9

Manipulating Large Objects

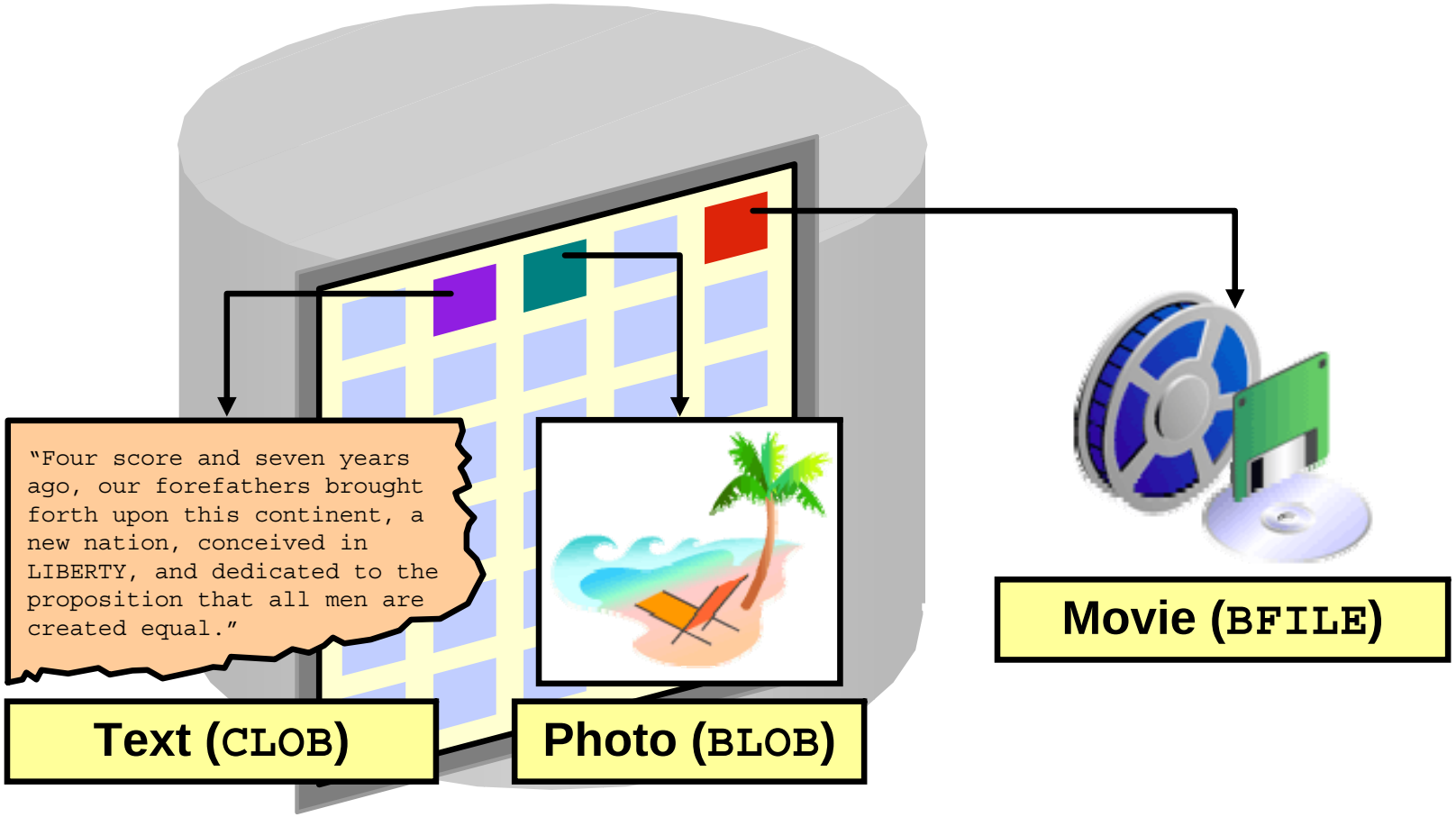
Objectives

After completing this lesson, you should be able to do the following:

- Compare and contrast LONG and LOB (large object) data types
- Create and maintain LOB data types
- Differentiate between internal and external LOBs
- Use the DBMS_LOB PL/SQL package
- Describe the use of temporary LOBs

What Is a LOB?

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

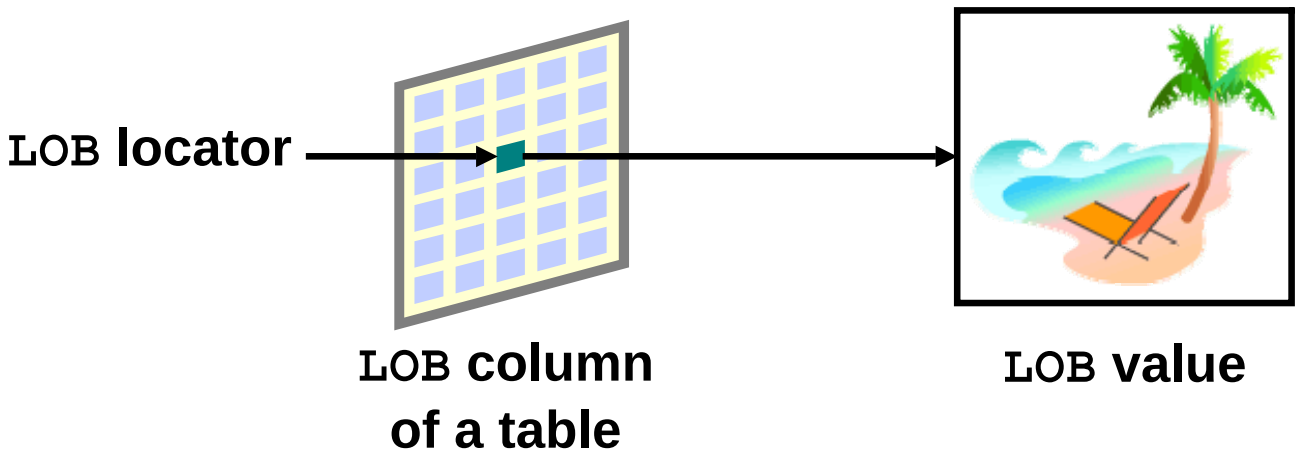


Contrasting LONG and LOB Data Types

LONG and LONG RAW	LOB
Single LONG column per table	Multiple LOB columns per table
Up to 2 GB	Up to 4 GB
SELECT returns data	SELECT returns locator
Data stored in-line	Data stored in-line or out-of-line
Sequential access to data	Random access to data

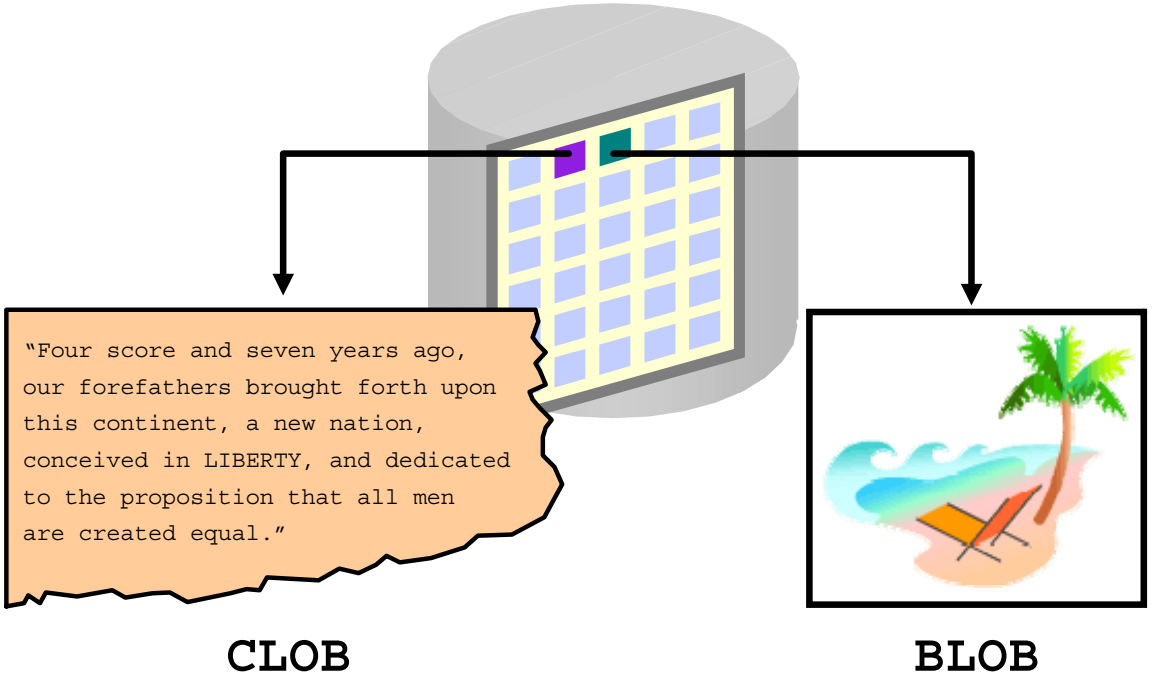
Anatomy of a LOB

The LOB column stores a locator to the LOB's value.



Internal LOBs

The LOB value is stored in the database.



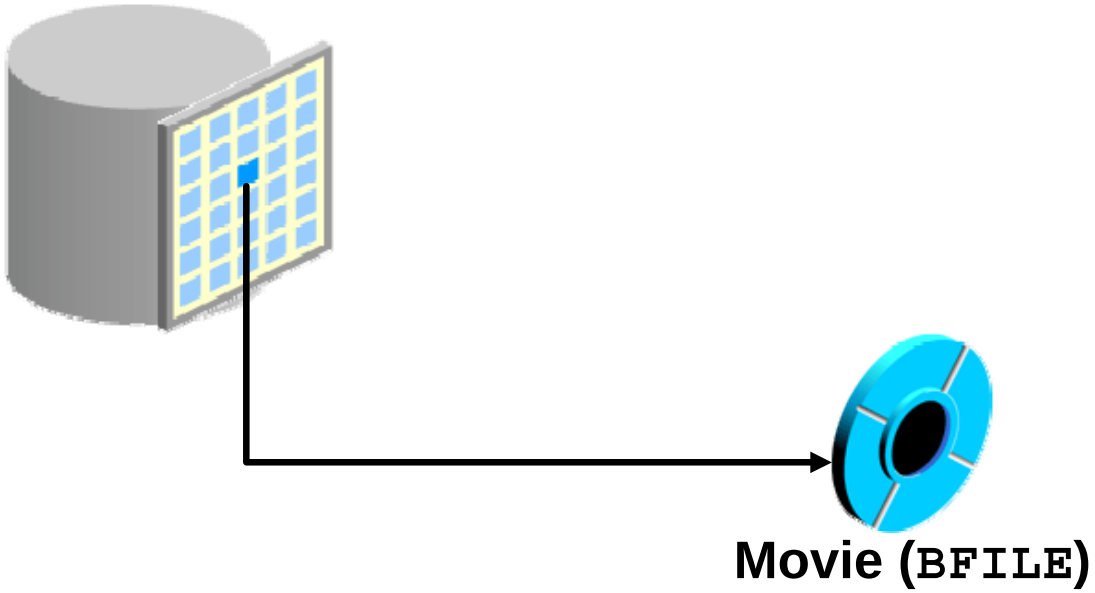
Managing Internal LOBs

- To interact fully with LOB, file-like interfaces are provided in:
  - PL/SQL package DBMS_LOB
  - Oracle Call Interface (OCI)
  - Oracle Objects for object linking and embedding (OLE)
  - Pro*C/C++ and Pro*COBOL precompilers
  - Java Database Connectivity (JDBC)
- The Oracle server provides some support for LOB management through SQL.

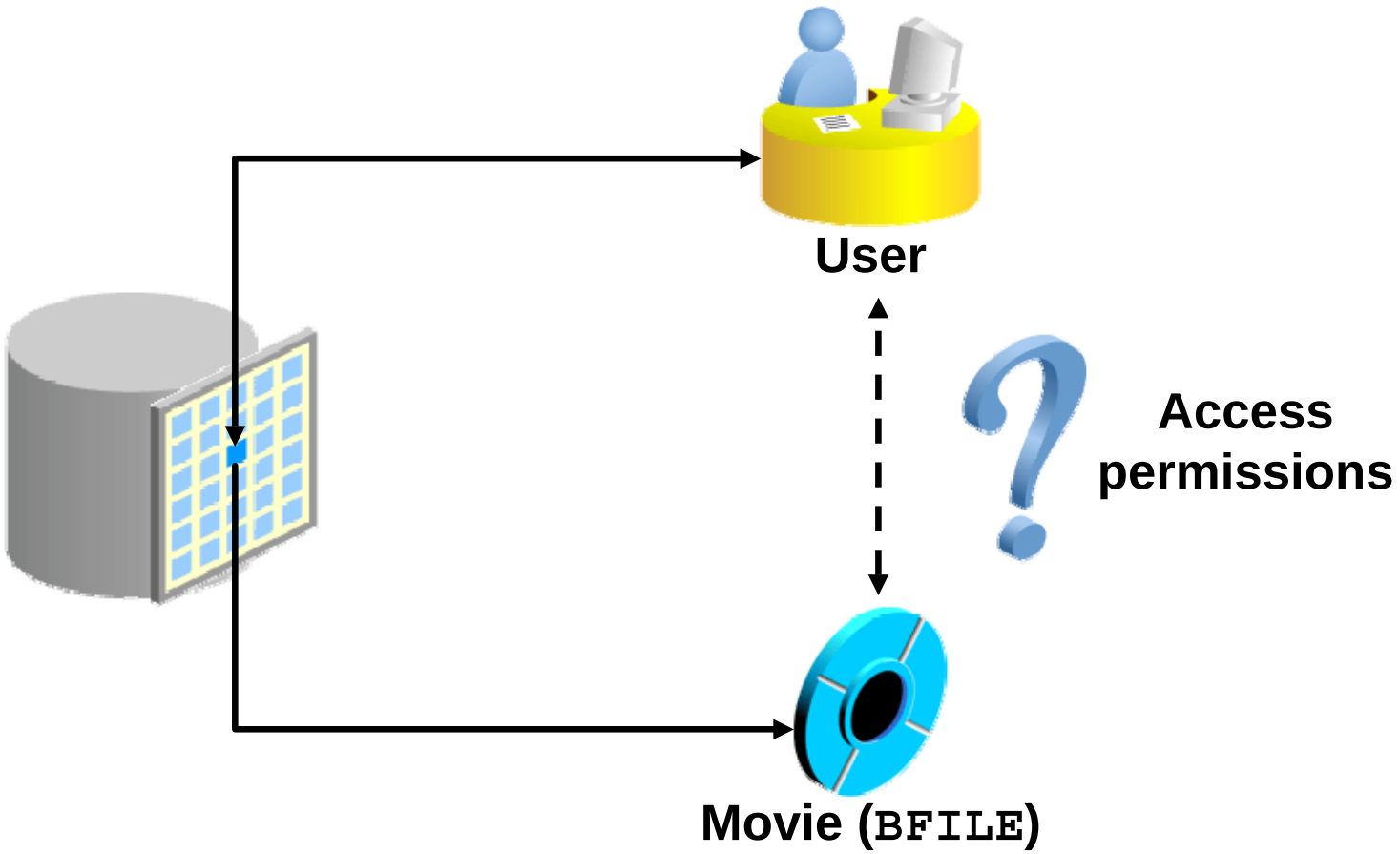
What Are BFILEs?

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

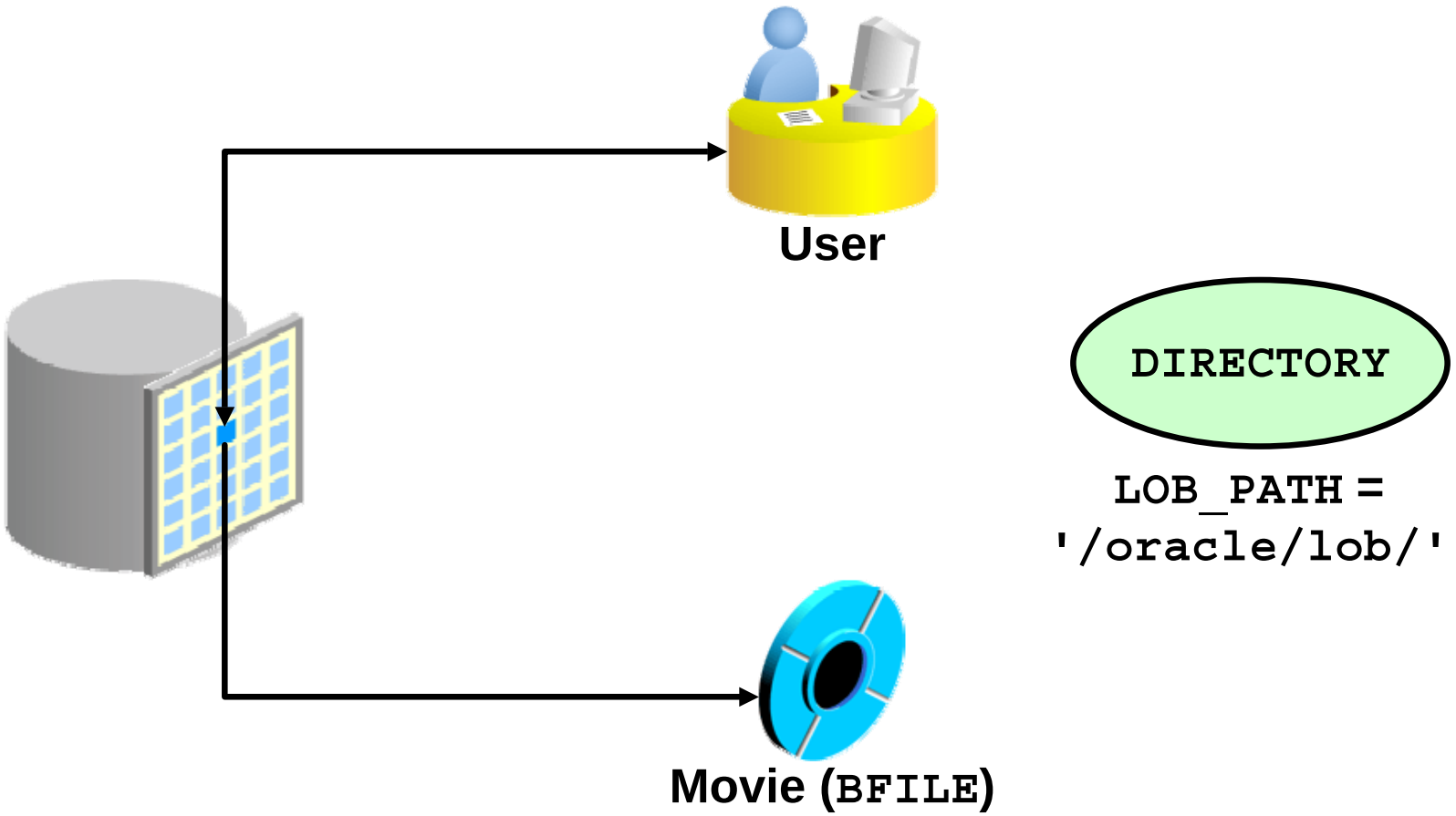
- Attributes in an object type
- Column values in a table



Securing BFILEs



A New Database Object: DIRECTORY



Guidelines for Creating DIRECTORY Objects

- Do not create DIRECTORY objects on paths with database files.
- Limit the number of people who are given the following system privileges:
  - CREATE ANY DIRECTORY
  - DROP ANY DIRECTORY
- All DIRECTORY objects are owned by SYS.
- Create directory paths and properly set permissions before using the DIRECTORY object so that the Oracle server can read the file.

Managing BFILEs

The DBA or the system administrator:

1. Creates an OS directory and supplies files
2. Creates a DIRECTORY object in the database
3. Grants the READ privilege on the DIRECTORY object to appropriate database users

The developer or the user:

4. Creates an Oracle table with a column defined as a BFILE data type
5. Inserts rows into the table using the BFILENAME function to populate the BFILE column
6. Writes a PL/SQL subprogram that declares and initializes a LOB locator, and reads BFILE

Preparing to Use BFILEs

1. Create an OS directory to store the physical data files:

```
mkdir /temp/data_files
```

2. Create a DIRECTORY object by using the CREATE DIRECTORY command:

```
CREATE DIRECTORY data_files
                AS '/temp/data_files';
```

3. Grant the READ privilege on the DIRECTORY object to appropriate users:

```
GRANT READ ON DIRECTORY data_files
   TO SCOTT, MANAGER_ROLE, PUBLIC;
```

Populating BFILE Columns with SQL

- Use the BFILENAME function to initialize a BFILE column. The function syntax is:

```
FUNCTION BFILENAME(directory_alias IN VARCHAR2,
                   filename IN VARCHAR2)
RETURN BFILE;
```

- Example:
  - Add a BFILE column to a table:

```
ALTER TABLE employees ADD video BFILE;
```

  - Update the column using the BFILENAME function:

```
UPDATE employees
  SET video = BFILENAME('DATA_FILES', 'King.avi')
WHERE employee_id = 100;
```

Populating a BFILE Column with PL/SQL

```
CREATE PROCEDURE set_video(
  dir_alias VARCHAR2, dept_id NUMBER) IS
  filename VARCHAR2(40);
  file_ptr BFILE;
  CURSOR emp_csr IS
    SELECT first_name FROM employees
    WHERE department_id = dept_id FOR UPDATE;
BEGIN
  FOR rec IN emp_csr LOOP
    filename := rec.first_name || '.gif';
    file_ptr := BFILENAME(dir_alias, filename);
    DBMS_LOB.FILEOPEN(file_ptr);
    UPDATE employees SET video = file_ptr
      WHERE CURRENT OF emp_csr;
    DBMS_OUTPUT.PUT_LINE('FILE: ' || filename ||
     ' SIZE: ' || DBMS_LOB.GETLENGTH(file_ptr));
    DBMS_LOB.FILECLOSE(file_ptr);
  END LOOP;
END set_video;
```

Using DBMS_LOB Routines with BFILEs

The DBMS_LOB.FILEEXISTS function can check whether the file exists in the OS. The function returns:

- 0 if the file does not exist
- 1 if the file does exist

```
CREATE FUNCTION get_filesize(file_ptr IN OUT BFILE)
RETURN NUMBER IS
  file_exists BOOLEAN;
  length NUMBER:= -1;
BEGIN
  file_exists := DBMS_LOB.FILEEXISTS(file_ptr)=1;
  IF file_exists THEN
    DBMS_LOB.FILEOPEN(file_ptr);
    length := DBMS_LOB.GETLENGTH(file_ptr);
    DBMS_LOB.FILECLOSE(file_ptr);
  END IF;
  RETURN length;
END;
/
```

Migrating from LONG to LOB

Oracle Database 10*g* enables the migration of LONG columns to LOB columns.

- Data migration consists of the procedure to move existing tables containing LONG columns to use LOBs:

```
ALTER TABLE [<schema>.] <table_name>
    MODIFY (<long_col_name> {CLOB | BLOB | NCLOB})
```

- Application migration consists of changing existing LONG applications for using LOBs.

Migrating from LONG to LOB

- Implicit conversion: From LONG (LONG RAW) or a VARCHAR2(RAW) variable to a CLOB (BLOB) variable, and vice versa
- Explicit conversion:
  - TO_CLOB() converts LONG, VARCHAR2, and CHAR to CLOB.
  - TO_BLOB() converts LONG RAW and RAW to BLOB.
- Function and procedure parameter passing:
  - CLOBs and BLOBs are passed as actual parameters.
  - VARCHAR2, LONG, RAW, and LONG RAW are formal parameters, and vice versa.
- LOB data is acceptable in most of the SQL and PL/SQL operators and built-in functions.

DBMS_LOB Package

- Working with LOBs often requires the use of the Oracle-supplied DBMS_LOB package.
- DBMS_LOB provides routines to access and manipulate internal and external LOBs.
- Oracle Database 10*g* enables retrieving LOB data directly using SQL without a special LOB API.
- In PL/SQL, you can define a VARCHAR2 for a CLOB and a RAW for a BLOB.

DBMS_LOB Package

- Modify LOB values:
  APPEND, COPY, ERASE, TRIM, WRITE, LOADFROMFILE
- Read or examine LOB values:
  GETLENGTH, INSTR, READ, SUBSTR
- Specific to BFILEs:
  FILECLOSE, FILECLOSEALL, FILEEXISTS,
  FILEGETNAME, FILEISOPEN, FILEOPEN

DBMS_LOB Package

- NULL parameters get NULL returns.
- Offsets:
  - BLOB, BFILE: Measured in bytes
  - CLOB, NCLOB: Measured in characters
- There are no negative values for parameters.

DBMS_LOB.READ and DBMS_LOB.WRITE

```
PROCEDURE READ (
  lobsrc IN BFILE|BLOB|CLOB ,
  amount IN OUT BINARY_INTEGER,
  offset IN INTEGER,
  buffer OUT RAW|VARCHAR2 )
```

```
PROCEDURE WRITE (
  lobdst IN OUT BLOB|CLOB,
  amount IN OUT BINARY_INTEGER,
  offset IN INTEGER := 1,
  buffer IN RAW|VARCHAR2 )  -- RAW for BLOB
```

Initializing LOB Columns Added to a Table

- Create the table with columns using the LOB type, or add the LOB columns using ALTER TABLE.

```
ALTER TABLE employees
     ADD (resume CLOB, picture BLOB);
```

- Initialize the column LOB locator value with the DEFAULT option or DML statements using the:
  - EMPTY_CLOB() function for a CLOB column
  - EMPTY_BLOB() function for a BLOB column

```
CREATE TABLE emp_hiredata (
   employee_id  NUMBER(6),
   full_name    VARCHAR2(45),
   resume       CLOB DEFAULT EMPTY_CLOB(),
   picture      BLOB DEFAULT EMPTY_BLOB());
```

Populating LOB Columns

- Insert a row into a table with LOB columns:

```
INSERT INTO emp_hiredata
  (employee_id, full_name, resume, picture)
 VALUES (405, 'Marvin Ellis', EMPTY_CLOB(), NULL);
```

- Initialize a LOB using the EMPTY_BLOB() function:

```
UPDATE emp_hiredata
 SET resume = 'Date of Birth: 8 February 1951',
    picture = EMPTY_BLOB()
 WHERE employee_id = 405;
```

- Update a CLOB column:

```
UPDATE emp_hiredata
 SET resume = 'Date of Birth: 1 June 1956'
 WHERE employee_id = 170;
```

Updating LOB by Using DBMS_LOB in PL/SQL

```
DECLARE
  lobloc CLOB;        -- serves as the LOB locator
  text   VARCHAR2(50) := 'Resigned = 5 June 2000';
  amount NUMBER ;     -- amount to be written
  offset INTEGER;     -- 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 := ' Resigned = 30 September 2000';
  SELECT resume INTO lobloc FROM emp_hiredata
  WHERE employee_id = 170 FOR UPDATE;
  amount := length(text);
  DBMS_LOB.WRITEAPPEND(lobloc, amount, text);
  COMMIT;
END;
```

Selecting CLOB Values by Using SQL

```
SELECT employee_id, full_name , resume -- CLOB
FROM emp_hiredata
WHERE employee_id IN (405, 170);
```

	EMPLOYEE_ID	FULL_NAME	RESUME
1	405	Marvin Ellis	(CLOB) Date of Birth: 8 February 1951 Resigned = 5 June 2000
2	170	Joe Fox	(CLOB) Date of Birth: 1 June 1956 Resigned = 30 September 2000

Selecting CLOB Values by Using DBMS_LOB

- DBMS_LOB.SUBSTR (lob, amount, start_pos)
- DBMS_LOB.INSTR (lob, pattern)

```
SELECT DBMS_LOB.SUBSTR (resume, 5, 18),
       DBMS_LOB.INSTR (resume,' = ')
FROM   emp_hiredata
WHERE  employee_id IN (170, 405);
```

	DBMS_LOB.SUBSTR(RESUME,5,18)	DBMS_LOB.INSTR(RESUME,'=')
1	Febru	40
2	June	36

Selecting CLOB Values in PL/SQL

```
SET LINESIZE 50 SERVEROUTPUT ON FORMAT WORD_WRAP
DECLARE
  text VARCHAR2(4001);
BEGIN
 SELECT resume INTO text
 FROM emp_hiredata
 WHERE employee_id = 170;
 DBMS_OUTPUT.PUT_LINE('text is: '|| text);
END;
/
```

```
anonymous block completed
text is: Date of Birth: 1 June 1956 Resigned = 30 September 2000
```

Removing LOBs

- Delete a row containing LOBs:

```
DELETE
FROM  emp_hiredata
WHERE employee_id = 405;
```

- Disassociate a LOB value from a row:

```
UPDATE emp_hiredata
SET resume = EMPTY_CLOB()
WHERE employee_id = 170;
```

Temporary LOBs

- Temporary LOBs:
  - Provide an interface to support creation of LOBs that act like local variables
  - Can be BLOBs, CLOBs, or NCLOBs
  - Are not associated with a specific table
  - Are created using the DBMS_LOB.CREATETEMPORARY procedure
  - Use DBMS_LOB routines
- The lifetime of a temporary LOB is a session.
- Temporary LOBs are useful for transforming data in permanent internal LOBs.

Creating a Temporary LOB

PL/SQL procedure to create and test a temporary LOB:

```
CREATE OR REPLACE PROCEDURE is_templob_open(
  lob IN OUT BLOB, retval OUT INTEGER) IS
BEGIN
  -- create a temporary LOB
  DBMS_LOB.CREATETEMPORARY (lob, TRUE);
  -- see if the LOB is open: returns 1 if open
  retval := DBMS_LOB.ISOPEN (lob);
  DBMS_OUTPUT.PUT_LINE (
    'The file returned a value...' || retval);
  -- free the temporary LOB
  DBMS_LOB.FREETEMPORARY (lob);
END;
/
```

Summary

In this lesson, you should have learned how to:

- Identify four built-in types for large objects: `BLOB`, `CLOB`, `NCLOB`, and `BFILE`
- Describe how `LOB`s replace `LONG` and `LONG RAW`
- Describe two storage options for `LOB`s:
  - Oracle server (internal `LOB`s)
  - External host files (external `LOB`s)
- Use the `DBMS_LOB` PL/SQL package to provide routines for `LOB` management
- Use temporary `LOB`s in a session

Practice 9: Overview

This practice covers the following topics:

- Creating object types using the `CLOB` and `BLOB` data types
- Creating a table with `LOB` data types as columns
- Using the `DBMS_LOB` package to populate and interact with the `LOB` data