

## Chapter 21

# LOBS

- ❑ Introduction
- ❑ LOB Data types
- ❑ Defining and Manipulating LOBs
- ❑ DBMS\_LOB Package

### Introduction

Oracle supports LOBs(Large Objects) which can hold large amount of raw binary data, such as graphics images, as well as large amount of character data.

Oracle extended SQL DDL and DML to provide support for LOBs. You can also manipulate LOBs using DBMS\_LOB package and OCI (Oracle Call Interface).

Depending upon the way in which LOBs are stored they can be classified as follows.

### Internal LOBs

These are stored in the database tablespace. They support transaction processing like any other scalar data type. CLOB, BLOB and NCLOB belong to this category.

### External LOBs

These are not stored in the database. Instead they are stored in the Operating System files. Only a pointer pointing to the actual data is stored in the database. They do not support transaction processing and integrity checking. BFILE data type belongs to this category.

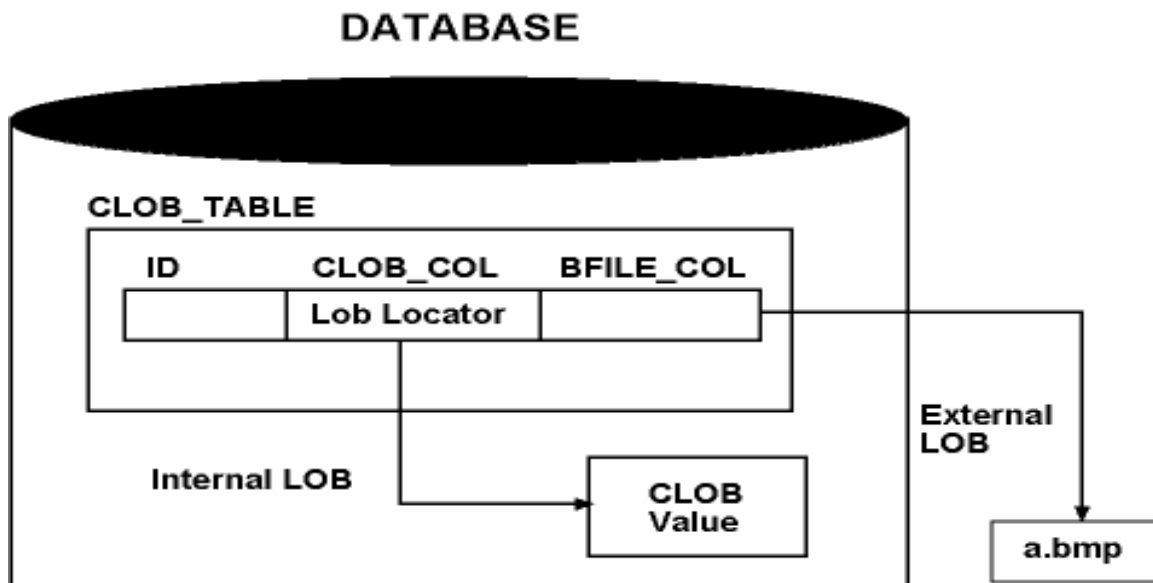
See figure 1, to understand how data is stored in internal lob and external lob.

## LOB Datatypes

The following are the different LOB datatypes that are supported by Oracle8.

Data Type	Description
CLOB	The data is consisting of single-byte character data.
NCLOB	The data is consisting of multi-byte or single-byte fixed length character data that corresponds to the national character set.
BLOB	The data is consisting of RAW binary data, such as bitmap images.
BFILE	The data is stored in an operating system file. Only a reference to the file is stored in the database. This is the example for External LOB.

**Table 1:** LOB Datatypes.



In the above example the table contains three columns – ID, CLOB\_COL and BFILE\_COL. CLOB\_COL is of CLOB type and BFILE\_COL is of BFILE type. CLOB column stores lob locator, which points to the location where the complete data is stored. BFILE column contains the name of the file (a.bmp) but the file is physically stored outside the database as a file in the operating systems file system.

### LOB Locator

The data of the LOB column is NOT stored in the row along with the other columns of the row, instead only a locator is stored in the database and the actual data is stored elsewhere. The locator is similar to a pointer and points to the location where the data is actually stored. In

---

case of Internal LOB the data is stored within the database, and in case of external LOB the data is stored outside the database as a file in the file system of the operating system.

The value that is stored in the row to point to the actual location of the Internal LOB is called as LOB Locator. It is used to locate the LOB data that is stored elsewhere in the database.

## LONG vs. LOBs

LONG datatype and LOBs are similar in some respects and differ in other. LOB data types can be taken as an extension to LONG RAW data type. LONG RAW is the only data type that supported large binary data in Oracle7. The following are the differences between LONG RAW data type and LOB data types.

LONG type	LOB type
Can contain up to 2 GB of data	Can contain up to 4 GB of data
A table can contain only one LONG column	A table can contain more than one LOB column
A sub-query cannot select a LONG column	A subquery can select LOB column

## Defining and Manipulating LOBs

A LOB column is defined just like any other column. Data of the LOB column can be accessed either directly or through DBMS\_LOB package. Let us first see how to create a table with LOB columns.

The following example creates a table with a CLOB type column and a BFILE type column.

```
create table lob_table
(
  id      number(5),
  clob_col clob,
  bfile_col bfile
);
```

The following functions are used to manipulate LOB columns.

### EMPTY\_BLOB and EMPTY\_CLOB Functions

These functions are part of the SQL DML. They are used to initialize LOB locator to empty locator in INSERT and UPDATE commands.

---

**Note:** Before you start writing data to a LOB using either OCI or DBMS\_LOB package, the LOB column must be initialized to empty locator using these functions.

---

The following example stores empty locator into CLOB\_COL.

```
insert into lob_table (id, clob_col)
  values (100, empty_clob());
```

### BFILENAME Function

This is used to initialize a BFILE column. This is used in SQL INSERT and UPDATE commands.

```
BFILENAME( Directory_alias, Filename) RETURN BFILE;
```

<b>Directory_alias</b>	Is the alias of the directory in which the file is stored. This directory alias is created using CREATE DIRECTORY command.
<b>Filename</b>	Is the name of the file, which contains the contents that are accessed using BFILE column.

```
insert into lob_table
  values( 101, EMPTY_CLOB(), BFILENAME('IMAGES','BMP1.BMP'));
```

The above INSERT command inserts a row into LOB\_TABLE in which BFILE\_COL refers to BMP1.BMP file in directory with alias IMAGES.

### Directory Alias

Directory alias is to be created using CREATE DIRECTORY command. Directory alias is used to refer to a physical directory (directory in the file system of the operating system) using an alias in Oracle.

The directory alias can be used to change the physical directory to which alias points, without having to change the alias itself. This makes the job of making BFILE column pointing to the right directory and right file easier, even though the files are moved from one physical directory to another.

```
create directory "IMAGES" as 'c:\bitmaps';
```

The above statement creates a directory alias called IMAGES, which is an alias to the physical directory C:\BITMAPS.

---

**Note:** The directory alias is case sensitive. When using directory alias in BFILENAME function, if proper case is not given then Oracle doesn't recognize the directory alias. But at the time of creation, Oracle converts the directory alias to uppercase if the name is not given in double quotes (" "). So if you want to preserve the case of the directory alias then enclose it in double quotes.

---

Now, the alias IMAGES can be used to refer to the physical directory in BFILENAME function as shown in the example below.

```
Insert into lob_table
values(101, 'Some data in CLOB cloumn',
       BFILENAME('IMAGES','BMP1.BMP'));
```

---

**Note:** it is not possible to display the value of BFILE column using SELECT command in SQL\*PLUS.

---

Users who have to access the directory, should be granted privilege to access the directory as follows:

```
grant read on directory LOB_DIR to Srikanth;
```

The above GRANT command grants READ permission on LOB\_DIR directory to user SRIKANTH.

## Deleting LOBs

When a LOB value is deleted, the locator to the LOB value as well as the LOB value are deleted in case of internal LOBs (CLOB, NCLOB, and BLOB). But in case of BFILE (external LOB) only the locator is deleted and the actual data is not deleted. This is obvious considering the fact that in case of external LOBs database stores only a pointer to the actual content but not the content itself. So for BFILES you need to delete data manually.

---

**Note:** The function discussed above deal with complete LOB column. They do not allow you to modify a part of the LOB column. For information regarding modifying a part of the LOB column, see DBMS\_LOB package, later in this chapter.

---

## DBMS\_LOB Package

Functions and procedures in DBMS\_LOB package can be used to perform various operations related to LOBs. These methods of DBMS\_LOB are very important while dealing with LOBs as normal functions such as SUBSTR cannot be used with LOBs.

The following is the list of functions and procedures in DBMS\_LOB package.

Method	Description
READ(locator,nobytes,offset, output)	Reads <i>nobytes</i> of a LOB value starting from <i>offset</i> and places the read piece in <i>output</i> .
WRITE(locator, nobytes, offset, buffer)	Writes <i>nobytes</i> from <i>buffer</i> into a LOB starting at <i>offset</i> in a LOB value.

---

APPEND(dest_locator, source_locator)	Appends one LOB value at the end of another LOB value.
ERASE(locator, nobytes, offset)	Erases <i>nobytes</i> from the given <i>offset</i> of a LOB value.
TRIM(locator, newlength)	Reduces the size of a LOB value by trimming characters on the right.
SUBSTR(locator, nobytes, offset)	Extracts a portion of a LOB value and returns that value. The difference between this and READ procedure is that READ procedure places the extracted value in the given variable whereas this returns the value.
COPY(dest,source,nobytes, dest_offset,source_offset)	This is a combination of READ and WRITE procedures. This modifies a LOB value by replacing a part of it with data from another LOB.
GETLENGTH(locator)	Returns the length of a LOB value.
INSTR(locator,pattern,offset,oc cur)	Searches for the given <i>pattern</i> and returns the position at which the pattern is found. If <i>offset</i> is given search starts at the offset. And if occur parameter is also given then it looks for occur number of <i>occurrence</i> .
COMPARE(locator1,locator2,no bytes,offset1,offset2)	Compares two LOB values and returns 0 if they are same, 1 if first one is bigger than second one, and -1 if second one is bigger than first one. <i>offset1</i> and <i>offset2</i> may be used to specify at which position in first and second LOBs the search should begin. <i>nobytes</i> specifies the number of bytes to be compared.

---

**Table 2:** DBMS\_LOB Functions and Procedures

### Examples using DBMS\_LOB Package

The following examples use procedures and functions given in the table 2.

#### INSTR function

The following example looks for the pattern 'Oracle' in CLOB\_COL column of the row where ID is 10 in LOB\_TABLE.

```

declare
    c_lob      CLOB;
    pos       Number(2);
begin
    /* get the value of CLOB_LOB column
       from the row where ID is 10 */

```

```
select  clob_col into  c_lob
from  lob_table
where  id = 10;

/* Use INSTR function to find out the
   position where the pattern 'Oracle'
   occurs in the LOB value.
   Search start at first character and look for the
   first occurrence of the pattern
*/

pos := dbms_lob.instr(c_lob,'Oracle',1,1);

-- if INSTR returns 0 it means pattern is not found
if pos = 0 then
    dbms_output.put_line('Could not find pattern Oracle');
else
    dbms_output.put_line('Found Oracle at = ' || to_char(pos));
end if;

end;
```

### COPY procedure

The following example copies 10 characters of one LOB to another lob.

```
declare
    source_clob  CLOB;
    dest_clob    CLOB;

begin
    /* get CLOB_COLUMNS of ID 20 */
    select  clob_col into  source_clob
    from  lob_table
    where  id = 20;

    /* retrieve CLOB_COLUMN of id 10 for
       updation */

    select  clob_col into  dest_clob
    from  lob_table
    where  id = 15
    for update; -- required to update LOB value

    /*Now copy data with the following parameters
       No. of character to be copied = 10
       Offset in the destination LOB = 50
       Offset in the source LOB = 10 */
```

---

```
dbms_lob.copy( dest_clob,source_clob,10,50,10);
```

```
end;
```

## BFILe Related Procedures and Functions

The following are the procedures and functions in DBMS\_LOB that are related to BFILE data type.

Procedure or Function	Description
FILEOPEN(bfile,mode)	Opens the specified file.
FILECLOSE(bfile)	Closes the specified file
FILECLOSEALL	Closes all open files.
FILEEXISTS(bfile)	Checks whether the file referenced by BFILE locator exists.
FILEGETNAME(loc,dir,fil e)	Gets the name of external file referenced by BFILE locator.
FILEISOPEN(bfile)	Checks whether external file is currently open.

**Table 3:** BFILE Related Procedures and Functions in DBMS\_LOB Package.

The following example displays the directory and filename of a BFILE column.

```
declare
    bf          bfile;
    d varchar2(20);
    f varchar2(20);
begin
    /* get the BFILE_COL value for the ID 20 */
    select bfile_col into bf
    from lob_table
    where id = 20;

    /* get directory name into d and filename into f */
    dbms_lob.filegetname(bf,d,f);

    dbms_output.put_line('Directory: ' || d || ' File : ' || f);
end;
```

The following example checks whether the file referenced by a BFILE\_COL is existing on the disk.

```
declare
    bf bfile;
begin
    /* get the BFILE_COL value for the ID 20 */
```

```
select bfile_col into bf
from lob_table
where id = 20;

/* FILEEXISTS returns non-zero if the given locator
points to an existing file on the disk */

if dbms_lob.fileexists(bf) <> 0 then
    dbms_output.put_line(' BFILE for ID 20 is found ');
else
    dbms_output.put_line(' BFILE for ID 20 is not found');
end if;

end;
```

The following program is used to read the content of BFILE\_COL of LOB\_TABLE.

```
declare
    value BFILE;
    buf char(100);
    amt BINARY_INTEGER := 100;
    pos INTEGER :=1;
BEGIN
    SELECT bfile_col INTO value FROM lob_table
    WHERE id = 1;
    dbms_lob.fileopen(value, dbms_lob.file_readonly);
    LOOP
        dbms_lob.read(value,amt,pos, buf);
        -- process contents of buf
        pos := pos + amt;
    END LOOP;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            dbms_lob.fileclose(value);
end;
/
```

The above program first retrieves BFILE\_COL column's data from LOB\_TABLE. Then FILEOPEN method is used to open the file using BFILE locator. Then LOOP is used to read contents from BFILE using READ method until READ method raises NO\_DATA\_FOUND exception. Each time a chunk of the data is read and processed.

DBMS\_LOB.FILE\_READONLY constant is used to specify that the file is to be opened file read-only.

## Summary

Oracle8 provides a new set of datatypes called LOB data types. These data types can support up to 4GB of data and do not have restrictions that constrain LONG and LONG RAW datatypes. LOB types like BLOB, CLOB and NCLOB store data in the database but BFILE type stores data out of database and maintains only a locator to the actually data.

Oracle8 enhanced its SQL to support LOB data types. Oracle also provided a new package DBMS\_LOB, which contains a collection of procedures and functions that manipulates LOB values.

## Exercises

1. Create APPLICANT table with the following columns.

<u>Column</u>	<u>Description</u>
name	name of the applicant
resume	a column of CLOB type
photo	a column of BFILE to refer to the file that contains photo image.

2. Insert a row into APPLICANT table with the following details.  
NAME - 'Nike', RESUME - 'Subjects : Oracle, Vb, WindowsNT, C++ ... ', and PHOTO - 'nike.bmp' which is in a directory referred by 'IMAGES' directory alias.
3. Write a PL/SQL block to find out whether the pattern 'WindowsNT' is existing in the RESUME column of applicant 'Nike'. If found display the starting position otherwise display error message using DBMS\_OUTPUT package.
4. Check whether the file, which is referred by Nike's record, is existing on the disk.
5. Insert a new row into APPLICANT with the following details.  
NAME - 'Ditchi', RESUME - Empty, PHOTO - Empty.
6. Create IMAGES directory to point to 'c:\images'. And change the value of PHOTO column of 'Ditchi' to 'ditchi.bmp' in IMAGES directory.
7. Recreate IMAGES directory to point to 'd:\images'.
8. Display the length of RESUME column of applicant 'Nike'.