Authorization

You must have the CREATE TABLE privilege on the database in which the table is created.

The creator receives all the following privileges on the newly created table:

- DROP TEMPORARY TABLE
- DUMP
- REFERENCE
- RESTORE
- SELECT

Privileges Granted Automatically

None.

Global Temporary Trace Tables

Like global temporary tables, global temporary trace tables have a persistent definition, but do not retain rows across sessions.

Global temporary trace tables are not hashed. Instead, each table row is assigned a sequential hash sequence. You can perform an INSERT … SELECT operation on the table to copy its rows into a normal hashed table with a primary key, indexes, or additional attributes common to other base table types if you need to do so.

Define the information you want to trace as the input to the UDF. The UDF can then call the FNC_Trace_Write_DL function (see *SQL Reference: UDF, UDM, and External Stored Procedure Programming*) that places the information in a global temporary trace table. If the trace is not turned on (see "SET SESSION FUNCTION TRACE" on page 1154), the system does no function traceback, though there is still some overhead because the system still calls the UDF.

Make sure to run the trace UDF in nonprotected mode once it has been debugged to ensure that it executes in-line with the procedure. A premature end of the procedure or a rollback of the transaction has no impact on the global temporary trace table, and its contents are not deleted until the session logs off. Enabling function traceback makes your procedure run more slowly because each trace call forces the data to be written into the trace table. Nothing is buffered to ensure that nothing is lost while you are testing a function. See "Example 3: Complete Function Traceback Scenario" on page 309 for a complete example on how to use a global temporary trace table to do function traceback.

Rules and Limitations for Global Temporary Trace Tables

Because global temporary trace tables are not hashed, they have many definition and manipulation restrictions that distinguish them from ordinary global temporary tables (see "Global Temporary Tables" on page 670).

The following rules and limitations apply to global temporary trace tables:

- You cannot define them as set tables. All global temporary trace tables are restricted to the multiset table type.
- You cannot specify fallback characteristics for a global temporary trace table. They are always created as NO FALLBACK by default.

  If you specify FALLBACK, the system returns an error.
- You cannot define a primary key for the table.
- You cannot define *any* indexes on the table, including a primary index.[18]
- You cannot define any column in the table with a BLOB or CLOB data type.
- You cannot define any columns with a UDT type.

  To get around this, you can use FNC calls to dump the predefined attributes of a UDT into non-UDT columns of the trace table. See *SQL Reference: UDF, UDM, and External Stored Procedure Programming* for information about using FNC calls.
- You cannot specify any column constraints other than NULL and NOT NULL.
- You cannot specify default column values for the table.
- The first two columns of the table must be defined as follows, in the order indicated:
  - `BYTE(2)`

    This column records the number of the AMP on which the logged UDF is running.
  - `INTEGER`

    This column records a sequence number that describes the order in which the function trace row was generated.

  Any other columns are optional and user-defined.
- If you try to use the same trace table to simultaneously trace a UDF that runs on an AMP and an external stored procedure or UDF that runs on a PE, the sequence number written to the second column is not in true sequential order. This is because the AMP bases the next sequence number on the sequence of the last row of the trace table and adds one to it no matter where the row originated.

  The rows inserted into the trace table from the PE are hashed to one AMP based on the PE vproc number and the current sequence number. Therefore, if the current sequence number for the PE is 5 and the trace row is added to AMP 1, then a trace write into that table from AMP 1 has the sequence number 6.

  The best practice is to avoid simultaneously tracing on an AMP and PE.

  See *SQL Reference: UDF, UDM, and External Stored Procedure Programming* for more information.

18. This restriction explicitly includes both partitioned and nonpartitioned primary indexes.

- If you specify ON COMMIT PRESERVE ROWS, then the system preserves the rows of the materialized trace table after the transaction commits or after it aborts; otherwise, all rows are deleted from the table.

  If the trace table was materialized for the session by the transaction that aborts, it is deleted, so in that case, no rows are preserved. Once the trace table is successfully materialized, however, the system retains any rows a UDF had written to it before the abort occurred.
- You cannot join a global temporary trace table with another table.
- You cannot update a global temporary trace table. In other words, you cannot perform DELETE, INSERT, or UPDATE statements against a global temporary trace table, though you can specify DELETE ALL.
- You can INSERT … SELECT from a global temporary trace table into another table.
- You can select the contents of the trace table for output to a response spool file for examination.

  The SELECT statement used for this purpose can specify a WHERE clause to determine the criteria used to select the rows. It can also specify an ORDER BY clause.
- You can perform DROP TEMPORARY TABLE on a global temporary trace table.
- Because there is no primary index, all requests that are normally primary index retrievals are, instead, full-table scans.

Example 1

This example defines the mandatory columns for a global temporary trace table as well as several optional, application-specific, columns. When the transaction that materializes *udf_test* commits, its contents are to be deleted.

```
CREATE GLOBAL TEMPORARY TRACE TABLE udf_test
   (proc_ID      BYTE(2),
    sequence     INTEGER,
    udf_name     CHARACTER(15),
    in_quantity  INTEGER,
    trace_val1   FLOAT,
    trace_text   CHARACTER(30))
ON COMMIT DELETE ROWS;
```

A simple query to retrieve the results recorded in *udf_test* might look like this:

```
SELECT *
FROM udf_test
ORDER BY proc_id ASC, sequence ASC;
```

Example 2

The following example defines a simple trace table that has a single column variable length string to capture function output.

This single VARCHAR column approach lends itself to a flexible trace output that can be used by many different functions without having to resort to specific single-purpose trace table column definitions.

```
CREATE GLOBAL TEMPORARY TRACE TABLE udf_test, NO LOG
 (proc_ID      BYTE(2),
  sequence     INTEGER,
  trace_string VARCHAR(256))
ON COMMIT DELETE ROWS;
```

Example 3: Complete Function Traceback Scenario

The following simple stored procedure trace example is a complete scenario that shows the various aspects of using global temporary trace tables to debug a procedure.

The following CREATE FUNCTION request, returned by a SHOW FUNCTION request submitting using BTEQ, defines the traceback UDF that was used for this example. You can make the procedure as simple, or as complex, as your application requires. This particular trace UDF is fairly simple.

```
SHOW FUNCTION sptrace;

 *** Text of DDL statement returned.
 *** Total elapsed time was 1 second.

------------------------------------------------------------------------
CREATE FUNCTION sptrace (
  p1 VARCHAR(100) CHARACTER SET LATIN)
RETURNS INTEGER
SPECIFIC sptrace
LANGUAGE C
NO SQL
PARAMETER STYLE TD_GENERAL
NOT DETERMINISTIC
RETURNS NULL ON NULL INPUT
EXTERNAL NAME sptrace
 *** Text of DDL statement returned.
------------------------------------------------------------------------
#define SQL_TEXT Latin_Text
#include <sqltypes_td.h>
/*
 Install in SYSLIB for general use and change to NOT PROTECTED mode
 ALTER FUNCTION sptrace EXECUTE NOT PROTECTED;
*/
/* Assumes that the following trace table has been created */
/*
```

```
CREATE MULTISET GLOBAL TEMPORARY TRACE TABLE tracetst,
NO FALLBACK, NO LOG (
  proc_id   BYTE(2),
  sequence  INTEGER,
  trace_str VARCHAR(100))
ON COMMIT PRESERVE ROWS;

Turn on tracing with following SQL (decide what your options mean):

SET SESSION FUNCTION TRACE USING 'T' FOR TABLE tracetst;
*/
void sptrace(VARCHAR_LATIN     *trace_text,
             INTEGER           *result,
             char               sqlstate[6])
{
  SQL_TEXT    trace_string[257];
  void        *argv[20];  /* only need 1 arg -- its an array */
  int         length[20]; /* one length for each argument */
  int         tracelen;
    /* read trace string */

    FNC_Trace_String(trace_string);
    /* Get length of string */
    tracelen = strlen(trace_string);
    /* Make sure tracing is turned on */
    if (tracelen == 0)
      return;
    if (trace_string[0] == 'T')

      {
        argv[0] = trace_text;
        length[0] = strlen(trace_text) +1;
        /* Have something to trace */
        FNC_Trace_Write_DL(1, argv, length);
      }
}
```

The following SQL text defines the stored procedure that calls the trace function *sptrace*.

```
SHOW PROCEDURE sptracedemo;
 *** Text of DDL statement returned.
 *** Total elapsed time was 1 second.
------------------------------------------------------------------
CREATE PROCEDURE sptracedemo  (
  num_rows INTEGER  )
BEGIN
-- Teradata mode stored procedure
   DECLARE dummy INTEGER;
   DECLARE start_val INTEGER;
   DECLARE fnum  FLOAT;

SET dummy = sptrace('Start of sptrace demo');
```

```
BEGIN
  DECLARE EXIT HANDLER FOR SQLSTATE '52010'
    BEGIN
      -- Table already exists, delete the contents
      DELETE  sp_demo1 ALL;
      SET dummy=sptrace('Deleted contents of sp_demo1 in handler');
      -- Get error here and procedure will exit
    END;
  CREATE TABLE sp_demo1 (
    a INTEGER,
    b FLOAT);
  SET dummy = sptrace('Table sp_demo1 created');
END;

SET start_val = 1;
SET fnum = 25.3;

WHILE start_val <= num_rows DO
   INSERT INTO sp_demo1 (start_val, fnum);
   SET dummy = sptrace('did: insert (' || start_val ||','|| fnum
               ||');');
   SET start_val = start_val +1;
   SET fnum = sqrt(fnum);
END WHILE;

SET dummy = sptrace('Got to end of sptrace demo'); END;
```

The remainder of the scenario constitutes the actual example.

Note that global temporary trace tables are user-defined except for first two columns, which are always fixed.

First show the definition of the global temporary trace table, *tracetst*, used for this example.

```
SHOW TABLE tracetst;

 *** Text of DDL statement returned.
 *** Total elapsed time was 1 second.

------------------------------------------------------------------
CREATE MULTISET GLOBAL TEMPORARY TRACE TABLE tracetst,NO FALLBACK,
CHECKSUM = DEFAULT,
NO LOG (
  proc_id   BYTE(2),
  sequence  INTEGER,
  trace_str VARCHAR(100) CHARACTER SET LATIN NOT CASESPECIFIC)
ON COMMIT PRESERVE ROWS;
```

The next request enables function traceback for the session (see “SET SESSION FUNCTION TRACE” on page 1154 for additional information). The variable *t* is a user-defined string, interpreted by your trace UDF, that can be up to 255 characters long.

```
SET SESSION FUNCTION TRACE USING 't' FOR TABLE tracetst;

 *** Set SESSION accepted.
 *** Total elapsed time was 1 second.
```

Function traceback has been enabled for the session.

Now select the contents of the global temporary trace table *tracetst* to ensure that it has no rows.

```
SELECT *
FROM tracetst
ORDER BY 1, 2;

 *** Query completed. No rows found.
 *** Total elapsed time was 1 second.
```

The trace table is empty.

Run the trace procedure *sptracedemo*, whose definition is indicated by the SHOW PROCEDURE request at the end of this example:

```
CALL sptracedemo(3);
 *** Procedure has been executed.
 *** Total elapsed time was 1 second.
```

Select the contents of your global temporary trace table *tracetst* after the procedure executes. Refer to the stored procedure definition definition to see exactly what it traces.

```
SELECT *
FROM tracetst
ORDER BY 1, 2;

 *** Query completed. 6 rows found. 3 columns returned.
 *** Total elapsed time was 1 second.

proc_id Sequence trace_str
------- -------- ------------------------------------------------------------
FF3F           1 Start of sptrace demo
FF3F           2 Table sp_demo1 created
FF3F           3 did: insert (          1, 2.53000000000000E 001);
FF3F           4 did: insert (          2, 5.02991053598372E 000);
FF3F           5 did: insert (          3, 2.24274620409526E 000);
FF3F           6 Got to end of sptrace demo
```

Now call *sptracedemo* again.

```
CALL sptracedemo(3);

 *** Procedure has been executed.
 *** Total elapsed time was 1 second.
```

Select the contents of the global temporary trace table again. Note that it takes a different path because the global temporary trace table has already been populated with rows this time.

```
SELECT *
FROM tracetst
ORDER BY 1, 2;

 *** Query completed. 12 rows found. 3 columns returned.
 *** Total elapsed time was 1 second.
```

```
proc_id Sequence trace_str
------- ------------------------------------------------------
FF3F     1 Start of sptrace demo
FF3F     2 Table sp_demo1 created
FF3F     3 did: insert (          1, 2.53000000000000E 001);
FF3F     4 did: insert (          2, 5.02991053598372E 000);
FF3F     5 did: insert (          3, 2.24274620409526E 000);
FF3F     6 Got to end of sptrace demo
FF3F     7 Start of sptrace demo
FF3F     8 deleted contents of sp_demo1 in handler
FF3F     9 did: insert (          1, 2.53000000000000E 001);
FF3F    10 did: insert (          2, 5.02991053598372E 000);
FF3F    11 did: insert (          3, 2.24274620409526E 000);
FF3F    12 Got to end of sptrace demo
```

Disable function traceback to show that nothing additional is written to the trace table when the procedure is called, but function traceback is not enabled.

```
SET SESSION FUNCTION TRACE OFF;

 *** Set SESSION accepted.
 *** Total elapsed time was 1 second.

CALL sptracedemo(3);

 *** Procedure has been executed.
 *** Total elapsed time was 1 second.

SELECT *
FROM tracetst
ORDER BY 1, 2;

 *** Query completed. 12 rows found. 3 columns returned.
 *** Total elapsed time was 1 second.

proc_id Sequence trace_str
------- ------------------------------------------------------
FF3F     1 Start of sptrace demo
FF3F     2 Table sp_demo1 created
FF3F     3 did: insert (          1, 2.53000000000000E 001);
FF3F     4 did: insert (          2, 5.02991053598372E 000);
FF3F     5 did: insert (          3, 2.24274620409526E 000);
FF3F     6 Got to end of sptrace demo
FF3F     7 Start of sptrace demo
FF3F     8 deleted contents of sp_demo1 in handler
FF3F     9 did: insert (          1, 2.53000000000000E 001);
FF3F    10 did: insert (          2, 5.02991053598372E 000);
FF3F    11 did: insert (          3, 2.24274620409526E 000);
FF3F    12 Got to end of sptrace demo
```

The identical 12 rows are selected from *tracetst*, so function traceback was successfully disabled.

Related Topics

See "SET SESSION FUNCTION TRACE" on page 1154 for information about enabling function traceback.

Also see "Global Temporary Tables" on page 670 and "Volatile and Global Temporary Tables, Teradata Session Mode, and DML Performance" on page 671 for general information about global temporary tables and *SQL Reference: UDF, UDM, and External Stored Procedure Programming* for information about coding external functions.