1

Introduction to Large Objects and SecureFiles

Large Objects are used to hold large amounts of data inside Oracle Database, SecureFiles provides performance comparable to file system performance, and DBFS provides file system interface to files stored in Oracle Database.

- What Are Large Objects?
  Large Objects (LOBs), SecureFiles LOBs, and Database File System (DBFS) work together with various database features to support application development.
- Where Should We Use LOBs?
  Large objects are suitable for semistructured and unstructured data.
- LOB Classifications
  LOBs store a variety of data such as audio, video, documents, and so on. Based on the type of data stored in the LOB or memory management mechanism used, there are different classifications.
- LOB Locator and LOB Value
  A LOB instance has a locator and a value. A LOB locator is a reference, or a *pointer*, to where the LOB value is physically stored. The LOB value is the data stored in the LOB.
- LOB Restrictions
  You have to keep a few restrictions in mind while working with LOB data.
- How to Navigate This Book
  This section elaborates how to navigate this book using a flow chart that provides information about the relevant chapters you must read for understanding various concepts or performing various tasks.

1.1 What Are Large Objects?

Large Objects (LOBs), SecureFiles LOBs, and Database File System (DBFS) work together with various database features to support application development.

Large Objects

The maximum size for a single LOB can range from 8 terabytes to 128 terabytes depending on how your database is configured. Storing data in LOBs enables you to access and manipulate the data efficiently in your application.

SecureFile LOBs

SecureFile LOBs are LOBs that are created in a tablespace managed with Automatic Segment Space Management (ASSM). SecureFiles is the default storage mechanism for LOBs in database tables. Oracle strongly recommends SecureFiles for storing and managing LOBs.

Database File System (DBFS)

Database File System (DBFS) provides a file system interface to files that are stored in an Oracle Database.

Files stored in an Oracle Database are usually stored as SecureFiles LOBs, and path names, directories, and other file system information is stored in the database tables. SecureFiles

LOBs is the default storage method for DBFS, but BasicFiles LOBs can be used in some situations.

With DBFS, you can make references from SecureFiles LOB locators to files stored outside the database. These references are called DBFS Links or Database File System Links.

1.2 Where Should We Use LOBs?

Large objects are suitable for semistructured and unstructured data.

Large object features enable you to store the following types of data in the database and also in the operating system files that are accessed from the database.

- Semistructured data

  Semistructured data has a logical structure that is not typically interpreted by the database, for example, an XML document that your application or an external service processes. Oracle Database provides features such as Oracle XML DB, Oracle Multimedia, and Oracle Spatial and Graph to help your application work with semistructured data.
- Unstructured data

  Unstructured data is easily not broken down into smaller logical structures and is not typically interpreted by the database or your application, such as a photographic image stored as a binary file.

Data unsuited for LOBs

- Simple Structured Data
  Simple structured data can be organized into relational tables that are structured based on business rules.
- Complex Structured Data
  Complex structured data is suited for the object-relational features of the Oracle Database such as collections, references, and user-defined types.

Maximum Size of a LOB

The maximum permissible LOB size for your configuration depends on the block size setting of the tablespace. It is calculated as `(4 gigabytes - 1)*(space usable for data in the LOB block)`. For example, if a LOB is stored in a tablespace of block size 8K, then the approximate maximum LOB size is about 32 terabytes.

1.3 LOB Classifications

LOBs store a variety of data such as audio, video, documents, and so on. Based on the type of data stored in the LOB or memory management mechanism used, there are different classifications.

- Large Object Data Types
  Oracle Database provides a set of large object data types as SQL data types, where the term *LOB* generally refers to the set.
- Types of LOBs
  This section describes the three types of LOB data that Oracle supports.
- LOBs in Object Data Types
  Typically, there is no difference in the use of a LOB instance in a LOB column or in an object data type, as its member.

- Oracle Data Types Stored in LOBs
  Many data types provided with Oracle Database are stored as or created with LOB types.

1.3.1 Large Object Data Types

Oracle Database provides a set of large object data types as SQL data types, where the term *LOB* generally refers to the set.

In general, the descriptions given for the data types in this table and related sections, also apply to the corresponding data types provided for other programmatic environments.

The following table describes each large object data type that the database supports and describes the kind of data that uses it.

Table 1-1 Types of Large Object Data

SQL Data Type	Description
`BLOB`	Binary Large Object Stores any kinds of data in binary format. Used for images, audio, and video.
`CLOB`	Character Large Object Stores string data in the database character set format. Used for large strings or documents that use the database character set exclusively. Characters in the database character set are in a fixed width format.
`NCLOB`	National Character Set Large Object Stores string data in National Character Set format, typically large strings or documents. Supports characters of varying width format.
`BFILE`	External Binary File A binary file stored outside of the database in the host operating system file system, but accessible from database tables. `BFILE`s can be accessed from your application on a **read-only** basis. Use `BFILE`s to store static data, such as image data, that is not manipulated in applications. Any kind of data, that is, any operating system file, can be stored in a `BFILE`. For example, you can store character data in a `BFILE` and then load the `BFILE` data into a `CLOB`, specifying the character set upon loading.

1.3.2 Types of LOBs

This section describes the three types of LOB data that Oracle supports.

Persistent LOBs

A persistent LOB is a LOB instance that exists in a table row in the database. Persistent LOBs participate in database transactions. You can recover persistent LOBs in the event of transaction or media failure, and any changes to a persistent LOB value can be committed or rolled back. In other words, all the Atomicity, Consistency, Isolation, and Durability (ACID) properties that apply to database objects apply to persistent LOBs. Persistent LOBs can be of data types `BLOB`, `CLOB` and `NCLOB`.

Temporary LOBs

A temporary LOB instance is created when you instantiate a LOB only within the scope of your local application. Temporary LOBs are transient, just like other local variables in an application. A temporary LOB becomes persistent when you insert it into a table row. Temporary LOBs can be of data types `BLOB`, `CLOB` and `NCLOB`.

A Value LOB is a special kind of read-only temporary LOB with optimizations for better performance and manageability compared to a reference LOB. Many applications use LOBs to store medium-sized objects, about a few mega-bytes in size, and just want to read the LOB value in the context of a SQL query. Oracle recommends that you use Value LOBs for applications which use LOBs as a larger `VARCHAR` or `RAW` data type.

BFILEs

`BFILE`s are data objects stored in operating system files, outside the database tablespaces. Data stored in a table column of type `BFILE` is physically located in an operating system file, not in the database.

`BFILE`s are read-only data types. The database allows read-only byte stream access to data stored in `BFILE`s. You cannot write to or update a `BFILE` from within your application.

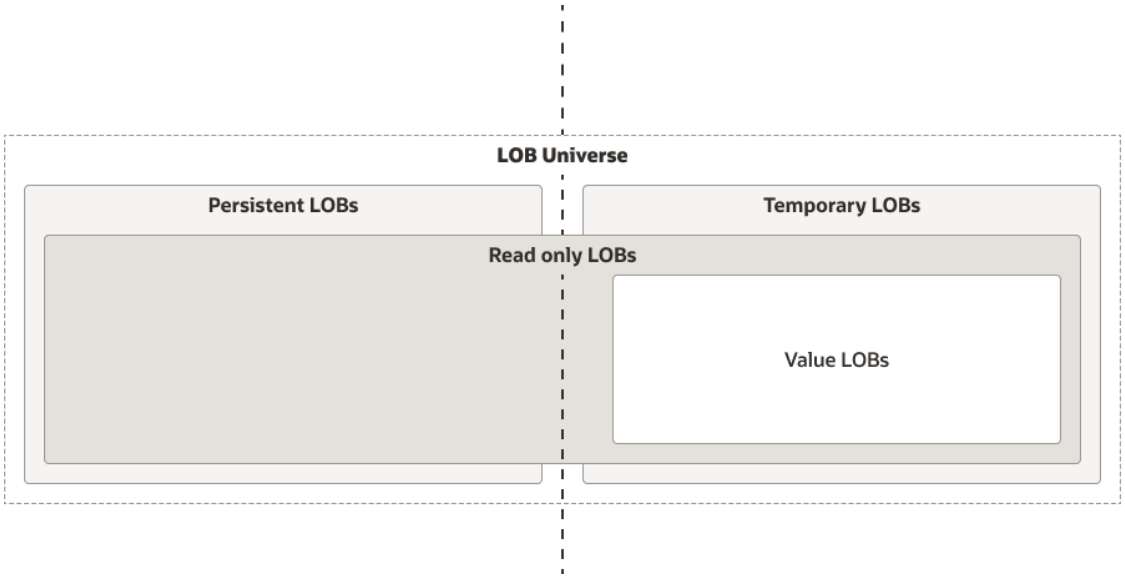
You typically use `BFILE`s to hold:

- Binary data that does not change while your application is running, such as graphics
- Data that is loaded into other large object types, such as a `BLOB` or `CLOB`, where the data can then be manipulated
- Data that is appropriate for byte-stream access, such as multimedia

Any storage device accessed by your operating system can hold `BFILE` data, including hard disk drives, CD-ROMs, PhotoCDs, and DVDs. The database can access `BFILE`s provided the operating system supports stream-mode access to the operating system files.

> **Note:**
>
> All the information related to BFILEs is exclusively documented either in BFILEs or in Managing LOBs: Database Administration.

The following picture summarizes the relationship between different kinds of LOBs.



1.3.3 LOBs in Object Data Types

Typically, there is no difference in the use of a LOB instance in a LOB column or in an object data type, as its member.

In this guide, the term *LOB attribute* refers to a LOB instance that is a member of an object data type. Unless otherwise specified, discussions that apply to LOB columns also apply to LOB attributes.

1.3.4 Oracle Data Types Stored in LOBs

Many data types provided with Oracle Database are stored as or created with LOB types.

The following list mentions a few data types that you can store with LOB types:

- `VARCHAR2` or `RAW` data types of size greater than 4000 bytes
- `JSON` data type
- `XMLType` stored as `BINARY XML` or `CLOB`
- `VARRAY` stored as LOB

1.4 LOB Locator and LOB Value

A LOB instance has a locator and a value. A LOB locator is a reference, or a *pointer*, to where the LOB value is physically stored. The LOB value is the data stored in the LOB.

A LOB locator can be assigned to any LOB instance of the same type, such as `BLOB`, `CLOB`, `NCLOB`, or `BFILE`. When you use a LOB in an operation such as passing a LOB as a parameter, you are actually passing a LOB locator. For the most part, you can work with a LOB instance in your application without being concerned with the semantics of LOB locators. There is no requirement to dereference LOB locators, as is required with pointers in some programming languages.

There are two different techniques to access and modify LOBs:

- Using LOBs Without Locators
  LOBs can be used in many operations similar to how VARCHAR2 or RAW data types are used. Such LOB operations can be performed without the use of LOB locators.
- Using LOBs with Locators
  You can use the LOB locator to access and modify LOB values by passing the LOB locator to the LOB APIs supplied with the database. These operations support efficient piecewise read and write to LOBs.

1.4.1 Using LOBs Without Locators

LOBs can be used in many operations similar to how VARCHAR2 or RAW data types are used. Such LOB operations can be performed without the use of LOB locators.

LOB operations that are similar to `VARCHAR2` and `RAW` types include:

- SQL and PLSQL built-in functions and implicit assignments

> **See Also:**
>
> – SQL Semantics for LOBs
> – PL/SQL Semantics for LOBs

- Data interface on LOBs that enables you to insert or select entire LOB data in a LOB column without using a LOB locator as follows:
  – Use a bind variable associated with a LOB column to insert character data into a `CLOB`, or `RAW` data into a `BLOB`. For example, in PLSQL you can insert a `VARCHAR2` buffer into a `CLOB` column, and in OCI you can bind a buffer of type `SQLT_CHAR` to a `CLOB` column.
  – Define an output buffer in your application that holds character data selected from a `CLOB` or `RAW` data selected from a `BLOB`. For example, in PLSQL you can select the `CLOB` output of a query into a `VARCHAR2` buffer, and in OCI you can define a `CLOB` query result item to a buffer of type `SQLT_CHAR`.

> **See Also:**
>
> Data Interface for LOBs

1.4.2 Using LOBs with Locators

You can use the LOB locator to access and modify LOB values by passing the LOB locator to the LOB APIs supplied with the database. These operations support efficient piecewise read and write to LOBs.

You should use this mode if your application needs to perform random or piecewise read or write calls to LOBs, which means it needs to specify the offset or amount of the operation to read or write a part of the LOB value.

> **See Also:**
>
> Locator Interface for LOBs

1.5 LOB Restrictions

You have to keep a few restrictions in mind while working with LOB data.

LOB columns are subject to the following rules and restrictions:

- You cannot specify a LOB as a primary key column.
- You cannot specify LOB columns in the `ORDER BY` clause of a query, the `GROUP BY` clause of a query, or an aggregate function.
- You cannot specify a LOB column in a `SELECT... DISTINCT` or `SELECT... UNIQUE` statement or in a join. However, you can specify a LOB attribute of an object type column in a `SELECT... DISTINCT` statement, a query that uses the `UNION`, or a `MINUS` set operator if the object type of the column has a `MAP` or `ORDER` function defined on it.

- Clusters cannot contain LOBs, either as key or nonkey columns.
- Even though compressed `VARRAY` data types are supported, they are less performant.
- The following data structures are supported only as temporary instances. You cannot store these instances in database tables:
  - `VARRAY` of any LOB type
  - `VARRAY` of any type containing a LOB type, such as an object type with a LOB attribute
  - `ANYDATA` of any LOB type
  - `ANYDATA` of any type containing a LOB
- The first (`INITIAL`) extent of a LOB segment must contain at least three database blocks.
- The minimum extent size is 14 blocks. For an 8K block size (the default), this is equivalent to 112K.
- When creating an `AFTER UPDATE` DML trigger, you cannot specify a LOB column in the `UPDATE OF` clause. For a table on which you have defined an `AFTER UPDATE DML` trigger, if you use OCI functions or the `DBMS_LOB` package to change the value of a LOB column or the LOB attribute of an object type column, the database does not fire the DML trigger.
- You cannot specify a LOB column as part of an index key. However, you can specify a LOB column in the indextype specification of a functional or domain index. In addition, Oracle Text lets you define an index on a `CLOB` column.
- In SQL Loader, a field read from a LOB cannot be used as an argument to a clause.
- Case-insensitive searches on `CLOB` columns often do not succeed. If you perform the following case-insensitive search on a `CLOB` column:

```
ALTER SESSION SET NLS_COMP=LINGUISTIC;
ALTER SESSION SET NLS_SORT=BINARY_CI;
SELECT * FROM ci_test WHERE LOWER(clob_col) LIKE 'aa%';
```

  The select fails without the `LOWER` function. You can perform case-insensitive searches with Oracle Text or the `DBMS_LOB.INSTR()` function.

See Also:

- Restrictions on SQL Operations on LOBs
- Guidelines and Restrictions for Implicit Conversions with LOBs
- Restrictions for Data Interface on Remote LOBs
- Restrictions when using remote LOB locators
- Restrictions on Mounted File Systems
- Restrictions on Types of Files Stored at DBFS Mount Points
- Restrictions on Index Organized Tables with LOB Columns
- Restrictions on Migrating LOBs with Data Pump

1.6 How to Navigate This Book

This section elaborates how to navigate this book using a flow chart that provides information about the relevant chapters you must read for understanding various concepts or performing various tasks.