15

Manipulating Large Objects

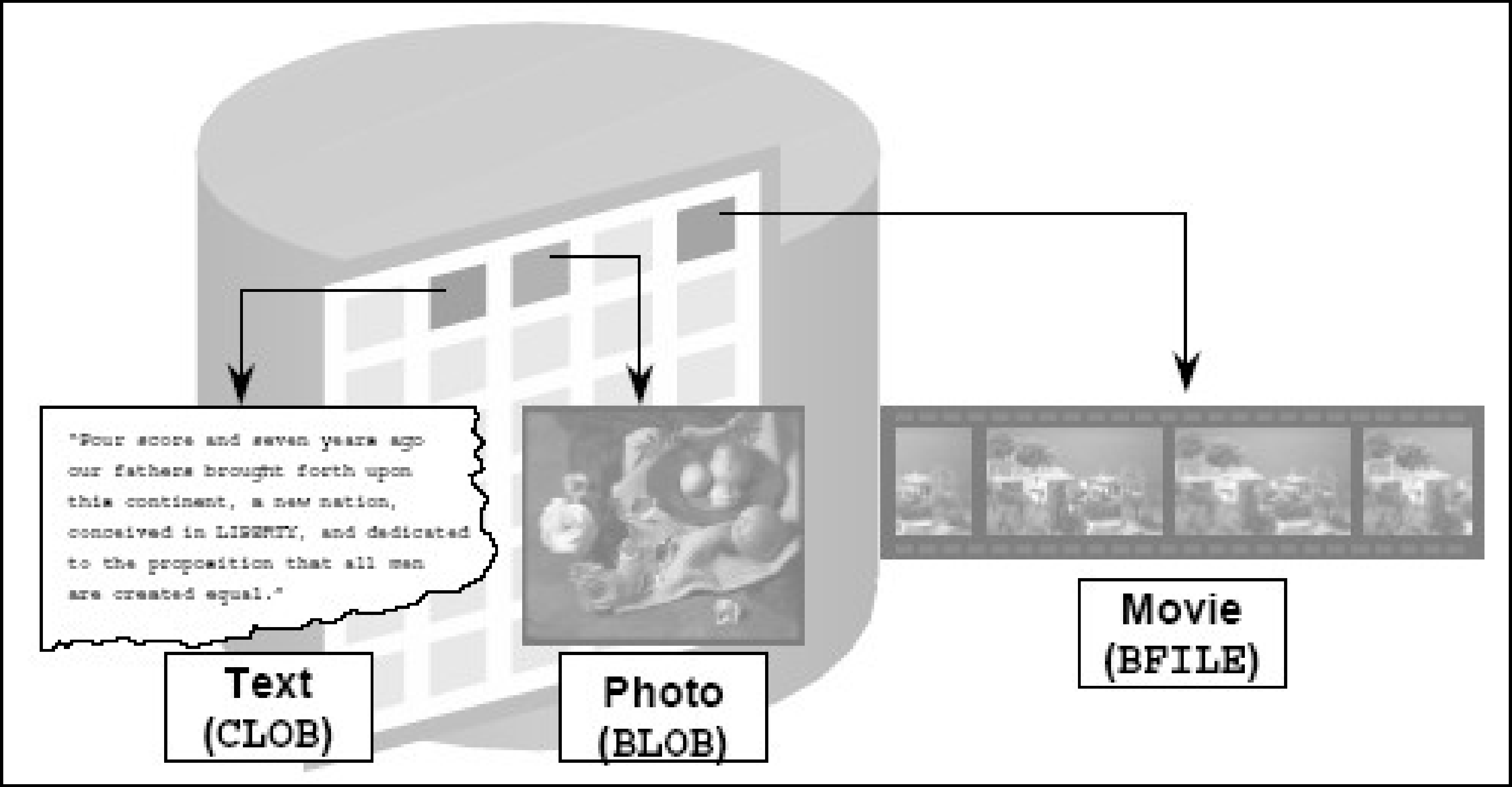
Objectives

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

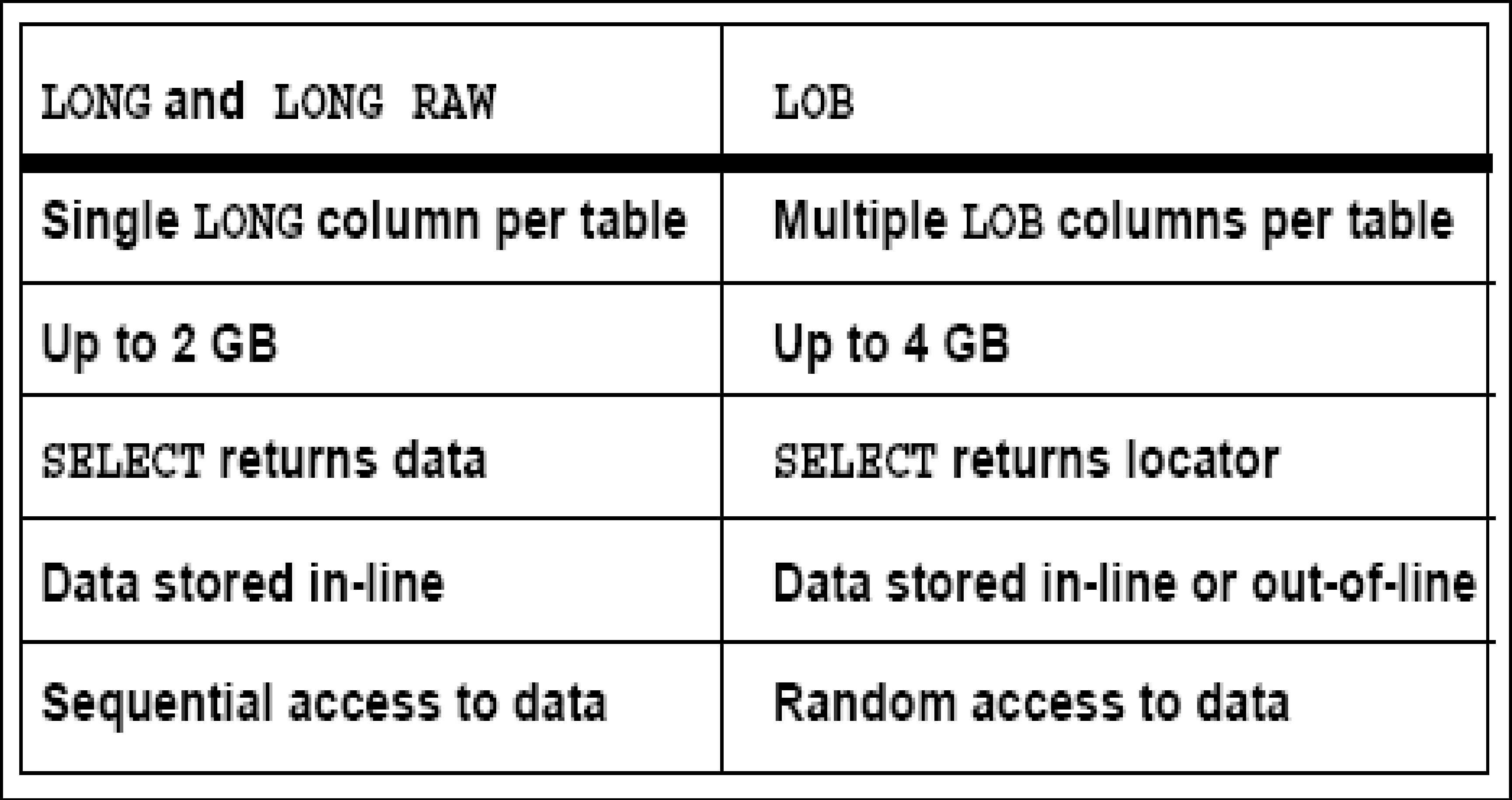
- Compare and contrast LONG and large object (LOB) data types
- Create and maintain LOB data types
- Differentiate between internal and external LOBs
- Use the DBMS_LOB PL/SQL package
- Describe the use of temporary LOBs

What Is a LOB?

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


"Four score and seven years ago
our fathers brought forth upon
this continent, a new nation,
conceived in LIBERTY, and dedicated
to the proposition that all men
are created equal."
Text
(CLOB)
Photo
(BLOB)
Movie
(BFILE)

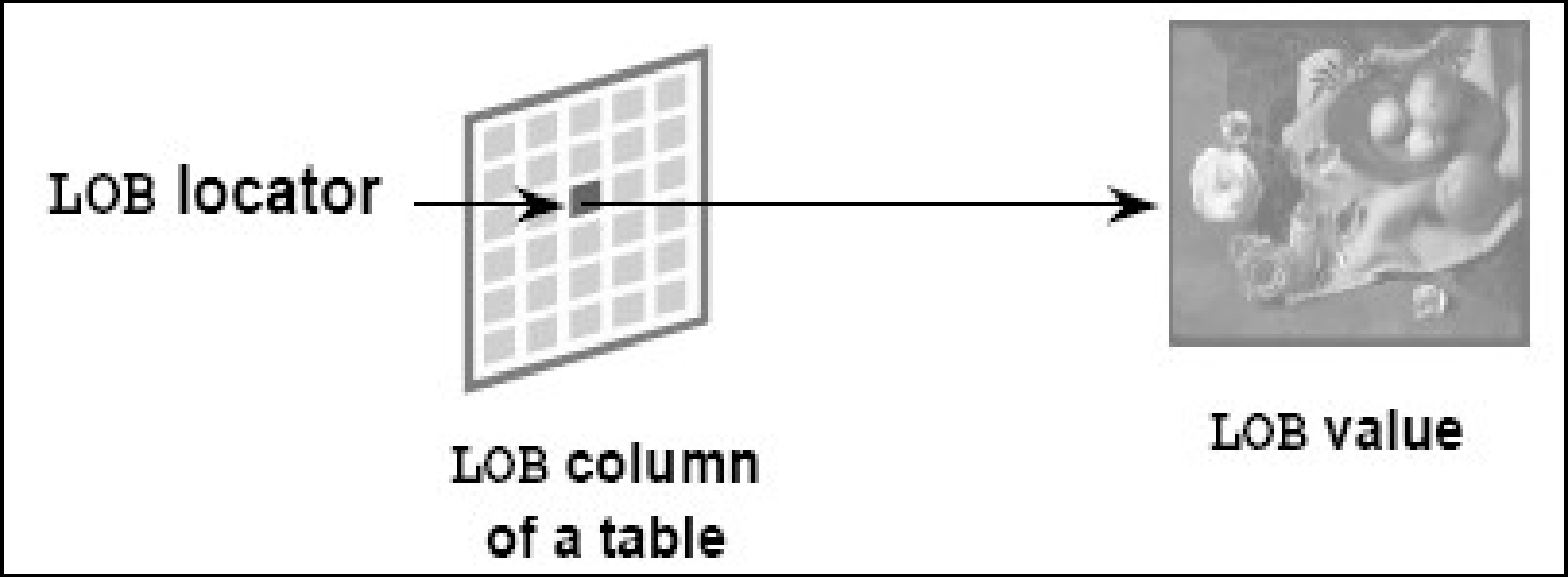
Contrasting LONG and LOB Data Types



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

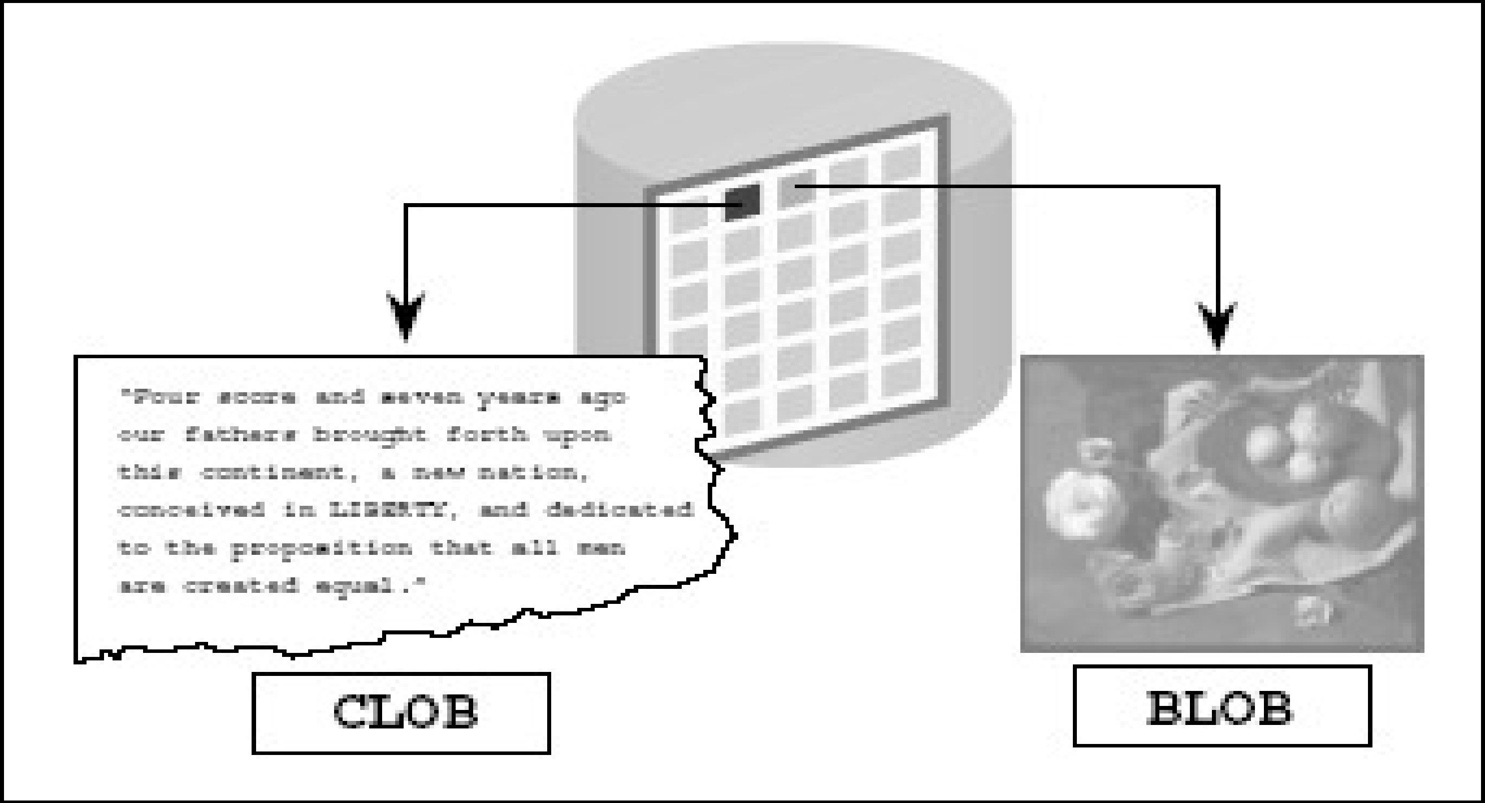
Anatomy of a LOB

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



Internal LOBs

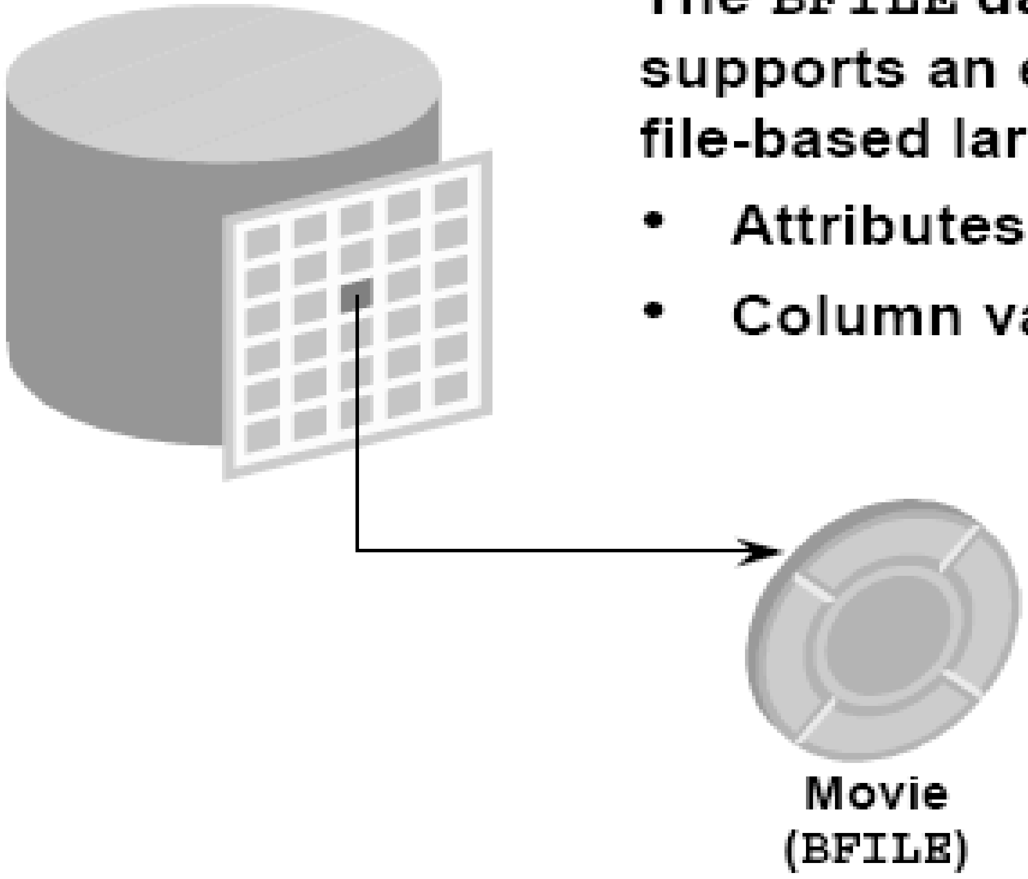
The LOB value is stored in the database.


"Four score and seven years ago
our fathers brought forth upon
this continent, a new nation,
conceived in LIBERTY, and dedicated
to the proposition that all men
are created equal."
CLOB
BLOB

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
  - JDBC
- The Oracle server provides some support for LOB management through SQL.

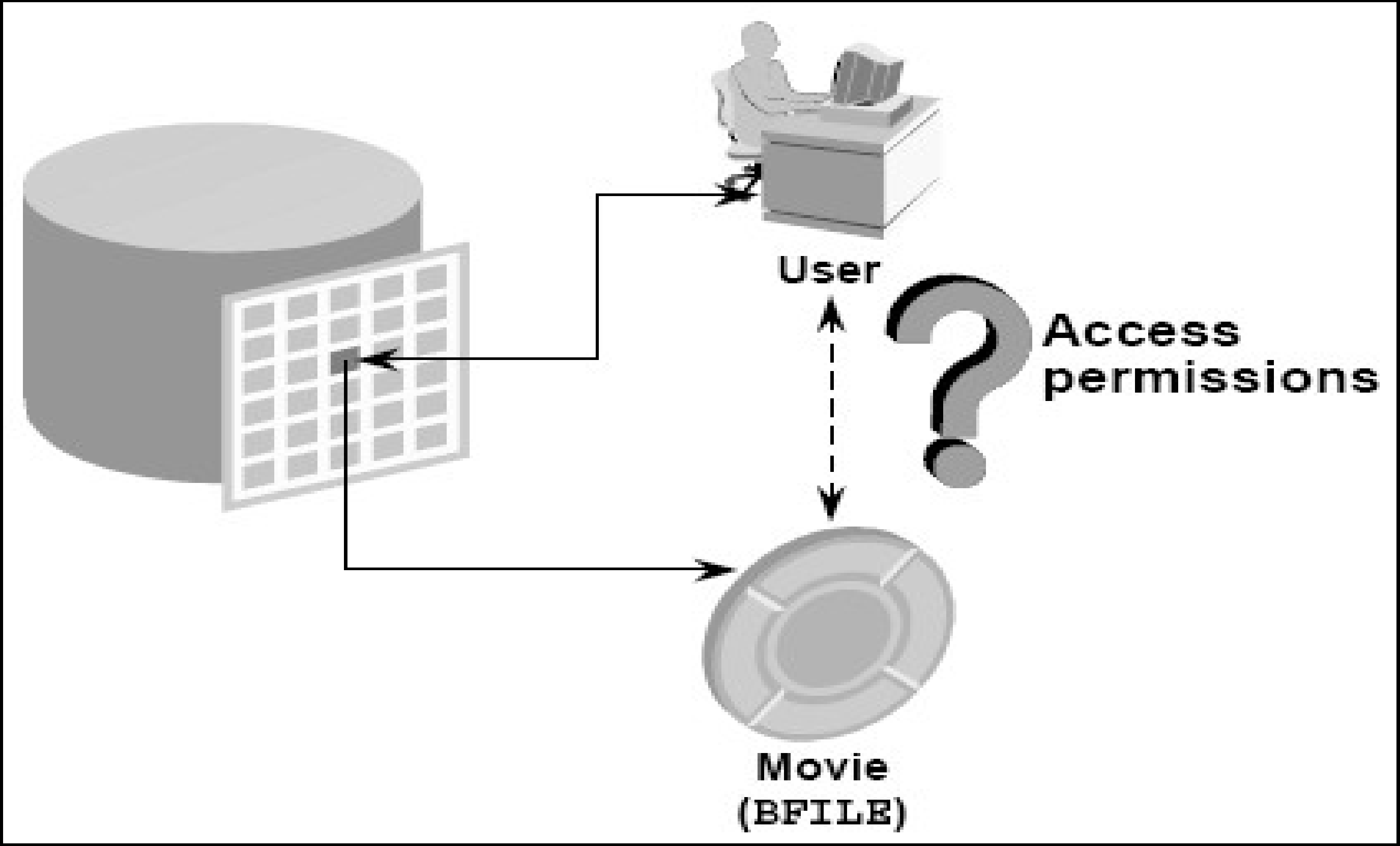
What Are BFILEs?



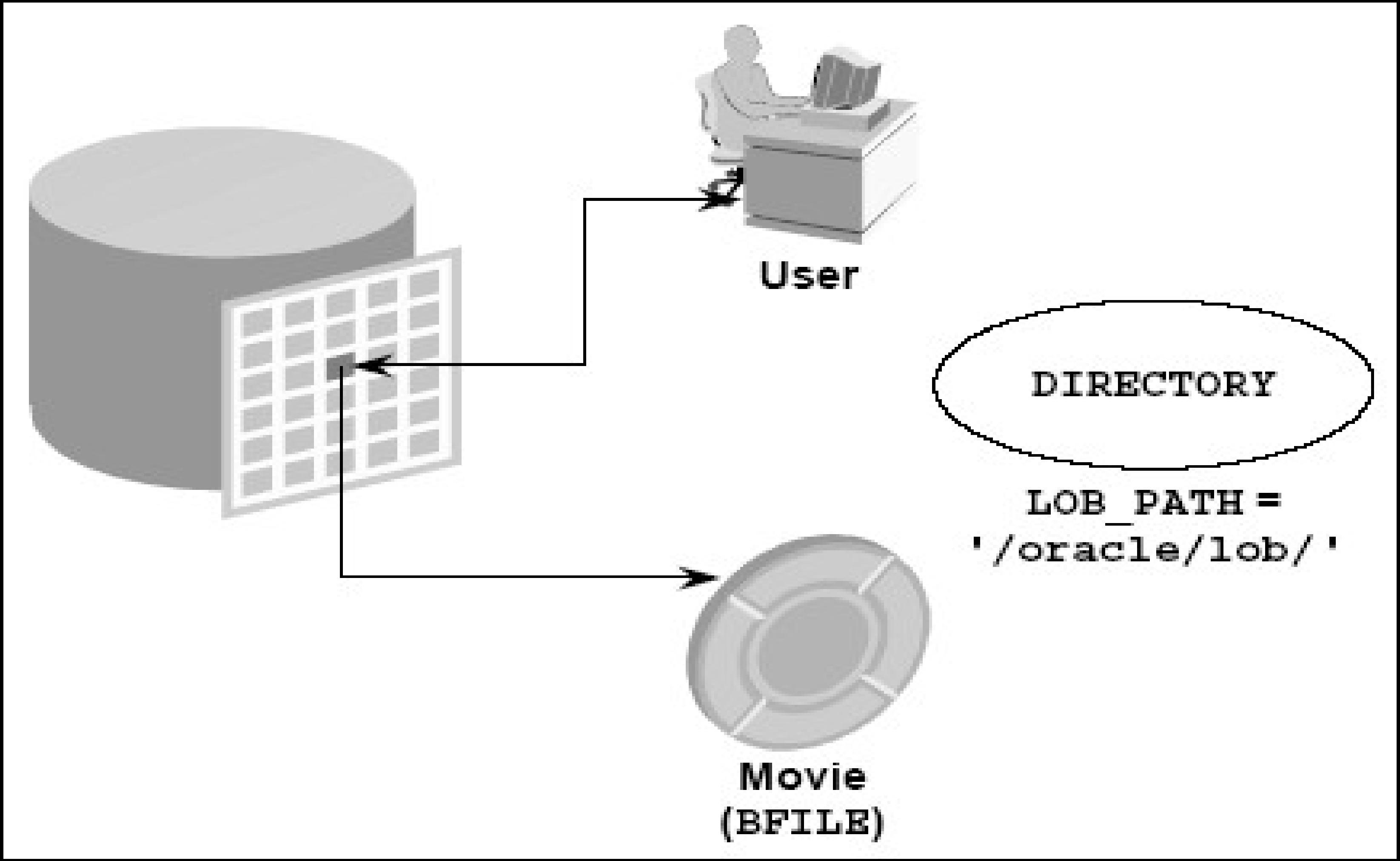
The `BFILE` data type supports an external or file-based large object as:

- 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

- Create an OS directory and supply files.
- Create an Oracle table with a column that holds the BFILE data type.
- Create a DIRECTORY object.
- Grant privileges to read the DIRECTORY object to users.
- Insert rows into the table by using the BFILENAME function.
- Declare and initialize a LOB locator in a program.

Read the BFILE.

Preparing to Use BFILEs

- Create or modify an Oracle table with a column that holds the BFILE data type.

```
ALTER TABLE employees  ADD emp_video BFILE;
```

- Create a DIRECTORY object by using the CREATE DIRECTORY command.

```
CREATE DIRECTORY dir_name  AS os_path;
SQL>conn / as sysdba;
CREATE DIRECTORY files_dir AS ‘D:\Emp_bmp\’;
```

- Grant privileges to read the DIRECTORY object to users.

```
GRANT READ ON DIRECTORY dir_name TO user|role|PUBLIC;
GRANT READ ON DIRECTORY files_dir TO HR, SCOTT;
```

The BFILENAME Function

Use the BFILENAME function to initialize a BFILE column.

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

Loading BFILEs

```
CREATE OR REPLACE PROCEDURE load_emp_bfile
            (p_file_loc IN VARCHAR2) IS
   v_file BFILE;
   v_filename VARCHAR2(16);
   CURSOR emp_cursor IS SELECT first_name FROM employees
                                    WHERE department_id = 60 FOR UPDATE;
BEGIN
   FOR emp_record IN emp_cursor LOOP
            v_filename := emp_record.first_name || '.bmp';
            v_file := BFILENAME (p_file_loc, v_filename);
            DBMS_LOB.FILEOPEN (v_file);
            UPDATE employees SET emp_video = v_file
               WHERE CURRENT OF emp_cursor;
            DBMS_OUTPUT.PUT_LINE('LOADED FILE: '||v_filename
                        || ' SIZE: ' || DBMS_LOB.GETLENGTH (v_file));
            DBMS_LOB.FILECLOSE (v_file);
   END LOOP;
END load_emp_bfile;
/
SET SERVEROUTPUT ON
EXECUTE load_emp_bfile (‘FILES_DIR’)
```

ORACLE

Loading BFILEs

Use the DBMS_LOB.FILEEXISTS function to vefiry that the file exists in the operating system. The function returns 0 if the file does not exist, and returns 1 if the file does exist.

```
CREATE OR REPLACE PROCEDURE load_emp_bfile
   (p_file_loc IN VARCHAR2)
IS
   v_file BFILE; v_filename VARCHAR2(16);
   v_file_exists BOOLEAN;
   CURSOR emp_cursor IS SELECT first_name FROM employees
                              WHERE  department_id = 60 FOR UPDATE;
BEGIN
     FOR emp_record IN emp_cursor LOOP
     v_filename := emp_record.first_name || '.bmp';
     v_file := BFILENAME (p_file_loc, v_filename);
     v_file_exists := (DBMS_LOB.FILEEXISTS  (v_file) = 1);
     IF v_file_exists THEN
              DBMS_LOB.FILEOPEN (v_file); ...
```

ORACLE

Migrating from LONG to LOB

The Oracle9*i* server allows 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: 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 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

The DBMS_LOB Package

- Working with LOB often requires the use of the Oracle-supplied package DBMS_LOB.
- DBMS_LOB provides routines to access and manipulate internal and external LOBs.
- Oracle9*i* enables retrieving LOB data directly using SQL, without using any special LOB API.
- In PL/SQL you can define a VARCHAR2 for a CLOB and a RAW for BLOB.

The 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

The 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
```

Adding LOB Columns to a Table

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

Table altered.

Populating LOB Columns

- Insert a row into a table with LOB columns:

```
INSERT INTO employees (employee_id, first_name, last_name,
email,                  hire_date, job_id, salary, resume, picture)
VALUES ( 405, 'Marvin', 'Ellis', 'MELLIS',
              SYSDATE, 'AD_ASST', 4000, EMPTY_CLOB(),NULL);
```

```
1 row created.
```

- Initialize a LOB column using the EMPTY_BLOB() function:

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

```
1 row updated.
```

Updating LOB by Using SQL

UPDATE CLOB column

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

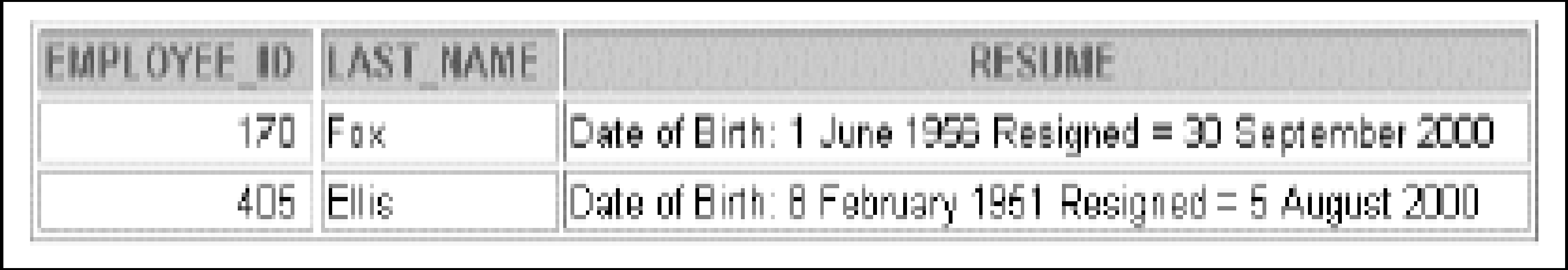
```
1 row updated.
```

Updating LOB by Using DBMS_LOB in PL/SQL

```
DECLARE
    lobloc CLOB;              -- serves as the LOB locator
    text VARCHAR2(32767):='Resigned: 5 August 2000';
    amount NUMBER ;           -- amount to be written
    offset INTEGER;           -- where to start writing
BEGIN
    SELECT resume INTO lobloc  FROM employees
    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 employees
    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, last_name , resume -- CLOB
FROM employees
WHERE employee_id IN (405, 170);
```

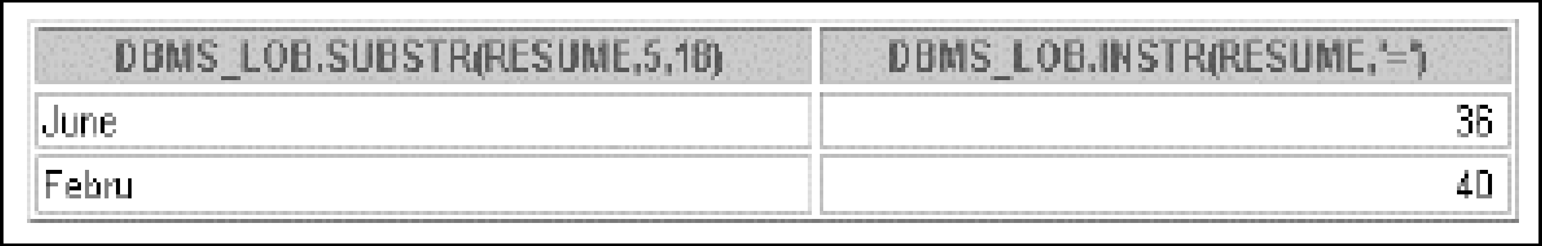


EMPLOYEE_ID	LAST_NAME	RESUME
170	Fox	Date of Birth: 1 June 1956 Resigned = 30 September 2000
405	Ellis	Date of Birth: 8 February 1951 Resigned = 5 August 2000

Selecting CLOB Values by Using DBMS_LOB

- DBMS_LOB.SUBSTR (lob_column, no_of_chars, starting)
- DBMS_LOB.INSTR (lob_column, pattern)

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



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

Selecting CLOB Values in PL/SQL

```
DECLARE
    text VARCHAR2(4001);
BEGIN
               SELECT resume INTO text
               FROM employees
               WHERE employee_id = 170;
               DBMS_OUTPUT.PUT_LINE('text is: '|| text);
END;
/
```

```
Text is: Date of Birth: 1 June 1956 Resigned = 30 September 2000
PL/SQL procedure successfully completed.
```

Removing LOBs

- Delete a row containing LOBs:

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

1 row deleted.

- Disassociate a LOB value from a row:

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

1 row updated.

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 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.

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 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

```
SET SERVEROUTPUT ON
DECLARE
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME ('FILES_DIR', 'John.BMP');
  Amount        INTEGER := 4000;
BEGIN
  DBMS_LOB.CREATETEMPORARY (Dest_loc,TRUE);
  /* Opening the BFILE is mandatory: */
  DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
  DBMS_OUTPUT.PUT_LINE ('Kich thuoc file nguon :'||
                                       DBMS_LOB.GETLENGTH (SRC_LOC));
  /* Opening the LOB is optional: */
  DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
  DBMS_OUTPUT.PUT_LINE ('Kich thuoc truoc khi load :'||
                                       DBMS_LOB.GETLENGTH (DEST_LOC));
  DBMS_LOB.LOADFROMFILE (Dest_loc, Src_loc, Amount);
  DBMS_OUTPUT.PUT_LINE ('Kich thuoc sau khi load :'||
                                       DBMS_LOB.GETLENGTH (DEST_LOC));
  /* Closing the LOB is mandatory if you have opened it: */
  DBMS_LOB.CLOSE (Src_loc);
  DBMS_LOB.CLOSE (Dest_loc);
  /* Free the temporary LOB: */
  DBMS_LOB.FREETEMPORARY(Dest_loc);
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 LOBs replace LONG and LONG RAW
- Describe two storage options for LOBs:
  - The Oracle server (internal LOBs)
  - External host files (external LOBs)
- Use the DBMS_LOB PL/SQL package to provide routines for LOB management
- Use temporary LOBs in a session

Practice 15 Overview

This practice covers the following topics:

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