4 Value LOBs

Value LOBs, are a subset of Temporary LOBs, that are autonomous, read-only and more performant.

- About Value LOBs
  Use Persistent and Temporary LOBs, henceforth referred to as Reference LOBs, for applications which require reads and writes on the LOB.
- When to Use Value LOBs
  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.
- Creating a Value LOB
  A value LOB is a read-only temporary LOB that is generated by a SQL statement and is auto-freed at the next SQL fetch. Use value LOBs only for scenarios where a LOB fetched from SQL is only read before the next fetch is performed.
- Value LOBs in Queries
  Whether a query fetches a Value or a Reference LOB is a compile-time decision, and can be obtained by using a describe on the query.
- Performing DML Operations on LOBs with QUERY AS VALUE
  The QUERY AS VALUE property is applicable only for SQL queries, so it does not affect any DML on the table. Therefore any DMLs such as `INSERT` or `UPDATE` work identically for LOB columns declared as QUERY AS VALUE or QUERY AS REFERNCE.
- Value LOB APIs in Different Programmatic Interfaces
  This section lists the value LOB specific APIs in different programmatic interfaces.
- Restrictions on Value LOBs
  Keep the following restrictions in mind while working with value LOBs.

4.1 About Value LOBs

Use Persistent and Temporary LOBs, henceforth referred to as Reference LOBs, for applications which require reads and writes on the 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.

Value LOBs have the following characteristics:

- A Value LOB is a special kind of read-only temporary LOB with optimizations for better performance and manageability compared to a reference LOB.
- All LOB read APIs are supported on value LOBs, but none of the write APIs are supported. LOB read APIs allow the data to be read piecewise and support random access by allowing the user to specify the amount and the offset of the operation.
- A value LOB gets automatically freed when the next fetch for a cursor is performed. In the case of temporary LOB, it has always been the user's responsibility to free the temporary LOB when their application is done processing it.

Several SQL operators such as `substr`, `to_clob()` create temporary LOBs, but it is the responsibility of the user to free these temporary LOBs. If you don't free the temporary LOBs, it can considerably slow down your system. With Value LOBs, a user can use LOBs similar to the `VARCHAR32k` data type without the responsibility to free them. Oracle Database automatically frees the LOBs as soon as the next set of rows are fetched in SQL. Since the concept of fetch duration exists only for SQL, value LOBs exist only in the context of a SQL query.

- A value LOB can be arbitrary sized, similar to a reference LOB.

  As all LOB read APIs are supported on value LOBs, there is no limit on the LOB size. However, value LOBs are best suited for documents of size up to a few megabytes, which can be prefetched to the client.
- A value LOB, in most instances, has faster performance than a reference LOB. Oracle highly recommends that you use value LOBs if your application fetches a LOB for read purposes as part of a SQL query and consumes the LOB data before the next fetch is performed on the cursor.
- PL/SQL doesn't have the concept of value LOBs.

  Value LOBs exist in the SQL query, as well as on the client side programmatic interfaces, such as JDBC, OCI, and ODP.NET. In PL/SQL, a temporary LOB is automatically freed when a user overwrites the variable containing that LOB. Hence any value LOBs passed from SQL to PL/SQL are converted to read-only temporary LOBs, and have the duration of the variable that holds the LOB. For more information, see PL/SQL APIs for Value LOBs.

4.2 When to Use Value LOBs

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. All the LOB reads are performed before the next set of SQL rows are fetched. This implies that if you want to access the value LOB content after fetching the next set of SQL rows, then you must read and save the value LOB content on the client side. Oracle recommends that you use Persistent and Temporary LOBs in the following scenarios:

- for applications which require reads and writes to the LOB
- where the LOB is expected to last for a longer duration
- when you can't fully read and cache the LOB content at the client side, but rely on the LOB locator to read LOB data from server

Caution:

Before transforming a reference LOB column to a Value LOB column, ensure that your business use case requires the usage of Value LOBs.

4.3 Creating a Value LOB

A value LOB is a read-only temporary LOB that is generated by a SQL statement and is auto-freed at the next SQL fetch. Use value LOBs only for scenarios where a LOB fetched from SQL is only read before the next fetch is performed.

A value LOB is always a temporary LOB irrespective of its origin. When the next fetch is performed on the cursor,Oracle server automatically frees the value LOBs from the previous fetch. So the value LOBs from the previous fetch won't be accessible. A value LOB can be seen as disposable LOB: once it is read, it is not needed anymore, and it is freed. This prevents accumulation of temporary LOBs, which translates to better performance and scalability of the query.

You can create a value LOB in a SQL query in the following ways:

- Creating Value LOBs Using DDL
- Creating Value LOBs Using SQL Operators
- Creating Value LOB in Views
- Creating Value LOBs Using LOB_VALUE Operator

Most of the examples in the following sections use the `agents` table. Following is the structure of the `agents` table.

Column Name	Column Type
`ID`	`NUMBER`
`NAME`	`VARCHAR2(100)`
`VALUELOB`	`CLOB VALUE`
`REFERENCELOB`	`CLOB`
`CV`	`BLOB`
`PHOTO`	`BLOB VALUE`

- Creating Value LOBs Using DDL
  If your application is written in a way that all LOBs from a particular table follow the Value LOB use case, then use the `query as value` syntax for the LOB column as part of the `CREATE TABLE` or `ALTER TABLE` statements.
- Creating Value LOBs Using SQL Operators
  Any SQL operator that has a LOB input and a LOB output will produce a value LOB if the input is a value LOB, as shown in the following examples:
- Creating Value LOBs Using LOB_VALUE Operator
  You can convert any persistent or temporary LOB to a Value LOB by using the `LOB_VALUE` operator, as shown in the following examples.
- Creating Value LOB in Views
  LOB columns with QUERY AS VALUE property can be part of a view. You can create them in two ways.

4.3.1 Creating Value LOBs Using DDL

If your application is written in a way that all LOBs from a particular table follow the Value LOB use case, then use the `query as value` syntax for the LOB column as part of the `CREATE TABLE` or `ALTER TABLE` statements.

This fetches all LOB locators from the specified column as value without changing the rest of the application. The default for create tables is `query as reference` which fetches a LOB locator as a Reference LOB which can be persistent or temporary.

Example

The following example creates a table with the name agent with columns of various LOB types, including `valuelob` and `photo` which are value LOBs:

```
create table agents (id number, name varchar2(100), valuelob clob,
referencelob clob, cv blob)
lob(valuelob) query as value;
alter table agents add (photo blob) lob(photo) query as value;
```

The following example shows how you can switch the query property of a LOB column between `value` or `reference` using the `ALTER TABLE MODIFY LOB` clause.

```
alter table agents modify lob(cv) query as value;
```

Note:

The `query as value` or `query as reference` property only impacts how a LOB is selected out in a query. It does not change how the LOB is physically stored in the table.

When describing a table, a columns defined with `query as value` will contain the `VALUE` keyword with the data type.

```
SQL> desc agents;
Name                     Null?      Type
---------------------- --------  ----------------------------
ID                                 NUMBER
NAME                               VARCHAR2(100)
VALUELOB                           CLOB VALUE
REFERENCELOB                       CLOB
CV                                 BLOB
PHOTO                              BLOB VALUE
```

The following example shows when you run a query on a column defined with the `query as value` property, it returns a value LOB.

```
select valuelob from agents;
```

Note:

The `query as value` property is applicable only to LOB locators, hence it does not affect the data interface on LOBs. In other words, there is no difference between how the data interface works with LOBs declared with `query as reference` and `query as value`.

4.3.2 Creating Value LOBs Using SQL Operators

Any SQL operator that has a LOB input and a LOB output will produce a value LOB if the input is a value LOB, as shown in the following examples:

```
-- Produces a value LOB
SELECT SUBSTR(valuelob, 5, 5) from agents;

-- Produces a value LOB
SELECT to_blob(valuelob, 0) from agents;

-- Produces a value LOB
SELECT CONCAT(valuelob, valuelob) from agents;
```

Any SQL operator that has a LOB input and a LOB output will produce an old fashioned temporary LOB if the input is a reference LOB, or if the input is not a LOB type, as shown in the following examples:

```
-- Produces a reference Temporary LOB
SELECT SUBSTR(referencelob, 5, 5) from agents;

-- Produces a reference Temporary LOB
SELECT to_blob(substr(referencelob,5,5), 0) from agents;

-- Produces a reference Temporary LOB as name is varchar type
SELECT to_clob(name) from agents;
```

If a SQL operator takes in 2 LOBs, then it is an error if one of them is a value lob and the other is not, as shown in the following example:

```
-- Raises an error
SELECT CONCAT(valuelob, referencelob) from agents;
```

4.3.3 Creating Value LOBs Using LOB_VALUE Operator

You can convert any persistent or temporary LOB to a Value LOB by using the `LOB_VALUE` operator, as shown in the following examples.

```
-- Produces a value LOB
SELECT lob_value(referencelob) from agents;

-- Produces a value LOB
SELECT lob_value(substr(referencelob, 2, 10)) from agents;

-- Produces a value LOB
SELECT lob_value(to_clob(name)) from agents;
```

The `LOB_VALUE()` operator is useful when a LOB column in a table cannot be set as `QUERY AS VALUE` because different queries on that column may need to select the LOB as Reference or Value. Use this operator to convert Reference LOBs to Value LOBs to take advantage of the performance benefits offered by Value LOBs.

PL/SQL functions do not produce Value LOBs. To get a Value LOB from PL/SQL functions, use the `LOB_VALUE()` operator on the output of a PL/SQL function, as shown in the following example.

```
-- Produces a value LOB
SELECT LOB_VALUE(lob_producing_plsql_function(...)) from table;
```

Note:

The reverse conversion of a Value LOB to a Reference LOB is not permitted.

4.3.4 Creating Value LOB in Views

LOB columns with QUERY AS VALUE property can be part of a view. You can create them in two ways.

If view column refers to a value LOB column, view column will be a value LOB as well. The following example shows how `valuelob_v` column is a value LOB as part of a view.

```
-- valuelob_v column is a value LOB column
create view agents_v as
   select id id_v, valuelob valuelob_v from agents;
```

If view column uses a SQL/LOB operator that returns value LOB, it will be value LOB.

```
-- valuelob_v2 column is a value LOB
create view agents_v2 as
    select id id_v2, substr(valuelob, 5, 5) valuelob_v2 from agents;
-- valuelob_v3 column is a value LOB
create view agents_v3 as
    select id id_v3, lob_value(referencelob) valuelob_v3 from agents;
-- valuelob_v4 column is a value LOB
create view agents_v4 as
    select id id_v4, lob_value(substr(referencelob, 5, 5)) valuelob_v4 from
agents;
```

The describe command of SQLPLUS shows if a LOB column for a view is value LOB:

```
SQL> desc agents_v;
Name                                      Null?    Type
----------------------------------------- --------
----------------------------
ID_V                                               NUMBER
NAME_V                                             VARCHAR2(100)
VALUELOB_V                                         CLOB VALUE

SQL> desc agents_v2;
Name                                      Null?    Type
----------------------------------------- --------
----------------------------
ID_V2                                              NUMBER
```

```
NAME_V2                                           VARCHAR2(100)
VALUELOB_V2                                       CLOB VALUE
```

4.4 Value LOBs in Queries

Whether a query fetches a Value or a Reference LOB is a compile-time decision, and can be obtained by using a describe on the query.

Wherever a SQL function for JSON returns a LOB value, it returns a reference LOB by default. However, the returning clause can specify that the LOB be value-based. For example:

```
JSON_SERIALIZE (data returning CLOB VALUE)
```

Example: Explain plan for Value LOBs

The explain plan output shows "/* LOB_BY_VALUE */" hint for value LOB in the plan.

```
SQL> explain plan for select valuelob, substr(valuelob, 5, 5) from agents;
...
-----------------------------------------------------------------------------
| Id  | Operation        | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT |         |     1 |  2002 |     2   (0)| 00:00:01 |
|   1 | TABLE ACCESS FULL| AGENTS  |     1 |  2002 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------
...
Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "VALUELOB" /*+ LOB_BY_VALUE */ [LOB,4000]
...
```

Example: Explain plan for Reference LOBs without LOB_BY_VALUE

```
SQL> explain plan for select referencelob, substr(referencelob, 5, 5) from
agents;
...
-----------------------------------------------------------------------------
| Id  | Operation         | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |         |     1 |  2002 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS FULL| AGENTS  |     1 |  2002 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------
...
Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "REFERENCELOB"[LOB,4000]
...
```

In certain situations, the Oracle server automatically determines that a LOB should be selected as a Value LOB.

Example: Explain plan showing automatic conversion to Value LOB

```
SQL> explain plan for select length(referencelob) from agents;
...
----------------------------------------------------------------------------
| Id  | Operation         | Name   |  Rows | Bytes | Cost (%CPU)|  Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |        |     1 |  2002 |     2   (0)|  00:00:01 |
|   1 |  TABLE ACCESS FULL| AGENTS |     1 |  2002 |     2   (0)|  00:00:01 |
----------------------------------------------------------------------------
...
Column Projection Information (identified by operation id):
-----------------------------------------------------------
   1 - "REFERENCELOB" /*+ LOB_BY_VALUE */ [LOB,4000]
...
```

See Also:

Example 4-3.

4.5 Performing DML Operations on LOBs with QUERY AS VALUE

The QUERY AS VALUE property is applicable only for SQL queries, so it does not affect any DML on the table. Therefore any DMLs such as `INSERT` or `UPDATE` work identically for LOB columns declared as QUERY AS VALUE or QUERY AS REFERNCE.

DML RETURNING

If a user performs a DML with a RETURNING clause for the LOB locator, the returned LOB locator will be a reference LOB irrespective of the QUERY AS property of the LOB column. Hence a user can INSERT an `empty_lob()` with RETURNING clause, and use the returning locator to write into the persistent LOB, as shown in the following example:

```
-- insert with returning clause
declare
  c clob;
begin
  -- returns a reference to the persistent LOB
  insert into agents values (1, 'My Name', empty_clob(), NULL, NULL, NULL)
      returning valuelob into c;
  -- updates the persistent LOB
  dbms_lob.writeappend(c, length('I am a value LOB'), 'I am a value LOB');
end;
/
```

SELECT FOR UPDATE

Along the same lines, if the user selects out a locator with the `FOR UPDATE` clause, the intention is to update the persistent LOB stored in the LOB column. So the returned LOB locator will be a reference LOB irrespective of the QUERY AS property of the LOB column, as shown in the following example.

```
-- Select for update
declare
  c clob;
begin
  -- returns a reference to the persistent LOB
  select valuelob into c from agents where id = 1 for update;
  -- This updates the persistent LOB
  dbms_lob.writeappend(c, length('!!!'), '!!!');
end;
/
```

4.6 Value LOB APIs in Different Programmatic Interfaces

This section lists the value LOB specific APIs in different programmatic interfaces.

- Prefetching of Value LOBs
  Setting a large LOB prefetch size on the client side avoids round trips to server and leads to a drastically improved performance for value LOBs.
- Value LOB API Support
  Once a LOB variable is initialized with either a persistent or a temporary LOB locator, subsequent read operations on the LOB can be performed using APIs such as the `DBMS_LOB` package subprograms.
- PL/SQL APIs for Value LOBs
  This section describes the PL/SQL APIs used with value LOBs.
- OCI APIs for Value LOBs
  This section describes the OCI APIs used with value LOBs.
- Interoperability with Older Clients
  When you send a value LOB to a client that's at version 21c or earlier, the value LOB is converted to a read-only temporary reference LOB.

4.6.1 Prefetching of Value LOBs

Setting a large LOB prefetch size on the client side avoids round trips to server and leads to a drastically improved performance for value LOBs.

The default LOB prefetch size for value LOBs in JDBC, OCI and ODP.NET is 32k.

Note:

In OCI, setting a LOB prefetch size of 0 will be interpreted by Oracle as a prefetch size of 32k for value LOBs, and 0 for reference LOBs. Any other setting of LOB prefetch size is honored for both value and reference LOBs.

Oracle recommends setting the LOB prefetch size large enough to accommodate at least 80% of your LOB read size for value LOBs.

See Also:

Prefetching LOB Data and Length

4.6.2 Value LOB API Support

Once a LOB variable is initialized with either a persistent or a temporary LOB locator, subsequent read operations on the LOB can be performed using APIs such as the `DBMS_LOB` package subprograms.

See Also:

LOB APIs

The operations supported on value LOBs are divided into the following categories:

Table 4-1 Operations supported by value LOB APIs

Category	Value LOB Behavior
Sanity Checking	Supported
Open/Close	Open is allowed in read-only mode. If read-write is specified, an error is thrown.
Read Operations	Supported
Modify Operations	Not supported, since value LOBs are read-only.
Operations involving multiple locators	`OCILobLocatorAssign()` is not supported.
	`OCILobIsEqual()` will always return `FALSE` because `OCILobLocatorAssign` is not supported.
	`OCILobAppend()`, `OCILobCopy2()`,and `OCILobLoadFromFile2()`: These operations support a Value LOB as the source LOB. The destination LOB has to be selected `FOR UPDATE`, which converts it to a reference LOB, hence permitting the operation to go through. The destination LOB cannot be a value LOB.
Operations specific to SecureFiles	Not supported, since value LOBs are temporary LOBs and not Securefiles.
Operations specific to temporary LOBs	`CreateTemporary` cannot produce a value LOB. `FreeTemporary` and `IsTemporary` are supported but not required, since value LOBs are freed automatically.

4.6.3 PL/SQL APIs for Value LOBs

This section describes the PL/SQL APIs used with value LOBs.

Value LOBs exist in the SQL query, as well as on the client-side programmatic interfaces such as JDBC, OCI and ODP.NET. In PL/SQL, a temporary LOB is freed automatically when a user overwrites the variable containing that LOB. Hence any value LOBs passed from SQL to

PL/SQL are converted to read-only temporary LOBs, and have the duration of the variable that holds the LOB. When the variable goes out of scope, the temporary LOB is freed automatically.

In other words, there are no value LOBs in PL/SQL. All LOBs in PL/SQL are reference LOBs. If a Value LOB is sent from SQL to PL/SQL, it becomes a read-only temporary LOB. This LOB supports the APIs listed in Table 4-1. If this LOB is passed from PL/SQL to a client, such as JDBC or OCI, then it continues to be a read-only temporary LOB. This means it follows the same API support as in Table 4-1, but it will not be freed automatically when the next fetch is performed on the cursor. It will be the user's responsibility to free this LOB when they are done with it.

Example 4-1 PL/SQL API for Value LOBs

The following example shows how you can create a value LOB by using the `LOB_VALUE` operator on the output of a PL/SQL function.

```
SELECT LOB_VALUE(lob_producing_plsql_function(...)) from table;
```

OUT Binds

Since `OUT` binds are not part of a SQL query, they cannot return a value LOB. When you attempt to return a value LOB as an `OUT` bind variable, the value LOB is converted to a read-only temporary LOB. It is the responsibility of the application developer to free the LOB after using it since it is a reference LOB. Note that the `LOB_VALUE` operator does not work on `OUT` binds.

4.6.4 OCI APIs for Value LOBs

This section describes the OCI APIs used with value LOBs.

Example 4-2 Explicit Describe returning OCI_ATTR_LOB_IS_VALUE on the Table Using OCIDescribeAny

```
void explicitDescribe()
{
  /* Assume: create table lobtab (c clob) lob(c) query as value */
  ub1          isValueLob = 0;
  OCIParam     *colhd;
  OCIParam     *paramp = NULL;
  OCIParam     *lstHandle = NULL;
  text         *table = "lobtab";
  OCIDescribe *dschp = NULL;

  OCIHandleAlloc(envhp, (void **)&dschp, OCI_HTYPE_DESCRIBE, 0, NULL);
  checkerr(errhp, OCIDescribeAny(svchp, errhp, (void *)table, strlen(table),
                                 OCI_OTYPE_NAME, 0, OCI_PTYPE_TABLE, dschp));

  checkerr(errhp, OCIAttrGet((dvoid *) dschp, (ub4) OCI_HTYPE_DESCRIBE,
                  (dvoid *)&paramp, (ub4*)0, (ub4)OCI_ATTR_PARAM, errhp));

  /* Get the number of columns */
  checkerr(errhp, OCIAttrGet((void *)paramp, OCI_DTYPE_PARAM, (void
*)&noofcols,
                             (ub4 *)0, OCI_ATTR_NUM_COLS, errhp));

  /* Get the column list. */
  checkerr(errhp, OCIAttrGet(paramp, OCI_DTYPE_PARAM, &lstHandle, 0,
                             OCI_ATTR_LIST_COLUMNS, errhp));

  /* Go through the column list */
  for (int i = 1; i <= noofcols; i++)
  {
    /* Get parameter for column i */
    checkerr(errhp, OCIParamGet(lstHandle, OCI_DTYPE_PARAM,
                                (OCIError *)errhp, (void**)&colhd, (ub4)i),
                                (text *)"param get");

    isValueLob = 0;
    checkerr(errhp,OCIAttrGet(colhd, OCI_DTYPE_PARAM,
                                       &(isValueLob), 0,
                                        OCI_ATTR_LOB_IS_VALUE,
                                        (OCIError *)errhp),
                                        (text*)"attr get");

    printf("Is value lob = %d\n",isValueLob); /* Expected output: Is value
lob = 1 */
   }
   ...
}
```

Example 4-3 Implicit Describe returning OCI_ATTR_LOB_IS_VALUE on a Query's Column Handle

```
text          *select_sql = (text *)"select valuelob from agents"; /*
generates value LOB */
OCILobLocator *lob1;
Boolean        isValLob = 0;
OCIParam      *colhd;

/* Prepare select statement */
checkerr(errhp, OCIStmtPrepare(stmthp, errhp, select_sql,  /* select valuelob
from agents */
                               (ub4) strlen((char *) select_sql), ...);

/* Execute select statement */
checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, ...);

/* Implicit Describe: Get parameter for select item #1 */
checkerr(errhp, OCIParamGet(stmthp, OCI_HTYPE_STMT,
                            (OCIError *)errhp,
                            (void**)&colhd, (ub4)1),
                            (text *)"param valuelob column");

/* Check if colhd (valulob column) returns value LOB */
checkerr(errhp,OCIAttrGet(colhd, OCI_DTYPE_PARAM,
                          &(isValueLob), 0,
                          OCI_ATTR_LOB_IS_VALUE,
                          (OCIError *)errhp),
                          (text *)"attr get");
```

Example 4-4 Checking for OCI_ATTR_LOB_IS_VALUE after fetching a LOB Locator

```
void CheckValue(OCILobLocator *lob1)
{
  boolean isValLob = 0;

  /* Check if lob1 is value LOB */
  OCIAttrGet(lob1, OCI_DTYPE_LOB, &isValLob, sizeof(boolean),
              OCI_ATTR_LOB_IS_VALUE, errhp);
}
```

Example 4-5 Checking for OCI_ATTR_LOB_IS_READONLY after fetching a LOB Locator

Recall that there are no Value LOBs in PL/SQL. If a Value LOB is sent from SQL to PL/SQL, it becomes a Read-Only Temporary LOB. This LOB will follow the same API support as in Table 4-1. If this LOB is passed from PL/SQL to a client like JDBC or OCI, then it will continue to be a Read-Only Temporary LOB. The example below shows how to check the read-only property of a LOB locator by using the `OCI_ATTR_LOB_IS_READONLY` attribute.

```
void CheckReadOnly(OCILobLocator *lob1)
{
  boolean isReadOnly = 0;

  /* Check if lob1 is read-only */
  OCIAttrGet(lob1, OCI_DTYPE_LOB, &isReadOnly, sizeof(boolean),
              OCI_ATTR_LOB_IS_READONLY, errhp);
}
```

4.6.5 Interoperability with Older Clients

When you send a value LOB to a client that's at version 21c or earlier, the value LOB is converted to a read-only temporary reference LOB.

This means it will follow the same API support as in <the category table in 4.4.2>, but it will not be freed automatically when the next fetch is performed on the cursor. It will be the user's responsibility to free this LOB when they are done with it.

4.7 Restrictions on Value LOBs

Keep the following restrictions in mind while working with value LOBs.

- Value LOBs are read-only LOBs, so you are not permitted to perform modify operations on them.
- Locator assignment operations, such as `OCILobLocatorAssign()` are not supported on value LOBs.
- Value LOBs is not supported with the following operation in JDBC:
  - Client-side result cache
- Value LOBs are not supported with the following operations in OCI:
  - Scrollable cursors
  - Client-side result cache

- Value LOBs cannot be combined with reference LOBs in the same operation in a query, for example:

  ```
  select concat(valuelob, referencelob) from agents; -- Error expected
  ```