7

PL/SQL Semantics for LOBs

This chapter covers topics related to PL/SQL semantics for LOBs.

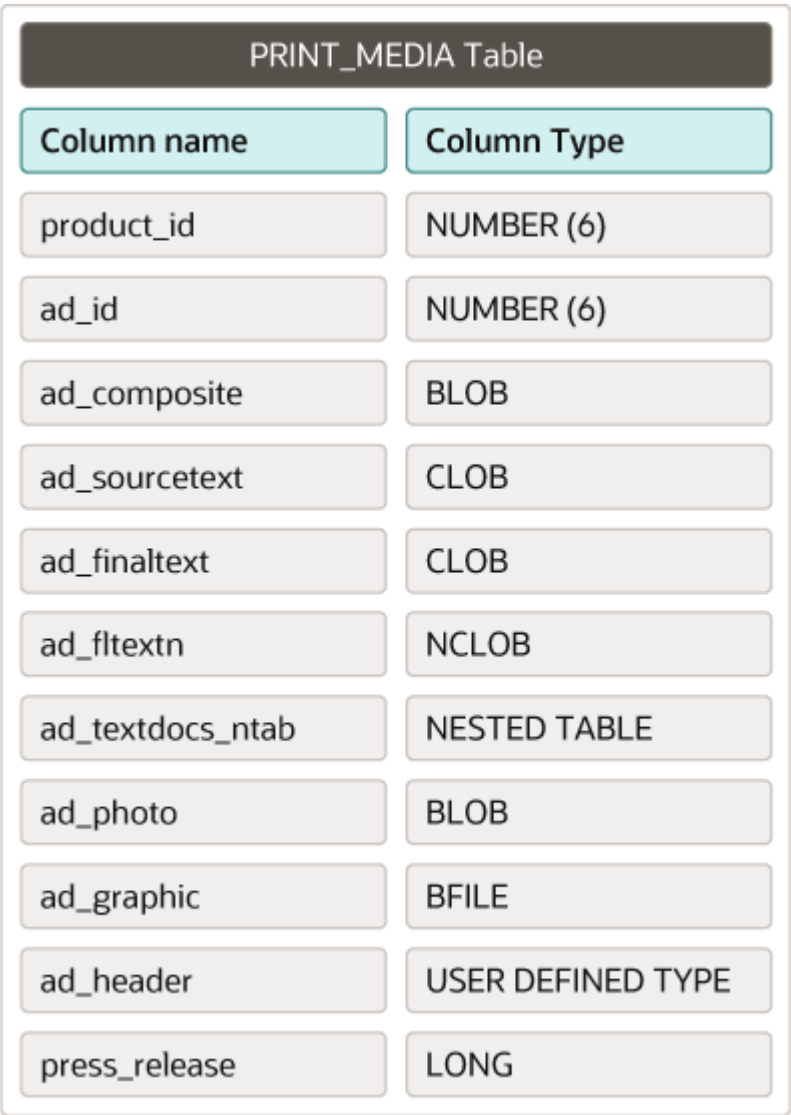
- Implicit Conversion with LOBs
  This section describes the implicit conversion process in PL/SQL from one LOB type to another LOB type or from a LOB type to a non-LOB type.
- Explicit Data Type Conversion Functions
  This section describes the explicit conversion functions in SQL and PL/SQL to convert other data types to and from `CLOB`, `NCLOB`, and `BLOB` data types.
- Temporary LOBs Created by SQL and PL/SQL Built-in Functions
  When a LOB is returned from a SQL or PL/SQL built-in function, then the result returned is a temporary LOB. Similarly, a LOB returned from a user-defined PL/SQL function or procedure, as a value or an OUT parameter, may be a temporary LOB.

7.1 Implicit Conversion with LOBs

This section describes the implicit conversion process in PL/SQL from one LOB type to another LOB type or from a LOB type to a non-LOB type.

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

Figure 7-1 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

- Implicit Conversion Between CLOB and NCLOB Data Types in SQL
  This section describes support for implicit conversions between CLOB and NCLOB data types.
- Implicit Conversions Between CLOB and VARCHAR2
  This section describes support for implicit conversions between `CLOB` and `VARCHAR2` data types.
- Implicit Conversions Between BLOB and RAW
  This section describes support for implicit conversions between BLOB and RAW data types.
- Guidelines and Restrictions for Implicit Conversions with LOBs
  This section describes the techniques that you use to access LOB columns or attributes using the Data Interface for LOBs.
- Detailed Examples for Implicit Conversions with LOBs
  The example in this section demonstrates using multiple `VARCHAR` and `RAW` binds in `INSERT` and `UPDATE` operations.

7.1.1 Implicit Conversion Between CLOB and NCLOB Data Types in SQL

This section describes support for implicit conversions between CLOB and NCLOB data types.

The database enables you to perform operations such as cross-type assignment and cross-type parameter passing between `CLOB` and `NCLOB` data types. The database performs implicit conversions between these types when necessary to preserve properties such as character set formatting.

Note that, when implicit conversions occur, each character in the source LOB is changed to the character set of the destination LOB, if needed. In this situation, some degradation of performance may occur if the data size is large. When the character set of the destination and the source are the same, there is no degradation of performance.

After an implicit conversion between `CLOB` and `NCLOB` types, the destination LOB is implicitly created as a temporary LOB. This new temporary LOB is independent from the source LOB. If the implicit conversion occurs as part of a define operation in a `SELECT` statement, then any modifications to the destination LOB do not affect the persistent LOB in the table that the LOB was selected from as shown in the following example:

```
SQL> -- check lob length before update
SQL> SELECT DBMS_LOB.GETLENGTH(ad_sourcetext) FROM Print_media
  2       WHERE product_id=3106 AND ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
         205

SQL>
SQL> DECLARE
  2   clob1 CLOB;
  3   amt NUMBER:=10;
  4  BEGIN
  5    -- select a clob column into a clob, no implicit convesion
  6    SELECT ad_sourcetext INTO clob1 FROM Print_media
  7      WHERE product_id=3106 and ad_id=13001 FOR UPDATE;
  8    -- Trim the selected lob to 10 bytes
  9    DBMS_LOB.TRIM(clob1, amt);
 10  END;
 11  /

PL/SQL procedure successfully completed.

SQL> -- Modification is performed on clob1 which points to the
SQL> -- clob column in the table
SQL> SELECT DBMS_LOB.GETLENGTH(ad_sourcetext) FROM Print_media
  2       WHERE product_id=3106 AND ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
          10

SQL>
SQL> ROLLBACK;

Rollback complete.

SQL> -- check lob length before update
SQL> SELECT DBMS_LOB.GETLENGTH(ad_sourcetext) FROM Print_media
  2       WHERE product_id=3106 AND ad_id = 13001;
```

```
DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
         205

SQL>
SQL> DECLARE
  2   nclob1 NCLOB;
  3   amt NUMBER:=10;
  4  BEGIN
  5
  6    -- select a clob column into a nclob, implicit conversion occurs
  7    SELECT ad_sourcetext INTO nclob1 FROM Print_media
  8      WHERE product_id=3106 AND ad_id=13001 FOR UPDATE;
  9
 10    DBMS_LOB.TRIM(nclob1, amt); -- Trim the selected lob to 10 bytes
 11  END;
 12  /

PL/SQL procedure successfully completed.

SQL> -- Modification to nclob1 does not affect the clob in the table,
SQL> -- because nclob1 is a independent temporary LOB

SQL> SELECT DBMS_LOB.GETLENGTH(ad_sourcetext) FROM Print_media
  2       WHERE product_id=3106 AND ad_id = 13001;

DBMS_LOB.GETLENGTH(AD_SOURCETEXT)
---------------------------------
         205
```

See Also:

Oracle Database SQL Language Reference for details on implicit conversions supported for all data types.

7.1.2 Implicit Conversions Between CLOB and VARCHAR2

This section describes support for implicit conversions between `CLOB` and `VARCHAR2` data types.

Implicit conversions from `CLOB` to `VARCHAR2` and from `VARCHAR2` to `CLOB` data types are supported in PL/SQL.

See Also:

SQL Semantics for LOBs for details on LOB support in SQL statements.

Note:

While this section uses `VARCHAR2` data type as an example for simplicity, other character types like `CHAR` can also participate in implicit conversions with CLOBs.

Assigning a CLOB to a VARCHAR2 in PL/SQL

When assigning a CLOB to a VARCHAR2, the data stored in the CLOB column is retrieved and stored into the VARCHAR2 buffer. If the buffer is not large enough to contain all the CLOB data, then a truncation error is thrown and no data is written to the buffer. This is consistent with VARCHAR2 semantics. After successful completion of this assignment operation, the VARCHAR2 variable holds the data as a regular character buffer. This operation can be performed in the following ways:

- SELECT persistent or temporary CLOB data into a character buffer variable such as CHAR or VARCHAR2. In a single SELECT statement, you can have more than one of such defines.
- Assign a CLOB to a VARCHAR2 or CHAR variable.
- Pass CLOB data types to built-in SQL and PL/SQL functions and operators that accept VARCHAR2 arguments, such as the INSTR function and the SUBSTR function.
- Pass CLOB data types to user-defined PL/SQL functions that accept VARCHAR2 data types.

The following example illustrates the way CLOB data is accessed when the CLOBs are treated as VARCHAR2s:

```
DECLARE
  myStoryBuf VARCHAR2(32000);
  myLob  CLOB;
BEGIN
  -- Select a LOB into a VARCHAR2 variable
  SELECT ad_sourcetext INTO myStoryBuf FROM print_media WHERE ad_id = 12001;
  DBMS_OUTPUT.PUT_LINE(myStoryBuf);
  -- Assign a LOB to a VARCHAR2 variable
  SELECT ad_sourcetext INTO myLob FROM print_media WHERE ad_id = 12001;
  myStoryBuf := myLob;
  DBMS_OUTPUT.PUT_LINE(myStoryBuf);
END;
/
```

Assigning a VARCHAR2 to a CLOB in PL/SQL

A VARCHAR2 can be assigned to a CLOB in the following scenarios:

- INSERT or UPDATE character data stored in VARCHAR2 or CHAR variables into a CLOB column. Multiple such binds are allowed in a single INSERT or UPDATE statement.
- Assign a VARCHAR2 or CHAR variable to a CLOB variable.
- Pass VARCHAR2 data types to user-defined PL/SQL functions that accept LOB data types.

```
DECLARE
  myLOB CLOB;
BEGIN
  -- Select a VARCHAR2 into a LOB variable
  SELECT 'ABCDE' INTO myLOB FROM print_media WHERE ad_id = 11001;
  -- myLOB is a temporary LOB.
  -- Use myLOB as a lob locator
  DBMS_OUTPUT.PUT_LINE('Is temp? '||DBMS_LOB.ISTEMPORARY(myLOB));

  -- Insert a VARCHAR2 into a lob column
  INSERT INTO print_media(product_id, ad_id, AD_SOURCETEXT) VALUES (1000, 1,
```

```
'ABCDE');

  -- Assign a VARCHAR2 to a LOB variable
  myLob := 'XYZ';
 END;
/
```

7.1.3 Implicit Conversions Between BLOB and RAW

This section describes support for implicit conversions between BLOB and RAW data types.

Most discussions related to PL/SQL semantics for implicit conversion between `CLOB` and `VARCHAR2` data types also apply to the implicit conversion process between `BLOB` and `RAW` data types, unless mentioned otherwise. However, to provide concise description, most examples in this chapter do not explicitly mention `BLOB` and `RAW` data types. The following operations involving BLOB data types support implicit conversions:

- `INSERT` or `UPDATE` binary data stored in `RAW` variables into a `BLOB` column. Multiple such binds are allowed in a single `INSERT` or `UPDATE` statement.
- `SELECT` persistent or temporary `BLOB` data into a binary buffer variable such as `RAW`. Multiple such defines are allowed in a single `SELECT` statement.
- Assign a `BLOB` to a `RAW` variable, or assign a `RAW` to a `BLOB` variable.
- Pass `BLOB` data types to built-in or user-defined PL/SQL functions defined to accept the `RAW` data type or pass the `RAW` data type to built-in or user-defined PL/SQL functions defined to accept the `BLOB` data types.

7.1.4 Guidelines and Restrictions for Implicit Conversions with LOBs

This section describes the techniques that you use to access LOB columns or attributes using the Data Interface for LOBs.

Data from `CLOB` and `BLOB` columns or attributes can be referenced by regular SQL statements, such as `INSERT`, `UPDATE`, and `SELECT`.

There is no piecewise `INSERT`, `UPDATE`, or fetch routine in PL/SQL. Therefore, the amount of data that can be accessed from a LOB column or attribute is limited by the maximum character buffer size in PL/SQL, which is 32767 bytes. For this reason, only LOBs less than 32 kilo bytes in size can be accessed by PL/SQL applications using the data interface for persistent LOBs.

If you must access a LOB with a size more than `32 kilobytes -1` bytes, using the data interface, then you must make JDBC or OCI calls from the PL/SQL code to use the APIs for piecewise insert and fetch.

Use the following guidelines for using the Data Interface to access LOB columns or attributes:

- `SELECT` operations

  LOB columns or attributes can be selected into character or binary buffers in PL/SQL. If the LOB column or attribute is longer than the buffer size, then an exception is raised without filling the buffer with any data. LOB columns or attributes can also be selected into LOB locators.
- INSERT operations

You can INSERT into tables containing LOB columns or attributes using regular INSERT statements in the VALUES clause. The field of the LOB column can be a literal, a character data type, a binary data type, or a LOB locator.

- UPDATE operations

  LOB columns or attributes can be updated as a whole by UPDATE... SET statements. In the SET clause, the new value can be a literal, a character data type, a binary data type, or a LOB locator.

- There are restrictions for binds of more than 4000 bytes:
  - If a table has both LONG and LOB columns, then you can bind more than 4000 bytes of data to either the LONG or LOB columns, but not both in the same statement.
  - In an INSERT AS SELECT operation, binding of any length data to LOB columns is not allowed.
  - If you bind more than 4000 bytes of data to a BLOB or a CLOB, and the data consists of a SQL operator, then Oracle Database limits the size of the result to at most 4000 bytes. For example, the following statement inserts only 4000 bytes because the result of LPAD is limited to 4000 bytes:

    ```
    INSERT INTO print_media (ad_sourcetext) VALUES (lpad('a', 5000, 'a'));
    ```

  - The database does not do implicit hexadecimal to RAW or RAW to hexadecimal conversions on data that is more than 4000 bytes in size. You cannot bind a buffer of character data to a binary data type column, and you cannot bind a buffer of binary data to a character data type column if the buffer is over 4000 bytes in size. Attempting to do so results in your column data being truncated at 4000 bytes.

    For example, you cannot bind a VARCHAR2 buffer to a BLOB column if the buffer is more than 4000 bytes in size. Similarly, you cannot bind a RAW buffer to a CLOB column if the buffer is more than 4000 bytes in size.

7.1.5 Detailed Examples for Implicit Conversions with LOBs

The example in this section demonstrates using multiple VARCHAR and RAW binds in INSERT and UPDATE operations.

Example 7-1 Using Character and RAW Binds in INSERT and UPDATE Operations

The following example demonstrates using Character and RAW binds for LOB columns in INSERT and UPDATE operations

```
DECLARE
  bigtext VARCHAR2(32767);
  smalltext VARCHAR2(2000);
  bigraw RAW (32767);
BEGIN
  bigtext := LPAD('a', 32767, 'a');
  smalltext := LPAD('a', 2000, 'a');
  bigraw := utl_raw.cast_to_raw (bigtext);

  /* Multiple long binds for LOB columns are allowed for INSERT: */
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext, ad_composite)
    VALUES (2004, 1, bigtext, bigraw);

  /* Single long bind for LOB columns is allowed for INSERT: */
  INSERT INTO print_media (product_id, ad_id, ad_sourcetext)
```

```
    VALUES (2005, 2, smalltext);

  bigtext := LPAD('b', 32767, 'b');
  smalltext := LPAD('b', 20, 'a');
  bigraw := utl_raw.cast_to_raw (bigtext);

  /* Multiple long binds for LOB columns are allowed for UPDATE: */
  UPDATE print_media SET ad_sourcetext = bigtext, ad_composite = bigraw,
    ad_finaltext = smalltext;

  /* Single long bind for LOB columns is allowed for UPDATE: */
  UPDATE print_media SET ad_sourcetext = smalltext, ad_finaltext = bigtext;

  /* The following is NOT allowed because we are trying to insert more than
     4000 bytes of data in a LONG and a LOB column: */
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext, press_release)
    VALUES (2030, 3, bigtext, bigtext);

  /* Insert of data into LOB attribute is allowed */
  INSERT INTO print_media(product_id, ad_id, ad_header)
     VALUES (2049, 4, adheader_typ(null, null, null, bigraw));

  /* The following is not allowed because we try to perform INSERT AS
     SELECT data INTO LOB */
  INSERT INTO print_media(product_id, ad_id, ad_sourcetext)
    SELECT 2056, 5, bigtext FROM dual;

END;
/
```

Example 7-2 Multiple Defines for LOBs in SELECT

The following example demonstrates performing a `SELECT` operation to retrieve multiple persistent or temporary CLOBs from a SQL query into a `VARCHAR2` variable, or a `BLOB` to a `RAW` variable.

```
DECLARE
  ad_src_buffer    VARCHAR2(32000);
  ad_comp_buffer   RAW(32000);
BEGIN
  /* This retrieves the LOB columns if they are up to 32000 bytes,
   * otherwise it raises an exception */
  SELECT ad_sourcetext, ad_composite INTO ad_src_buffer, ad_comp_buffer FROM
print_media
    WHERE product_id=2004 AND ad_id=5;

  /* This retrieves the temporary LOB produced by SUBSTR if it is up to 32000
bytes,
   * otherwise it raises an exception */
  SELECT substr(ad_sourcetext, 2) INTO ad_src_buffer FROM print_media
    WHERE product_id=2004 AND ad_id=5;END;
/
```

Example 7-3 Implicit Conversions between `BLOB` and `RAW`

Implicit assignment works for variables declared explicitly and for variables declared by referencing an existing column type using the `%TYPE` attribute as show in the following example.

The example assumes that column `long_col` in table `t` has been migrated from a `LONG` to a `CLOB` column.

```
CREATE TABLE t (long_col LONG); -- Alter this table to change LONG column to
LOB
DECLARE
   a VARCHAR2(100);
   b t.long_col%type; -- This variable changes from LONG to CLOB
BEGIN
   SELECT * INTO b FROM t;
   a := b;  -- This changes from "VARCHAR2 := LONG to VARCHAR2 := CLOB
   b := a;  -- This changes from "LONG := VARCHAR2 to CLOB := VARCHAR2
END;
```

Example 7-4 Calling PL/SQL and C Procedures from PL/SQL

You can call a PL/SQL or C procedure from PL/SQL. You can pass a `CLOB` as an actual parameter, where a `VARCHAR2` is the formal parameter, or you can pass a `VARCHAR2` as an actual parameter, where a `CLOB` is the formal parameter. The same holds good for `BLOB`s and `RAW`s. One example of when these cases can arise is when either the formal or the actual parameter is an anchored type, that is, the variable is declared using the `table_name.column_name%type` syntax. PL/SQL procedures or functions can accept a `CLOB` or a `VARCHAR2` as a formal parameter. This holds for both built-in and user-defined procedures and functions.

The following example demonstrates implicit conversion during procedure calls:

```
CREATE OR REPLACE PROCEDURE foo(vvv IN VARCHAR2, ccc INOUT CLOB) AS
  ...
 BEGIN
  ...
  END;
  /
 DECLARE
  vvv VARCHAR2[32000] := rpad('varchar', 32000, 'varchar')
  ccc CLOB := rpad('clob', 32000, 'clob')
BEGIN
  foo(vvv, ccc); -- No implicit conversion needed here
  foo(ccc, vvv); -- Implicit conversion for both parameters done here
END;
/
```

Example 7-5 Implicit Conversion with PL/SQL built-in functions

The following example illustrates the use of `CLOB`s in PL/SQL built-in functions.

```
DECLARE
  my_ad CLOB;
  revised_ad CLOB;
  myGist VARCHAR2(100):= 'This is my gist.';
  revisedGist VARCHAR2(100);
BEGIN
  INSERT INTO print_media (product_id, ad_id, ad_sourcetext)
    VALUES (2004, 5, 'Source for advertisement 1');

  -- select a CLOB column into a CLOB variable
```

```
  SELECT ad_sourcetext INTO my_ad FROM print_media
    WHERE product_id=2004 AND ad_id=5;

  -- perform VARCHAR2 operations on a CLOB variable
  revised_ad := UPPER(SUBSTR(my_ad, 1, 20));

  -- revised_ad is a temporary LOB
  -- Concat a VARCHAR2 at the end of a CLOB
  revised_ad := revised_ad || myGist;

  -- The following statement raises an error if my_ad is
  -- longer than 100 bytes
  myGist := my_ad;
END;
/
```

7.2 Explicit Data Type Conversion Functions

This section describes the explicit conversion functions in SQL and PL/SQL to convert other data types to and from `CLOB`, `NCLOB`, and `BLOB` data types.

- `TO_CLOB()`: Converts from `VARCHAR2`, `NVARCHAR2`, or `NCLOB` to a `CLOB`
- `TO_NCLOB()`: Converts from `VARCHAR2`, `NVARCHAR2`, or `CLOB` to an `NCLOB`
- `TO_BLOB(varchar|clob, destcsid,[mime_type])`: Converts the object from its current character set to the given character set in `destcsid`. The resultant object is `BLOB`. Following are various ways in which you can use the conversion function:
  - `TO_BLOB(character, destcsid)`
  - `TO_BLOB(character, destcsid, mime_type)`
  - `TO_BLOB(clob, destcsid)`
  - `TO_BLOB(clob, destcsid, mime_type)`

  If the `destcsid` is 0, then it converts to the database character set ID. The parameter `mime_type` is applicable only to `INSERT` and `UPDATE` statements on Secure File `LOB` columns. If the `mime_type` parameter is used in `SELECT` statements or in temporary or BasicFile `LOB`s, then it is ignored.
- `TO_BLOB(varchar)`: Converts the input to `RAW` before converting to `BLOB`. In other words, `TO_BLOB(HEXTORAW(varchar))` and `TO_BLOB(varchar)` are equivalent.

  > **Note:**
  >
  > `TO_BLOB(CLOB)` is not supported.

- `TO_CHAR()`: Converts a `CLOB` to a `CHAR` type. When you use this function to convert a character LOB into the database character set, if the LOB value to be converted is larger than the target type, then the database returns an error. Implicit conversions also raise an error if the LOB data does not fit.
- `TO_NCHAR()`: Converts an `NCLOB` to an `NCHAR` type. When you use this function to convert a character LOB into the national character set, if the LOB value to be converted is larger

than the target type, then the database returns an error. Implicit conversions also raise an error if the LOB data does not fit.

- `CAST` does not directly support any of the LOB data types. When you use `CAST` to convert a `CLOB` value into a character data type, an `NCLOB` value into a national character data type, or a `BLOB` value into a `RAW` data type, the database implicitly converts the LOB value to character or raw data and then explicitly casts the resulting value into the target data type. If the resulting value is larger than the target type, then the database returns an error.

7.3 Temporary LOBs Created by SQL and PL/SQL Built-in Functions

When a LOB is returned from a SQL or PL/SQL built-in function, then the result returned is a temporary LOB. Similarly, a LOB returned from a user-defined PL/SQL function or procedure, as a value or an OUT parameter, may be a temporary LOB.

In PL/SQL, a temporary LOB has the same lifetime (duration) as the local PL/SQL program variable in which it is stored. It can be passed to subsequent SQL or PL/SQL `VARCHAR2` functions or queries as a PL/SQL local variable. The temporary LOB goes out of scope at the end of the program block at which time, the LOB is freed. These are the same semantics as those for PL/SQL `VARCHAR2` variables. At any time, nonetheless, you can use a `DBMS_LOB.FREETEMPORARY()` call to release the resources taken by the local temporary LOBs.

> **Note:**
>
> If a SQL or PL/SQL function returns a temporary LOB, or if a LOB is an OUT parameter for a PL/SQL function or procedure, then you must free it as soon as you are done with it. Failure to do so may cause temporary LOB accumulation and can considerably slow down your system.

The following example illustrates implicit creation of temporary LOBs using SQL built-in functions:

```
DECLARE
  vc1 VARCHAR2(32000);
  lb1 CLOB;
  lb2 CLOB;
BEGIN
  SELECT clobCol1 INTO vc1 FROM tab WHERE colID=1;
  -- lb1 is a temporary LOB
  SELECT clobCol2 || clobCol3 INTO lb1 FROM tab WHERE colID=2;

  lb2 := vc1|| lb1;
  -- lb2 is a still temporary LOB, so the persistent data in the database
  -- is not modified. An update is necessary to modify the table data.
  UPDATE tab SET clobCol1 = lb2 WHERE colID = 1;

DBMS_LOB.FREETEMPORARY(lb2); -- Free up the space taken by lb2


<... some more queries ...>

END; -- at the end of the block, lb1 is automatically freed
```

Here is another example of implicit creation of temporary LOBs using PL/SQL built-in functions.

```
1 DECLARE
2   myStory CLOB;
3   revisedStory CLOB;
4   myGist VARCHAR2(100);
5   revisedGist VARCHAR2(100);
6 BEGIN
7     -- select a CLOB column into a CLOB variable
8      SELECT Story INTO myStory FROM print_media WHERE product_id=10;
9      -- perform VARCHAR2 operations on a CLOB variable
10     revisedStory := UPPER(SUBSTR(myStory, 100, 1));
11     -- revisedStory is a temporary LOB
12     -- Concat a VARCHAR2 at the end of a CLOB
13     revisedStory := revisedStory || myGist;
14     -- The following statement raises an error because myStory is
15     -- longer than 100 bytes
16     myGist := myStory;
17 END;
/
```

Note that in the preceding example:

- In line number 7, a temporary `CLOB` is implicitly created and is pointed to by the `revisedStory CLOB` locator.
- In line number 13, `myGist` is appended to the end of the temporary LOB, which has the same effect as the following code snippet:

  ```
  DBMS_LOB.WRITEAPPEND(revisedStory, myGist, length(myGist));
  ```

In some scenarios, implicitly created temporary LOBs in PL/SQL statements can change the representation of previously defined LOB locators. The following code snippet explains this scenario:

Change in Locator-Data Linkage

```
1 DECLARE
2     myStory CLOB;
3     amt number:=100;
4     buffer VARCHAR2(100):='some data';
5 BEGIN
6     -- select a CLOB column into a CLOB variable
7     SELECT Story INTO myStory FROM print_media WHERE product_id=10;
8     DBMS_LOB.WRITE(myStory, amt, 1, buf);
9     -- write to the persistent LOB in the table
10
11     myStory:= UPPER(SUBSTR(myStory, 100, 1));
12     -- perform VARCHAR2 operations on a CLOB variable, temporary LOB created.
13     -- Changes are not reflected in the database table from this point on.
14
15     UPDATE print_media SET Story = myStory WHERE product_id = 10;
16     -- an update is necessary to synchronize the data in the table.
17 END;
```

In the preceding example, `myStory` represents a persistent LOB column in the `print_media` table. The `DBMS_LOB.WRITE` procedure writes the data directly to the table without an `UPDATE` statement in the code.

Subsequently in line number 11, a temporary LOB is created and assigned to `myStory` because `myStory` is now used like a local `VARCHAR2` variable. The LOB locator `myStory` now points to the newly-created temporary LOB.

Therefore, modifications to `myStory` are no longer reflected in the database. To propagate the changes to the database table now, you must use an `UPDATE` statement. Note that for the previous persistent LOB, the `UPDATE` statement is not required.

> **See Also:**
>
> Working with Remote LOBs in SQL and PL/SQL for PL/SQL functions that support remote `LOBs` and `BFILEs`