

SQL_EXEC_PROC

SQL_EXEC_PROC action

Function The action executes given SQL command (stored procedure or function) along with the parameters.

Declaration

```
SQL_EXEC_PROC handleIdent_Int, retCodeIdent_Int, stringExpr BIND _locVar1,
_locVar2, ...

SQL_EXEC_PROC handleIdent_Int, retCodeIdent_Int, stringExpr BIND
_locVarRowIdent

SQL_EXEC_PROC handleIdent_Int, retCodeIdent_Int, stringExpr BIND
_locVarRecordIdent
```

Parameters

handleIdent_Int	in	Identifier of the <i>Int</i> type - unique number (handle) of a connection to database.
retCodeIdent_Int	out	Identifier of the <i>Int</i> type - return code.
strExpr	in	Expression of the <i>Text</i> type - SQL procedure or function.
_locVar1, _locVar2, ...	in /out	Local variables.
_locVarRowIdent	in /out	Reference to a row of local variable of the <i>Record</i> type.
_locVarRecordIdent	in /out	Identifier of local variable of the <i>Record</i> type.

Return code Value of the parameter *retCodeIdent_Int* - see the table of [error codes](#). It is also possible to get [extended error information](#).

Description

The action executes specified SQL command over the database opened by the action [SQL_CONNECT](#). The SQL command is represented by the expression *stringExpr*. The command contains input, output or input/output parameters and their values are specified after the keyword *BIND*. If the values of the parameters are changed after the command was executed, the [D2000 DBManager](#) process sets relevant local variables to values changed (with actual timestamps). If the parameter *_locVarRecordIdent* is defined, the SQL command is to be called for each row of the structure. If calling returns an error, the action will not continue with another rows.

Note

SQL command syntax is differ for *ODBC* and *OCI* versions of the [D2000 DBManager](#) process:

- ODBC version of the D2000 DBManager process:**

According to the ODBC convention, a parameter is specified by a question mark, the syntax of procedure call is

"{ call PROCEDURE_NAME(?,?,...) }"

and the syntax of function call is

"{ ? = call FUNCTION_NAME(?,?,...) }"

Examples:

A function with two parameters: "{ ? = call TEST_FUNC(?,?) }"

A procedure with three parameters "{ call TEST_PROC(?,?,?) }"

A procedure with three parameters, while the second parameter is a constant: "{ call TEST_PROC(?,5,?) }"

This example works just for *Sybase SQL Anywhere*, not for *Oracle* (for ODBC, all specified parameters must be bound as variables, not constants). To lift this restriction use the OCI syntax also in the ODBC version: "*BEGIN TEST_PROC(:par1,5,:par3); END;*". On the other hand, *Sybase SQL Anywhere* (unlike *Oracle*) supports output parameters just for procedures, not for functions.

Example for Oracle:

"BEGIN TEST_PROC(?); END;"

Example for PostgreSQL:

"call TEST_PROC(IN ?)"

- **OCI version of the D2000 DBManager process:**

A parameter is marked by colon and its name, the syntax to call a procedure is "*BEGIN PROCEDURE_NAME(:par1,:par2,...); END;*" and the syntax to call a function is "*BEGIN :result := FUNCTION_NAME(:par1,:par2,...); END;*"

Examples:

A function with two parameters: "*BEGIN :res := TEST_FUNC(:par1,:par2); END;*"

A procedure with three parameters: "*BEGIN TEST_PROC(:par1,:par2,:par3); END;*"

A procedure with three parameters, while the second parameter is a constant: "*BEGIN TEST_PROC(:par1,5,:par3); END;*"

The sequence of the parameters is defined by a sequence in the string (in the previous example *:res, :par1 a :par2*). If the parameters are declared with the same name, they will be taken as one parameter, i.e. the SQL command "*BEGIN :res := TEST_FUNC(:parX,:parX); END;*" contains these two parameters *:res* (function result) and *:parX* (two output parameters with the same value).

In the OCI version of the **D2000 DBManager process**, SQL command may be an entire sequence, e.g.

"BEGIN :res := TEST_FUNC(:par1,:par2); IF :res=0 THEN :res := TEST_FUNC2(:par1,:par2); END IF; END;"

Within calling the procedure, the user may specify the type of the parameter which will bind by the modifiers IN, INOUT, OUT placed in front of symbol of bound value.

Example:

```
"
{ call TEST_PROC(IN ?, INOUT?, OUT?) }
"
"BEGIN TEST_PROC(IN :par1, INOUT :par2, OUT :par3); END;"
```

The use of modifiers is optional (default value is INOUT). The modifiers are not case sensitive.

Note: in the case of the PostgreSQL database, it is necessary to specify the IN modifier for all input parameters, we recommend specifying the INOUT and OUT modifiers as well.

Example

1. ODBC version of the process D2000 DBManager

Example: Creating stored procedures in SQL Anywhere:

```
/* par1 is input/output parameter, par2 is input one and par3 is output parameter */
create procedure TEST_PROC(@par1 varchar(10) output, @par2 integer, @par3 integer output)
as
declare @vysl integer
begin
select @par=@par+'XYZ'
select @par3=2*@par2
end
```

```
/* example of function with two parameters (Sybase supports just input function parameters) */
create function TEST_FUNC(in @par1 real,in @par2 integer)
returns real as
begin
return(@par1*@par2)
end
```

Calls from script:

```

BEGIN
INT  _myInt
INT  _iRetCode
INT  _iHandle
TEXT _myText
REAL _myReal
INT  _myInt1
INT  _myInt2

_myText := "ABC"
_myInt1 := 10

SQL_CONNECT MyDB, _iHandle, _iRetCode

; procedure call
SQL_EXEC_PROC _iHandle, _iRetCode, "{ call TEST_PROC(?,?,?) }" BIND
_myText, _myInt1, _myInt2
; value of myText is "ABCXYZ" and value of _myInt2 is 20 (2 * 10)

; call of procedure with a constant
SQL_EXEC_PROC _iHandle, _iRetCode, "{ call TEST_PROC(?,3,?) }" BIND
_myText, _myInt1
; value of _myText is "ABCXYZXYZ" and value of _myInt2 is 6 (2 * 3)

; function call
SQL_EXEC_PROC _iHandle, _iRetCode, "{ ? = call TEST_FUNC(?,?) }" BIND
_myReal, _myInt1, _myInt2
; value of _myReal is 60 (10 * 6)

; call of function with a constant
SQL_EXEC_PROC _iHandle, _iRetCode, "{ ? = call TEST_FUNC(?,3.3) }" BIND
_myReal, _myInt1
; value of _myReal is 33 (10 * 3.3)

```

2. OCI version of the process D2000 DBManager

Example: creating stored procedures in Oracle 9i:

```

/* par1 is input/output parameter, par2 is input and par3 is output parameter */
CREATE OR REPLACE PROCEDURE "MYUSER"."TEST_PROC" (
  par1 in out varchar, par2 integer, par3 out integer
)
as
begin
  par1 := par1 || 'XYZ';
  par3 := 2 * par2;
end;

```

```

/* par1, par2 are input parameters, succ is output one */
CREATE OR REPLACE FUNCTION "MYUSER"."TEST_FUNC" (
  par1 in float, par2 in float, succ out integer
)
return float
as
begin
  if par2 = 0.0 then
    succ := 0;
    return 0;
  else
    succ := 1;
    return par1/par2;
  end if;
end;

```

Calls from script:

```

BEGIN
INT  _myInt
INT  _iRetCode
INT  _iHandle
TEXT _myText
REAL _myReal
INT  _myInt1
INT  _myInt2
INT  _Succ

_myText := "ABC"
_myInt1 := 10

SQL_CONNECT MyDB, _iHandle, _iRetCode

; procedure call
SQL_EXEC_PROC _iHandle, _iRetCode, "BEGIN TEST_PROC(:p1,:p2,:p3); END;"
BIND _myText, _myInt1, _myInt2
;value of _myText is "ABCXYZ" and value of _myInt2 is 20 (2 * 10)

; function call
SQL_EXEC_PROC _iHandle, _iRetCode, "BEGIN :ret := TEST_FUNC(:par1,:par2,:par3); END;" BIND _myReal, _myInt1, _myInt2, _Succ
; value of _myReal is 0.5 (10 / 20) and value of _Succ is 1

```

Related topics

[DB_TRANS_OPEN](#)
[DB_TRANS_COMMIT](#)
[DB_TRANS_ROLLBACK](#)
[DB_TRANS_CLOSE](#)

[SQL_CONNECT](#)
[SQL_DISCONNECT](#)
[SQL_EXEC_DIRECT](#)

[SQL_PREPARE](#)
[SQL_BINDIN](#)
[SQL_FETCH](#)
[SQL_FREE](#)

[SQL_SELECT](#)

[All database related actions](#)



Related pages:

[Script actions](#)