101
DBMS_HS_PASSTHROUGH

The `DBMS_HS_PASSTHROUGH` PL/SQL package allows you to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows operations in statements for which there is no equivalent in Oracle.

This chapter discusses the following topics:

- DBMS_HS_PASSTHROUGH Overview
- DBMS_HS_PASSTHROUGH Operational Notes
- Summary of DBMS_HS_PASSTHROUGH Subprograms

See Also:

Oracle Database Heterogeneous Connectivity User's Guide for more information about this package

DBMS_HS_PASSTHROUGH Overview

You can execute passthrough SQL statements directly at the non-Oracle system using the PL/SQL package `DBMS_HS_PASSTHROUGH`. Any statement executed with this package is executed in the same transaction as standard SQL statements.

See Also:

Oracle Database Heterogeneous Connectivity User's Guide for information about this package

DBMS_HS_PASSTHROUGH Operational Notes

The `DBMS_HS_PASSTHROUGH` package is a virtual package. It conceptually resides at the non-Oracle system. In reality, however, calls to this package are intercepted by Heterogeneous Services and mapped to one or more Heterogeneous Services calls. The driver, in turn, maps these Heterogeneous Services calls to the API of the non-Oracle system. The client application should invoke the procedures in the package through a database link in exactly the same way as it would invoke a non-Oracle system stored procedure. The special processing done by Heterogeneous Services is transparent to the user.

Summary of DBMS_HS_PASSTHROUGH Subprograms

This table lists the `DBMS_HS_PASSTHROUGH` subprograms and briefly describes them.

Table 101-1 *DBMS_HS_PASSTHROUGH Package Subprograms*

Subprogram	Description
BIND_INOUT_VARIABLE Procedure	Binds IN OUT bind variables
BIND_INOUT_VARIABLE_RAW Procedure	Binds IN OUT bind variables of datatype RAW
BIND_OUT_VARIABLE Procedure	Binds an OUT variable with a PL/SQL program variable
BIND_OUT_VARIABLE_RAW Procedure	Binds an OUT variable of datatype RAW with a PL/SQL program variable
BIND_VARIABLE Procedure	Binds an IN variable positionally with a PL/SQL program variable
BIND_VARIABLE_RAW Procedure	Binds IN variables of type RAW
CLOSE_CURSOR Procedure	Closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system
EXECUTE_IMMEDIATE Procedure	Runs a (non-SELECT) SQL statement immediately, without bind variables
EXECUTE_NON_QUERY Function	Runs a (non-SELECT) SQL statement
FETCH_ROW Function	Fetches rows from a query
GET_VALUE Procedure	Retrieves column value from SELECT statement, or retrieves OUT bind parameters
GET_VALUE_RAW Procedure	Similar to GET_VALUE, but for datatype RAW
OPEN_CURSOR Function	Opens a cursor for running a passthrough SQL statement at the non-Oracle system
PARSE Procedure	Parses SQL statement at non-Oracle system

BIND_INOUT_VARIABLE Procedure

This procedure binds IN OUT bind variables.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c       IN   BINARY_INTEGER NOT NULL,
   p       IN   BINARY_INTEGER NULL,
   v       OUT <dty>,
   n       IN   VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

See Also:

For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure.

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-2 BIND_INOUT_VARIABLE Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.
`p`	Position of the bind variable in the SQL statement: Starts at 1.
`v`	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
`n`	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable `:ename` is 1, the name is `:ename`. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-3 BIND_INOUT_VARIABLE Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

BIND_INOUT_VARIABLE_RAW Procedure

This procedure binds `IN OUT` bind variables of datatype `RAW`.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_INOUT_VARIABLE_RAW (
   c      IN     BINARY_INTEGER NOT NULL,
   p      IN     BINARY_INTEGER NOT NULL,
   v      IN OUT RAW,
   n      IN     VARCHAR2);
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-4 BIND_INOUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable in the SQL statement: Starts at 1.
v	This value is used for two purposes: - To provide the IN value before the SQL statement is run. - To determine the size of the out value.
n	(Optional) Name the bind variable. For example, in SELECT * FROM emp WHERE ename=:ename, the position of the bind variable :ename is 1, the name is :ename. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-5 BIND_INOUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

BIND_OUT_VARIABLE Procedure

This procedure binds an OUT variable with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE (
   c       IN   BINARY_INTEGER NOT NULL,
   p       IN   BINARY_INTEGER NULL,
   v       OUT <dty>,
   n       IN   VARCHAR2);
```

<dty> is either DATE, NUMBER, or VARCHAR2.

See Also:

For binding OUT variables of datatype RAW, see BIND_OUT_VARIABLE_RAW Procedure.

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-6 BIND_OUT_VARIABLE Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.
`p`	Position of the bind variable in the SQL statement: Starts at 1.
`v`	Variable in which the `OUT` bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use `GET_VALUE` to retrieve the value of the `OUT` parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using `BIND_OUT_VARIABLE`.
`n`	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable `:ename` is 1, the name is `:ename`. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-7 BIND_OUT_VARIABLE Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

BIND_OUT_VARIABLE_RAW Procedure

This procedure binds an `OUT` variable of datatype `RAW` with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_OUT_VARIABLE_RAW (
   c     IN  BINARY_INTEGER NOT NULL,
   p     IN  BINARY_INTEGER NOT NULL,
   v     OUT RAW,
   n     IN  VARCHAR2);
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-8 BIND_OUT_VARIABLE_RAW Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.
`p`	Position of the bind variable in the SQL statement: Starts at 1.
`v`	Variable in which the `OUT` bind variable stores its value. The package remembers only the "size" of the variable. After the SQL statement is run, you can use `GET_VALUE` to retrieve the value of the `OUT` parameter. The size of the retrieved value should not exceed the size of the parameter that was passed using `BIND_OUT_VARIABLE_RAW`.
`n`	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable `:ename` is 1, the name is `:ename`. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-9 BIND_OUT_VARIABLE_RAW Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

BIND_VARIABLE Procedure

This procedure binds an `IN` variable positionally with a PL/SQL program variable.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE (
   c       IN BINARY_INTEGER NOT NULL,
   p       IN BINARY_INTEGER NOT NULL,
   v       IN <dty>,
   n       IN VARCHAR2);
```

`<dty>` is either `DATE`, `NUMBER`, or `VARCHAR2`.

See Also:

To bind `RAW` variables use BIND_VARIABLE_RAW Procedure.

Pragmas

```
Purity level defined: WNDS, RNDS
```

Parameters

Table 101-10 BIND_VARIABLE Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed using the routines `OPEN_CURSOR` and `PARSE` respectively.
`p`	Position of the bind variable in the SQL statement: Starts at 1.
`v`	Value that must be passed to the bind variable name.
`n`	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable :`ename` is 1, the name is :`ename`. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-11 BIND_VARIABLE Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

BIND_VARIABLE_RAW Procedure

This procedure binds `IN` variables of type `RAW`.

Syntax

```
DBMS_HS_PASSTHROUGH.BIND_VARIABLE_RAW (
   c     IN BINARY_INTEGER NOT NULL,
   p     IN BINARY_INTEGER NOT NULL,
   v     IN RAW,
   n     IN VARCHAR2);
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-12 BIND_VARIABLE_RAW Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.
`p`	Position of the bind variable in the SQL statement: Starts at 1.
`v`	Value that must be passed to the bind variable.
`n`	(Optional) Name of the bind variable. For example, in `SELECT * FROM emp WHERE ename=:ename`, the position of the bind variable `:ename` is 1, the name is `:ename`. This parameter can be used if the non-Oracle system supports "named binds" instead of positional binds. Passing the position is still required.

Exceptions

Table 101-13 BIND_VARIABLE_RAW Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

CLOSE_CURSOR Procedure

This function closes the cursor and releases associated memory after the SQL statement has been run at the non-Oracle system. If the cursor was not open, then the operation is a "no operation".

Syntax

```
DBMS_HS_PASSTHROUGH.CLOSE_CURSOR (
   c IN BINARY_INTEGER NOT NULL);
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-14 CLOSE_CURSOR Procedure Parameters

Parameter	Description
`c`	Cursor to be released.

Exceptions

Table 101-15 CLOSE_CURSOR Procedure Exceptions

Exception	Description
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

EXECUTE_IMMEDIATE Procedure

This function runs a SQL statement immediately. Any valid SQL command except `SELECT` can be run immediately.

The statement must not contain any bind variables. The statement is passed in as a `VARCHAR2` in the argument. Internally the SQL statement is run using the `PASSTHROUGH` SQL protocol sequence of `OPEN_CURSOR`, `PARSE`, `EXECUTE_NON_QUERY`, `CLOSE_CURSOR`.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE (
   s IN VARCHAR2 NOT NULL)
RETURN BINARY_INTEGER;
```

Parameters

Table 101-16 EXECUTE_IMMEDIATE Procedure Parameters

Parameter	Description
`s`	`VARCHAR2` variable with the statement to be executed immediately.

Return Values

The number of rows affected by the execution of the SQL statement.

Exceptions

Table 101-17 EXECUTE_IMMEDIATE Procedure Exceptions

Exception	Description
`ORA-28551`	SQL statement is invalid.
`ORA-28554`	Max open cursors.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

EXECUTE_NON_QUERY Function

This function runs a SQL statement. The SQL statement cannot be a `SELECT` statement. A cursor has to be open and the SQL statement has to be parsed before the SQL statement can be run.

Syntax

```
DBMS_HS_PASSTHROUGH.EXECUTE_NON_QUERY (
   c IN BINARY_INTEGER NOT NULL)
  RETURN BINARY_INTEGER;
```

Parameters

Table 101-18 EXECUTE_NON_QUERY Function Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.

Return Values

The number of rows affected by the SQL statement in the non-Oracle system

Exceptions

Table 101-19 EXECUTE_NON_QUERY Function Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

FETCH_ROW Function

This function fetches rows from a result set.

The result set is defined with a SQL `SELECT` statement. When there are no more rows to be fetched, the exception `NO_DATA_FOUND` is raised. Before the rows can be fetched, a cursor has to be opened, and the SQL statement has to be parsed.

Syntax

```
DBMS_HS_PASSTHROUGH.FETCH_ROW (
   c   IN BINARY_INTEGER NOT NULL,
   f   IN BOOLEAN)
  RETURN BINARY_INTEGER;
```

Pragmas

```
Purity level defined : WNDS
```

Parameters

Table 101-20 FETCH_ROW Function Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines `OPEN_CURSOR` and `PARSE` respectively.
`first`	(Optional) Reexecutes `SELECT` statement. Possible values: - `TRUE`: reexecute `SELECT` statement. - `FALSE`: fetch the next row, or if run for the first time, then execute and fetch rows (default).

Return Values

The returns the number of rows fetched. The function returns "0" if the last row was already fetched.

Exceptions

Table 101-21 FETCH_ROW Function Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

GET_VALUE Procedure

This procedure has two purposes: it retrieves the select list items of `SELECT` statements after a row has been fetched, and it retrieves the `OUT` bind values, after the SQL statement has been run.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE (
   c       IN  BINARY_INTEGER NOT NULL,
   p     IN  BINARY_INTEGER NOT NULL,
   v     OUT <dty>);
```

`<dty>` is either `DATE`, `NUMBER`, or `VARCHAR2`.

See Also:

For retrieving values of datatype `RAW`, see GET_VALUE_RAW Procedure.

Pragmas

```
Purity level defined : WNDS
```

Parameters

Table 101-22 GET_VALUE Procedure Parameters

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.
p	Position of the bind variable or select list item in the SQL statement: Starts at 1.
v	Variable in which the OUT bind variable or select list item stores its value.

Exceptions

Table 101-23 GET_VALUE Procedure Exceptions

Exception	Description
ORA-1403	Returns NO_DATA_FOUND exception when running the GET_VALUE after the last row was fetched (that is, FETCH_ROW returned "0").
ORA-28550	The cursor passed is invalid.
ORA-28552	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
ORA-28553	The position of the bind variable is out of range.
ORA-28555	A NULL value was passed for a NOT NULL parameter.

GET_VALUE_RAW Procedure

This procedure is similar to GET_VALUE, but for datatype RAW.

Syntax

```
DBMS_HS_PASSTHROUGH.GET_VALUE_RAW (
   c     IN  BINARY_INTEGER NOT NULL,
   p     IN  BINARY_INTEGER NOT NULL,
   v     OUT RAW);
```

Pragmas

```
Purity level defined : WNDS
```

Parameters

Table 101-24 GET_VALUE_RAW Procedure Parameters

Parameter	Description
c	Cursor associated with the passthrough SQL statement. Cursor must be opened and parsed, using the routines OPEN_CURSOR and PARSE respectively.

Table 101-24 (Cont.) GET_VALUE_RAW Procedure Parameters

Parameter	Description
`p`	Position of the bind variable or select list item in the SQL statement: Starts at 1.
`v`	Variable in which the `OUT` bind variable or select list item stores its value.

Exceptions

Table 101-25 GET_VALUE_RAW Procedure Exceptions

Exception	Description
`ORA-1403`	Returns `NO_DATA_FOUND` exception when running the `GET_VALUE` after the last row was fetched (that is, `FETCH_ROW` returned "0").
`ORA-28550`	The cursor passed is invalid.
`ORA-28552`	Procedure is not run in right order. (Did you first open the cursor and parse the SQL statement?)
`ORA-28553`	The position of the bind variable is out of range.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.

OPEN_CURSOR Function

This function opens a cursor for running a passthrough SQL statement at the non-Oracle system. This function must be called for any type of SQL statement.

The function returns a cursor, which must be used in subsequent calls. This call allocates memory. To deallocate the associated memory, call the procedure `CLOSE_CURSOR`.

Syntax

```
DBMS_HS_PASSTHROUGH.OPEN_CURSOR
  RETURN BINARY_INTEGER;
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Return Values

The cursor to be used on subsequent procedure and function calls.

Exceptions

Table 101-26 OPEN_CURSOR Function Exceptions

Exception	Description
`ORA-28554`	Maximum number of open cursor has been exceeded. Increase Heterogeneous Services' `OPEN_CURSORS` initialization parameter.

PARSE Procedure

This procedure parses an SQL statement at a non-Oracle system.

Syntax

```
DBMS_HS_PASSTHROUGH.PARSE (
   c        IN  BINARY_INTEGER NOT NULL,
   stmt     IN  VARCHAR2 NOT NULL);
```

Pragmas

```
Purity level defined : WNDS, RNDS
```

Parameters

Table 101-27 PARSE Procedure Parameters

Parameter	Description
`c`	Cursor associated with the passthrough SQL statement. Cursor must be opened using function `OPEN_CURSOR`.
`stmt`	Statement to be parsed.

Exceptions

Table 101-28 PARSE Procedure Exceptions

Exception	Description
`ORA-28550`	The cursor passed is invalid.
`ORA-28551`	SQL statement is illegal.
`ORA-28555`	A `NULL` value was passed for a `NOT NULL` parameter.