```

GET_LINE procedure - Get a line from a file

The GET_LINE procedure gets a line of text from a specified file. The line of text does not include the end-of-line terminator. When there are no more lines to read, the procedure throws a NO_DATA_FOUND exception.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle of the opened file.

buffer
An output argument of type VARCHAR(32672) that contains a line of text from the file.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Read through and display the records in the file empfile.csv.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile       UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias      VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_filename      VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_empline       VARCHAR(200);
  DECLARE    v_count         INTEGER DEFAULT 0;
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLSTATE1 CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'SET SQLSTATE1 = SQLSTATE;

  SET v_empfile = UTL_FILE.FOPEN(v_dirAlias,v_filename,'r');

  loop1: LOOP
    CALL UTL_FILE.GET_LINE(v_empfile, v_empline);
    IF SQLSTATE1 = '02000' THEN -- NO DATA FOUND
      LEAVE loop1;
    END IF;
    CALL DBMS_OUTPUT.PUT_LINE(v_empline);
    SET v_count = v_count + 1;
  END LOOP;
  CALL DBMS_OUTPUT.PUT_LINE('End of file ' || v_filename || ' - ' || v_count
        || ' records retrieved');
  CALL UTL_FILE.FCLOSE(v_empfile);
END@

CALL proc1@
```

This example results in the following output:

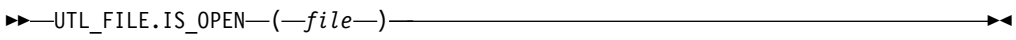
```
10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220

20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300

30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060

50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214

60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580
```

```
70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893

90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380

100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092

End of file empfile.csv - 8 records retrieved
```

IS_OPEN function - Determine whether a specified file is open

The IS_OPEN function determines whether a specified file is open.

Syntax



Return value

This function returns a value of type BOOLEAN that indicates if the specified file is open (TRUE) or closed (FALSE).

Function parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

The following example demonstrates that before writing text to a file, you can call the IS_OPEN function to check if the file is open.

```
SET SERVEROUTPUT ON@

CREATE OR REPLACE PROCEDURE proc1()
BEGIN
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE  isOpen          BOOLEAN;
  DECLARE  v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE  v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', '/tmp');
  SET v_filehandle = UTL_FILE.FOPEN(v_dirAlias,v_filename,'w');
  SET isOpen = UTL_FILE.IS_OPEN( v_filehandle );
    IF isOpen != TRUE THEN
      RETURN -1;
    END IF;
  CALL UTL_FILE.PUT_LINE(v_filehandle,'Some text to write to the file.');
  CALL DBMS_OUTPUT.PUT_LINE('Updated file: ' || v_filename);
  CALL UTL_FILE.FCLOSE(v_filehandle);
END@

CALL proc1@
```

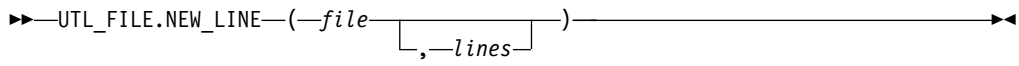
This example results in the following output.

```
Updated file: myfile.csv
```

NEW_LINE procedure - Write an end-of-line character sequence to a file

The NEW_LINE procedure writes an end-of-line character sequence to a specified file.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle.

lines
An optional input argument of type INTEGER that specifies the number of end-of-line character sequences to write to the file. The default is 1.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Write a file that contains a triple-spaced list of employee records.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile_src    UTL_FILE.FILE_TYPE;
  DECLARE    v_empfile_tgt    UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias       VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file       VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_dest_file      VARCHAR(20) DEFAULT 'empfilenew.csv';
  DECLARE    v_empline        VARCHAR(200);
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLSTATE1 CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'SET SQLSTATE1 = SQLSTATE;

  SET v_empfile_src = UTL_FILE.FOPEN(v_dirAlias,v_src_file,'r');
  SET v_empfile_tgt = UTL_FILE.FOPEN(v_dirAlias,v_dest_file,'w');

  loop1: LOOP
    CALL UTL_FILE.GET_LINE(v_empfile_src,v_empline);
    IF SQLSTATE1 = '02000' THEN -- NO DATA FOUND
      LEAVE loop1;
    END IF;
    CALL UTL_FILE.PUT(v_empfile_tgt,v_empline);
    CALL UTL_FILE.NEW_LINE(v_empfile_tgt, 2);
  END LOOP;

  CALL DBMS_OUTPUT.PUT_LINE('Wrote to file: ' || v_dest_file);
  CALL UTL_FILE.FCLOSE_ALL;
END@

CALL proc1@
```

This example results in the following output:

```
Wrote to file: empfilenew.csv
```

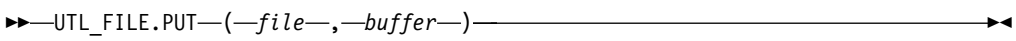
The file that is updated, `empfilenew.csv`, contains the following data:

```
10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220

20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300

30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060

50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214

60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580

70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893

90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380

100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092
```

PUT procedure - Write a string to a file

The PUT procedure writes a string to a specified file. No end-of-line character sequence is written at the end of the string.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle.

buffer
An input argument of type VARCHAR(32672) that specifies the text to write to the file.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Use the PUT procedure to add a string to a file and then use the NEW_LINE procedure to add an end-of-line character sequence.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile_src    UTL_FILE.FILE_TYPE;
  DECLARE    v_empfile_tgt    UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias       VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file       VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_dest_file      VARCHAR(20) DEFAULT 'empfilenew.csv';
```

```
  DECLARE    v_empline        VARCHAR(200);
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLSTATE1 CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'SET SQLSTATE1 = SQLSTATE;

  SET v_empfile_src = UTL_FILE.FOPEN(v_dirAlias,v_src_file,'r');
  SET v_empfile_tgt = UTL_FILE.FOPEN(v_dirAlias,v_dest_file,'w');

  loop1: LOOP
    CALL UTL_FILE.GET_LINE(v_empfile_src,v_empline);
    IF SQLSTATE1 = '02000' THEN -- NO DATA FOUND
      LEAVE loop1;
    END IF;
    CALL UTL_FILE.PUT(v_empfile_tgt,v_empline);
    CALL UTL_FILE.NEW_LINE(v_empfile_tgt, 2);
  END LOOP;

  CALL DBMS_OUTPUT.PUT_LINE('Wrote to file: ' || v_dest_file);
  CALL UTL_FILE.FCLOSE_ALL;
END@

CALL proc1@
```

This example results in the following output:

```
Wrote to file: empfilenew.csv
```

The updated file, `empfilenew.csv`, contains the following data:

```
10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220


20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300


30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060


50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214


60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580


70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893


90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380


100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092
```

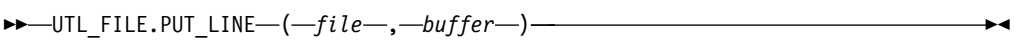
Usage notes

After using the PUT procedure to add a string to a file, use the NEW_LINE procedure to add an end-of-line character sequence to the file.

PUT_LINE procedure - Write a line of text to a file

The PUT_LINE procedure writes a line of text, including an end-of-line character sequence, to a specified file.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle of file to which the line is to be written.

buffer
An input argument of type VARCHAR(32672) that specifies the text to write to the file.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Use the PUT_LINE procedure to write lines of text to a file.

```
CALL proc1@
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE    v_empfile_src    UTL_FILE.FILE_TYPE;
  DECLARE    v_empfile_tgt    UTL_FILE.FILE_TYPE;
  DECLARE    v_dirAlias       VARCHAR(50) DEFAULT 'empdir';
  DECLARE    v_src_file       VARCHAR(20) DEFAULT 'empfile.csv';
  DECLARE    v_dest_file      VARCHAR(20) DEFAULT 'empfilenew2.csv';
  DECLARE    v_empline        VARCHAR(200);
  DECLARE    v_count          INTEGER DEFAULT 0;
  DECLARE SQLCODE INTEGER DEFAULT 0;
  DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
  DECLARE SQLSTATE1 CHAR(5) DEFAULT '00000';
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000'SET SQLSTATE1 = SQLSTATE;

  SET v_empfile_src = UTL_FILE.FOPEN(v_dirAlias,v_src_file,'r');
  SET v_empfile_tgt = UTL_FILE.FOPEN(v_dirAlias,v_dest_file,'w');

  loop1: LOOP
    CALL UTL_FILE.GET_LINE(v_empfile_src,v_empline);
    IF SQLSTATE1 = '02000' THEN -- NO DATA FOUND
      LEAVE loop1;
    END IF;
    SET v_count = v_count + 1;
    CALL UTL_FILE.PUT(v_empfile_tgt,'Record ' || v_count || ': ');
    CALL UTL_FILE.PUT_LINE(v_empfile_tgt,v_empline);
  END LOOP;
  CALL DBMS_OUTPUT.PUT_LINE('End of file ' || v_src_file || ' - ' || v_count
    || ' records retrieved');
  CALL UTL_FILE.FCLOSE_ALL;
END@

CALL proc1@
```

This example results in the following output:

```
End of file empfile.csv - 8 records retrieved
```

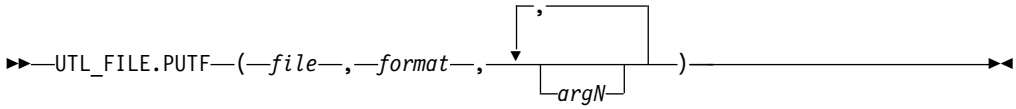
The file that is updated, `empfilenew2.csv`, contains the following data:

```
Record 1: 10,CHRISTINE,I,HAAS,A00,3978,1/1/1965,PRES,18,F,8/24/1933,52750,1000,4220

Record 2: 20,MICHAEL,L,THOMPSON,B01,3476,10/10/1973,MANAGER,18,M,2/2/1948,41250,800,3300

Record 3: 30,SALLY,A,KWAN,C01,4738,4/5/1975,MANAGER,20,F,5/11/1941,38250,800,3060

Record 4: 50,JOHN,B,GEYER,E01,6789,8/17/1949,MANAGER,16,M,9/15/1925,40175,800,3214

Record 5: 60,IRVING,F,STERN,D11,6423,9/14/1973,MANAGER,16,M,7/7/1945,32250,500,2580

Record 6: 70,EVA,D,PULASKI,D21,7831,9/30/1980,MANAGER,16,F,5/26/1953,36170,700,2893

Record 7: 90,EILEEN,W,HENDERSON,E11,5498,8/15/1970,MANAGER,16,F,5/15/1941,29750,600,2380

Record 8: 100,THEODORE,Q,SPENSER,E21,972,6/19/1980,MANAGER,14,M,12/18/1956,26150,500,2092
```

PUTF procedure - Write a formatted string to a file

The PUTF procedure writes a formatted string to a specified file.

Syntax



Procedure parameters

file
An input argument of type UTL_FILE.FILE_TYPE that contains the file handle.

format
An input argument of type VARCHAR(1024) the specifies the string to use for formatting the text. The special character sequence, %s, is substituted by the value of *argN*. The special character sequence, \n, indicates a new line.

argN
An optional input argument of type VARCHAR(1024) that specifies a value to substitute in the format string for the corresponding occurrence of the special character sequence %s. Up to five arguments, *arg1* through *arg5*, can be specified. *arg1* is substituted for the first occurrence of %s, *arg2* is substituted for the second occurrence of %s, and so on.

Authorization

EXECUTE privilege on the UTL_FILE module.

Example

Format employee data.

```
SET SERVEROUTPUT ON@

CREATE PROCEDURE proc1()
BEGIN
  DECLARE v_filehandle    UTL_FILE.FILE_TYPE;
  DECLARE v_dirAlias      VARCHAR(50) DEFAULT 'mydir';
  DECLARE v_filename      VARCHAR(20) DEFAULT 'myfile.csv';
  DECLARE v_format        VARCHAR(200);
  SET v_format = '%s %s, %s\nSalary: $%s Commission: $%s\n\n';
  CALL UTL_DIR.CREATE_OR_REPLACE_DIRECTORY('mydir', '/tmp');
  SET v_filehandle = UTL_FILE.FOPEN(v_dirAlias,v_filename,'w');
  CALL UTL_FILE.PUTF(v_filehandle,v_format,'000030','SALLY','KWAN','40175','3214');
```

```
  CALL DBMS_OUTPUT.PUT_LINE('Wrote to file: ' || v_filename);
  CALL UTL_FILE.FCLOSE(v_filehandle);
END@

CALL proc1@
```

This example results in the following output:

```
Wrote formatted text to file: myfile.csv
```

The formatted file, `myfile.csv`, contains the following data:

```
000030 SALLY, KWAN
Salary: $40175 Commission: $3214
```

UTL_FILE.FILE_TYPE

UTL_FILE.FILE_TYPE is a file handle type that is used by routines in the UTL_FILE module.

Example

Declare a variable of type UTL_FILE.FILE_TYPE.

```
  DECLARE  v_filehandle    UTL_FILE.FILE_TYPE;
```

UTL_MAIL module

The UTL_MAIL module provides the capability to send e-mail.

The schema for this module is SYSIBMADM.

The UTL_MAIL module includes the following routines.

Table 24. Built-in routines available in the UTL_MAIL module

Routine name	Description
SEND procedure	Packages and sends an e-mail to an SMTP server.
SEND_ATTACH_RAW procedure	Same as the SEND procedure, but with BLOB attachments.
SEND_ATTACH_VARCHAR2	Same as the SEND procedure, but with VARCHAR attachments

Usage notes

In order to successfully send an e-mail using the UTL_MAIL module, the database configuration parameter SMTP_SERVER must contain one or more valid SMTP server addresses.

Examples

Example 1: To set up a single SMTP server with the default port 25:

```
db2 update db cfg using smtp_server 'smtp.ibm.com'
```

Example 2: To set up a single SMTP server that uses port 2000, rather than the default port 25:

```
db2 update db cfg using smtp_server 'smtp2.ibm.com:2000'
```

Example 3: To set a list of SMTP servers:

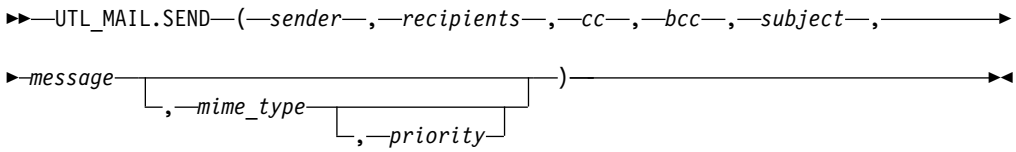
```
db2 update db cfg using smtp_server
   'smtp.example.com,smtp1.example.com:23,smtp2.example.com:2000'
```

Note: The e-mail is sent to each of the SMTP servers, in the order listed, until a successful reply is received from one of the SMTP servers.

SEND procedure - Send an e-mail to an SMTP server

The SEND procedure provides the capability to send an e-mail to an SMTP server.

Syntax



Parameters

sender
An input argument of type VARCHAR(256) that specifies the e-mail address of the sender.

recipients
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of the recipients.

cc An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of copy recipients.

bcc
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of blind copy recipients.

subject
An input argument of type VARCHAR(32672) that specifies the subject line of the e-mail.

message
An input argument of type VARCHAR(32672) that specifies the body of the e-mail.

mime_type
An optional input argument of type VARCHAR(1024) that specifies the MIME type of the message. The default is `'text/plain; charset=us-ascii'`.

priority
An optional argument of type INTEGER that specifies the priority of the e-mail The default value is 3.

Authorization

EXECUTE privilege on the UTL_MAIL module.

Examples

Example 1: The following anonymous block sends a simple e-mail message.

```
BEGIN
  DECLARE v_sender VARCHAR(30);
  DECLARE v_recipients VARCHAR(60);
  DECLARE v_subj VARCHAR(20);
  DECLARE v_msg VARCHAR(200);

  SET v_sender = 'kkent@mycorp.com';
  SET v_recipients = 'bwayne@mycorp.com,pparker@mycorp.com';
  SET v_subj = 'Holiday Party';
  SET v_msg = 'This year''s party is scheduled for Friday, Dec. 21 at ' ||
     '6:00 PM. Please RSVP by Dec. 15th.';
  CALL UTL_MAIL.SEND(v_sender, v_recipients, NULL, NULL, v_subj, v_msg);
END@
```

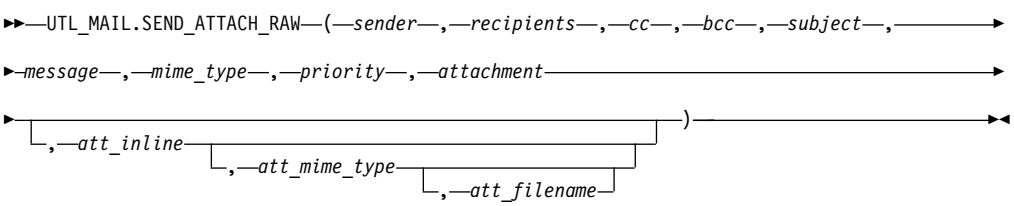
This example results in the following output:

```
BEGIN
  DECLARE v_sender VARCHAR(30);
  DECLARE v_recipients VARCHAR(60);
  DECLARE v_subj VARCHAR(20);
  DECLARE v_msg VARCHAR(200);

  SET v_sender = 'kkent@mycorp.com';
  SET v_recipients = 'bwayne@mycorp.com,pparker@mycorp.com';
  SET v_subj = 'Holiday Party';
  SET v_msg = 'This year''s party is scheduled for Friday, Dec. 21 at ' ||
     '6:00 PM. Please RSVP by Dec. 15th.';
  CALL UTL_MAIL.SEND(v_sender, v_recipients, NULL, NULL, v_subj, v_msg);
END
DB20000I  The SQL command completed successfully.
```

SEND_ATTACH_RAW procedure - Send an e-mail with a BLOB attachment to an SMTP server

The SEND_ATTACH_RAW procedure provides the capability to send an e-mail to an SMTP server with a binary attachment.

Syntax



Parameters

sender
An input argument of type VARCHAR(256) that specifies the e-mail address of the sender.

recipients
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of the recipients.

cc An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of copy recipients.

bcc
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of blind copy recipients.

subject
An input argument of type VARCHAR(32672) that specifies the subject line of the e-mail.

message
An input argument of type VARCHAR(32672) that specifies the body of the e-mail.

mime_type
An input argument of type VARCHAR(1024) that specifies the MIME type of the message. The default is `'text/plain; charset=us-ascii'`.

priority
An input argument of type INTEGER that specifies the priority of the e-mail The default value is 3.

attachment
An input argument of type BLOB(10M) that contains the attachment.

att_inline
An optional input argument of type BOOLEAN that specifies whether the attachment is viewable inline. If set to "true", then the attachment is viewable inline, "false" otherwise. The default value is "true".

att_mime_type
An optional input argument of type VARCHAR(1024) that specifies the MIME type of the attachment. The default value is `application/octet`.

att_filename
An optional input argument of type VARCHAR(512) that specifies the file name containing the attachment. The default value is NULL.

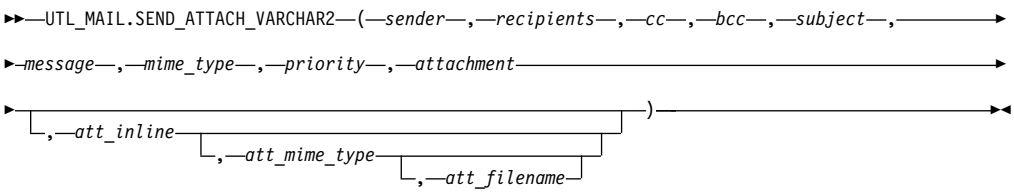
Authorization

EXECUTE privilege on the UTL_MAIL module.

SEND_ATTACH_VARCHAR2 procedure - Send an e-mail with a VARCHAR attachment to an SMTP server

The SEND_ATTACH_VARCHAR2 procedure provides the capability to send an e-mail to an SMTP server with a text attachment.

Syntax



Parameters

sender
An input argument of type VARCHAR(256) that specifies the e-mail address of the sender.

recipients
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of the recipients.

cc An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of copy recipients.

bcc
An input argument of type VARCHAR(32672) that specifies the comma-separated e-mail addresses of blind copy recipients.

subject
An input argument of type VARCHAR(32672) that specifies the subject line of the e-mail.

message
An input argument of type VARCHAR(32672) that specifies the body of the e-mail.

mime_type
An input argument of type VARCHAR(1024) that specifies the MIME type of the message. The default is `'text/plain; charset=us-ascii'`.

priority
An input argument of type INTEGER that specifies the priority of the e-mail The default value is 3.

attachment
An input argument of type VARCHAR(32000) that contains the attachment.

att_inline
An optional input argument of type BOOLEAN that specifies whether the attachment is viewable inline. If set to "true", then the attachment is viewable inline, "false" otherwise. The default value is "true".

att_mime_type
An optional input argument of type VARCHAR(1024) that specifies the MIME type of the attachment. The default value is `'text/plain; charset=us-ascii'`.

att_filename
An optional input argument of type VARCHAR(512) that specifies the file name containing the attachment. The default value is NULL.

Authorization

EXECUTE privilege on the UTL_MAIL module.

UTL_SMTP module

The UTL_SMTP module provides the capability to send e-mail over the Simple Mail Transfer Protocol (SMTP).

The UTL_SMTP module includes the following routines.

Table 25. Built-in routines available in the UTL_SMTP module

Routine Name	Description
CLOSE_DATA procedure	Ends an e-mail message.
COMMAND procedure	Execute an SMTP command.
COMMAND_REPLIES procedure	Execute an SMTP command where multiple reply lines are expected.
DATA procedure	Specify the body of an e-mail message.
EHLO procedure	Perform initial handshaking with an SMTP server and return extended information.

Table 25. Built-in routines available in the UTL_SMTP module (continued)

Routine Name	Description
HELO procedure	Perform initial handshaking with an SMTP server.
HELP procedure	Send the HELP command.
MAIL procedure	Start a mail transaction.
NOOP procedure	Send the null command.
OPEN_CONNECTION function	Open a connection.
OPEN_CONNECTION procedure	Open a connection.
OPEN_DATA procedure	Send the DATA command.
QUIT procedure	Terminate the SMTP session and disconnect.
RCPT procedure	Specify the recipient of an e-mail message.
RSET procedure	Terminate the current mail transaction.
VRFY procedure	Validate an e-mail address.
WRITE_DATA procedure	Write a portion of the e-mail message.
WRITE_RAW_DATA procedure	Write a portion of the e-mail message consisting of RAW data.

The following table lists the public variables available in the module.

Table 26. Built-in types available in the UTL_SMTP module

Public variable	Data type	Description
connection	RECORD	Description of an SMTP connection.
reply	RECORD	SMTP reply line.

The CONNECTION record type provides a description of an SMTP connection.

```
ALTER MODULE SYSIBMADM.UTL_SMTP PUBLISH TYPE connection AS ROW
(
  /* name or IP address of the remote host running SMTP server */
    host VARCHAR(255),
  /* SMTP server port number */
    port INTEGER,
  /* transfer timeout in seconds */
    tx_timeout INTEGER,
);
```

The REPLY record type provides a description of an SMTP reply line. REPLIES is an array of SMTP reply lines.

```
ALTER MODULE SYSIBMADM.UTL_SMTP PUBLISH TYPE reply AS ROW
(
  /* 3 digit reply code received from the SMTP server */
    code INTEGER,
  /* the text of the message received from the SMTP server */
    text VARCHAR(508)
);
```

Examples

Example 1: The following procedure constructs and sends a text e-mail message using the UTL_SMTP module.

```
CREATE OR REPLACE PROCEDURE send_mail(
IN p_sender VARCHAR(4096),
IN p_recipient VARCHAR(4096),
IN p_subj VARCHAR(4096),
IN p_msg VARCHAR(4096),
IN p_mailhost VARCHAR(4096))
SPECIFIC send_mail
LANGUAGE SQL
BEGIN
  DECLARE v_conn UTL_SMTP.CONNECTION;
  DECLARE v_crlf VARCHAR(2);
  DECLARE v_port INTEGER CONSTANT 25;

  SET v_crlf = CHR(13) || CHR(10);
  SET v_conn = UTL_SMTP.OPEN_CONNECTION(p_mailhost, v_port, 10);
  CALL UTL_SMTP.HELO(v_conn, p_mailhost);
  CALL UTL_SMTP.MAIL(v_conn, p_sender);
  CALL UTL_SMTP.RCPT(v_conn, p_recipient);
  CALL UTL_SMTP.DATA(
    v_conn,
    'Date: ' || TO_CHAR(SYSDATE, 'Dy, DD Mon YYYY HH24:MI:SS') || v_crlf ||
    'From: ' || p_sender || v_crlf ||
    'To: ' || p_recipient || v_crlf ||
    'Subject: ' || p_subj || v_crlf ||
    p_msg);
  CALL UTL_SMTP.QUIT(v_conn);
END@

CALL send_mail('bwayne@mycorp.com','pparker@mycorp.com','Holiday Party',
'Are you planning to attend?','smtp.mycorp.com')@
```

Example 2: The following example uses the OPEN_DATA, WRITE_DATA, and CLOSE_DATA procedures instead of the DATA procedure.

```
CREATE OR REPLACE PROCEDURE send_mail_2(
IN p_sender VARCHAR(4096),
IN p_recipient VARCHAR(4096),
IN p_subj VARCHAR(4096),
IN p_msg VARCHAR(4096),
IN p_mailhost VARCHAR(4096)) SPECIFIC send_mail_2
LANGUAGE SQL
BEGIN
  DECLARE v_conn UTL_SMTP.CONNECTION;
  DECLARE v_crlf VARCHAR(2);
  DECLARE v_port INTEGER CONSTANT 25;

  SET v_crlf = CHR(13) || CHR(10);
  SET v_conn = UTL_SMTP.OPEN_CONNECTION(p_mailhost, v_port, 10);
  CALL UTL_SMTP.HELO(v_conn, p_mailhost);
  CALL UTL_SMTP.MAIL(v_conn, p_sender);
  CALL UTL_SMTP.RCPT(v_conn, p_recipient);
  CALL UTL_SMTP.OPEN_DATA(v_conn);
  CALL UTL_SMTP.WRITE_DATA(v_conn, 'From: ' || p_sender || v_crlf);
  CALL UTL_SMTP.WRITE_DATA(v_conn, 'To: ' || p_recipient || v_crlf);
  CALL UTL_SMTP.WRITE_DATA(v_conn, 'Subject: ' || p_subj || v_crlf);
  CALL UTL_SMTP.WRITE_DATA(v_conn, v_crlf || p_msg);
  CALL UTL_SMTP.CLOSE_DATA(v_conn);
  CALL UTL_SMTP.QUIT(v_conn);
END@

CALL send_mail_2('bwayne@mycorp.com','pparker@mycorp.com','Holiday Party',
'Are you planning to attend?','smtp.mycorp.com')@
```

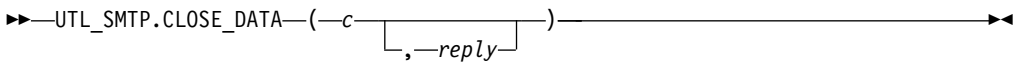
CLOSE_DATA procedure - End an e-mail message

The CLOSE_DATA procedure terminates an e-mail message

The procedure terminates an e-mail message by sending the following sequence: <CR><LF>.<CR><LF>

This is a single period at the beginning of a line.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to be closed.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

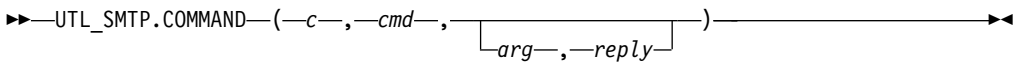
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

COMMAND procedure - Run an SMTP command

The COMMAND procedure provides the capability to execute an SMTP command.

Note: Use COMMAND_REPLIES if multiple reply lines are expected to be returned.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to which the command is to be sent.

cmd
An input argument of type VARCHAR(510) that specifies the SMTP command to process.

arg
An optional input argument of type VARCHAR(32672) that specifies an argument to the SMTP command. The default is NULL.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

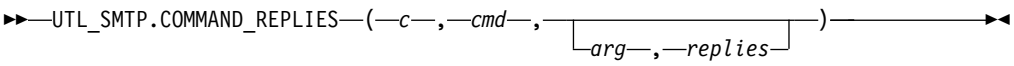
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

COMMAND_REPLIES procedure - Run an SMTP command where multiple reply lines are expected

The COMMAND_REPLIES function processes an SMTP command that returns multiple reply lines.

Note: Use COMMAND if only a single reply line is expected.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to which the command is to be sent.

cmd
An input argument of type VARCHAR(510) that specifies the SMTP command to process.

arg
An optional input argument of type VARCHAR(32672) that specifies an argument to the SMTP command. The default is NULL.

replies
An optional output argument of type REPLIES that returns multiple reply lines from the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

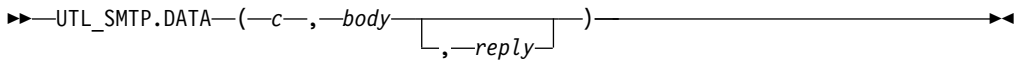
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

DATA procedure - Specify the body of an e-mail message

The DATA procedure provides the capability to specify the body of the e-mail message.

The message is terminated with a <CR><LF>.<CR><LF> sequence.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to which the command is to be sent.

body
An input argument of type VARCHAR(32000) that specifies the body of the e-mail message to be sent.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

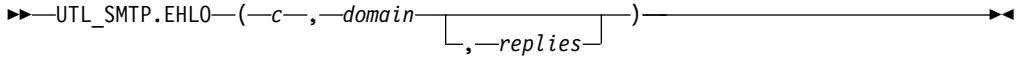
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

EHLO procedure - Perform initial handshaking with an SMTP server and return extended information

The EHLO procedure performs initial handshaking with the SMTP server after establishing the connection.

The EHLO procedure allows the client to identify itself to the SMTP server. The HELO procedure performs the equivalent functionality, but returns less information about the server.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the connection to the SMTP server over which to perform handshaking.

domain
An input argument of type VARCHAR(255) that specifies the domain name of the sending host.

replies
An optional output argument of type REPLIES that return multiple reply lines from the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

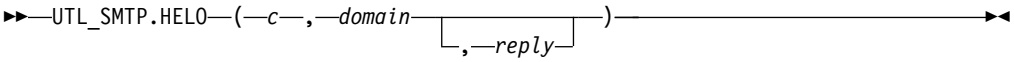
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

HELO procedure - Perform initial handshaking with an SMTP server

The HELO procedure performs initial handshaking with the SMTP server after establishing the connection.

The HELO procedure allows the client to identify itself to the SMTP server. The EHLO procedure performs the equivalent functionality, but returns more information about the server.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the connection to the SMTP server over which to perform handshaking.

domain
An input argument of type VARCHAR(255) that specifies the domain name of the sending host.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

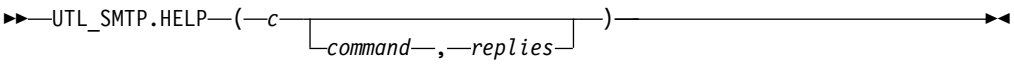
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

HELP procedure - Send the HELP command

The HELP function provides the capability to send the HELP command to the SMTP server.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to which the command is to be sent.

command
An optional input argument of type VARCHAR(510) that specifies the command about which help is requested.

replies
An optional output argument of type REPLIES that returns multiple reply lines from the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

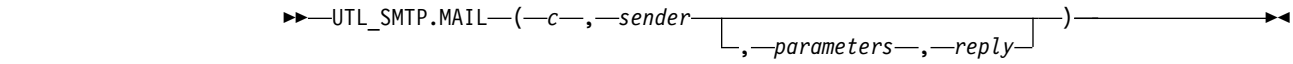
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

MAIL procedure - Start a mail transaction

Start the MAIL procedure by using the function invocation syntax in a PL/SQL assignment statement.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the connection to the SMTP server on which to start a mail transaction.

sender
An input argument of type VARCHAR(256) that specifies the e-mail address of the sender.

parameters
An optional input argument of type VARCHAR(32672) that specifies the optional mail command parameters in the format `key=value`.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

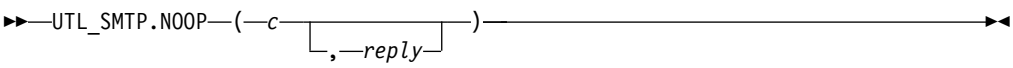
Authorization

EXECUTE privilege on the UTL_SMTP module.

NOOP procedure - Send the null command

The NOOP procedure sends a null command to the SMTP server. The NOOP has no effect on the server except to obtain a successful response.

Syntax



Parameters

c
An input or output argument of type CONNECTION that specifies the SMTP connection on which to send the command.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

OPEN_CONNECTION function - Return a connection handle to an SMTP server

The OPEN_CONNECTION function returns a connection handle to an SMTP server.

The function returns a connection handle to the SMTP server.

Syntax

Parameters

host
An input argument of type VARCHAR(255) that specifies the name of the SMTP server.

port
An input argument of type INTEGER that specifies the port number on which the SMTP server is listening.

tx_timeout
An input argument of type INTEGER that specifies the time out value, in seconds. To instruct the procedure not to wait, set this value to 0. To instruct the procedure to wait indefinitely, set this value to NULL.

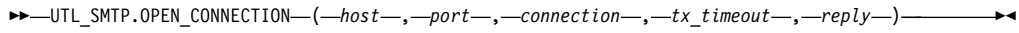
Authorization

EXECUTE privilege on the UTL_SMTP module.

OPEN_CONNECTION procedure - Open a connection to an SMTP server

The OPEN_CONNECTION procedure opens a connection to an SMTP server.

Syntax



Parameters

host
An input argument of type VARCHAR(255) that specifies the name of the SMTP server.

port
An input argument of type INTEGER that specifies the port number on which the SMTP server is listening.

connection
An output argument of type CONNECTION that returns a connection handle to the SMTP server.

tx_timeout
An optional input argument of type INTEGER that specifies the time out value, in seconds. To instruct the procedure not to wait, set this value to 0. To instruct the procedure to wait indefinitely, set this value to NULL.

reply
An output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

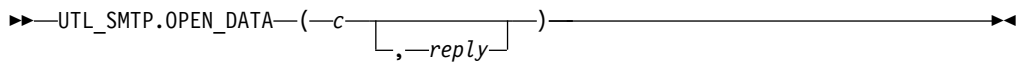
Authorization

EXECUTE privilege on the UTL_SMTP module.

OPEN_DATA procedure - Send the DATA command to the SMTP server

The OPEN_DATA procedure sends the DATA command to the SMTP server.

Syntax



Parameters

c An input argument of type CONNECTION that specifies the SMTP connection on which to send the command

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

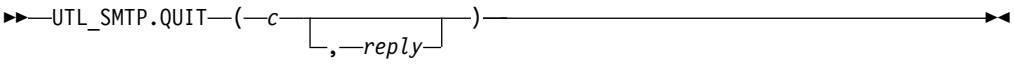
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

QUIT procedure - Close the session with the SMTP server

The QUIT procedure closes the session with an SMTP server.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection to terminate.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Usage notes

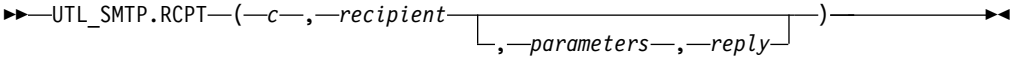
This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

RCPT procedure - Provide the e-mail address of the recipient

The RCPT procedure provides the e-mail address of the recipient.

Note: To schedule multiple recipients, invoke the RCPT procedure multiple times.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection on which to add a recipient.

recipient
An input argument of type VARCHAR(256) that specifies the e-mail address of the recipient.

parameters
An optional input argument of type VARCHAR(32672) that specifies the mail command parameters in the format `key=value`.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

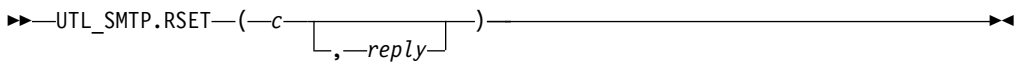
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

RSET procedure - End the current mail transaction

The RSET procedure provides the capability to terminate the current mail transaction.

Syntax



Parameters

c
An input or output argument of type CONNECTION that specifies the SMTP connection on which to cancel the mail transaction.

reply
An optional output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

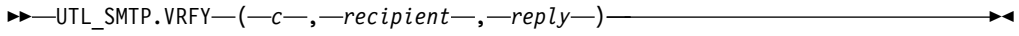
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

VRFY procedure - Validate and verify the recipient's e-mail address

The VRFY function provides the capability to validate and verify a recipient e-mail address. If valid, the recipient's full name and fully qualified mailbox is returned.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection on which to verify the e-mail address.

recipient
An input argument of type VARCHAR(256) that specifies the e-mail address to be verified.

reply
An output argument of type REPLY that returns a single reply line from the SMTP server. It is the last reply line if multiple reply lines are returned by the SMTP server.

Authorization

EXECUTE privilege on the UTL_SMTP module.

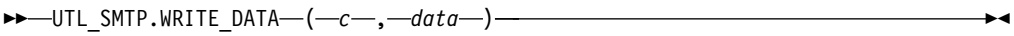
Usage notes

This procedure can be invoked using function invocation syntax in a PL/SQL assignment statement.

WRITE_DATA procedure - Write a portion of an e-mail message

The WRITE_DATA procedure provides the capability to add data to an e-mail message. The WRITE_DATA procedure may be called repeatedly to add data.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection on which to add data.

data
An input argument of type VARCHAR(32000) that specifies the data to be added to the e-mail message.

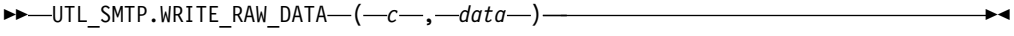
Authorization

EXECUTE privilege on the UTL_SMTP module.

WRITE_RAW_DATA procedure - Add RAW data to an e-mail message

The WRITE_RAW_DATA procedure provides the capability to add data to an e-mail message. The WRITE_RAW_DATA procedure may be called repeatedly to add data.

Syntax



Parameters

c An input or output argument of type CONNECTION that specifies the SMTP connection on which to add data.

data
An input argument of type BLOB(15M) that specifies the data to be added to the e-mail message.

Authorization

EXECUTE privilege on the UTL_SMTP module.

Index

A

ANALYZE_DATABASE procedure 103
ANALYZE_PART_OBJECT procedure 104
ANALYZE_SCHEMA procedure 105
APPEND procedure 21

B

BIND_VARIABLE_BLOB procedure 61
BIND_VARIABLE_CHAR procedure 62
BIND_VARIABLE_CLOB procedure 62
BIND_VARIABLE_DATE procedure 63
BIND_VARIABLE_DOUBLE procedure 63
BIND_VARIABLE_INT procedure 64
BIND_VARIABLE_NUMBER procedure 64
BIND_VARIABLE_RAW procedure 65
BIND_VARIABLE_TIMESTAMP procedure 65
BIND_VARIABLE_VARCHAR procedure 66
BROKEN procedure 14

C

CANONICALIZE procedure 106
CHANGE procedure 15
CLOSE procedures 22
CLOSE_CURSOR procedure 66
CLOSE_DATA procedure 177
COLUMN_VALUE_BLOB procedure 67
COLUMN_VALUE_CHAR procedure 67
COLUMN_VALUE_CLOB procedure 68
COLUMN_VALUE_DATE procedure 68
COLUMN_VALUE_DOUBLE procedure 69
COLUMN_VALUE_INT procedure 70
COLUMN_VALUE_LONG procedure 70
COLUMN_VALUE_NUMBER procedure 71
COLUMN_VALUE_RAW procedure 72
COLUMN_VALUE_TIMESTAMP procedure 73
COLUMN_VALUE_VARCHAR procedure 73
COMMA_TO_TABLE procedure 108
COMMAND procedure 177
COMMAND_REPLIES procedure 178
COMPARE function 22
COMPILE_SCHEMA procedure 110
CONNECTION procedure 130
CONVERTTOBLOB procedure 23
CONVERTTOCLOB procedure 24
COPY procedure 25
CREATE_DIRECTORY procedure 151
CREATE_OR_REPLACE_DIRECTORY procedure 152
CREATE_PIPE function 41
CREATE_WRAPPED procedure 10
CURRENTAPPS procedure 141
CURRENTSQL procedure 142

D

DATA procedure 179
data types
  FILE_TYPE 170
DB_VERSION procedure 110
DBMS_ALERT module 1
DBMS_DDL module 9
DBMS_JOB module
  BROKEN procedure 14
  CHANGE procedure 15
  INTERVAL procedure 16
  NEXT_DATE procedure 16
  overview 12
  REMOVE procedure 17
  RUN procedure 18
  SUBMIT procedure 18
  WHAT procedure 19
DBMS_LOB module
  APPEND procedures 21
  CLOSE procedures 22
  COMPARE function 22
  CONVERTTOBLOB procedure 23
  CONVERTTOCLOB procedure 24
  COPY procedures 25
  ERASE procedures 26
  GET_STORAGE_LIMIT function 26
  GETLENGTH function 27
  INSTR function 27
  ISOPEN function 28
  OPEN procedures 28
  overview 20
  READ procedures 29
  SUBSTR function 29
  TRIM procedures 30
  WRITE procedures 30
  WRITEAPPEND procedures 31
DBMS_OUTPUT module 32
DBMS_PIPE module 39
DBMS_RANDOM module 54
  INITIALIZE procedure 56
  NORMAL function 58
  RANDOM function 56
  SEED procedure 55
  SEED_STRING procedure 55
  STRING function 57
  TERMINATE procedure 56
  VALUE function 56
DBMS_SQL module
  BIND_VARIABLE_BLOB procedure 61
  BIND_VARIABLE_CHAR procedure 62
  BIND_VARIABLE_CLOB procedure 62
  BIND_VARIABLE_DATE procedure 63
DBMS_SQL module *(continued)*
  BIND_VARIABLE_DOUBLE procedure 63
  BIND_VARIABLE_INT procedure 64
  BIND_VARIABLE_NUMBER procedure 64
  BIND_VARIABLE_RAW procedure 65
  BIND_VARIABLE_TIMESTAMP procedure 65
  BIND_VARIABLE_VARCHAR procedure 66
  CLOSE_CURSOR procedure 66
  COLUMN_VALUE_BLOB procedure 67
  COLUMN_VALUE_CHAR procedure 67
  COLUMN_VALUE_CLOB procedure 68
  COLUMN_VALUE_DATE procedure 68
  COLUMN_VALUE_DOUBLE procedure 69
  COLUMN_VALUE_INT procedure 70
  COLUMN_VALUE_LONG procedure 70
  COLUMN_VALUE_NUMBER procedure 71
  COLUMN_VALUE_RAW procedure 72
  COLUMN_VALUE_TIMESTAMP procedure 73
  COLUMN_VALUE_VARCHAR procedure 73
  DEFINE_COLUMN_BLOB procedure 74
  DEFINE_COLUMN_CHAR procedure 74
  DEFINE_COLUMN_CLOB procedure 75
  DEFINE_COLUMN_DATE procedure 75
  DEFINE_COLUMN_DOUBLE procedure 76
  DEFINE_COLUMN_INT procedure 76
  DEFINE_COLUMN_LONG procedure 77
  DEFINE_COLUMN_NUMBER procedure 77
  DEFINE_COLUMN_RAW procedure 78
  DEFINE_COLUMN_TIMESTAMP procedure 78
  DEFINE_COLUMN_VARCHAR procedure 79
  DESCRIBE_COLUMNS procedure 79
  DESCRIBE_COLUMNS2 procedure 82
  EXECUTE procedure 84

DBMS_SQL module *(continued)*
EXECUTE_AND_FETCH procedure 85
FETCH_ROWS procedure 88
IS_OPEN function 90
IS_OPEN procedure 91
LAST_ROW_COUNT procedure 92
OPEN_CURSOR procedure 94
overview 58
PARSE procedure 95
VARIABLE_VALUE_BLOB procedure 97
VARIABLE_VALUE_CHAR procedure 98
VARIABLE_VALUE_CLOB procedure 98
VARIABLE_VALUE_DATE procedure 99
VARIABLE_VALUE_DOUBLE procedure 99
VARIABLE_VALUE_INT procedure 100
VARIABLE_VALUE_NUMBER procedure 100
VARIABLE_VALUE_RAW procedure 101
VARIABLE_VALUE_TIMESTAMP procedure 101
VARIABLE_VALUE_VARCHAR procedure 101
DBMS_UTILITY module
ANALYZE_DATABASE procedure 103
ANALYZE_PART_OBJECT procedure 104
ANALYZE_SCHEMA procedure 105
CANONICALIZE procedure 106
COMMA_TO_TABLE procedures 108
COMPILE_SCHEMA procedure 110
DB_VERSION procedure 110
EXEC_DDL_STATEMENT procedure 111
FORMAT_ERROR_BACKTRACE 113
GET_CPU_TIME function 115
GET_DEPENDENCY procedure 116
GET_HASH_VALUE function 117
GET_TIME function 118
NAME_RESOLVE procedure 119
NAME_TOKENIZE procedure 123
overview 102
TABLE_TO_COMMA procedures 126
VALIDATE procedure 128
DBSUMMARY procedure 142
DEFINE_COLUMN_BLOB procedure 74
DEFINE_COLUMN_CHAR procedure 74
DEFINE_COLUMN_CLOB procedure 75
DEFINE_COLUMN_DATE procedure 75
DEFINE_COLUMN_DOUBLE procedure 76
DEFINE_COLUMN_INT procedure 76
DEFINE_COLUMN_LONG procedure 77
DEFINE_COLUMN_NUMBER procedure 77
DEFINE_COLUMN_RAW procedure 78
DEFINE_COLUMN_TIMESTAMP procedure 78
DEFINE_COLUMN_VARCHAR procedure 79
DESCRIBE_COLUMNS procedure 79
DESCRIBE_COLUMNS2 procedure 82
DISABLE procedure 33
DROP_DIRECTORY procedure 153

E

EHLO procedure 179
ENABLE procedure 33
ERASE procedure 26
EXEC_DDL_STATEMENT procedure 111
EXECUTE procedure 84
EXECUTE_AND_FETCH procedure 85

F

FCLOSE procedure 155
FCLOSE_ALL procedure 156
FCOPY procedure 157
FETCH_ROWS procedure 88
FFLUSH procedure 158
FILE_TYPE data type 170
FOPEN function 160
FORMAT_ERROR_BACKTRACE function 113
FREMOVE procedure 161
FRENAME procedure 161
functions
CREATE_PIPE 41
FOPEN 160
FORMAT_CALL_STACK 112
IS_OPEN 164
modules 1
NEXT_ITEM_TYPE 42
PACK_MESSAGE 44
RECEIVE_MESSAGE 47
REMOVE_PIPE 48
UNIQUE_SESSION_NAME 52

G

GET_CPU_TIME procedure 115
GET_DEPENDENCY procedure 116
GET_DIRECTORY_PATH procedure 153
GET_HASH_VALUE function 117
GET_LINE procedure
files 163
message buffers 34
GET_LINES procedure 35
GET_STORAGE_LIMIT function 26
GET_TIME function 118
GETLENGTH function 27

H

HELO procedure 180
HELP procedure 180

I

INITIALIZE procedure 56
INSTR function 27
INTERVAL procedure 16
IS_OPEN function 90, 164
IS_OPEN procedure 91
ISOPEN function 28

L

LAST_ROW_COUNT procedure 92
LOCKWAIT procedure 147

M

MAIL procedure 181
modules
DBMS_ALERT 1
DBMS_DDL 9
DBMS_JOB 12
DBMS_LOB 20
DBMS_OUTPUT 32
DBMS_PIPE 39
DBMS_RANDOM 54
DBMS_SQL 58
DBMS_UTILITY 102
MONREPORT 128
overview 1
UTL_DIR 151
UTL_FILE 154
UTL_MAIL 170
UTL_SMTP 174
MONREPORT module
CONNECTION procedure 130
CURRENTAPPS procedure 141
CURRENTSQL procedure 142
DBSUMMARY procedure 142
details 128
LOCKWAIT procedure 147
PKGCACHE procedure 150

N

n
details 112
NAME_RESOLVE procedure 119
NAME_TOKENIZE procedure 123
NEW_LINE procedure 37, 165
NEXT_DATE procedure 16
NEXT_ITEM_TYPE function 42
NOOP procedure 182
NORMAL function 58

O

OPEN procedures 28
OPEN_CONNECTION function 182
OPEN_CONNECTION procedure 183
OPEN_CURSOR procedure 94
OPEN_DATA procedure 183

P

PACK_MESSAGE function 44
PACK_MESSAGE_RAW procedure 45
PARSE procedure 95
PKGCACHE procedure 150
procedures
  CREATE_DIRECTORY 151
  CREATE_OR_REPLACE_DIRECTORY 152
  CREATE_WRAPPED 10
  DISABLE 33
  DROP_DIRECTORY 153
  ENABLE 33
  FCLOSE 155
  FCLOSE_ALL 156
  FCOPY 157
  FFLUSH 158
  FREMOVE 161
  FRENAME 161
  GET_DIRECTORY_PATH 153
  GET_LINE 34, 163
  GET_LINES 35
  NEW_LINE 37, 165
  PACK_MESSAGE_RAW 45
  PURGE 46
  PUT 38, 166
  PUT_LINE 38, 168
  PUTF 169
  REGISTER 2
  REMOVE 3
  REMOVEALL 4
  RESET_BUFFER 50
  SEND_MESSAGE 51
  SET_DEFAULTS 4
  SIGNAL 5
  UNPACK_MESSAGE 53
  WAITANY 6
  WAITONE 7
PURGE procedure 46
PUT procedure
  put partial line in message buffer 38
  write string to file 166
PUT_LINE procedure
  put complete line in message buffer 38
  write text to file 168
PUTF procedure 169

Q

QUIT procedure 184

R

RANDOM function 56
RCPT procedure 184
READ procedures 29
RECEIVE_MESSAGE function 47
REGISTER procedure 2
REMOVE procedure
  delete job definition from database 17
  remove registration for specified alert 3
REMOVE_PIPE function 48
REMOVEALL procedure 4
RESET_BUFFER procedure 50
routines
  modules 1
RSET procedure 185
RUN procedure 18

S

SEED procedure 55
SEED_STRING procedure 55
SEND procedure 171
SEND_ATTACH_RAW procedure 172
SEND_ATTACH_VARCHAR2 procedure 173
SEND_MESSAGE procedure 51
SET_DEFAULTS procedure 4
SIGNAL procedure 5
STRING function 57
SUBMIT procedure 18
SUBSTR scalar function 29

T

TABLE_TO_COMMA procedures 126
TERMINATE procedure 56
TRIM procedures 30

U

UNIQUE_SESSION_NAME function 52
UNPACK_MESSAGE procedures 53
UTL_DIR module 151
UTL_FILE module 154
UTL_MAIL module
  overview 170
  SEND procedure 171
  SEND_ATTACH_RAW procedure 172
  SEND_ATTACH_VARCHAR2 procedure 173
UTL_SMTP module
  CLOSE procedure 177
  COMMAND procedure 177
  COMMAND_REPLIES procedure 178
  DATA procedure 179
  EHLO procedure 179
  HELO procedure 180
  HELP procedure 180
  MAIL procedure 181
  NOOP procedure 182
  OPEN_CONNECTION function 182
  OPEN_CONNECTION procedure 183
  OPEN_DATA procedure 183
  overview 174
  QUIT procedure 184
  RCPT procedure 184
  RSET procedure 185
  VRFY procedure 185
  WRITE_DATA procedure 186
  WRITE_RAW_DATA procedure 186

V

VALIDATE procedure 128
VALUE function 56
VARIABLE_VALUE_BLOB procedure 97
VARIABLE_VALUE_CHAR procedure 98
VARIABLE_VALUE_CLOB procedure 98
VARIABLE_VALUE_DATE procedure 99
VARIABLE_VALUE_DOUBLE procedure 99
VARIABLE_VALUE_INT procedure 100
VARIABLE_VALUE_NUMBER procedure 100
VARIABLE_VALUE_RAW procedure 101
VARIABLE_VALUE_TIMESTAMP procedure 101
VARIABLE_VALUE_VARCHAR procedure 101
VRFY procedure 185

W

WAITANY procedure 6
WAITONE procedure 7
WHAT procedure 19
WRAP function 9
WRITE procedures 30
WRITE_DATA procedure 186
WRITE_RAW_DATA procedure 186
WRITEAPPEND procedures 31

Printed in USA