LOB Guide

Version 2.4.0

Table of Contents

License Statement

Acknowledgements

WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material. <<<

Revision History

Version	Date
2.2.0	TBD
2.1.0	May 1, 2017
2.0.1	July 7, 2016
2.0.0	June 6, 2016
1.3.0	January, 2016

Chapter 1. About This Document

This guide describes how to use Large Object (LOB) datatypes in Trafodion SQL.

1.1. Intended Audience

This manual is intended for programmers who use LOB datatypes.

1.2. New and Changed Information

This is a new guide.

1.3. Document Organization

Chapter	Description
Introduction	Introduces LOBs and covers following topics: • 1.1 What is a LOB • 1.2 LOB Data Types • 1.3 LOB Storage • 1.4 LOB Handle • 1.5 LOB Restrictions • 1.6 LOB Related SQL Statements and Functions.
Working With LOBs	Explains how to use a LOB with SQL statement and covers following topics: • 2.1 Creating a SQL Table with LOB Columns • 2.2 Inserting into a SQL Table Containing LOB Columns • 2.3 Inserting into a SQL Table Containing LOB Columns Using Select Clause • 2.4 Updating a SQL Table Containing LOB Columns • 2.5 Selecting Column from a SQL Table Containing LOB Columns • 2.6 Extracting LOB Data from a SQL Table Containing LOB Columns • 2.7 Deleting Column from a SQL Table Containing LOB columns • 2.8 Dropping a SQL Table Containing LOB Columns • 2.9 Garbage Collection • 2.10 Cleanup of a SQL Table Containing LOB Columns • 2.11 SHOWDDL for LOBs • 2.12 Getting Statement for LOB Tables

1.4. Notation Conventions

This list summarizes the notation conventions for syntax presentation in this manual.

- UPPERCASE LETTERS

  Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required.

```
SELECT
```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

```
file-name
```

- [ ] Brackets

  Brackets enclose optional syntax items.

```
DATETIME [start-field TO] end-field
```

  A group of items enclosed in brackets is a list from which you can choose one item or none.

  The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines.

  For example:

```
DROP SCHEMA schema [CASCADE]
DROP SCHEMA schema [ CASCADE | RESTRICT ]
```

- { } Braces

  Braces enclose required syntax items.

```
FROM { grantee [, grantee ] ... }
```

  A group of items enclosed in braces is a list from which you are required to choose one item.

  The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines.

  For example:

```
INTERVAL { start-field TO end-field }
{ single-field }
INTERVAL { start-field TO end-field | single-field }
```

- | Vertical Line

  A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

```
{expression | NULL}
```

- … Ellipsis

  An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times.

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

  An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times.

  For example:

```
expression-n ...
```

- Punctuation

Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown.

```
DAY (datetime-expression)
@script-file
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown.

For example:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

  Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma.

  ```
  DAY (datetime-expression) DAY(datetime-expression)
  ```

  If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

  ```
  myfile.sh
  ```

- Line Spacing

  If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line.

  This spacing distinguishes items in a continuation line from items in a vertical list of selections.

  ```
  match-value [NOT] LIKE _pattern
     [ESCAPE esc-char-expression]
  ```

1.5. Comments Encouraged

We encourage your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to user@trafodion.apache.org.

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document.

Chapter 2. Introduction

2.1. What is a LOB

LOB (Large Object), a set of large object data types used to store large volumes of data, provides random and piece-wise access to the data.

Database support for LOB is not universal.

2.2. LOB Data Types

The following are the data types supported by Trafodion.

Table 1-1 Descriptions for BLOB and CLOB

Data Type	Desciprtion
BLOB	Binary Large Object. Holds large blocks of unstructured data in binary format. BLOB is ideal for storing and handling unstructured data, such as images, audio, video and other multimedia objects.
CLOB	Character Large Object. Stores unusually large amounts of string data in character set format. CLOB is ideal for storing and processing semi-structured data, such as a large document or string.

2.3. LOB Storage

The basic design idea is to allow a database row to have multiple LOB columns and a table to have multiple such rows. The design takes a combination approach of row store and column store.

When a SQL table with a LOB column is created, there are several dependent objects that are created to hold metadata information on the LOB as well as the data.

The LOB data itself is stored in an HDFS file while the Trafodion table contains a unique LOB handle for each LOB value. The LOB handle provides the identifier that is used to query the LOB metadata tables to find the location of the LOB data files.

The naming conventions for the LOB tables are as follows:

Table 1-2 Naming Conventions for LOB Tables

Naming Conventions	
SQL Table	TLOB

Naming Conventions	
LOB MD table (one per SQL table containing LOB columns)	LOBMD_{object UID}
LOB Descriptor Handle Table (one per LOB column)	LOBDescHandle_{object UID}_{LOB number}
LOB Descriptor Chunks Table (one per LOB column)	LOBDescChunks_{object UID}_{LOB number}
LOB Data Table (in HDFS one per LOB column)	LOBP_{object UID}_{LOB number}

2.4. LOB Handle

The LOB handle is used to describe a LOB value. A SQL table that contains a LOB column will contain this handle structure in each row for each LOB value inserted.

The actual LOB data will be stored in unstructured HDFS files as column store whereas the LOB handle—that describes the location, offset information, descriptor information and so on can be thought of as a unique identifier to the lob data, is stored in the database row of the table as a traditional row store.

The handle can be thought of as a file locator as is more commonly referred to in LOB related terminology.

2.4.1. External Structure

The following is an example of an external structure that is stored in the row of a SQL table:

LOBH000000020001000473355576046048801710743811060283701182122798943813543630l7"TRAFODION"."SCH"

2.5. LOB Restrictions

- LOB columns cannot appear in FROM clauses as join predicates.
- LOB columns cannot appear in STORE BY, ORDER BY or GROUP BY clauses.
- LOB columns cannot appear in WHERE clauses as predicates other than LIKE (meaning that no comparison is allowed).
- LOB columns cannot appear in SELECT clauses as aggregate function arguments.
- LOB columns cannot appear in SELECT DISTINCT clauses.
- LOB columns cannot be used as primary keys.
- LOB columns cannot be used in CREATE INDEX statements.

- LOB columns cannot be used in statistics update statements.
- The file that contains data to insert from or to extract to needs to be on platform as a Linux or HDFS files.

2.6. LOB Related SQL Statements and Functions

2.6.1. Supported SQL Statements

The following SQL statements are supported:

- CREATE TABLE
- SELECT
- INSERT
- UPDATE
- UPDATE with APPEND option
- EXTRACT

2.6.2. Unsupported SQL Statements

The following SQL statements are not supported:

- ALTER TABLE
- MERGE

2.6.3. Supported LOB Conversion SQL Functions

The following LOB conversion SQL functions are supported:

- LOBTOSTRING
- LOBTOFILE
- LOBTOBUFFER
- STRINGTOLOB
- FILETOLOB
- BUFFERTOLOB

- EXTERNALTOLOB

Chapter 3. Working with LOBs

3.1. Creating a SQL Table with LOB Columns

When creating a SQL table with LOB columns, following relevant tables and files are created as well:

- One LOB MD table.
- Two dependent descriptor tables.
- HDFS data file (locates at /user/trafodion/lobs) for each column.

3.2. Syntax

```
CREATE TABLE table-name (column-spec[, column-spec]…)
```

```
column-spec is:
lob-column-spec     |
other-column-spec

lob-column-spec is:
column-name {lob-data-type}[column-constraint]

other-column-spec is:
column-name {data-type}[column-constraint]

lob-data-type is:
BLOB | CLOB [({numeric literal} [unit])] [STORAGE 'storage literal']

unit is:
empty |
K     |
M     |
G
```

3.2.1. Semantics

- *`storage literal`*

  Currently Trafodion only supports `'EXTERNAL'` here.

  External LOB object that are not managed by Trafodion.

- *empty*

  Number of bytes specified by the numeric literal.

- *K*

  Numeric literal value * 1024.

- *M*

  Numeric literal value * 1024 * 1024.

- *G*

  Numeric literal value * 1024 * 1024 * 1024.

3.2.2. Examples

- This example creates a table tlob1 with 2 columns and primary key on the c1.

```
CREATE TABLE tlob1 (c1 INT NOT NULL, c2 BLOB, PRIMARY KEY (c1));
```

- This example creates a table tlob2 with 3 columns and primary key on the c1.

```
CREATE TABLE tlob2 (c1 INT NOT NULL, c2 BLOB, c3 CLOB, PRIMARY KEY (c1));
```

- This example creates a table tlob130txt_limit50 with 2 columns and primary key on the c1.

```
CREATE TABLE tlob130txt_limit50 (c1 INT NOT NULL, c2 CLOB(50), PRIMARY KEY (c1));
```

- This example creates a table tlob130bin_limit1K with 2 columns and primary key on the c1.

```
CREATE TABLE tlob130bin_limit1K (c1 INT NOT NULL, c2 BLOB(1 K), PRIMARY KEY (c1));
```

- This example creates a table tlob130ext with 4 columns and primary key on the c1.

```
CREATE TABLE tlob130ext (c1 INT NOT NULL, c2 BLOB, c3 CLOB, c4 BLOB STORAGE
'EXTERNAL', PRIMARY KEY (c1));
```

3.2.3. HDFS Location of LOB Data

When a LOB table is created, the underlying LOB data needs to be stored in HDFS. It is in the /user/trafodion/lobs by default.

All columns of a table that are declared as LOB types will have all their data in one file derived from the table's Object UID and the LOB number of that column which gets assigned during creation.

The following is a LOB file with 2 columns you will see 2 files in HDFS:

/user/trafodion/lobs/LOBP_03683514167332904796_0001

/user/trafodion/lobs/LOBP_03683514167332904796_0002

As rows are added to this table, the LOB data for each row gets appended to the corresponding column's LOB data file.

3.3. Inserting into a SQL Table Containing LOB Columns

3.3.1. Syntax

```
INSERT INTO table-name [(target-col-list)] insert-source
```

```
target-col-list is:
colname[, colname]...

insert-source is:
VALUES(column-expr[, column-expr]...)

column-expr is:
lob-query-expr           |
other-query-expr

lob-query-expr is:
NULL | ?                                                                    |
EMPTY_BLOB()                                                               |
EMPTY_CLOB()                                                               |
STRINGTOLOB('string literal expression')                                   |
FILETOLOB('lob source file name')                                          |
BUFFERTOLOB(LOCATION lob source buffer address, LENGTH lob length value)   |
EXTERNALTOLOB('external lob source file name')

lob source file name is:
hdfs:///{local hdfs file name}     |
{local linux file name}            |
{file:///linux file name}

external lob source file name is:
hdfs:///{local hdfs file name}
```

3.3.2. Semantics

- *other-query-expr*

  For the syntax and description of *other-query-expr*, see the *query-expr* in the SELECT Statement.

- *EMPTY_BLOB(), EMPTY_CLOB()*

  Returns an empty LOB handle.

- *STRINGTOLOB*

  Converts a simple string literal into LOB format.

  - *string literal expression*

    is a series of characters enclosed in single quotes.

- *FILETOLOB*

Converts data from a local linux/hdfs file into LOB format.

- *BUFFERTOLOB*

  Takes an address and a size of an input buffer, and converts the data pointed to by that buffer into LOB.

  - *lob source buffer address*

    The long value of the user buffer address in int64.

  - *lob length value*

    The length of the user specified lob buffer in int64.

3.3.3. Considerations

The source for inserting into a LOB can be any of the following:

- A parameter.

  An unnamed parameter can be used to prepare a statement and then during an execution, either a function or a simple string parameter can be passed in which will be converted to LOB data.

- `EMPTY_BLOB()` or `EMPTY_CLOB()`

If you want to insert `EMPTY_BLOB()` or `EMPTY_CLOB()` into a lob column, the CQD `TRAF_BLOB_AS_VARCHAR` or `TRAF_CLOB_AS_VARCHAR` which is **ON** by default must be turned **OFF** before creating the table, otherwise an error will be raised and the column definition of the lob column is VARCHAR.

  - If `EMPTY_BLOB()` or `EMPTY_CLOB()` is specified, then a dummy lob handle is created.
    - No data is associated with the empty LOBs yet, but these dummy LOB handles can later be used to populate with new LOB data. If the LOB had data previously associated with it, it will be erased.
    - The dummy LOB handle will get the same datatype as the underlying column.

      For example, if the LOB column was defined as `'EXTERNAL'` during table creation, then the LOB column gets that type. If it's not defined, then it is considered as a regular LOB.

  - An empty LOB is distinct from a LOB containing a string of length zero or a null LOB.

- An in-memory LOB which is simple string data.

  To insert a string literal, you need to provide `STRINGTOLOB('string literal expression')`.

- An on-platform file (linux/hdfs file) containing binary or text data.

  To insert an on-platform file, you need to provide `FILETOLOB('lob source file name')`.

- A user buffer of a specified length allocated in user space.

  To insert a buffer, you need to provide the address and size of the buffer.

- An external LOB.

  When an external LOB is specified via `EXTERNALTOLOB('external lob source file name')`, the data associated with the external HDFS file is not transferred into the Trafodion LOB. Instead, Trafodion stores the file path/handle of the external file.

  For example, if you have a directory of pictures, you can specify the full hdfs path to each picture file to this function and the path will get stored in the Trafodion table. Later during retrieval, the file name will be used to go to the actual file to retrieve the data.

3.3.4. Examples

- This example uses a parameter.

```
PREPARE S FROM INSERT INTO t130lob2 VALUES (1, ?);
EXECUTE S USING 'fgfgfhfhfhfhhfhfhfhjfkkfkfflllflflfll';
```

- This example does not turn the CQD `TRAF_BLOB_AS_VARCHAR` OFF before creating the table test1, thus it fails to insert `EMPTY_BLOB()` into c2 whose column definition is VARCHAR.

```
>>CREATE TABLE test1(C1 INT, C2 BLOB);
--- SQL operation complete.

>>CQD TRAF_BLOB_AS_VARCHAR 'OFF';
--- SQL operation complete.

>>INSERT INTO test1 VALUES(1, EMPTY_BLOB());
*** ERROR[4035] Type LOB cannot be cast to type VARCHAR(100000).
*** ERROR[8822] The statement was not prepared.

>>SHOWDDL TABLE test1;
CREATE TABLE TRAFODION.SEABASE.TEST1
  (
    C1   INT DEFAULT NULL NOT SERIALIZED
  , C2   VARCHAR(100000) CHARACTER SET ISO88591 COLLATE DEFAULT DEFAULT NULL NOT
SERIALIZED
  )
 ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

- This example turns the CQD `TRAF_CLOB_AS_VARCHAR` OFF before creating the table test2 and inserting `EMPTY_CLOB()` into c2 whose column definition is CLOB.

```
>>CQD TRAF_CLOB_AS_VARCHAR 'OFF';
--- SQL operation complete.

>>CREATE TABLE test2 (C1 INT, C2 CLOB);
--- SQL operation complete.

>>INSERT INTO test2 VALUES(1, EMPTY_CLOB());
--- 1 row(s) inserted.

>>SHOWDDL TABLE test2;
CREATE TABLE TRAFODION.SEABASE.TEST2
  (
    C1                                INT DEFAULT NULL NOT SERIALIZED
  , C2                                CLOB DEFAULT NULL NOT SERIALIZED
  )
 ATTRIBUTES ALIGNED FORMAT
;
--- SQL operation complete.
```

- This example uses the `STRINGTOLOB` function that converts a simple string literal into LOB format before inserting.

```
INSERT INTO tlob1 VALUES(1,stringtolob('inserted row'));
```

- This example uses the `FILETOLOB` function that converts data from a local file into LOB format, and stores all data into HDFS associated with that value.

```
INSERT INTO tlob130txt1 VALUES(1,filetolob('lob_input_a1.txt'));
```

- This example takes an int64 value as an input which is an address to a buffer and a size parameter. The buffer contents are converted to LOB format and stored in HDFS.

```
INSERT INTO tlob1 VALUES (1, buffertolob(LOCATION 124647474, SIZE 2048));
```

- This example uses different functions to convert strings, files, external lob into LOB data. The EXTERNALTOLOB function takes an external file.

```
INSERT INTO tlob130ext VALUES(1, STRINGTOLOB('first lob'),
FILETOLOB('hdfs:///lobs/lob_input_a1.txt'),
EXTERNALTOLOB('hdfs:///lobs/lob_input_a1.txt'));
```

3.4. Inserting into a SQL Table Containing LOB Columns Using Select Clause

3.4.1. Syntax

```
INSERT INTO target-table [(target-col-list-expr)] SELECT [source-col-list-expr] FROM
source-table
```

```
target-col-list-expr is:
target-colname[, target-colname]...

target-colname is:
lob-column-name                  |
other-column-name

source-col-list-expr is:
source-colname[, source-colname]...

source-colname is:
hive varchar column name         |
trafodion varchar column name    |
trafodion char column name       |
trafodion lob column name

source-table is:
hive table                       |
trafodion table
```

3.4.2. semantics

- *target-col-list-expr*

  names a single column or multiple columns enclosed in parentheses in the target table in which to insert values.

  The data type of each target column must be compatible with the data type of its corresponding source value.

  Within the list, each target column must have the same position as its associated source value.

  - *target-colname*

    is a SQL identifier that specifies a target column name.

- *source-col-list-expr*

  names a single column or multiple columns enclosed in parentheses in the source table from which to get values.

  - *source-colname*

    is a SQL identifier that specifies a source column name.

3.4.3. Considerations

When inserting from a source Trafodion table column into a target table, the source column subtype of the LOB column needs to match the target table column. That is, you cannot insert from an external LOB column into a regular LOB column in the target. They both need to be the same type.

The source for the **select clause** can be any of the following:

- A source hive table column that is a hive varchar column
- A source table column that is a Trafodion varchar, char and LOB column
- A source table column that is also a LOB datatype.

3.4.4. Examples

- This example inserts the first 10 rows of d_date_sk and d_date_id selected from the source hive table hive.hive.date_dim into the target table t130lob2.

```
INSERT INTO t130lob2 SELECT [first 10] d_date_sk,d_date_id FROM hive.hive.date_dim;
```

- This example inserts the c1 and c2 selected from the source Trafodion table t130var into the c1 and c2 of the target table t130lob2.

```
INSERT INTO t130lob2(c1,c2) SELECT c1,c2 FROM t130var;
```

- This example inserts the c1 and c2 selected from the source Trafodion table t130char into the c1 and c2 of the target table t130lob2.

```
INSERT INTO t130lob2(c1,c2) SELECT c1,c2 FROM t130char;
```

- This example inserts the c1, c2 and c3 selected from the source Trafodion table t130lob2 into the target table t130lob3, and then shows the result.

```
INSERT INTO t130lob3 SELECT c1,c2,c3 FROM t130lob2;
SELECT c1,lobtostring(c2,100),lobtostring(c3,100) FROM t130lob3;
```

3.5. Updating a SQL Table Containing LOB Columns

The source for updating a LOB can be divided into 2 groups:

- Update using parameters/functions
- Update using LOB handle

3.5.1. Updating Using Parameters/Functions

The following parameters/functions can be used to update.

- A parameter
- `EMPTY_BLOB` or `EMPTY_CLOB`
- An in-memory lob which is a simple string data

  To insert this string, a literal needs to be provided
- An on-platform file (linux/hdfs file) containing text or binary data
- A user buffer of a specified length allocated in user space
- An external LOB file in HDFS

Syntax

```
UPDATE table-name {set-clause-type1 | set-clause-type2}
```

```
set-clause-type1 is:
SET set-clause[, set-clause ]..

set-clause is:
lob_column-name = {lob_query-expr}

lob_query-expr is:
NULL | ?
|
EMPTY_BLOB()
|
EMPTY_CLOB()
|
STRINGTOLOB('string literal expression'[, APPEND])
|
FILETOLOB('lob source file name'[, APPEND])
|
BUFFERTOLOB(LOCATION lob source buffer address, LENGTH lob length value[, APPEND])
|
EXTERNALTOLOB('external lob source file name'[, APPEND])

lob source file name is:
hdfs:///{local hdfs file name}      |
{local linux file name}             |
{file:///linux file name}

external lob source file name is:
hdfs:///{local hdfs file name}
```

Semantics

For more information, see Semantics in Inserting into a SQL Table Containing LOB Columns.

Examples

- In the table tlob1where c1 is 3, updates (appends) the value of c2 to lob_update.txt.

```
UPDATE tlob1 SET c2=filetolob('lob_update.txt', append) WHERE c1 = 3;
```

- In the table tlob1, updates (overwrites) the value of c2 to anoush.jpg.

```
UPDATE tlob1 SET c2=filetolob('anoush.jpg');
```

- In the table tlob1 where c1 is 3, updates (appends) the value of c2 to lob_update.txt stored in hdfs:///lobs/.

```
UPDATE tlob1 SET c2=filetolob('hdfs:///lobs/lob_update.txt', append) WHERE c1 = 3;
```

- In the table tlob1, updates (overwrites) the value of c2 to the buffer location at 1254674 with 4000-byte length.

```
UPDATE tlob1 SET c2=buffertolob(LOCATION 12546474, SIZE 4000);
```

- In the table tlob130ext where c1 is 2, updates (overwrites) the value of c4 to lob_input_d1.txt stored in hdfs:///lobs/.

```
UPDATE tlob130ext SET c4=externaltolob('hdfs:///lobs/lob_input_d1.txt') WHERE c1=2;
```

- In the table t130lob2 where c1 is 1, updates (overwrites) value of the c2 to xyxyxyxyxyxyx.

```
PREPARE S FROM UPDATE t130lob2 SET c2=? WHERE c1 =1;
EXECUTE S USING 'xyxyxyxyxyxyx';
```

3.5.2. Updating Using Lob Handle

A LOB handle is specified to the update (similar to extract).

Syntax

```
UPDATE LOB (LOB 'lob handle', lob update expression)
```

```
lob update expression is:
LOCATION lob source buffer address, LENGTH lob length value [, APPEND])      |
EMPTY_BLOB()                                                                  |
EMPTY_CLOB()
```

For more information about examples, see **Trafodion JDBC Type 4 Programmer's Reference Guide**.

3.5.3. Considerations

- The keyword APPEND can be used to append to lob data that may already exist in a LOB column. If it is not specified, the data is overwritten.
- When a LOB column is updated or overwritten, a new section is written into the LOB data file. The old section remains in HDFS but won't have any pointer or reference to it.

- When a LOB column is appended, a new chunk is inserted into HDFS and a new row is added in the LOB Descriptor Chunks table to keep track of this new chunk, belonging to the existing row/LOB data.

  So a LOB column that contains data that has been appended several times can contain multiple descriptor rows in the Descriptor Chunks table to describe where each chunk belonging to this LOB begins/ends.

- `APPEND` cannot be used on function `EXTERNALTOLOB`.
- When an external LOB is updated outside of Trafodion, an update needs to be done to update the descriptor files in Trafodion.

  For example, if the external file changes in size, the descriptor file needs to be updated to reflect that. Since this is outside the control of Trafodion, the user needs to do this. If not, when extracting data, only partial data may be returned.

- If a column is declared with the `STORAGE 'External'` attribute, the `STRINGTOLOB` or `FILETOLOB` functions cannot be used to insert data into that column.

  Similarly, if a column is declared without the `STORAGE 'External'` attribute, the `EXTERNALTOLOB` function cannot be used to insert/update data into that column.

  That is, once the storage for a LOB column is declared at CREATE time, the attribute is set and cannot be modified.

3.6. Selecting Column from a SQL Table Containing LOB Columns

Selecting a row from a table will give back the lob handle for the lob column.

The entry in each lob column in the SQL table only contains the LOB handle.

Once the LOB handle has been retrieved, it can be used to retrieve the actual lob data from HDFS.

3.6.1. Syntax

```
SELECT lob_query-expr[, lob_query-expr] FROM table-name
```

```
lob_query_expr is:
lobtostring(lob column)      |
lob column
```

3.6.2. Examples

- This example selects c2 from table tlob1 and returns the lob handle of c2.

```
SELECT c2 FROM tlob1;
C2
--------------------------------------------------------------------------------
----------------

LOBH000002000104239091936503896833196948570103822596837182123109611822902160
21"TRAFODION"."SEABASE"

--- 1 row(s) selected.
```

3.7. Extracting LOB Data from a SQL Table Containing LOB Columns

Extract lob data can be done in two ways:

- Extract lob data into a file for a given lob handle

  Extract from a LOB column straight into an on-platform linux or HDFS file in one shot.

  Trafodion engine will take care of buffering the data and retrieve the lob data into the target file.

- Extract lob data into a user specified buffer

  Extract from a LOB column into a target user buffer of a specified size.

  Trafodion engine will retrieve exactly the amount of requested data.

  The user/application will be responsible for redriving the extract until end of data.

  The extracted buffer data can then be written to files or another location as the application chooses.

3.7.1. Extracting Lob Data into a File for a Given Lob Handle

Syntax

```
EXTRACT LOBTOFILE (LOB 'lob handle as quoted string', 'filename URI format' [OPTION])
```

```
OPTION is:
[, TRUNCATE]
[, CREATE, TRUNCATE]
[, APPEND]
[, CREATE, APPEND]
```

Semantics

- `TRUNCATE`

  If the target file exists, Trafodion will truncate and write to it.

  If the target file does not exist, an error will be raised.

- `CREATE, TRUNCATE`

  If the target file exists, Trafodion will truncate and write to it.

  If the target file does not exist, Trafodion will create a file and write to it.

- `APPEND`

  If the target file exists, Trafodion will append to it.

  If the target file does not exist, an error will be raised.

- `CREATE, APPEND`

  If the target file exists, Trafodion will append to it.

  If the target file does not exist, Trafodion will create a file and append to it.

Considerations

If the target file exists, the `OPTION` must be specified, or else an error will be raised. This is the default behavior.

If the target file does not exist, you can create a target file by specifying the `OPTION`.

Examples

- This example extracts LOB to the tlob130_txt1.txt:

```
EXTRACT LOBTOFILE (LOB
'LOBH000002000105201179972925836255198841214372060931846182123174860183056540 20"TRAF
ODION"."LOB130"', 'tlob130_txt1.txt');
Success. Targetfile:tlob130_txt1.txt  Length: 19
```

- This example extracts LOB to the tlob130_deep.jpg:

```
EXTRACT LOBTOFILE (LOB
'LOBH000002000105201179972925836817198841214372105168125182123174860625866540 20"TRAF
ODION"."LOB130"',
Success. Targetfile:tlob130_deep.jpg  Length: 159018
```

3.7.2. Extracting Lob Data into a User Specified Buffer

Extract from a LOB column into a target user buffer of a specified size. The Trafodion engine will retrieve exactly the amount of requested data. The user/application will be responsible for redriving the extract until end of data. Then the extracted buffer data can be written to files or another location as the application chooses.

Extract LOB data into a user specified buffer like a cursor until EOD is returned. For this method, the user specifies an input buffer and specifies the input length in an in/out variable.

For each execution of the extract, Trafodion will return SUCCESS, ERROR or EOD. For the cases of SUCCESS or EOD, a length will also be returned to the user, so the user knows exactly how much data was actually extracted and returned.

Syntax

```
EXTRACT LOBTOBUFFER (LOB 'lob handle as quoted string', LOCATION lob output buffer
address as long, SIZE input/output address of length container as long)
```

3.7.3. Extracting Lob Length for a Given Lob Handle

Extract from a LOB column straight into an on-platform linux or HDFS file in one shot.

Syntax

```
EXTRACT LOBLENGTH (LOB 'lob handle as quoted string'[, LOCATION address of length
container for lob length])
```

Semantics

- *`LOCATION address of length container for lob length`*

  This is used by programs/applications that will use this syntax to retrieve the LOB length prior to extracting data. The address should be an address of a 64-bit container that will hold the LOB length.

  If the length is omitted or 0, only the status message is returned that displays the length.

Examples

- This example extracts LOB length and returns 30.

```
EXTRACT LOBLENGTH (LOB
'LOBH000000080003055412147848117050211955412147854606441321821233052637376201902 4"TR
AFODION"."TESTEXTLOB"');
LOB Length: 30
--- SQL operation complete.
```

- This example extracts LOB length and returns 4.

```
EXTRACT LOBLENGTH (LOB
'LOBH000002000104239091936503896833196948570103822596837182123109611822902160 21"TRAF
ODION"."SEABASE"');
LOB Length: 4
--- SQL operation complete.
```

3.7.4. Considerations

- LOB Max Extract Data Length

  CQD `LOB_OUTPUT_SIZE` (default 32000) controls the maximum data length that can be extracted.

- LOB Max Extract Chunk Length

  CQD `LOB_MAX_CHUNK_MEM_SIZE` (512 MB expressed in bytes [536870912]) controls the maximum chunk of data that can be read from HDFS into memory and written to the target file location.

- LOB Max Size

  CQD `LOB_MAX_SIZE` (default 10G expressed in M [10000M]).

- Extract Target Locations

  The file to extract to can be a local linux file or a local HDFS file.

3.8. Deleting Column from a SQL Table Containing LOB columns

3.8.1. Syntax

```
DELETE lob-column-name FROM table-name [WHERE CLAUSE]
```

3.8.2. Considerations

When one or more rows containing LOB columns are deleted from LOB table, only the metadata information is dropped and the hdfs data remains as it is. The references to the lob data are removed from the lob descriptor file.

This mechanism has not been implemented yet as a separate utility but it is triggered as a part of insert, update and append operations. For more information, see Garbage Collection.

3.9. Dropping a SQL Table Containing LOB Columns

Drop works like any other drop table. All dependent tables are deleted. All files in hdfs (data and descriptor) files are also deleted.

For more information, see DROP TABLE Statement in Trafodion SQL Reference Manual.

3.10. Garbage Collection

When a lob datafile for a column has reached a certain limit, defined by a CQD `LOB_GC_LIMIT_SIZE`, then a compaction is triggered automatically.
The default Garbage Collection (GC) Limit is 10GB and can be changed if needed.

The need for GC arises because when a delete operation or an update operation is performed, the old data black in the hdfs file will be left as unused.
In the case of update, the old data will be left as unused and the new data will be written into a new section, so all these “holes” in the LOB data file are needlessly occupying space.

The LOB descriptor chunks file is looked at to see which ranges and offsets are actually used. The LOB datafile is

temporarily saved. The compaction is done into a new tempfile. When the sections have all been copied into the tempfile, Trafodion will delete the existing lob data file and rename the tempfile.

Finally, the saved copy of the LOB datafile is dropped. The saved copy is there just in case you need to fall back to it in case of an error. Since this operation is triggered as part of an IUD operation, a definite slowdown will occur for that insert/update operation compared to subsequent inserts/updates.

Also, each lob column of a table can be compacted separately as needed. GC does not have to be done to all columns of the LOB table all at once.

Currently the GC is done in the same transaction as the transaction being used for the insert or update operation. If any part of the GC fails, then the entire transaction is aborted.

When Trafodion has support for local transactions, Trafodion will do the GC in a separate transaction or in a separate process, so you can fail the GC with a warning and allow the insert to go through.

Setting the CQD `LOB_GC_LIMIT_SIZE` to 0 would prevent GC from occurring.

3.11. Cleanup of a SQL Table Containing LOB Columns

Cleanup works like cleanup of any other table. The command ensures all dependent SQL LOB tables and hdfs files are dropped ignoring errors if any.

For more information, see CLEANUP Statement in Trafodion SQL Reference Manual.

3.12. SHOWDDL for LOB

SHOWDDL for LOB with a special option will show all the dependent objects, names and details about the table.

3.12.1. Syntax

```
SHOWDDL table-name, LOB DETAILS
```

3.12.2. Examples

- This example displays the details of the table t1ob1.

```
>>SHOWDDL tlob1, LOB DETAILS;
CREATE TABLE TRAFODION.SEABASE.TLOB1
```

```
  (
    C1                                INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , C2                                BLOB DEFAULT NULL NOT SERIALIZED
  , PRIMARY KEY (C1 ASC)
  )
;

LOB Metadata
============

CREATE TABLE TRAFODION.SEABASE.LOBMD_04239091936503896833
  (
    LOBNUM                            SMALLINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , STORAGETYPE                       SMALLINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , LOCATION                          VARCHAR(4096) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT NULL NOT DROPPABLE SERIALIZED
  , PRIMARY KEY (LOBNUM ASC)
  )
;

***********************************************
LobNum: 1

Data Storage
============

Location: /user/trafodion/lobs
DataFile: LOBP_04239091936503896833_0001

LOB Descriptor Handle
=====================

CREATE TABLE TRAFODION.SEABASE."LOBDescHandle_04239091936503896833_0001"
  (
    DESCPARTNKEY                      LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , NUMCHUNKS                         INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , LOBLEN                            LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  )
  STORE BY (DESCPARTNKEY ASC)
  SALT USING 8 PARTITIONS
;

LOB Descriptor Chunks
=====================

CREATE TABLE TRAFODION.SEABASE."LOBDescChunks_04239091936503896833_0001"
  (
    DESCPARTNKEY                      LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
```

```
SERIALIZED
  , DESCSYSKEY                        LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKNUM                          INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKLEN                          LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , INTPARAM                          LARGEINT DEFAULT NULL SERIALIZED
  , STRINGPARAM                       VARCHAR(400) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL SERIALIZED
  , PRIMARY KEY (DESCPARTNKEY ASC, DESCSYSKEY ASC, CHUNKNUM ASC)
  )
  SALT USING 8 PARTITIONS
;

--- SQL operation complete.
```

- This example displays the details of the table tlob130ext.

```
>>CREATE TABLE tlob130ext (c1 INT NOT NULL, c2 BLOB, c3 CLOB, c4 BLOB STORAGE
'EXTERNAL', primary key (c1));

--- SQL operation complete.

>>SHOWDDL tlob130ext, LOB DETAILS;

CREATE TABLE TRAFODION.SCH.TLOB130EXT
  (
    C1                                INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , C2                                BLOB DEFAULT NULL NOT SERIALIZED
  , C3                                CLOB DEFAULT NULL NOT SERIALIZED
  , C4                                BLOB DEFAULT NULL NOT SERIALIZED
  , PRIMARY KEY (C1 ASC)
  )
;

LOB Metadata
============

CREATE TABLE TRAFODION.SCH.LOBMD__04474425229029907479
  (
    LOBNUM                            SMALLINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , STORAGETYPE                       SMALLINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , LOCATION                          VARCHAR(4096) CHARACTER SET ISO88591 COLLATE
DEFAULT NO DEFAULT NOT NULL NOT DROPPABLE SERIALIZED
  , PRIMARY KEY (LOBNUM ASC)
  )
;

**********************************************
```

```
LobNum: 1

Data Storage
============

Location: /user/trafodion/lobs
DataFile: LOBP_04474425229029907479_0001

LOB Descriptor Handle
=====================

CREATE TABLE TRAFODION.SCH."LOBDescHandle__04474425229029907479_0001"
  (
    DESCPARTNKEY                    LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , NUMCHUNKS                       INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , LOBLEN                          LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  )
  STORE BY (DESCPARTNKEY ASC)
  SALT USING 8 PARTITIONS
;

LOB Descriptor Chunks
=====================

CREATE TABLE TRAFODION.SCH."LOBDescChunks__04474425229029907479_0001"
  (
    DESCPARTNKEY                    LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DESCSYSKEY                      LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKNUM                        INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKLEN                        LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DATAOFFSET                      LARGEINT DEFAULT NULL SERIALIZED
  , STRINGPARAM                     VARCHAR(400) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL SERIALIZED
  , PRIMARY KEY (DESCPARTNKEY ASC, DESCSYSKEY ASC, CHUNKNUM ASC)
  )
  SALT USING 8 PARTITIONS
;

***********************************************
LobNum: 2

Data Storage
============

Location: /user/trafodion/lobs
DataFile: LOBP_04474425229029907479_0002
```

```
LOB Descriptor Handle
=====================

CREATE TABLE TRAFODION.SCH."LOBDescHandle__04474425229029907479_0002"
  (
    DESCPARTNKEY                     LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , NUMCHUNKS                        INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , LOBLEN                           LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  )
  STORE BY (DESCPARTNKEY ASC)
  SALT USING 8 PARTITIONS
;

LOB Descriptor Chunks
=====================

CREATE TABLE TRAFODION.SCH."LOBDescChunks__04474425229029907479_0002"
  (
    DESCPARTNKEY                     LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DESCSYSKEY                       LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKNUM                         INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKLEN                         LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DATAOFFSET                       LARGEINT DEFAULT NULL SERIALIZED
  , STRINGPARAM                      VARCHAR(400) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL SERIALIZED
  , PRIMARY KEY (DESCPARTNKEY ASC, DESCSYSKEY ASC, CHUNKNUM ASC)
  )
  SALT USING 8 PARTITIONS
;

**********************************************
LobNum: 3

Data Storage
============

<External HDFS location>
<External HDFS file>

LOB Descriptor Handle
=====================

CREATE TABLE TRAFODION.SCH."LOBDescHandle__04474425229029907479_0003"
  (
    DESCPARTNKEY                     LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , NUMCHUNKS                        INT NO DEFAULT NOT NULL NOT DROPPABLE
```

```
SERIALIZED
  , LOBLEN                          LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  )
  STORE BY (DESCPARTNKEY ASC)
  SALT USING 8 PARTITIONS
;

LOB Descriptor Chunks
=====================

CREATE TABLE TRAFODION.SCH."LOBDescChunks__04474425229029907479_0003"
  (
    DESCPARTNKEY                    LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DESCSYSKEY                      LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKNUM                        INT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , CHUNKLEN                        LARGEINT NO DEFAULT NOT NULL NOT DROPPABLE
SERIALIZED
  , DATAOFFSET                      LARGEINT DEFAULT NULL SERIALIZED
  , STRINGPARAM                     VARCHAR(400) CHARACTER SET ISO88591 COLLATE
DEFAULT DEFAULT NULL SERIALIZED
  , PRIMARY KEY (DESCPARTNKEY ASC, DESCSYSKEY ASC, CHUNKNUM ASC)
  )
  SALT USING 8 PARTITIONS
;

--- SQL operation complete.
```

3.13. Get Lob Statistics for a LOB Table

There are two ways to get lob statistics for a lob table:

- Get Statement: the lob information is formatted for human readability.
- Select Statement: the lob information is formatted for machine readability.

3.13.1. Get Statement

Syntax

```
GET LOB STATS FOR TABLE table-name;
```

Examples

- This Get Statement displays statistics for the table tlob130gt2.

```
>>CREATE TABLE tlob130gt2 (c1 INT NOT NULL, c2 BLOB, c3 CLOB, c4 BLOB STORAGE
'EXTERNAL', PRIMARY KEY (c1));

--- SQL operation complete.

>>GET LOB STATS FOR TABLE tlob130gt2;

Lob Information for table: "TRAFODION".LOB130.TLOB130GT2
========================

ColumnName:  C2
Lob Location:  /user/trafodion/lobs
LOB Data File:  LOBP_07468755986685501835_0001
LOB EOD:  0
LOB Used Len:  0
ColumnName:  C3
Lob Location:  /user/trafodion/lobs
LOB Data File:  LOBP_07468755986685501835_0002
LOB EOD:  0
LOB UsedLen:  0
ColumnName:  C4
Lob Location:  External HDFS Location
LOB Data File:  External HDFS File
LOB EOD:  0
LOB Used Len:  0
--- SQL operation complete.
```

3.13.2. Select Statement

Syntax

```
SELECT * FROM TABLE(LOB STATS table-name);
```

Examples

- This Select Statement displays statistics for the table tlob130gt.

```
>>CREATE TABLE tlob130gt (c1 INT NOT NULL, c2 BLOB, c3 CLOB, c4 BLOB, PRIMARY KEY
(c1));

--- SQL operation complete.

>>SELECT left(trim(catalog_name) || '.' || trim(schema_name) || '.' ||
trim(object_name) , 20),
left(trim(column_name),5),
left(trim(lob_location),5),
left(trim(lob_data_file),20),
LOB_DATA_FILE_SIZE_EOD,
LOB_DATA_FILE_SIZE_USED
FROM TABLE(lob stats(tlob130gt));

(EXPR)      (EXPR)     (EXPR)     (EXPR)     LOB_DATA_FILE_SIZE_EOD
LOB_DATA_FILE_SIZE_USED
------      ------     ------     ------     ----------------------
----------------------

TRAFODION.SCH.TLOB13   C2   /user/trafodion/lobs   LOBP_044744252290302   15   10
TRAFODION.SCH.TLOB13   C3   /user/trafodion/lobs   LOBP_044744252290302   15   10
TRAFODION.SCH.TLOB13   C4   /user/trafodion/lobs   LOBP_044744252290302   45   30

--- 3 row(s) selected.
```

- This Select Statement displays statistics for the table tlob130gt2.

```
>>CREATE TABLE tlob130gt2 (c1 INT NOT NULL, c2 BLOB, c3 CLOB, c4 BLOB STORAGE
'EXTERNAL', PRIMARY KEY (c1));

--- SQL operation complete.

>>SELECT left(trim(catalog_name) || '.' || trim(schema_name) || '.' ||
trim(object_name), 20),
left(trim(column_name),5),
left(trim(lob_location),15),
left(trim(lob_data_file),20),
LOB_DATA_FILE_SIZE_EOD,
LOB_DATA_FILE_SIZE_USED
FROM TABLE(lob stats(tlob130gt2));

(EXPR)      (EXPR)     (EXPR)     (EXPR)     LOB_DATA_FILE_SIZE_EOD
LOB_DATA_FILE_SIZE_USED
------      ------     ------     ------     ----------------------
----------------------

TRAFODION.SCH.TLOB13   C2   /user/trafodion/lobs   LOBP_044744252290300   0   0
TRAFODION.SCH.TLOB13   C3   /user/trafodion/lobs   LOBP_044744252290300   0   0
TRAFODION.SCH.TLOB13   C4   External HDFS Location External HDFS File     0   0

--- 3 row(s) selected.
```