2 Persistent LOBs

A persistent LOB is a LOB instance that exists in a table row in the database. Persistent LOBs can be stored as SecureFiles or BasicFiles.

The term LOB can represent LOBs of either SecureFiles or BasicFiles type, unless the storage type is explicitly indicated. It can be either by name for both storage types, or by reference to archiving or linking, which only applies to the SecureFiles storage type. Oracle strongly recommends SecureFiles for storing and managing LOBs.

SecureFiles LOB storage is the default in the `CREATE TABLE` statement, if no storage type is explicitly specified. All new LOB columns use SecureFiles LOB storage by default, which is the recommended method for storing and managing LOBs. SecureFiles LOB storage is designed to provide great performance and scalability to meet or exceed the performance of traditional network file system. However, you must use BasicFiles LOB storage for LOB storage in tablespaces that are not managed with Automatic Segment Space Management (ASSM). SecureFiles LOBs can only be created in tablespaces managed with Automatic Segment Space Management (ASSM).

- Creating a Table with LOB Columns
  You can use the `CREATE TABLE` statement or an `ALTER TABLE ADD` column statement to create a new LOB column. This section introduces basic DDL operations on LOBs to get you started quickly.
- Inserting and Updating LOB Values in Tables
  Oracle Database provides various methods to insert and update the data available in LOB columns of database tables.
- Selecting LOB Values from Tables
  You can select a LOB into a Character Buffer, a RAW Buffer, or a LOB variable for performing read and write operations.
- Performing DML and Query Operations on LOBs in Nested Tables
  This section describes the `INSERT`, `UPDATE`, and `SELECT` operations on LOBs in Nested Tables. To update LOBs in a nested table, you must lock the row containing the LOB explicitly.
- Performing Parallel DDL, Parallel DML (PDML), and Parallel Query (PQ) Operations on LOBs
  Oracle supports parallel execution of the following operations when performed on partitioned tables with SecureFiles LOBs or BasicFiles LOBs.
- Sharding with LOBs
  LOBs can be used in a sharded environment. This section discusses the interfaces to support LOBs in sharded tables.

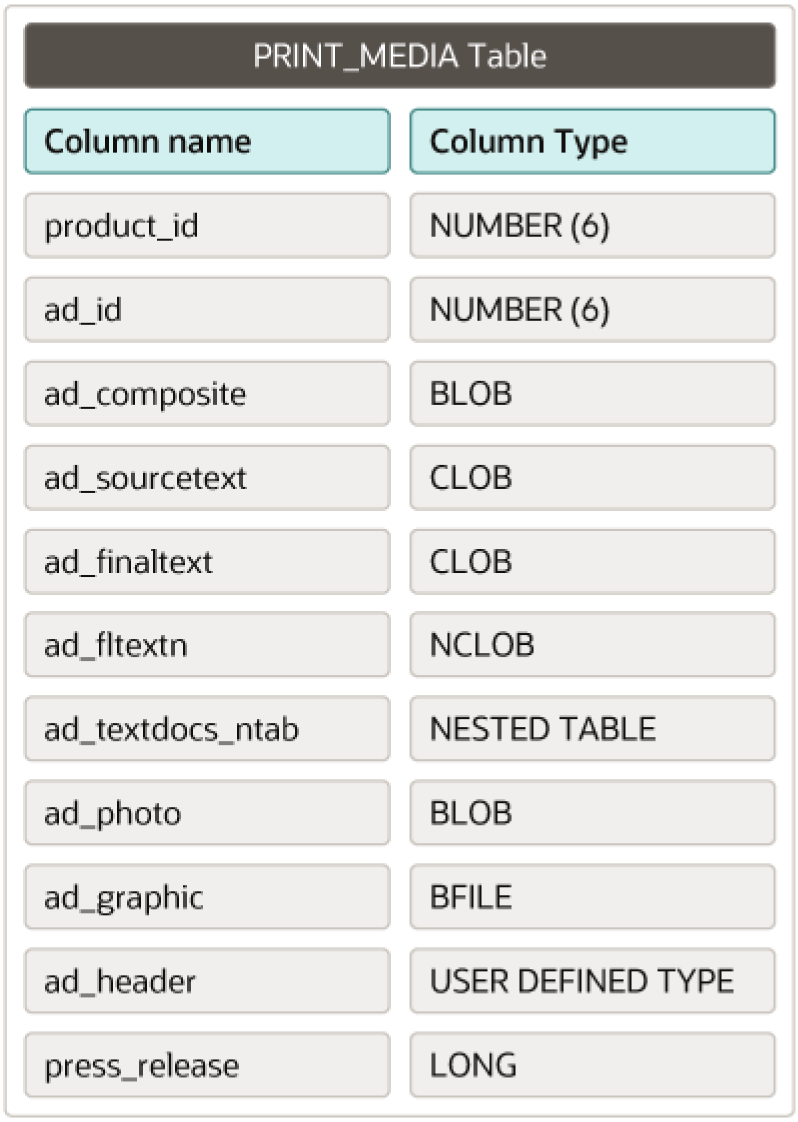
2.1 Creating a Table with LOB Columns

You can use the `CREATE TABLE` statement or an `ALTER TABLE ADD` column statement to create a new LOB column. This section introduces basic DDL operations on LOBs to get you started quickly.

Following is an example of creating a table with columns of various LOB types, including LOBs in Object Types and nested tables:

```
CREATE USER pm identified by password;
GRANT CONNECT, RESOURCE to pm IDENTIFIED BY pm;
CONNECT pm/pm

-- Create an object type with a LOB
CREATE TYPE adheader_typ AS OBJECT (
   header_name    VARCHAR2(256),
   creation_date  DATE,
   header_text    VARCHAR(1024),
   logo           BLOB );

CREATE TYPE textdoc_typ AS OBJECT (
   document_typ   VARCHAR2(32),
   formatted_doc  BLOB);

-- Create a nested table type of Object type containing a LOB
CREATE TYPE Textdoc_ntab AS TABLE of textdoc_typ;

-- Create a table of Object type, and specify a default value for LOB column
CREATE TABLE adheader_tab of adheader_typ (
    logo DEFAULT EMPTY_BLOB(),
    CONSTRAINT header_name CHECK (header_name IS NOT NULL),
     header_text DEFAULT NULL);
-- Create a table with columns of different LOB types,
-- and of object type with LOBs, and nested table containing LOB
CREATE TABLE print_media
(product_id NUMBER(6),
ad_id NUMBER(6),
ad_composite BLOB,
ad_sourcetext CLOB,
ad_finaltext CLOB,
ad_fltextn NCLOB,
ad_testdocs_ntab textdoc_tab,
ad_photo BLOB,
ad_graphic BFILE,
ad_header adheader_typ,
press_release LONG) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;

CREATE UNIQUE INDEX printmedia_pk
  ON print_media (product_id, ad_id);
```

Figure 2-1 print_media 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

You can also perform advanced DDL operations, like the following, on LOBs:

- Specify LOB storage parameters: You can override the default LOB storage settings by specifying parameters like `SECUREFILE/BASICFILE`, `TABLESPACE` where the LOB data will be stored, `ENABLE/DISABLE STORAGE IN ROW`, `RETENTION`, caching, logging, etc. You can also specify SecureFile specific parameters like `COMPRESSION`, `DEDUPLICATION` and `ENCRYPTION`.
- Alter an existing LOB column: You can use the `ALTER TABLE MODIFY LOB` syntax to change any LOB storage parameters that don't require LOB data movement and the `ALTER TABLE MOVE LOB` syntax to change any LOB storage parameters that require LOB data movement.
- Create indexes on LOB columns: You can build a functional or a domain index on a LOB column. You cannot build a B-tree or bitmap index on a LOB column.
- Partition a table containing LOB columns: All partitioning schemes supported by Oracle are fully supported on LOBs.
- Use LOBs in Index-Organized tables.

See Also:

Persistent LOBs: Advanced DDL

2.2 Inserting and Updating LOB Values in Tables

Oracle Database provides various methods to insert and update the data available in LOB columns of database tables.

- Inserting and Updating with a Buffer
  You can insert a character string directly into a `CLOB` or `NCLOB` column. Similarly, you can insert a raw buffer into a `BLOB` column. This is the most efficient way to insert data into a LOB.
- Inserting and Updating by Selecting a LOB From Another Table
  You can insert into a LOB column of a table by selecting data from a LOB column of the same table or a different table. You can also insert data into a LOB column of a table by selecting a LOB returned by a SQL operator or a PL/SQL function.
- Inserting and Updating with a NULL or Empty LOB
  You can set a persistent LOB, that is, a LOB column in a table or a LOB attribute in an object type that you defined, to be NULL or empty.
- Inserting and Updating with a LOB Locator
  If you are using a Programmatic Interface, which has a LOB variable that was previously populated by a persistent or temporary LOB locator, then you can insert a row by initializing the LOB bind variable.

2.2.1 Inserting and Updating with a Buffer

You can insert a character string directly into a `CLOB` or `NCLOB` column. Similarly, you can insert a raw buffer into a `BLOB` column. This is the most efficient way to insert data into a LOB.

The following code snippet inserts a character string into a `CLOB` column:

```
/* Store records in the archive table Online_media: */
INSERT INTO Online_media (product_id, product_text) VALUES (3060, 'some text
about this CRT Monitor');
```

The following code snippet updates the value in a `CLOB` column with character buffer:

```
UPDATE Online_media set product_text = 'some other text' where product_id =
3060;
```

See Also:

Data Interface for LOBs for more information about `INSERT` and `UPDATE` operations

2.2.2 Inserting and Updating by Selecting a LOB From Another Table

You can insert into a LOB column of a table by selecting data from a LOB column of the same table or a different table. You can also insert data into a LOB column of a table by selecting a LOB returned by a SQL operator or a PL/SQL function.

Ensure that you meet the following conditions while selecting data from columns that are part of more than one table:

- The LOB data type is the same for both the columns in the tables
- Implicit conversion is allowed between the two LOB data types used in both the columns

When a `BLOB`, `CLOB`, or `NCLOB` is copied from one row to another in the same table or a different table, the actual LOB value is copied, not just the LOB locator.

The following code snippet demonstrates inserting a LOB column from by selecting a LOB from another table. The columns `online_media.product_text` and `print_media.ad_sourcetext` are both `CLOB` types.

```
/* Insert values into Print_media by selecting from Online_media: */
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
(SELECT product_id, 11001, product_text FROM Online_media WHERE product_id =
3060);

/* Insert values into Print_media by selecting a SQL function returning a
CLOB */
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
(SELECT product_id, 11001, substr(product_text, 5) FROM Online_media WHERE
product_id = 3060);

/* Updating a row by selecting a LOB from another table (persistent LOBs) */

UPDATE Print_media SET ad_sourcetext = (SELECT product_text FROM online_media
WHERE product_id = 3060);
 WHERE product_id = 3060 AND ad_id = 11001;

/* Updating a row by selecting a SQL function returning a CLOB */

UPDATE Print_media SET ad_sourcetext = (SELECT substr(product_text, 5) FROM
online_media WHERE product_id = 3060);
WHERE product_id = 3060 AND ad_id = 11001;
```

The following code snippet demonstrates updating a LOB column from by selecting a LOB from another table.

```
/* Updating a row by selecting a LOB from another table (persistent LOBs) */
UPDATE Print_media SET ad_sourcetext = (SELECT product_text FROM online_media
WHERE product_id = 3060);
WHERE product_id = 3060 AND ad_id = 11001;

/* Updating a row by selecting a SQL function returning a CLOB */
UPDATE Print_media SET ad_sourcetext = (SELECT substr(product_text, 5) FROM
online_media WHERE product_id = 3060)
WHERE product_id = 3060 AND ad_id = 11001;
```

See Also:

- *Oracle Database SQL Language Reference* for more information on `INSERT`
- Performing Parallel DDL, Parallel DML (PDML), and Parallel Query (PQ) Operations on LOBs for information about how to make the `INSERT AS SELECT` operation run in parallel

2.2.3 Inserting and Updating with a NULL or Empty LOB

You can set a persistent LOB, that is, a LOB column in a table or a LOB attribute in an object type that you defined, to be NULL or empty.

Inserting a NULL LOB value

A persistent LOB set to NULL has no locator. A NULL value is stored in the row in the table, not a locator. This is the same process as for scalar data types. To INSERT a NULL value into a LOB column, simply use a statement like:

```
INSERT INTO print_media(product_id, ad_id, ad_sourcetext) VALUES (1, 1, NULL);
```

This is useful in situations where you want to use a `SELECT` statement, such as the following, to determine whether or not the LOB holds a NULL value:

```
SELECT COUNT (*) FROM print_media WHERE ad_graphic IS NULL;
```

Caution:

You cannot call `DBMS_LOB` functions or LOB APIs in other Programmatic Interfaces on a NULL LOB, so you must then use a SQL `UPDATE` statement to reset the LOB column to a non-NULL (or empty) value.

Inserting an EMPTY LOB value

Before you can write data to a persistent LOB using an API like `DBMS_LOB.WRITE` or `OCILobWrite2`, the LOB column must be non-`NULL`, that is, it must contain a locator that points to an empty or a populated LOB value.

You can initialize a `BLOB` column value by using the `EMPTY_BLOB()` function as a default predicate. Similarly, a `CLOB` or `NCLOB` column value can be initialized by using the `EMPTY_CLOB()` function. Use the `RETURNING` clause in the `INSERT` and `UPDATE` statement, to minimize the number of round trips while writing the LOB using APIs.

Following PL/SQL block initializes a `CLOB` column with an empty LOB using the `EMPTY_CLOB()` function and also updates the LOB value in a column with an empty `CLOB` using the `EMPTY_CLOB()` function.

```
DECLARE
    c CLOB;
    amt INTEGER := 11;
    buf VARCHAR(11) := 'Hello there';
BEGIN
  /* Insert empty_clob() */
  INSERT INTO Print_media(product_id, ad_id, ad_sourcetext) VALUES (1, 1,
EMPTY_CLOB()) RETURNING ad_source INTO c;
  /* The following statement updates the persistent LOB directly */
  DBMS_LOB.WRITE(c, amt, 1, buf);

  /* Update column to an empty_clob() */
  UPDATE Print_media SET ad_sourcetext = EMPTY_CLOB() WHERE product_id = 2
AND ad_id = 2 RETURNING ad_source INTO c;
  /* The following statement updates the persistent LOB directly */
  DBMS_LOB.WRITE(c, amt, 1, buf);
END;
/
```

2.2.4 Inserting and Updating with a LOB Locator

If you are using a Programmatic Interface, which has a LOB variable that was previously populated by a persistent or temporary LOB locator, then you can insert a row by initializing the LOB bind variable.

You can populate a LOB variable with a persistent LOB or a temporary LOB by either selecting one out from the database using SQL or by creating a temporary LOB. This section provides information about how to achieve this in various programmatic environments.

- PL/SQL: Inserting a Row by Initializing a LOB Locator Bind Variable
  The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using PL/SQL APIs.
- JDBC (Java): Inserting a Row by Initializing a LOB Locator Bind Variable
  The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using JDBC APIs:
- OCI (C): Inserting a Row by Initializing a LOB Locator Bind Variable
  The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using OCI APIs:
- Pro*C/C++ (C/C++): Inserting a Row by Initializing a LOB Locator Bind Variable
  The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using Pro*C/C++ APIs:
- Pro*COBOL (COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable
  The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using Pro*COBOL APIs:

2.2.4.1 PL/SQL: Inserting a Row by Initializing a LOB Locator Bind Variable

The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using PL/SQL APIs.

```
/* inserting a row through an insert statement */

CREATE OR REPLACE PROCEDURE insertLOB_proc (Lob_loc IN BLOB) IS
BEGIN
  /* Insert the BLOB into the row */
  DBMS_OUTPUT.PUT_LINE('------------ LOB INSERT EXAMPLE ------------');
  INSERT INTO print_media (product_id, ad_id, ad_photo)
```

```
        VALUES (3106, 60315, Lob_loc);
END;
/
```

2.2.4.2 JDBC (Java): Inserting a Row by Initializing a LOB Locator Bind Variable

The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using JDBC APIs:

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class linsert
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "password");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       ResultSet rset = stmt.executeQuery (
  "SELECT ad_photo FROM Print_media WHERE product_id = 3106 AND ad_id = 13001");
       if (rset.next())
       {
          // retrieve the LOB locator from the ResultSet
          BLOB adphoto_blob = ((OracleResultSet)rset).getBLOB (1);
          OraclePreparedStatement ops =
          (OraclePreparedStatement) conn.prepareStatement(
"INSERT INTO Print_media (product_id, ad_id, ad_photo) VALUES (2268, "
+ "21001, ?)");
          ops.setBlob(1, adphoto_blob);
          ops.execute();
          conn.commit();
          conn.close();
       }
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

2.2.4.3 OCI (C): Inserting a Row by Initializing a LOB Locator Bind Variable

The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using OCI APIs:

```
/* Insert the Locator into table using Bind Variables. */
#include <oratypes.h>
#include <lobdemo.h>
void insertLOB_proc(OCILobLocator *Lob_loc, OCIEnv *envhp,
                    OCIError *errhp, OCISvcCtx *svchp, OCIStmt *stmthp)
{
  int            product_id;
  OCIBind       *bndhp3;
  OCIBind       *bndhp2;
  OCIBind       *bndhp1;
  text          *insstmt =
   (text *) "INSERT INTO Print_media (product_id, ad_id, ad_sourcetext) \
             VALUES (:1, :2, :3)";

  printf ("----------- OCI Lob Insert Demo --------------\n");
  /* Insert the locator into the Print_media table with product_id=3060 */
  product_id = (int)3060;

  /* Prepare the SQL statement */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Binds the bind positions */
  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                                (void *) &product_id, (sb4) sizeof(product_id),
                                SQLT_INT, (void *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 2,
                                (void *) &product_id, (sb4) sizeof(product_id),
                                SQLT_INT, (void *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 3,
                                (void *) &Lob_loc, (sb4) 0,  SQLT_CLOB,
                                (void *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the SQL statement */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
}
```

2.2.4.4 Pro*C/C++ (C/C++): Inserting a Row by Initializing a LOB Locator Bind Variable

The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using Pro*C/C++ APIs:

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
```

```
void Sample_Error()
{
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
   EXEC SQL ROLLBACK WORK RELEASE;
   exit(1);
}

void insertUseBindVariable_proc(Rownum, Lob_loc)
   int Rownum, Rownum2;
   OCIBlobLocator *Lob_loc;
{
   EXEC SQL WHENEVER SQLERROR DO Sample_Error();
   EXEC SQL INSERT INTO Print_media (product_id, ad_id, ad_photo)
      VALUES (:Rownum, :Rownum2, :Lob_loc);
}
void insertBLOB_proc()
{
   OCIBlobLocator *Lob_loc;

   /* Initialize the BLOB Locator: */
   EXEC SQL ALLOCATE :Lob_loc;

   /* Select the LOB from the row where product_id = 2268 and ad_id=21001: */
   EXEC SQL SELECT ad_photo INTO :Lob_loc
      FROM Print_media WHERE product_id = 2268 AND ad_id = 21001;

   /* Insert into the row where product_id = 3106 and ad_id = 13001: */
   insertUseBindVariable_proc(3106, 13001, Lob_loc);

   /* Release resources held by the locator: */
   EXEC SQL FREE :Lob_loc;
}

void main()
{
   char *samp = "pm/password";
   EXEC SQL CONNECT :pm;
   insertBLOB_proc();
   EXEC SQL ROLLBACK WORK RELEASE;
}
```

2.2.4.5 Pro*COBOL (COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable

The following code snippet demonstrates how to insert a row by initializing a LOB locator bind variable using Pro*COBOL APIs:

You can insert a row by initializing a LOB locator bind variable in COBOL (Pro*COBOL).

```
IDENTIFICATION DIVISION.
    PROGRAM-ID. INSERT-LOB.
    ENVIRONMENT DIVISION.
    DATA DIVISION.
    WORKING-STORAGE SECTION.

    01 BLOB1 SQL-BLOB.
    01  USERID   PIC X(11) VALUES "PM/password".
        EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
 PROCEDURE DIVISION.
 INSERT-LOB.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL CONNECT :USERID END-EXEC.
* Initialize the BLOB locator
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.
* Populate the LOB
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
        SELECT AD_PHOTO INTO :BLOB1 FROM PRINT_MEDIA
         WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 END-EXEC.

* Insert the value with PRODUCT_ID of 3060
     EXEC SQL
        INSERT INTO PRINT_MEDIA (PRODUCT_ID, AD_PHOTO)
           VALUES (3060, 11001, :BLOB1)END-EXEC.

* Free resources held by locator
 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED:".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
     STOP RUN.
```

2.3 Selecting LOB Values from Tables

You can select a LOB into a Character Buffer, a RAW Buffer, or a LOB variable for performing read and write operations.

- Selecting a LOB into a Character Buffer or a Raw Buffer
  You can directly select a CLOB or NCLOB value into a character buffer or a BLOB value. This is called the Data Interface, and is the most efficient way for selecting from a LOB column.
- Selecting a LOB into a LOB Variable for Read Operations
  You can select a persistent or temporary LOB into a LOB variable, and then use APIs to perform various read operations on it.
- Selecting a LOB into a LOB Variable for Write Operations
  To perform a write operation using a LOB locator, you must lock the row in the table in order to prevent other database users from writing to the LOB during a transaction.

2.3.1 Selecting a LOB into a Character Buffer or a Raw Buffer

You can directly select a CLOB or NCLOB value into a character buffer or a BLOB value. This is called the Data Interface, and is the most efficient way for selecting from a LOB column.

See Also:

- Data Interface for LOBs
- PL/SQL Semantics for LOBs

2.3.2 Selecting a LOB into a LOB Variable for Read Operations

You can select a persistent or temporary LOB into a LOB variable, and then use APIs to perform various read operations on it.

Following code selects a LOB Locator into a variable:

```
DECLARE
    perslob CLOB;
    templob CLOB;
    amt INTEGER := 11;
    buf VARCHAR(100);
BEGIN
    SELECT ad_source, substr(ad_source, 3) INTO perslob, templob FROM
Print_media WHERE product_id = 1 AND ad_id = 1;
    DBMS_LOB.READ(perslob, amt, buf);
    DBMS_LOB.READ(templob, amt, buf);
END;
/
```

See Also:

- A Selected Locator Becomes a Read-Consistent Locator
- LOB Locators and Transaction Boundaries

2.3.3 Selecting a LOB into a LOB Variable for Write Operations

To perform a write operation using a LOB locator, you must lock the row in the table in order to prevent other database users from writing to the LOB during a transaction.

You can use one of the following mechanisms for this operation:

- Performing an `INSERT` or an `UPDATE` operation with a `RETURNING` clause.

  See Also:

  Inserting and Updating with a NULL or Empty LOB

- Performing a `SELECT` for an `UPDATE` operation. The following code snippet shows how to select a LOB value to perform a write operation using `UPDATE`.

  ```
  DECLARE
      c CLOB;
      amt INTEGER := 9;
      buf VARCHAR(100) := 'New Value';
  BEGIN
      SELECT ad_sourcetext INTO c FROM Print_media WHERE product_id = 1 AND
  ad_id = 1 FOR UPDATE;
      DBMS_LOB.WRITE(c, amt, 1, buf);
  END;
  /
  ```

- Using an OCI `pin` or `lock` function in OCI programs.

2.4 Performing DML and Query Operations on LOBs in Nested Tables

This section describes the `INSERT`, `UPDATE`, and `SELECT` operations on LOBs in Nested Tables. To update LOBs in a nested table, you must lock the row containing the LOB explicitly.

To lock the row containing the LOB, you must specify the `FOR UPDATE` clause in the subquery prior to updating the LOB value. The following example shows how to perform DML and query operations on LOBs in nested tables.

Note:

Locking the row of a parent table does not lock the row of a nested table containing LOB columns.

Example 2-1 Performing DML and Query Operations on LOBs in Nested Tables

```
CONNECT pm/pm;


------------------------------------------------------------------------
---------- Inserting LOBs in Nested Tables -----------------------------
------------------------------------------------------------------------

-- INSERT a row into the NT column of print_media with actual data for lob
INSERT INTO print_media (product_id, ad_id, ad_textdocs_ntab)
VALUES
(1, 1, textdoc_tab(textdoc_typ('txt', to_blob('BABABABABABA')),
                   textdoc_typ('pdf', to_blob('AAAAAAAAAAAA'))));

-- INSERT a row into the NT column of print_media with empty_lob for the lob
INSERT INTO print_media (product_id, ad_id, ad_textdocs_ntab)
VALUES
(2, 2, textdoc_tab(textdoc_typ('txt', empty_blob()),
                   textdoc_typ('pdf', empty_blob())));
```

```
SET SERVEROUTPUT ON

------------------------------------------------------------------------
---------- Read/Write LOBs in Nested Tables using locators --------------
------------------------------------------------------------------------

-- INSERT-RETURNING, then write to the LOBs
DECLARE
  txt textdoc_tab;
BEGIN
  INSERT INTO print_media p(product_id, ad_id, ad_textdocs_ntab) VALUES
    (3, 3, textdoc_tab(textdoc_typ('txt', empty_blob()),
                       textdoc_typ('pdf', empty_blob())))
  RETURNING p.ad_textdocs_ntab into txt;

  for elem in 1 .. txt.count loop
    DBMS_LOB.WRITEAPPEND(txt(elem).formatted_doc, 2, hextoraw(elem||'FF'));
  end loop;
END;
/

SELECT ad_textdocs_ntab FROM print_media WHERE product_id = 3;

-- SELECT on NT lob, then read
DECLARE
  txt textdoc_tab;
  pos INTEGER;
  amt INTEGER;
  buf RAW(40);
BEGIN
  SELECT ad_textdocs_ntab INTO txt FROM print_media WHERE product_id = 1;

  for elem in 1 .. txt.count loop
    amt := 40;
    pos := 1;
    DBMS_LOB.READ(txt(elem).formatted_doc, amt, pos, buf);
    DBMS_OUTPUT.PUT_LINE(buf);
  end loop;
END;
/

-- SELECT for update on the NT lob, then write
DECLARE
  txt textdoc_tab;
  pos INTEGER;
  amt INTEGER;
  buf RAW(40);
BEGIN
  SELECT ad_textdocs_ntab INTO txt FROM print_media
  WHERE product_id = 1 FOR UPDATE;

  for elem in 1 .. txt.count loop
    DBMS_LOB.WRITEAPPEND(txt(elem).formatted_doc, 2, hextoraw(elem||'FF'));
  end loop;
END;
/
```

```
SELECT ad_textdocs_ntab FROM print_media WHERE product_id = 1;
```

2.5 Performing Parallel DDL, Parallel DML (PDML), and Parallel Query (PQ) Operations on LOBs

Oracle supports parallel execution of the following operations when performed on partitioned tables with SecureFiles LOBs or BasicFiles LOBs.

- `CREATE TABLE AS SELECT`
- `INSERT AS SELECT`
- Multitable `INSERT`
- `SELECT`
- `DELETE`
- `UPDATE`
- `MERGE` (conditional `UPDATE` and `INSERT`)
- `ALTER TABLE MOVE`
- SQL Loader
- Import/Export

Additionally, Oracle supports parallel execution of the following operations when performed on non-partitioned tables with only SecureFile LOBs:

- `CREATE TABLE AS SELECT`
- `INSERT AS SELECT`
- Multitable `INSERT`
- `SELECT`
- `DELETE`
- `UPDATE`
- `MERGE` (conditional `UPDATE` and `INSERT`)
- `ALTER TABLE MOVE`
- SQL Loader

Restrictions on parallel operations with LOBs

- Parallel insert direct load (PIDL) is disabled if a table also has a BasicFiles LOB column, in addition to a SecureFiles LOB column.
- PDML is disabled if LOB column is part of a constraint.
- PDML does not work when there are any domain indexes defined on the LOB column.
- Parallelism must be specified only for top-level non-partitioned tables.
- Use the `ALTER TABLE MOVE` statement with LOB storage clause, to change the storage properties of LOB columns instead of the `ALTER TABLE MODIFY` statement. The `ALTER TABLE MOVE` statement is more efficient because it executes in parallel and does not generate undo logs.

See Also:

Oracle Database Administrator's Guide section "Managing Processes for Parallel SQL Execution"

Oracle Database SQL Language Reference section "ALTER TABLE"

2.6 Sharding with LOBs

LOBs can be used in a sharded environment. This section discusses the interfaces to support LOBs in sharded tables.

The following interfaces are supported:

- Query and DML statements
  - Cross shard queries involving LOBs are supported.
  - DML statements involving more than one shard are not supported. This behavior is similar to scalar columns.
  - DML statements involving a single shard are supported from coordinator.
  - Locator selected from a shard can be passed as bind value to the same shard.
- `OCILob`
  All non-BFILE related OCILob APIs in a sharding environment are supported, with some restrictions.

  On the coordinator, the `OCI_ATTR_LOB_REMOTE` attribute of a LOB descriptor returns `TRUE` if the LOB was obtained from a sharded table.

  Restrictions: For APIs that take two locators as input, `OCILobAppend`, `OCILobCompare` for example, both of the locators should be obtained from the same shard. If locators are from different shards an error is given.
- `DBMS_LOB`

  All non-BFILE related DBMS_LOB APIs in a sharding environment are supported, with some restrictions. On the coordinator, `DBMS_LOB.isremote` returns `TRUE` if the LOB was obtained from a sharded table.

  Restrictions: For APIs that take two locators as input, `DBMS_LOB.append` and `DBMS_LOB.compare` for example, both of the locators should be obtained from the same shard. If the locators are from different shards an error given.

See Also:

Sharded Tables