14

Managing LOBs: Database Administration

You must perform various administrative tasks to set up, maintain, and use a database that contains LOBs.

> **Note:**
>
> LOBs are not supported when the Container Database root and Pluggable Databases are in different character sets. For more information, refer to Relocating a PDB Using CREATE PLUGGABLE DATABASE.

- Initialization Parameter for SecureFiles LOBs
  As a database administrator, you can configure the conditions that control or allow creation of SecureFiles LOBs or BasicFiles LOBs. Typically, you set up the `DB_SECUREFILE` parameter in the `init.ora` file for this purpose.
- Database Character Set Considerations
  The database character set cannot be changed from a single-byte to a multibyte character set if there are populated user-defined `CLOB` columns in the database tables.
- Database Utilities for Loading Data into LOBs
  Certain utilities are recommended for bulk loading data into LOB columns as part of the database set up or maintenance tasks.
- LOB Migration with Data Pump
- BFILEs Management
  This section describes various administrative tasks to manage databases that contain `BFILE`s.
- Managing LOB Signatures
  This section describes how to configure LOB signatures.

14.4 LOB Migration with Data Pump

See Migrating LOBs with Data Pump.

14.1 Initialization Parameter for SecureFiles LOBs

As a database administrator, you can configure the conditions that control or allow creation of SecureFiles LOBs or BasicFiles LOBs. Typically, you set up the `DB_SECUREFILE` parameter in the `init.ora` file for this purpose.

The `DB_SECUREFILE` initialization parameter is dynamic and can be modified with the `ALTER SYSTEM` statement in the following way:

```
ALTER SYSTEM SET DB_SECUREFILE = 'ALWAYS';
```

The valid values for this parameter are described in the following table:

Value	Description
NEVER	Prevents SecureFiles LOBs from being created. If NEVER is specified, then any LOBs that are specified as SecureFiles LOBs are created as BasicFiles LOBs. If storage options are not specified, then the BasicFiles LOB defaults are used. All SecureFiles LOB-specific storage options and features such as compress, encrypt, and deduplicate throw an exception.
IGNORE	Always create BasicFile LOBs, and ignore any errors that the SecureFile LOB options might cause. If IGNORE is specified, then the SECUREFILE keyword and all SecureFiles LOB options are ignored.
PERMITTED	Allows SecureFiles LOBs to be created, if specified by users. Otherwise, BasicFiles LOBs are created.
PREFERRED(default)	Attempts to create a SecureFiles LOB unless BasicFiles LOB is explicitly specified for the LOB or the parent LOB (if the LOB is in a partition or sub-partition).
ALWAYS	Attempts to create SecureFiles LOBs, but creates any LOBs not in ASSM tablespaces as BasicFiles LOBs, unless the SECUREFILE parameter is explicitly specified. Any BasicFiles LOB storage options specified are ignored, and the SecureFiles LOB defaults are used for all storage options not specified.
FORCE	Attempts to create all LOBs as SecureFiles LOBs even if users specify BASICFILE. This option is not recommended. Instead, PREFERRED or ALWAYS should be used.

14.2 Database Character Set Considerations

The database character set cannot be changed from a single-byte to a multibyte character set if there are populated user-defined CLOB columns in the database tables.

The national character set cannot be changed between AL16UTF16 and UTF8 if there are populated user-defined NCLOB columns in the database tables.

> **See Also:**
>
> Choosing a Character Set

14.3 Database Utilities for Loading Data into LOBs

Certain utilities are recommended for bulk loading data into LOB columns as part of the database set up or maintenance tasks.

The following utilities are recommended for bulk loading data into LOB columns as part of database setup or maintenance tasks:

- SQL*Loader
- External Tables
- Oracle Data Pump

- Loading LOBs with SQL*Loader
  Learn about conventional and direct-path loads, when Oracle recommends that you use direct-path loads, and what rules and guidelines you should follow to avoid issues.
- Loading BFILEs with SQL*Loader
  This section describes how to load data from files in the file system into a `BFILE` column using SQL*Loader.
- Loading LOBs with External Tables
  External tables are particularly useful for loading large numbers of records from a single file, so that each record appears in its own row in the table.

14.3.1 Loading LOBs with SQL*Loader

Learn about conventional and direct-path loads, when Oracle recommends that you use direct-path loads, and what rules and guidelines you should follow to avoid issues.

There are two options for loading large object (LOB) data:

A **conventional path load** executes SQL `INSERT` statements to populate tables in an Oracle Database.

A **direct-path load** eliminates much of the Oracle Database overhead by formatting Oracle data blocks, and writing the data blocks directly to the database files. Additionally, a direct-path load does not compete with other users for database resources, so it can usually load data at near disk speed. Be aware that there are also other restrictions, security, and backup implications for direct path loads, which you should review.

For each of these options of loading large object data (LOBs), you can use the following techniques to load data into LOBs:

- Loading LOB data from primary data files.

  When you load data from a primary data file, the data for the LOB column is part of the record in the file that you are loading.
- Loading LOB data from a secondary data file using LOB files.

  When you load data from a secondary data file, the data for a LOB column is in a different file from the primary data file. Instead of the data itself, the primary data file contains information about the location of the content of the LOB data in other files.

Recommendations for Using SQL*Loader to Load LOBs

Oracle recommends that you keep the following guidelines and rules in mind when loading LOBs using SQL*Loader:

- Tables that you want to load must already exist in the database. SQL*Loader never creates tables. It loads existing tables that either contain data, or are empty.
- When you load data from LOB files, specify the maximum length of the field corresponding to a LOB-type column. If the maximum length is specified, then SQL*Loader uses this length as a hint to help optimize memory usage. You should ensure that the maximum length you specify does not underestimate the true maximum length.

- If you use conventional path loads, then be aware that failure to load a particular LOB does not result in the rejection of the record containing that LOB; instead, the record ends up containing an empty LOB.
- If you use direct-path loads, then be aware that loading LOBs can take up substantial memory. If the message `SQL*Loader 700 (out of memory)` appears when loading LOBs, then internal code is probably batching up more rows in each load call than can be supported by your operating system and process memory. One way to work around this problem is to use the `ROWS` option to read a smaller number of rows in each data save.

  Only use direct path loads to load XML documents that are known to be valid into XMLtype columns that are stored as CLOBS. Direct path load does not validate the format of XML documents as the are loaded as CLOBs.

  With direct-path loads, errors can be critical. In direct-path loads, the LOB could be **empty** or **truncated**. LOBs are sent in pieces to the server for loading. If there is an error, then the LOB piece with the error is discarded and the rest of that LOB is not loaded. As a result, if the entire LOB with the error is contained in the first piece, then that LOB column is either empty or truncated.

  You can also use the Direct Path API to load LOBs.

Privileges Required for Using SQL*Loader to Load LOBs

The following privileges are required for using SQL*Loader to load LOBs:

- You must have `INSERT` privileges on the table that you want to load.
- You must have `DELETE` privileges on the table that you want to load, if you want to use the `REPLACE` or `TRUNCATE` option to empty out the old data before loading the new data in its place.

Example 14-1 Loading LOB from a primary data file using Delimited Fields

Review this example to see how to load LOB data in delimited fields. Note the callouts "1" and "2" in **bold**:

Control File Contents

```
LOAD DATA
INFILE 'sample.dat' "str '|'"
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name        CHAR(25),
1  "RESUME"     CHAR(507) ENCLOSED BY '<startlob>' AND '<endlob>')
```

Data File (`sample.dat`)

```
Julia Nayer,<startlob>        Julia Nayer
                          500 Example Parkway
                          jnayer@example.com ...   <endlob>
2  |Bruce Ernst, .......
```

> **Note:**
>
> The callouts, in bold, to the left of the example correspond to the following notes:
>
> 1. `<startlob>` and `<endlob>` are the enclosure strings. With the default byte-length semantics, the maximum length for a LOB that can be read using `CHAR(507)` is 507 bytes. If character-length semantics were used, then the maximum would be 507 characters. For more information, refer to character-length semantics.
> 2. If the record separator `'|'` had been placed right after `<endlob>` and followed with the newline character, then the newline would have been interpreted as part of the next record. An alternative would be to make the newline part of the record separator (for example, `'|\n'` or, in hexadecimal notation, `X'7C0A'`).

Example 14-2 Loading a LOB from secondary data file, using Delimited Fields:

In this example, note the callout "1" in **bold**:

Control File Contents

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name     CHAR(20),
1  "RESUME"    LOBFILE( CONSTANT 'jqresume') CHAR(2000)
               TERMINATED BY "<endlob>\n")
```

Data File (sample.dat)

```
Johny Quest,
Speed Racer,
```

Secondary Data File (jqresume.txt)

```
    Johny Quest
500 Oracle Parkway
   ... <endlob>
    Speed Racer
400 Oracle Parkway
   ... <endlob>
```

> **Note:**
>
> The callout, in bold, to the left of the example corresponds to the following note:
>
> 1. Because a maximum length of 2000 is specified for `CHAR`, SQL*Loader knows what to expect as the maximum length of the field, which can result in memory usage optimization. *If you choose to specify a maximum length, then you should be sure not to underestimate its value.* The `TERMINATED BY` clause specifies the string that terminates the LOBs. Alternatively, you can use the `ENCLOSED BY` clause. The `ENCLOSED BY` clause allows a bit more flexibility with the relative positioning of the LOBs in the `LOBFILE`, because the LOBs in the `LOBFILE` do not need to be sequential.

Related Topics

- Oracle Call Interface Direct Path Load Interface
- Loading Objects, LOBs, and Collections with SQL*Loader

14.3.2 Loading BFILEs with SQL*Loader

This section describes how to load data from files in the file system into a `BFILE` column using SQL*Loader.

> **Note:**
>
> - The `BFILE` data type stores unstructured binary data in operating system files outside the database. A `BFILE` column or attribute stores a file locator that points to a server-side external file containing the data.
> - A particular file to be loaded as a `BFILE` does not have to actually exist at the time of loading. SQL*Loader assumes that the necessary `DIRECTORY` objects have been created.

> **See Also:**
>
> DIRECTORY Objects for more information

A control file field corresponding to a `BFILE` column consists of the column name followed by the `BFILE` directive.

The `BFILE` directive takes as arguments a `DIRECTORY` object name followed by a `BFILE` name. Both of these can be provided as string constants, or they can be dynamically sourced through some other field.

> **See Also:**
>
> *Oracle Database Utilities* for details on SQL*Loader syntax

The following two examples illustrate the loading of `BFILE`s.

> **Note:**
>
> You need to set up the following data structures for certain examples to work:
>
> ```
> CONNECT pm/pm
> CREATE OR REPLACE DIRECTORY adgraphic_photo as '/tmp';
> CREATE OR REPLACE DIRECTORY adgraphic_dir as '/tmp';
> ```

In the following example, only the file name is specified dynamically. The directory name, `adgraphic_photo`, is in quotation marks. Therefore, the string is used as is, and is not capitalized.

Control file:

```
LOAD DATA
INFILE sample9.dat
INTO TABLE Print_media
FIELDS TERMINATED BY ','
(product_id  INTEGER EXTERNAL(6),
 FileName    FILLER CHAR(30),
 ad_graphic  BFILE(CONSTANT "adgraphic_photo", FileName))
```

Data file:

```
007, modem_2268.jpg,
008, monitor_3060.jpg,
009, keyboard_2056.jpg,
```

In the following example, the `BFILE` and the `DIRECTORY` objects are specified dynamically.

Control file:

```
LOAD DATA
INFILE sample10.dat
INTO TABLE Print_media
FIELDS TERMINATED BY ','
(
 product_id INTEGER EXTERNAL(6),
 ad_graphic BFILE (DirName, FileName),
 FileName  FILLER CHAR(30),
 DirName   FILLER CHAR(30)
)
```

Data file:

```
007,monitor_3060.jpg,ADGRAPHIC_PHOTO,
008,modem_2268.jpg,ADGRAPHIC_PHOTO,
009,keyboard_2056.jpg,ADGRAPHIC_DIR,
```

14.3.3 Loading LOBs with External Tables

External tables are particularly useful for loading large numbers of records from a single file, so that each record appears in its own row in the table.

> **Note:**
>
> Loading LOBs with External Tables

- Overview of LOBs and External Tables
  Learn the benefits of using external tables with your database to read and write data, and to understand how to create them.

14.3.3.1 Overview of LOBs and External Tables

Learn the benefits of using external tables with your database to read and write data, and to understand how to create them.

External tables enable you to treat the contents of external files as if they are rows in a table in your Oracle Database. After you create an external table, you can then use SQL statements to read rows from the external table, and insert them into another table.

To perform these operations, Oracle Database uses one of the following access drivers:

- The `ORACLE_LOADER` access driver reads text files and other file formats, similar to SQL Loader.
- The `ORACLE_DATAPUMP` access driver creates binary files that store data returned by a query. It also returns rows from files in binary format.

When you create an external table, you specify column and data types for the external table. The access driver has a list of columns in the data file, and maps the contents of the field in the data file to the column with the same name in the external table. The access driver takes care of finding the fields in the data source, and converting these fields to the appropriate data type for the corresponding column in the external table. After you create an external table, you can load the target table by using an `INSERT AS SELECT` statement.

One of the advantages of using external tables to load data over SQL Loader is that external tables can load data in parallel. The easiest way to do this is to specify the `PARALLEL` clause as part of `CREATE TABLE` for both the external table and the target table.

Example 14-3

This example creates a table, `CANDIDATE`, that can be loaded by an external table. When it is loaded, it then creates an external table, `CANDIDATE_XT`. Next, it executes an `INSERT` statement to load the table. The `INSERT` statement includes the `+APPEND` hint, which uses direct load to insert the rows into the table `CANDIDATES`. The `PARALLEL` parameter tells SQL that the tables can be accessed in parallel.

The `PARALLEL` parameter setting specifies that there can be four (`4`) parallel query processes reading from `CANDIDATE_XT`, and four parallel processes inserting into `CANDIDATE`. Note that LOBs that are stored as `BASICFILE` cannot be loaded in parallel. You can only load `SECUREFILE LOBS` in parallel. The variable *`additional-external-table-info`* indicates where additional external table information can be inserted.

```
CREATE TABLE CANDIDATES

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),
```

```
   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  ) PARALLEL 4;


CREATE TABLE CANDIDATE_XT

  (candidate_id       NUMBER,

   first_name         VARCHAR2(15),

   last_name          VARCHAR2(20),

   resume             CLOB,

   picture            BLOB

  ) PARALLEL 4;

ORGANIZATION EXTERNAL additional-external-table-info PARALLEL 4;

INSERT /*+APPEND*/ INTO CANDIDATE SELECT * FROM CANDIDATE_XT;
```

File Locations for External Tables Created By Access Drivers

All files created or read by `ORACLE_LOADER` and `ORACLE_DATAPUMP` reside in directories pointed to by directory objects. Either the DBA or a user with the `CREATE DIRECTORY` privilege can create a directory object that maps a new to a path on the file system. These users can grant `READ`, `WRITE` or `EXECUTE` privileges on the created directory object to other users. A user granted `READ` privilege on a directory object can use external tables to read files from directory for the directory object. Similarly, a user with `WRITE` privilege on a directory object can use external tables to write files to the directory for the directory object.

Example 14-4 Creating Directory Object

The following example shows how to create a directory object and grant `READ` and `WRITE` access to user `HR`:

```
create directory HR_DIR as /usr/hr/files/exttab;

grant read, write on directory HR_DIR to HR;
```

> **Note:**
>
> When using external tables in an Oracle Real Application Clusters (Oracle RAC) environment, you must make sure that the directory pointed to by the directory object maps to a directory that is accessible from all nodes.

14.5 BFILEs Management

This section describes various administrative tasks to manage databases that contain BFILEs.

- Guidelines for DIRECTORY Usage
  Learn about the guidelines for efficient management of DIRECTORY objects.
- Rules for Using Directory Objects and BFILEs
  You can create a directory object or BFILE objects if these conditions are met.
- Setting Maximum Number of Open BFILEs
  Only limited number of BFILEs can be open simultaneously in each session. Learn to define this number in this section.

14.5.1 Guidelines for DIRECTORY Usage

Learn about the guidelines for efficient management of DIRECTORY objects.

The main goal of the DIRECTORY feature is to enable a simple, flexible, non-intrusive, yet secure mechanism for the DBA to manage access to large files in the server file system. But to realize this goal, it is very important that the DBA follow these guidelines when using DIRECTORY objects:

- Do not map a DIRECTORY object to a data file directory. A DIRECTORY object should not be mapped to physical directories that contain Oracle data files, control files, log files, and other system files. Tampering with these files (accidental or otherwise) could corrupt the database or the server operating system.
- Only the DBA should have system privileges. The system privileges such as CREATE ANY DIRECTORY or DROP ANY DIRECTORY(granted to the DBA initially) should be used carefully and not granted to other users indiscriminately. In most cases, only the database administrator should have these privileges.
- Use caution when granting the DIRECTORY privilege. Privileges on DIRECTORY objects should be granted to different users carefully. The same holds for the use of the WITH GRANT OPTION clause when granting privileges to users.
- Do not drop or replace DIRECTORY objects when database is in operation. If this were to happen, then operations *from all sessions* on all files associated with this DIRECTORY object fail. Further, if a DROP or REPLACE command is executed before these files could be successfully closed, then the references to these files are lost in the programs, and system resources associated with these files are not be released until the session(s) is shut down.

  The only recourse left to PL/SQL users, for example, is to either run a program block that calls DBMS_LOB.FILECLOSEALL and restart their file operations, or exit their sessions altogether. Hence, it is imperative that you use these commands with prudence, and preferably during maintenance downtimes.
- Use caution when revoking a user's privilege on DIRECTORY objects. Revoking a user's privilege on a DIRECTORY object using the REVOKE statement causes all subsequent operations on dependent files from the user's session to fail. The user must either re-acquire the privileges to close the file, or run a FILECLOSEALL in the session and restart the file operations.

In general, using DIRECTORY objects for managing file access is an extension of system administration work at the operating system level. With some planning, files can be logically organized into suitable directories that have READ privileges for the Oracle process.

DIRECTORY objects can be created with READ privileges that map to these physical directories, and specific database users granted access to these directories.

> **See Also:**
>
> Security on Directory Objects

14.5.2 Rules for Using Directory Objects and BFILEs

You can create a directory object or BFILE objects if these conditions are met.

When you create a directory object or BFILE objects, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
- The operating system directory path named in the Oracle DIRECTORY object must be an existing operating system directory path.
- The operating system directory path named in the Oracle DIRECTORY object should not contain any symbolic links in its components.

14.5.3 Setting Maximum Number of Open BFILEs

Only limited number of BFILEs can be open simultaneously in each session. Learn to define this number in this section.

The initialization parameter, SESSION_MAX_OPEN_FILES, defines an upper limit on the number of simultaneously open files in a session.

The default value for this parameter is 10. Using this default, you can open a maximum of 10 files at the same time in each session. To alter this limit, the database administrator must change the parameter value in the init.ora file**.** For example:

```
SESSION_MAX_OPEN_FILES=20
```

If the number of unclosed files reaches the SESSION_MAX_OPEN_FILES value, then you cannot open additional files in the session. To close all open files, use the DBMS_LOB.FILECLOSEALL call.

> **See Also:**
>
> DIRECTORY Objects

14.6 Managing LOB Signatures

This section describes how to configure LOB signatures.

You can configure signature-based security for large object (LOB) locators using the LOB_SIGNATURE_ENABLE initialization parameter.

- To enable signature, set the `LOB_SIGNATURE_ENABLE` initialization parameter at `init.ora`, or using the following `ALTER SYSTEM` command. Also ensure that you have set the compatibility to 12.2.0.2 or above.

  ```
  ALTER SYSTEM SET LOB_SIGNATURE_ENABLE = [TRUE|FALSE];
  ```

- The following `ALTER` statement helps to encrypt, re-key, and delete the signature keys.

  ```
  ALTER DATABASE DICTIONARY [ENCRYPT|REKEY|DELETE] CREDENTIALS;
  ```

For more information, refer to the Oracle Database Security Guide.

> **See Also:**
>
> *Oracle Database Security Guide*