Advanced PLSQ

Lesson – Large Objects(LOBs)

Lesson Objectives

On completion of this lesson on REF Cursors, you will be able to:

- Need for LOBs
- Types of LOBs
- Comparison of LOBs with LONGs
- Components of LOBs
- Working with BFILEs.
- Use of DBMS_LOB for working with LOBs
- Initializing and Populating LOB columns
- Managing Internal LOBs
- Loading BLOB data using BFILE
- SQL LOADER support for LOBs

What is a Large-Object

LOBs

- LOBs are used to store large unstructured data such as text, graphics, films, and sound waveforms.

- A LOB data-type is a data-type that is used to store such unstructured data.

- Normally, structured data(data stored in data-types other than LOB data-types) could be few 100 bytes in length. But data such as video-clippings, images, text, graphics can be thousands of time larger. Also, multimedia data may reside in OS files, which may need to be accessed from a database.

Internal and External LOBs

Categories of LOBs based upon storage-aspect

- **LOBs can be stored internally(inside the database) or in host files(OS files). There are two categories of LOBs :**
  - **Internal LOBs(CLOB, NCLOB, BLOB) : Stored in the database**
  - **External LOBs(BFILE) : Stored outside the database as files. BFILEs can be accessed only in read-only mode from an Oracle Server.**

…….contd

LOB-interpretations by Oracle

- Internal LOBS are categorized as follows based upon their interpretation by Oracle :
  - **The BLOB data-type is interpreted by the Oracle Server as a bit-stream.**
  - **The CLOB data-type is interpreted as a single-byte character stream**
  - **The NCLOB data-type is interpreted as a multi-byte character stream**
- External LOBS are stored outside the Oracle Database, hence Oracle can only point to those file.

Implicit conversion between Char/Varchar and CLOB

- Oracle Database 10G onwards implicit conversion between CLOB and VARCHAR2/Char data types happens.

```
create table x(c1 varchar2(10));
insert into x values('abc');

create table y(c1 clob);
insert into y values('abc');
```

- The following implicit conversions are supported :

```
insert into x select * from y;
insert into y select * from x;
```

- The above conversion is subject to size-limit non-violation of size

Alteration between Char/Varchar and CLOB

- Oracle Database 10G does not allow alterations between CLOB and VARCHAR/Char.

```
create table x(c1 varchar2(10));
insert into x values('abc');

create table y(c1 clob);
insert into y values('abc');
```

- The following alterations will give errors :

```
alter table x modify c1 clob;
            ORA-22858: invalid alteration of datatype
alter table y modify c1 varchar2(10);
            ORA-22859: invalid modification of columns
```

Implicit conversion from CLOB to LONG

- Implicit conversion from CLOB to LONG happens, but LONG to CLOB is not implicitly possible :

```
create table x(c1 long);
insert into x values('abc');

create table y(c1 clob);
insert into y values('abc');
```

- The following is allowed, subject to size-limit :

```
insert into x select * from y;
```

- But, the following is not allowed :

```
insert into y select * from x;
ORA-00997: illegal use of LONG datatype
```

Direct alteration from LONG to CLOB

- Direct alteration of datatype from LONG to CLOB is possible :

```
create table x(c1 long);
insert into x values('abc');
alter table x modify c1 clob;
```

- But, the following is not allowed :

```
create table y(c1 clob);
insert into y values('abc');
alter table y modify c1 long;
ORA-22859: invalid modification of columns
```

Implicit conversion from BLOB to LONG RAW

- Implicit conversion from BLOB to LONG RAW happens, but LONG RAW to BLOB is not implicitly possible :

```
create table x(c1 long) raw;
insert into x values('abc');

create table y(c1 blob);
insert into y values('abc');
```

- The following is allowed, subject to size-limit :

```
insert into x select * from y;
```

- But, the following is not allowed :

```
insert into y select * from x;
ORA-00997: illegal use of LONG datatype
```

Direct alteration from LONG RAW to BLOB

- Direct alteration of datatype from LONG RAW to BLOB is possible :

```
create table x(c1 long raw);
insert into x values('abc');
alter table x modify c1 cbob;
```

- But, the following is not allowed :

```
create table y(c1 blob);
insert into y values('abc');
alter table y modify c1 long raw;
ORA-22859: invalid modification of columns
```

1.9: LOBS versus LONGs
Comparison of LOB datatypes with the LONG datatypes

- Long and Long Raw data-types were previously used for large text and unstructured data. These data-types are now superseded by the LOB data-types. Oracle Database 10G provides functions to migrate from LONG columns to LOB columns.

- Following is a comparison of LOBs with LONGs :
  - A table can have multiple LOB columns, but a table can have only one LONG column
  - LOBs can be upto 4GB. LONGs can be upto 2GB.
  - LOBs can be Object-type attributes. LONGs cannot.
  - LOBs return the locator. LONGs return data.

- LOBs store a locator in the table and the data in a different LOB segment. LONGs store all the data in the same data block.

TO_LOB function

- The TO_LOB function can be used to convert a LONG or LONG RAW value in a column to CLOB or BLOB value respectively:

```
create table olddata(c long);
insert into olddata values('hi how are you?');

create table newdata(c clob);
insert into newdata select c from olddata;
```

This gives an error : ORA-00997: illegal use of LONG datatype

```
insert into newdata select to_lob(c) from olddata;
```

This will work through the use of TO_LOB

TO_LOB function ……contd

- The TO_LOB function can be used to convert a LONG or LONG RAW value in a column to CLOB or BLOB value respectively:

```
create table olddatar(c long raw);
insert into olddatar values('abcd');

create table newdatar(c blob);
insert into newdatar select c from olddatar;
```

This gives an error : ORA-00997: illegal use of LONG datatype

```
insert into newdatar select to_lob(c) from olddatar;
```

This will work through the use of TO_LOB

LOB Locator and LOB value

➢LOB value :	The data that constitutes the real object being stored
➢LOB locator :	A pointer to the location of the LOB value stored in the database

➢Regardless of where the value of the LOB is stored, a locator should be stored in the row. The locator is like a pointer to the actual location of the LOB value. Thus, a LOB column does not contain the data, it contains the locator(pointer) to the LOB value.

Applicabilty on LOBs

- The internal LOBs are stored in the Oracle Server. A BLOB, CLOB, NCLOB can be :
  - An attribute of a user-defined type
  - A column in a table
  - A bind or host variable(Variable x clob)
  - A PLSQL variable, parameter.

- **External LOB : BFILE**

The BFILE data-type supports an external or file-based large object as :

- An attribute of a user-defined type
- A column in a table
- A bind or host variable(Variable x clob)
- A PLSQL variable, parameter.

Read-only BFILE

- The operations that are required for using BFILEs are possible through the DBMS_LOB package.
- BFILEs are read-only.
- Integrity and durability must be provided by the OS.
- The file must be created and placed in appropriate directory, giving the Oracle process privileges to read the file.
- When the LOB row is deleted, the Oracle Server does not delete the file.
- The maximum size of an External LOB depends on the OS, but cannot exceed 4 GB.
- Oracle Backup and Recovery methods support only the LOB locators, not the physical files.

1.14: Directory Object

Abstraction for an OS folder path

- A DIRECTORY is a non-schema database object that enables the access and usage of BFILEs in Oracle Database.

- A DIRECTORY specifies an alias for a directory on the file system of the server under which a BFILE is located. By granting suitable privileges for a DIRECTORY object, you can provide secure access to files in the corresponding directories on a user-by-user basis(certain directories can be made read-only or inaccessible to certain users).

```
GRANT READ ON DIRECTORY <DIRECTORY_name> to users;
GRANT WRITE ON DIRECTORY <DIRECTORY_name> to users;
GRANT ALL ON DIRECTORY <DIRECTORY_name> to users;
```

- Also, these DIRECTORY objects can be used while referring to files(open, close, read, and so on) in PLSQL. Thus, it provides application abstraction from hard-coded path-names and gives flexibility in managing file-locations.

- The DIRECTORY object is owned by SYS and created by the DBA(or any other user with CREATE ANY DIRECTORY privilege). The DIRECTORY object has object-privileges unlike other non-schema objects. Privileges to DIRECTORY object can be Granted and Revoked.

- The DIRECTORY object information is stored in the DBA_DIRECTORIES and ALL_DIRECTORIES data-dictionary views.

Guidelines for Directory Object

- DIRECTORY objects should not point to paths that contain your database files(data files, log files and/or control files), because tampering with these files could corrupt the database.
- The CREATE ANY DIRECTORY and DROP ANY DIRECTORY system privileges should be used carefully and not granted to users indiscriminately.
- All DIRECTORY objects are non-schema objects, all are owned by SYS.
- First create the OS Directory paths with appropriate permissions and then create the DIRECTORY objects, as Oracle does not create the OS paths.
- If you migrate the database to a different OS, you will be required to change the path value of the DIRECTORY object using the REPLACE option

```
create or replace directory x as 'E:\hemant';
```

Managing BFILEs

- Managing BFILEs requires coordination between the Database or System Administrator and between the Developer and the User of the files.

- The Database or System Administrator should perform the following privileged tasks :
  - Create the OS Directory and set permissions so that the Oracle Server can read the contents of the OS Directory. Load the files in the OS Directory
  - Create the database DIRECTORY object that references the OS Directory
  - Grant READ privileges on the database DIRECTORY object to database users requiring access to it.

Managing BFILEs ….contd

- The Designer, application developer or user should perform the following tasks :
  - Create a table containing a column with BFILE data-type
  - Insert rows into the table using the BFILENAME function to populate the BFILE column associating the field to an OS file in the named DIRECTORY
  - Write PLSQL sub-programs that :
    - Declare and Initialize the BFILE LOB Locator
    - Select the row and column containing the BFILE into the LOB locator
    - Read the BFILE with a DBMS_LOB function, using the locator file reference

Populating BFILE Columns

- At the OS, create folder ‘D:\Photos’ containing video files ‘video1.mp4’ and ‘video2.mp4’
- Grant appropriate privileges to scott user :

```
Connect system/abc;
Grant CREATE ANY DIRECTORY to scott;
Grant DROP ANY DIRECTORY to scott;
```

- Connect as scott :

```
Conn scott/abc;


Create table employee(empid number, empname varchar2(30));
Insert into employee values(1,’Ram’);
Insert into employee values(2,’Laxman’);
```

Populating BFILE Columns ….contd

- Alter the table to add a column named video

```
Alter table employee add video BFILE;
```

- Create a Directory Object and verify the creation :

```
Create or replace Directory data_files as 'd:\photos';

select owner, DIRECTORY_NAME,DIRECTORY_PATH from all_directories where
directory_name='DATA_FILES';

Update employee set video=BFILENAME('data_files', 'video1.mp4') where empid=1;
Update employee set video=BFILENAME('data_files', 'video2.mp4') where empid=2;

select * from employee;
```

Initialize a BFILE column

- The BFILENAME function is a built-in function that you use to initialize a BFILE column using 2 parameters
  - FUNCTION BFILENAME(directory_alias in varchar2, filename in varchar2) RETURN BFILE;
  - directory_alias is the name of the DIRECTORY database object that references to the OS directory containing the files
  - filename is for the name of the BFILE to be read
- The BFILENAME function returns(creates) a pointer (or LOB locator) to the external file stored in a physical directory, which is assigned a Directory alias name that is used as the first parameter of the function.
- You can populate the BFILE column using the BFILENAME function in either :
  - The VALUES clause of the INSERT command
  - The SET clause of the UPDATE command;

1.16: BFILENAME

Initialize a BFILE column ……contd

```
Create table newemployee(empid number, empname varchar2(30), video BFILE);
Insert into newemployee values(1,'Ram', BFILENAME('data_files', 'video1.mp4'));
Insert into newemployee values(2,'Laxman', BFILENAME('data_files', 'video2.mp4'));
```

An UPDATE operation can be used to change the pointer reference target of the BFILE column (that is associate a particular rows BFILE column with a new file at the OS level).

- A BFILE column can also be initialized to NULL value and updated later with the BFILENAME function, as shown in the above two cases.(Initially when the column of type BFILE is created, it’s initial value is NULL);
- You may also update a BFILE column to NULL as follows :

```
Update newemployee set video=null;
Update newemployee set video= bfilename('data_files', 'video2.mp4') where empid=1;
Update newemployee set video= bfilename('data_files', 'video1.mp4') where empid=2;
```

- After a BFILE column has been associated with a file, subsequent read operations on the BFILE can be performed by using the DBMS_LOB package. However, these files are read-only when accessed through the BFILEs. Therefore, these files cannot be updated or deleted through BFILEs.

BFILE operations using DBMS_LOB

- At the OS, create folder ‘D:\Photos’ containing video files ‘saloni.mp4’ and ‘hetal.mp4’

```
create table employeenew(empid number, ename varchar2(20), video BFILE);
insert into employeenew(empid, ename) values(1,'saloni');
insert into employeenew (empid, ename) values(2,'hetal');
```

- At the OS, create folder ‘D:\Photos’ containing video files ‘saloni.mp4’ and ‘hetal.mp4’

```
Create or replace Directory data_files as 'd:\photos';
```

BFILE operations using DBMS_LOB …contd

```
Create or Replace procedure set_video(dir_alias varchar2) is
   Filename varchar2(40);
   File_ptr BFILE;
   Cursor csr is Select ename from employeenew for update;
Begin
   For rec in csr loop
                Filename:=rec.ename || '.MP4';
                File_ptr:=BFILENAME(dir_alias, filename);
                DBMS_LOB.FILEOPEN(file_ptr);
                Update employeenew set video=file_ptr where current of csr;
                Dbms_output.put_line('FILE '||filename||' SIZE '||dbms_lob.getlength(file_ptr));
                DBMS_LOB.FILECLOSE(file_ptr);
   End loop;
End set_video;
/
EXEC set_video('DATA_FILES');
```

DBMS_LOB.FILEEXISTS

- The BFILENAME function directly sets the file_ptr variable to the specified filename in the specified directory object, even if the specified file does not exist.

- Hence, DBMS_LOB.FILEEXISTS function can verify if the file exists in the OS. The function returns 0 if the file does not exist, and returns 1 if the file exists. The DBMS_LOB.FILEEXISTS function expects a BFILE locator as a parameter, and returns an Integer 1 or 0.

```
insert into employeenew (empid, ename) values(3,'hemant');
insert into employeenew (empid, ename) values(4,'manoj');
```

DBMS_LOB.FILEEXISTS ……contd

```
Create or Replace procedure setvideo(dir_alias varchar2) is
        Filename varchar2(40);
        File_ptr BFILE;
        A boolean;
        Cursor csr is Select ename from employeenew for update;
Begin
        For rec in csr loop
                Filename:=rec.ename || '.MP4';
                File_ptr:=BFILENAME(dir_alias, filename);
                a:=dbms_lob.fileexists(file_ptr)=1;
                If a then
                        DBMS_LOB.FILEOPEN(file_ptr);
                        Update employeenew set video=file_ptr where current of csr;
```

DBMS_LOB.FILEEXISTS ……contd

```
                    Dbms_output.put_line('FILE          '||filename||'          SIZE
  '||dbms_lob.getlength(file_ptr));
                    DBMS_LOB.FILECLOSE(file_ptr);
              Else
                    Dbms_output.put_line('FILE not found for this employee' || rec.ename);
              End if;
        End loop;
End setvideo;
/

EXEC setvideo('DATA_FILES');
```

1.19: TO_CLOB and TO_BLOB

PLSQL conversion functions

- TO_CLOB : Converts LONG to CLOB through PLSQL

```
create table x(c1 long, c2 varchar2(10), c3 char(10));
insert into x values('abc', 'xyz','pqr');
insert into x values('aaa', 'bbb','ccc');

create table y(c1 clob, c2 clob, c3 clob);
```

- The following will give an error at the SQL-engine level

```
insert into y(c1) select to_clob(c1) from x;
ERROR at line 1:
ORA-00932: inconsistent datatypes: expected NUMBER got LONG
```

PLSQL conversion functions ....contd

- Hence, following use of TO_CLOB is needed thru PLSQL :

```
declare
        a long;
        b clob;
begin
        select c1 into a from x where c2='xyz';
        b:=to_clob(a);
        insert into y(c1) select b from dual;
end;
/
```

```
SQL> select c1 from y;
```

1.19: TO_CLOB and TO_BLOB

PLSQL conversion functions ……..contd

- This gives error as to_clob only works on plsql variables and not on columns of tables

```
begin
	insert into y(c1) select to_clob(c1) from x where c2='xyz';
end;
/
```

Initializing and Populating LOB columns

- LOB columns get implicit nulls

```
Drop table employee;
create table employee(empid number(6), name varchar2(10), resume clob, picture blob);
insert into employee(empid, name) values(100,'A');
```

- LOB columns get explicit nulls

```
insert into employee values(101,'B', null, null);
```

```
select count(*) from employee where resume is null;
```

Output : 2

```
select count(*) from employee where resume is null;
```

Output : 2

Initializing and Populating LOB columns .....contd

- The below will not put a NULL value, but now the LOB columns contain locators with empty contents, that is, the LOB locators are pointing to an empty LOB value.

```
insert into employee values(102,'C',empty_clob(), empty_blob());

select count(*) from employee where resume is null;
```

Output : 2

```
select count(*) from employee where resume is null;
```

Output : 2

Initializing and Populating LOB columns …..contd

- Thus, you can initialize a columns LOB locator value with the EMPTY_CLOB() and EMPTY_BLOB() functions for CLOB/NCLOB and BLOB columns respectively. That is, the LOB columns will not then contain NULL values, but will contain LOB locators pointing to empty LOB value.
- The EMPTY_CLOB() and EMPTY_BLOB() functions can also be used in the DEFAULT option in a CREATE TABLE command as follows :

```
create table myemployee(empid number(6), name varchar2(10), resume clob DEFAULT EMPTY_CLOB(), picture blob DEFAULT EMPTY_BLOB());

insert into myemployee(empid, name) values(100,'A');

insert into myemployee(empid, name) values(101,'B');
```

- Here, the LOB locator values are populated in their respective columns when a row is inserted into the table and the LOB columns have not been specified in the INSERT command.

Updating LOB by Using DBMS_LOB in PLSQL

```
create table emp_hiredata(employee_id number(6), full_name varchar2(45), resume clob default empty_clob(), picture blob default empty_blob());

insert into emp_hiredata(employee_id, full_name, resume, picture) values(405, 'Marvin Ellis', EMPTY_CLOB(), NULL);

insert into emp_hiredata(employee_id, full_name, resume, picture) values(170, 'Ram Shankar', EMPTY_CLOB(), EMPTY_BLOB());

update emp_hiredata set resume=’Date of birth is 8th July, 1976’, picture=empty_blob() where employee_id=405;

update emp_hiredata set resume=’Date of birth is 2nd December, 1977’ where employee_id=170;
commit;
```

Updating LOB by Using DBMS_LOB in PLSQL …contd

```
declare
        lobloc CLOB;                        --serves as LOB locator
        text varchar2(50):='MCA M.PHIL';
        amount number;      --amount to be written
        offset number;      --where to start writing
begin
        select resume into lobloc from emp_hiredata where employee_id=405 for update;
        offset:=DBMS_LOB.GETLENGTH(lobloc)+2;
        amount:=length(text);
        DBMS_LOB.WRITE(lobloc, amount, offset, text);
        Text :='MCA M.PHIL MBA';
        select resume into lobloc from emp_hiredata where employee_id=170 for update;
        amount:=length(text);
        DBMS_LOB.WRITEAPPEND(lobloc, amount, text);
        Commit;
End;
/
```

Updating LOB by Using DBMS_LOB in PLSQL …contd

- Here, the LOBLOC serves as the LOB locator, and the AMOUNT is set to the length of the text you want to add.
- The SELECT FOR UPDATE statement locks the row and returns the LOB locator for the RESUME LOB column.
- The WRITE packaged procedure is to write the text into the LOB value at the specified offset.
- WRITEAPPEND appends to the existing LOB value.
- To verify the data changes in the table :

```
select resume from emp_hiredata;
```

1.21: DBMS_BLOB in PLSQL

Updating LOB by Using DBMS_LOB in PLSQL …contd

- This is how we used to fetch a CLOB column in releases prior to Oracle 9i, where it was not possible to fetch CLOB column directly into a character column. The column value needed to be bound to a LOB locator, which was accessed by the DBMS_LOB package. Now, you can directly fetch a CLOB column by binding it to a character variable.

```
declare
        text varchar2(100);
begin
        select resume into text from emp_hiredata where employee_id=405 for update;
        text:=text||' PLSQL';
        update emp_hiredata set resume=text where employee_id=405;
        select resume into text from emp_hiredata where employee_id=170 for update;
        text:=text||' ADV PLSQL';
        update emp_hiredata set resume=text where employee_id=170;
        commit;
end;
/
select resume from emp_hiredata;
```

Updating LOB by Using DBMS_LOB in PLSQL …contd

- In oracle 10g, CLOB values are implicitly converted to varchar2. In prior versions, you first retrieved the CLOB locator value from the column into a CLOB variable, and then read the LOB contents specifying the amount and offset in the DBMS_LOB.READ procedure as follows :

```
declare
        Rlob clob;
        Text varchar2(2000);
        Amt number:=4;
        Offset number:=3;
begin
        Select resume into rlob from emp_hiredata where employee_id=170;
        DBMS_LOB.READ(rlob, amt, offset, text);
        DBMS_OUTPUT.PUT_LINE('TEXT is '|| text);
End;
/
```

Setting LOBs to Empty-LOB pointer or NULL

- You can delete a row containing LOBs :

```
Delete from emp_hiredata where employee_id=405;
```

- Disassociation of a LOB value from a row :

```
Update emp_hiredata set resume=empty_clob() where employee_id=170;
Select count(*) from emp_hiredata where resume is null;
Or
Update emp_hiredata set resume=null where employee_id=405;
Select count(*) from emp_hiredata where resume is null;
Or
Update emp_hiredata set resume=" where employee_id=170;
Select count(*) from emp_hiredata where resume is null;
```

- Thus, to destroy only the reference to the LOB, you must update the row by replacing the LOB column value with NULL or an empty string(''), or by using the empty_clob()/empty_blob() function.

1.23: BLOB

Storing a image file in oracle database

```
create table loadalbum(name varchar2(100), image blob);
CREATE OR REPLACE DIRECTORY DOCUMENTS AS 'd:\photos';

declare
        l_blob    blob;
        l_bfile   bfile;
begin
        insert into loadalbum values ('saloni', EMPTY_BLOB()) returning image into l_blob;
        -- First create a Empty binary large object and get a reference
        l_bfile := bfilename('DOCUMENTS', 'salonig.jpg' );
        --Get the pointer to a file in directory
        dbms_lob.fileopen(l_bfile );
        -- Open file
        dbms_lob.loadfromfile( l_blob, l_bfile, dbms_lob.getlength( l_bfile ) );
        --loads bfile data into internal lob
        dbms_lob.fileclose( l_bfile );
        --closes the file
end;
/
```

Doing bulk upload of images to database

- You can use SQL*Loader utility to bulk upload images as follows :

```
CREATE TABLE photoalbum (photolob BLOB);
```

- Create a file (example photos.txt) in the folder ‘D:\PHOTOS’ which will contain list of images to be uploaded.

```
saloni.jpg
kiyara.jpg
hetal.jpg
tiny.jpg
aarav.jpg
```

Doing bulk upload of images to database ……contd

- Create a control file required by SQL*Loader to upload data. Create new file called **loadphotos.ctl** in the ‘D:\PHOTOS’ and insert following content into it.

```
load data
infile photos.txt
into table photoalbum
(ext_fname filler char(200),
photolob lobfile(ext_fname) terminated by EOF)
```

- The meaning of above code is “load the data listed in the file photos.txt into a table called photoalbum. The data will be loaded into the column of that table called ‘photoblob’ and has lobfile characteristics, that is, it is binary data. Expect the file names for the binary files being loaded to be up to 200 characters in length. When you reach the end of the list of photos, terminate the load process”.

Doing bulk upload of images to database ……contd

- Please note that photos.txt is used in control file and we are not giving absolute path, but relative path. So control file **loadphotos.ctl** and **photos.txt** should be in same directory and so also all image files to be loaded

- Go to command prompt and in the folder ‘d:\photos’, run the SQL LOADER :

```
sqlldr scott/tiger control=loadphotos.ctl

Select count(*) from photoalbum;
```

1.25: BLOB bulk Upload

Uploading Word Document to oracle database

- Create the following table :

```
CREATE TABLE my_docs
(doc_id   NUMBER,
 bfile_loc BFILE,
doc_title VARCHAR2(255),
doc_blob  BLOB DEFAULT EMPTY_BLOB() );    // Default value will be empty binary large object
```

- Create directory object as follows :

```
CREATE OR REPLACE DIRECTORY DOC_DIR AS  'D:\photos';
```

Uploading Word Document to oracle database….contd

- Create a procedure for uploading the file. Here, inputs will be file_name and file_id(just for uniquely identifying each file/row in the table) as follows :

```
Create or replace PROCEDURE load (in_doc IN VARCHAR2, in_id  IN NUMBER) IS
temp_blob BLOB := empty_blob();
bfile_loc         BFILE;
BEGIN
bfile_loc := BFILENAME('DOC_DIR', in_doc);
INSERT INTO my_docs (doc_id, bfile_loc, doc_title)  VALUES (in_id, bfile_loc, in_doc);
SELECT doc_blob INTO temp_blob FROM my_docs WHERE doc_id = in_id FOR UPDATE;
DBMS_LOB.OPEN(bfile_loc, DBMS_LOB.LOB_READONLY);
DBMS_LOB.OPEN(temp_blob, DBMS_LOB.LOB_READWRITE);
DBMS_LOB.LOADFROMFILE(temp_blob, bfile_loc, dbms_lob.getlength(bfile_loc));
DBMS_LOB.CLOSE(temp_blob);
DBMS_LOB.CLOSE(bfile_loc);
COMMIT;
END ;
```

Uploading Word Document to oracle database….contd

- Suppose you want to upload app1.docx file present in the DOC_DIR directory(D:\photos\ at OS level) created above.

- The doc name is app1.docx in the D:\photos\ folder at the OS level. Execute the above procedure as given below.

```
exec load('app1.docx', 1);
```