10
Distributed LOBs

This section describes the ways in which you can work with LOB data in remote tables.

Distributed LOBs are LOBs that are fetched from one server to another, and may optionally be returned to the client. Distributed LOBs can be persistent or temporary LOBs for both reference and value LOB columns.

In sharding, a table is horizontally partitioned with subsets of rows in a table stored in different sharded databases. The client connects to the coordinator database, which in turn works with shards to provide a consolidated view of a table. Sharded LOBs are an extension of Distributed LOBs. LOB data between different shards is transported as distributed LOBs and the result is provided to the client through the coordinator database.

All Persistent LOBs and Temporary LOBs originating from JSON support Distributed and Sharded LOBs.

- Working with Remote LOBs in SQL and PL/SQL
  This section describes the SQL and PL/SQL functions that are supported on remote LOBs.
- Using the Data Interface on Remote LOBs
  The data interface enables you to bind and define a `CHARACTER` buffer for a `CLOB` column and a `RAW` buffer for a `BLOB` column. This interface is supported for remote LOB columns too.
- Working with Remote Locators
  You can select a persistent LOB locator from a remote table into a local variable and this can be done in any programmatic interface like PL/SQL, JDBC or OCI. The remote columns can be of type `BLOB`, `CLOB` or `NCLOB`.

See Also:

Sharding with LOBs

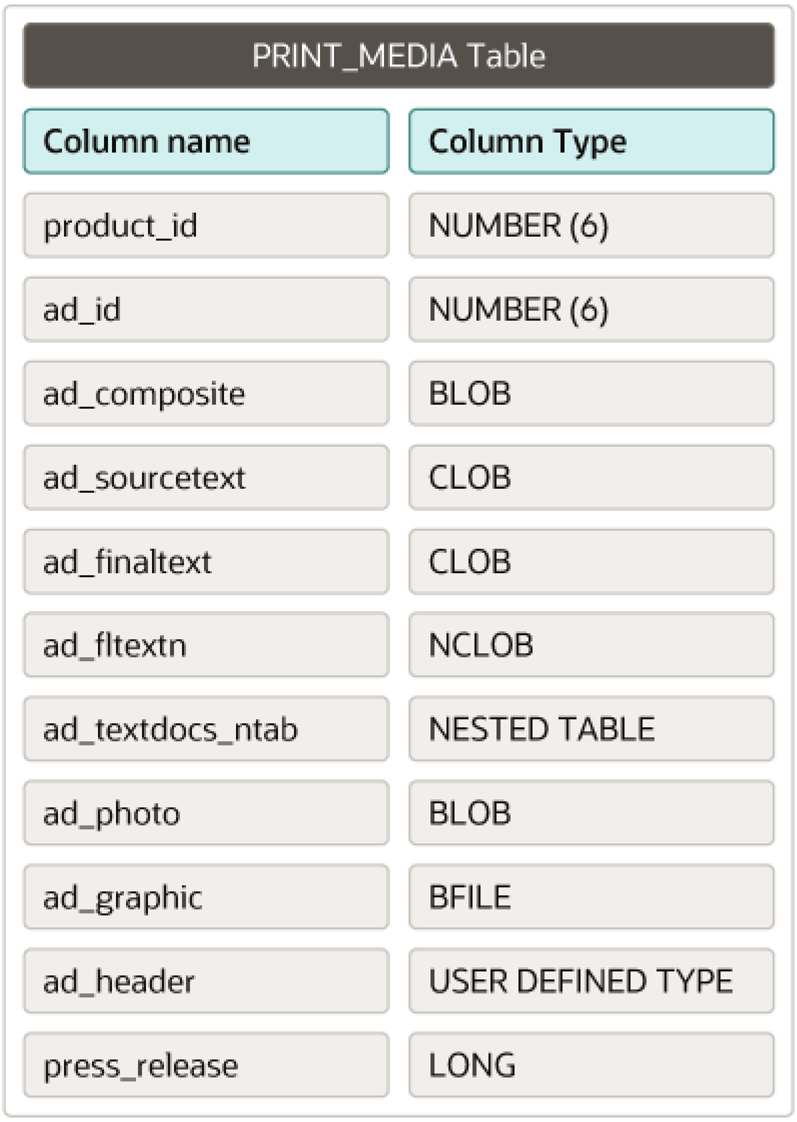
10.1 Working with Remote LOBs in SQL and PL/SQL

This section describes the SQL and PL/SQL functions that are supported on remote LOBs.

SQL Functions

All the SQL built-in functions and user-defined functions that are supported on local LOBs and BFILEs, are also supported on remote LOBs and BFILEs, as long as the final value returned by the nested functions is not a LOB type. This includes functions for remote persistent and temporary LOBs and for BFILEs.

Most of the examples in the following sections use `print_media` table. Following is the structure of the table:



PRINT_MEDIA Table

Column name	Column Type
product_id	NUMBER (6)
ad_id	NUMBER (6)
ad_composite	BLOB
ad_sourcetext	CLOB
ad_finaltext	CLOB
ad_fltextn	NCLOB
ad_textdocs_ntab	NESTED TABLE
ad_photo	BLOB
ad_graphic	BFILE
ad_header	USER DEFINED TYPE
press_release	LONG

Built-in SQL functions, which are executed on a remote site, can be part of any SQL statement, like `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. For example:

```
SELECT LENGTH(ad_sourcetext) FROM print_media@remote_site -- CLOB
SELECT LENGTH(ad_fltextn) FROM print_media@remote_site;    -- NCLOB
SELECT LENGTH(ad_composite) FROM print_media@remote_site; -- BLOB
SELECT product_id from print_media@remote_site WHERE LENGTH(ad_sourcetext) >
3;

UPDATE print_media@remote_site SET product_id = 2 WHERE LENGTH(ad_sourcetext)
> 3;

SELECT TO_CHAR(foo@dbs2(...)) FROM dual@dbs2;
-- where foo@dbs2 returns a temporary LOB
```

PL/SQL functions

Built-in and user-defined PL/SQL functions that are executed on the remote site and operate on remote LOBs and BFILEs are allowed, as long as the final value returned by nested functions is not a LOB.

```
SELECT product_id FROM print_media@dbs2 WHERE foo@dbs2(ad_sourcetext, 'aa') >
0;
-- foo is a user-define function returning a NUMBER

DELETE FROM print_media@dbs2 WHERE DBMS_LOB.GETLENGTH@dbs2(ad_graphic) = 0;
```

Restrictions on Remote User Defined Functions

The SQL and PL/SQL functions fall under the following non-comprehensive list of categories:

- SQL functions that are not supported on LOBs
  The SQL functions like the `DECODE` function, which are not supported for LOBs, are not supported on remote LOBs as well.
- Functions that accept exactly one LOB argument (where all the other arguments are of non-LOB data types) and does not return a LOB
  The functions, like the LENGTH function, are supported. For example:

  ```
  SELECT LENGTH(ad_composite) FROM print_media@remote_site;
  SELECT LENGTH(ad_header.logo) FROM print_media@remote_site; -- LOB in
  object
  SELECT product_id from print_media@remote_site WHERE LENGTH(ad_sourcetext)
  > 3;
  ```

- Functions that return a LOB

  These functions may return the original LOB or produce a temporary LOB. These functions can be performed on the remote site, as long as the result returned to the local site is not a LOB.

  – Functions returning a temporary LOB are: `REPLACE`, `SUBSTR`, `CONCAT`, ||, `TRIM`, `LTRIM`, `RTRIM`, `LOWER`, `UPPER`, `NLS_LOWER`, `NLS_UPPER`, `LPAD`, and `RPAD`.

  – Functions returning the original LOB locator are: `NVL`, `DECODE`, and `CASE`.

  For example, **the following statements are supported:**

  ```
  SELECT TO_CHAR(CONCAT(ad_sourcetext, ad_sourcetext)) FROM
  print_media@remote_site;
  SELECT TO_CHAR(SUBSTR(ad_fltextnfs, 1, 3)) FROM print_media@remote_site;
  ```

  But the **following statements are not supported:**

  ```
  SELECT CONCAT(ad_sourcetext, ad_sourcetext) FROM print_media@remote_site;
  SELECT SUBSTR(ad_sourcetext, 1, 3) FROM print_media@remote_site;
  ```

- Functions that take in more than one LOB argument:

  These are: `INSTR`, `LIKE`, `REPLACE`, `CONCAT`, ||, `SUBSTR`, `TRIM`, `LTRIM`, `RTRIM`, `LPAD`, and `RPAD`. All these functions are relevant only for `CLOB`s and `NCLOB`s.

These functions are supported only if all the LOB arguments are on the same `dblink`. For example, **the following is supported:**

```
SELECT TO_CHAR(CONCAT(ad_sourcetext, ad_sourcetext)) FROM
print_media@remote_site; -- CLOB
SELECT TO_CHAR(CONCAT(ad_fltextn, ad_fltextn)) FROM
print_media@remote_site; -- NCLOB
```

But the following is not supported

```
SELECT TO_CHAR(CONCAT(a.ad_sourcetext, b.ad_sourcetext)) FROM
print_media@db1 a, print_media@db2 b WHERE a.product_id = b.product_id;
```

- PL/SQL functions operating on LOBs:
  A function in one `dblink` cannot operate on LOB data in another dblink. For example, the following statement is not supported:

  ```
  SELECT a.product_id FROM print_media@dbs1 a, print_media@dbs2 b WHERE
  CONTAINS@dbs1(b.ad_sourcetext, 'aa') >0;
  ```

- Multiple LOBs in a query block:
  One query block cannot contain tables and functions at different `dblinks`. For example, the following statement is not supported

  ```
  SELECT a.product_id FROM print_media@dbs2 a, print_media@dbs3 b
      WHERE CONTAINS@dbs2(a.ad_sourcetext, 'aa') > 0 AND
      foo@dbs3(b.ad_sourcetext) > 0;
  --  foo is a user-defined function in dbs3
  ```

- LOB operators and columns are supported if they are in a `SELECT` list and `where` clause in a join query.
- Oracle-provided PL/SQL functions and procedures can return LOB locators.
- Only remote LOBs support SQL operators returning temporary LOBs.
- Only the views supplied by Oracle, support returning LOBs.

10.2 Using the Data Interface on Remote LOBs

The data interface enables you to bind and define a `CHARACTER` buffer for a `CLOB` column and a `RAW` buffer for a `BLOB` column. This interface is supported for remote LOB columns too.

The advantage of using the data interface over using LOB locators is that it makes only one round-trip to the remote server to fetch the LOB data. If used in as part of an array bind or define, it will use only one round-trip for the entire array operation.

The examples discussed in the book use the `print_media` table created in the following two schemas: `dbs1` and `dbs2`. The `CLOB` column of the `print_media` table used in the examples shown is `ad_finaltext`. The examples provided for PL/SQL, OCI, and Java in the following sections use binds and defines for this one column, but multiple columns can also be accessed. Following is the functionality supported:

- You can bind and define a `CLOB` as `VARCHAR2` and a `BLOB` as `RAW`.
- Array binds and defines are supported.

- PL/SQL
- JDBC
- OCI
- Remote LOBs

PL/SQL

This section describes how to use the remote data interface with LOBs in PL/SQL.

The data interface only supports data of size less than 32KB in PL/SQL. The following snippet shows a PL/SQL example:

```
CONNECT pm/pm
declare
  my_ad varchar(6000) := lpad('b', 6000, 'b');
BEGIN
  INSERT INTO print_media@dbs2(product_id, ad_id, ad_finaltext)
       VALUES (10000, 10, my_ad);
  -- Reset the buffer value
  my_ad := 'a';
  SELECT ad_finaltext INTO my_ad FROM print_media@dbs2
       WHERE product_id = 10000;
END;
/
```

If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, `my_ad` has to be of type `RAW`. If the LOB is greater than 32KB - 1 in size, then PL/SQL raises a truncation error and the contents of the buffer are undefined.

JDBC

This section demonstrates how to use the remote data interface with LOBs in JDBC.

The following code snippets work with all JDBC drivers:

Bind:

This is for the non-streaming mode:

```
...
String sql = "insert into print_media@dbs2 (product_id, ad_id, ad_final_text)" +
             " values (:1, :2, :3)";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt( 1, 2 );
    pstmt.setInt( 2, 20);
    pstmt.setString( 3, "Java string" );
    int rows = pstmt.executeUpdate();
...
```

Note: Oracle supports the non-streaming mode for strings of size up to 2 GB. However, the memory size of your computer may be a limiting factor.

For the streaming mode, the same code as the preceding works, except that the `setString()` statement is replaced by one of the following:

```
pstmt.setCharacterStream( 3, new LabeledReader(), 1000000 );
pstmt.setAsciiStream( 3, new LabeledAsciiInputStream(), 1000000 );
```

Note: You can use the streaming interface to insert Gigabyte sized character and binary data into a LOB column.

Here, `LabeledReader()` and `LabeledAsciiInputStream()` produce character and ASCII streams respectively. If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, then the preceding example works if the bind is of type `RAW`:

```
pstmt.setBytes( 3, <some byte[] array> );

pstmt.setBinaryStream( 3, new LabeledInputStream(), 1000000 );
```

Here, `LabeledInputStream()` produces a binary stream.

Define:

For non-streaming mode:

```
OracleStatement stmt = (OracleStatement)(conn.createStatement());
  stmt.defineColumnType( 1, Types.VARCHAR );
  ResultSet rst = stmt.executeQuery("select ad_finaltext from print_media@dbs2" );
  while( rst.next() )
     {
       String s = rst.getString( 1 );
       System.out.println( s );
     }
```

Note: If the LOB size is greater than 32767 bytes, the data is truncated and no error is thrown.

For streaming mode:

```
OracleStatement stmt = (OracleStatement)(conn.createStatement());
  stmt.defineColumnType( 1, Types.LONGVARCHAR );
  ResultSet rs = stmt.executeQuery("select ad_finaltext from print_media@dbs2" );
  while(rs.next()) {
    Reader reader = rs.getCharacterStream( 1 );
    int data = 0;
    data = reader.read();
    while( -1 != data ){
      System.out.print( (char)(data) );
      data = reader.read();
    }
    reader.close();
  }
```

Note: Specifying the datatype as `LONGVARCHAR` lets you select the entire LOB. If the define type is set as `VARCHAR` instead of `LONGVARCHAR`, the data will be truncated at 32k.

If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, then the preceding examples work if the define is of type `LONGVARBINARY`:

```
...
   OracleStatement stmt = (OracleStatement)conn.createStatement();

   stmt.defineColumnType( 1, Types.INTEGER );
   stmt.defineColumnType( 2, Types.LONGVARBINARY );

   ResultSet rset = stmt.executeQuery("SELECT ID, LOBCOL FROM LOBTAB@MYSELF");

   while(rset.next())
    {
     /* using getBytes() */
     /*
     byte[] b = rset.getBytes("LOBCOL");
     System.out.println("ID: " + rset.getInt("ID") + "  length: " + b.length);
     */

        /* using getBinaryStream() */
        InputStream byte_stream = rset.getBinaryStream("LOBCOL");
        byte [] b = new byte [100000];
        int b_len = byte_stream.read(b);
        System.out.println("ID: " + rset.getInt("ID") + "  length: " + b_len);

        byte_stream.close();
    }
...
```

OCI

This section demonstrates how to use the remote data interface with LOBs in OCI.

The data interface only supports data of size less than 2 gigabytes (the maximum value possible of a variable declared as `sb4`) for OCI. The following pseudocode can be enhanced to be a part of an OCI program:

```
...
text *sql = (text *)"insert into print_media@dbs2
                    (product_id, ad_id, ad_finaltext)
                    values (:1, :2, :3)";
OCIStmtPrepare(...);
OCIBindByPos(...); /* Bind data for positions 1 and 2
                    * which are independent of LOB */
OCIBindByPos(stmthp, &bndhp[2], errhp, (ub4) 3,
             (dvoid *) charbuf1, (sb4) len_charbuf1, SQLT_CHR,
             (dvoid *) 0, (ub2 *)0, (ub2 *)0, 0, 0, OCI_DEFAULT);
OCIStmtExecute(...);

...

text *sql = (text *)"select ad_finaltext from print_media@dbs2
                    where product_id = 10000";
OCIStmtPrepare(...);
OCIDefineByPos(stmthp, &dfnhp[2], errhp, (ub4) 1,
             (dvoid *) charbuf2, (sb4) len_charbuf2, SQLT_CHR,
             (dvoid *) 0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
OCIStmtExecute(...);
...
```

For a `BLOB` column, you must use the `SQLT_BIN` type. For example, if you define the `ad_finaltext` column as a `BLOB` column instead of a `CLOB` column, then you must bind and define the column data using the `SQLT_BIN` type. If the LOB is greater than `2GB - 1` bytes in size, then OCI raises a truncation error and the contents of the buffer are undefined.

Remote LOBs

This section discusses the restrictions on the usage of Data Interface on Remote LOBs.

Certain syntax is not supported for remote LOBs.

- Queries involving more than one database are not supported:

  ```
  SELECT t1.lobcol, a2.lobcol FROM t1, t2.lobcol@dbs2 a2 WHERE
  LENGTH(t1.lobcol) = LENGTH(a2.lobcol);
  ```

  Neither is this query (in a PL/SQL block):

```
SELECT t1.lobcol INTO varchar_buf1 FROM t1@dbs1
UNION ALL
SELECT t2.lobcol INTO varchar_buf2 FROM t2@dbs2;
```

- Only binds and defines for data going into remote persistent LOB columns are supported, so that parameter passing in PL/SQL where `CHAR` data is bound or defined for remote LOBs is not allowed because this could produce a remote temporary LOB, which are not supported. These statements all produce errors:

  ```
  SELECT foo() INTO varchar_buf FROM table1@dbs2; -- foo returns a LOB

  SELECT foo()@dbs INTO char_val FROM DUAL; -- foo returns a LOB

  SELECT XMLType().getclobval INTO varchar_buf FROM table1@dbs2;
  ```

- If the remote object is a view such as

  ```
  CREATE VIEW v AS SELECT foo() a FROM ... ; -- foo returns a LOB
  /* The local database then tries to get the CLOB data and returns an error */
  SELECT a INTO varchar_buf FROM v@dbs2;
  ```

  This returns an error because it produces a remote temporary LOB, which is not supported.

- `RETURNING INTO` does not support implicit conversions between `CHAR` and `CLOB`.
- PL/SQL parameter passing is not allowed where the actual argument is a LOB type and the remote argument is a `VARCHAR2`, `NVARCHAR2`, `CHAR`, `NCHAR`, or `RAW`.

- Remote Data Interface Example in PL/SQL
  This section describes how to use the remote data interface with LOBs in PL/SQL.
- Remote Data Interface Examples in JDBC
  This section demonstrates how to use the remote data interface with LOBs in JDBC.
- Remote Data Interface Example in OCI
  This section demonstrates how to use the remote data interface with LOBs in OCI.
- Restrictions for Data Interface on Remote LOBs
  This section discusses the restrictions on the usage of Data Interface on Remote LOBs.

See Also:

- Oracle Database JDBC Developer's Guide
- Data Interface for LOBs

10.2.1 Remote Data Interface Example in PL/SQL

This section describes how to use the remote data interface with LOBs in PL/SQL.

The data interface only supports data of size less than 32KB in PL/SQL. The following snippet shows a PL/SQL example:

```
CONNECT pm/pm
declare
```

```
  my_ad varchar(6000) := lpad('b', 6000, 'b');
BEGIN
  INSERT INTO print_media@dbs2(product_id, ad_id, ad_finaltext)
       VALUES (10000, 10, my_ad);
  -- Reset the buffer value
  my_ad := 'a';
  SELECT ad_finaltext INTO my_ad FROM print_media@dbs2
       WHERE product_id = 10000;
END;
/
```

If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, `my_ad` has to be of type `RAW`. If the LOB is greater than 32KB - 1 in size, then PL/SQL raises a truncation error and the contents of the buffer are undefined.

10.2.2 Remote Data Interface Examples in JDBC

This section demonstrates how to use the remote data interface with LOBs in JDBC.

The following code snippets work with all JDBC drivers:

Bind:

This is for the non-streaming mode:

```
...
String sql = "insert into print_media@dbs2 (product_id, ad_id, ad_final_text)" +
             " values (:1, :2, :3)";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt( 1, 2 );
    pstmt.setInt( 2, 20);
    pstmt.setString( 3, "Java string" );
    int rows = pstmt.executeUpdate();
...
```

Note:

Oracle supports the non-streaming mode for strings of size up to 2 GB. However, the memory size of your computer may be a limiting factor.

For the streaming mode, the same code as the preceding works, except that the `setString()` statement is replaced by one of the following:

```
pstmt.setCharacterStream( 3, new LabeledReader(), 1000000 );
pstmt.setAsciiStream( 3, new LabeledAsciiInputStream(), 1000000 );
```

Note:

You can use the streaming interface to insert Gigabyte sized character and binary data into a LOB column.

Here, `LabeledReader()` and `LabeledAsciiInputStream()` produce character and ASCII streams respectively. If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, then the preceding example works if the bind is of type `RAW`:

```
pstmt.setBytes( 3, <some byte[] array> );

pstmt.setBinaryStream( 3, new LabeledInputStream(), 1000000 );
```

Here, `LabeledInputStream()` produces a binary stream.

Define:

For non-streaming mode:

```
OracleStatement stmt = (OracleStatement)(conn.createStatement());
  stmt.defineColumnType( 1, Types.VARCHAR );
  ResultSet rst = stmt.executeQuery("select ad_finaltext from print_media@dbs2" );
  while( rst.next() )
     {
       String s = rst.getString( 1 );
       System.out.println( s );
     }
```

Note:

If the LOB size is greater than 32767 bytes, the data is truncated and no error is thrown.

For streaming mode:

```
OracleStatement stmt = (OracleStatement)(conn.createStatement());
  stmt.defineColumnType( 1, Types.LONGVARCHAR );
  ResultSet rs = stmt.executeQuery("select ad_finaltext from print_media@dbs2" );
  while(rs.next()) {
    Reader reader = rs.getCharacterStream( 1 );
    int data = 0;
    data = reader.read();
    while( -1 != data ){
      System.out.print( (char)(data) );
      data = reader.read();
    }
    reader.close();
  }
```

Note:

Specifying the datatype as `LONGVARCHAR` lets you select the entire LOB. If the define type is set as `VARCHAR` instead of `LONGVARCHAR`, the data will be truncated at 32k.

If `ad_finaltext` were a `BLOB` column instead of a `CLOB`, then the preceding examples work if the define is of type `LONGVARBINARY`:

```
...
   OracleStatement stmt = (OracleStatement)conn.createStatement();

   stmt.defineColumnType( 1, Types.INTEGER );
   stmt.defineColumnType( 2, Types.LONGVARBINARY );

   ResultSet rset = stmt.executeQuery("SELECT ID, LOBCOL FROM LOBTAB@MYSELF");

   while(rset.next())
```

```
    {
     /* using getBytes() */
     /*
     byte[] b = rset.getBytes("LOBCOL");
     System.out.println("ID: " + rset.getInt("ID") + "  length: " + b.length);
     */

        /* using getBinaryStream() */
        InputStream byte_stream = rset.getBinaryStream("LOBCOL");
        byte [] b = new byte [100000];
        int b_len = byte_stream.read(b);
        System.out.println("ID: " + rset.getInt("ID") + "  length: " + b_len);

        byte_stream.close();
    }
...
```

See Also:

Oracle Database JDBC Developer's Guide

10.2.3 Remote Data Interface Example in OCI

This section demonstrates how to use the remote data interface with LOBs in OCI.

The data interface only supports data of size less than 2 gigabytes (the maximum value possible of a variable declared as sb4) for OCI. The following pseudocode can be enhanced to be a part of an OCI program:

```
...
text *sql = (text *)"insert into print_media@dbs2
                    (product_id, ad_id, ad_finaltext)
                    values (:1, :2, :3)";
OCIStmtPrepare(...);
OCIBindByPos(...); /* Bind data for positions 1 and 2
                     * which are independent of LOB */
OCIBindByPos(stmthp, &bndhp[2], errhp, (ub4) 3,
             (dvoid *) charbuf1, (sb4) len_charbuf1, SQLT_CHR,
             (dvoid *) 0, (ub2 *)0, (ub2 *)0, 0, 0, OCI_DEFAULT);
OCIStmtExecute(...);

...

text *sql = (text *)"select ad_finaltext from print_media@dbs2
                    where product_id = 10000";
OCIStmtPrepare(...);
OCIDefineByPos(stmthp, &dfnhp[2], errhp, (ub4) 1,
             (dvoid *) charbuf2, (sb4) len_charbuf2, SQLT_CHR,
             (dvoid *) 0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
OCIStmtExecute(...);
...
```

For a BLOB column, you must use the SQLT_BIN type. For example, if you define the ad_finaltext column as a BLOB column instead of a CLOB column, then you must bind and define the column data using the SQLT_BIN type. If the LOB is greater than 2GB - 1 bytes in size, then OCI raises a truncation error and the contents of the buffer are undefined.

10.2.4 Restrictions for Data Interface on Remote LOBs

This section discusses the restrictions on the usage of Data Interface on Remote LOBs.

Certain syntax is not supported for remote LOBs.

- Queries involving more than one database are not supported:

  ```
  SELECT t1.lobcol, a2.lobcol FROM t1, t2.lobcol@dbs2 a2 WHERE
  LENGTH(t1.lobcol) = LENGTH(a2.lobcol);
  ```

  Neither is this query (in a PL/SQL block):

  ```
  SELECT t1.lobcol INTO varchar_buf1 FROM t1@dbs1
  UNION ALL
  SELECT t2.lobcol INTO varchar_buf2 FROM t2@dbs2;
  ```

- `RETURNING INTO` does not support implicit conversions between `CHAR` and `CLOB`.
- PL/SQL parameter passing is not allowed where the actual argument is a LOB type and the remote argument is a `VARCHAR2`, `NVARCHAR2`, `CHAR`, `NCHAR`, or `RAW`.

10.3 Working with Remote Locators

You can select a persistent LOB locator from a remote table into a local variable and this can be done in any programmatic interface like PL/SQL, JDBC or OCI. The remote columns can be of type `BLOB`, `CLOB` or `NCLOB`.

The following SQL statement is the basis for all the examples with remote LOB locator in this chapter.

```
CREATE TABLE lob_tab (c1 NUMBER, c2 CLOB);
```

In the following example, the table `lob_tab` (with columns `c2` of type `CLOB` and `c1` of type number) defined in the remote database is accessible using database link `db2` and a local `CLOB` variable `lob_var1`.

```
SELECT c2 INTO lob_var1 FROM lob_tab@db2 WHERE c1=1;
SELECT c2 INTO lob_var1 FROM lob_tab@db2 WHERE c1=1 for update;
```

In PL/SQL, the function `dbms_lob.isremote` can be used to check if a particular LOB belongs to a remote table. Similarly, in `OCI`, you can use the `OCI_ATTR_LOB_REMOTE` attribute of `OCILobLocator` to check if a particular LOB belongs to a remote table. For example,

```
IF(dbms_lob.isremote(lob_var1)) THEN
dbms_output.put_line('LOB locator is remote)
ENDIF;
```

- Using Local and Remote Locators as Bind with Queries and DML on Remote Tables
  This section discusses the bind values for queries and DML statements.
- Using Remote Locator
  This section demonstrates the usage of remote locator in PL/SQL and with OCILOB API with examples.

- Using Remote Locators with OCILOB API
  Most `OCILOB` APIs support operations on remote `LOB` locators. The following list of `OCILOB` functions returns an error when a remote `LOB` locator is passed to them:
- Restrictions when using remote LOB locators
  Remote LOB locators have the following restrictions:

See Also:

- ISREMOTE Function
- OCI_ATTR_LOB_REMOTE Attribute

10.3.1 Using Local and Remote Locators as Bind with Queries and DML on Remote Tables

This section discusses the bind values for queries and DML statements.

For the Queries and DMLs (`INSERT`, `UPDATE`, `DELETE`) with bind values, the following four cases are possible. The first case involves local tables and locators and is the standard LOB functionality, while the other three cases are part of the distributed LOBs functionality and have restrictions listed at the end of this section.

- Local table with local locator as bind value.
- Local table with remote locator as bind value
- Remote table with local locator as bind value
- Remote table with remote locator as bind value

Queries of the following form which use a remote LOB locator as a bind value are supported:

```
SELECT name FROM lob_tab@db2 WHERE length(c1)=length(:lob_v1);
```

In the above query, `c1` is an LOB column and `lob_v1` is a remote locator.

DMLs of the following forms using a remote LOB locator are supported. Here, the bind values can be local, remote persistent, or temporary LOB locators.

```
UPDATE lob_tab@db2 SET c1=:lob_v1;
```

```
INSERT into lob_tab@db2 VALUES (:1, :2);
```

You can pass a remote locator to most built-in SQL functions such as `LENGTH, INSTR, SUBSTR`, and `UPPER`. For example:

```
Var lob1 CLOB;
BEGIN
    SELECT c2 INTO lob1 FROM lob_tab@db2 WHERE c1=1;
END;
/
SELECT LENGTH(:lob1) FROM DUAL;
```

Note:

DMLs with `returning` clause are not supported on remote tables for both scalar and LOB columns.

10.3.2 Using Remote Locator

This section demonstrates the usage of remote locator in PL/SQL and with OCILOB API with examples.

- PL/SQL
- OCILOB API

PL/SQL

A remote locator can be passed as a parameter to built in PL/SQL functions like `LENGTH`, `INSTR`, `SUBSTR`, `UPPER` and so on which accepts LOB as input. For example,

```
DECLARE
    substr_data VARCHAR2(4000);
    remote_loc CLOB;
BEGIN
    SELECT c2 into remote_loc
    FROM lob_tab@db2 WHERE c1=1;
    substr_data := substr(remote_loc, position, length)
END;
```

All `DBMS_LOB` APIs other than the APIs targeted for BFILEs support operations on remote LOB locators.

The following example shows how to pass remote locator as input to `dbms_lob` operations.

```
DECLARE
  lob CLOB;
  buf VARCHAR2(120) := 'TST';
  amt NUMBER(2);
  len NUMBER(2);
BEGIN
  amt :=30;
  SELECT c2 INTO lob FROM lob_tab@db2 WHERE c1=3 FOR UPDATE;
  DBMS_LOB.WRITE(lob, amt, 1, buf);
  amt :=30;
  DBMS_LOB.READ(lob, amt, 1, buf);
  len := DBMS_LOB.GETLENGTH(lob);
  DBMS_OUTPUT.PUT_LINE(buf);
  DBMS_OUTPUT.PUT_LINE(amt);
  DBMS_OUTPUT.PUT_LINE('get length output = ' || len);
END;
/
```

OCILOB API

Most `OCILOB` APIs support operations on remote `LOB` locators. The following list of `OCILOB` functions returns an error when a remote `LOB` locator is passed to them:

- `OCILobLocatorAssign`
- `OCILobArrayRead()`
- `OCILobArrayWrite()`
- `OCILobLoadFromFile2()`

The following example shows how to pass a remote locator to `OCILOB` API.

```
void select_read_remote_lob()
{
  text *select_sql = (text *)"SELECT c2 lob_tab@dbs1 where c1=1";
  ub4 amtp = 10;
  ub4 nbytes = 0;
  ub4 loblen=0;
  OCILobLocator * one_lob;
  text strbuf[40];

 /* initialize single locator */
 OCIDescriptorAlloc(envhp, (dvoid **) &one_lob,
                 (ub4) OCI_DTYPE_LOB,
                 (size_t) 0, (dvoid **) 0)

 OCIStmtPrepare(stmthp, errhp, select_sql, (ub4)strlen((char*)select_sql),
                 (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);

 OCIDefineByPos(stmthp, &defp, errhp, (ub4) 1,
                     (dvoid *) &one_lob,
                     (sb4) -1,
                     (ub2) SQLT_CLOB,
                     (dvoid *) 0, (ub2 *) 0,
                     (ub2 *) 0, (ub4) OCI_DEFAULT));

 /* fetch the remote locator into the local variable one_lob */
 OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)0,
                 (OCISnapshot *)0, OCI_DEFAULT);

 /* Get the length of the remote LOB */
 OCILobGetLength(svchp, errhp,
                (OCILobLocator *) one_lob, (ub4 *)&loblen)

 printf("LOB length = %d\n", loblen);

 memset((void*)strbuf, (int)'\0', (size_t)40);

 / * Read the data from the remote LOB */
 OCILobRead(svchp, errhp, one_lob, &amtp,
                (ub4) 1, (dvoid *) strbuf, (ub4)& nbytes, (dvoid *)0,
                (OCICallbackLobRead) 0,
                (ub2) 0, (ub1) SQLCS_IMPLICIT));
 printf("LOB content = %s\n", strbuf);
```

```
}
```

See Also:

OCI Programmer's Guide, for the complete list of `OCILOB` APIs

10.3.3 Using Remote Locators with OCILOB API

Most `OCILOB` APIs support operations on remote `LOB` locators. The following list of `OCILOB` functions returns an error when a remote `LOB` locator is passed to them:

- `OCILobLocatorAssign`
- `OCILobArrayRead()`
- `OCILobArrayWrite()`
- `OCILobLoadFromFile2()`

The following example shows how to pass a remote locator to `OCILOB` API.

```
void select_read_remote_lob()
{
  text *select_sql = (text *)"SELECT c2 lob_tab@dbs1 where c1=1";
  ub4 amtp = 10;
  ub4 nbytes = 0;
  ub4 loblen=0;
  OCILobLocator * one_lob;
  text strbuf[40];

 /* initialize single locator */
 OCIDescriptorAlloc(envhp, (dvoid **) &one_lob,
                   (ub4) OCI_DTYPE_LOB,
                   (size_t) 0, (dvoid **) 0)

 OCIStmtPrepare(stmthp, errhp, select_sql, (ub4)strlen((char*)select_sql),
                   (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);

 OCIDefineByPos(stmthp, &defp, errhp, (ub4) 1,
                       (dvoid *) &one_lob,
                       (sb4) -1,
                       (ub2) SQLT_CLOB,
                       (dvoid *) 0, (ub2 *) 0,
                       (ub2 *) 0, (ub4) OCI_DEFAULT));

 /* fetch the remote locator into the local variable one_lob */
 OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)0,
                   (OCISnapshot *)0, OCI_DEFAULT);

 /* Get the length of the remote LOB */
 OCILobGetLength(svchp, errhp,
                (OCILobLocator *) one_lob, (ub4 *)&loblen)

 printf("LOB length = %d\n", loblen);

 memset((void*)strbuf, (int)'\0', (size_t)40);

 / * Read the data from the remote LOB */
 OCILobRead(svchp, errhp, one_lob, &amtp,
                (ub4) 1, (dvoid *) strbuf, (ub4)& nbytes, (dvoid *)0,
                (OCICallbackLobRead) 0,
                (ub2) 0, (ub1) SQLCS_IMPLICIT));
 printf("LOB content = %s\n", strbuf);

}
```

See Also:

OCI Programmer's Guide, for the complete list of `OCILOB` APIs

10.3.4 Restrictions when using remote LOB locators

Remote LOB locators have the following restrictions:

- You cannot select a remote temporary LOB locator into a local variable using the `SELECT` statement. For example,

  ```
  select substr(c2, 3, 1) from lob_tab@db2 where c1=1
  ```

  The preceding query returns an error.
- Remote LOB functionality is not supported for Index Organized tables (IOT). An attempt to get a locator from a remote IOT table will result in an error.
- Both the local database and the remote database have to be of Database release 12.2 or higher version.
- With distributed LOBs functionality, the tables that you use in the `from` clause or `where` clause should be collocated on the same database. If you use remote locators as bind variables in the `where` clauses, then they should belong to the same remote database. You cannot have one locator from one database (say, DB1) and another locator from another database (say, DB2) to be used as bind variables.
- Collocated tables or locators use the same database link. It is possible to have two different DB Links pointing to the same database. In the following example, both `dblink1` and `dblink2` point to the same remote database, but with different authentication methods. Oracle Database *does not* support such operations.

  ```
  INSERT into tab1@dblink1 SELECT * from tab2@dblink2;
  ```
- Any `DBMS_LOB` or `OCILob` APIs that accept two locators must obtain both the LOB locators through the same database link. Operations, as specified in the following example, are *not supported*:

  ```
  SELECT ad_sourcetext INTO clob1 FROM print_media@db1 WHERE product_id =
  10011;
  ```

```
SELECT ad_sourcetext INTO clob2 FROM print_media@db2 WHERE product_id =
10011;
DBMS_LOB.COPY(clob1, clob2, length(clob2));
```

- Bind values should be of the same LOB type as the column LOB type. For example, you must bind `NCLOB` locators to `NCLOB` columns and `CLOB` locators to `CLOB` columns. Implicit conversion between `NCLOB` and `CLOB` types is not supported in case of remote LOBs.
- DML statements with Array Binds are not supported when the bind operation involves a remote locator, or if the table involved is a remote table.
- You cannot select a `BFILE` column from a remote table into a local variable.