Manejo de Oracle Large Objects (LOB)

Por Francisco Riccio

Introducción

Oracle desde la versión 8i nos provee un tipo de dato llamado LOB, el cual nos permite almacenar largas estructuras de información estructurada y no estructurada como texto , gráficos, audio y video. Asimismo la información multimedia puede residir tanto en la misma base de datos como en el sistema operativo. Este tipo de dato se crea en reemplazo a los tipos de datos antiguos que existían como: LONG, RAW y LONG RAW debido a todos las restricciones y problemas de mantenimiento que presentaban.

La capacidad máxima que un LOB puede albergar es de 4 GB.

Los LOBS se categorizan en CLOB (almacenan texto que contiene grandes cantidades de bytes), NCLOB (es similar al CLOB solo que almacena texto cuyo juego de caracteres está definido por el National Character Set de la base de datos), BLOB (almacena información multimedia dentro de la base de datos), BFILE (similar al BLOB solo que la información multimedia está almacenada en el sistema operativo).

La versión Oracle Database 11g ha hecho varias mejoras sobre los LOB presentando una nueva propuesta llamada Secure Files, el cual entrega mejores tiempos de respuesta en el acceso a los datos, ahorro en espacio y seguridad.

Implementación

Todo tipo de dato LOB tiene dos partes con que trabajaremos:

a) LOB Value, el cual constituye el valor a almacenar por ejemplo: un texto o contenido multimedia.

b) LOB Locator, es un puntero a la ubicación del valor LOB (LOB Value) que es almacenado en la base de datos.

Si el texto o información multimedia se guarda dentro de la base de datos, el contenido se almacena en un segmento separado de la tabla. Este segmento es de tipo LOB y almacena solo el LOB Value mientras la tabla que se definió con él campo LOB solo lleva el LOB Locator como puntero al segmento LOB.

CLOB

Los CLOB almacenan texto que contienen grandes cantidad de bytes. Reemplaza al tipo de dato LONG.

Existe automáticamente una conversión implícita entre los CLOB y VARCHAR2.

Crearemos una tabla con un campo CLOB:

```
SQL> create table TABLA_CLOB (id number, valor CLOB);

Table created.
```

Oracle recomienda que inicialicemos un campo CLOB con un LOB Locator vació y no dejarlo como NULL. Para realizar esto podemos hacerlo mediante la función EMPTY_CLOB() desde la creación de la tabla o después, por ejemplo:

```
SQL> create table TABLA_CLOB (id number, valor CLOB default EMPTY_CLOB());

Table created.
```

ó

```
SQL> alter table TABLA_CLOB modify valor default EMPTY_CLOB();

Table altered.
```

Nota: La función EMPTY_CLOB asegura que no habrá ningún valor en el campo mientras el valor NULL almacena el valor NULO.

Durante la creación de la tabla que alberga campos CLOB, podemos indicar que el LOB Value (contenido) sea almacenado en otro tablespace que es lo más recomendable, ejemplo:

```
SQL> create table TABLA_CLOB (id number, valor CLOB default EMPTY_CLOB())
     LOB(valor) STORE AS (tablespace EXAMPLE);

Table created.
```

En el ejemplo definimos que el contenido multimedia será almacenado en el tablespace EXAMPLE, por lo cual se creará un segmento de tipo LOB en dicho tablespace. Podemos validar está información en la vista DBA_SEGMENTS, ejemplo:

```
SQL> select segment_name, segment_type, tablespace_name from dba_segments where owner='FRICCIO';

SEGMENT_NAME                   SEGMENT_TYPE        TABLESPACE_NAME
------------------------------ ------------------- --------------------
TABLA_CLOB                     TABLE               USERS
SYS_LOB0000077566C00002$$      LOBSEGMENT          EXAMPLE
SYS_IL0000077566C00002$$       LOBINDEX            EXAMPLE
```

Para insertar un valor sobre un campo CLOB lo hacemos tan similar como si fuera un campo VARCHAR, ejemplo:

```
SQL> insert into TABLA_CLOB values (1,'Esto es un texto');

1 row created.

SQL> commit;

Commit complete.
```

Revisaremos algunas funciones útiles:

a) Si deseamos obtener una parte del contenido de un CLOB usamos la función DBMS_LOB.SUBSTR, ejemplo:

```
SQL> select valor from TABLA_CLOB;

VALOR
--------------------------------------------------------
Esto es un texto

SQL> select DBMS_LOB.SUBSTR(valor,5,12) from TABLA_CLOB;

DBMS_LOB.SUBSTR(VALOR,5,12)
--------------------------------------------------------
texto
```

En este ejemplo obtenemos desde la posición 12 del texto 5 caracteres.

Nota: Esta función también trabaja con los tipos de datos BLOB y BFILE.

b) Si deseamos obtener la posición de un texto usamos la función DBMS_LOB.INSTR, ejemplo:

```
SQL> select * from TABLA_CLOB;

        ID VALOR
---------- --------------------------------------------------
         1 Esto es un texto

SQL> select DBMS_LOB.INSTR(valor,'e',1,2) from TABLA_CLOB;

DBMS_LOB.INSTR(VALOR,'E',1,2)
-----------------------------
                           13
```

En este ejemplo conseguimos la posición de la letra "e" en su segunda ocurrencia a partir del primer

carácter del texto.

c) Si deseamos agregar más texto a un CLOB usamos la función DBMS_LOB.WRITEAPPEND, ejemplo:

```
set serveroutput on
declare
 v_clob CLOB;
begin
 select valor into v_clob from TABLA_CLOB where id=1 for update;
 /*Modificamos el valor del CLOB*/
 DBMS_LOB.WRITEAPPEND(v_clob,6,' de ejemplo');
 dbms_output.put_line(v_clob);
 COMMIT;
end;
/
```

Donde la función DBMS_LOB.WRITEAPPEND pide de parámetro la variable CLOB a modifcar, la cantidad de caracteres y el texto a añadir.

BLOB

Reemplaza al tipo de dato LONG RAW y almacena el contenido multimedia dentro de la base de datos.

Para trabajar con BLOB y BFILES (más adelante se especifica) se requiere de Objetos Directorios en la base de datos. Los Objetos Directorios no son objetos que le pertenecen a un esquema, todos los directorios creados son adueñados por el usuario SYS. Para crear directorios necesitamos el privilegio de sistema CREATE ANY DIRECTORY. Estos Objetos Directorios serán una referencia a una ubicación de un directorio del sistema operativo.

Su sintaxis es la siguiente:

SQL> create or replace directory <nombre_directorio> as '<ruta_so>';

Ejemplo:

```
SQL> create or replace directory DIR_TMP as '/tmp';

Directory created.
```

Donde podemos entregar permisos de lectura y escritura a otros usuarios de la siguiente manera:

SQL> grant read,write on directory <nombre_directorio> to <nombre_usuario>;

```
SQL> grant read,write on directory DIR_TMP to HR;

Grant succeeded.
```

Crearemos una tabla con un campo BLOB:

```
SQL> create table TABLA_BLOB (id number, valor BLOB);

Table created.
```

Oracle recomienda que inicialicemos un campo BLOB con un LOB Locator vació y no dejarlo como NULL. La función EMPTY_BLOB nos ayuda en este propósito, por ejemplo:

```
SQL> create table TABLA_BLOB (id number, valor BLOB default EMPTY_BLOB());

Table created.
```

ó

```
SQL> alter table TABLA_BLOB modify valor default EMPTY_BLOB();

Table altered.
```

En la creación de la tabla que contiene campos BLOB podemos indicar que el LOB Value (contenido) sea almacenado en otro tablespace que es lo recomendable, ejemplo:

```
SQL> create table TABLA_BLOB (id number, valor BLOB default EMPTY_BLOB())
     LOB(valor) STORE AS (tablespace EXAMPLE);

Table created.
```

Aquí definimos que el contenido multimedia será almacenado en el tablespace EXAMPLE, por lo cual se creará un segmento de tipo LOB en dicho tablespace. Podemos validar está información en la vista DBA_SEGMENTS, ejemplo:

```
SQL> select segment_name, segment_type, tablespace_name from dba_segments where owner='FRICCIO';

SEGMENT_NAME                   SEGMENT_TYPE       TABLESPACE_NAME
------------------------------ ------------------ --------------------
TABLA_BLOB                     TABLE              USERS
SYS_LOB0000077557C00002$$      LOBSEGMENT         EXAMPLE
SYS_IL0000077557C00002$$       LOBINDEX           EXAMPLE
```

Para almacenar el contenido multimedia en un campo BLOB, debemos ejecutar un script como el que se adjunta.

```
set serveroutput on
declare
 v_blob  BLOB;
 v_bfile BFILE;
begin
 /*Insertamos una fila y nos retornará el BLOB*/
 insert into TABLA_BLOB(id,valor)
 values (1,EMPTY_BLOB()) returning valor into v_blob;
 /*Copiaremos el contenido binario de una imagen en una variable BLOB*/
 v_bfile:=BFILENAME('DIR_TMP','oracle.jpg');
 DBMS_LOB.OPEN(v_bfile,DBMS_LOB.LOB_READONLY);
 DBMS_LOB.LOADFROMFILE(v_blob,v_bfile,DBMS_LOB.GETLENGTH(v_bfile));
 DBMS_LOB.CLOSE(v_bfile);
 COMMIT;
end;
/
```

El objetivo es obtener el valor de LOB Locator y asignarle el valor del contenido multimedia.

Se adjunta una muestra de cómo se almacena el contenido en formato binario.

```
SQL> select VALOR from tabla_blob;

VALOR
--------------------------------------------------------------------------------
FFD8FFE000104A46494600010201020FA02FA0000FFED09EA50686F746F73686F7020332E30003842
494D03E900000000007800030000004800480000000002D80228FFE1FFE202F902460347052803FC
```

En el siguiente ejemplo copiaremos la información de un BLOB almacenado en la base de datos en un archivo en el sistema operativo; para realizar esta labor nos vamos a apoyar del paquete UTL_FILE.

```
set serveroutput on
declare
 v_blob     BLOB;
 v_archivo  UTL_FILE.FILE_TYPE;
 v_offset   number:=1;
begin
 select valor into v_blob from TABLA_BLOB where id=1;
 /*Crearemos un archivo que se escribira en formato binario
   en chunks de 32K, el cual es el maximo tamano para un chunk*/
 v_archivo:=UTL_FILE.FOPEN('DIR_TMP','imagen.jpg','WB',32767);
 loop
  /*El criterio de salida es cuando el valor del offset sea el tamaño del archivo*/
  exit when v_offset>DBMS_LOB.GETLENGTH(v_blob);
  /*A partir de un offset sobre el BLOB se leera en bloques de 32K*/
  /*La informacion leida se escribira en el archivo*/
  UTL_FILE.PUT_RAW(v_archivo,DBMS_LOB.SUBSTR(v_blob,32767,v_offset));
  v_offset:=v_offset+32767;
 end loop;
 UTL_FILE.FCLOSE(v_archivo);
end;
/
```

BFILES

Los BFILEs almacenan información multimedia pero el contenido es almacenado físicamente en el sistema operativo, por dicha razón los BFILES solo se pueden acceder en modo lectura es decir no podemos hacer modificaciones o cambios al contenido.

El campo BFILE solo almacena el LOB Locator hacia una dirección donde se encuentra físicamente el contenido multimedia en el Sistema Operativo. Es importante que si nuestros sistemas cuentan con BFILES considerar en nuestra política de backups incluir los directorios de los archivos que son referenciados en las columnas BFILE de nuestra base de datos.

Debemos tener presente que existe una máxima cantidad de archivos concurrentes que pueden ser leídos por sesión, la cual está limitada por el parámetro SESSION_MAX_OPEN_FILES (el valor por default es 10), por lo cual limita también la cantidad de lecturas concurrentes sobre campos BFILE por sesión.

Crearemos una tabla con un campo BFILE:

```
SQL> create table TABLA_BFILE (id number, valor BFILE);

Table created.
```

Insertaremos un contenido multimedia en la tabla creada:

```
SQL> insert into TABLA_BFILE values (1,BFILENAME('DIR_TMP','oracle.jpg'));

1 row created.

SQL> commit;

Commit complete.
```

Podemos apreciar que para insertar un contenido multimedia en un campo BFILE se utiliza la función BFILENAME, el cual crea un LOB Locator que es un puntero hacia el archivo oracle.jpg que se encuentra en el Objeto Directorio DIR_TMP. El directorio DIR_TMP fue creado en la sección BLOB y hace referencia al directorio /tmp del Sistema Operativo.

El paquete DBMS_LOB tiene una serie de funciones que nos ayudan a operar sobre los archivos, una de ellas es por ejemplo la función GETLENGTH que nos devuelve el tamaño del archivo leído.

Ejemplo:

```
set serveroutput on
declare
 v_bfile BFILE;
begin
 select valor into v_bfile from TABLA_BFILE where id=1;
 dbms_output.put_line('El archivo ocupa: '||DBMS_LOB.GETLENGTH(v_bfile)||' bytes.');
end;
/
```

La función FILEGETNAME nos devuelve el nombre del directorio y archivo que apunta un BFILE.

```
set serveroutput on
declare
 v_bfile BFILE;
 v_directorio VARCHAR2(255);
 v_archivo VARCHAR2(255);
begin
 select valor into v_bfile from TABLA_BFILE where id=1;
 DBMS_LOB.FILEGETNAME(v_bfile,v_directorio,v_archivo);
 dbms_output.put_line('El nombre del archivo es: '||v_archivo);
end;
/
```

La función FILEEXISTS nos indica si el archivo que apunta el BFILE existe en el sistema operativo.

```
set serveroutput on
declare
 v_bfile BFILE;
begin
 select valor into v_bfile from TABLA_BFILE where id=1;
 if DBMS_LOB.FILEEXISTS(v_bfile)=1 then
   dbms_output.put_line('El archivo existe.');
 else
   dbms_output.put_line('El archivo no existe.');
 end if;
end;
/
```

Nota: Muchas de estas funciones mostradas son válidas también para los tipos de datos BLOB.

Temporary LOB

Los temporary LOB son variables temporales que solo viven durante el ciclo de vida de una sesión y almacenan un tipo de dato LOB. Una de sus características es que no generan redo por lo cual los hace más rápido respecto a los LOB convencionales y no soporta el uso de la función EMPTY_CLOB/EMPTY_BLOB. Al crearse automáticamente son creados como vacíos.

Para crear un temporary LOB usamos los procedures CREATETEMPORARY y FREETEMPORARY (limpia su información de la memoria) del paquete DBMS_LOB.

```
set serveroutput on
declare
 v_blob BLOB;
begin
 DBMS_LOB.CREATETEMPORARY(v_blob,TRUE);
 /*Aqui lo trabajamos como si fuera un LOB cualquiera*/
 DBMS_LOB.FREETEMPORARY(v_blob,TRUE);
end;
/
```

SecureFile LOB

SecureFile LOB es nuevo a partir de Oracle Database 11g, el cual ha sido una reingeniería sobre los tipos de datos LOB. Su uso da mejor performance, reducción de espacio y seguridad sobre los LOB convencionales

A partir de Oracle Database 11g el uso de LOB sin SecureFile se le conoce como BasicFile LOB.

SecureFile LOB solo pude ser creado cuando el segmento de tipo SecureFile LOB se creará en un tablespace de tipo ASSM y el parámetro DB_SECUREFILE no tiene el valor de NEVER e IGNORE.

Crearemos una tabla con un campo BLOB en formato SecureFile:

```
SQL> create table TABLA_SECUREFILE(id number,valor BLOB)
     LOB(valor) STORE AS SECUREFILE (TABLESPACE EXAMPLE);

Table created.
```

SecureFile LOB tiene ciertas propiedades en ventaja de un LOB almacenado como BasicFile, los cuales son: Deduplicación, Compresión y Encriptación.

Deduplicación:

Cada valor en formato SecureFile almacena un hash index y si usamos la opción deduplicación validará que si un valor hash ya se encuentra registrado el valor no es insertado y se creará un puntero al valor ya ingresado previamente de esta manera mejora el uso del espacio en disco.

Para habilitarlo utilizamos la propiedad DEDUPLICATE, ejemplo:

```
SQL> create table TABLA_SECUREFILE(id number,valor BLOB)
     LOB(valor) STORE AS SECUREFILE (TABLESPACE EXAMPLE DEDUPLICATE);

Table created.
```

Compresión:

Sus opciones son:

- COMPRESS HIGH, provee la mejor compresión pero incurre en consumo de CPU.
- COMPRESS MEDIUM, es el valor por default.
- NOCOMPRESS, deshabilita la compresión.

Ejemplo de su implementación:

```
SQL> create table TABLA_SECUREFILE(id number,valor BLOB)
     LOB(valor) STORE AS SECUREFILE (DEDUPLICATE COMPRESS HIGH);

Table created.
```

Encriptación:

La encriptación se realiza a nivel de bloque de sistema operativo y se pueden usar los siguientes algoritmos (3DES168,AES128,AES192(default),AES256).

Ejemplo de su implementación:

Crearemos el wallet, el cual es la llave secreta que permite encriptar y desencriptar un valor encriptado.

a) Crearemos una carpeta para almacenar el wallet.

```
[oracle@pcoracle ~]$ mkdir $ORACLE_HOME/wallets
```

b) En el archivo SQLNET.ORA del servidor de base de datos le agregamos la siguiente línea:

ENCRYPTION_WALLET_LOCATION=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=/u01/app/oracle/product/11.2.0/dbhome_1)))

c) Reiniciamos el listener:

```
[oracle@pcoracle ~]$ lsnrctl reload

LSNRCTL for Linux: Version 11.2.0.3.0 - Production on 07-JAN-2013 02:47:47

Copyright (c) 1991, 2011, Oracle.  All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=tcp)(HOST=)(PORT=1521))
The command completed successfully
```

d) Creamos el wallet configurándole su clave.

SQL> alter system SET KEY IDENTIFIED BY "clave";

```
SQL> alter system SET KEY IDENTIFIED BY "mipassword";

System altered.
```

Nota: Por default el comando SET KEY IDENTIFIED BY abre el wallet, si la instancia la reinciamos debemos abrir el wallet de forma manual de la siguiente manera:

Oracle Database 10g:

SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN AUTHENTICATED BY "clave";

Oracle Database 11g:

SQL> ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "clave";

Asimismo si deseamos cerrar el wallet ejecutamos el siguiente comando:

Oracle Database 10g:

SQL> ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;

Oracle Database 11gR2:

ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "<clave>";

Creación del SecureFile.

```
SQL> create table TABLA_SECURE_FILE(id number,valor BLOB)
     LOB(valor) STORE AS SECUREFILE(ENCRYPT USING 'AES192');

Table created.
```

Si la tabla es creada y no ha sido encriptado el campo podemos hacerlo posteriormente con el comando ALTER TABLE, ejemplo:

```
SQL> create table TABLA_SECURE_FILE(id number,valor CLOB)
     LOB(valor) STORE AS SECUREFILE;

Table created.

SQL> alter table TABLA_SECURE_FILE modify(valor ENCRYPT USING 'AES256');

Table altered.
```

Nota: Si un wallet no fue abierto y se trata de leer un dato encriptado conseguiremos el error ORA-28365: wallet is not open.

Desfragmentación

A partir de la versión Oracle Database 10g podemos realizar desfragmentaciones con el comando SHRINK a nuestras tablas. Si queremos ejecutar el comando SHRINK sobre un segmento LOB debemos ejecutarlo de la siguiente manera:

```
SQL>alter table <nombre_tabla> MODIFY LOB(<campo_lob>)(SHRINK SPACE);

SQL> alter table TABLA_SECURE_FILE MODIFY LOB(valor)(SHRINK SPACE);

Table altered.
```

Nota: Recordemos que no podemos ejecutar una operación de SHRINK en segmentos que están comprimidos.

Si deseamos desfragmentar el campo LOB a partir de una operación move, ejecutamos lo siguiente:

```
SQL> alter table <nombre_tabla> move LOB(<campo_lob>) STORE AS (tablespace <tablespace>);

SQL> alter table TABLA_SECURE_FILE move LOB(valor) STORE AS (tablespace EXAMPLE);

Table altered.
```

Conclusión

Oracle Database 8i inició una gran mejora sobre los tipos de datos que nos permitían trabajar hasta ese momento con texto largo y documentos multimedia (LONG/RAW/LONG RAW) creando el tipo de dato LOB en pro de eliminar una serie de problemas y limitantes que tenían estos tipos de datos previos.

La versión Oracle Database 11g ha traído una mejora considerable sobre el tipo de dato LOB llamado SecureFile. Es recomendable que podamos migrar a los tipos de datos SecureFile en caso aún estemos trabajando con los tipos de datos antiguos. Oracle Database nos entrega una serie de opciones para realizar la migración sin problemas.

Publicado por Ing. Francisco Riccio. Es un IT Specialist en IBM Perú e instructor de cursos oficiales de certificación Oracle. Está reconocido por Oracle como un Oracle ACE y certificado en productos de Oracle Application & Base de Datos.

e-mail: francisco@friccio.com

web: www.friccio.com