Oracle8*i*

Application Developer's Guide - Large Objects (LOBs)

Release 8.1.5

February, 1999

Part No. A68004-01

ORACLE

Oracle8*i* Application Developer's Guide - Large Objects (LOBs), Release 8.1.5

Part No. A68004-01

Primary Author: Denis Raphaely, Susan Kotsovolos

Contributing Authors: Rosanne Park, John Gibb

Contributors: Michael Chiocca, R. Govindarajan, Gopal Kirsur, Anindo Roy

Contents

2 Advanced Topics

3 Internal Persistent LOBs

4 Temporary LOBs

5 External LOBs (BFILEs)

6 LOBs and Partitioned Tables

Index

Send Us Your Comments

Oracle8*i* Application Developer's Guide - Large Objects (LOBs), Release 8.1.5

Part No. A68004-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to the Information Development department in the following ways:

- Electronic mail - infodev@us.oracle.com
- FAX - (650) 506-7228 Attn: Oracle Server Documentation
- Postal service:
  Oracle Corporation
  Server Documentation Manager
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This Guide describes features of application development on the Oracle Server having to do with *Large Objects (LOBs)*. Information in this Guide applies to versions of the Oracle Server that run on all platforms, and does not include system-specific information.

The Preface includes the following sections:

Information in This Guide

The *Oracle8i Application Developer's Guide - Large Objects (LOBs)* is intended for programmers developing new applications that use LOBs, as well as those who have already implemented this technology and now wish to take advantage of new features.

The increasing importance of multimedia data as well as unstructured data has led to this topic being presented as an independent volume within the Oracle Application Developers documentation set.

Feature Coverage and Availability

The *Oracle8i Application Developer's Guide - Large Objects (LOBs)* contains information that describes the features and functionality of the Oracle8 and the Oracle8 Enterprise Edition products. Oracle8 and Oracle8 Enterprise Edition have the same basic features. However, several advanced features are available only with the Enterprise Edition, and some of these are optional. For example, to use object functionality, you must have the Enterprise Edition and the Objects Option.

There are no special restrictions in dealing with `LOB`s. However, you will need the Partitioning option to use `LOB`s in partitioned tables. Also, you will not be able to use `LOB`s with object types unless you have purchased the object option. For information about the differences between Oracle8 and the Oracle8 Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8i and the Oracle8i Enterprise Edition.*

New Features Introduced with Oracle8*i*

The new features included in the Oracle8*i*, release 8.1.5 are as follows:

- Temporary `LOB`s
- Varying width `CLOB` and `NCLOB` support
- Support for `LOB`s in partitioned tables
- New API for `LOB`s (`open`/`close`/`isopen`, `writeappend`, `getchunksize`)
- Support for LOBs in non-partitioned index-organized tables
- Copying the value of a LONG to a LOB

Other Guides

Use the *PL/SQL User's Guide and Reference* to learn PL/SQL and to get a complete description of this high-level programming language, which is Oracle Corporation's procedural extension to SQL.

The Oracle Call Interface (OCI) is described in the*Oracle Call Interface Programmer's Guide.* You can use the OCI to build third-generation language (3GL) applications that access the Oracle Server.

Oracle Corporation also provides the Pro* series of precompilers, which allow you to embed SQL and PL/SQL in your application programs. If you write 3GL

application programs in Ada, C, C++, COBOL, or FORTRAN that incorporate embedded SQL, refer to the corresponding precompiler manual. For example, if you program in C or C++, refer to the *Pro*C/C++ Precompiler Programmer's Guide*..

Oracle 8i offers the opportunity of working with Java in the database. The Oracle Java documentation set includes the *Enterprise JavaBeans and CORBA Developer's Guide*, the *Oracle8i JDBC Developer's Guide and Reference* , the *Oracle8i Java Developer's Guide* , the *Oracle8i JPublisher User's Guide* and the *Oracle8i Java Stored Procedures Developer's Guide*.You can access Oracle's development environment for multimedia technology in a number of different ways.

- To build self-contained applications that integrate with the database, you can learn about how to use Oracle's extensibility framework in *Oracle8i Data Cartridge Developer's Guide*
- To utilize Oracle's own intermedia applications, refer to *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference.*

For SQL information, see the *Oracle8i SQL Reference* and *Oracle8i Administrator's Guide*. If you need information about Oracle replication with LOB data, refer to *Oracle8i Replication.* LOBsFor basic Oracle concepts, see *Oracle8i Concepts*..

How This Book Is Organized

The *Oracle8i* Application Developer's Guide - Large Objects (LOBs) contains six chapters organized into two volumes. A brief summary of what you will find in each chapter follows:

VOLUME I

Chapter 1, "Introduction to Working With LOBs"

In this chapter we describe the LOB datatype in terms of three main kinds of LOBs: Internal persistent LOBs, Internal temporary LOBs, and External LOBs (BFILEs). We discuss the use of LOBs to promote internationalization by way of CLOBS, and the advantages of using LOBs over LONGs. We then turns to the various programmatic environments by which you can operate on LOBs

- The PL/SQL language by means of the **DBMS_LOB package** as described in *Oracle8i Application Developer's Reference - Packages.*
- The C language by means of the **Oracle Call Interface (OCI)** described in the *Oracle Call Interface Programmer's Guide*

- The C++ language by means of the **Pro*C/C++ precompiler** as described in the *Pro*C/C++ Precompiler Programmer's Guide*
- The COBOL language by means of the **Pro*COBOL precompiler** as described in the *Pro*COBOL Precompiler Programmer's Guide*
- The Visual Basic language by means of **Oracle Objects For OLE (OO4O)** as described in its accompanying online documentation.
- The Java language by means of the **JDBC Application Programmers Interface (API)** as described in the *Oracle8i JDBC Developer's Guide and Reference* .

  The chapter also includes an example scenario that frames examples provided throughout the rest of the book. Various general topics that underlie LOB operations are discussed as an introduction to the later chapters.

Chapter 2, "Advanced Topics"

The last chapter in the book covers advanced topics that touch on all the other chapters. Specifically, we focus on:

- Read consistency
- The LOB buffering subsystem
- LOBs and the issue of spanning transactions
- LOBs in the object cache
- Working with varying-width character data
- Guidelines for optimal performance

VOLUME II

Chapter 3, "Internal Persistent LOBs"

The basic operations concerning internal persistent LOBs are discussed, along with pertinent issues in the context of the scenario outlined in Chapter 1. We introduce the Unified Modeling Language (UML) notation with a special emphasis on *use cases*. Specifically, each basic operation is described as a use case. A full description of UML is beyond the scope of this book, but the small set of conventions used in this book appears later in the Preface. Wherever possible, we provide the same example in each of the programmatic environments.

Chapter 4, "Temporary LOBs"

This chapter follows the same pattern as Chapter 2 but here focuses on the new feature of temporary LOBs. The new API and its attendant issues are discussed in detail.

Chapter 5, "External LOBs (BFILEs)"

The focus in this chapter is on external LOBs, also known as BFILEs. The same treatment is provided here as in Chapters 2 and 3, namely every operation is treated as a use case, and we provide matching code examples in every available programmatic environment.

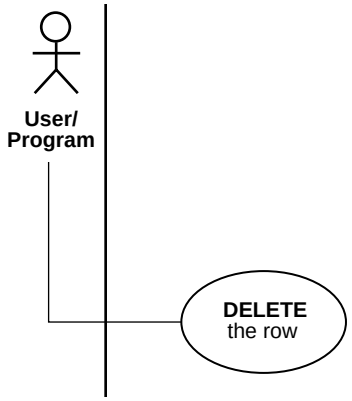
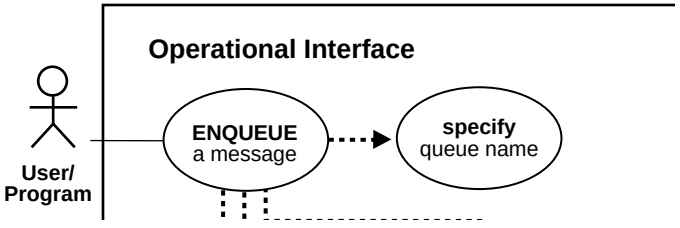
Chapter 6, "LOBs and Partitioned Tables"

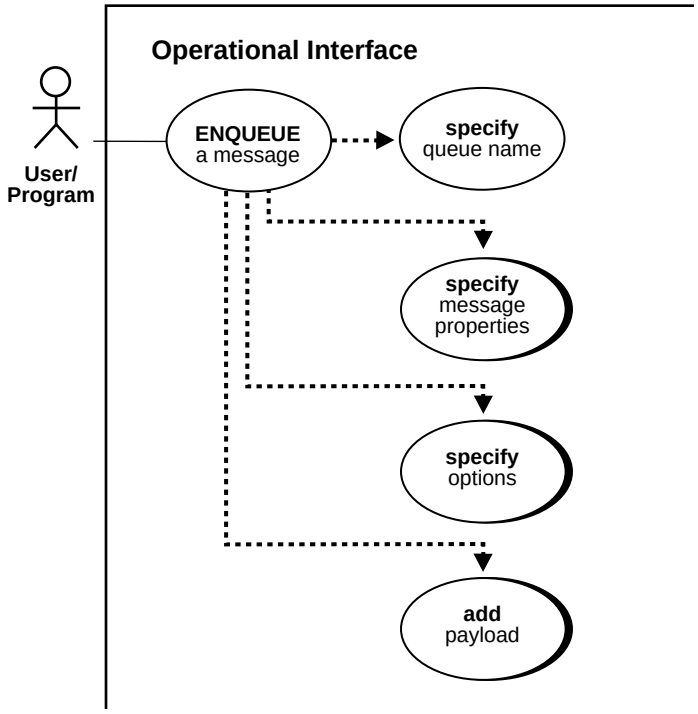
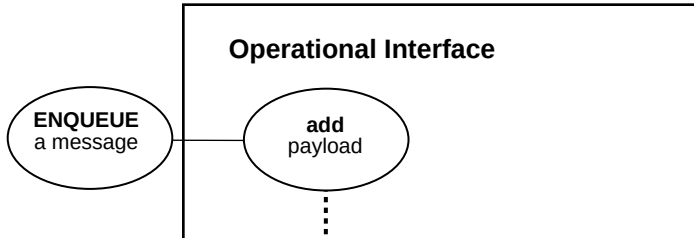
This new feature is also presented in terms of the overarching scenario. Please note that using LOBs in partitioned tables requires that you purchase the partition option.

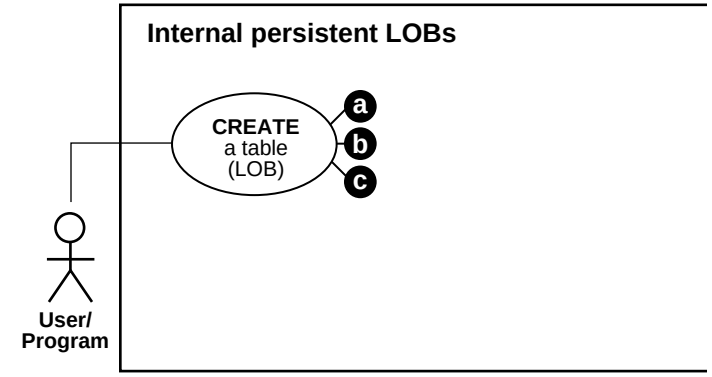
Visual Modeling

This release of the documentation introduces the Universal Modeling Language (UML) as a way of explaining the technology that we hope will help you develop applications. A full presentation of the UML is beyond the scope of this documentation set, however we do provide a description of the subset of UML notation that we use in a chapter devoted to visual modeling in*Oracle8i Application Developer's Guide - Fundamentals*. What follows here is a selection from that chapter of those elements that are used in this book.

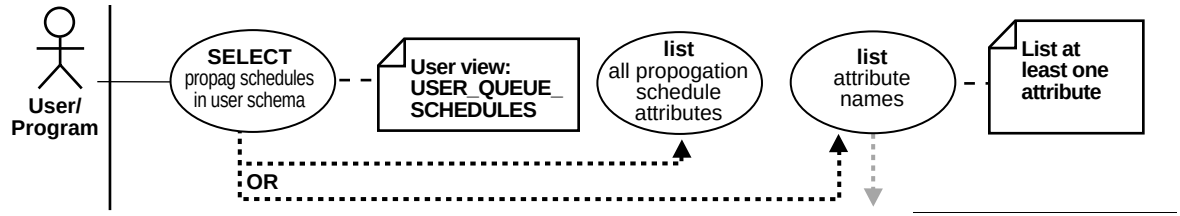
Use Case Diagrams

Graphic Element	Description
	This release of the documentation introduces and makes heavy use of the *Use Case Diagram*. Each primary use case is instigated by an *actor* ('stickman') that could be a human user, an application, or a sub-program. The actor is connected to the primary use case which is depicted as an oval (bubble) enclosing the use case action. The totality of primary use cases is described by means of a *Use Case Model Diagram*.
	Primary use cases may require other operations to complete them. In this diagram fragment ▪ **`specify`** `queue name` is one of the sub-operations, or secondary use cases, needed to complete ▪ **`ENQUEUE`** `a message` The downward lines from the primary use case lead to the other required operations (not shown).

Graphic Element	Description
	Secondary use cases that have drop shadows 'expand' in that they are described by means of their own use case diagrams. There are two reasons for doing this: (a) it makes it easier to understand the logic of the operation; (b) it would not have been possible to place all the operations and sub-operations on the same page. In this example ▪ **specify** message properties, ▪ **specify** options ▪ **add** payload are all expanded in separate use case diagrams.
	This diagram fragment shows the use case diagram ad expanded. While the standard diagram has the actor as the initiator), here the use case itself is the point of departure for the sub-operation. In this example, the expanded view of ▪ **add** payload represents a constituent operation of ▪ **ENQUEUE** a message

Graphic Element	Description
	This convention (a, b, c) shows that there are three different ways of creating a table that contains LOBs.
	This fragment shows one of the uses of a NOTE box, here distinguishing the first of a number of ways of creating a table containing LOBs.

Graphic Element



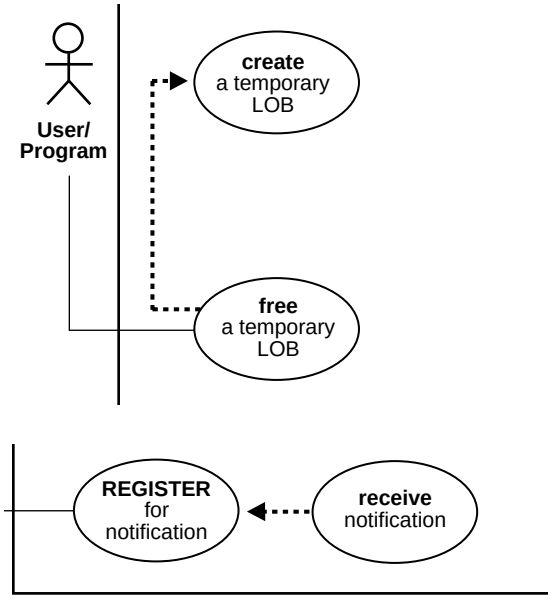
Description

This drawing shows two other common use of NOTE boxes:

(a) as a way of presenting an alternative name, as in this case the action **SELECT** `propagation schedules` in `the user schema` is represented by the view `USER_QUEUE_SCHEDULES`

(b) the action **`list`** `attribute names` is qualified by the note to the user that you must list at least one attribute if you elect not to list all the propagation schedule attributes.

Graphic Element



Description

The dotted arrow in the use case diagram indicates dependency. In this example

- **`free`** a `temporary LOB`

requires that you first

- **`create`** a `temporary LOB`

Put another way: you should not execute the free operation on a `LOB` that is not temporary.

What you need to remember is that the target of the arrow shows the operation that must be performed first.

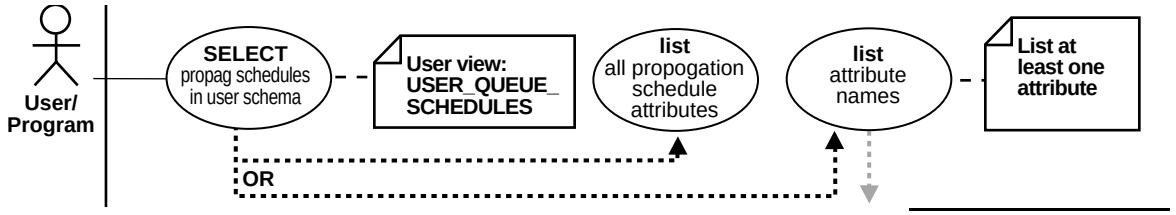
Use cases and their sub-operations can be linked in complex relationships. In this example of a callback, you must earlier

- **`REGISTER`** `for notification`

in order to later

- **`receive`** a `notification`

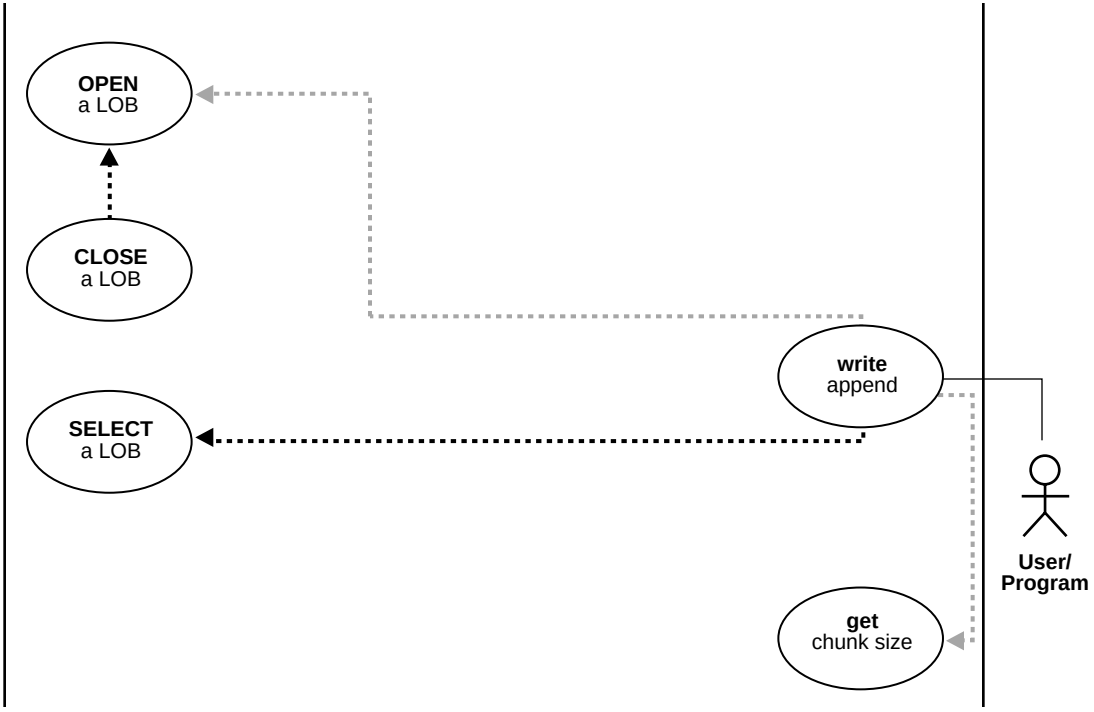
Graphic Element



Description

In this case the branching paths of an OR condition are shown. In invoking the view, you may choose either to list all the attributes or you may view one or more attributes. The fact that you may stipulate which of the attributes you wish made visible is indicated by the grayed arrow.

Graphic Element



Description

Not all lined operations are mandatory. While the black dashed-line and arrow indicate that you must perform the targeted operation to complete the use case, actions that are optional are shown by the grey dashed-line and arrow. In this example, executing

- **`write`** append

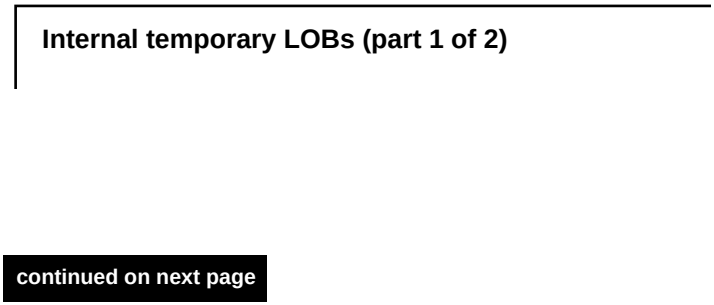
on a LOB requires that you first

- **`SELECT`** a LOB

As a facilitating operations, you may choose to

- **`OPEN`** a LOB and/or **`get`** chunk size

However, note that if you **`OPEN`** a LOB, you will later have to **`CLOSE`** `it`.

Graphic Element	Description
	*Use Case Model Diagrams* summarize all the use cases in a particular domain, such as `Internal temporary LOBs`. Often these diagrams are too complex to contain within a single page. When that happens we have resorted to dividing the diagram into two parts. Please note that there is no sequence implied in this division. In some cases we have had to split a diagram simply because it is too long for the page. In such cases, we have included this marker.

Conventions Used in this Guide

The following notational and text formatting conventions are used in this guide:

[ ]
Square brackets indicate that the enclosed item is optional. Do not type the brackets.

{ }
Braces enclose items of which only one is required.

|
A vertical bar separates items within braces, and may also be used to indicate that multiple values are passed to a function parameter.

...
In code fragments, an ellipsis means that code not relevant to the discussion has been omitted.

`font change`
SQL or C code examples are shown in monospaced font.

italics
Italics are used for OCI parameters, OCI routines names, file names, and data fields.

UPPERCASE
Uppercase is used for SQL keywords, like SELECT or UPDATE.

This guide uses special text formatting to draw the reader's attention to some information. A paragraph that is indented and begins with a bold text label may have special meaning. The following paragraphs describe the different types of information that are flagged this way.

Note: The "Note" flag indicates that the reader should pay particular attention to the information to avoid a common problem or increase understanding of a concept.

Warning: An item marked as "Warning" indicates something that an OCI programmer must be careful to do or not do in order for an application to work correctly.

See Also: Text marked "See Also" points you to another section of this guide, or to other documentation, for additional information about the topic being discussed.

Your Comments Are Welcome

We value and appreciate your comment as an Oracle user and reader of our manuals. As we write, revise, and evaluate our documentation, your opinions are the most important feedback we receive.

You can send comments and suggestions about this manual to the following e-mail address:

infodev@us.oracle.com

If you prefer, you can send letters or faxes containing your comments to the following address:

Server Technologies Documentation Manager

Oracle Corporation

500 Oracle Parkway

Redwood Shores, CA 94065

Fax: (650) 506-7228

1

Introduction to Working With LOBs

This introductory chapter discusses with the following topics:

- The LOB Datatype
- Varying-Width Character Data
- LOBs in Comparison to LONG and LONG RAW Types
- LOB Restrictions
- Using SQL DML for Basic Operations on LOBs
- Programmatic Environments for Operating on LOBs
- An Example Application
- The Most Basic Operation: Getting and Using the LOB Locator
- Indexing a LOB Column

The LOB Datatype

Oracle8 regards LOBs as being of two kinds depending on their location with regard to the database — **internal LOBs** and **external LOBs**, also referred to as **BFILEs** (binary files). Note that when we discuss some aspect of working with LOBs without specifying whether the LOB is internal or external, the characteristic under discussion pertains to both internal and external LOBs.

Internal LOBs are further divided into those that are **persistent** and those that are **temporary**.

Internal LOBs

Internal LOBs, as their name suggests, are stored inside database tablespaces in a way that optimizes space and provides efficient access. Internal LOBs use copy semantics and participate in the transactional model of the server. You can recover internal LOBs in the event of transaction or media failure, and any changes to a internal LOB value can be committed or rolled back. In other words, all the ACID properties that pertain to using database objects pertain to using internal LOBs.

Internal LOB Datatypes

There are three SQL datatypes for defining instances of internal LOBs:

- **BLOB**, a LOB whose value is composed of unstructured binary ("raw") data.
- **CLOB**, a LOB whose value is composed of character data that corresponds to the database character set defined for the Oracle8 database.
- **NCLOB**, a LOB whose value is composed of character data that corresponds to the national character set defined for the Oracle8 database.

External LOBs (BFILEs)

External LOBs (BFILES) are large binary data objects stored in operating system files outside of database tablespaces. These files use reference semantics. Apart from conventional secondary storage devices such as hard disks, BFILEs may also be located on tertiary block storage devices such as CD-ROM, PhotoCDs and DVDs. But note that you cannot locate a single BFILE on more than one device, for instance, striped across a disk array.

The SQL datatype BFILE allows *read-only* byte stream I/O access to large files existing on the filesystem of the database server. The Oracle Server can access

BFILEs provided the underlying server operating system supports a stream-mode access to these operating system (OS) files.

> **Note:** External LOBs do not participate in transactions. Any support for integrity and durability must be provided by the underlying file system as governed by the operating system.

External LOB Datatype

There is one external SQL LOB datatype:

- **BFILE**, a LOB whose value is composed of binary ("raw") data, and is stored outside of the database tablespaces in a server-side operating system file.

Varying-Width Character Data

You can create a table with CLOB/NCLOB columns even if the CHAR/NCHAR database character set is varying width. You can also create a table with a type that has a CLOB attribute irrespective of whether the CHAR database character set is of varying width. However, NCLOBs are not allowed as attributes in object types.

The CLOB/NCLOB value is stored in the database using the 2 byte Unicode character set which is fixed width. The stored Unicode value is translated to the (possibly varying width) character set that you request on either the client or the server. When you insert data into the CLOB/NCLOB, the data input can be in a varying width character set. This varying width character data is implicitly converted into Unicode before the data is stored in the database. Note that all translations to and from Unicode are implicitly performed by Oracle.

You can perform the full gamut of LOB operations on CLOB/NCLOBs (read, write, trim, erase, compare, etc.) All programmatic environments that provide access to CLOBs/NCLOBs work on CLOBs/NCLOBs in databases where the CHAR/NCHAR character set is of varying width. This includes SQL, PL/SQL, OCI, PRO*C, DBMS_LOB, and so on. However, you should take note of the following issue that pertain to specific environments.

DBMS_LOB Package

Regardless of the client-side character set, the offset and amount parameters are always in characters for CLOBs/NCLOBs and in bytes for BLOBs/BFILEs.

OCI

The following decisions only apply to varying-width client-side character sets. For fixed-width client side character sets, the offset and amount parameters are always in characters for CLOBs and NCLOBs and in bytes for BLOBs and BFILEs.

General Rule:

- The amount parameter: When the amount parameter refers to the server-side LOB, the amount is in characters. When the amount parameter refers to the client-side buffer, the amount is in bytes.
- The offset parameter: Regardless of whether the client-side character set is varying-width, the offset parameter is always in characters for CLOBs and NCLOBs, and in bytes for BLOBs and BFILEs.
- OCILobFileGetLength: Regardless of whether the client-side character set is varying-width, the output length is in characters for CLOBs and NCLOBs and in bytes for BLOBs and BFILEs.
- OCILobRead: If the client-side character set is varying-width, for CLOBs and NCLOBs, the input amount is in characters and the output amount is in bytes. The input amount refers to the number of characters to read from the server-side CLOB or NCLOB. The output amount indicates how many bytes were read into the buffer 'bufp'.
- OCILobWrite: If the client-side character set is varying-width, for CLOBs and NCLOBs, the input amount is in bytes and the output amount is in characters. The input amount refers to the number of bytes of data that are in the input buffer 'bufp'. The output amount refers to the number of characters written into the server-side CLOB or NCLOB.

Other Operations:

For all other LOB operations, irrespective of the client-side character set, the amount parameter is in characters for CLOBs and NCLOBs. These include OCILobCopy, OCILobErase, OCILobLoadFromFile, and OCILobTrim. All these operations refer to the amount of LOB data on the server.

For more information, see: *Oracle8i National Language Support Guide*

LOBs in Comparison to LONG and LONG RAW Types

LOBs are similar to LONG and LONG RAW types, but differ in the following ways:

- You can store multiple LOBs in a single row but you can store only one LONG or LONG RAW per row.
- A LOBs can be attributes of a user-defined datatype but this is not possible with either a LONG or LONG RAW.
- Only the LOB locator is stored in the table column; BLOB and CLOB data can be stored in separate tablespaces and BFILE data is stored as an external file. In the case of a LONG or LONG RAW the entire value is stored in the table column. For inline LOBs, Oracle will store up to 3964 bytes of data in the table column.
- When you access a LOB column, it is the locator which is returned. When you access a LONG or LONG RAW, the entire value is returned.
- A LOB can be up to 4 gigabytes in size. The BFILE maximum is operating system dependent, but cannot exceed 4 gigabytes. The valid accessible range is 1 to (2^{32}-1). By contrast, a LONG or LONG RAW is limited to 2 gigabytes.
- There is greater flexibility in manipulating data in a random, piece-wise manner with LOBs than there is with LONG or LONG RAW data. LOBs can be accessed at random offsets while LONGs must be accessed from the beginning to the desired location
- You can replicate LOBs in both local and distributed environments, but this is not possible with aLONG or LONG RAW (see Oracle8i Replication).

Existing LONG columns can be converted to LOBs using the TO_LOB() function (see "Copy LONG to LOB" on page 2-62 in Chapter 2, ". Internal Persistent LOBs"). However note that Oracle8*i* does not support conversion of LOBs back to LONGs.

LOB Restrictions

The use of LOBs are subject to some restrictions:

- Distributed LOBs are not supported. Specifically, this means that the user cannot use a remote locator in the SELECT and WHERE clauses. This includes using DBMS_LOB package functions. In addition, references to objects in remote tables with or without LOB attributes is not allowed.

  For example, the following operations are invalid:

  - SELECT lobcol from table1@remote_site;
  - INSERT INTO lobtable select type1.lobattr from table1@remote_site;
  - SELECT dbms_lob.getlength(lobcol) from table1@remote_site;

  Valid operations on LOB columns in remote tables include:

  - CREATE TABLE as select * from table1@remote_site;
  - INSERT INTO t select * from table1@remote_site;
  - UPDATE t set lobcol = (select lobcol from table1@remote_site);
  - INSERT INTO table1@remote...
  - UPDATE table1@remote...
  - DELETE table1@remote...
- When binding an internal LOB in order to use piece-wise INSERT/UPDATE, the bind variable may be of type SQLT_CHR or SQLT_LBI but is limited to 4k. You cannot bind a SQLT_LNG to a LOB or a SQLT_LBI that is longer than 4k.

Also, LOBs are not allowed in the following places:

- LOBs are not allowed in clustered tables and thus cannot be a cluster key.
- LOBs are not allowed in GROUP BY, ORDER BY, SELECT DISTINCT, aggregates and JOINS. However, UNION ALL is allowed on tables with LOBs. UNION, MINUS, and SELECT DISTINCT are allowed on LOB attributes if the object type has a MAP or ORDER function.
- LOBS are not analyzed in ANALYZE... COMPUTE/ESTIMATE STATISTICS statements.
- LOBs are not allowed in partitioned index organized tables but are allowed non-partitioned index organized tables.
- LOBs are not allowed in VARRAYs.

- NCLOBs are not allowed as attributes in object types but NCLOB parameters are allowed in methods.
- You can use the LOB column / attribute in a trigger body subject to the following conditions. In general, the :new and :old LOB values bound in the trigger are read-only which means that you cannot write to the LOB. More specifically:
  - **a.** In before row and after row triggers -
    - * you can read the :old value of a LOB in both the triggers.
    - * you can read the :new value of the LOB only in an after-row trigger.
  - **b.** In INSTEAD OF triggers on views, you can read both the :new and :old values.
  - **c.** You cannot specify the LOB column in an OF clause (Note that a BFILE can be modified without updating the underlying tables on which it is based).
  - **d.** If you use OCI functions or DBMS_LOB routines to update LOB values or LOB attributes on object columns, the functions or routines will not fire the triggers defined on the tables containing the columns or attributes.

For more information about firing triggers on extensible indexes see:

- *Oracle8i Data Cartridge Developer's Guide*

- Client-side PL / SQL procedures may not call the DBMS_LOB package routines. However, you can use server-side PL / SQL procedures or anonymous blocks in Pro*C / C++ to call the DBMS_LOB package routines.

DBA Actions Required Prior to Working with LOBs

Set Maximum Number of Open BFILEs

A limited number of BFILEs can be open simultaneously per session. 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. That is, you can open a maximum of 10 files at the same time per session if the default value is utilized. If you want to alter this limit, the database administrator can change the value of this parameter in the init.ora file**.** For example:

```
SESSION_MAX_OPEN_FILES=20
```

If the number of unclosed files exceeds the SESSION_MAX_OPEN_FILES value then you will not be able to open any more files in the session. To close all open files, use the FILECLOSEALL call.

Using SQL DML for Basic Operations on LOBs

SQL DML provides basic operations — INSERT, UPDATE, SELECT, DELETE — that let you make changes to the entire values of *internal* LOBs within the Oracle ORDBMS. To work with parts of internal LOBs, you will need to use one of the interfaces that have been developed to handle more complex requirements.

Oracle8 supports read-only operations on external LOBs. So if you need to update / write to external LOBs, you will have to develop client side applications suited to your needs

Programmatic Environments for Operating on LOBs

Oracle now offers you six different environments for working with LOBs:

- The PL/SQL language by means of the **DBMS_LOB package** as described in *Oracle8i Application Developer's Reference - Packages* (see "Using the DBMS_LOB Package for Working With LOBs" on page 1-12).
- The C language by means of the **Oracle Call Interface (OCI)** as described in the *Pro*COBOL Precompiler Programmer's Guide*(see "Using the Oracle Call Interface (OCI) with LOBs" on page 1-15).
- The C++ language by means of the **Pro*C/C++ precompiler** as described in the *Pro*C/C++ Precompiler Programmer's Guide* (see "Using C++ (Pro*C/C++) to Work with LOBs" on page 1-23).
- The COBOL language by means of the **Pro*COBOL precompiler** as described in the *Pro*COBOL Precompiler Programmer's Guide* (see "Using COBOL (Pro*COBOL) to Work with LOBs" on page 1-26).
- The Visual Basic language by means of **Oracle Objects For OLE (OO4O)** as described in its accompanying online help (see "Using Visual Basic (OO4O) to Work with LOBs" on page 1-29).
- The Java language by means of the **JDBC Application Programmers Interface (API)** as described in the *Oracle8i Java Developer's Guide* (see "Using Java (JDBC) to Work with LOBs" on page 1-34).

Comparison of Six Interfaces

The following chart compares the six LOB interfaces.

Table 1–1 Comparison of Interfaces for working with LOBs

OCI (ociap.h)	DBMS_LOB (dbmslob.sql)	Pro*C & Pro*COBOL	Visual Basic	Java
N / A	DBMS_ LOB.COMPARE	N / A	ORALOB.Compare	use DBMS_ LOB.COMPARE
N / A	DBMS_LOB.INSTR	N / A	ORALOB.Matchpos	position
N / A	DBMS_LOB.SUBSTR	N / A	N / A	getBytes
OCILobAppend	DBMS_LOB.APPEND	APPEND	ORALOB.Append	use length and then putBytes
OCILobAssign	N / A [use Pl / SQL assign operator]	ASSIGN	ORALOB.Clone	N / A [use equal sign]
OCILobCharSetForm	N / A	N / A	N / A	N / A
OCILobCharSetId	N / A	N / A	N / A	N / A
OCILobClose	DBMS_LOB.CLOSE	CLOSE	N / A	BLOB / CLOB: uze close() on stream object BFILE: use DBMS_ LOB.CLOSE
OCILobCopy	DBMS_LOB.COPY	COPY	ORALOB.Copy	use read and write
OCILobDisableBuffering	N / A	DISABLE BUFFERING	ORALOB.DisableBuffering	N / A
OCILobEnableBuffering	N / A	ENABLE BUFFERING	ORALOB.EnableBuffering	N / A
OCILobErase	DBMS_LOB.ERASE	ERASE	ORALOB.Erase	use DBMS_ LOB.ERASE
OCILobFileClose	DBMS_ LOB.FILECLOSE	CLOSE	ORABFILE.Close	closeFile
OCILobFileCloseAll	DBMS_ LOB.FILECLOSEALL	FILE CLOSE ALL	ORABFILE.CloseAll	use DBMS_ LOB.FILECLOSEALL
OCILobFileExists	DBMS_ LOB.FILEEXISTS	DESCRIBE [FILEEXISTS]	ORABFILE.Exist	fileExists

Table 1–1 Comparison of Interfaces for working with LOBs (Cont.)

OCI (ociap.h)	DBMS_LOB (dbmslob.sql)	Pro*C & Pro*COBOL	Visual Basic	Java
OCILobFileGetChunkSize	DBMS_ LOB.GETCHUNKSIZE	DESCRIBE [CHUNKSIZE]	ORALOB.ChunkSize	N / A
OCILobFileGetName	DBMS_ LOB.FILEGETNAME	DESCRIBE [DIRECTORY, FILENAME]	ORABFILE.DirectoryName ORABFILE.FileName	getDirAlias getName
OCILobFileIsOpen	DBMS_ LOB.FILEISOPEN	DESCRIBE [ISOPEN]	ORABFILE.IsOpen	use DBMS_ LOB.ISOPEN
OCILobFileOpen	DBMS_ LOB.FILEOPEN	OPEN	ORABFILE.Open	openFile
OCILobFileSetName	N / A (use BFILENAME operator)	FILE SET	DirectoryName FileName	use BFILENAME
OCILobFlushBuffer	N / A	FLUSH BUFFER	ORALOB.FlushBuffer	N / A
OCILobGetLength	DBMS_ LOB.GETLENGTH	DESCRIBE [LENGTH]	ORALOB.Size	length
OCILobIsEqual	N / A	N / A	N / A	equals
OCILobIsOpen	DBMS_LOB.ISOPEN	DESCRIBE [ISOPEN]	ORALOB.IsOpen	BLOB / CLOB: create stream object BEILE: use DBMS_LOB.ISOPEN
OCILobLoadFromFile	DBMS_ LOB.LOADFROMFILE	LOAD FROM FILE	ORALOB.CopyFromBfile	use read and then write
OCILobLocatorIsInit	N / A [always initialize]	N / A	ORALOB.IsNull	N / A
OCILobOpen	DBMS_LOB.OPEN	OPEN	ORALOB.open	use DBMS_ LOB.OPEN
OCILobRead	DBMS_LOB.READ	READ	ORALOB.Read	getBytes
OCILobTrim	DBMS_LOB.TRIM	TRIM	ORALOB.Trim	use DBMS_ LOB.TRIM
OCILobWrite	DBMS_LOB.WRITE	WRITEORALOB.	Write	putBytes
OCILobWriteAppend	DBMS_ LOB.WRITEAPPEND	WRITE APPEND	N / A	use length and then putBytes

The following subsections describe each of the interfaces in more detail.

Using the DBMS_LOB Package for Working With LOBs

The DBMS_LOB package can be used to read and modify internal LOBs (persistent and temporary) either entirely or in a piece-wise manner. This package can also be used for read operations on BFILEs.

For more information see:

- *Oracle8i Application Developer's Reference - Packages* for detailed documentation, including parameters, parameter types, return values, and example code.

As described in more detail below, DBMS_LOB routines work based on LOB locators. For the successful completion of DBMS_LOB routines, you must provide an input locator that represents a LOB that exists in the database tablespaces or external filesystem *before* you invoke the routine.

For internal LOBs, you must first use SQL DDL to define tables that contain LOB columns, and subsequently SQL DML to initialize or populate the locators in these LOB columns.

For external LOBs, you must define a DIRECTORY object that maps to a valid physical directory containing the external LOBs that you intend to access. Also, these files must exist, and must be set to have read permissions for the Oracle server process. If your operating system uses case-sensitive path names, be sure you specify the directory in the correct format.

Once the LOBs are defined and created, you may then SELECT a LOB locator into a local PL/SQL LOB variable and use this variable as an input parameter to DBMS_LOB for access to the LOB value. Examples provided with each DBMS_LOB routine will illustrate this in the following sections.

The routines that can modify BLOB, CLOB, and NCLOB values are:

Table 1–2 DBMS_LOB Routines that Modify BLOB, CLOB, and NCLOB values

Function/Procedure	Description
APPEND()	appends the LOB value to another LOB
COPY()	copies a portion of a LOB to another LOB
ERASE()	erases part of a LOB, starting at a specified offset
LOADFROMFILE()	load BFILE data into an internal LOB

Table 1–2 (Cont.) DBMS_LOB Routines that Modify BLOB, CLOB, and NCLOB

Function/Procedure	Description
TRIM()	trims the LOB value to the specified shorter length
WRITE()	writes data to the LOB at a specified offset
WRITEAPPEND()	writes data to the end of the LOB

The routines involved in reading or examining LOB values are:

Table 1–3 DBMS_LOB Routines Involved in Reading or Examining LOB values

Function/Procedure	Description
COMPARE()	compares the value of two LOBs
GETCHUNKSIZE()	gets the chunk size for reading and writing
GETLENGTH()	gets the length of the LOB value
INSTR()	returns the matching position of the nth occurrence of the pattern in the LOB
READ()	reads data from the LOB starting at the specified offset
SUBSTR()	returns part of the LOB value starting at the specified offset

The following routines have to do with temporary lobs:

Table 1–4 DBMS_LOB Routines that Operate on Temporary LOBs

Function/Procedure	Description
CREATETEMPORARY()	creates a temporary LOB
ISTEMPORARY()	checks if a LOB locator refers to a temporary LOB
FREETEMPORARY()	frees a temporary LOB

The read-only routines specific to BFILEs are:

Table 1–5 DBMS_LOB Read-Only Routines that are Specific to BFILEs

Function/Procedure	Description
FILECLOSE()	closes the file
FILECLOSEALL()	closes all previously opened files
FILEEXISTS()	checks if the file exists on the server
FILEGETNAME()	gets the directory alias and file name
FILEISOPEN()	checks if the file was opened using the input BFILE locators
FILEOPEN()	opens a file

The following routines have to do with opening and closing LOBs:

Table 1–6 DBMS_LOB Open and Close Routines

Function/Procedure	Description
OPEN()	opens a LOB
ISOPEN()	sees if a LOB is open
CLOSE()	closes a LOB

We will describe these routines in greater detail as we explore specific LOB operations (e.g., INSERT a row containing a LOB).

Using the Oracle Call Interface (OCI) with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it through the OCI API. You can access both internal and external LOBs for read purposes, and you can also write to internal LOBs.

The OCI includes functions that you can use to access data stored in BLOBs, CLOBs, NCLOBs, and BFILEs. These functions are listed in the tables below, and are discussed in greater detail later in the chapter.

Users who want to read or write data in UCS2 format can set the 'csid' parameter in OCILobRead and OCILobWrite to OCI_UCS2ID. The 'csid' parameter indicates the csid for the buffer parameter. You can set the 'csid' parameter to any character set id. If the csid parameter is set, it will override the NLS_LANG environment variable.

For more information see:

- *Oracle Call Interface Programmer's Guide* for detailed documentation, including parameters, parameter types, return values, and example code.
- *Oracle8i National Language Support Guide* for detailed information about implementing applications in different languages.

The routines that can modify BLOB, CLOB, and NCLOB values are:

Table 1–7 OCI Functions that Modify BLOB, CLOB, and NCLOB values

Function/Procedure	Description
OCILobAppend()	appends LOB value to another LOB.
OCILobCopy()	copies a portion of a LOB into another LOB.
OCILobErase()	erases part of a LOB, starting at a specified offset.
OCILobLoadFromFile()	loads BFILE data into an internal LOB.
OCILobTrim()	truncates a LOB.
OCILobWrite()	writes data from a buffer into a LOB, overwriting existing data.

Table 1–7 OCI Functions that Modify BLOB, CLOB, and NCLOB values

Function/Procedure	Description
OCILobWriteAppend()	writes data from a buffer to the end of the LOB.

The routines that read or examine LOB values are:

Table 1–8 OCI Routines that Read or Examine LOB Values

Function/Procedure	Description
OCILobGetChunkSize()	gets the size of the Chunk for reading and writing
OCILobGetLength()	returns the length of a LOB or a BFILE.
OCILobRead()	reads a specified portion of a non-null LOB or a BFILE into a buffer.

The following routines are have to do with temporary lobs:

Table 1–9 OCI Routines that Operate on Temporary LOBs

Function/Procedure	Description
OCILobCreateTemporary()	creates a temporary LOB
OCILobIsTemporary()	sees if a temporary LOB exists
OCILobFreeTemporary()	frees a temporary LOB

Read-only routines specific to BFILEs are:

Table 1–10 OCI Read-Only Routines that are Specific to BFILES

Function/Procedure	Description
OCILobFileClose()	closes an open BFILE.
OCILobFileCloseAll()	closes all open BFILEs.
OCILobFileExists()	checks whether a BFILE exists.

Table 1–10 (Cont.) OCI Read-Only Routines that are Specific to BFILES

Function/Procedure	Description
OCILobFileGetName()	returns the name of a BFILE.
OCILobFileIsOpen()	checks whether a BFILE is open.
OCILobFileOpen()	opens a BFILE.

These routines are used for working with LOB locators:

Table 1–11 OCI LOB-Locator Routines

Function/Procedure	Description
OCILobAssign()	assigns one LOB locator to another.
OCILobCharSetForm()	returns the character set form of a LOB.
OCILobCharSetId()	returns the character set ID of a LOB.
OCILobFileSetName()	sets the name of a BFILE in a locator.
OCILobIsEqual()	checks whether two LOB locators refer to the same LOB.
OCILobLocatorIsInit()	checks whether a LOB locator is initialized.

The following three routines have to do with LOB-buffering:

Table 1–12 OCI LOB-Buffering Routines

Function/Procedure	Description
OCILobDisableBuffering()	disables the buffering subsystem use.
OCILobEnableBuffering()	uses the LOB buffering subsystem for subsequent reads and writes of LOB data.
OCILobFlushBuffer()	flushes changes made to the LOB buffering subsystem to the database (sever)

The following routines have to do with opening and closing LOBs:

Table 1–13 OCI LOB-Buffering Routines

Function/Procedure	Description
`OCILobOpen()`	opens a LOB
`OCILobIsOpen()`	sees if a LOB is open
`OCILobClose()`	closes a LOB

A sample main() and LOB procedure

In order to work with the OCI examples in the remainder of the book, you could use a main() like the following. Here, its use with the seeIfLOBIsOpen procedure is shown as an example.

```
int main(char *argv, int argc)
{
  /* Declare OCI Handles to be used */
  OCIEnv        *envhp;
  OCIServer     *srvhp;
  OCISvcCtx     *svchp;
  OCIError      *errhp;
  OCISession    *authp;
  OCIStmt       *stmthp;
  OCILobLocator *Lob_loc;

  /* Create and Initialize an OCI Environment: */
  (void) OCIEnvCreate(&envhp, (ub4)OCI_DEFAULT, (dvoid *)0,
                      (dvoid * (*)(dvoid *, size_t)) 0,
                      (dvoid * (*)(dvoid *, dvoid *, size_t))0,
                      (void (*)(dvoid *, dvoid *))0,
                      (size_t) 0, (dvoid **) 0);

  /* Allocate error handle: */
  (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &errhp, OCI_HTYPE_ERROR,
                        (size_t) 0, (dvoid **) 0);

  /* Allocate server contexts: */
  (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &srvhp, OCI_HTYPE_SERVER,
                        (size_t) 0, (dvoid **) 0);

  /* Allocate service context: */
  (void) OCIHandleAlloc((dvoid *) envhp, (dvoid **) &svchp, OCI_HTYPE_SVCCTX,
                        (size_t) 0, (dvoid **) 0);
```

```
  /* Attach to the Oracle database:  */
  (void) OCIServerAttach(srvhp, errhp, (text *)"", strlen(""), 0);

  /* Set the server context attribute in the service context: */
  (void) OCIAttrSet ((dvoid *) svchp, OCI_HTYPE_SVCCTX,
                     (dvoid *)srvhp, (ub4) 0,
                     OCI_ATTR_SERVER, (OCIError *) errhp);

  /* Allocate the session handle: */
  (void) OCIHandleAlloc((dvoid *) envhp,
                        (dvoid **)&authp, (ub4) OCI_HTYPE_SESSION,
                        (size_t) 0, (dvoid **) 0);

  /* Set the username in the session handle:*/
  (void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                    (dvoid *) "samp", (ub4)4,
                    (ub4) OCI_ATTR_USERNAME, errhp);
  /* Set the password in the session handle: */
  (void) OCIAttrSet((dvoid *) authp, (ub4) OCI_HTYPE_SESSION,
                    (dvoid *) "samp", (ub4) 4,
                    (ub4) OCI_ATTR_PASSWORD, errhp);

  /* Authenticate and begin the session: */
  checkerr(errhp, OCISessionBegin (svchp,  errhp, authp, OCI_CRED_RDBMS,
                                   (ub4) OCI_DEFAULT));

  /* Set the session attribute in the service context: */
  (void) OCIAttrSet((dvoid *) svchp, (ub4) OCI_HTYPE_SVCCTX,
                    (dvoid *) authp, (ub4) 0,
                   (ub4) OCI_ATTR_SESSION, errhp);

  /* ------- At this point a valid session has been created -----------*/
  printf ("user session created \n");

  /* Allocate a statement handle: */
  checkerr(errhp, OCIHandleAlloc( (dvoid *) envhp, (dvoid **) &stmthp,
           OCI_HTYPE_STMT, (size_t) 0, (dvoid **) 0));

  /* =============  Sample procedure call begins here ==================*/

  printf ("calling seeIfLOBIsOpen...\n");
  seeIfLOBIsOpen(envhp, errhp, svchp, stmthp);

  return 0;
}
```

```
void checkerr(errhp, status)
OCIError *errhp;
sword status;
{
  text errbuf[512];
  sb4 errcode = 0;

  switch (status)
  {
  case OCI_SUCCESS:
    break;
  case OCI_SUCCESS_WITH_INFO:
    (void) printf("Error - OCI_SUCCESS_WITH_INFO\n");
    break;
  case OCI_NEED_DATA:
    (void) printf("Error - OCI_NEED_DATA\n");
    break;
  case OCI_NO_DATA:
    (void) printf("Error - OCI_NODATA\n");
    break;
  case OCI_ERROR:
    (void) OCIErrorGet((dvoid *)errhp, (ub4) 1, (text *) NULL, &errcode,
                       errbuf, (ub4) sizeof(errbuf), OCI_HTYPE_ERROR);
    (void) printf("Error - %.*s\n", 512, errbuf);
    break;
  case OCI_INVALID_HANDLE:
    (void) printf("Error - OCI_INVALID_HANDLE\n");
    break;
  case OCI_STILL_EXECUTING:
    (void) printf("Error - OCI_STILL_EXECUTE\n");
    break;
  case OCI_CONTINUE:
    (void) printf("Error - OCI_CONTINUE\n");
    break;
  default:
    break;
  }
}

/* Select the locator into a locator variable */

sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
```

```
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text      *sqlstmt =
                (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return (0);
}

void seeIfLOBIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  int isOpen;

  /* allocate locator resources */
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

  /* Select the locator */
  (void)select_frame_locator(Lob_loc, errhp, svchp, stmthp);

  /* See if the LOB is Open */
  checkerr (errhp, OCILobIsOpen(svchp, errhp, Lob_loc, &isOpen));

  if (isOpen)
  {
```

```
    printf("  Lob is Open\n");
    /* ... Processing given that the LOB has already been Opened */
  }
  else
  {
    printf("  Lob is not Open\n");
    /* ... Processing given that the LOB has not been Opened */
  }

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Using C++ (Pro*C/C++) to Work with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it by using embedded SQL. You can access both internal and external LOBs for read purposes, and you can also write to internal LOBs.

Embedded SQL statements allow you to access data stored in BLOBs, CLOBs, NCLOBs, and BFILEs. These statements are listed in the tables below, and are discussed in greater detail later in the chapter.

For more information see:

- *Pro*C/C++ Precompiler Programmer's Guide* for detailed documentation, including syntax, host variables, host variable types and example code.

Unlike locators in PL/SQL, locators in Pro*C/C++ are mapped to locator pointers which are then used to refer to the LOB or BFILE value. For the successful completion of an embedded SQL LOB statement you must provide an *allocated* input locator pointer that represents a LOB that exists in the database tablespaces or external file system *before* you execute the statement.

Once a locator pointer has been allocated, you may then SELECT a LOB locator into a LOB locator pointer variable and use that variable in an embedded SQL LOB statement to access and manipulate the LOB value. Examples provided with each embedded SQL LOB statement will illustrate this in the following sections.

The statements that can modify BLOB, CLOB, and NCLOB values are:

Table 1–14 Embedded SQL Statements that Modify BLOB, CLOB, and NCLOB values

Statement	Description
APPEND	appends a LOB value to another LOB.
COPY	copies all or a portion of a LOB into another LOB.
ERASE	erases part of a LOB, starting at a specified offset.
LOAD FROM FILE	loads BFILE data into an internal LOB at a specified offset.
TRIM	truncates a LOB.
WRITE	writes data from a buffer into a LOB at a specified offset.
WRITE APPEND	writes data from a buffer into a LOB at the end of the LOB.

The statements that read or examine LOB values are:

Table 1–15 Embedded SQL Statements that Read or Examine LOB Values

Statement	Description
DESCRIBE [CHUNKSIZE]	gets the size of the Chunk for writing.
DESCRIBE [LENGTH]	returns the length of a LOB or a BFILE.
READ	reads a specified portion of a non-null LOB or a BFILE into a buffer.

The statements that deal with temporary LOBs are:

Table 1–16 Embedded SQL Statements that Operate on Temporary LOBs

Statement	Description
CREATE TEMPORARY	creates a temporary LOB.
DESCRIBE [ISTEMPORARY]	sees if a LOB locator refers to a temporary LOB.
FREE TEMPORARY	frees a temporary LOB.

The statements specific to BFILEs are:

Table 1–17 Embedded SQL Statements that are Specific to BFILES

Statement	Description
FILE CLOSE ALL	closes all open BFILEs.
DESCRIBE [FILEEXISTS]	checks whether a BFILE exists.
DESCRIBE [DIRECTORY, FILENAME]	returns the directory alias and / or filename of a BFILE.

These statements are used for working with LOB locators:

Table 1–18 LOB Locator Embedded SQL Statements

Statement	Description
ASSIGN	assigns one LOB locator to another.
FILE SET	sets the directory alias and filename of a BFILE in a locator.

The following three statements have to do with the LOB Buffering Subsystem:

Table 1–19 LOB Buffering Subsystem Embedded SQL statements

Statement	Description
DISABLE BUFFERING	disables the use of the buffering subsystem.
ENABLE BUFFERING	uses the LOB buffering subsystem for subsequent reads and writes of LOB data.
FLUSH BUFFER	flushes changes made to the LOB buffering subsystem to the database (server)

The following statements have to do with opening and closing LOBs and BFILEs:

Table 1–20 Embedded SQL Statements for Opening and CLosing LOBs and BFILEs

Statement	Description
OPEN	opens a LOB or BFILE.
DESCRIBE [ISOPEN]	sees if a LOB or BFILE is open.
CLOSE	closes a LOB or BFILE.

Using COBOL (Pro*COBOL) to Work with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it by using embedded SQL. You can access both internal and external LOBs for read purposes, and you can also write to internal LOBs.

Embedded SQL statements allow you to access data stored in BLOBs, CLOBs, NCLOBs, and BFILEs. These statements are listed in the tables below, and are discussed in greater detail later in the chapter.

Unlike locators in PL/SQL, locators in Pro*COBOL are mapped to locator pointers which are then used to refer to the LOB or BFILE value. For the successful completion of an embedded SQL LOB statement you must provide an *allocated* input locator pointer that represents a LOB that exists in the database tablespaces or external file system *before* you execute the statement.

Once a locator pointer has been allocated, you may then SELECT a LOB locator into a LOB locator pointer variable and use that variable in an embedded SQL LOB statement to access and manipulate the LOB value. Examples provided with each embedded SQL LOB statement will illustrate this in the following sections.

In cases in which the Pro*COBOL interface does not supply the required functionality, you can call the OCI via C. We do not provide an example because such programs are operating system dependent.

For more information see:

- *Pro*COBOL Precompiler Programmer's Guide* for detailed documentation, including syntax, host variables, host variable types and example code.

The statements that can modify BLOB, CLOB, and NCLOB values are:

Table 1–21 Embedded SQL Statements that Modify BLOB, CLOB, and NCLOB values

Statement	Description
APPEND	appends a LOB value to another LOB.
COPY	copies all or a portion of a LOB into another LOB.
ERASE	erases part of a LOB, starting at a specified offset.
LOAD FROM FILE	loads BFILE data into an internal LOB at a specified offset.
TRIM	truncates a LOB.

Table 1–21 Embedded SQL Statements that Modify BLOB, CLOB, and NCLOB values

Statement	Description
WRITE	writes data from a buffer into a LOB at a specified offset.
WRITE APPEND	writes data from a buffer into a LOB at the end of the LOB.

The statements that read or examine LOB values are:

Table 1–22 Embedded SQL Statements that Read or Examine LOB Values

Statement	Description
DESCRIBE [CHUNKSIZE]	gets the size of the Chunk for writing.
DESCRIBE [LENGTH]	returns the length of a LOB or a BFILE.
READ	reads a specified portion of a non-null LOB or a BFILE into a buffer.

The statements that deal with temporary LOBs are:

Table 1–23 Embedded SQL Statements that Operate on Temporary LOBs

Statement	Description
CREATE TEMPORARY	creates a temporary LOB.
DESCRIBE [ISTEMPORARY]	sees if a LOB locator refers to a temporary LOB.
FREE TEMPORARY	frees a temporary LOB.

The statements specific to BFILEs are:

Table 1–24 Embedded SQL Statements that are Specific to BFILES

Statement	Description
FILE CLOSE ALL	closes all open BFILEs.
DESCRIBE [FILEEXISTS]	checks whether a BFILE exists.

Table 1–24 (Cont.) Embedded SQL Statements that are Specific to BFILES

Statement	Description
DESCRIBE [DIRECTORY, FILENAME]	returns the directory alias and / or filename of a BFILE.

These statements are used for working with LOB locators:

Table 1–25 LOB Locator Embedded SQL Statements

Statement	Description
ASSIGN	assigns one LOB locator to another.
FILE SET	sets the directory alias and filename of a BFILE in a locator.

The following three statements have to do with the LOB Buffering Subsystem:

Table 1–26 LOB Buffering Subsystem Embedded SQL statements

Statement	Description
DISABLE BUFFERING	disables the use of the buffering subsystem.
ENABLE BUFFERING	uses the LOB buffering subsystem for subsequent reads and writes of LOB data.
FLUSH BUFFER	flushes changes made to the LOB buffering subsystem to the database (server)

The following statements have to do with opening and closing LOBs and BFILEs:

Table 1–27 Embedded SQL Statements for Opening and CLosing LOBs and BFILEs

Statement	Description
OPEN	opens a LOB or BFILE.
DESCRIBE [ISOPEN]	sees if a LOB or BFILE is open.
CLOSE	closes a LOB or BFILE.

Using Visual Basic (OO4O) to Work with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it via the OO4O API. Specifically, you employ the `OraBlob`, `OraClob` and `OraBFile` objects. You can access both internal and external `LOB`s for read purposes, and you can also write to internal `LOB`s.

The `OraBlob`, `OraClob` interfaces in OO4O provides methods for performing operations on large objects in the database including BLOB, CLOB and NCLOB data types. The `OraBFile` interface provides methods for performing operations on BFILE data in the database. These interfaces (`OraBlob`, `OraClob`, `OraBFile`) encapsulate LOB locators, so the user does not deal with locators but instead uses the methods and properties provided to perform operations and get state information.

`OraMyBFile` refers to the locator obtained from a PL/SQL "`OUT`" parameter as a result of executing a PL/SQL procedure (either by doing an `OraDatabase.ExecuteSQL` or by using the `OraSqlStmt` object). Note that an OraConnect.`BeginTrans` has been called since the locator became invalid after the `COMMIT`.

When `OraBlob`, `OraClob` objects are retrieved as a part of a dynaset, these objects represent LOB locators of the dynaset current row. If the dynaset current row changes due to move operation, `OraBlob`, `OraClob` objects will represent LOB locator for the new current row. In order to retain the LOB locator of the `OraBlob`, `OraClob` object independent of the dynaset move operation, use the `Clone` method. This method returns the `OraBlob` and `OraClob` object. One could also use these objects as PL/SQL bind parameters. Here is an example which shows both types of usage. The functions and samples are explained in greater detail as part of the reference documentation.

```
Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraSoundClone as OraBlob,
OraMyBfile as OraBFile

OraConnection.BeginTrans
set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab order by clip_
id", ORADYN_DEFAULT)
set OraSound1 = OraDyn.Fields("Sound").value
set OraSoundClone = OraSound1

OraParameters.Add "id", 1,ORAPARAM_INPUT
OraParameters.Add "mybfile", Empty,ORAPARAM_OUTPUT
OraParameters("mybfile").ServerType = ORATYPE_BFILE
```

```
OraDatabase.ExecuteSQL ("begin  GetBFile(:id, :mybfile ") end")

Set OraMyBFile = OraParameters("mybfile").value
'Go to Next row
OraDyn.MoveNext

OraDyn.Edit
'Lets update OraSound1 data with that from the BFILE
OraSound1.CopyFromBFile  OraMyBFile
OraDyn.Update

OraDyn.MoveNext
'Go to Next row
OraDyn.Edit
'Lets update OraSound1 by appending with LOB data from 1st row represenetd by
'OraSoundClone
OraSound1.Append  OraSoundClone
OraDyn.Update

OraConnection.CommitTrans
```

In the above example `OraSound1` represents the locator for the current row in the dynaset where as `OraSoundClone` represents the locator for the 1st row. A change in the current row (say a `OraDyn.MoveNext`) will mean that `OraSound1` will actually represent locator for the 2nd row whereas `OraSoundClone` will represent the locator in the 1st row (`OraSoundClone` only refers the locator for the 1st row irrespective of any `OraDyn` row navigation).

`OraMyBFile` refers to the locator got an PL/SQL "OUT" parameter as a result of executing a PL/SQL procedure (either by doing an `OraDatabase.ExecuteSQL` or by using the `OraSqlStmt` object). Note that an `OraConnect.BeginTrans` has been called since with a database "COMMIT" the locator becomes invalid.

OO4O includes methods and properties that you can use to access data stored in `BLOB`s, `CLOB`s, `NCLOB`s, and `BFILE`s. These methods and properties are listed in the tables below, and are discussed in greater detail later in the chapter.

See Also: OO4O online help for detailed information including parameters, parameter types, return values, and example code.

The routines that can modify `BLOB`, `CLOB`, and `NCLOB` values are:

Table 1–28 OO4O Methods That Modify BLOB, CLOB, and NCLOB Values

Function/Procedure	Description
`OraBlob.Append`, `OraClob.Append`	appends `LOB` value to another `LOB`.
`OraBlob.Copy`, `OraClob.Copy`	copies a portion of a LOB into another `LOB`.
`OraBlob.Erase`, `OraClob.Erase`	erases part of a `LOB`, starting at a specified offset.
`OraBlob.CopyFromBFile`, `OraClob.CopyFromBFile`	loads `BFILE` data into an internal LOB.
`OraBlob.Trim`, `OraClob.Trim`	truncates a `LOB`.
`OraBlob.CopyFromFile`, `OraClob.CopyFromFile`	writes data from a file to a LOB
`OraBlob.Write`, `OraClob.Write`	writes data from a file to a LOB

The routines that read or examine `LOB` values are:

Table 1–29 Oo4o Methods that Read or Examine LOB Values

Function/Procedure	Description
OraBlob.Read, OraClob.Read, OraBFile.Read	reads a specified portion of a non-null `LOB into a buffer`
OraBlob.CopyToFile, OraClob.CopyToFile	reads a specified portion of a non-null `LOB` to a file.

The following methods have to do with opening and closing `LOB`s:

Table 1–30 OO4O Methods for Operating on BFILEs

Methods	Description
`OraBFile.Open`	`opens BFILE`.

Table 1–30 OO4O Methods for Operating on BFILEs

Methods	Description
`OraBFile.Close`	`closes BFILE.`

The following methods have to do with `LOB`-buffering:

Table 1–31 0040 LOB-Buffering methods

Function/Procedure	Description
`OraBlob.FlushBuffer, OraClob.FlushBuffer`	flushes changes made to the `LOB` buffering subsystem to the database (sever)
`OraBlob.EnableBufferin g`	Enables buffering of LOB operations
`OraClob.EnableBufferin g`	
`OraBlob.DisableBufferi ng`	Disables buffering of LOB operations
`OraClob.DisableBufferi ng`	

Table 1–32 0040 LOB- properties

Property	Description
`IsNull (Read)`	indicates when a LOB is Null
PollingAmount(Read / Write)	Get / Set the total amount for Read / Write polling operation
Offset(Read / Write)	Get / Set the offset for Read / Write operation. By default, it is set to 1.
Status(Read)	Returns the polling status.Possible values are ORALOB_NEED_DATA There is more data to be read / written ORALOB_NO_DATA There is no more data to be read / written ORALOB_SUCCESS LOB data read / written successfully
Size(Read)	Returns the length of the LOB data

Methods specific to BFILEs are:

Table 1–33 0040 Read-Only methods that are Specific to BFILES

Methods	Description
OraBFile.Close	closes an open BFILE.
OraBFile.CloseAll	closes all open BFILEs.
OraBFile.Open	opens a BFILE.
OraBFile.IsOpen	determine if a BFILE is open

Table 1–34 OO4O Properties that are Specific to BFILES

Properties	Description
OraBFile.DirectoryName	gets / Sets the server side directory alias name.
OraBFile.FileName(Read/Write)	gets / Sets the server side filename.
OraBFile.Exists	checks whether a BFILE exists.

Using Java (JDBC) to Work with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of an internal LOB in Java by means of the JDBC API via the Oracle.sql.BLOB and Oracle.sql.CLOB objects. These objects also implement the java.sql.Blob and java.sql.Clob interfaces according to the JDBC 2.0 specification. With this implementation, an Oracle.sql.BLOB can be used wherever a java.sql.Blob is expected and an Oracle.sql.CLOB can be used wherever a java.sql.Clob is expected.

The JDBC interface will let you access both internal and external LOBs for read purposes, and you can also write to internal LOBs.

The BLOB and CLOB classes in JDBC provide methods for performing operations on large objects in the database including BLOB, CLOB and NCLOB data types. The BFILE class provides methods for performing operations on BFILE data in the database. These classes (BLOB, CLOB, BFILE) encapsulate LOB locators, so the user does not deal with locators but instead uses the methods and properties provided to perform operations and get state information. Any of Oracle's LOB functionality not provided by these classes can be accessed by a call to the DBMS_LOB PL/SQL package. This technique is used repeatedly in the examples throughout the book.

You can get a reference to any of the above LOBs either as a column of an OracleResultSet or as an "OUT" type PL/SQL parameter from an OraclePreparedStatement. When BLOB and CLOB objects are retrieved as a part of aN OracleResultSet, these objects represent LOB locators of the currently selected row. If the current row changes due to a move operation (for example, rset.next()), the retrieved locator still refers to the original LOB row. In order to retrieve the locator for the most current row, you must call getXXXX() on the OracleResultSet each time a move operation is made (where XXXX is a BLOB, CLOB or BFILE).

For more information see:

- *Oracle8i JDBC Developer's Guide and Reference* for detailed documentation, including parameters, parameter types, return values, and example code.

`oracle.sql.BLOB` methods for modifying values:

Table 1–35 oracle.sql.BLOB Methods for Modifying Values

Function/Procedure	Description
`int putBytes(long, byte[])`	inserts the byte array into the `LOB`, starting at the given offset

`oracle.sql.BLOB` methods for reading or examining values:

Table 1–36 oracle.sql.BLOB Methods for Reading or Examining Values

Function/Procedure	Description
`byte[] getBytes(long, int)`	gets the contents of the `LOB` as an array of bytes, given an offset
`long position(byte[], long)`	finds the given byte array within the `LOB`, given an offset
`long position(oracle.jdbc2.Blob, long)`	finds the given `BLOB` within the `LOB`
`public boolean equals(java.lang.Object)`	compares this `LOB` with another
`public long length()`	returns the length of the `LOB`
`public int getChunkSize()`	returns the `ChunkSize` of the `LOB`

`oracle.sql.BLOB` `LOB`-buffering methods and properties:

Table 1–37 oracle.sql.BLOB LOB-Buffering Methods and Properties

Function/Procedure	Description
`public java.io.InputStream getBinaryStream())`	streams the `LOB` as a binary stream
`public java.io.OutputStream getBinaryOutputStream()`	writes to `LOB` as a binary stream

`oracle.sql.CLOB` methods for modifying values

Table 1–38 oracle.sql.CLOB Methods for Modifying Values

Function/Procedure	Description
`int putString(long, java.lang.String)`	inserts the string into the LOB, starting at the given offset
`int putChars(long, char[])`	inserts the character array into the LOB, starting at the given offset

`oracle.sql.CLOB` methods for reading or examining values:

Table 1–39 oracle.sql.CLOB Methods for Reading or Examining Values

Function/Procedure	Description
`byte[] getBytes()`	gets the contents of the LOB as an array of bytes
`java.lang.String getSubString(long, int)`	returns a substring of the LOB as a string
`int getChars(long, int, char[])`	reads a subset of the LOB into a character array
`long position(java.lang.String, long)`	finds the given String within the LOB, given an offset
`long position(oracle.jdbc2.Clob, long)`	finds the given CLOB within the LOB, given an offset
`boolean equals(java.lang.Object)`	compares this LOB with another
`long length()`	returns the length of the LOB
`int getChunkSize()`	returns the ChunkSize of the LOB

`oracle.sql.CLOB` LOB-buffering methods and properties:

Table 1–40 oracle.sql.CLOB LOB-Buffering Methods and Properties

Function/Procedure	Description
`java.io.InputStream getAsciiStream()`	streams the LOB as an ASCII stream
`java.io.InputStream getStream()`	streams the LOB as a byte array
`java.io.OutputStream getAsciiOutputStream()`	writes to the LOB as an ASCII stream
`java.io.Reader getCharacterStream()`	streams the LOB as a character stream
`java.io.Writer getCharacterOutputStream()`	writes to LOB as a character stream

`oracle.sql.BFILE` methods for reading or examining values

Table 1–41 oracle.sql.BFILE Methods for reading or Examining Values

Function/Procedure	Description
`byte[] getBytes()`	`gets the contents of the LOB as an array of bytes`
`byte[] getBytes(long, int)`	`gets the contents of the LOB as an array of bytes, given an offset`
`int getBytes(long, int, byte[])`	`reads a subset of the LOB into a byte array`
`long position(oracle.sql.BFILE , long)`	`finds the given BFILE contents within the LOB, given an offset`
`long position(byte[], long)`	`finds the given byte array within the LOB, given an offset`
`boolean equals(java.lang.Object)`	compares this LOB with another
`long length()`	returns the length of the LOB
`boolean fileExists()`	checks if the OS file referenced by this BFILE exists
`public void openFile()`	opens the OS file referenced by this BFILE

Table 1–41 oracle.sql.BFILE Methods for reading or Examining Values

Function/Procedure	Description
`public void closeFile()`	closes the OS file referenced by this BFILE
`public boolean isFileOpen()`	checks if this BFILE is already open
`public java.lang.String getDirAlias()`	gets the directory alias for this BFILE
`public java.lang.String getName()`	gets the file name referenced by this BFILE

`oracle.sql.BFILE` methods for LOB-buffering methods and properties:

Table 1–42 oracle.sql.CLOB Methods for Modifying Values

Function/Procedure	Description
`public java.io.InputStream getBinaryStream()`	streams the LOB as a binary stream
`public java.io.InputStream getStream()`	streams the LOB as a byte array

An Example Application

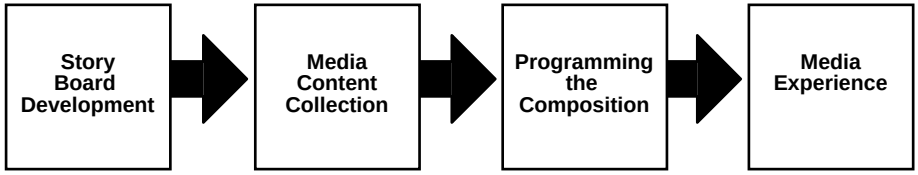
Oracle8 supports LOBs, *large objects* which can hold up to 4 gigabytes of binary or character data. What does this mean for you, the application developer?

Consider the following hypothetical application:

The Multimedia Content-Collection System

Multimedia data is used in an increasing variety of media channels — film, television, webpages, and CD-ROM being the most prevalent. The media experiences having to do with these different channels vary in many respects (interactivity, physical environment, the structure of information, to name a few). Yet despite these differences, there is often considerable similarity in the multimedia authoring process, especially with regard to assembling content.

Figure 1–1 The Multimedia Authoring Process



For instance, a television station that creates complex documentaries, an advertising agency that produces advertisements for television, and a software production house that specializes in interactive games for the web could all make good use of a database management system for collecting and organizing the multimedia data. Presumably, they each have sophisticated editing software for composing these elements into their specific products, but the complexity of such projects creates a need for a pre-composition application for organizing the multimedia elements into appropriate groups.

Taking our lead from movie-making, our hypothetical application for collecting content uses the *clip* as its basic unit of organization. Any clip is able to include one or more of the following media types:

- character text (e.g.,storyboard, transcript, subtitles,),
- images (e.g., photographs, video frames),
- line drawings (e.g., maps),

- audio (e.g., sound-effects, music, interviews)

Since this is a pre-editing application, the precise relationship of elements within a clip (such as the synchronization of voice-over audio with a photograph) and between clips (such as the sequence of clips) is not defined.

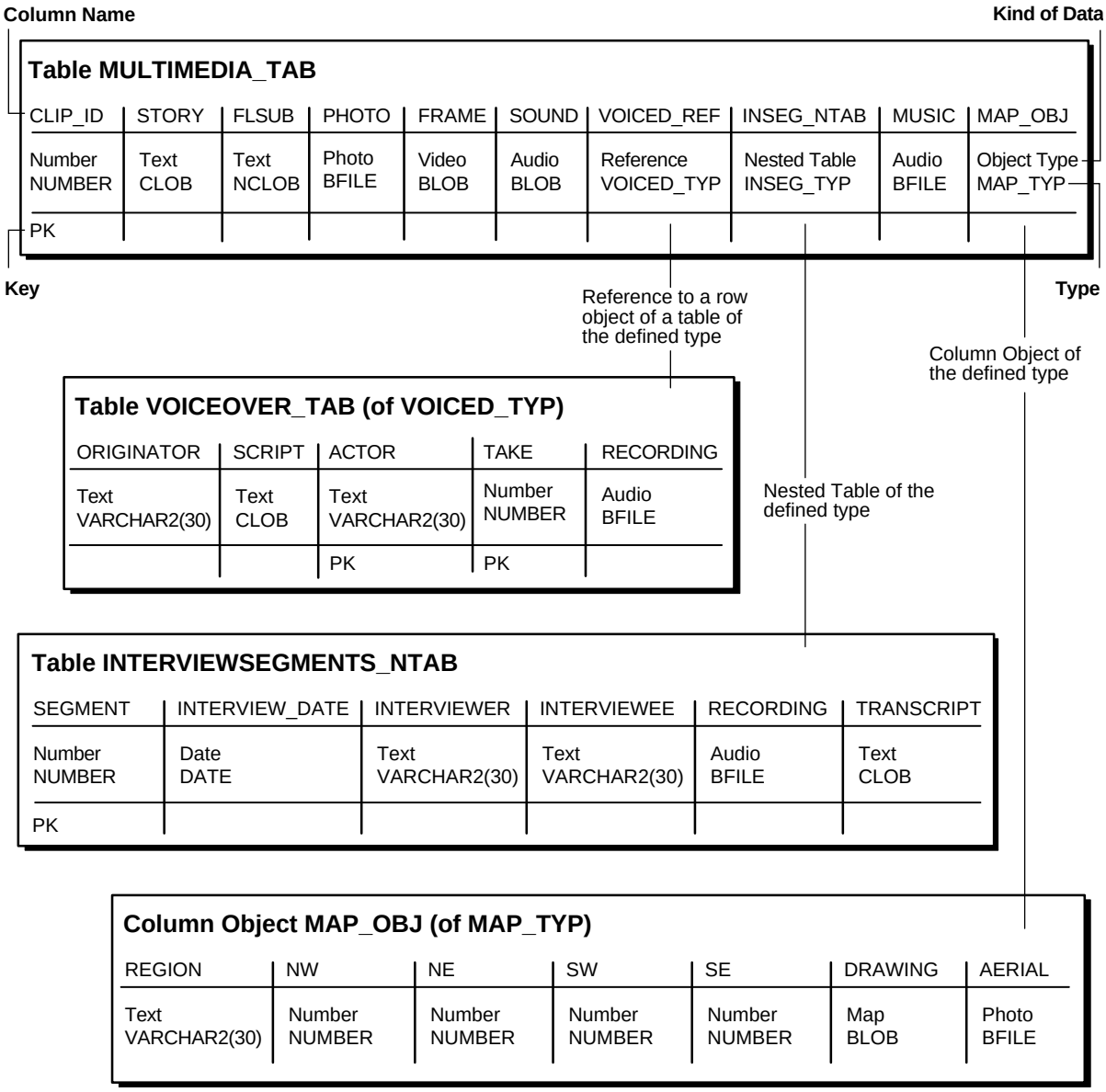
The application should allow multiple editors working simultaneously to store, retrieve and manipulate the different kinds of multimedia data. We assume that some material is gathered from in-house databases. At the same time, it should also be possible to purchase and download data from professional services.

Note: The Example is Only An Example

Our mission in this chapter is not to create this real-life application, but to describe everything you need to know about working with LOBs. Consequently, we only implement the application sufficiently to demonstrate the technology. For example, we deal with only a limited number of multimedia types. We make no attempt to create the client-side applications for manipulating the LOBs. And we do not deal with deployment issues such as, the fact that you should implement disk striping of LOB files, if possible, for best performance.

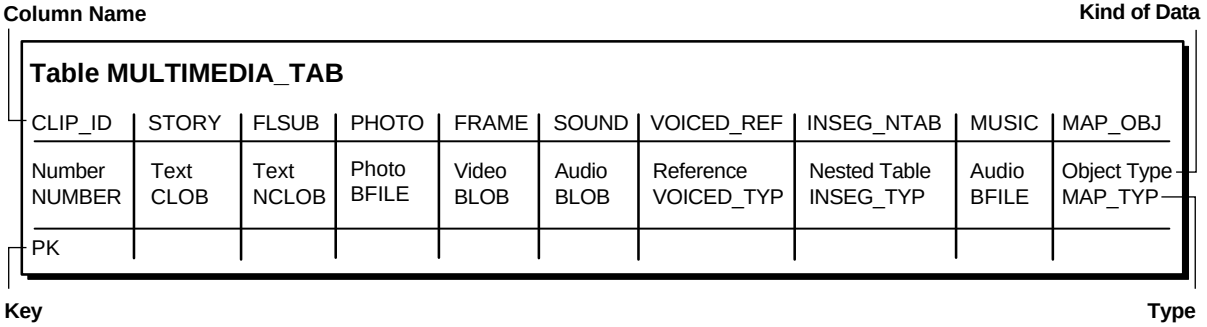
Applying an Object-Relational Design to the Application

Figure 1–2 Schema Plan for Table MULTIMEDIA_TAB



The Structure of the Multimedia_tab Table

Figure 1–3 Schema Plan for Table MULTIMEDIA_TAB



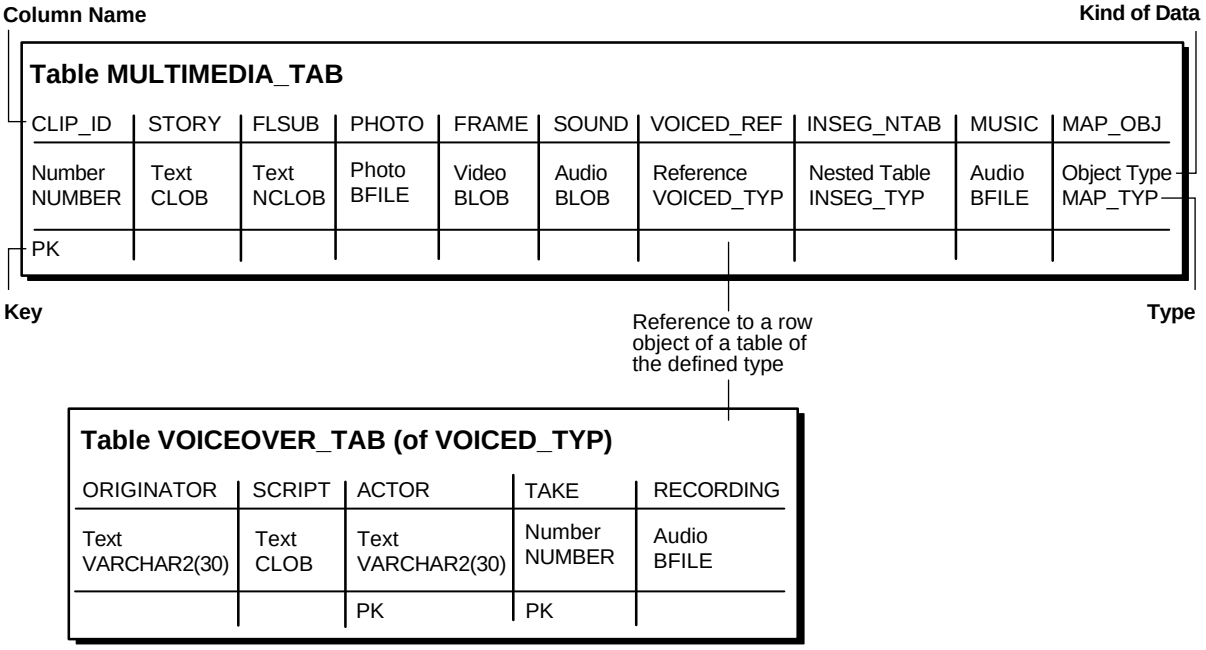
- CLIP_ID: Every row (clip object) must have a number which identifies the clip. This number is generated by the Oracle number SEQUENCER as a matter of convenience, and has nothing to do with the eventual ordering of the clip.
- STORY: The application design requires that every clip must also have text, that is a storyboard, that describes the clip. Since we do not wish to limit the length of this text, or restrict its format, we use a CLOB datatype.
- FLSUB: Subtitles have many uses — for closed-captioning, as titles, as overlays that draw attention, and so on. A full-fledged application would have columns for each of these kinds of data but we are considering only the specialized case of a foreign language subtitle, for which we use the NCLOB datatype.
- PHOTO: Photographs are clearly a staple of multimedia products. We assume there is a library of photographs stored in the PhotoLib_tab archive. Since a large database of this kind would be stored on tertiary storage that was periodically updated, the column for photographs makes use of the BFILE datatype.
- FRAME: It is often necessary to extract elements from dynamic media sources for further processing For instance, VRML game-builders and animation cartoonists are often interested in individual cells. Our application takes up the need to subject film / video to frame-by-frame analysis such as was performed on the film of the Kennedy assassination. While it is assumed that the source is on persistent storage, our application allows for an individual frame to be stored as a BLOB.

- SOUND: The table includes a column for sound-effects in the form of a BLOB.
- VOICED_REF: This column allows for a reference to a specific row in a table which must be of the type Voiced_typ. In our application, this is a reference to a row in the table VoiceOver_tab whose purpose is to store audio recordings for use as voice-over commentaries. For instance, these might be readings by actors of words spoken or written by people for whom no audio recording can be made, perhaps because they are no longer alive, or because they spoke or wrote in a foreign language.

  This structure offers the application builder a number of different strategies from those discussed thus far. Instead of loading material into the row from an archival source, an application can simply reference the data. This means that the same data can be referenced from other tables within the application, or by other applications. The single stipulation is that the reference can only be to tables of the same type. Put another way: the reference, Voiced_ref, can refer to row objects in any table which conforms to the type, Voiced_typ.

  Note that Voiced_typ combines the use of two LOB datatypes: a CLOB to store the script which the actor reads, and a BFILE for the audio recordings.

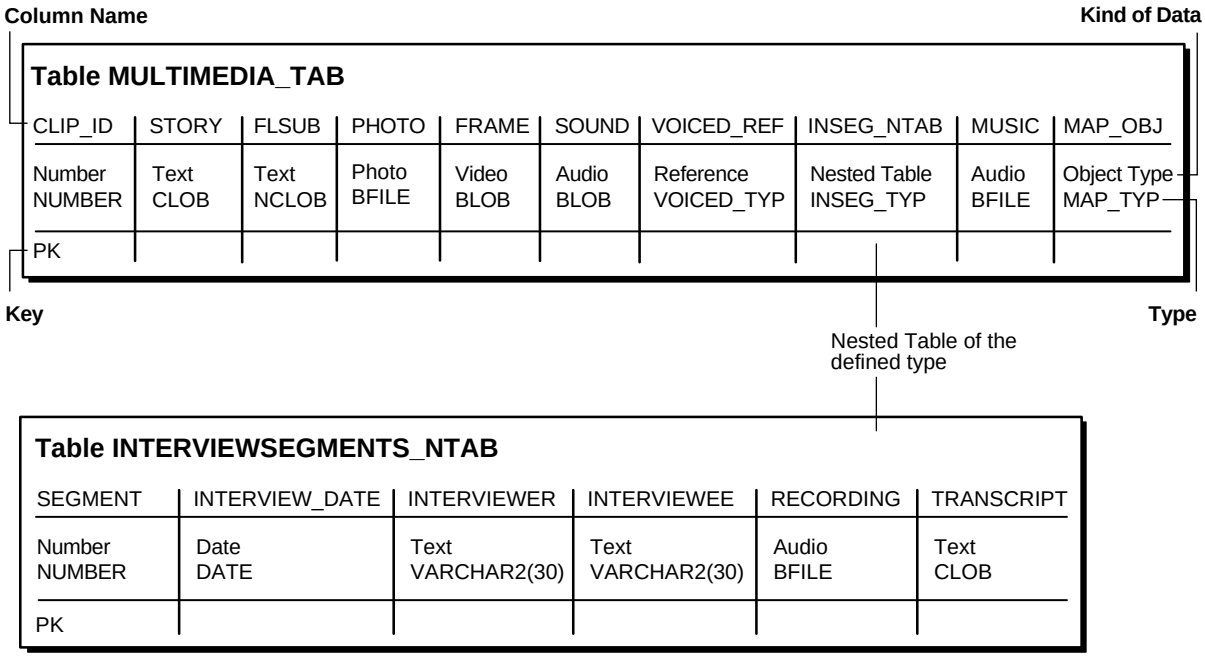
Figure 1–4 Schema Design for Inclusion of VOICED_REF Reference



- `INSEG_NTAB`: While it is not possible to store a Varray of LOBs, application builders are able to store a variable number of multimedia elements in a single row by means of nested tables. In the case of our application, a nested table `InSeg_ntab` of the predefined type `InSeg_typ` can be used to store zero, one or many interview segments in a given clip. So, for instance, a hypothetical user could use this facility to collect together one or more interview segments having to do with the same theme that occurred at different times.

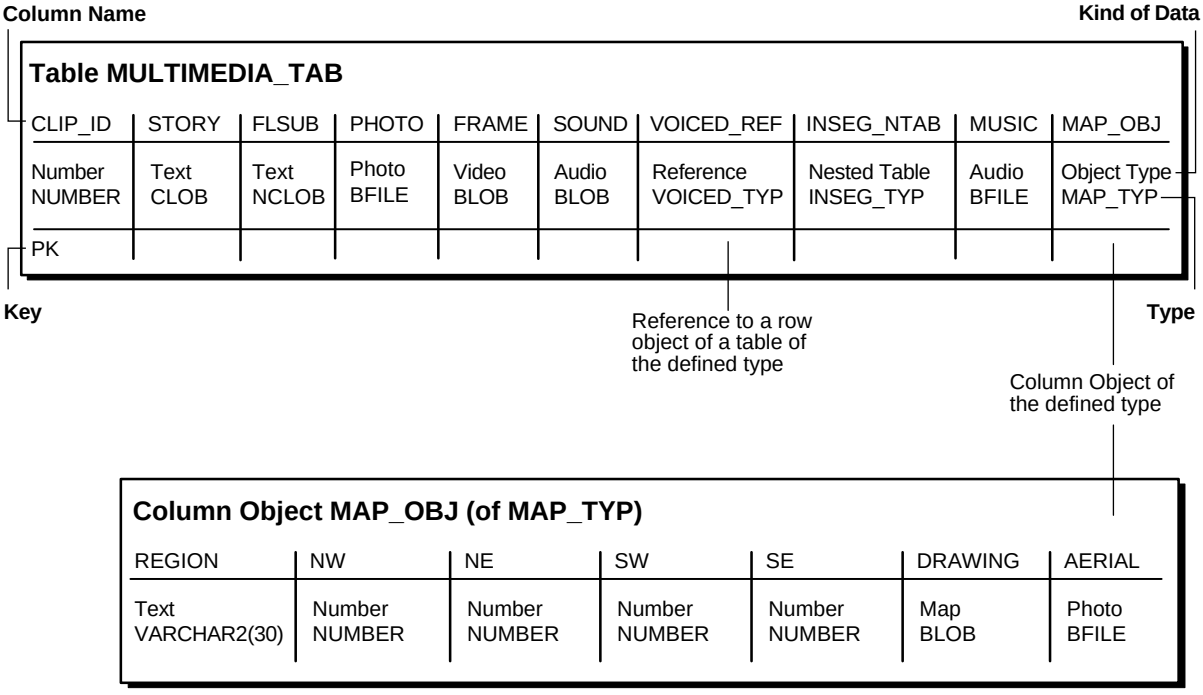
  In this case, the nested table makes use of two LOB datatypes — a BFILE to store the audio recording of the interview, and a CLOB for transcript. Since such segments might be of great length, it is important to keep in mind that LOBs cannot be more than 4 gigabytes in size.

Figure 1–5 Schema Design for Inclusion of a Nested Table INTERVIEWSEGMENTS_TAB



- `MUSIC`: The ability to handle music must be one of the basic requirements of any multimedia database management system. In this case, the BFILE datatype is used to store the audio as an operating system file.
- `MAP_OBJ`: Multimedia applications must be be able to handle many different kinds of line art — cartoons, diagrams, and fine art, to name a few. In our application, provision is made for a clip to contain a map as a column object, `MAP_OBJ`, of the object type `MAP_TYP`. In this case, the object is contained by value, being embedded in the row. As defined in our application, `MAP_TYP` has only one LOB in its structure — a BLOB for the drawing itself. However, as in the case of the types underlying REFs and nested tables, there is no restriction on the number of LOBs that an object type may contain.

Figure 1–6 Schema Design for Inclusion of a Column Object MAP_OBJ



The Most Basic Operation: Getting and Using the LOB Locator

LOB Value and Locators

Inline storage of the LOB value

Data stored in a LOB is termed the LOB's *value*. The value of an internal LOB may or may not be stored inline with the other row data. If the internal LOB value is less than approximately 4000 bytes, then the value is stored inline; otherwise it is stored outside the row. Since LOBs are intended to be large objects, inline storage will only be relevant if your application mixes small and large LOBs.

As mentioned below ("ENABLE | DISABLE STORAGE IN ROW" on page 1-45), the LOB value is automatically moved out of the row once it extends beyond approximately 4000 bytes.

LOB locators

Regardless of where the value of the internal LOB is stored, a *locator* is stored in the row. You can think of a LOB locator as a pointer to the actual location of the LOB value. A *LOB locator* is a locator to an internal LOB while a *BFILE locator* is a locator to an external LOB. When the term *locator* is used without an identifying prefix term, it refers to both LOB locators and BFILE locators.

Internal LOB Locators

For internal LOBs, the LOB column stores a locator to the LOB's value which is stored in a database tablespace. Each LOB column / attribute for a given row has its own distinct LOB locator and copy of the LOB value stored in the database tablespace.

LOB Locator Operations

Setting the LOB Column/Attribute to contain a locator

Before you can start writing data to an internal LOB, the LOB column / attribute must be made non-null, that is, it must contain a locator. Similarly, before you can start accessing the BFILE value, the BFILE column / attribute must be made non-null.

- For internal LOBs, you can accomplish this by initializing the internal LOB to empty in an INSERT / UPDATE statement using the functions EMPTY_BLOB() for BLOBs or EMPTY_CLOB() for CLOBs and NCLOBs.

For more information see:

- "INSERT a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()" on page 3-26

- For external LOBs, you can initialize the BFILE column to point to an external file by using the BFILENAME() function.

For more information see:

- "INSERT a Row by means of BFILENAME()" on page 5-22.

Invoking the EMPTY_BLOB() or EMPTY_CLOB() function in and of itself does not raise an exception. However, using a LOB locator that was set to empty to access or manipulate the LOB value in any PL/SQL DBMS_LOB or OCI routine will raise an exception. Valid places where empty LOB locators may be used include the VALUES clause of an INSERT statement and the SET clause of an UPDATE statement.

The following INSERT statement

- populates *story* with the character string 'JFK interview',
- sets *flsub*, *frame* and *sound* to an empty value,
- sets *photo* to NULL, and
- initializes *music* to point to the file 'JFK_interview' located under the logical directory 'AUDIO_DIR' (see the CREATE DIRECTORY command in the *Oracle8i Reference*. Character strings are inserted using the default character set for the instance.

```
INSERT INTO Multimedia_tab VALUES (101, 'JFK interview', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL,
    BFILENAME('AUDIO_DIR', 'JFK_interview'), NULL);
```

Similarly, the LOB attributes for the *Map_typ* column in Multimedia_tab can be initialized to NULL or set to empty as shown below. Note that you cannot initialize a LOB object attribute with a literal.

```
INSERT INTO Multimedia_tab
  VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(), NULL, EMPTY_BLOB(),
          EMPTY_BLOB(), NULL, NULL, NULL,
          Map_typ('Moon Mountain', 23, 34, 45, 56, EMPTY_BLOB(), NULL);
```

Accessing a LOB through a locator

SELECTing a LOB Performing a SELECT on a LOB returns the locator instead of the LOB value. In the following PL/SQL fragment you select the LOB locator for *story* and place it in the PL/SQL locator variable *Image1* defined in the program block. When you use PL/SQL DBMS_LOB functions to manipulate the LOB value, you refer to the LOB using the locator.

```
DECLARE
    Image1       BLOB;
    ImageNum     INTEGER := 101;
BEGIN
    SELECT story INTO Image1 FROM Multimedia_tab
        WHERE clip_id = ImageNum;
    DBMS_OUTPUT.PUT_LINE('Size of the Image is: ' ||
        DBMS_LOB.GETLENGTH(Image1));
    /* more LOB routines */
END;
```

In the case of OCI, locators are mapped to locator pointers which are used to manipulate the LOB value. As mentioned before, the OCI LOB interface is described briefly in "Support Libraries" on page 1-309, and more extensively in the *Oracle Call Interface Programmer's Guide*.

LOB Locators and Transaction Boundaries

If you begin a transaction and then select a locator, the locator contains the transaction ID. Note that you can implicitly be in a transaction without explicitly beginning one. For example, SELECT ... FOR UPDATE implicitly begins a transaction. In such a case, the locator will contain a transaction ID. By contrast, if you select a locator outside of a transaction, the locator does not contain a transaction ID. Note that a transaction ID will not be assigned until the first DML statement executes. Therefore, locators that are selected out prior to such a DML statement will not contain a transaction ID.

You can always read the LOB data using the locator irrespective of whether the locator contains a transaction id. However, if the locator contains a transaction id, you cannot write to the LOB outside of that particular transaction. If the locator does not contain a transaction id, you can write to the LOB after beginning a transaction either explicitly or implicitly. We can show the relationship between transactions and locators by considering a few examples. However, if the locator contains a transaction id and the transaction is serializable, you cannot read or write outside of

that particular transaction. If the transaction is non-serializable, you can read, but not write outside of that transaction. The following examples show the relationship between locators and non-serializable transactions

Select the Locator with No Current Transaction

Case 1:

1. `Select the locator with no current transaction.`

   At this point, the locator does not contain a transaction id.
2. `Begin the transaction.`
3. `Use the locator to read data from the LOB.`
4. `Commit or rollback the transaction.`
5. `Use the locator to read data from the LOB.`
6. `Begin a transaction.`

   The locator does not contain a transaction id.
7. `Use the locator to write data to the LOB.`

   This operation is valid because the locator did not contain a transaction id prior to the write. After this call, the locator contains a transaction id.

Case 2:

1. `Select the locator with no current transaction.`

   At this point, the locator does not contain a transaction id.
2. `Begin the transaction.`

   The locator does not contain a transaction id.
3. `Use the locator to read data from the LOB.`

   The locator does not contain a transaction id.
4. `Use the locator to write data to the LOB`

   This operation is valid because the locator did not contain a transaction id prior to the write. After this call, the locator contains a transaction id. You can continue to read from and / or write to the LOB.
5. `Commit or rollback the transaction.`

The locator continues to contain the transaction id.

6. `Use the locator to read data from the LOB.`

   This is a valid operation.

7. `Begin a transaction.`

   The locator already contains the previous transaction's id.

8. `Use the locator to write data to the LOB.`

   This write operation will fail because the locator does not contain the transaction id that matches the current transaction.

Select the Locator within a Transaction

Case 3:

1. `Select the locator within a transaction.`

   At this point, the locator contains the transaction id.

2. `Begin the transaction.`

   The locator contains the previous transaction's id.

3. `Use the locator to read data from the LOB.`

   This operation is valid even though the transaction id in the locator does not match the current transaction.

   For more information on the LOB value that is Read see:

   - "Read-Consistent Locators" on page 2-2

   ---

4. `Use the locator to write data to the LOB`

   This operation fails because the transaction id in the locator does not match the current transaction.

Case 4:

1. `Begin a transaction.`
2. `Select the locator.`

The locator contains the transaction id because it was selected within a transaction.

3. `Use the locator to read from and/or write to the LOB.`

   These operations are valid.

4. `Commit or rollback the transaction.`

   The locator continues to contain the transaction id.

5. `Use the locator to read data from the LOB.`

   This operation is valid even though there's a transaction id in the locator and the transaction was previously committed or rolled back.

For more information on the LOB value that is Read see:

- "Read-Consistent Locators" on page 2-2

---

6. `Use the locator to write data to the LOB`

   This operation fails because the transaction id in the locator is for a transaction that was previously committed or rolled back.

Open, Close and IsOpen Interfaces for Internal LOBs

These interfaces let you open and close an internal LOB and test whether an internal LOB is already open.

It is not mandatory that you wrap all LOB operations inside the `Open`/`Close` APIs. The addition of this feature will not impact already-existing applications that write to LOBs without first opening them, since these calls did not exist in 8.0.

It is important to note that openness is associated with the LOB, not the locator. The locator does not save any information as to whether the LOB to which it refers is open.

Open and Close with Extensible Indexes

If you do not wrap LOB operations inside an `Open`/`Close` call, each modification to the LOB will implicitly open and close the LOB thereby firing any triggers on an extensible index. Note that in this case, any extensible indexes on the LOB will become updated as soon as LOB modifications are made. Therefore, extensible LOB indexes are always valid and may be used at any time. By contrast, if you wrap your LOB operations inside the `Open`/`Close` operations, triggers will not be fired

for each LOB modification. Instead, the trigger on extensible indexes will be fired at the Close call. For example, you might design your application so that extensible indexes are not be updated until you call Close. However, this means that any extensible indexes on the LOB will not be valid in-between the Open/Close calls.

Note that the definition of a 'transaction' within which an open LOB value must be closed is one of the following:

- between 'DML statements that start a transaction (including SELECT ... FOR UPDATE)' and COMMIT
- within an autonomous transaction block

A LOB opened when there is no transaction must be closed before the end of the session. If there are still open lobs at the end of the session, the openness will be discarded and no triggers on extensible indexes will be fired.

Errors

It is an error to commit the transaction before closing all opened LOBs that were opened by the transaction. When the error is returned, the openness of the open LOBs is discarded. At this point, the user must decide whether to close all the LOBs and reissue the call to commit, or rollback the transaction. Note that the changes to the LOB are not discarded if the COMMIT returns an error. At transaction rollback time, the openness of all open LOBs that are still open for that transaction will be discarded. Discarding the openness means that the LOBs won't be closed, thereby firing the triggers on extensible indexes.

It is also an error to open/close the same LOB twice either with different locators or with the same locator.

Example 1

```
DECLARE
   Lob_loc1 CLOB;
   Lob_loc2 CLOB;
   Buffer   VARCHAR2(32767);
   Amount   BINARY_INTEGER := 32767;
   Position INTEGER := 1;
BEGIN
   /* Select a LOB: */
   SELECT Story INTO Lob_loc1 FROM Multimedia_tab WHERE Clip_ID = 1;

   /* The following statement opens the LOB outside of a transaction
      so it must be closed before the session ends: */
   DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
```

```
   /* The following statement begins a transaction. Note that Lob_loc1 and
      Lob_loc2 point to the same LOB:  */
   SELECT Story INTO Lob_loc2 FROM Multimedia_tab WHERE Clip_ID = 1 for update;
   /* The following LOB open operation is allowed since this lob has
      not been opened in this transaction:  */
   DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READWRITE);
   /* Fill the buffer with data to write to the LOB */
   buffer := 'A good story';
   Amount := 12;
   /* Write the buffer to the LOB: */
   DBMS_LOB.WRITE(Lob_loc2, Amount, Position, Buffer);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Lob_loc2);
   /* The COMMIT ends the transaction. It is allowed because all LOBs
      opened in the transaction were closed. */
   COMMIT;
   /* The the following statement closes the LOB that was opened
       before the transaction started: */
   DBMS_LOB.CLOSE(Lob_loc1);
END;
```

Example 2:

```
DECLARE
   Lob_loc CLOB;
BEGIN
   /* Note that the FOR UPDATE clause starts a transaction: */
   SELECT Story INTO Lob_loc FROM Multimedia_tab WHERE Clip_ID = 1 for update;
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /* COMMIT returns an error because there is still an open LOB associated
   with this transaction: */
   COMMIT;
END;
```

Indexing a LOB Column

You cannot build B-tree or bitmap indexes on a LOB column. However, depending on your application and its usage of the LOB column, you might be able to improve the performance of queries by building indexes specifically attuned to your domain. Oracle's extensibility interfaces allow for Extensible Indexing, a framework for implementing such domain specific indexes.

For more information regarding building domain specific indexes, see: *Oracle8i Data Cartridge Developer's Guide.*

Depending on the nature of the contents of the LOB column, one of the Oracle intermedia options could also be used for building indexes. For example, if a text document is stored in a CLOB column, you can build a text index (provided by Oracle) to speed up the performance of text-based queries over the CLOB column.

For more information regarding Oracle's intermedia options, see: *Oracle8i interMedia Audio, Image, and Video User's Guide and Reference* and *Oracle8i Context Cartridge Reference.*

2

Advanced Topics

The material in this chapter is a supplement and elaboration of the use cases described in the following chapters.You will probably find the topics discussed here to be more relevant once you have explored the use cases.

- Read-Consistent Locators
- LOBs in the Object Cache
- LOB Buffering Subsystem
- User Guidelines for Best Performance Practices
- Working with Varying-Width Character Data

Read-Consistent Locators

Oracle provides the same read consistency mechanisms for LOBs as for all other database reads and updates of scalar quantities (refer to *Oracle8i Concepts* for general information about read consistency). However, read consistency has some special applications to LOB locators that need to be understood.

A SELECTed locator, regardless of the existence of the FOR UPDATE clause, becomes a *read consistent locator*, and remains a read consistent locator until the LOB value is updated through that locator. A read consistent locator contains the snapshot environment as of the point in time of the SELECT.

This has some complex implications. Let us say that you have created a read consistent locator (L1) by way of a SELECT operation. In reading the value of the internal LOB through L1, the LOB is read as of the point in time of the SELECT statement even if the SELECT statement includes a FOR UPDATE. Further, if the LOB value is updated through a different locator (L2) in the same transaction, L1 does not see L2's updates. In addition, L1 will not see committed updates made to the LOB through *another* transaction.

Furthermore, if the read consistent locator L1 is copied to another locator L2 (for example, by a PL/SQL assignment of two locator variables — L2:= L1), then L2 becomes a read consistent locator along with L1 and any data read is read *as of the point in time of the SELECT for L1*.

Clearly you can utilize the existence of multiple locators to access different transformations of the LOB value. However, in taking this course, you must be careful to keep track of the different values accessed by different locators. The following code demonstrates the relationship between read-consistency and updating in a simple example.

Using `Multimedia_tab` as defined previously and PL/SQL, three CLOBs are created as potential locators: `clob_selected`, `clob_updated` and `clob_copied`.

- At the time of the first SELECT INTO (at t1), the value in *story* is associated with the locator *clob_selected.*
- In the second operation (at t2), the value in *story* is associated with the locator *clob_updated.* Since there has been no change in the value of *story* between t1 and t2, both *clob_selected* and *clob_updated* are read consistent locators that effectively have the same value even though they reflect snapshots taken at different moments in time.

- The third operation (at t3) copies the value in *clob_selected* to *clob_copied*. At this juncture, all three locators see the same value. The example demonstrates this with a series of DBMS_LOB.READ() calls.
- At time t4, the program utilizes DBMS_LOB.WRITE() to alter the value in *clob_updated*, and a DBMS_LOB.READ() reveals a new value.
- However, a DBMS_LOB.READ() of the value through *clob_selected* (at t5) reveals that it is a read consistent locator, continuing to refer to the same value as of the time of its SELECT.
- Likewise, a DBMS_LOB.READ() of the value through *clob_copied* (at t6) reveals that it is a read consistent locator, continuing to refer to the same value as *clob_selected*.

Example of a Read Consistent Locator

```
INSERT INTO Multimedia_tab VALUES (1, 'abcd', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

COMMIT;

DECLARE
  num_var           INTEGER;
  clob_selected     CLOB;
  clob_updated      CLOB;
  clob_copied       CLOB;
  read_amount       INTEGER;
  read_offset       INTEGER;
  write_amount      INTEGER;
  write_offset      INTEGER;
  buffer            VARCHAR2(20);

BEGIN
  -- At time t1:
  SELECT story INTO clob_selected
     FROM Multimedia_tab
     WHERE clip_id = 1;

  -- At time t2:
  SELECT story INTO clob_updated
     FROM Multimedia_tab
     WHERE clip_id = 1
     FOR UPDATE;
```

```
-- At time t3:
clob_copied := clob_selected;
-- After the assignment, both the clob_copied and the
-- clob_selected have the same snapshot as of the point in time
-- of the SELECT into clob_selected

-- Reading from the clob_selected and the clob_copied will
-- return the same LOB value. clob_updated also sees the same
-- LOB value as of its select:
read_amount := 10;
read_offset := 1;
dbms_lob.read(clob_selected, read_amount, read_offset,
     buffer);
dbms_output.put_line('clob_selected value: ' || buffer);
-- Produces the output 'abcd'

read_amount := 10;
dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
dbms_output.put_line('clob_copied value: ' || buffer);
-- Produces the output 'abcd'

read_amount := 10;
dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
dbms_output.put_line('clob_updated value: ' || buffer);
-- Produces the output 'abcd'

-- At time t4:
write_amount := 3;
write_offset := 5;
buffer := 'efg';
dbms_lob.write(clob_updated, write_amount, write_offset,
     buffer);

read_amount := 10;
dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
dbms_output.put_line('clob_updated value: ' || buffer);
-- Produces the output 'abcdefg'

-- At time t5:
read_amount := 10;
dbms_lob.read(clob_selected, read_amount, read_offset,
     buffer);
dbms_output.put_line('clob_selected value: ' || buffer);
-- Produces the output 'abcd'
```

```
  -- At time t6:
  read_amount := 10;
  dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_copied value: ' || buffer);
  -- Produces the output 'abcd'
END;
/
```

Updated locators

When you update the value of the internal LOB through the LOB locator (L1), L1 (that is, the *locator* itself) is updated to contain the current snapshot environment *as of the point in time after the operation was completed* on the LOB value through the locator L1. L1 is then termed an *updated locator*. This operation allows you to see your own changes to the LOB value on the next read through the *same locator, L1*.

> **Note:** the snapshot environment in the locator is *not* updated if the locator is used to merely read the LOB value. It is only updated *when you modify* the LOB value through the locator via the PL/SQL DBMS_LOB package or the OCI LOB APIs.

Any committed updates made by a different transaction are seen by L1 only if your transaction is a read-committed transaction and if you use L1 to update the LOB value after the other transaction committed.

> **Note:** When you update an internal LOB's value, the modification is always made to the most current LOB value.

Updating the value of the internal LOB through the OCI LOB APIs or the PL/SQL DBMS_LOB package can be thought of as updating the LOB value *and then reselecting* the locator that refers to the new LOB value.

Note that updating the LOB value through SQL is merely an UPDATE statement. It is up to you to do the reselect of the LOB locator or use the RETURNING clause in the UPDATE statement (see the *PL/SQL User's Guide and Reference*) so that the locator can see the changes made by the UPDATE statement. Unless you reselect the LOB locator or use the RETURNING clause, you may think you are reading the latest value when this is not the case. For this reason you should avoid mixing SQL DML with OCI and DBMS_LOB piecewise operations.

Using the *Multimedia_tab* as defined previously, a CLOB locator is created: clob_selected.

- At the time of the first SELECT INTO (at t1), the value in *story* is associated with the locator *clob_selected*.
- In the second operation (at t2), the value in *story* is modified through the SQL UPDATE command, bypassing the *clob_selected* locator. The locator still sees the value of the LOB as of the point in time of the original SELECT. In other words, the locator does not see the update made via the SQL UPDATE command. This is illustrated by the subsequent DBMS_LOB.READ() call.
- The third operation (at t3) re-selects the LOB value into the locator *clob_selected*. The locator is thus updated with the latest snapshot environment which allows the locator to see the change made by the previous SQL UPDATE command. Therefore, in the next DBMS_LOB.READ(), an error is returned because the LOB value is empty (i.e., it does not contain any data).

Example of Repercussions of Mixing SQL DML with DBMS_LOB

```
INSERT INTO Multimedia_tab VALUES (1, 'abcd', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

COMMIT;

DECLARE
  num_var           INTEGER;
  clob_selected     CLOB;
  read_amount       INTEGER;
  read_offset       INTEGER;
  buffer            VARCHAR2(20);

BEGIN

  -- At time t1:
  SELECT story INTO clob_selected
  FROM Multimedia_tab
  WHERE clip_id = 1;

  read_amount := 10;
  read_offset := 1;
  dbms_lob.read(clob_selected, read_amount, read_offset,
       buffer);
  dbms_output.put_line('clob_selected value: ' || buffer);
  -- Produces the output 'abcd'
```

```
  -- At time t2:
  UPDATE Multimedia_tab SET story = empty_clob()
      WHERE clip_id = 1;
  -- although the most current current LOB value is now empty,
  -- clob_selected still sees the LOB value as of the point
  -- in time of the SELECT

  read_amount := 10;
  dbms_lob.read(clob_selected, read_amount, read_offset,
     buffer);
  dbms_output.put_line('clob_selected value: ' || buffer);
  -- Produces the output 'abcd'

  -- At time t3:
  SELECT story INTO clob_selected FROM Multimedia_tab WHERE
       clip_id = 1;
  -- the SELECT allows clob_selected to see the most current
  -- LOB value

  read_amount := 10;
  dbms_lob.read(clob_selected, read_amount, read_offset,
       buffer);
  -- ERROR: ORA-01403: no data found
END;
/
```

> **Note:** We advise that you avoid updating the same LOB with different locators. You will avoid many pitfalls if you use only one locator to update the same LOB value.

Using the table *Multimedia_tab* as defined previously, two CLOBs are created as potential locators: `clob_updated` and `clob_copied`.

- At the time of the first `SELECT INTO` (at t1), the value in *story* is associated with the locator *clob_updated.*
- The second operation (at t2) copies the value in *clob_updated* to *clob_copied.* At this juncture, both locators see the same value. The example demonstrates this with a series of DBMS_LOB.READ() calls.
- At this juncture (at t3), the program utilizes DBMS_LOB.WRITE() to alter the value in *clob_updated*, and a DBMS_LOB.READ() reveals a new value.

- However, a DBMS_LOB.READ() of the value through *clob_copied* (at t4) reveals that it still sees the value of the LOB as of the point in time of the assignment from *clob_updated* (at t2).
- It is not until *clob_updated* is assigned to *clob_copied* (t5) that *clob_copied* sees the modification made by *clob_updated*.

Example of an Updated LOB Locator

```
INSERT INTO Multimedia_tab VALUES (1,'abcd', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

COMMIT;

DECLARE
  num_var          INTEGER;
  clob_updated     CLOB;
  clob_copied      CLOB;
  read_amount      INTEGER; ;
  read_offset      INTEGER;
  write_amount     INTEGER;
  write_offset     INTEGER;
  buffer           VARCHAR2(20);
BEGIN

-- At time t1:
  SELECT story INTO clob_updated FROM Multimedia_tab
      WHERE clip_id = 1
      FOR UPDATE;

  -- At time t2:
  clob_copied := clob_updated;
  -- after the assign, clob_copied and clob_updated see the same
  -- LOB value

  read_amount := 10;
  read_offset := 1;
  dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_updated value: ' || buffer);
  -- Produces the output 'abcd'

  read_amount := 10;
  dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_copied value: ' || buffer);
  -- Produces the output 'abcd'
```

```
  -- At time t3:
  write_amount := 3;
  write_offset := 5;
  buffer := 'efg';
  dbms_lob.write(clob_updated, write_amount, write_offset,
        buffer);

  read_amount := 10;
  dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_updated value: ' || buffer);
  -- Produces the output 'abcdefg'


  -- At time t4:
  read_amount := 10;
  dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_copied value: ' || buffer);
  -- Produces the output 'abcd'


  -- At time t5:
  clob_copied := clob_updated;

  read_amount := 10;
  dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_copied value: ' || buffer);
  -- Produces the output 'abcdefg'
END;
/
```

LOB Bind Variables

When a LOB locator is used as the source to update another internal LOB (as in a SQL INSERT or UPDATE statement, the DBMS_LOB.COPY() routine, and so on), the snapshot environment in the source LOB locator determines the LOB value that is used as the source. If the source locator (for example L1) is a read consistent locator, then the LOB value as of the point in time of the SELECT of L1 is used. If the source locator (for example L2) is an updated locator, then the LOB value associated with L2's snapshot environment at the time of the operation is used.

Using the table *Multimedia_tab* as defined previously, three CLOBs are created as potential locators: clob_selected, clob_updated and clob_copied.

- At the time of the first SELECT INTO (at t1), the value in *story* is associated with the locator *clob_updated.*
- The second operation (at t2) copies the value in *clob_updated* to *clob_copied.* At this juncture, both locators see the same value.
- Then (at t3), the program utilizes DBMS_LOB.WRITE() to alter the value in *clob_updated,* and a DBMS_LOB.READ() reveals a new value.
- However, a DBMS_LOB.READ of the value through *clob_copied* (at t4) reveals that *clob_copied* does not see the change made by *clob_updated.*
- Therefore (at t5), when *clob_copied* is used as the source for the value of the INSERT statement, we insert the value associated with *clob_copied* (i.e. without the new changes made by *clob_updated*). This is demonstrated by the subsequent DBMS_LOB.READ() of the value just inserted.

Example of Updating a LOB with a PL/SQL Variable

```
INSERT INTO Multimedia_tab VALUES (1, 'abcd', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

COMMIT;

DECLARE
  num_var           INTEGER;
  clob_selected     CLOB;
  clob_updated      CLOB;
  clob_copied       CLOB;
  read_amount       INTEGER;
  read_offset       INTEGER;
  write_amount      INTEGER;
  write_offset      INTEGER;
  buffer            VARCHAR2(20);
BEGIN

  -- At time t1:
  SELECT story INTO clob_updated FROM Multimedia_tab
      WHERE clip_id = 1
      FOR UPDATE;

  read_amount := 10;
  read_offset := 1;
  dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_updated value: ' || buffer);
  -- Produces the output 'abcd'
```

```
  -- At time t2:
  clob_copied := clob_updated;


  -- At time t3:
  write_amount := 3;
  write_offset := 5;
  buffer := 'efg';
  dbms_lob.write(clob_updated, write_amount, write_offset,
       buffer);

  read_amount := 10;
  dbms_lob.read(clob_updated, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_updated value: ' || buffer);
  -- Produces the output 'abcdefg'
  -- note that clob_copied doesn't see the write made before
  -- clob_updated


  -- At time t4:
  read_amount := 10;
  dbms_lob.read(clob_copied, read_amount, read_offset, buffer);
  dbms_output.put_line('clob_copied value: ' || buffer);
  -- Produces the output 'abcd'

  -- At time t5:
  -- the insert uses clob_copied view of the LOB value which does
  -- not include clob_updated changes
 INSERT INTO Multimedia_tab VALUES (2, clob_copied, EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL)
    RETURNING story INTO clob_selected;

  read_amount := 10;
  dbms_lob.read(clob_selected, read_amount, read_offset,
       buffer);
  dbms_output.put_line('clob_selected value: ' || buffer);
  -- Produces the output 'abcd'
END;
/
```

LOB locators cannot span transactions

Modifying an internal LOB's value through the LOB locator via DBMS_LOB, OCI, or SQL INSERT or UPDATE statements changes the locator from a read consistent locator to an updated locator. Further, the INSERT or UPDATE statement automatically starts a transaction and locks the row. Once this has occurred, the locator may *not* be used outside the current transaction to modify the LOB value. In other words, LOB locators that are used to write data cannot span transactions. However, the locator may be used to read the LOB value unless you are in a serializable transaction.

For more information about the relationship between LOBs and transaction boundaries see:

- "LOB Locators and Transaction Boundaries" on page 1-49

Using the table *Multimedia_tab* defined previously, a CLOB locator is created: clob_updated.

- At the time of the first SELECT INTO (at t1), the value in *story* is associated with the locator *clob_updated.*
- The second operation (at t2), utilizes the DBMS_LOB.WRITE() command to alter the value in *clob_updated*, and a DBMS_LOB.READ() reveals a new value.
- The commit statement (at t3) ends the current transaction.
- Therefore (at t4), the subsequent DBMS_LOB.WRITE() operation fails because the *clob_updated* locator refers to a different (already committed) transaction. This is noted by the error returned. You must re-select the LOB locator before using it in further DBMS_LOB (and OCI) modify operations.

Example of Locator Not Spanning a Transaction

```
INSERT INTO Multimedia_tab VALUES (1, 'abcd', EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);

COMMIT;

DECLARE
  num_var          INTEGER;
  clob_updated     CLOB;
  read_amount      INTEGER;
  read_offset      INTEGER;
```

```
  write_amount     INTEGER;
  write_offset     INTEGER;
  buffer           VARCHAR2(20);

BEGIN
          -- At time t1:
     SELECT      story
     INTO        clob_updated
     FROM        Multimedia_tab
     WHERE       clip_id = 1
     FOR UPDATE;

     read_amount := 10;
     read_offset := 1;
     dbms_lob.read(clob_updated, read_amount, read_offset,
          buffer);
     dbms_output.put_line('clob_updated value: ' || buffer);
     -- This produces the output 'abcd'

     -- At time t2:
     write_amount := 3;
     write_offset := 5;
     buffer := 'efg';
     dbms_lob.write(clob_updated, write_amount, write_offset,
          buffer);

     read_amount := 10;
     dbms_lob.read(clob_updated, read_amount, read_offset,
         buffer);
     dbms_output.put_line('clob_updated value: ' || buffer);
     -- This produces the output 'abcdefg'


    -- At time t3:
    COMMIT;


    -- At time t4:
    dbms_lob.write(clob_updated , write_amount, write_offset,
         buffer);
    -- ERROR: ORA-22990: LOB locators cannot span transactions
END;
/
```

LOBs in the Object Cache

When you create an object in the object cache that contains an internal LOB attribute, the LOB attribute is implicitly set to empty. You may not use this empty LOB locator to write data to the LOB. You must first *flush* the object, thereby inserting a row into the table and creating an empty LOB — that is, a LOB with 0 length. Once the object is refreshed in the object cache (use OCI_PIN_LATEST), the real LOB locator is read into the attribute, and you can then call the OCI LOB API to write data to the LOB.

When creating an object with a BFILE attribute, the BFILE is set to NULL. It must be updated with a valid directory alias and filename before reading from the file.

When you copy one object to another in the object cache with a LOB locator attribute, only the LOB locator is copied. This means that the LOB attribute in these two different objects contain exactly the same locator which refers to *one and the same* LOB *value*. Only when the target object is flushed is a separate, physical copy of the LOB value made, which is distinct from the source LOB value.

> **See Also:** "Example of a Read Consistent Locator" on page 2-3 for a description of what version of the LOB value will be seen by each object if a write is performed through one of the locators.

Therefore, in cases where you want to modify the LOB that was the target of the copy, *you must flush the target object, refresh the target object, and then* write to the LOB through the locator attribute.

LOB Buffering Subsystem

Oracle8 provides a LOB buffering subsystem (LBS) for advanced OCI based applications such as DataCartridges, Web servers, and other client-based applications that need to buffer the contents of one or more LOBs in the client's address space. The client-side memory requirement for the buffering subsystem during its maximum usage is 512K bytes. It is also the maximum amount that you can specify for a single read or write operation on a LOB that has been enabled for buffered access.

Advantages of LOB Buffering

The advantages of buffering, especially for client applications that perform a series of small reads and writes (often repeatedly) to specific regions of the LOB, are:

- Buffering enables deferred writes to the server. You can buffer up several writes in the LOB's buffer in the client's address space and eventually *flush* the buffer to the server. This reduces the number of network roundtrips from your client application to the server, and hence, makes for better overall performance for LOB updates.
- Buffering reduces the overall number of LOB updates on the server, thereby reducing the number of LOB versions and amount of logging. This results in better overall LOB performance and disk space usage.

Considerations in the Use of LOB Buffering

The following caveats hold for buffered LOB operations:

- Oracle8 provides a simple buffering subsystem, and *not* a cache. To be specific, Oracle8 does not guarantee that the contents of a LOB's buffer are always in synchronize with the LOB value in the server. Unless you *explicitly flush* the contents of a LOB's buffer, you will not see the results of your buffered writes reflected in the actual LOB on the server.
- Error recovery for buffered LOB operations is your responsibility. Owing to the deferred nature of the actual LOB update, error reporting for a particular buffered read or write operation is deferred until the next access to the server based LOB.
- Transactions involving buffered LOB operations cannot migrate across user sessions — the LBS is a single user, single threaded system.
- Oracle8 does not guarantee transactional support for buffered LOB operations. To ensure transactional semantics for buffered LOB updates, you must maintain logical savepoints in your application to rollback all the changes made to the buffered LOB in the event of an error. You should always wrap your buffered LOB updates within a logical savepoint (see "Example of LOB Buffering" on page 2-21).
- In any given transaction, once you have begun updating a LOB using buffered writes, it is your responsibility to ensure that the same LOB is not updated through any other operation within the scope of the same transaction *that bypasses the buffering subsystem*.

You could potentially do this by using an SQL statement to update the server-based LOB. Oracle8 cannot distinguish, and hence prevent, such an operation. This will seriously affect the correctness and integrity of your application.

- Buffered operations on a LOB are done through its locator, just as in the conventional case. A locator that is enabled for buffering will provide a consistent read version of the LOB, until you perform a write operation on the LOB through that locator.

  See Also: "Read-Consistent Locators" on page 2-2.

  Once the locator becomes an updated locator by virtue of its being used for a buffered write, it will always provide access to the most up-to-date version of the LOB *as seen through the buffering subsystem*. Buffering also imposes an additional significance to this updated locator — all further buffered writes to the LOB can be done *only through this updated locator*. Oracle8 will return an error if you attempt to write to the LOB through other locators enabled for buffering.

  See Also: "Updated locators" on page 2-5.

- You can pass an updated locator that was enabled for buffering as an IN parameter to a PL/SQL procedure. However, passing an IN OUT or an OUT parameter will produce an error, as will an attempt to return an updated locator.
- You cannot assign an updated locator that was enabled for buffering to another locator. There are a number of different ways that assignment of locators may occur — through OCILobAssign(), through assignment of PL/SQL variables, through OCIObjectCopy() where the object contains the LOB attribute, and so on. Assigning a consistent read locator that was enabled for buffering to a locator that did not have buffering enabled, turns buffering on for the target locator. By the same token, assigning a locator that was not enabled for buffering to a locator that did have buffering enabled, turns buffering off for the target locator.

  Similarly, if you SELECT into a locator for which buffering was originally enabled, the locator becomes overwritten with the new locator value, thereby turning buffering off.

- Appending to the LOB value using buffered write(s) is allowed, but only if the starting offset of these write(s) is exactly one byte (or character) past the end of the BLOB (or CLOB / NCLOB). In other words, the buffering subsystem does not support appends that involve creation of zero-byte fillers or spaces in the server based LOB.
- For CLOBs, Oracle8 requires that the character set form for the locator bind variable on the client side be the same as that of the LOB in the server. This is usually the case in most OCI LOB programs. The exception is when the locator is SELECTed from a *remote* database, which may have a different character set form from the database which is currently being accessed by the OCI program. In such a case, an error is returned. If there is no character set form input by the user, then we assume it is SQLCS_IMPLICIT.

LOB Buffering Operations

The Physical Structure of the LOB Buffer

Each user *session* has a fixed page pool of 16 pages, which are to be shared by all LOBs accessed in buffering mode from that session. Each *page* has a fixed size of up to 32K bytes (not characters) — to be precise, pagesize = n x CHUNKSIZE ~= 32K. A LOB's buffer consists of one or more of these pages, up to a maximum of 16 per session. The maximum amount that you ought to specify for any given buffered read or write operation is 512K bytes, remembering that under different circumstances the maximum amount you may read / write could be smaller.

Using the LOB Buffering System

Consider that a LOB is divided into fixed-size, logical regions. Each page is mapped to one of these fixed size regions, and is in essence, their in-memory copy. Depending on the input offset and amount specified for a read or write operation, Oracle8 allocates one or more of the free pages in the page pool to the LOB's buffer. A *free page* is one that has not been read or written by a buffered read or write operation.

For example, assuming a page size of 32K, for an input offset of 1000 and a specified read / write amount of 30000, Oracle8 reads the first 32K byte region of the LOB into a page in the LOB's buffer. For an input offset of 33000 and a read / write amount of 30000, the second 32K region of the LOB is read into a page. For an input offset of 1000, and a read / write amount of 35000, the LOB's buffer will contain *two* pages — the first mapped to the region 1 — 32K, and the second to the region 32K+1 — 64K of the LOB.

This mapping between a page and the LOB region is temporary until Oracle8 maps another region to the page. When you attempt to access a region of the LOB that is not already available in full in the LOB's buffer, Oracle8 allocates any available free page(s) from the page pool to the LOB's buffer. If there are no free pages available in the page pool, Oracle8 reallocates the pages as follows. It ages out the *least recently used* page among the *unmodified* pages in the LOB's buffer and reallocates it for the current operation.

If no such page is available in the LOB's buffer, it ages out the least recently used page among the *unmodified* pages of *other* buffered LOBs in the same session. Again, if no such page is available, then it implies that all the pages in the page pool are *dirty* (i.e. they have been modified), and either the currently accessed LOB, or one of the other LOBs, need to be flushed. Oracle8 notifies this condition to the user as an error. Oracle8 *never* flushes and reallocates a dirty page implicitly — you can either flush them explicitly, or discard them by disabling buffering on the LOB.

To illustrate the above discussion, consider two LOBs being accessed in buffered mode — L1 and L2, each with buffers of size 8 pages. Assume that 6 of the 8 pages in L1's buffer are dirty, with the remaining 2 contain unmodified data read in from the server. Assume similar conditions in L2's buffer. Now, for the next buffered operation on L1, Oracle8 will reallocate the least recently used page from the two unmodified pages in *L1's buffer*. Once all the 8 pages in L1's buffer are used up for LOB writes, Oracle8 can service two more operations on L1 by allocating the two unmodified pages from *L2's buffer* using the least recently used policy. But for any further buffered operations on L1 or L2, Oracle8 returns an error.

If all the buffers are dirty and you attempt another read from or write to a buffered LOB, you will receive the following error:

```
Error 22280: no more buffers available for operation
```

There are two possible causes:

1. All buffers in the buffer pool have been used up by previous operations.

   In this case, flush the LOB(s) through the locator that is being used to update the LOB.

2. You are trying to flush a LOB without any previous buffered update operations.

   In this case, write to the LOB through a locator enabled for buffering before attempting to flush buffers.

Flushing the LOB Buffer

The term *flush* refers to a set of processes. Writing data to the LOB in the buffer through the locator transforms the locator into an *updated locator*. Once you have updated the LOB data in the buffer through the updated locator, a flush call will

- write the dirty pages in the LOB's buffer to the server-based LOB, thereby updating the LOB value,
- reset the updated locator to be a read consistent locator, and
- either free the flushed buffers or turn the status of the buffer pages back from dirty to unmodified.

After the flush, the locator becomes a read consistent locator and can be assigned to another locator (L2 := L1).

For instance, suppose you have two locators, L1 and L2. Let us say that they are both *read consistent locators* and consistent with the state of the LOB data in the server. If you then update the LOB by writing to the buffer, L1 becomes an updated locator. L1 and L2 now refer to different versions of the LOB value. If you wish to update the LOB in the server, you must use L1 to retain the read consistent state captured in L2. The flush operation writes a new snapshot environment into the locator used for the flush. The important point to remember is that you must use the updated locator (L1), when you flush the LOB buffer. Trying to flush a read consistent locator will generate an error.

This raises the question: What happens to the data in the LOB buffer? There are two possibilities. In the default mode, the flush operation retains the data in the pages that were modified. In this case, when you read or write to the same range of bytes no roundtrip to the server is necessary. Note that *flush* in this context does not clear the data in the buffer. It also does not return the memory occupied by the flushed buffer to the client address space.

Note: Unmodified pages may now be aged out if necessary.

In the second case, you set the flag parameter in `OCILobFlushBuffer()` to `OCI_LOB_BUFFER_FREE` to free the buffer pages, and so return the memory to the client address space. Note that *flush* in this context updates the LOB value on the server, returns a read consistent locator, and frees the buffer pages.

Flushing the Updated LOB

It is very important to note that you must flush a LOB that has been updated through the LBS:

- before committing the transaction,
- before migrating from the current transaction to another,
- before disabling buffering operations on a LOB
- before returning from an external callout execution into the calling function / procedure / method in PL / SQL.

Note: When the external callout is called from a PL / SQL block and the locator is passed as a parameter, all buffering operations, including the enable call, should be made within the callout itself. In other words, we recommend that you adhere to the following sequence:

- call the external callout,
- enable the locator for buffering,
- read / write using the locator,
- flush the LOB,
- disable the locator for buffering, and
- return to the calling function / procedure / method in PL / SQL.

Remember that Oracle8 never implicitly flushes the LOB.

Using Locators Enabled for Buffering

Note that there are several cases in which you can use buffer-enabled locators and others in which you cannot.

- A locator that is enabled for buffering can only be used with the following OCI APIs:

  `OCILobRead`(), `OCILobWrite`(), `OCILobAssign`(), `OCILobIsEqual`(), `OCILobLocatorIsInit`(), `OCILobCharSetId`(), `OCILobCharSetForm`().

- The following OCI APIs will return errors if used with a locator enabled for buffering:

OCILobCopy(), OCILobAppend(), OCILobErase(), OCILobGetLength(), OCILobTrim().

These APIs will also return errors when used with a locator which has not been enabled for buffering, but the LOB that the locator represents is already being accessed in buffered mode through some other locator.

- An error is returned from DBMS_LOB APIs if the input lob locator has buffering enabled.
- As in the case of all other locators, locators enabled for LOB buffering cannot span transactions.

Saving Locator State so as to Avoid a Reselect

Suppose you want to save the current state of the LOB before further writing to the LOB buffer. In performing updates while using LOB buffering, writing to an existing buffer does not make a roundtrip to the server, and so does not refresh the snapshot environment in the locator. This would not be the case if you were updating the LOB directly without using LOB buffering. In that case, every update would involve a roundtrip to the server, and so would refresh the snapshot in the locator. In order to save the state of a LOB that has been written through the LOB buffer, you therefore need to

1. Flush the LOB, thereby updating the LOB and the snapshot environment in the locator (L1). At this point, the state of the locator (L1) and the LOB are the same.
2. Assign the locator (L1) used for flushing and updating to another locator (L2). At this point, the states of the two locators (L1 and L2), as well as the LOB are all identical.

L2 now becomes a read consistent locator with which you are able to access the changes made through L1 up until the time of the flush, but not after! This assignment avoids incurring a roundtrip to the server to reselect the locator into L2.

Example of LOB Buffering

The following pseudocode for an OCI program based on the Multimedia_tab schema briefly explains the concepts listed above.

```
OCI_BLOB_buffering_program()
{
   int            amount;
   int            offset;
   OCILobLocator  lbs_loc1, lbs_loc2, lbs_loc3;
   void          *buffer;
```

```
int           bufl;

-- Standard OCI initialization operations - logging on to
-- server, creating and initializing bind variables etc.

init_OCI();

-- Establish a savepoint before start of LBS operations
exec_statement("savepoint lbs_savepoint");

-- Initialize bind variable to BLOB columns from buffered
-- access:
exec_statement("select frame into lbs_loc1 from Multimedia_tab
    where clip_id = 12");
exec_statement("select frame into lbs_loc2 from Multimedia_tab
    where clip_id = 12 for update");
exec_statement("select frame into lbs_loc2 from Multimedia_tab
    where clip_id = 12 for update");

-- Enable locators for buffered mode access to LOB:
OCILobEnableBuffering(lbs_loc1);
OCILobEnableBuffering(lbs_loc2);
OCILobEnableBuffering(lbs_loc3);

-- Read 4K bytes through lbs_loc1 starting from offset 1:
amount = 4096; offset = 1; bufl = 4096;
OCILobRead(.., lbs_loc1, offset, &amount, buffer, bufl,
   ..);
   if (exception)
       goto exception_handler;
       -- This will read the first 32K bytes of the LOB from
       -- the server into a page (call it page_A) in the LOB's
       -- client-side buffer.
       -- lbs_loc1 is a read consistent locator.

       -- Write 4K of the LOB throgh lbs_loc2 starting from
       -- offset 1:
       amount = 4096; offset = 1; bufl = 4096;
       buffer = populate_buffer(4096);
       OCILobWrite(.., lbs_loc2, offset, amount, buffer,
           bufl, ..);

   if (exception)
       goto exception_handler;
       -- This will read the first 32K bytes of the LOB from
```

```
    -- the server into a new page (call it page_B) in the
    -- LOB's buffer, and modify the contents of this page
    -- with input buffer contents.
    -- lbs_loc2 is an updated locator.

    -- Read 20K bytes through lbs_loc1 starting from
    -- offset 10K
    amount = 20480; offset = 10240;
    OCILobRead(.., lbs_loc1, offset, &amount, buffer,
        bufl, ..);

if (exception)
  goto exception_handler;
    -- Read directly from page_A into the user buffer.
    -- There is no round-trip to the server because the
    -- data is already in the client-side buffer.

    -- Write 20K bytes through lbs_loc2 starting from offset
    -- 10K
    amount = 20480; offset = 10240; bufl = 20480;
    buffer = populate_buffer(20480);
    OCILobWrite(.., lbs_loc2, offset, amount, buffer,
         bufl, ..);

if (exception)
    goto exception_handler;
    -- The contents of the user buffer will now be written
    -- into page_B without involving a round-trip to the
    -- server.  This avoids making a new LOB version on the
    -- server and writing redo to the log.

    -- The following write through lbs_loc3 will also
    -- result in an error:
    amount = 20000; offset = 1000; bufl = 20000;
    buffer = populate_buffer(20000);
    OCILobWrite(.., lbs_loc3, offset, amount, buffer,
         bufl, ..);

if (exception)
    goto exception_handler;
    -- No two locators can be used to update a buffered LOB
    -- through the buffering subsystem

-- The following update through lbs_loc3 will also
-- result in an error
```

```
        OCILobFileCopy(.., lbs_loc3, lbs_loc2, ..);

        if (exception)
            goto exception_handler;
            -- Locators enabled for buffering cannot be used with
            -- operations like Append, Copy, Trim etc.

        -- When done, flush LOB's buffer to the server:
        OCILobFlushBuffer(.., lbs_loc2, OCI_LOB_BUFFER_NOFREE);

        if (exception)
           goto exception_handler;
           -- This flushes all the modified pages in the LOB's buffer,
           -- and resets lbs_loc2 from updated to read consistent
           -- locator. The modified pages remain in the buffer
           -- without freeing memory.  These pages can be aged
           -- out if necessary.

        -- Disable locators for buffered mode access to LOB */
        OCILobDisableBuffering(lbs_loc1);
        OCILobDisableBuffering(lbs_loc2);
        OCILobDisableBuffering(lbs_loc3);

        if (exception)
           goto exception_handler;
           -- This disables the three locators for buffered access,
           -- and frees up the LOB's buffer resources.

        exception_handler:
        handle_exception_reporting();
        exec_statement("rollback to savepoint lbs_savepoint");
}
```

User Guidelines for Best Performance Practices

- Since LOBs are big, you can obtain the best performance by reading and writing large chunks of a LOB value at a time. This helps in several respects:
  - **a.** If accessing the LOB from the client side and the client is at a different node than the server, large reads / writes reduce network overhead.
  - **b.** If using the 'NOCACHE' option, each small read / write incurs an I / O. Reading / writing large quantities of data reduces the I / O.

c. Writing to the LOB creates a new version of the LOB CHUNK. Therefore, writing small amounts at a time will incur the cost of a new version for each small write. If logging is on, the CHUNK is also stored in the redo log.

- If you need to read / write small pieces of LOB data on the client, use LOB buffering — see OCILobEnableBuffering(), OCILobDisableBuffering(), OCILobFlushBuffer(), OCILobWrite(), OCILobRead(). Basically, turn on LOB buffering before reading / writing small pieces of LOB data.

> **See Also:** "LOB Buffering Subsystem" on page 2-14 for more information on LOB buffering.

- Use OCILobWrite() and OCILobRead() with a callback so data is streamed to / from the LOB. Make sure that the length of the entire write is set in the 'amount' parameter on input. Whenever possible, read and write in multiples of the LOB chunk size.
- Use a checkout / checkin model for LOBs. LOBs are optimized for the following:

  a. SQL UPDATE which replaces the entire LOB value

  b. Copy the entire LOB data to the client, modify the LOB data on the client side, copy the entire LOB data back to the database. This can be done using OCILobRead() and OCILobWrite() with streaming.

Working with Varying-Width Character Data

In using the OCI, or any of the programmatic environments that access OCI functionality, character set conversions are implicitly performed when translating from one character set to another. However, no implicit translation is ever performed from binary data to a character set. When you use the loadfromfile operation to populate a CLOB or NCLOB, you are populating the LOB with binary data from the BFILE. In that case, you will need to perform character set conversions on the BFILE data before executing loadfromfile.

LOBs in Index Organized Tables

Index Organized Tables now support internal and external LOB columns. The SQL DDL, DML and piecewise operations on LOBs in index organized tables exhibit the

same behavior as that observed in conventional tables. The only exception is the default behavior of LOBs during creation. The main differences are:

- Tablespace mapping: By default, or unless specified otherwise, the LOB's data and index segments will be created in the tablespace in which the primary key index segments of the index organized table are created.
- Inline as compared to Out-of-line storage: By default, all LOBs in an index organized table created without an overflow segment will be stored out of line. In other words, if an index organized table is created without an overflow segment, the LOBs in this table have their default storage attributes as DISABLE STORAGE IN ROW. If you forcibly try to specify an ENABLE STORAGE IN ROW clause for such LOBs, SQL will raise an error.

  On the other hand, if an overflow segment has been specified, LOBs in index organized tables will exactly mimic their behavior in conventional tables (see "Stipulating Tablespace and Storage Characteristics for Internal Lobs" on page 3-8 in Chapter 3, "Internal Persistent LOBs").

Consider the following example:

```
CREATE TABLE iotlob_tab (c1 INTEGER primary key, c2 BLOB, c3 CLOB, c4
VARCHAR2(20))
  ORGANIZATION INDEX
    TABLESPACE iot_ts
    PCTFREE 10 PCTUSED 10 INITRANS 1 MAXTRANS 1 STORAGE (INITIAL 4K)
    PCTTHRESHOLD 50 INCLUDING c2
  OVERFLOW
    TABLESPACE ioto_ts
    PCTFREE 10 PCTUSED 10 INITRANS 1 MAXTRANS 1 STORAGE (INITIAL 8K) LOB (c2)
    STORE AS lobseg (TABLESPACE lob_ts DISABLE STORAGE IN ROW
                     CHUNK 1 PCTVERSION 1 CACHE STORAGE (INITIAL 2m)
                     INDEX LOBIDX_C1 (TABLESPACE lobidx_ts STORAGE (INITIAL
                                                                    4K)));
```

Executing these statements will result in the creation of an index organized table `iotlob_tab` with the following elements:

- A primary key index segment in the tablespace `iot_ts`,
- An overflow data segment in the tablespace `ioto_ts`,
- Columns starting from column `C3` being explicitly stored in the overflow data segment,
- `BLOB` (column *C2*) data segments in the tablespace `lob_ts`,

- BLOB (C2) index segments in the tablespace lobidx_ts,
- CLOB (C3) data segments in the tablespace iot_ts,
- CLOB (C3) index segments in the tablespace iot_ts,
- the CLOB (C3) stored in line by virtue of the IOT having an overflow segment,
- the BLOB (C2) column explicitly forced to be stored out of line.

  Note that, if no overflow had been specified, both C2 and C3 would have been stored out of line by default.

Other LOB features, such as BFILEs and varying character width LOBs, are also supported in index organized tables, and their usage is the same as conventional tables.

Note: Support for LOBs in partitioned index organized tables will be provided in a future release.

3

Internal Persistent LOBs

In this chapter we describe how to work with internal persistent LOBs in terms of use cases. That is, we discuss each operation on a LOB (such as "See If a LOB is Open") in terms of a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: Internal Persistent LOBs" on page 2-2). A summary figure, "Use Case Model Diagram: Internal Persistent LOBs", locates all the use cases in single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case in which you are interested by clicking on the relevant use case title.

The individual use cases are themselves laid out as follows:

- A figure that depicts the use case (see "Preface" for a description of how to interpret these diagrams).
- A scenario that portrays one implementation of the use case in terms of the hypothetical multimedia application described above (see "An Example Application" on page 1-39 in Chapter 1, "Introduction to Working With LOBs").
- Code examples in each of the programmatic environments which can be utilized to implement the use case (see "Programmatic Environments for Operating on LOBs" on page 1-9 in Chapter 1, "Introduction to Working With LOBs").

Use Case Model: Internal Persistent LOBs

Table 3–1 Use Case Model: Internal Persistent LOBs Basic Operations

Use Case and Page
Three Ways to Create a Table Containing a LOB on page 3-6
CREATE a Table Containing One or More LOB Columns on page 3-14
CREATE a Table Containing an Object Type with a LOB Attribute on page 3-18
CREATE a Table with a Nested Table Containing a LOB on page 3-22
Three Ways Of Inserting One or More LOB Values into a Row on page 3-25
INSERT a LOB Value using EMPTY_CLOB() or EMPTY_BLOB() on page 3-26
INSERT a Row Containing a LOB as SELECT on page 3-28
INSERT a Row by Initializing a LOB Locator Bind Variable on page 3-30
Load Data into an Internal LOB (BLOB, CLOB, NCLOB) on page 3-38
Load a LOB with Data from a BFILE on page 3-46
See If a LOB Is Open on page 3-56
Copy LONG to LOB on page 3-64
Checkout a LOB on page 3-68
Checkin a LOB on page 3-79
Display the LOB Data on page 3-93
Read Data from the LOB on page 3-104
Read a Portion of the LOB (substr) on page 3-115
Compare All or Part of Two LOBs on page 3-123
See If a Pattern Exists in the LOB (instr) on page 3-131
Get the Length of a LOB on page 3-138
Copy All or Part of a LOB to another LOB on page 3-146
Copy a LOB Locator on page 3-157
See If One LOB Locator Is Equal to Another on page 3-165
See If a LOB Locator Is Initialized on page 3-171
Get Character Set ID on page 3-175
Get Character Set Form on page 3-178
Append One LOB to Another on page 3-181

Use Case and Page
Write Append to a LOB on page 3-191
Write Data to a LOB on page 3-200
Trim the LOB Data on page 3-216
Erase Part of a LOB on page 3-226
Enable LOB Buffering on page 3-235
Flush Buffer on page 3-241
Disable LOB Buffering on page 3-246
Three Ways to Update a LOB on page 3-254
UPDATE a LOB with EMPTY_CLOB() or EMPTY_BLOB() on page 3-255
UPDATE as SELECT on page 3-257
UPDATE by Initializing a LOB Locator Bind Variable on page 3-258
DELETE the Row of a Table Containing a LOB on page 3-266

Figure 3–1 Use Case Model Diagram: Internal Persistent LOBs (part 1 of 2)

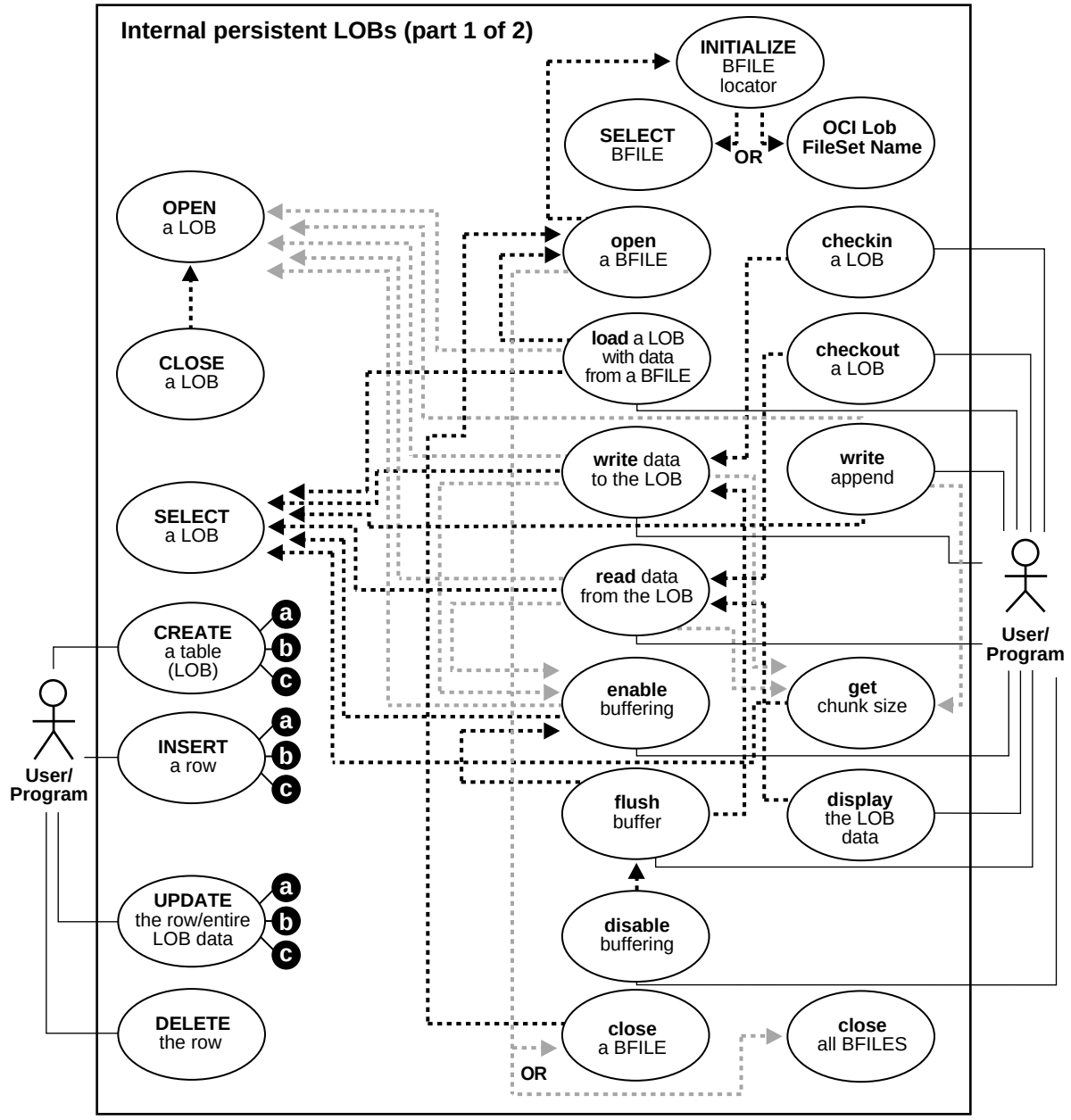
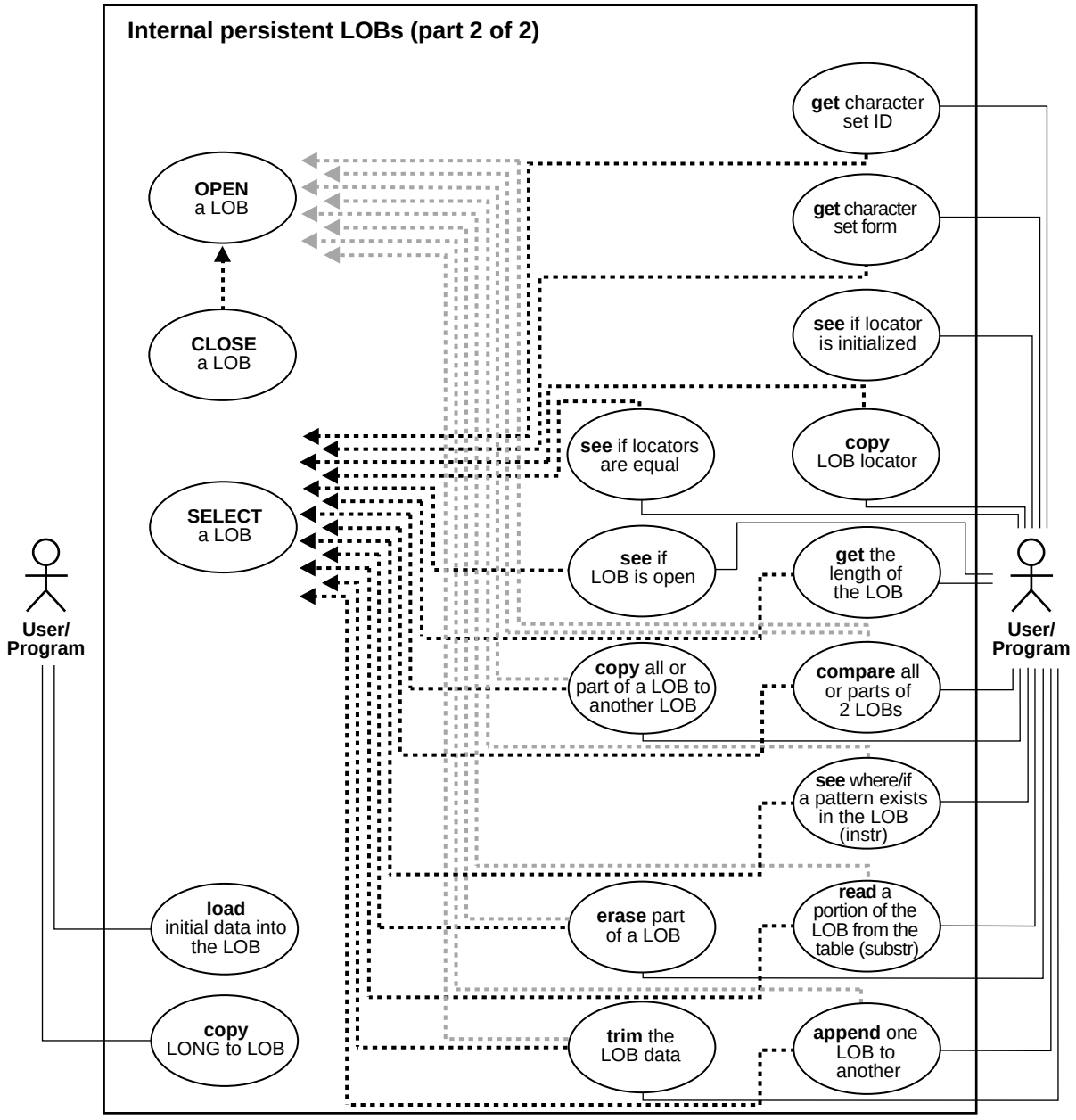
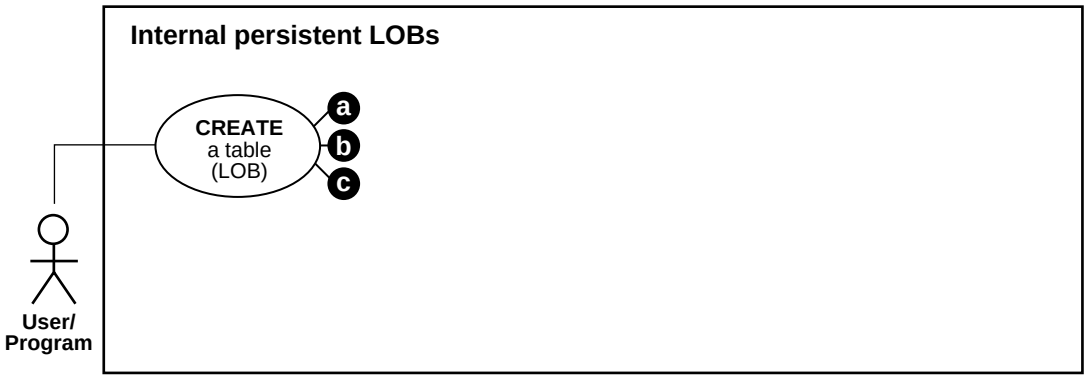


Figure 3–2 Use Case Model Diagram: Internal Persistent LOBs (part 2 of 2)



Three Ways to Create a Table Containing a LOB

Figure 3–3 Use Case Diagram: Three ways to CREATE a Table Containing a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

It is possible to incorporate LOBs into tables in three ways.

a. LOBs may be columns in a table — see "CREATE a Table Containing One or More LOB Columns" on page 3-14.

b. LOBs may be attributes of an object type — see "CREATE a Table Containing an Object Type with a LOB Attribute" on page 3-18.

c. LOBs may be contained within a nested table — see "CREATE a Table with a Nested Table Containing a LOB" on page 3-22.

In all cases SQL DDL is used — to define LOB columns in a table and LOB attributes in an object type.

Issues to Consider in Creating Tables that Will Contain LOBs

Initializing Internal LOBs to NULL or Empty

You can set an internal LOB — that is, a LOB column in a table, or a LOB attribute in an object type defined by you— to be NULL or empty. A LOB set to NULL has no locator. By contrast, an empty LOB stored in a table is a LOB of zero length that has a locator. So, if you SELECT from an empty LOB column / attribute, you get back a locator which you can use to populate the LOB with data via the OCI or DBMS_LOB routines. This is discussed in more detail below.

Alternatively, LOB columns, but not LOB attributes, may be initialized to a value. Which is to say — internal LOB attributes differ from internal LOB columns in that LOB attributes may not be initialized to a value other than NULL or empty. As discussed below, an external LOB (i.e. BFILE) can be initialized to NULL or to a filename.

You can initialize the LOBs in Multimedia_tab by using the following SQL INSERT statement:

```
INSERT INTO Multimedia_tab VALUES (1001, EMPTY_CLOB(), EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), EMPTY_BLOB(), NULL, NULL, NULL, NULL);
```

This sets the value of *story, flsub, frame* and *sound* to an empty value, and sets *photo, and music* to NULL.

Setting a LOB to NULL

You may want to set the internal LOB value to NULL upon inserting the row in cases where you do not have the LOB data at the time of the INSERT and / or if you want to issue a SELECT statement at some later time such as:

```
SELECT COUNT (*) FROM Voiced_tab WHERE Recording IS NOT NULL;
```

because you want to see all the voice-over segments that have been recorded, or

```
SELECT COUNT (*) FROM Voiced_tab WHERE Recording IS NULL;
```

if you wish to establish which segments still have to be recorded.

However, the drawback to this approach is that you must then issue a SQL UPDATE statement to reset the null LOB column — to EMPTY_BLOB() / EMPTY_CLOB() or to a value (e.g. 'Denzel Washington') for internal LOBs, or to a filename for external LOBs. The point is that you cannot call the OCI or the PL / SQL DBMS_LOB functions

on a LOB that is NULL. These functions only work with a locator, and if the LOB column is NULL, there is no locator in the row.

Setting an Internal LOB to Empty

If you do not want to set an internal LOB column to NULL, another option is for you to set the LOB value to empty by using the function EMPTY_BLOB () / EMPTY_CLOB() in the INSERT statement:

```
INSERT INTO a_table VALUES (EMPTY_BLOB());
```

Even better is to use the returning clause (thereby eliminating a round trip that is necessary for the subsequent SELECT), and then immediately call OCI or the PL/SQL DBMS_LOB functions to populate the LOB with data.

```
DECLARE
   Lob_loc  BLOB;
BEGIN
   INSERT INTO a_table VALUES (EMPTY_BLOB()) RETURNING blob_col INTO Lob_loc;
   /* Now use the locator Lob_loc to populate the BLOB with data */
END;
```

Stipulating Tablespace and Storage Characteristics for Internal Lobs

When defining LOBs in a table, you can explicitly indicate the tablespace and storage characteristics for each internal LOB. There are no extra tablespace or storage characteristics for external LOBs since they are not stored in the database. If you later wish to modify the LOB storage parameters, use the MODIFY LOB clause of the ALTER TABLE command. For example:

```
CREATE TABLE ContainsLOB_tab (n NUMBER, c CLOB)
      lob (c) STORE AS (CHUNK 4096
                        PCTVERSION 5
                        NOCACHE LOGGING
                        STORAGE (MAXEXTENTS 5)
                       );
```

Specifying a name for the LOB data segment makes for a much more intuitive working environment. When querying the LOB data dictionary views USER_LOBS, ALL_LOBS, DBA_LOBS (see *Oracle8i Reference*), you see the LOB data segment that you chose instead of system-generated names.

The LOB storage characteristics that can be specified for a LOB column or a LOB attribute include PCTVERSION, CACHE, NOCACHE, LOGGING, NOLOGGING, CHUNK

and ENABLE/DISABLE STORAGE IN ROW. For most users, defaults for these storage characteristics will be sufficient. If you want to fine-tune LOB storage, you should consider the following guidelines.

Tablespace and LOB Index

Best performance for LOBs can be achieved by specifying storage for LOBs in a tablespace that is different from the one used for the table that contains the LOB. If many different LOBs will be accessed frequently, it may also be useful to specify a separate tablespace for each LOB column/attribute in order to reduce device contention.

The LOB index is an internal structure that is strongly associated with the LOB storage. This implies that a user may not drop the LOB index and rebuild it. Note that the LOB index cannot be altered. The system determines which tablespace to use for the LOB data and LOB index depending on the user specification in the LOB storage clause:

- If you do not specify a tablespace for the LOB data, the table's tablespace is used for the LOB data and index.
- If you specify a tablespace for the LOB data, both the LOB data and index use the tablespace that was specified.

If in creating tables in 8.1 you specify a tablespace for the LOB index for a non-partitioned table, your specification of the tablespace will be ignored and the LOB index will be co-located with the LOB data. Partitioned LOBs do not include the LOB index syntax.

Specifying a separate tablespace for the LOB storage segments will allow for a decrease in contention on the table's tablespace.

PCTVERSION

When a LOB is modified, a new version of the LOB page is made in order to support consistent read of prior versions of the LOB value.

PCTVERSION is the percentage of all used LOB data space that can be occupied by old versions of LOB data pages. As soon as old versions of LOB data pages start to occupy more than the PCTVERSION amount of used LOB space, Oracle will try to reclaim the old versions and reuse them. In other words, PCTVERSION is the percent of used LOB data blocks that is available for versioning of old LOB data.

Default: 10 (%) Minimum: 0 (%) Maximum: 100 (%)

In order to decide what value PCTVERSION should be set to, you should consider how often LOBs are updated, and how often you read the updated LOBs.

Table 3–2 Recommended PCTVERSION Settings in Different Cases

LOB Update Pattern	LOB Read Pattern	PCTVERSION
Updates XX% of LOB data	Reads updated LOBs	XX%
Updates XX% of LOB data	Reads LOBs but not the updated LOBs	0%
Updates XX% of LOB data	Reads both LOBs and non-updated LOBs	XX%
Never updates LOB	Reads LOBs	0%

Example 1:

Several LOB updates concurrent with heavy reads of LOBs.

set PCTVERSION = 20%

Setting PCTVERSION to twice the default allows more free pages to be used for old versions of data pages. Since large queries may require consistent reads of LOBs, it may be useful to retain old versions of LOB pages. In this case LOB storage may grow because Oracle will not reuse free pages aggressively.

Example 2:

LOBs are created and written just once and are primarily read-only afterwards. Updates are infrequent.

set PCTVERSION = 5% or lower

The more infrequent and smaller the LOB updates are, the less space needs to be reserved for old copies of LOB data. If existing LOBs are known to be read-only, you could safely set PCTVERSION to 0% since there would never be any pages needed for old versions of data.

CACHE / NOCACHE

Use the CACHE option on LOBs if the same LOB data will be accessed frequently. Use the NOCACHE option (the default) if LOB data will be read only once, or infrequently.

LOGGING / NOLOGGING

[NO] LOGGING has a similar application with regard to using LOBs as it does for other table operations. In the normal case, if the [NO]LOGGING clause is omitted, this means that neither NO LOGGING nor LOGGING is specified and the logging attribute

of the table or table partition defaults to the logging attribute of the tablespace in which it resides.

For LOBs, there is a further alternative depending on how CACHE is stipulated.

- If the [NO]LOGGING clause is omitted and CACHE is specified, LOGGING is automatically implemented (because you cannot have CACHE NOLOGGING).
- If the [NO]LOGGING clause is omitted and CACHE is not specified, the process defaults in the same way as it does for tables and partitioned tables. That is, the [NO]LOGGING value is obtained from the tablespace in which the LOB value resides.

The following issues should also be kept in mind.

- LOBs will always generate undo for LOB index pages. Regardless of whether LOGGING or NOLOGGING is set LOBs will never generate rollback information (undo) for LOB data pages because old LOB data is stored in versions. Rollback information that is created for LOBs tends to be small because it is only for the LOB index page changes.
- When LOGGING is set Oracle will generate full redo for LOB data pages. NOLOGGING is intended to be used when a customer does not care about media recovery. Thus, if the disk / tape / storage media fails, you will not be able to recover your changes from the log since the changes were never logged.

  An example of when NOLOGGING is useful is bulk loads or inserts. For instance, when loading data into the LOB, if you don't care about redo and can just start the load over if it fails, set the LOB's data segment storage characteristics to NOCACHE NOLOGGING. This will give good performance for the initial load of data. Once you have completed loading the data, you can use ALTER TABLE to modify the LOB storage characteristics for the LOB data segment to be what you really want for normal LOB operations -- i.e. CACHE or NOCACHE LOGGING.

  Note: CACHE implies that you also get LOGGING.

CHUNK

Set CHUNK to the number of blocks of LOB data that will be accessed at one time i.e. the number of blocks that will be read / written via OCILobRead(), OCILobWrite(), DBMS_LOB.READ(), or DBMS_LOB.WRITE() during one access of the LOB value. Note that the default value for CHUNK is one Oracle block and does not vary across platforms. For example, if only one block of LOB data is accessed at

a time, set CHUNK to the size of one block. For example, if the database block size is 2K, then set CHUNK to 2K.

If you explicitly specify the storage characteristics for the LOB, make sure that INITIAL and NEXT for the LOB data segment storage are set to a size that is larger than the CHUNK size. For example, if the database block size is 2K and you specify a CHUNK of 8K, make sure that the INITIAL and NEXT are bigger than 8K and preferably considerably bigger (for example, at least 16K).

Put another way: If you specify a value for INITIAL, NEXT or the LOB CHUNK size, make sure that:

- CHUNK <= NEXT

and

- CHUNK <= INITIAL

ENABLE | DISABLE STORAGE IN ROW

You use the ENABLE | DISABLE STORAGE IN ROW clause to indicate whether the LOB should be stored inline (i.e. in the row) or out of line. You may not alter this specification once you have made it: if you ENABLE STORAGE IN ROW, you cannot alter it to DISABLE STORAGE IN ROW and vice versa. The default is ENABLE STORAGE IN ROW.

The maximum amount of LOB data that will be stored in the row is the maximum VARCHAR size (4000). Note that this includes the control information as well as the LOB value. If the user indicates that the LOB should be stored in the row, once the LOB value and control information is larger than 4000, the LOB value is automatically moved out of the row.

This suggests the following guideline. If the LOB is small (i.e. < 4000 bytes), then storing the LOB data out of line will decrease performance. However, storing the LOB in the row increases the size of the row. This will impact performance if the user is doing a lot of base table processing, such as full table scans, multi-row accesses (range scans) or many UPDATE/SELECT to columns other than the LOB columns. If the user doesn't expect the LOB data to be < 4000, i.e. if all LOBs are big, then the default is the best choice since

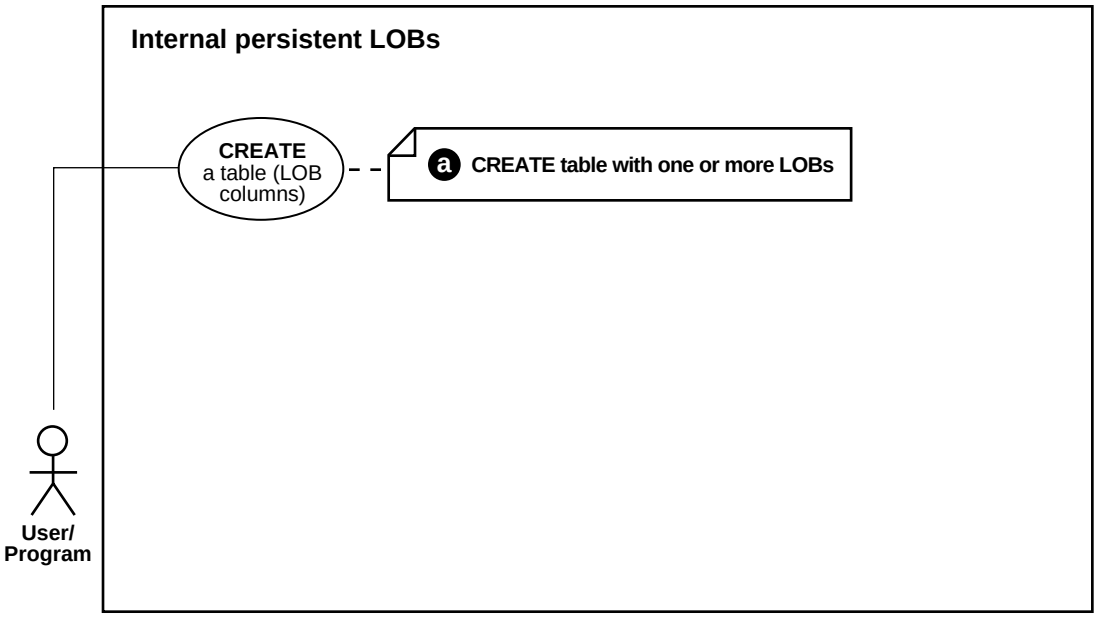
(a) the LOB data is automatically moved out of line once it gets bigger than 4000 (which will be the case here since the LOB data is big to begin with), and

(b) performance will be slightly better since we still store some control information in the row even after we move the LOB data out of the row.

For LOBs in index organized tables, inline LOB storage is allowed only if the table is created with an overflow segment (see "LOBs in Index Organized Tables" on page 2-25 in Chapter 2, "Advanced Topics").

CREATE a Table Containing One or More LOB Columns

Figure 3–4 Use Case Diagram: CREATE a Table Containing a LOB Column



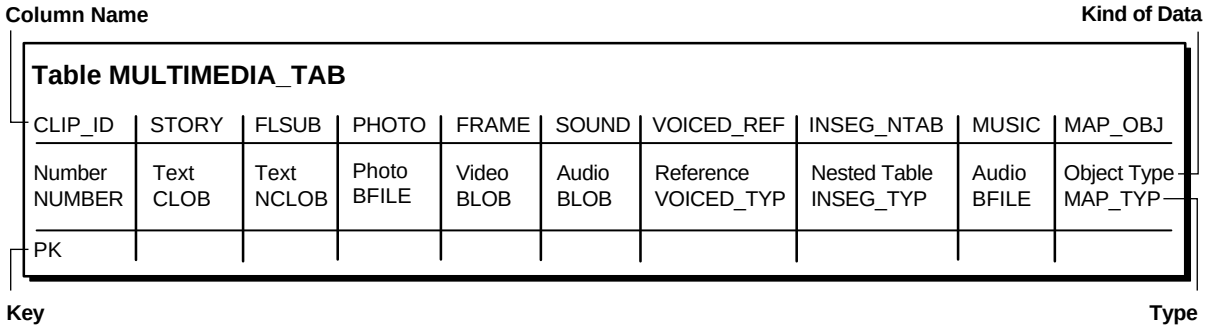
To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The heart of our hypothetical application is the table `Multimedia_tab`. The varied types which make up the columns of this table make it possible to collect together the many different kinds multimedia elements used in the composition of clips.

Figure 3–5 MULTIMEDIA_TAB as Example of Creating a Table Containing a LOB Column



Example: Create a Table Containing One or More LOB Columns using SQL DDL

Note: You may need to set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER samp CASCADE;
DROP DIRECTORY AUDIO_DIR;
DROP DIRECTORY FRAME_DIR;
DROP DIRECTORY PHOTO_DIR;

CREATE USER samp identified by samp;
GRANT CONNECT, RESOURCE to samp;
CREATE DIRECTORY AUDIO_DIR AS '/tmp/';
CREATE DIRECTORY FRAME_DIR AS '/tmp/';
CREATE DIRECTORY PHOTO_DIR AS '/tmp/';
GRANT READ ON DIRECTORY AUDIO_DIR to samp;
GRANT READ ON DIRECTORY FRAME_DIR to samp;
GRANT READ ON DIRECTORY PHOTO_DIR to samp;
```

Note (continued):

```
CONNECT samp/samp
CREATE TABLE a_table (blob_col BLOB);
CREATE TYPE Voiced_typ AS OBJECT (
   Originator      VARCHAR2(30),
   Script          CLOB,
   Actor           VARCHAR2(30),
   Take            NUMBER,
   Recording       BFILE
);

CREATE TABLE VoiceoverLib_tab of Voiced_typ (
Script DEFAULT EMPTY_CLOB(),
   CONSTRAINT TakeLib CHECK (Take IS NOT NULL),
   Recording DEFAULT NULL
);

CREATE TYPE InSeg_typ AS OBJECT (
   Segment         NUMBER,
   Interview_Date  DATE,
   Interviewer     VARCHAR2(30),
   Interviewee     VARCHAR2(30),
   Recording       BFILE,
   Transcript      CLOB
);

CREATE TYPE InSeg_tab AS TABLE of InSeg_typ;
CREATE TYPE Map_typ AS OBJECT (
   Region          VARCHAR2(30),
   NW              NUMBER,
   NE              NUMBER,
   SW              NUMBER,
   SE              NUMBER,
   Drawing         BLOB,
   Aerial          BFILE
);
CREATE TABLE Map_Libtab of Map_typ;
CREATE TABLE Voiceover_tab of Voiced_typ (
Script DEFAULT EMPTY_CLOB(),
   CONSTRAINT Take CHECK (Take IS NOT NULL),
   Recording DEFAULT NULL
);
```

Since one can use SQL DDL directly to create a table containing one or more LOB columns, it is not necessary to use the DBMS_LOB package.

```
CREATE TABLE Multimedia_tab (
   Clip_ID         NUMBER NOT NULL,
   Story           CLOB default EMPTY_CLOB(),
   FLSub           NCLOB default EMPTY_CLOB(),
   Photo           BFILE default NULL,
   Frame           BLOB default EMPTY_BLOB(),
   Sound           BLOB default EMPTY_BLOB(),
   Voiced_ref      REF Voiced_typ,
   InSeg_ntab      InSeg_tab,
   Music           BFILE default NULL,
   Map_obj         Map_typ
 ) NESTED TABLE    InSeg_ntab STORE AS InSeg_nestedtab;
```

Notes

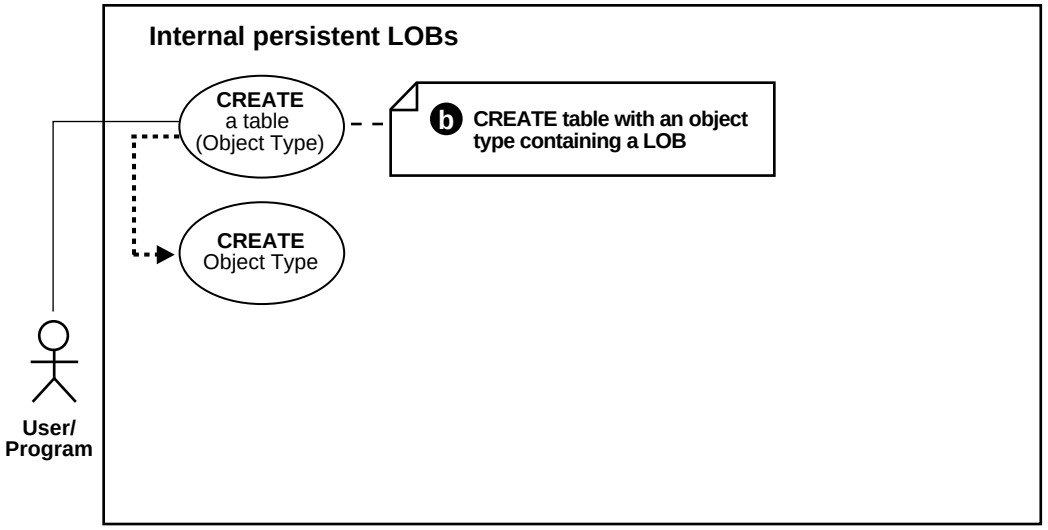
- The result of using the functions EMPTY_BLOB () and EMPTY_CLOB() means that the LOB is initialized, but not populated with data. LOBs that are empty are not null, and vice versa. This topic is discussed in more detail in "INSERT a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()" on page 3-26.
- For information about creating nested tables that have ore or more columns of LOB datatype see "CREATE a Table with a Nested Table Containing a LOB" on page 3-22
- The creation of an object column containing one or more LOBs is discussed under the heading "CREATE a Table Containing an Object Type with a LOB Attribute" on page 3-18.

For more information see:

- *Oracle8i SQL Reference* for a complete specification of the syntax for using LOBs in the DDL commands CREATE TABLE and ALTER TABLE with:
  - BLOB, CLOB, NCLOB and BFILE columns
  - EMPTY_BLOB and EMPTY_CLOB functions
  - LOB storage clause for internal LOB columns, and LOB attributes of embedded objects

CREATE a Table Containing an Object Type with a LOB Attribute

Figure 3–6 Use Case Diagram: Create a table Containing an Object Type as a LOB Attribute



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

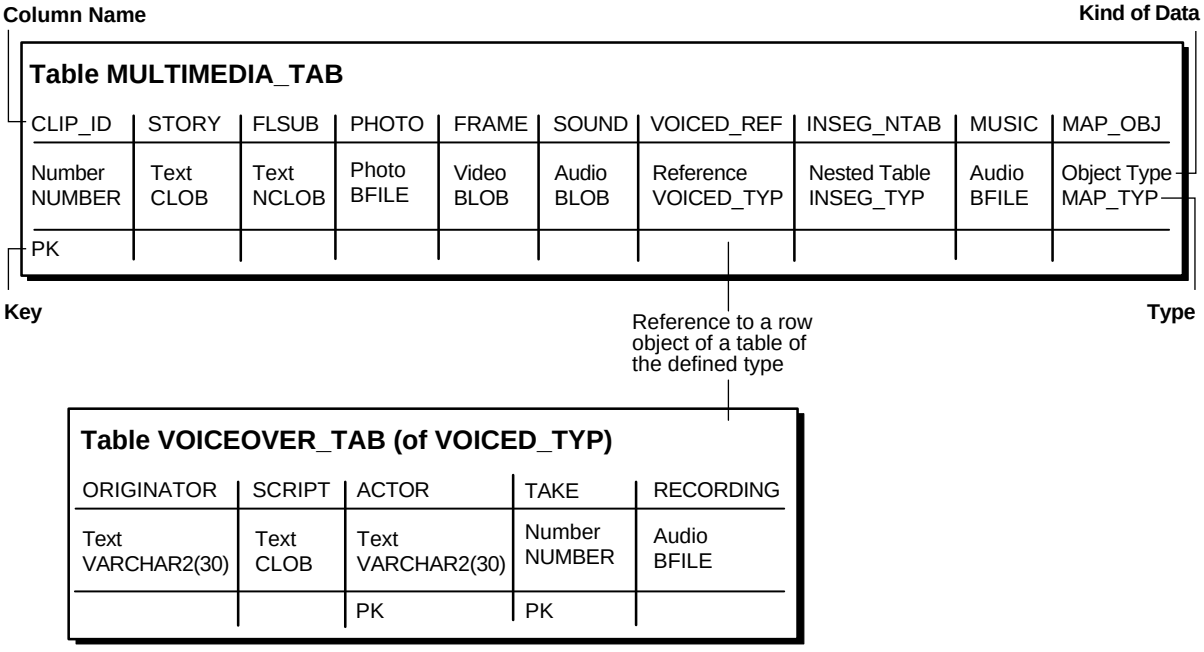
As shown in the diagram, you must create the object type that contains the LOB attributes before you can proceed to create a table that makes use of that object type.

Our example application contains examples of two different ways in which object types can contain LOBs:

- `Multimedia_tab` contains a column `Voiced_ref` that references row objects in the table `VoiceOver_tab` which is based on the type `Voiced_typ`. This type contains two kinds of LOBs — a `CLOB` to store the script that's read by the actor, and a `BFILE` to hold the audio recording.

- The table `Multimedia_tab` contains a column `Map_obj` that contains column objects of the type Map_typ. This type utilizes the `BLOB` datatype for storing maps in the form of drawings.

Figure 3–7 VOICED_TYP As An Example of Creating a Type Containing a LOB

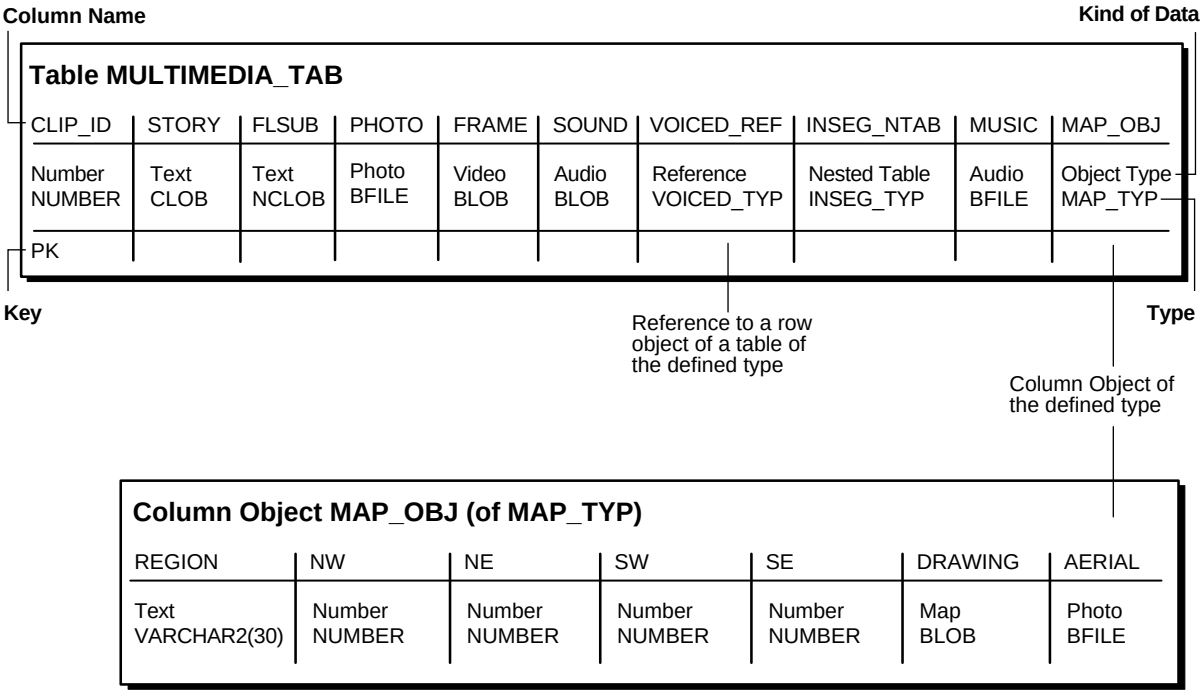


Example: Create a Table Containing an Object Type with a LOB Attribute Using SQL DDL

```
/* Create type Voiced_typ as a basis for tables that can contain recordings of
   voice-over readings using SQL DDL: */
CREATE TYPE Voiced_typ AS OBJECT (
   Originator      VARCHAR2(30),
   Script          CLOB,
   Actor           VARCHAR2(30),
   Take            NUMBER,
   Recording       BFILE
);
```

```
/* Create table Voiceover_tab Using SQL DDL: */
CREATE TABLE Voiceover_tab of Voiced_typ (
Script DEFAULT EMPTY_CLOB(),
   CONSTRAINT Take CHECK (Take IS NOT NULL),
   Recording DEFAULT NULL
);
```

Figure 3–8 MAP_TYP As An Example of Creating a Type Containing a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

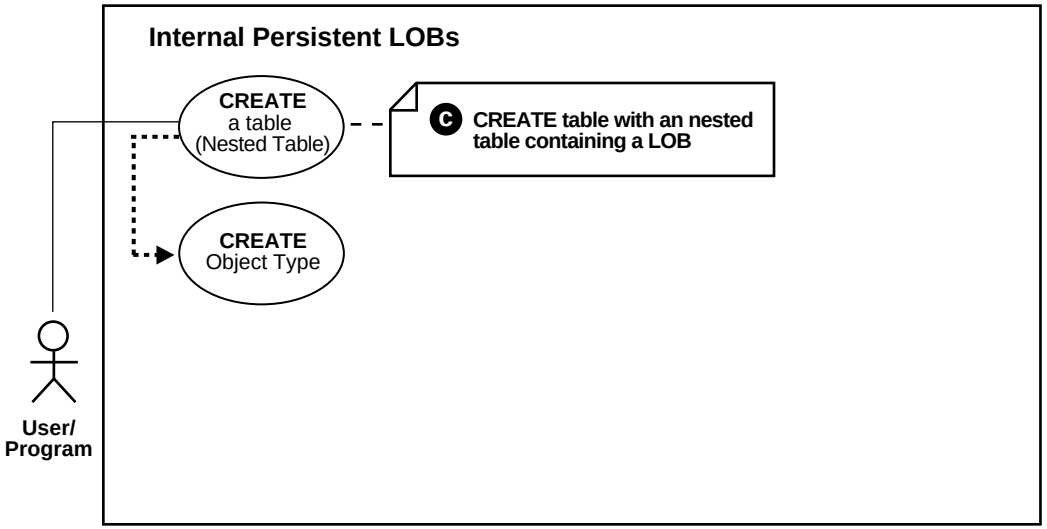
```
/* Create Type Map_typ using SQL DDL as a basis for the table that will contain
   the column object: */
CREATE TYPE Map_typ AS OBJECT (
   Region          VARCHAR2(30),
   NW              NUMBER,
   NE              NUMBER,
   SW              NUMBER,
   SE              NUMBER,
   Drawing         BLOB,
   Aerial          BFILE
);

/* Create support table MapLib_tab as an archive of maps using SQL DDL: */
CREATE TABLE MapLib_tab of Map_typ;
```

For more information see:

- *Oracle8i SQL Reference* for a complete specification of the syntax for using LOBs in the DDL commands CREATE TYPE and ALTER TYPE with BLOB, CLOB, and BFILE attributes (noting that NCLOBs cannot be attributes of an object type).

CREATE a Table with a Nested Table Containing a LOB

Figure 3–9 Use Case Diagram: Create a table with a Nested Table Containing a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

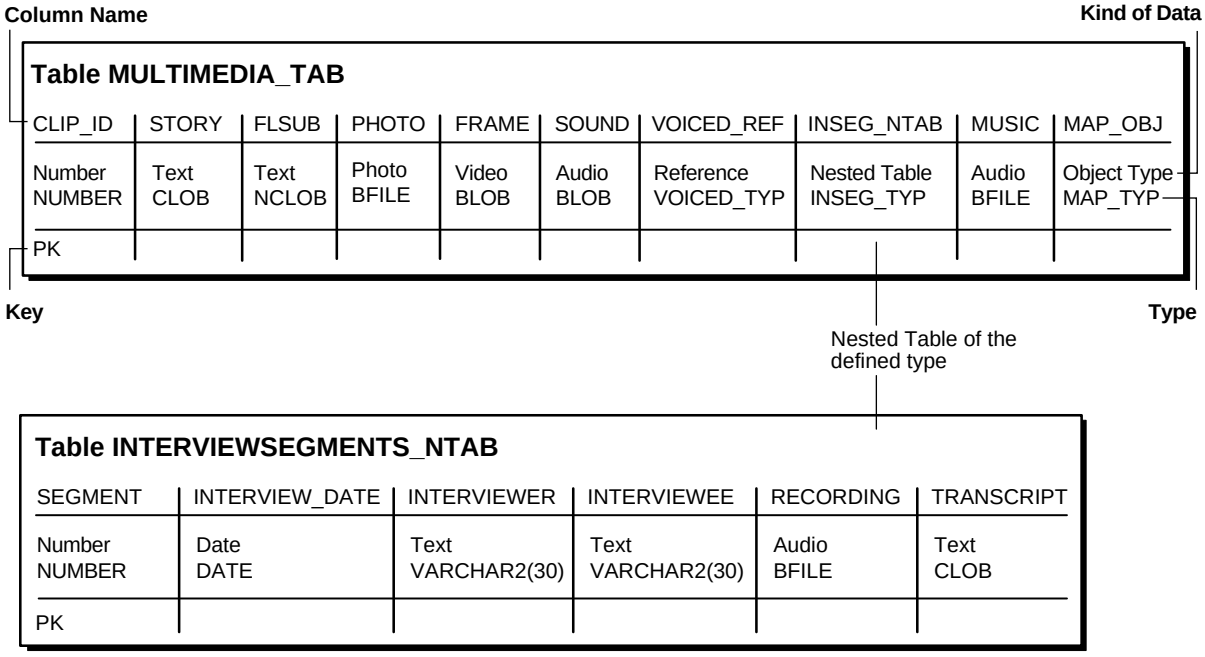
Scenario

As shown in the diagram, you must create the object type that contains the LOB attributes before you can proceed to create a nested table based on that object type.

In our example, `Multimedia_tab` contains a nested table `Inseg_ntab` that is based on the type `InSeg_typ`. This type makes use of two LOB datatypes — a `BFILE` for audio recordings of the interviews, and a `CLOB` should the user wish to make transcripts of the recordings.

We have already described how to create a table with LOB columns (see "CREATE a Table Containing One or More LOB Columns" on page 3-14), so here we only describe the SQL DDL syntax the creating the underlying type:

Figure 3–10 INTERVIEWSEGMENTS_NTAB As An Example of Creating a Nested Table Containing a LOB



Example: Create a Table with a Nested Table Containing a LOB Using SQL DDL

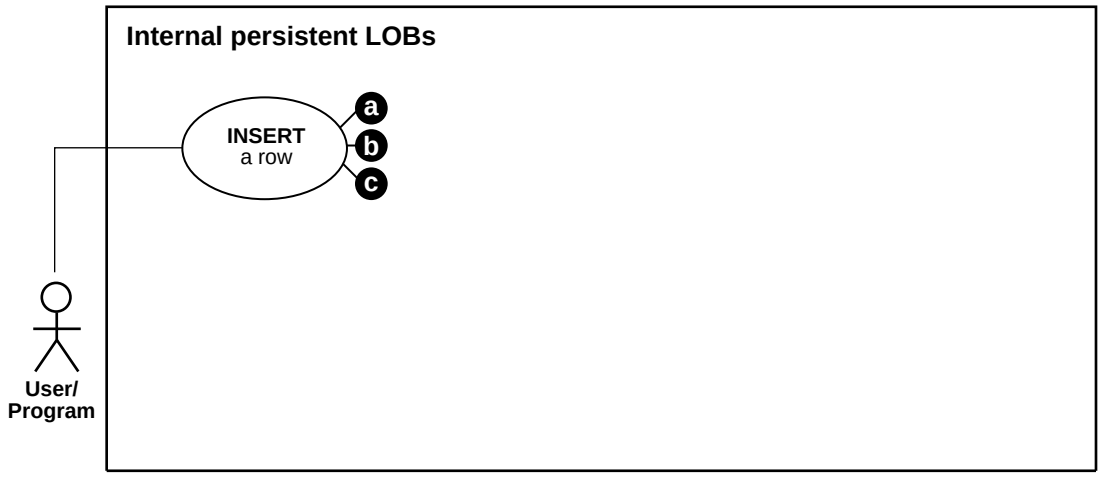
```
/* Create a type InSeg_typ as the base type for the nested table containing
   a LOB: */
CREATE TYPE InSeg_typ AS OBJECT (
   Segment         NUMBER,
   Interview_Date  DATE,
   Interviewer     VARCHAR2(30),
   Interviewee     VARCHAR2(30),
   Recording       BFILE,
   Transcript      CLOB
);

/* Type created, but need a nested table of that type to embed in
 multi_media_tab; so: */
CREATE TYPE InSeg_tab AS TABLE of InSeg_typ;
```

The actual embedding of the nested table is accomplished when the structure of the containing table is defined. In our example, this is effected by means of the following statement at the time that `Multimedia_tab` is created.

```
NESTED TABLE InSeg_ntab STORE AS InSeg_nestedtab;
```

Three Ways Of Inserting One or More LOB Values into a Row

Figure 3–11 Three Ways of Inserting LOB Values into a Row



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

There are three different ways of inserting LOB values into a row:

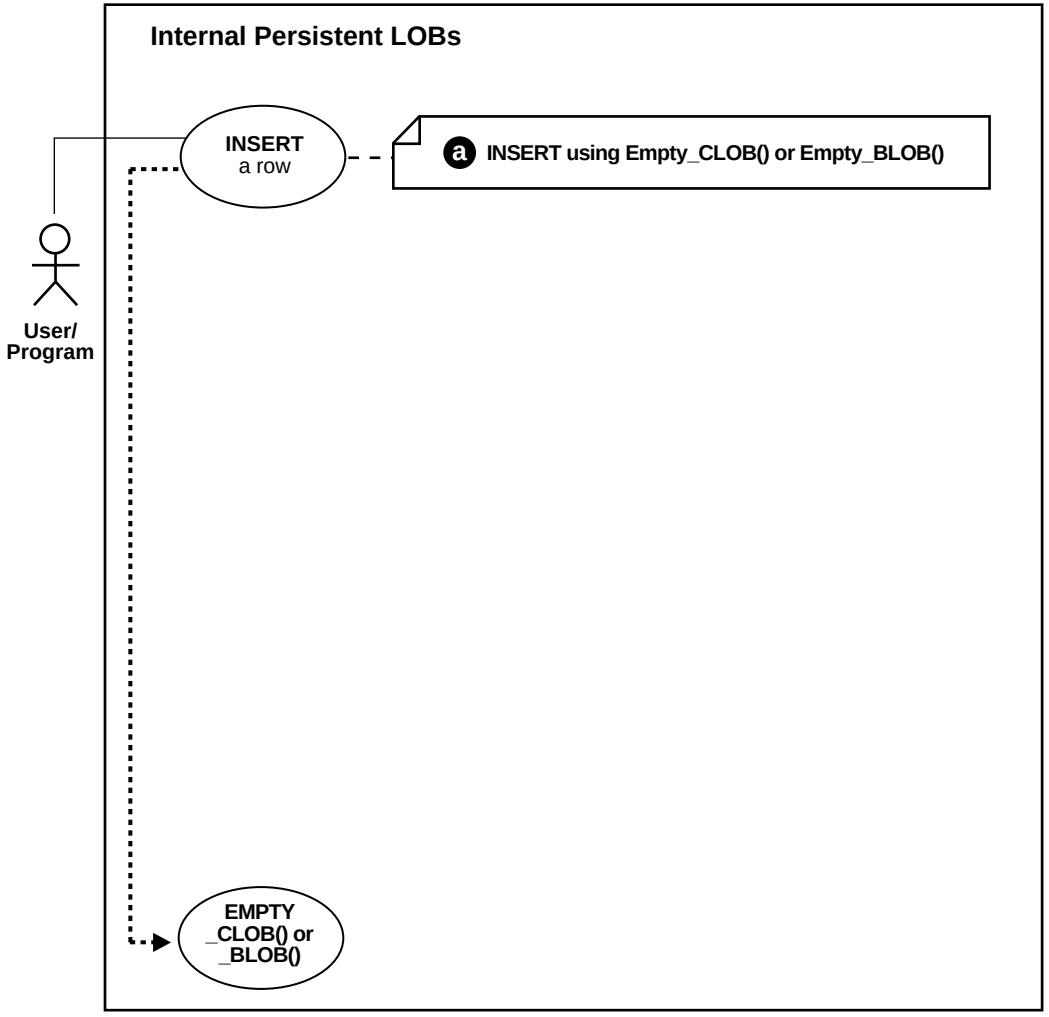
a. LOBs may be inserted into a row by first initializing a locator — see "INSERT a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()" on page 3-26

b. LOBs may be inserted by selecting a a row from another table— see "INSERT a Row Containing a LOB as SELECT" on page 3-28.

c. LOBs may be inserted by first initializing a LOB locator bind variable — see "INSERT a Row by Initializing a LOB Locator Bind Variable" on page 3-30.

INSERT a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()

Figure 3–12 Use Case Diagram: INSERT a Row using EMPTY_CLOB() or EMPTY_BLOB()



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Making a LOB Column Non-Null

Before you can start writing data to an internal LOB, the LOB column must be made non-null; that is, it must contain a locator that points to an empty or populated LOB value. You can initialize a BLOB column's value by using the function EMPTY_BLOB() as a default predicate. Similarly, a CLOB or NCLOB column's value can be initialized by using the function EMPTY_CLOB(). You can perform this initialization during CREATE TABLE (see "CREATE a Table Containing One or More LOB Columns") or, as in this case, by means of an INSERT.

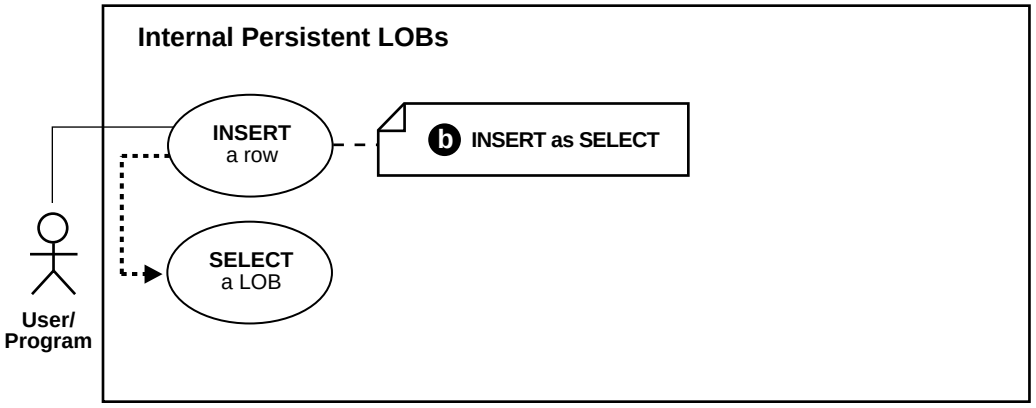
Example: Insert a Value by means of EMPTY_CLOB() / EMPTY_BLOB() using SQL

These functions are available as special functions in Oracle8 SQL DML, and are not part of the DBMS_LOB package.

```
/* In the new row of table Multimedia_tab,
   the columns STORY and FLSUB are initialized using EMPTY_CLOB(),
   the columns FRAME and SOUND are initialized using EMPTY_BLOB(),
   the column TRANSSCRIPT in the nested table is initialized using EMPTY_CLOB(),
  the column DRAWING in the column object is initialized using EMPTY_BLOB(): */
INSERT INTO Multimedia_tab
   VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(), NULL, EMPTY_BLOB(), EMPTY_BLOB(),
   NULL, InSeg_tab(InSeg_typ(1, NULL, 'Ted Koppell', 'Jimmy Carter', NULL,
   EMPTY_CLOB())), NULL, Map_typ('Moon Mountain', 23, 34, 45, 56, EMPTY_BLOB(),
   NULL));

/* In the new row of table Voiceover_tab, the column SCRIPT is initialized using
   EMPTY_CLOB(): */
INSERT INTO Voiceover_tab
    VALUES ('Abraham Lincoln', EMPTY_CLOB(), 'James Earl Jones', 1, NULL);
```

INSERT a Row Containing a LOB as SELECT

Figure 3–13 Use Case Diagram: Insert a Row as SELECT



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

With regard to LOBs, one of the advantages of utilizing an object-relational approach is that you can define a type as a common template for related tables. For instance, it makes sense that both the tables that store archival material and the working tables that use those libraries share a common structure. The following code fragment is based on the fact that a library table `VoiceoverLib_tab` is of the same type (`Voiced_typ`) as `Voiceover_tab` referenced by the `Voiced_ref` column of the `Multimedia_tab` table. It inserts values into the library table, and then inserts this same data into `Multimedia_tab` by means of a `SELECT` operation.

Note that the internal `LOB` types — `BLOB`, `CLOB`, and `NCLOB` — use *copy semantics*, as opposed to the *reference semantics* that apply to `BFILE`s. When a `BLOB`, `CLOB`, or `NCLOB` is copied from one row to another row in the same table or in a different table, the actual `LOB` value is copied, not just the `LOB` locator. For example, assuming `Voiceover_tab` and `VoiceoverLib_tab` have identical schemas, the

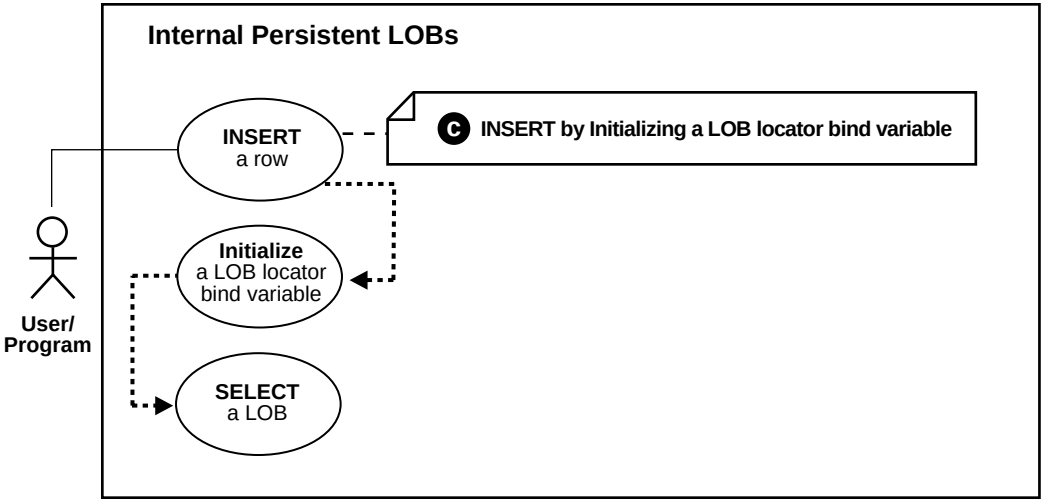
statement creates a new LOB locator in the table Voiceover_tab, and copies the LOB data from VoiceoverLib_tab to the location pointed to by a new LOB locator which is inserted in table Voiceover_tab.

Example: Insert a Row by Selecting from Another Table Using SQL DML

```
/* Store records in the archive table VoiceoverLib_tab: */
INSERT INTO VoiceoverLib_tab
   VALUES ('George Washington', EMPTY_CLOB(), 'Robert Redford', 1, NULL);

/* Insert values into Voiceover_tab by selecting from VoiceoverLib_tab: */
INSERT INTO Voiceover_tab
   (SELECT * from VoiceoverLib_tab
       WHERE Take = 1);
```

INSERT a Row by Initializing a LOB Locator Bind Variable

Figure 3–14 Use Case Diagram: INSERT a Row by Initializing a LOB Locator Bind Variable



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

In this example we use a `LOB` locator bind variable to take `Sound` data that is in one row of `Multimedia_tab` and insert it into another row.

- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using SQL DML" on page 3-31
- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using C (OCI)" on page 3-31
- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Pro*COBOL" on page 3-33

- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using C++ (Pro*C / C++)" on page 3-35
- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Visual Basic (OO4O)" on page 3-36
- "Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Java (JDBC)" on page 3-36

Example: Insert a Row by Initializing a LOB Locator Bind Variable Using SQL DML

```
/* Note that the example procedure insertUseBindVariable_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE insertUseBindVariable_proc
   (Rownum IN NUMBER, Blob_loc IN BLOB) IS
BEGIN
   INSERT INTO Multimedia_tab (Clip_ID, Sound) VALUES (Rownum, Blob_loc);
END;

DECLARE
   Blob_loc  BLOB;
BEGIN
   /* Select the LOB from the row where Clip_ID = 1,
      Initialize the LOB locator bind variable: */
   SELECT Sound INTO Blob_loc
      FROM Multimedia_tab
      WHERE Clip_ID = 1;
  /* Insert into the row where Clip_ID = 2: */
  insertUseBindVariable_proc (2, Blob_loc);
  COMMIT;
END;
```

Example: Insert a Row by Initializing a LOB Locator Bind Variable Using C (OCI)

```
/* Select the locator into a locator variable */

sb4 select_MultimediaLocator (Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCIStmt       *stmthp;
OCISvcCtx     *svchp;
{

  OCIDefine *defnp1;
```

```
  text  *sqlstmt =
    (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1";

  /* Prepare the SQL statement */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                 (ub2)SQLT_BLOB,(dvoid *)0, (ub2 *)0, (ub2 *)0,
                                  (ub4)OCI_DEFAULT));

  /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return (0);

}
/* Insert the selected Locator into table using Bind Variables.
   This function selects a locator from the Multimedia_tab and inserts
   it into the same table in another row.
 */
void insertUseBindVariable (envhp, errhp, svchp, stmthp)
OCIEnv        *envhp;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  int            clipid;
  OCILobLocator *Lob_loc;
  OCIBind       *bndhp2;
  OCIBind       *bndhp1;

  text          *insstmt =
   (text *) "INSERT INTO Multimedia_tab (Clip_ID,  Sound) VALUES (:1, :2)";


  /* Allocate locator resources */
  (void) OCIDescriptorAlloc((dvoid *) envhp,
                            (dvoid **) &Lob_loc, (ub4)OCI_DTYPE_LOB,
```

```
                              (size_t) 0, (dvoid **) 0);

  /* Select a LOB locator from the Multimedia Table */
  select_MultimediaLocator(Lob_loc, errhp, stmthp, svchp);


  /* Insert the locator into the Multimedia_tab with Clip_ID=2 */
  clipid = 2;

  /* Prepare the SQL statement */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Binds the bind positions */
  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                                (dvoid *) &clipid, (sb4) sizeof(clipid),
                                SQLT_INT, (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));
  checkerr (errhp, OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 2,
                                (dvoid *) &Lob_loc, (sb4) 0,  SQLT_BLOB,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the SQL statement */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  /* Free LOB resources*/
  OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

}
```

Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Pro*COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. INSERT-LOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 BLOB1 SQL-BLOB.
01  USERID   PIC X(11) VALUES "USER1/USER1".
```

```
    EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 INSERT-LOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
       CONNECT :USERID
    END-EXEC.

* Initialize the BLOB locator
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

* Populate the LOB
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
       SELECT SOUND INTO :BLOB1
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
    END-EXEC.

* Insert the value with CLIP_ID of 2.
    EXEC SQL
       INSERT INTO MULTIMEDIA_TAB (CLIP_ID, SOUND)
          VALUES (2, :BLOB1)
    END-EXEC.

* Free resources held by locator
 END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.

    EXEC SQL COMMIT WORK END-EXEC.
    STOP RUN.

 SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED:".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
```

```
            STOP RUN.
```

Example: Insert a Row by Initializing a LOB Locator Bind Variable Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
   EXEC SQL WHENEVER SQLERROR CONTINUE;
   printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
   EXEC SQL ROLLBACK WORK RELEASE;
   exit(1);
}

void insertUseBindVariable_proc(Rownum, Lob_loc)
   int Rownum;
   OCIBlobLocator *Lob_loc;
{
   EXEC SQL WHENEVER SQLERROR DO Sample_Error();
   EXEC SQL INSERT INTO Multimedia_tab (Clip_ID, Sound)
      VALUES (:Rownum, :Lob_loc);
}

void insertBLOB_proc()
{
   OCIBlobLocator *Lob_loc;

   /* Initialize the BLOB Locator: */
   EXEC SQL ALLOCATE :Lob_loc;
   /* Select the LOB from the row where Clip_ID = 1: */
   EXEC SQL SELECT Sound INTO :Lob_loc
      FROM Multimedia_tab WHERE Clip_ID = 1;
   /* Insert into the row where Clip_ID = 2: */
   insertUseBindVariable_proc(2, Lob_loc);
   /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
```

```
   char *samp = "samp/samp";
   EXEC SQL CONNECT :samp;
   insertBLOB_proc();
   EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Visual Basic (OO4O)

```
Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraSoundClone as OraBLOB

Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
'Clone it for future reference
Set OraSoundClone = OraSound1

'Go to Next row
OraDyn.MoveNext

'Lets update the current row and set the LOB to OraSoundClone
OraDyn.Edit
Set OraSound1 = OraSoundClone
OraDyn.Update
```

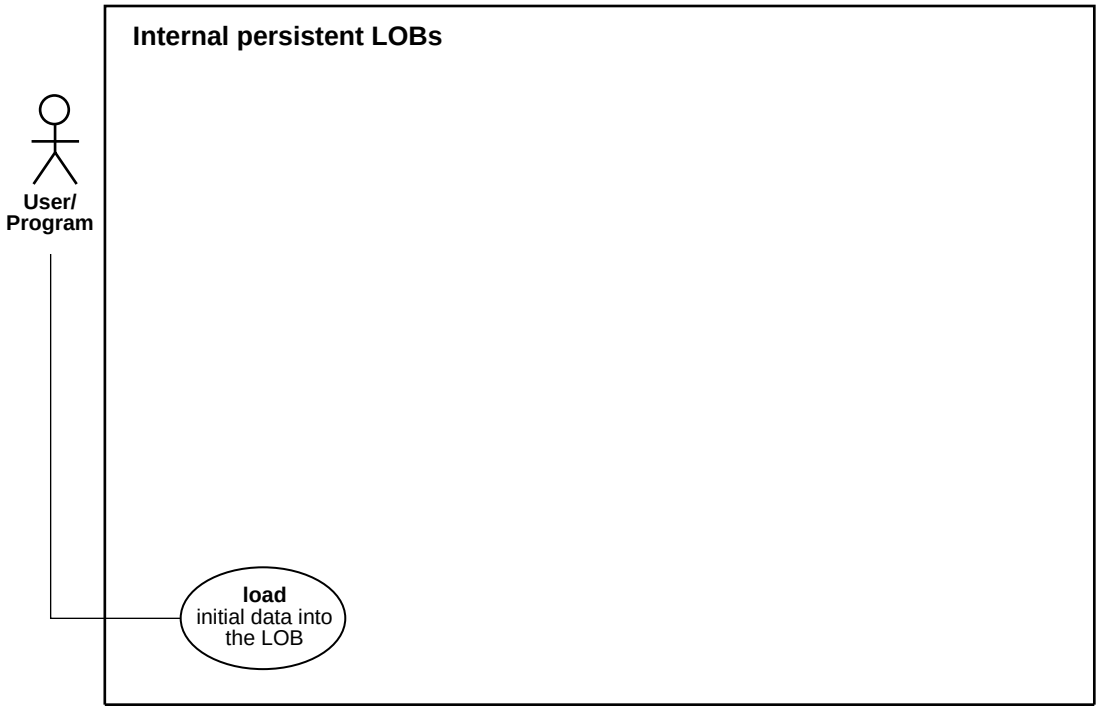
Example: Insert a Row by Initializing a LOB Locator Bind Variable Using Java (JDBC)

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_31
{
  public static void main (String args [])
```

```
      throws Exception
  {
    // Load the Oracle JDBC driver
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
      ResultSet rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 1");
      if (rset.next())
      {
        // retrieve the LOB locator from the ResultSet
        BLOB sound_blob = ((OracleResultSet)rset).getBLOB (1);

        OraclePreparedStatement ops =
        (OraclePreparedStatement) conn.prepareStatement(
          "INSERT INTO multimedia_tab (clip_id, sound) VALUES (2, ?)");

        ops.setBlob(1, sound_blob);
        ops.execute();
        conn.commit();
        conn.close();
      }
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
}
```

Load Data into an Internal LOB (BLOB, CLOB, NCLOB)

Figure 3–15 Use Case Diagram: Load the Initial Data into the Internal LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

Since LOBs can be quite large in size, it makes sense that SQL*Loader can load LOB data from either the main datafile (that is, in-line with the rest of the data) or from one or more secondary datafiles.

To load LOB data from the main datafile, the usual SQL*Loader formats can be used. The LOB data instances can be in predetermined size fields, delimited fields, or length-value pair fields.

LOB Data in Predetermined Size Fields

- This is a very fast and conceptually simple way to load LOBs; unfortunately, the LOBs to be loaded will not usually be of the same size (note: a possible work-around to this problem is to pad the LOB data with white space to make all of the LOBs the same length within the particular datafield; for information on the trimming of trailing white spaces see "Trimming of Blanks and Tabs" in the *Oracle8i Utilities*). To load LOBs using this format, use either CHAR or RAW as the loading datatype. For example:

Control File:

```
LOAD DATA
INFILE 'sample.dat' "fix 21"
INTO TABLE Multimedia_tab
    (Clip_ID POSITION(1:3) INTEGER EXTERNAL,
    Story POSITION(5:20)   CHAR DEFAULTIF Story=BLANKS)
```

Data file (sample.dat):

```
007 Once upon a time
```

Note:

- One space separates the Clip_ID,(007) from the beginning of the story. The story is 20 characters long.
- If the datafield containing the story is empty, then an empty LOB instead of a null LOB is produced. A null LOB is produced if the NULLIF directive was used instead of the DEFAULTIF directive. Also note that you can use loader datatypes other than CHAR to load LOBS. When loading BLOBs you would probably want to use the RAW datatype.

LOB Data in Delimited Fields

In this format, having different size LOBs within the same column (that is, datafile field) is not a problem. The trade-off for this added flexibility is performance. Loading in this format is somewhat slower because the loader has to scan through the data, looking for the delimiter string. For example:

Control File:

```
LOAD DATA
INFILE 'sample1.dat' "str X'7c0a'"
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(
Clip_ID     CHAR(3),
 Story       CHAR(507) ENCLOSED BY '<startlob>' AND '<endlob>'
)
```

Data file(sample1.dat):

```
007,    <startlob>     Once upon a time,The end.     <endlob>|
008,     <startlob>     Once upon another time ....The end.     <endlob>|
```

Note:

- <startlob> and <endlob> are the delimiting strings. Note that the maximum length for a LOB that can be read using the CHAR (507) is 507 bytes.
- If the record separator ' | ' was placed right after <endlob> and followed with the newline character, the newline would have been interpreted as part of the next record. One way around this problem would be to make the newline part of the record separator (for example, "| \n" or in hexadecimal notation: X"7c0a").

LOB Data in Length-value Pair Fields

You could use VARCHAR (see *Oracle8i Utilities*), VARCHARC, or VARRAW datatypes to load LOB data organized in this way. Note that this method of loading produces better performance over the previous method, however, it removes some of the flexibility (that is, it requires you to know the LOB length for each LOB before loading).

Control File:

```
LOAD DATA
INFILE 'sample2.dat' "str X'3c656e647265633e0a'"
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(
Clip_ID      INTEGER EXTERNAL (3),
 Story        VARCHARC (3, 500)
```

```
)
```

Data file (sample2.dat):

```
007,041    Once upon a time...  ....  The end.  <endrec>
008,000<endrec>
```

Note:

- If the escape character was not supported, the string used as a record separator in the example could have been expressed in hexadecimal.
- `Story` is a field corresponding to a `CLOB` column. In the control file, it is described as a `VARCHARC` whose length field is 3 characters long and maximum size is 500 bytes.
- The length subfield of the `VARCHARC` is 0 (that is, the value subfield is empty); consequently, the `LOB` instance is initialized to empty.
- Make sure the last character of the last line of the data file above is a line feed.

As mentioned earlier, `LOB` data can be so large that it is very reasonable to want to load it from secondary datafile(s). While you can use secondary data files as the source of `LOB` data, it is better to use `LOBFILE`s instead.

In the `LOBFILE`, `LOB` data instances are still thought to be in fields (predetermined size, delimited, length-value), but these fields are not organized into records (the concept of a record does not exist within `LOBFILES`); thus, the processing overhead of dealing with records is avoided. This type of organization of data is ideal for `LOB` loading.

One LOB per file

Each `LOBFILE` clause is the source of just one LOB. To load LOB data organized in this manner into the control file, follow the column/field name with the `LOBFILE` specification and the datatype specification. The following example illustrates loading `LOBS`, with one `LOB` per file.

Control File:

```
LOAD DATA
INFILE 'sample3.dat'
INTO TABLE Multimedia_tab
REPLACE
```

```
FIELDS TERMINATED BY ','
(
 Clip_ID        INTEGER EXTERNAL(5),
 ext_FileName   FILLER CHAR(40),
 Story          LOBFILE(ext_FileName) TERMINATED BY EOF
)
```

Data file (sample3.dat):

```
007,FirstStory.txt,
008,/tmp/SecondStory.txt,
```

Secondary Data file (FirstStory.txt):

```
Once upon a time ...
The end.
```

Secondary Data file (SecondStory.txt):

```
Once upon another time ....
The end.
```

Note:

- The FILLER field is mapped to the 40-byte long datafield which is read using the SQL*Loader CHAR datatype.
- The SQL*Loader gets the LOBFILE file name from the ext-FileName FILLER field. The data from a specified LOBFILE file (that is, from the first byte to the EOF character) is loaded to make a LOB instance. Note that if you specify a LOBFILE file that doesn't exist, the Story field is initialized to empty. Also note that since no SQL*Loader datatype is specified, the CHAR datatype is used.

Predetermined Size LOBs

In the control file, the size of the LOBs to be loaded into a particular column is specified. During the load, any LOB data loaded into that particular column is assumed to be of the specified size. The predetermined size of the fields allows the dataparser to perform very well. Unfortunately, it is often hard to guarantee that all of the LOBs are of the same size.

Control File:

```
LOAD DATA
```

```
INFILE 'sample4.dat'
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(
 Clip_ID    INTEGER EXTERNAL(5),
 Story      LOBFILE (CONSTANT 'FirstStory1.txt') CHAR(32)
)
```

Data file (sample4.dat):

```
007,
008,
```

Secondary Data file (FirstStory1.txt):

```
Once upon the time ...
The end,
Upon another time ...
The end,
```

Note:

- The loader loads 2000 bytes of data from the `FirstStory. txt LOBFILE`, using CHAR datatype, starting with the byte following the byte loaded last during the current loading session.
- There is a newline after the comma in the last line of the data file.

Delimited LOBs

The LOB data instances in the LOBFILE files are delimited. In this format, loading different size LOBs into the same column is not a problem. The trade-off for this added flexibility is performance. Loading in this format is somewhat slower because the loader has to scan through the data, looking for the delimiter string. For example:

Control File:

```
LOAD DATA
INFILE 'sample5.dat'
INTO TABLE Multimedia_tab
FIELDS  TERMINATED BY ','
(Clip_ID    INTEGER EXTERNAL(5),
Story         LOBFILE (CONSTANT 'FirstStory2.txt') CHAR(2000)
```

```
TERMINATED BY "<endlob>")
```

Data file (sample5.dat):

```
007,
008,
```

Secondary Data file (FirstStory2.txt):

```
Once upon a time...
The end.<endlob>
Once upon another time...
The end.<endlob>
```

Note:

Specifying maximum length (that is, 2000) gives a hint to the loader as to the maximum length of the field. This often results in optimized memory usage. (Note that if you use this hint, you should not estimate the value too low). The TERMINATED BY clause specifies the string that terminates the LOBs. You can also use the ENCLOSED BY clause. Note that the ENCLOSED BY clause allows a bit more flexibility as to the relative positioning of the LOBs in the LOBFILE (that is, the LOBs in the LOBFILE wouldn't have to follow one after another).

Length-Value Pair Specified LOBs

Each LOB in the LOBFILE is preceded by its length. You can use VARCHAR (see Oracle8 Utilities), VARCHARC, or VARRAW datatypes to load LOB data organized in this way. The controllable syntax for loading length-value pair specified LOBs is quite simple.

Note that this method of loading enjoys better performance over the previous one, but at the same time it takes some of the flexibility away (that is, it requires that you know the length of each LOB before loading).

Control File:

```
LOAD DATA
INFILE 'sample6.dat'
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(
Clip_ID     INTEGER EXTERNAL(5),
Story       LOBFILE (CONSTANT 'FirstStory3.txt') VARCHARC(4,2000)
```

```
)
```

Data file (sample6.dat):

```
007,
008,
```

Secondary Data file (FirstStory3.txt):

```
0031
Once upon a time ... The end.
0000
```

Note:

The VARCHARC(4, 2000) tells the loader that the LOBs in the LOBFILE are in length-value pair format and that the first four bytes should be interpreted as the length. The max_length part (that is, 2000) gives the hint to the loader as to the maximum size of the field.

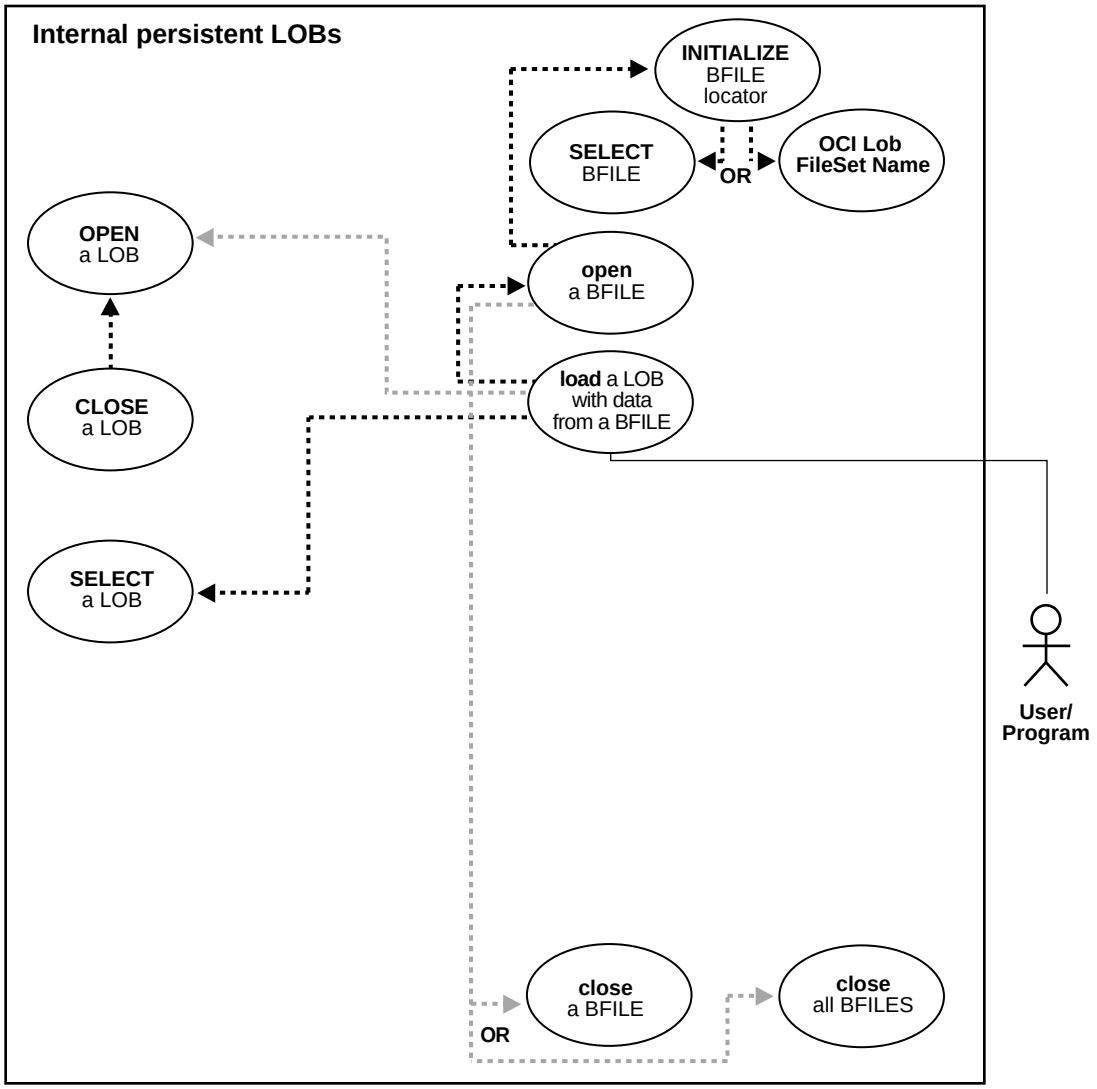
- 0031 tells the loader that the next 31 characters belong to the specified LOB.
- 0000 results in an empty LOB (not a NULL LOB).

Note the following LOB loading details:

- The failure to load a particular LOB doesn't result in the rejection of the record containing that LOB; instead, the record ends up containing an empty LOB.
- It is not necessary to specify the maximum length of the field corresponding to a LOB-type column. Nevertheless, if the maximum length is specified, it is taken as a hint to help optimize memory usage. It is very important that the maximum length specification doesn't underestimate the true maximum length.

Load a LOB with Data from a BFILE

Figure 3–16 Use Case Diagram: Load a LOB with data from a BFILE



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Character Set Conversion

In using the OCI, or any of the programmatic environments that access OCI functionality, character set conversions are implicitly performed when translating from one character set to another. However, no implicit translation is ever performed from binary data to a character set. When you use the `loadfromfile` operation to populate a `CLOB` or `NCLOB`, you are populating the `LOB` with binary data from the `BFILE`. In that case, you will need to perform character set conversions on the `BFILE` data before executing `loadfromfile`.

Scenario

The example procedure assumes that there is an operating system source file (`Washington_audio`) that contains the LOB data to be loaded into the target LOB (`Music`). The example procedure also assumes that the directory object `AUDIO_DIR` already exists and is mapped to the location of the source file.

- "Example: Load a LOB with Data from a BFILE Using the DBMS_LOB Package" on page 3-47
- "Example: Load a LOB with Data from a BFILE Using C (OCI)" on page 3-48
- "Example: Load a LOB with Data from a BFILE Using COBOL (Pro*COBOL)" on page 3-50
- "Example: Load a LOB with Data from a BFILE Using C++ (Pro*C/C++)" on page 3-52
- "Example: Load a LOB with Data from a BFILE Using Visual Basic (OO4O)" on page 3-53
- "Example: Load a LOB with Data from a BFILE Using Java (JDBC)" on page 3-54

Example: Load a LOB with Data from a BFILE Using the DBMS_LOB Package

```
/* Note that the example procedure loadLOBFromBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
   Dest_loc      BLOB;
```

```
   Src_loc        BFILE := BFILENAME('FRAME_DIR', 'Washington_frame');
   Amount         INTEGER := 4000;
BEGIN
   SELECT Frame INTO Dest_loc FROM Multimedia_tab
      WHERE Clip_ID = 3 FOR UPDATE;
   /* Opening the LOB is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
END;
```

Example: Load a LOB with Data from a BFILE Using C (OCI)

```
/* This example illustrates how to select a BLOB from a Multimedia
   table and load it with data from a BFILE
 */

sb4 select_lock_frame_locator_3(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=3 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
```

```
                                    (ub4) OCI_DEFAULT));

  return 0;
}

void LoadLobDataFromBFile(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{

  OCILobLocator *bfile;
  OCILobLocator *blob;
  ub4            amount= 4000;

  /* Allocate the Source (bfile) & destination (blob) locators desriptors*/
  OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&bfile,
                     (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid **)0);
  OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&blob,
                     (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

  /* Select a frame locator for update */
  printf (" select the frame locator...\n");
  select_lock_frame_locator_2(blob, errhp, svchp, stmthp);

  /* Set the Directory Alias and File Name of the frame file */
  printf ("  set the file name in bfile\n");
  checkerr (errhp, OCILobFileSetName(envhp, errhp, &bfile, (text*)"FRAME_DIR",
                                     (ub2)strlen("FRAME_DIR"),
                                     (text*)"Washington_frame",
                                     (ub2)strlen("Washington_frame")));

  printf (" open the bfile\n");
  /* Opening the BFILE locator is Mandatory */
  checkerr (errhp, (OCILobOpen(svchp, errhp, bfile, OCI_LOB_READONLY)));

  printf("  open the lob\n");
  /* Opening the BLOB locator is optional */
  checkerr (errhp, (OCILobOpen(svchp, errhp, blob, OCI_LOB_READWRITE)));

  /* Load the data from the audio file (bfile) into the blob */
  printf (" load the LOB from File\n");
  checkerr (errhp, OCILobLoadFromFile(svchp, errhp, blob, bfile, (ub4)amount,
                                      (ub4)1, (ub4)1));
```

```
  /* Closing the LOBs is Mandatory if they have been Opened */
  checkerr (errhp, OCILobClose(svchp, errhp, bfile));
  checkerr (errhp, OCILobClose(svchp, errhp, blob));

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) bfile, (ub4) OCI_DTYPE_FILE);
  (void) OCIDescriptorFree((dvoid *) blob, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Load a LOB with Data from a BFILE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-LOAD.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  DEST           SQL-BLOB.
01  BFILE1         SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
* Declare the amount to load.  The value here
* was chosen arbitrarily
01  LOB-AMT        PIC S9(9) COMP VALUE 10.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
LOB-LOAD.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
       CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* Set up the directory and file information
    MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
```

```
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Allocate and initialize the destination BLOB
     EXEC SQL ALLOCATE :DEST END-EXEC.
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
     EXEC SQL
        SELECT SOUND INTO :DEST
        FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3 FOR UPDATE
     END-EXEC.

* Open the source BFILE for READ
     EXEC SQL
        LOB OPEN :BFILE1 READ ONLY
     END-EXEC.

* Open the destination BLOB for READ/WRITE
     EXEC SQL
        LOB OPEN :DEST READ WRITE
     END-EXEC.

* Load the destination BLOB from the source BFILE
     EXEC SQL
        LOB LOAD :LOB-AMT FROM FILE :BFILE1 INTO :DEST
     END-EXEC.

* Close the source and destination LOBs
     EXEC SQL LOB CLOSE :BFILE1 END-EXEC.
     EXEC SQL LOB CLOSE :DEST END-EXEC.

 END-OF-BLOB.
     EXEC SQL FREE :DEST END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL
        COMMIT WORK RELEASE
     END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
```

```
   WHENEVER SQLERROR CONTINUE
END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

Example: Load a LOB with Data from a BFILE Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void loadLOBFromBFILE_proc()
{
  OCIBlobLocator *Dest_loc;
  OCIBFileLocator *Src_loc;
  char *Dir = "FRAME_DIR", *Name = "Washington_frame";
  int Amount = 4000;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Initialize the BFILE Locator */
  EXEC SQL ALLOCATE :Src_loc;
  EXEC SQL LOB FILE SET :Src_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Initialize the BLOB Locator */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL SELECT frame INTO :Dest_loc FROM Multimedia_tab
           WHERE Clip_ID = 3 FOR UPDATE;
  /* Opening the BFILE is Mandatory */
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
  /* Opening the BLOB is Optional */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
```

```
  EXEC SQL LOB LOAD :Amount FROM FILE :Src_loc INTO :Dest_loc;
  /* Closing LOBs and BFILEs is Mandatory if they have been OPENed */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;
  /* Release resources held by the Locators */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  loadLOBfromBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Load a LOB with Data from a BFILE Using Visual Basic (OO4O)

```
Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraMyBfile as OraBFile

OraConnection.BeginTrans
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

OraParameters.Add "id", 1,ORAPARAM_INPUT
OraParameters.Add "mybfile", Empty,ORAPARAM_OUTPUT
OraParameters("mybfile").ServerType = ORATYPE_BFILE

OraDatabase.ExecuteSQL ("begin  GetBFile(:id, :mybfile ") end")

Set OraMyBFile = OraParameters("mybfile").Value
'Go to Next row
OraDyn.MoveNext

OraDyn.Edit
'Lets update OraSound1 data with that from the BFILE
OraSound1.CopyFromBFile  OraMyBFile
OraDyn.Update

OraConnection.CommitTrans
```

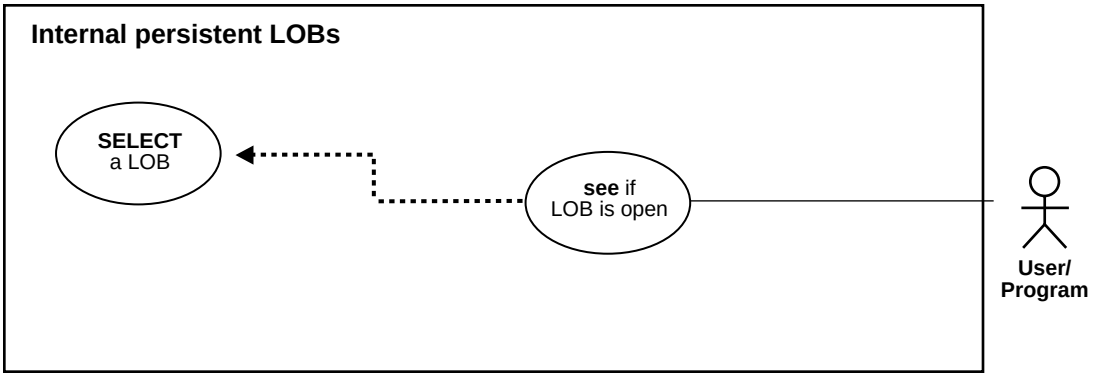
Example: Load a LOB with Data from a BFILE Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_45
{
  public static void main (String args [])
     throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BFILE src_lob = null;
     BLOB dest_lob = null;
     InputStream in = null;
     OutputStream out = null;
     byte buf[] = new byte[1000];
     ResultSet rset = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('AUDIO_DIR', 'Washington_audio') FROM DUAL");
```

```
   if (rset.next())
   {
      src_lob = ((OracleResultSet)rset).getBFILE (1);
      src_lob.openFile();
      in = src_lob.getBinaryStream();
   }

       rset = stmt.executeQuery (
          "SELECT sound FROM multimedia_tab WHERE clip_id = 99 FOR UPDATE");
   if (rset.next())
   {
      dest_lob = ((OracleResultSet)rset).getBLOB (1);

      // Fetch the output stream for dest_lob:
        out = dest_lob.getBinaryOutputStream();
   }

   int length = 0;
   int pos = 0;
   while ((in != null) && (out != null) && ((length = in.read(buf)) != -1))
   {
      System.out.println(
         "Pos = " + Integer.toString(pos) + ".  Length = " +
         Integer.toString(length));
      pos += length;
      out.write(buf, pos, length);
   }

   // Close all streams and file handles:
   in.close();
   out.flush();
   out.close();
   src_lob.closeFile();

   // Commit the transaction:
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  }
}
```

See If a LOB Is Open

Figure 3–17 Use Case Diagram: See If a LOB Is Open



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

- "Example: See If a LOB Is Open Using PL/SQL" on page 3-57
- "Example: See If a LOB Is Open Using C (OCI)" on page 3-57
- "Example: See If a LOB Is Open Using COBOL (Pro*COBOL)" on page 3-59
- "Example: See If a LOB Is Open Using C++ (Pro*C/C++)" on page 3-60
- "Example: See If a LOB Is Open Using Java (JDBC)" on page 3-61

Scenario

The following example opens a Video frame (`Frame`), and then evaluates to see if the `LOB` is open.

Example: See If a LOB Is Open Using PL/SQL

```
/* Note that the example procedure lobIsOpen_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE lobIsOpen_proc IS
   Lob_loc     BLOB;
   Retval      INTEGER;
BEGIN
   SELECT Frame INTO Lob_loc  FROM Multimedia_tab where Clip_ID = 1;

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc , DBMS_LOB.LOB_READONLY);

   /* See if the LOB is open: */
   Retval := DBMS_LOB.ISOPEN(Lob_loc);
  /* The value of Retval will be 1 meaning that the LOB is open. */
END;
```

Example: See If a LOB Is Open Using C (OCI)

```
/* Select the locator into a locator variable */

sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text      *sqlstmt =
    (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
```

```
                                    (ub4) OCI_DEFAULT));

  return (0);
}

void seeIfLOBIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  int isOpen;

  /* Allocate locator resources */
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

  /* Select the locator */
  (void)select_frame_locator(Lob_loc, errhp, svchp, stmthp);

  /* See if the LOB is Open */
  checkerr (errhp, OCILobIsOpen(svchp, errhp, Lob_loc, &isOpen));

  if (isOpen)
  {
    printf("  Lob is Open\n");
    /* ... Processing given that the LOB has already been Opened */
  }
  else
  {
    printf("  Lob is not Open\n");
    /* ... Processing given that the LOB has not been Opened */
  }

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *)  Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: See If a LOB Is Open Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-OPEN.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  BLOB1          SQL-BLOB.
01  LOB-ATTR-GRP.
    05 ISOPN       PIC S9(9) COMP.

01  SRC            SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
LOB-OPEN.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the target BLOB
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
       SELECT FRAME INTO :BLOB1
       FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
    END-EXEC.

* See if the LOB is OPEN
    EXEC SQL
       LOB DESCRIBE :BLOB1 GET ISOPEN INTO :ISOPN
    END-EXEC.

    IF ISOPN = 1
*      <Processing for the LOB OPEN case>
       DISPLAY "The LOB is open"
    ELSE
```

```
*         <Processing for the LOB NOT OPEN case>
             DISPLAY "The LOB is not open"
          END-IF.

         * Free the resources used by the BLOB
          END-OF-BLOB.
              EXEC SQL FREE :BLOB1 END-EXEC.

              EXEC SQL
                 COMMIT WORK RELEASE
              END-EXEC.
              STOP RUN.

          SQL-ERROR.
              EXEC SQL
                 WHENEVER SQLERROR CONTINUE
              END-EXEC.
              DISPLAY " ".
              DISPLAY "ORACLE ERROR DETECTED:".
              DISPLAY " ".
              DISPLAY SQLERRMC.
              EXEC SQL
                 ROLLBACK WORK RELEASE
              END-EXEC.
              STOP RUN.
```

Example: See If a LOB Is Open Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void seeIfLOBIsOpen()
{
  OCIBlobLocator *Lob_loc;
  int isOpen = 1;
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Frame INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* See if the LOB is Open: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN INTO :isOpen;
  if (isOpen)
    printf("LOB is open\n");
  else
    printf("LOB is not open\n");
  /* Note that in this example, the LOB is not open */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeIfLOBIsOpen();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: See If a LOB Is Open Using Visual Basic (OO4O)

Note: An example will be made available in a subsequent release.

Example: See If a LOB Is Open Using Java (JDBC)

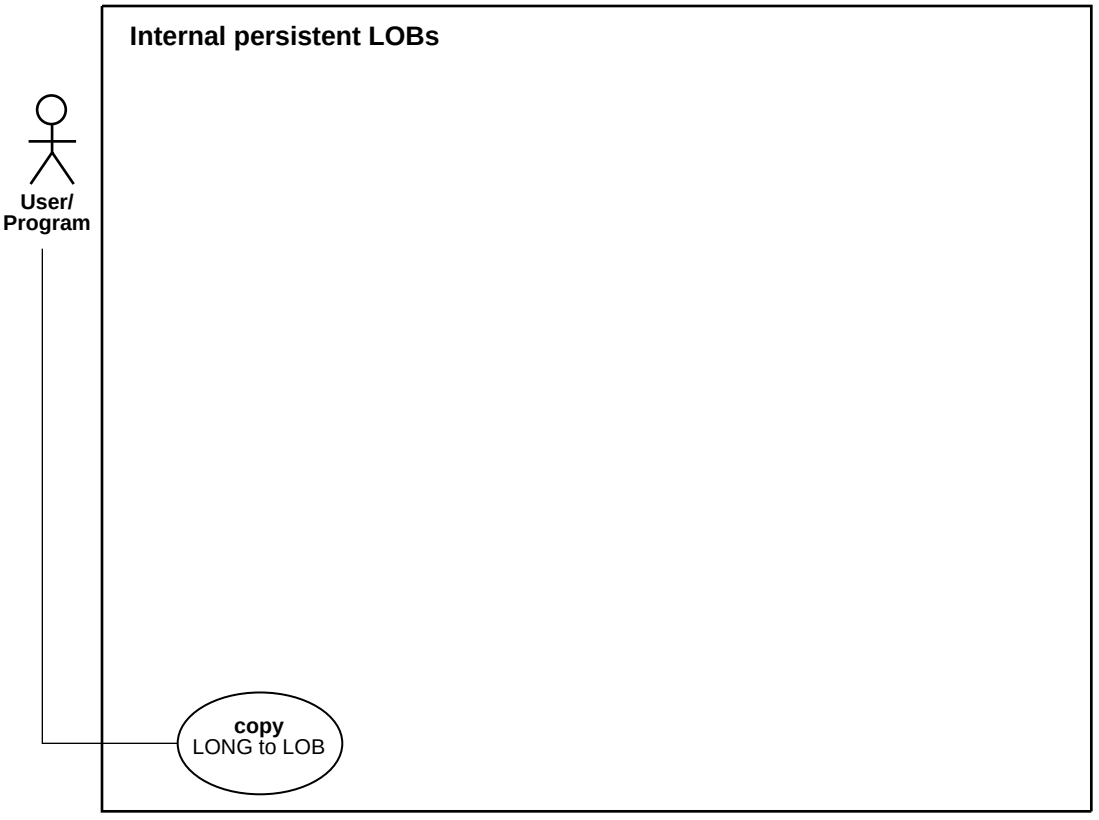
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.Types;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_48
{
  public Ex2_48 ()
  {
  }

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {

      BLOB blob = null;

      ResultSet rset = stmt.executeQuery (
          "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
   if (rset.next())
   {
     blob = ((OracleResultSet)rset).getBLOB (1);
   }

       OracleCallableStatement cstmt =
            (OracleCallableStatement) conn.prepareCall (
            "BEGIN ? := DBMS_LOB.ISOPEN(?); END;");
   cstmt.registerOutParameter (1, Types.NUMERIC);
   cstmt.setBLOB(2, blob);
   cstmt.execute();
   int result = cstmt.getInt(1);
```

```
    System.out.println("The result is: " + Integer.toString(result));

    OracleCallableStatement cstmt2 = (OracleCallableStatement)
        conn.prepareCall (
        "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
    cstmt2.setBLOB(1, blob);
    cstmt2.execute();

    System.out.println("The LOB has been opened with a call to DBMS_LOB.OPEN()");

    // Use the existing cstmt handle to re-query the status of the locator:
    cstmt.setBLOB(2, blob);
    cstmt.execute();
    result = cstmt.getInt(1);
    System.out.println("This result is: " + Integer.toString(result));

    stmt.close();
    cstmt.close();
    cstmt2.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
       e.printStackTrace();
     }
   }
}
```

Copy LONG to LOB

Figure 3–18 Use Case Diagram: Copy LONG to LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

Assume that the following archival source table `SoundsLib_tab` was defined and contains data:

```
CREATE TABLE SoundsLib_tab
(
   Id             NUMBER,
   Description    VARCHAR2(30),
   SoundEffects   LONG RAW
);
```

The example assumes that you want to copy the data from the LONG RAW column (SoundEffects) into the BLOB column (Sound) of the multimedia table, and uses the SQL function TO_LOB to accomplish this.

For more information see:

- *Oracle8i SQL Reference* for syntax of the function TO_LOB.

Example: Copy Long to LOB Using SQL

```
INSERT INTO Multimedia_tab (clip_id,sound) SELECT id, TO_LOB(SoundEffects)
   FROM SoundsLib_tab WHERE id =1;
```

Note: in order for the above to succeed, execute:

```
CREATE TABLE SoundsLib_tab (
   id             NUMBER,
   SoundEffects   LONG RAW);
```

This functionality is based on using an operator on LONGs called TO_LOB that converts the LONG to a LOB. The TO_LOB operator copies the data in all the rows of the LONG column to the corresponding LOB column, and then lets you apply the LOB functionality to what was previously LONG data. Note that the type of data that is stored in the LONG column must match the type of data stored in the LOB. For example, LONG RAW data must be copied to BLOB data, and LONG data must be copied to CLOB data.

Once you have completed this one-time only operation and are satisfied that the data has been copied correctly, you could then drop the LONG column. However, this will not reclaim all the storage originally required to store LONGs in the table. In order to avoid unnecessary, excessive storage, you are better advised to copy the LONG data to a LOB in a new or different table. Once you have made sure that the data has been accurately copied, you should then drop the original table.

One simple way to effect this transposing of LONGs to LOBs is to use the CREATE TABLE... SELECT statement, using the TO_LOB operator on the LONG column as part of the SELECT statement. You can also use INSERT... SELECT.

In the examples in the following procedure, the LONG column named LONG_COL in table LONG_TAB is copied to a LOB column named LOB_COL in table LOB_TAB. These tables include an ID column that contains identification numbers for each row in the table.

Complete the following steps to copy data from a LONG column to a LOB column:

1. Create a new table with the same definition as the table that contains the LONG column, but use a LOB datatype in place of the LONG datatype.

   For example, if you have a table with the following definition:

   ```
   CREATE TABLE Long_tab (
      id       NUMBER,
      long_col LONG);
   ```

   Create a new table using the following SQL statement:

   ```
   CREATE TABLE Lob_tab (
      id       NUMBER,
      blob_col BLOB);
   ```

   Note: When you create the new table, make sure you preserve the table's schema, including integrity constraints, triggers, grants, and indexes. The TO_LOB operator only copies data; it does not preserve the table's schema.

2. Issue an INSERT command using the TO_LOB operator to insert the data from the table with the LONG datatype into the table with the LOB datatype.

   For example, issue the following SQL statement:

   ```
   INSERT INTO Lob_tab
       SELECT id,
       TO_LOB(long_col)
       FROM long_tab;
   ```

3. When you are certain that the copy was successful, drop the table with the LONG column.

   For example, issue the following SQL command to drop the LONG_TAB table:

```
DROP TABLE Long_tab;
```

4. Create a synonym for the new table using the name of the table with LONG data. The synonym ensures that your database and applications continue to function properly.

   For example, issue the following SQL statement:

   ```
   CREATE SYNONYM Long_tab FOR Lob_tab;
   ```

Once the copy is complete, any applications that use the table must be modified to use the LOB data.

You can use the TO_LOB operator to copy the data from the LONG to the LOB in statements that employ CREATE TABLE...AS SELECT or INSERT...SELECT. In the latter case, you must have already ALTERed the table and ADDed the LOB column prior to the UPDATE. If the UPDATE returns an error (because of lack of undo space), you can incrementally migrate LONG data to the LOB using the WHERE clause. The WHERE clause cannot contain functions on the LOB but can test the LOB's nullness.

Note that use of TO_LOB is subject to the following limitations:

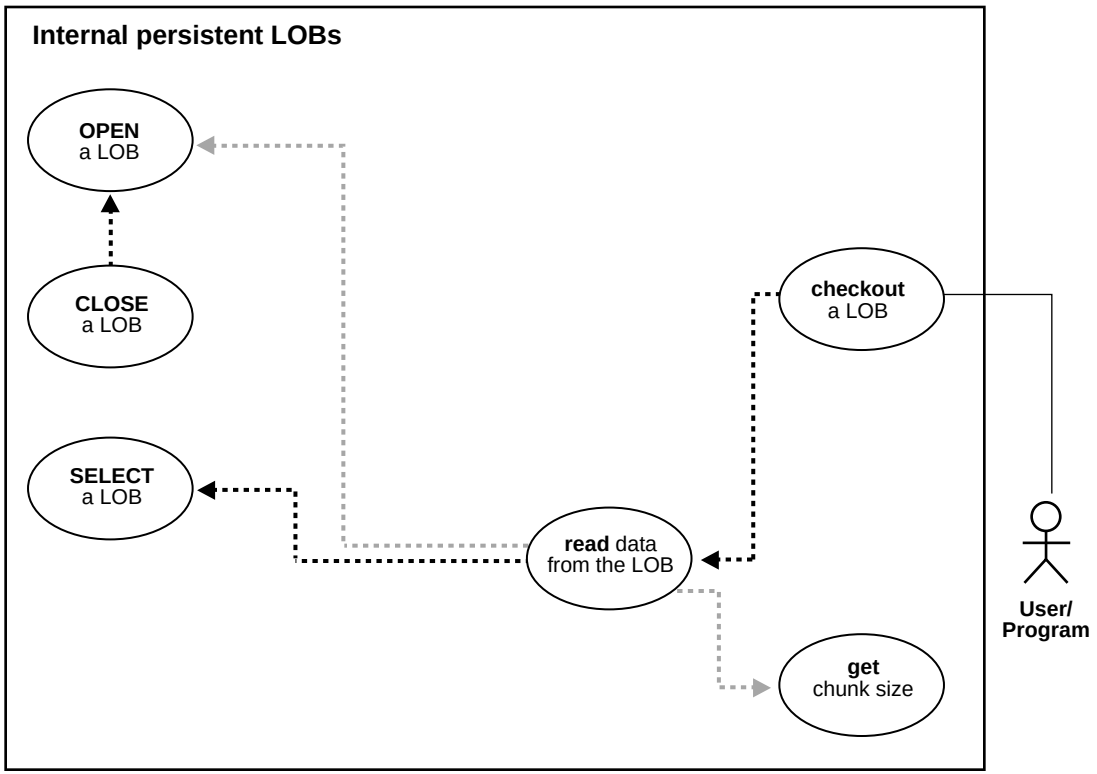
- You can use TO_LOB to copy data to a LOB column, but not to a LOB attribute.
- You cannot use TO_LOB with any remote table. Consequently, all the following statements will fail:

  ```
  INSERT INTO tb1@dblink (lob_col) SELECT TO_LOB(long_col) FROM tb2;
  INSERT INTO tb1 (lob_col) SELECT TO_LOB(long_col) FROM tb2@dblink;
  CREATE table tb1 AS SELECT TO_LOB(long_col) FROM tb2@dblink;
  ```

- If the target table (the table with the lob column) has a trigger — such as BEFORE INSERT or INSTEAD OF INSERT — the :NEW.lob_col variable can't be referenced in the trigger body.
- You cannot deploy TO_LOB inside any PL/SQL block.

Checkout a LOB

Figure 3–19 Use Case Diagram: Checkout a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Streaming Mechanism

The most efficient way to write large amounts of LOB data is to use `OCILobRead`() with the streaming mechanism enabled via polling or a callback. You should use the

OCI or PRO*C interface with streaming for the underlying read operation; using DBMS_LOB.READ will result in non-optimal performance.

Scenario

In the typical use of the checkout-checkin operation, the user wants to checkout a version of the LOB from the database to the client, modify the data on the client without accessing the database, and then in one fell swoop, checkin all the modifications that were made to the document on the client side.

Here we portray the checkout portion of the scenario: the code lets the user read the CLOB Transcript from the nested table InSeg_ntab which contains interview segments for the purpose of processing it in some text editor on the client. The checkin portion of the scenario is described in "Checkin a LOB" on page 3-79.

Example: CheckOut a LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure checkOutLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkOutLOB_proc IS
    Lob_loc     CLOB;
    Buffer      VARCHAR2(32767);
    Amount      BINARY_INTEGER := 32767;
    Position    INTEGER := 2147483647;
BEGIN
   /* Select the LOB: */
   SELECT Intab.Transcript INTO Lob_loc
       FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
          WHERE Mtab.Clip_ID = 1) Intab
             WHERE Intab.Segment = 1;
   * Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   LOOP
```

```
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Process the buffer: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

Example: CheckOut a LOB Using C (OCI)

```
/* This example will READ the entire contents of a BLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire BLOB has been read: */

#define MAXBUFLEN 32767

/* Select the locator into a locator variable: */
sb4 select_transcript_locator(Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *) "SELECT Intab.Transcript \
           FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
              WHERE Mtab.Clip_ID = 1) Intab \
                  WHERE Intab.Segment = 1";
  OCIDefine *defnp1;

 checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                 (ub4)strlen((char *)sqlstmt),
                                 (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

 checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                 (dvoid *)&Lob_loc, (sb4)0,
                                 (ub2) SQLT_CLOB,(dvoid *) 0,
                                 (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
 checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
```

```
                                (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                (ub4) OCI_DEFAULT));

 return 0;
}

void checkoutLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  boolean done;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB,(size_t) 0, (dvoid **) 0);
  /* Select the BLOB: */
  printf(" select the transcript locator...\n");
  select_transcript_locator(Lob_loc, errhp, stmthp, svchp);

  /* Open the CLOB: */
  printf (" open lob in checkOutLOB_proc\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  /* Setting amt = 0 will read till the end of LOB: */
  amt = 0;
  buflen = sizeof(bufp);

 /* Process the data in pieces: */
  printf (" read lob in pieces\n");
 offset = 1;
 memset(bufp, '\0', MAXBUFLEN);
 done = FALSE;
 while (!done)
   {
     retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *)bufp,
                         buflen, (dvoid *)0,(sb4 (*)(dvoid *, dvoid *, ub4,
                         ub1)) 0, (ub2) 0, (ub1) SQLCS_IMPLICIT);
```

```
    switch (retval)
      {
      case OCI_SUCCESS:             /* Only one piece or last piece */
    /* Process the data in bufp. amt will give the amount of data just read in
       bufp. This is in bytes for BLOBs and in characters for fixed
       width CLOBS and in bytes for variable width CLOBs */
    done = TRUE;
    break;
       case OCI_ERROR:
         checkerr (errhp, OCI_ERROR);
    done = TRUE;
    break;
       case OCI_NEED_DATA:          /* There are 2 or more pieces */
    /* Process the data in bufp. amt will give the amount of data just read in
       bufp. This is in bytes for BLOBs and in characters for fixed
       width CLOBS and in bytes for variable width CLOBs. */
    break;
       default:
         checkerr (errhp, retval);
    done = TRUE;
       break;
       } /* while */
   }
  /* Closing the CLOB is mandatory if you have opened it: */
  printf (" close lob in checkOutLOB_proc\n");
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: CheckOut a LOB Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CHECKOUT.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID   PIC X(11) VALUES "USER1/USER1".

       01  CLOB1          SQL-CLOB.
```

```
01  BUFFER         PIC X(5) VARYING.
01  AMT            PIC S9(9) COMP.
01  OFFSET         PIC S9(9) COMP VALUE 1.

01  D-BUFFER-LEN   PIC 9.
01  D-AMT          PIC 9.
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
READ-CLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
       CONNECT :USERID
    END-EXEC.
* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.

    EXEC SQL
       SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
          WHERE CLIP_ID = 2
    END-EXEC.

* Initiate polling read:
    MOVE 0 TO AMT.

* Read first piece of the CLOB into the buffer:
    EXEC SQL
       LOB READ :AMT FROM :CLOB1 AT :OFFSET INTO :BUFFER
    END-EXEC.
    DISPLAY "Reading a CLOB ...".
    DISPLAY " ".
    MOVE BUFFER-LEN TO D-BUFFER-LEN.
    DISPLAY "first read (", D-BUFFER-LEN, "): "
       BUFFER-ARR(1:BUFFER-LEN).

* Read subsequent pieces of the CLOB:
READ-LOOP.
    MOVE "     " TO BUFFER-ARR.
    EXEC SQL
       LOB READ :AMT FROM :CLOB1 INTO :BUFFER
    END-EXEC.
    MOVE BUFFER-LEN TO D-BUFFER-LEN.
```

```
           DISPLAY "next read (", D-BUFFER-LEN, "): "
              BUFFER-ARR(1:BUFFER-LEN).

           GO TO READ-LOOP.

         * Read the last piece of the CLOB:
          END-OF-CLOB.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :CLOB1 END-EXEC.
           MOVE BUFFER-LEN TO D-BUFFER-LEN.
           DISPLAY "last read (", D-BUFFER-LEN, "): "
              BUFFER-ARR(1:BUFFER-LEN).
           EXEC SQL
              COMMIT WORK RELEASE
           END-EXEC.

          SQL-ERROR.
           EXEC SQL
              WHENEVER SQLERROR CONTINUE
           END-EXEC.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED:".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
              ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

Example: CheckOut a LOB Using C++ (Pro*C/C++)

```
/* This example will READ the entire contents of a CLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire CLOB has been read: */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
```

```
}

#define BufferLength 256

void checkOutLOB_proc()
{
  OCIClobLocator *Lob_loc;
  int Amount;
  int Clip_ID, Segment;
  VARCHAR Buffer[BufferLength];

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;

  /* Use Dynamic SQL to retrieve the LOB: */
  EXEC SQL PREPARE S FROM
    'SELECT Intab.Transcript \
       FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
          WHERE Mtab.Clip_ID = :cid) Intab \
             WHERE Intab.Segment = :seg';
  EXEC SQL DECLARE C CURSOR FOR S;
  Clip_ID = Segment = 1;
  EXEC SQL OPEN C USING :Clip_ID, :Segment;
  EXEC SQL FETCH C INTO :Lob_loc;
  EXEC SQL CLOSE C;

  /* Open the LOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;

  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;

  /* Set the maximum size of the Buffer: */
  Buffer.len = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the LOB into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      printf("Checkout %d characters\n", Buffer.len);
    }
  printf("Checkout %d characters\n", Amount);

  /* Closing the LOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
```

```
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  checkOutLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: CheckOut a LOB Using Visual Basic (OO4O)

```
'Note that this code fragment assumes an orablob object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1. There are two ways
'of reading a lob using  orablob.read or orablob.copytofile

'Using OraBlob.Read mechanism
Dim OraDyn as OraDynaset, OraSound as OraBlob, amount_read%, chunksize%, chunk

chunksize = 32767
set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
set OraSound = OraDyn.Fields("Sound")
OraSound.PollingAmount = OraSound.Size 'Read entire BLOB contents
Do
    amount_read = OraSound.Read(chunk,chunksize) 'chunk returned is a variant of
type byte array
    If amount_read = 0 Then
        Exit Do
    End If
    OraMusic.offset = OraSound.offset + amount_read + 1
Loop Until amount_read = 0


'Using OraBlob.CopyToFile mechanism
Dim OraDyn as OraDynaset, OraSound as OraBlob, amount_read%, chunksize%, chunk

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

OraSound.pollingAmount = OraSound.Size
'Read entire BLOB contents
```

```
OraSound.CopyToFile "c:\mysound.aud"
```

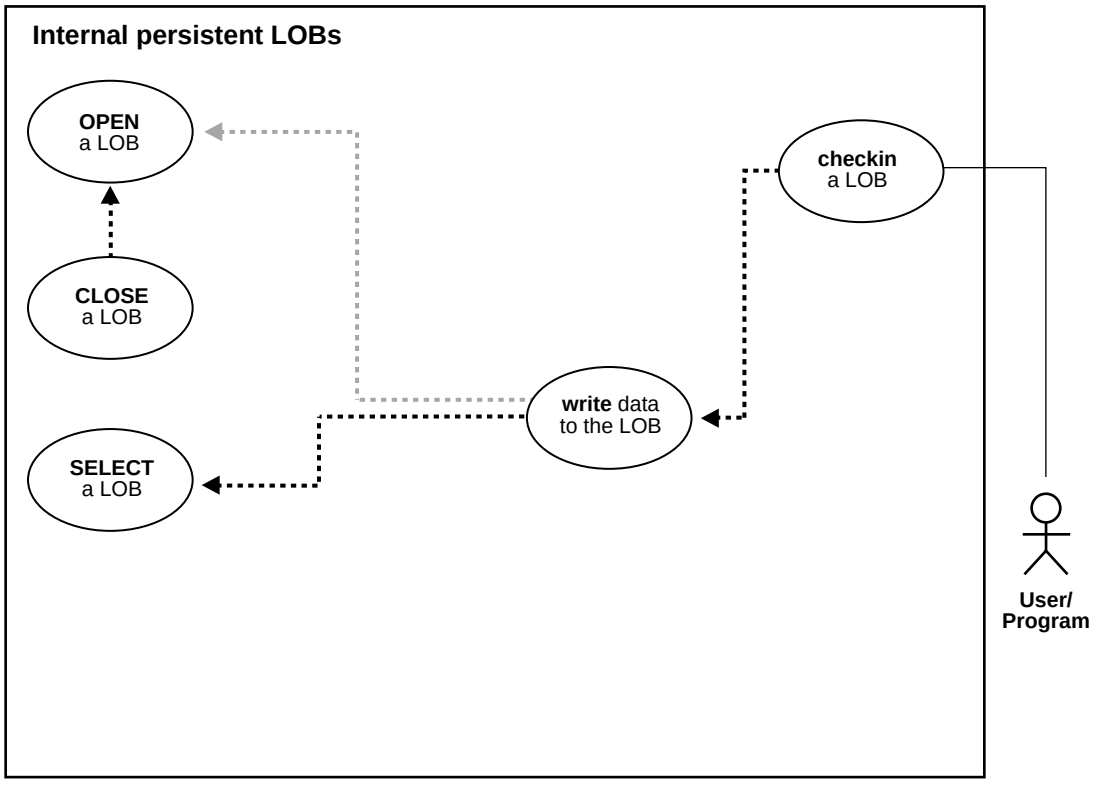
Example: CheckOut a LOB Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_59
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
     throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
       DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       CLOB src_lob = null;
```

```
        InputStream in = null;
        byte buf[] = new byte[MAXBUFSIZE];

        ResultSet rset = stmt.executeQuery (
           "SELECT intab.transcript FROM TABLE(
              SELECT mtab.inseg_ntab FROM multimedia_tab mtab
                 WHERE mtab.clip_id = 1) intab WHERE intab.segment = 1");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getCLOB (1);
           in = src_lob.getAsciiStream();
     }

     int length = 0;
     int pos = 0;
     while ((in != null) && ((length = in.read(buf)) != -1))
     {
        pos += length;
        System.out.println(Integer.toString(pos));
        // Process the buffer:
     }

     in.close();
     stmt.close();
     conn.commit();
     conn.close();
      }
      catch (SQLException e)
      {
         e.printStackTrace();
      }
    }
}
```

Checkin a LOB

Figure 3–20 Use Case Diagram: Checkin a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Streaming Mechanism

The most efficient way to write large amounts of LOB data is to use `OCILobWrite()` with the streaming mechanism enabled via polling or a callback

Scenario

The `checkin` operation demonstrated here follows from "Checkout a LOB" on page 3-68. In this case, the procedure writes the data back into the `CLOB Transcript` column within the nested table `InSeg_ntab` that contains interview segments. As noted above, you should the OCI or PRO*C interface with streaming for the underlying write operation; using `DBMS_LOB.WRITE` will result in non-optimal performance.

Example: Checkin a LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure checkInLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkInLOB_proc IS
    Lob_loc         CLOB;
    Buffer          VARCHAR2(32767);
    Amount          BINARY_INTEGER := 32767;
    Position        INTEGER := 2147483647;
    i               INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT Intab.Transcript INTO Lob_loc
        FROM TABLE(SELECT m.InSeg_ntab FROM Multimedia_tab Mtab
                   WHERE Clip_ID = 2) Intab
        WHERE Intab.Segment = 1
        FOR UPDATE;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE)
    FOR i IN 1..3 LOOP
        /* Fill the Buffer with data to be written. */
        /* Write data: */
        DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
        Position := Position + Amount;
    END LOOP;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Checkin a LOB Using C (OCI)

```
/* This example demonstrates how OCI provides for the ability to write
   arbitrary amounts of data to an Internal LOB in either a single piece
   of in multiple pieces using a streaming mechanism that utilizes standard
   polling.  A statically allocated Buffer is used to hold the data being
   written to the LOB. */

#define MAXBUFLEN 32767

/* Select the locator into a locator variable */
sb4 select_lock_transcript_locator(Lob_loc, errhp, stmthp,svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
     (text *) "SELECT Intab.Transcript \
           FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
              WHERE Mtab.Clip_ID = 2) Intab \
                 WHERE Intab.Segment = 1 FOR UPDATE";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

     return OCI_SUCCESS;
}

void checkinLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
```

```
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIClobLocator *Lob_loc;
  ub4 Total = 2.5*MAXBUFLEN;
  ub4 amtp;
  ub4 offset;
  ub4 remainder;
  ub4 nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sb4 err;

   /* Allocate locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB,(size_t) 0,(dvoid **) 0);
  /* Select the CLOB */
  printf(" select the transcript locator...\n");
  select_lock_transcript_locator(Lob_loc, errhp, stmthp, svchp);

  /* Open the CLOB */
  printf (" open the locator.\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));


  printf (" write the lob in pieces\n");
  if (Total > MAXBUFLEN)
    nbytes = MAXBUFLEN;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                    /* Only a single write is required */

  /* Fill the buffer with nbytes worth of data */

  remainder = Total - nbytes;

  /* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */
  amtp = 0;

  /* offset = <Starting position where to begin writing the data>; */
  offset = 1;

  if (0 == remainder)
  {
    amtp = nbytes;
    /* Here, (Total <= MAXBUFLEN ) so we can write in one piece */
```

```
    checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, amtp,
                                  offset, bufp, nbytes,
                                  OCI_ONE_PIECE, (dvoid *) 0,
                                  (sb4 (*)(dvoid *,dvoid *,ub4 *,ub1 *)) 0,
                                  0, SQLCS_IMPLICIT));
}
else
{
  /* Here (Total > MAXBUFLEN ) so we use streaming via standard polling */
  /* write the first piece.  Specifying first initiates polling. */
  err = OCILobWrite (svchp, errhp, Lob_loc, &amtp, offset, bufp, nbytes,
                     OCI_FIRST_PIECE, (dvoid *) 0,
                     (sb4 (*)(dvoid *,dvoid *,ub4 *,ub1 *)) 0,
                     0, SQLCS_IMPLICIT);
  if (err != OCI_NEED_DATA)
    checkerr (errhp, err);

  last = FALSE;
  /* write the next (interim) and last pieces */
  do
  {
    if (remainder > MAXBUFLEN)
      nbytes = MAXBUFLEN;     /* Still have more pieces to go */
    else
    {
      nbytes = remainder;     /* Here, (remainder <= MAXBUFLEN) */
      last = TRUE;            /* This is going to be the Final piece */
    }

    /* Fill the buffer with nbytes worth of data */

    if (last)
    {
      /* Specifying LAST terminates polling */
      err = OCILobWrite (svchp, errhp, Lob_loc, &amtp,
                         offset, bufp, nbytes,
                         OCI_LAST_PIECE, (dvoid *) 0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_SUCCESS)
        checkerr(errhp, err);
    }
    else
    {
      err = OCILobWrite (svchp, errhp, Lob_loc, &amtp,
```

```
                              offset, bufp, nbytes,
                              OCI_NEXT_PIECE, (dvoid *) 0,
                              (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                              0, SQLCS_IMPLICIT);
            if (err != OCI_NEED_DATA)
              checkerr (errhp, err);
          }
          /* Determine how much is left to write */
          remainder = remainder - nbytes;
        } while (!last);
      }

      /* At this point, (remainder == 0) */

      /* Closing the BLOB is mandatory if you have opend it */
      checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

      /* Free resources held by the locators*/
      (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

    }
```

Example: Checkin a LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKIN.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT INFILE
      ASSIGN TO "datfile.dat"
      ORGANIZATION IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.

FD INFILE
   RECORD CONTAINS 80 CHARACTERS.
01 INREC      PIC X(80).

WORKING-STORAGE SECTION.
01  USERID   PIC X(11) VALUES "USER1/USER1".
01  CLOB1          SQL-CLOB.
01  BUFFER         PIC X(80) VARYING.
01  AMT            PIC S9(9) COMP VALUE 0.
```

```
01  OFFSET         PIC S9(9) COMP VALUE 1.
01  END-OF-FILE    PIC X(1) VALUES "N".

01  D-BUFFER-LEN   PIC 9.
01  D-AMT          PIC 9.
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.

WRITE-CLOB.
    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL CONNECT :USERID END-EXEC.

* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.

    EXEC SQL
         SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
         WHERE CLIP_ID = 1 FOR UPDATE
    END-EXEC.

* Open the input file for reading:

    OPEN INPUT INFILE.

* Either write entire record or write first piece.
* Read a data file here and populate BUFFER-ARR and BUFFER-LEN.
* END-OF-FILE will be set to "Y" when the entire file has been
* read.
    PERFORM READ-NEXT-RECORD.
    MOVE INREC TO BUFFER-ARR.
    MOVE 80 TO BUFFER-LEN.
    IF (END-OF-FILE = "Y")
       MOVE 80 TO AMT
       EXEC SQL
            LOB WRITE ONE :AMT FROM :BUFFER
            INTO :CLOB1 AT :OFFSET
       END-EXEC
    ELSE
       DISPLAY "LOB WRITE FIRST"
       DISPLAY  BUFFER-ARR
       MOVE 321 TO AMT
       EXEC SQL
          LOB WRITE FIRST :AMT FROM :BUFFER INTO :CLOB1
       END-EXEC
```

```
    END-IF.

* Continue reading from the input data file
* and writing to the CLOB:
    PERFORM READ-WRITE
       UNTIL END-OF-FILE = "Y".
    PERFORM SIGN-OFF.
    STOP RUN.

READ-WRITE.
    PERFORM READ-NEXT-RECORD.
    MOVE INREC TO BUFFER-ARR.
    DISPLAY "READ-WRITE".
    DISPLAY INREC.
    MOVE 80 TO BUFFER-LEN.
    IF END-OF-FILE = "Y"
        DISPLAY "LOB WRITE LAST: ", BUFFER-ARR
        MOVE 1 TO BUFFER-LEN
        EXEC SQL
            LOB WRITE LAST :AMT FROM :BUFFER INTO :CLOB1
        END-EXEC
    ELSE
        DISPLAY "LOB WRITE NEXT: ", BUFFER-ARR
        MOVE 0 TO AMT
        EXEC SQL
            LOB WRITE NEXT :AMT FROM :BUFFER INTO :CLOB1
        END-EXEC
    END-IF.

READ-NEXT-RECORD.
    MOVE SPACES TO INREC.
    READ INFILE NEXT RECORD
         AT END
          MOVE "Y" TO END-OF-FILE.

SIGN-OFF.
    CLOSE INFILE.
    EXEC SQL FREE :CLOB1 END-EXEC.

    EXEC SQL COMMIT WORK RELEASE END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
```

```
END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

Example: Checkin a LOB Using C++ (Pro*C/C++)

```
/* This example demonstrates how Pro*C/C++ provides for the ability to WRITE
   arbitrary amounts of data to an Internal LOB in either a single piece
   or in multiple pieces using a Streaming Mechanism that utilizes standard
   polling. A static Buffer is used to hold the data being written: */

#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 512

void checkInLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Lob_loc;
  VARCHAR Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
```

```
EXEC SQL SELECT Story INTO :Lob_loc
          FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
/* Open the LOB: */
EXEC SQL LOB OPEN :Lob_loc READ WRITE;
Total = Amount = (multiple * BufferLength);
if (Total > BufferLength)
  nbytes = BufferLength;   /* We will use streaming via standard polling */
else
  nbytes = Total;                     /* Only a single WRITE is required */
/* Fill the Buffer with nbytes worth of data: */
memset((void *)Buffer.arr, 32, nbytes);
Buffer.len = nbytes;       /* Set the Length */
remainder = Total - nbytes;
if (0 == remainder)
  {
    /* Here, (Total <= BufferLength) so we can WRITE in ONE piece: */
    EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write ONE Total of %d characters\n", Amount);
  }
else
  {
    /* Here (Total > BufferLength) so use streaming via standard polling:
       WRITE the FIRST piece.  Specifying FIRST initiates polling: */
    EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
    printf("Write FIRST %d characters\n", Buffer.len);
    last = FALSE;
    /* WRITE the NEXT (interim) and LAST pieces: */
    do
      {
        if (remainder > BufferLength)
          nbytes = BufferLength;         /* Still have more pieces to go */
        else
          {
            nbytes = remainder;
            last = TRUE;          /* This is going to be the Final piece */
          }
        /* Fill the Buffer with nbytes worth of data: */
        memset((void *)Buffer.arr, 32, nbytes);
        Buffer.len = nbytes;       /* Set the Length */
        if (last)
          {
            EXEC SQL WHENEVER SQLERROR DO Sample_Error();
            /* Specifying LAST terminates polling: */
            EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Lob_loc;
            printf("Write LAST Total of %d characters\n", Amount);
```

```
            }
          else
            {
              EXEC SQL WHENEVER SQLERROR DO break;
              EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Lob_loc;
              printf("Write NEXT %d characters\n", Buffer.len);
            }
          /* Determine how much is left to WRITE: */
          remainder = remainder - nbytes;
        } while (!last);
    }
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* At this point, (Amount == Total), the total amount that was written */
  /* Close the LOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  checkInLOB_proc(1);
  EXEC SQL ROLLBACK WORK;
  checkInLOB_proc(4);
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Checkin a LOB Using Visual Basic (OO4O)

```
'Note that this code fragment assumes an orablob object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1. there are two ways
'of writing a lob using  orablob.write or orablob.copyfromfile

'Using OraBlob.Write mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim fnum As Integer
Dim OraDyn As OraDynaset, OraSound As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
```

```
chunksize = 500
ReDim curchunk(chunksize)
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

fnum = FreeFile

Open "c:\tmp\washington_audio" For Binary As #fnum
OraSound.offset = 1
OraSound.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

 Dim piece As Byte
 Get #fnum, , curchunk

 OraDyn.Edit

 piece = ORALOB_FIRST_PIECE
 amount_written = OraSound.Write(curchunk, chunksize, ORALOB_FIRST_PIECE)

 While OraSound.Status = ORALOB_NEED_DATA
    remainder = remainder - chunksize
    If amount_written <= chunksize Then
        chunksize = remainder
        piece = ORALOB_LAST_PIECE
    Else
        piece = ORALOB_NEXT_PIECE
    End If

    Get #fnum, , curchunk
    amount_written = OraSound.Write(curchunk, chunksize, piece)
 Wend

OraDyn.Update

'Using OraBlob.CopyFromFile mechanism
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab order by clip_
id", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value

OraDyn.Edit
OraSound.CopyFromFile "c:\tmp\washington_audio"
OraDyn.Update
```

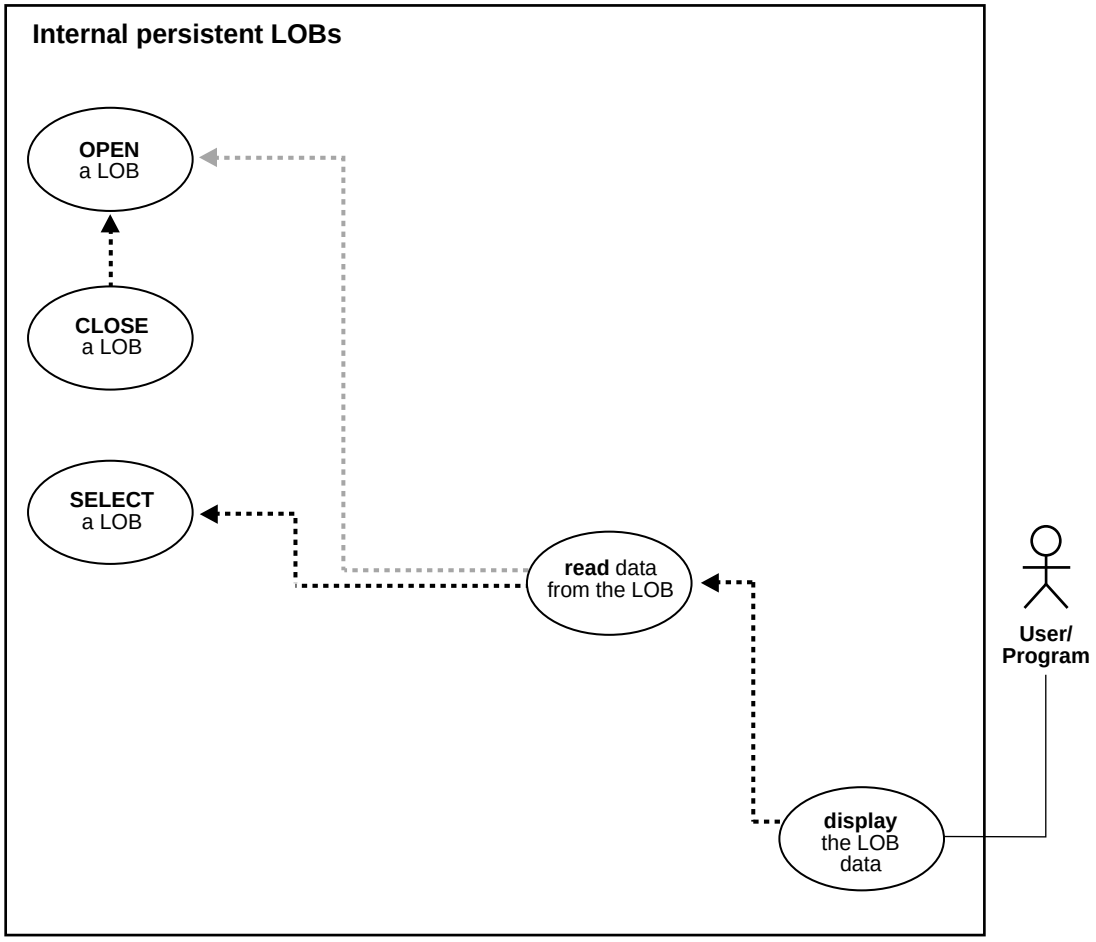
Example: Checkin a LOB Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_66
{
  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     CLOB lob_loc = null;
     String buf = new String ("Some Text To Write");

     ResultSet rset = stmt.executeQuery (
        "SELECT story FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
```

```
    if (rset.next())
    {
      lob_loc = ((OracleResultSet)rset).getCLOB (1);
    }

    long pos = 0;       // Offset within the CLOB where the data is to be written
    long length = 0;    // This is the size of the buffer to be written

    // This loop writes the buffer three times consecutively:
    for (int i = 0; i < 3; i++)
    {
       pos = lob_loc.length();

       // an alternative is: lob_loc.putString(pos, buf);
       lob_loc.putChars(pos, buf.toCharArray());

       // Some debug information:
       System.out.println(" putChars(" + Long.toString(pos) + ",
          buf.toCharArray());");
    }

    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

Display the LOB Data

Figure 3–21 Use Case Diagram: Display the LOB data



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Streaming Mechanism

The most efficient way to read large amounts of LOB data is to use `OCILobRead`() with the streaming mechanism enabled.

Scenario

As an example of displaying a LOB, our scenario stream-reads the image `Drawing` from the column object `Map_obj` onto the client-side in order to view the data.

- "Example: Display the LOB Data Using PL/SQL" on page 3-94
- "Example: Display the LOB Data Using C (OCI)" on page 3-95
- "Example: Display the LOB Data Using COBOL (Pro*COBOL)" on page 3-97
- "Example: Display the LOB Data Using C++ (Pro*C/C++)" on page 3-99
- "Example: Display the LOB Data Using Visual Basic (OO4O)" on page 3-100
- "Example: Display the LOB Data Using Java (JDBC)" on page 3-101

Example: Display the LOB Data Using PL/SQL

```
/* Note that the example procedure displayLOB_proc is not part of the
DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE displayLOB_proc IS
Lob_loc  BLOB;
Buffer   RAW(1024);
Amount   BINARY_INTEGER := 1024;
Position INTEGER := 1;
BEGIN
   /* Select the LOB: */
   SELECT m.Map_obj.Drawing INTO Lob_loc
   FROM Multimedia_tab m WHERE m.Clip_ID = 1;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   LOOP
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Display the buffer contents: */
```

```
      DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

Example: Display the LOB Data Using C (OCI)

```
/* This example will READ the entire contents of a BLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire BLOB has been read. */

#define MAXBUFLEN 32767

/* Select the locator into a locator variable */
sb4 select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
     (text *) "SELECT m.Map_obj.Drawing \
                 FROM Multimedia_tab m WHERE m.Clip_ID = 1";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
```

```
  return 0;
}

void displayLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIBlobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  boolean done;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  OCILobLocator *Lob_Loc;

  /* Allocate the Source (bfile) & destination (blob) locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *) envhp,
                            (dvoid **) &Lob_loc, (ub4)OCI_DTYPE_LOB,
                            (size_t) 0, (dvoid **) 0);
  /* Select the BLOB */
  printf(" select the mapobjectdrawing locator...\n");
  select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB */
  printf(" open the lob\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  /* Setting amt = 0 will read till the end of LOB*/
  amt = 0;
  buflen = sizeof(bufp);

 /* Process the data in pieces */
  printf(" Process the data in pieces\n");
  offset = 1;
  memset(bufp, '\0', MAXBUFLEN);
  done = FALSE;
  while (!done)
  {
    retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                        buflen, (dvoid *)0,
```

```
                        (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                        (ub2) 0, (ub1) SQLCS_IMPLICIT);
      switch (retval)
      {
      case OCI_SUCCESS:           /* Only one piece or last piece*/
        /* Process the data in bufp. amt will give the amount of data just read in
           bufp. This is in bytes for BLOBs and in characters for fixed
           width CLOBS and in bytes for variable width CLOBs
         */
        done = TRUE;
        break;
      case OCI_ERROR:
        checkerr (errhp, retval);
        done = TRUE;
        break;
      case OCI_NEED_DATA:         /* There are 2 or more pieces */
        /* Process the data in bufp. amt will give the amount of data just read in
           bufp. This is in bytes for BLOBs and in characters for fixed
           width CLOBS and in bytes for variable width CLOBs
         */
        break;
      default:
        checkerr (errhp, retval);
        done = TRUE;
        break;
      }
    } /* while */

    /* Closing the BLOB is mandatory if you have opend it */
     printf(" close the lob \n");
     checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

     /* Free resources held by the locators*/
     (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  }
```

Example: Display the LOB Data Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. DISPLAY-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
```

```
01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BLOB1          SQL-BLOB.
01  BUFFER2        PIC X(5) VARYING.
01  AMT            PIC S9(9) COMP.
01  OFFSET         PIC S9(9) COMP VALUE 1.
01  D-AMT          PIC 9.

    EXEC SQL VAR BUFFER2 IS RAW(5) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
DISPLAY-BLOB.
    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
       CONNECT :USERID
    END-EXEC.
* Allocate and initialize the BLOB locator:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

    EXEC SQL
       SELECT M.SOUND INTO :BLOB1
       FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
    END-EXEC.
    DISPLAY "Found column SOUND".
* Initiate polling read:
    MOVE 0 TO AMT.

    EXEC SQL LOB READ :AMT FROM :BLOB1 AT :OFFSET
       INTO :BUFFER2 END-EXEC.
    DISPLAY " ".
    MOVE AMT TO D-AMT.
    DISPLAY "first read (", D-AMT, "): " BUFFER2.

READ-BLOB-LOOP.
    MOVE "     " TO BUFFER2.
    EXEC SQL LOB READ :AMT FROM :BLOB1 INTO :BUFFER2 END-EXEC.
     MOVE AMT TO D-AMT.
     DISPLAY "next read (", D-AMT, "): " BUFFER2.
    GO TO READ-BLOB-LOOP.

END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
```

```
            EXEC SQL FREE :BLOB1 END-EXEC.
            MOVE AMT TO D-AMT.
            DISPLAY "last read (", D-AMT, "): " BUFFER2(1:AMT).
            EXEC SQL
                 COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Display the LOB Data Using C++ (Pro*C/C++)

```
/* This example will READ the entire contents of a BLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire BLOB has been read: */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 32767

void displayLOB_proc()
{
  OCIBlobLocator *Lob_loc;
```

```
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BLOB: */
  EXEC SQL SELECT m.Map_obj.Drawing INTO Lob_loc
           FROM Multimedia_tab m WHERE m.Clip_ID = 1;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;
  /* Set the maximum size of the Buffer: */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the BLOB into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      /* Process (Buffer.Length == BufferLength) amount of Buffer.Data */
    }
  /* Process (Buffer.Length == Amount) amount of Buffer.Data */
  /* Closing the BLOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Display the LOB Data Using Visual Basic (OO4O)

```
'Using OraClob.Read mechanism
Dim MySession As OraSession
```

```
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraStory = OraDyn.Fields("Story").Value
OraStory.PollingAmount = OraStory.Size 'Read entire CLOB contents
Do
   'chunk returned is a variant of type byte array:
    amount_read = OraStory.Read(chunk, chunksize)
    If amount_read = 0 Then
        Exit Do
    End If
    'Display the data here
    OraStory.offset = OraStory.offset + amount_read + 1
Loop Until amount_read = 0
```

Example: Display the LOB Data Using Java (JDBC)

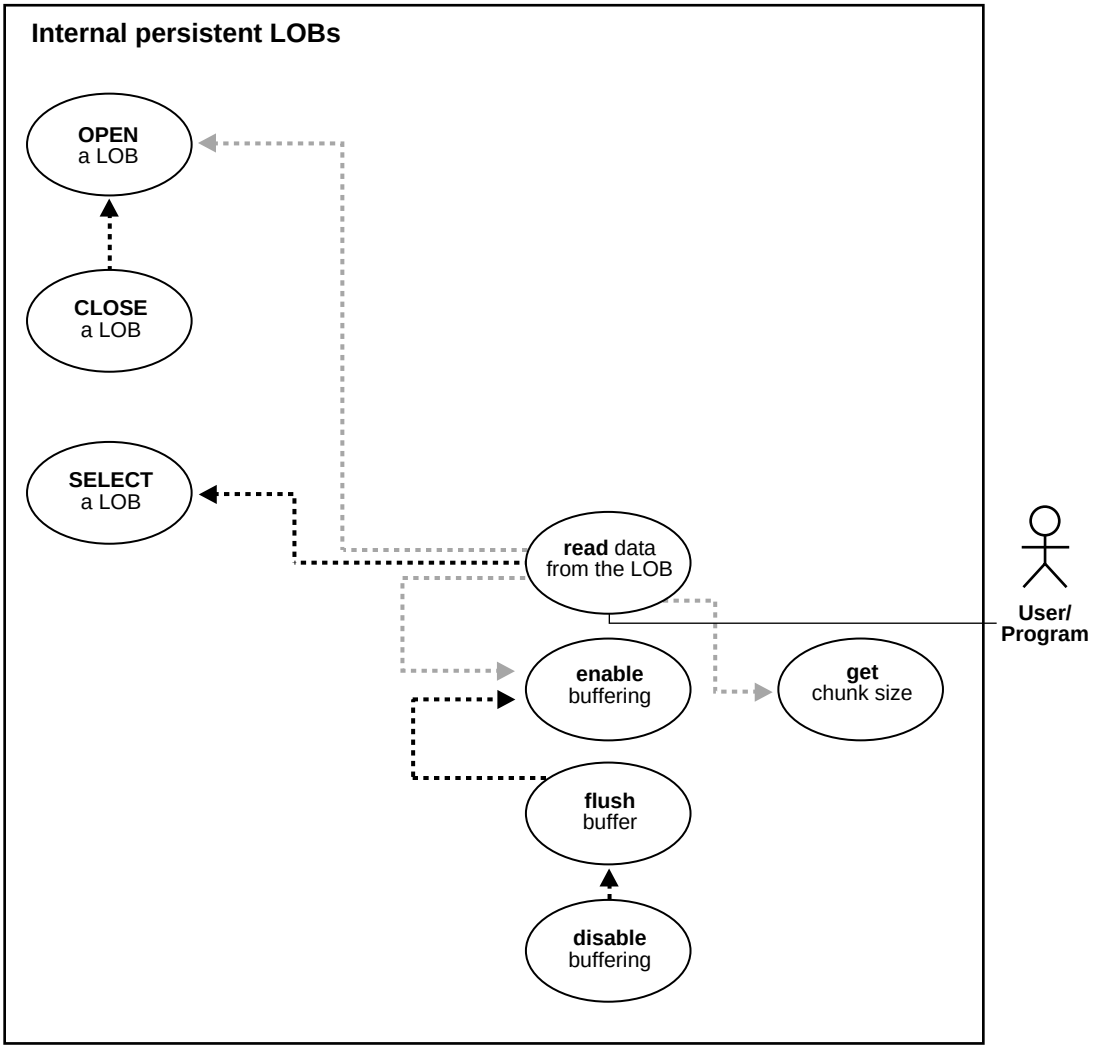
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_72
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
```

```
    throws Exception
{
  // Load the Oracle JDBC driver:
  Class.forName ("oracle.jdbc.driver.OracleDriver");

  // Connect to the database:
  Connection conn =
     DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

  // It's faster when auto commit is off:
  conn.setAutoCommit (false);

  // Create a Statement:
  Statement stmt = conn.createStatement ();

  try
  {
   BLOB lob_loc = null;
   InputStream in = null;
   byte buf[] = new byte[MAXBUFSIZE];
   int pos = 0;
   int length = 0;

   ResultSet rset = stmt.executeQuery (
        "SELECT m.map_obj.drawing FROM multimedia_tab m WHERE m.clip_id = 1");
 if (rset.next())
 {
   lob_loc = ((OracleResultSet)rset).getBLOB (1);
 }

 // read this LOB through an InputStream:
 in = lob_loc.getBinaryStream();

 while ((length = in.read(buf)) != -1)
 {
    pos += length;
    System.out.println(Integer.toString(pos));
    // Process the contents of the buffer here.
 }

 in.close();
 stmt.close();
 conn.commit();
 conn.close();
```

```
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Read Data from the LOB

Figure 3–22 Use Case Diagram: Read data from the LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Stream Read

The most efficient way to read large amounts of LOB data is to use OCILobRead() with the streaming mechanism enabled via polling or a callback.

When reading the LOB value, it is not an error to try to read beyond the end of the LOB. This means that you can always specify an input amount of 4 gigabytes regardless of the starting offset and the amount of data in the LOB. You do need to incur a round-trip to the server to call OCILobGetLength() to find out the length of the LOB value in order to determine the amount to read.

For example, assume that the length of a LOB is 5,000 bytes and you want to read the entire LOB value starting at offset 1,000. Also assume that you do not know the current length of the LOB value. Here's the OCI read call, excluding the initialization of all parameters:

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

When using polling mode, be sure to look at the value of the 'amount' parameter after each OCILobRead() call to see how many bytes were read into the buffer since the buffer may not be entirely full.

When using callbacks, the 'len' parameter, which is input to the callback, will indicate how many bytes are filled in the buffer. Be sure to check the 'len' parameter during your callback processing since the entire buffer may not be filled with data (see the*Oracle Call Interface Programmer's Guide*.).

Chunksize

A chunk is one or more Oracle blocks. You can specify the chunk size for the LOB when creating the table that contains the LOB. This corresponds to the chunk size used by Oracle when accessing or modifying the LOB value. Part of the chunk is used to store system-related information and the rest stores the LOB value. The getchunksize function returns the amount of space used in the LOB chunk to store the LOB value.

You will improve performance if the you execute `read` requests using a multiple of this chunk size. The reason for this is that you're using the same unit that the Oracle database uses when reading data from disk. If it is appropriate for your application, you should batch reads until you have enough for an entire chunk instead of issuing several `LOB` read calls that operate on the same `LOB` chunk.

Scenario

Our example reads the data from a single video Frame.

- "Example: Read Data from a LOB Using PL/SQL (DBMS_LOB Package)" on page 3-106
- "Example: Read Data from a LOB Using PL/SQL (DBMS_LOB Package)" on page 3-106
- "Example: Read Data from a LOB Using COBOL (Pro*COBOL)" on page 3-109
- "Example: Read Data from a LOB Using C++ (Pro*C/C++)" on page 3-111
- "Example: Read Data from a LOB Using Visual Basic (OO4O)" on page 3-112
- "Example: Read Data from a LOB Using Java (JDBC)" on page 3-112

Example: Read Data from a LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure readLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE readLOB_proc IS
    Lob_loc          BLOB;
    Buffer           RAW(32767);
    Amount           BINARY_INTEGER := 32767;
    Position         INTEGER := 1000;
    Chunksize        INTEGER;
BEGIN
    /* Select the LOB: */
   SELECT Frame INTO Lob_loc
      FROM Multimedia_tab
        WHERE Clip_ID = 1;
   /* Find out the chunksize for this LOB column: */
   Chunksize := DBMS_LOB.GETCHUNKSIZE(Lob_loc);
   IF (Chunksize < 32767) THEN
      Amount := (32767 / Chunksize) * Chunksize;
   END IF;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
```

```
   /* Read data from the LOB: */
   DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

Example: Read Data from a LOB Using C (OCI)

```
/* This example will READ the entire contents of a BLOB piecewise into a
   buffer using a standard polling method, processing each buffer piece
   after every READ operation until the entire BLOB has been read. */
#define MAXBUFLEN 32767

/* Select the locator into a locator variable */
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
     (text *) "SELECT Frame \
                FROM Multimedia_tab m WHERE m.Clip_ID = 1";

  printf("  prepare statement in select_frame_locator\n");
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  printf("  OCIDefineByPos in select_frame_locator\n");
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
   printf("  OCIStmtExecute in select_frame_locator\n");
   checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

     return 0;
```

```
}

void readLOB_proc(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;
  boolean done;

  OCILobLocator *Lob_loc;
  OCILobLocator *blob;

  /* Allocate the Source (bfile) & destination (blob) locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  /* Select the BLOB */
  printf(" call select_frame4read_locator\n");
  select_frame_locator(Lob_loc);

  /* Open the BLOB */
  printf(" call OCILobOpen\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, blob, OCI_LOB_READONLY)));

  /* Setting the amt to the buffer length.  Note here that amt is in bytes
     since we are using a BLOB  */
  amt = sizeof(bufp);
  buflen = sizeof(bufp);

 /* Process the data in pieces */
  printf(" process the data in piece\n");
  offset = 1;
  memset(bufp, '\0', MAXBUFLEN);

  retval = OCILobRead(svchp, errhp, Lob_loc, &amt, offset, (dvoid *) bufp,
                      buflen, (dvoid *)0,
                      (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                      (ub2) 0, (ub1) SQLCS_IMPLICIT);
  switch (retval)
    {
```

```
    case OCI_SUCCESS:              /* Only one piece  since amtp == bufp */
      /* Process the data in bufp. amt will give the amount of data just read in
         bufp. This is in bytes for BLOBs and in characters for fixed
         width CLOBS and in bytes for variable width CLOBs */
     break;
    case OCI_ERROR:
      /*   report_error();        this function is not shown here */
      break;
    default:
      (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
       done = TRUE;
       break;
  }

  /* Closing the BLOB is mandatory if you have opend it */
     checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

     /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
}
```

Example: Read Data from a LOB Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ONE-READ-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  BLOB1          SQL-BLOB.
       01  BUFFER2        PIC X(32767) VARYING.
       01  AMT            PIC S9(9) COMP.
       01  OFFSET         PIC S9(9) COMP VALUE 1.
       01  USERID   PIC X(11) VALUES "USER1/USER1".
           EXEC SQL INCLUDE SQLCA END-EXEC.

           EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

       PROCEDURE DIVISION.
       ONE-READ-BLOB.

           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
              CONNECT :USERID
```

```
     END-EXEC.

* Allocate and initialize the CLOB locator:
     EXEC SQL ALLOCATE :BLOB1 END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

     EXEC SQL
        SELECT FRAME INTO :BLOB1
        FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
     END-EXEC.

     EXEC SQL LOB OPEN :BLOB1 END-EXEC.

* Perform a single read:
     MOVE 32767 TO AMT.
     EXEC SQL
        LOB READ :AMT FROM :BLOB1 INTO :BUFFER2
     END-EXEC.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

 END-OF-BLOB.
     DISPLAY "BUFFER2: ", BUFFER2(1:AMT).
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.

     EXEC SQL
        COMMIT WORK RELEASE
     END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
        WHENEVER SQLERROR CONTINUE
     END-EXEC.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED:".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
        ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.
```

Example: Read Data from a LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 32767

void readLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  /* Here (Amount == BufferLength) so only one READ is needed: */
  char Buffer[BufferLength];
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Frame INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  /* Read the BLOB data into the Buffer: */
  EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
  printf("Read %d bytes\n", Amount);
  /* Close the BLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  readLOB_proc();
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Read Data from a LOB Using Visual Basic (OO4O)

```
'Using OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraStory = OraDyn.Fields("Story").Value
OraStory.ChunkSize = chunksize
OraStory.pollingAmount = OraStory.Size
'Read entire CLOB contents
Do
    amount_read = OraStory.Read(chunk)
   'chunk returned is a variant of type byte array
    If amount_read = 0 Then
        Exit Do
    End If
    'Display the data here
    OraStory.offset = OraStory.offset + amount_read + 1
Loop Until amount_read = 0
```

Example: Read Data from a LOB Using Java (JDBC)

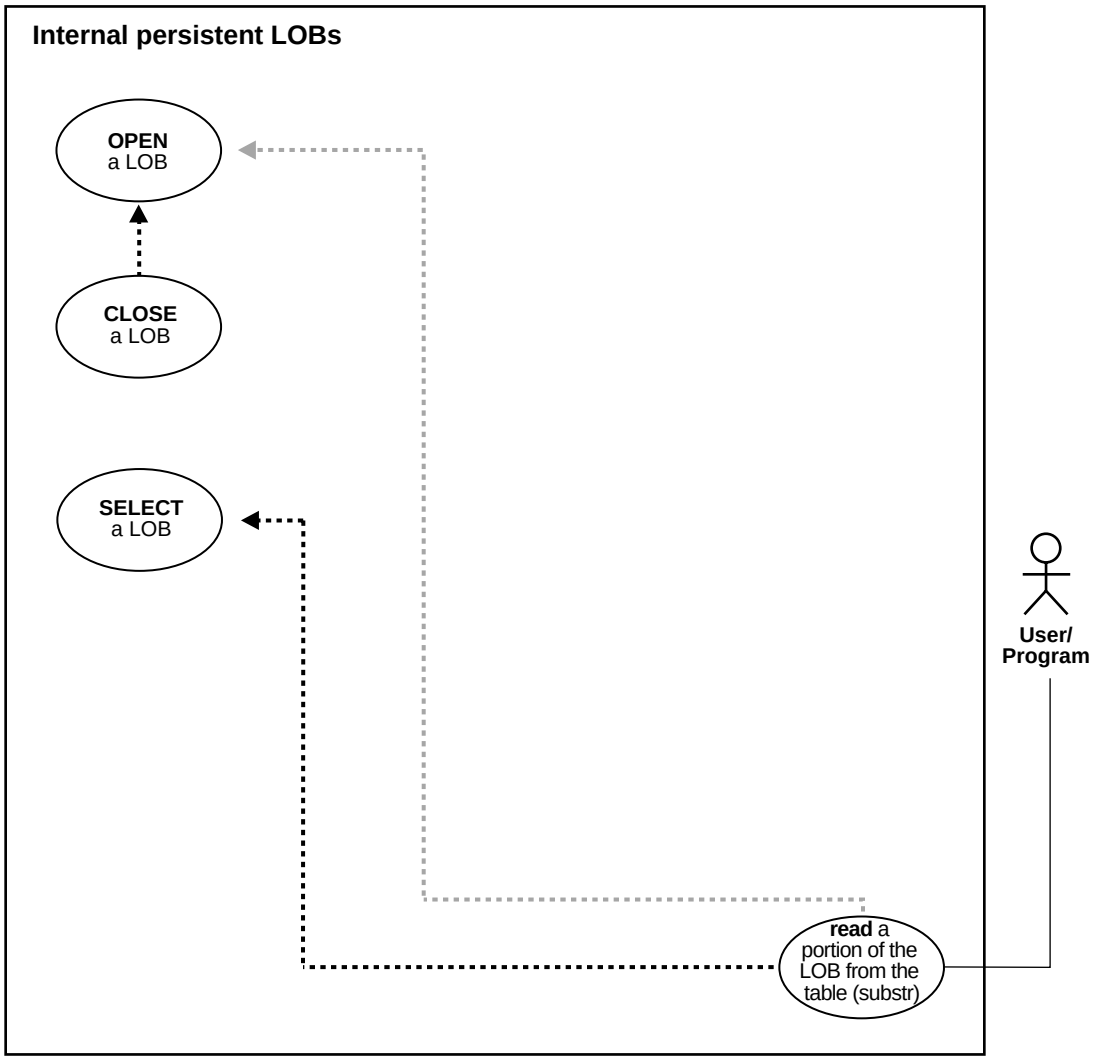
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_79
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BLOB lob_loc = null;
     byte buf[] = new byte[MAXBUFSIZE];

     ResultSet rset = stmt.executeQuery (
          "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
     if (rset.next())
     {
      lob_loc = ((OracleResultSet)rset).getBLOB (1);
     }

     // MAXBUFSIZE is the number of bytes to read and 1000 is the offset from
     // which to start reading
     buf = lob_loc.getBytes(1000, MAXBUFSIZE);

     // Display the contents of the buffer here:
     System.out.println(new String(buf));

     stmt.close();
```

```
        conn.commit();
      conn.close();

      }
      catch (SQLException e)
      {
         e.printStackTrace();
      }
    }
  }
```

Read a Portion of the LOB (substr)

Figure 3–23 Use Case Diagram: Read a portion of the LOB from the Table (substr)



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This example demonstrates reading a portion from sound-effect Sound.

- "Example: Read a Portion of the LOB (substr) Using PL/SQL (DBMS_LOB Package)" on page 3-116
- "Example: Read a Portion of the LOB (substr) Using COBOL (Pro*COBOL)" on page 3-117
- "Example: Read a Portion of the LOB (substr) Using C++ (Pro*C/C++)" on page 3-118
- "Example: Read a Portion of the LOB (substr) Using Visual Basic (OO4O)" on page 3-120
- "Example: Read a Portion of the LOB (substr) Using Java (JDBC)" on page 3-120

Example: Read a Portion of the LOB (substr) Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure substringLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE substringLOB_proc IS
    Lob_loc          BLOB;
    Amount           BINARY_INTEGER := 32767;
    Position         INTEGER := 1024;
    Buffer           RAW(32767);
BEGIN
    /* Select the LOB: */
    SELECT Sound INTO Lob_loc FROM Multimedia_tab
       WHERE Clip_ID = 1;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
    Buffer := DBMS_LOB.SUBSTR(Lob_loc, Amount, Position);
    /* Process the data */
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);
END;

/* In the following SQL statement, 255 is the amount to read
```

```
   and 1 is the starting offset from which to read: */
SELECT DBMS_LOB.SUBSTR(Sound, 255, 1) FROM Multimedia_tab WHERE Clip_ID = 1;
```

Example: Read a Portion of the LOB (substr) Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-SUBSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  BLOB1          SQL-BLOB.
01  BUFFER2        PIC X(32767) VARYING.
01  AMT            PIC S9(9) COMP.
01  POS            PIC S9(9) COMP VALUE 1.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

    EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

PROCEDURE DIVISION.
BLOB-SUBSTR.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
       CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

    EXEC SQL
       SELECT FRAME INTO :BLOB1
       FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
    END-EXEC.
    DISPLAY "Selected the BLOB".

* Open the BLOB for READ ONLY:
    EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.

* Execute PL/SQL to get SUBSTR functionality:
    MOVE 5 TO AMT.
```

```
            EXEC SQL EXECUTE
              BEGIN
                :BUFFER2 := DBMS_LOB.SUBSTR(:BLOB1,:AMT,:POS);
              END;
            END-EXEC.
            EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
            DISPLAY "Substr: ", BUFFER2-ARR(POS:AMT).

        END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BLOB1 END-EXEC.
            EXEC SQL
               COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
               WHENEVER SQLERROR CONTINUE
            END-EXEC.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
               ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Read a Portion of the LOB (substr) Using C++ (Pro*C/C++)

```
/* Pro*C/C++ lacks an equivalent embedded SQL form for the DBMS_LOB.SUBSTR()
   function.  However, Pro*C/C++ can interoperate with PL/SQL using anonymous
   PL/SQL blocks embedded in a Pro*C/C++ program as this example shows: */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

#define BufferLength 32767

void substringLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Position = 1;
  int Amount = BufferLength;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Open the BLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Invoke SUBSTR() from within an anonymous PL/SQL block: */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, :Amount, :Position);
    END;
  END-EXEC;
  /* Close the BLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Process the Data */
  /* Release resources used by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(0);
}
```

Example: Read a Portion of the LOB (substr) Using Visual Basic (OO4O)

```
'Note that reading a portion of a LOB (or BFILE) in OO4O is accomplished by
'setting the OraBlob.Offset and OraBlob.chunksize properties.
'Using OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn as OraDynaset, OraStory as OraClob, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraStory = OraDyn.Fields("Story").Value

'Let's read 100 bytes from the 500th byte onwards:
OraStory.Offset = 500
OraStory.PollingAmount = OraStory.Size 'Read entire CLOB contents
amount_read = OraStory.Read(chunk, 100)
'chunk returned is a variant of type byte array
```

Example: Read a Portion of the LOB (substr) Using Java (JDBC)

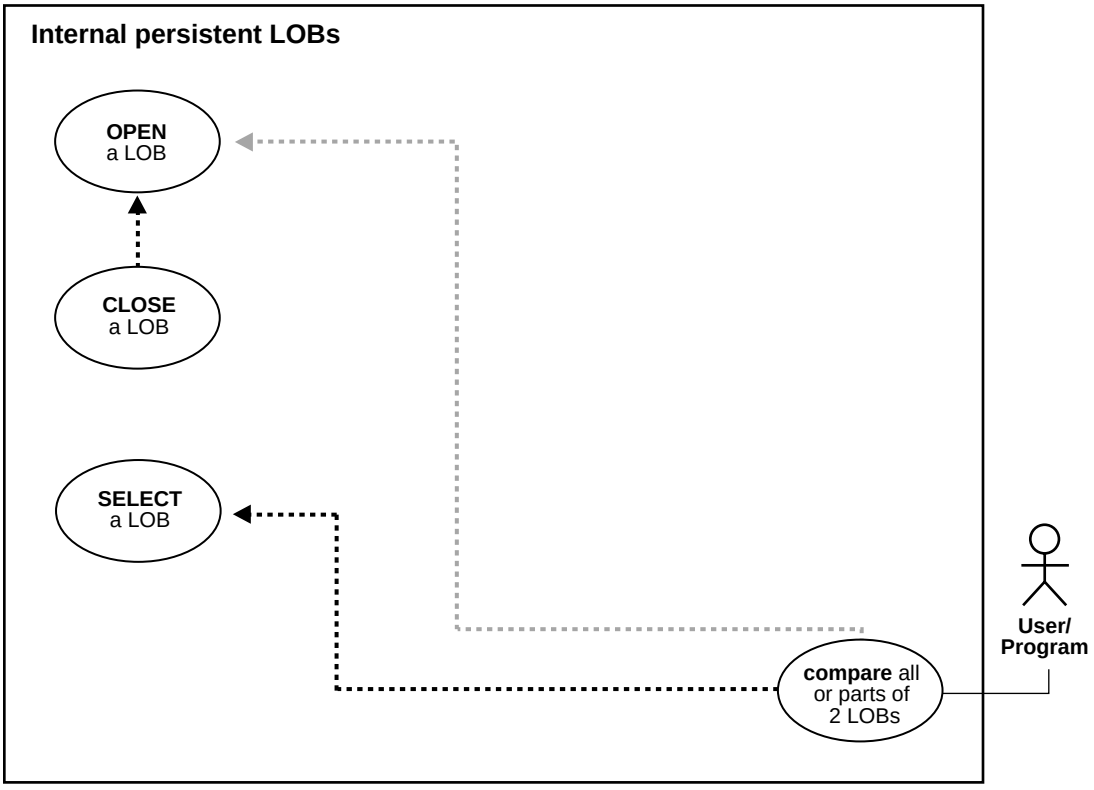
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_79
{

  static final int MAXBUFSIZE = 32767;
```

```
public static void main (String args [])
     throws Exception
{
  // Load the Oracle JDBC driver:
  Class.forName ("oracle.jdbc.driver.OracleDriver");

  // Connect to the database:
  Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

  // It's faster when auto commit is off:
  conn.setAutoCommit (false);

  // Create a Statement:
  Statement stmt = conn.createStatement ();

  try
  {
     BLOB lob_loc = null;
     byte buf[] = new byte[MAXBUFSIZE];

     ResultSet rset = stmt.executeQuery (
        "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
     if (rset.next())
     {
        lob_loc = ((OracleResultSet)rset).getBLOB (1);
     }

     OracleCallableStatement cstmt = (OracleCallableStatement)
        conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?,
                          DBMS_LOB.LOB_READONLY); END;");
     cstmt.setBLOB(1, lob_loc);
     cstmt.execute();

     // MAXBUFSIZE is the number of bytes to read and 1000 is the offset from
     // which to start reading:
     buf = lob_loc.getBytes(1000, MAXBUFSIZE);
    // Display the contents of the buffer here.

    cstmt = (OracleCallableStatement)
       conn.prepareCall ("BEGIN DBMS_LOB.CLOSE(?); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

    stmt.close();
```

```
      cstmt.close();
     conn.commit();
     conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Compare All or Part of Two LOBs

Figure 3–24 Use Case Diagram: Compare All or Part of Two LOBs



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The following example compares two frames from the archival table `VideoframesLib_tab` to see whether they are different and, depending on the

result of the comparison, inserts the `Frame` into the `Multimedia_tab`.

- "Example: Compare All or Part of Two LOBs Using PL/SQL (DBMS_LOB Package)" on page 3-124
- "Example: Compare All or Part of Two LOBs Using COBOL (Pro*COBOL)" on page 3-125
- "Example: Compare All or Part of Two LOBs Using C++ (Pro*C/C++)" on page 3-127
- "Example: Compare All or Part of Two LOBs Using Visual Basic (OO4O)" on page 3-128
- "Example: Compare All or Part of Two LOBs Using Java (JDBC)" on page 3-128

Example: Compare All or Part of Two LOBs Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure compareTwoLOBs_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE compareTwoLOBs_proc IS
    Lob_loc1            BLOB;
    Lob_loc2            BLOB;
    Amount              INTEGER := 32767;
    Retval              INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT Frame INTO Lob_loc1 FROM Multimedia_tab
        WHERE Clip_ID = 1;
    SELECT Frame INTO Lob_loc2 FROM Multimedia_tab
        WHERE Clip_ID = 2;
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc1, DBMS_LOB.LOB_READONLY);
    DBMS_LOB.OPEN (Lob_loc2, DBMS_LOB.LOB_READONLY);
    /* Compare the two frames: */
    retval := DBMS_LOB.COMPARE(Lob_loc1, Lob_loc2, Amount, 1, 1);
    IF retval = 0 THEN
       DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
    ELSE
       DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
    END IF;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc1);
    DBMS_LOB.CLOSE (Lob_loc2);
END;
```

Example: Compare All or Part of Two LOBs Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID   PIC X(11) VALUES "USER1/USER1".

01  BLOB1          SQL-BLOB.
01  BLOB2          SQL-BLOB.
01  BUFFER2        PIC X(32767) VARYING.
01  RET            PIC S9(9) COMP.
01  AMT            PIC S9(9) COMP.
01  POS            PIC S9(9) COMP VALUE 1024.
01  OFFSET         PIC S9(9) COMP VALUE 1.

    EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
COMPARE-BLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.
* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL ALLOCATE :BLOB2 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

    EXEC SQL
         SELECT FRAME INTO :BLOB1
         FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
    END-EXEC.

    EXEC SQL
         SELECT FRAME INTO :BLOB2
         FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2
    END-EXEC.

* Open the BLOBs for READ ONLY:
    EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
```

```
     EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.

* Execute PL/SQL to get COMPARE functionality:
     MOVE 4 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BLOB1,:BLOB2,:AMT,1,1);
       END;
     END-EXEC.

     IF RET = 0
*        Logic for equal BLOBs goes here
         DISPLAY "BLOBs are equal"
     ELSE
*        Logic for unequal BLOBs goes here
         DISPLAY "BLOBs are not equal"
     END-IF.
     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB2 END-EXEC.

 END-OF-BLOB.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BLOB1 END-EXEC.
     EXEC SQL FREE :BLOB2 END-EXEC.
     EXEC SQL
          COMMIT WORK RELEASE
     END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED:".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
         ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.
```

Example: Compare All or Part of Two LOBs Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void compareTwoLobs_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;
  int Amount = 32767;
  int Retval;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Select the LOBs: */
  EXEC SQL SELECT Frame INTO :Lob_loc1
     FROM Multimedia_tab WHERE Clip_ID = 1;
  EXEC SQL SELECT Frame INTO :Lob_loc2
     FROM Multimedia_tab WHERE Clip_ID = 2;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Compare the two Frames using DBMS_LOB.COMPARE() from within PL/SQL: */
  EXEC SQL EXECUTE
     BEGIN
        :Retval := DBMS_LOB.COMPARE(:Lob_loc1, :Lob_loc2, :Amount, 1, 1);
     END;
  END-EXEC;
  if (0 == Retval)
     printf("The frames are equal\n");
  else
     printf("The frames are not equal\n");
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
```

```
  /* Release resources held by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareTwoLobs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Compare All or Part of Two LOBs Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Dim OraDyn as OraDynaset, OraSound1 as OraBLOB, OraSoundClone as OraBLOB

Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
'Clone it for future reference
Set OraSoundClone = OraSound1

'Lets go to the next row and compare LOBs
OraDyn.MoveNext

MsgBox CBool(OraSound1.Compare(OraSoundClone, OraSoundClone.size, 1, 1))
```

Example: Compare All or Part of Two LOBs Using Java (JDBC)

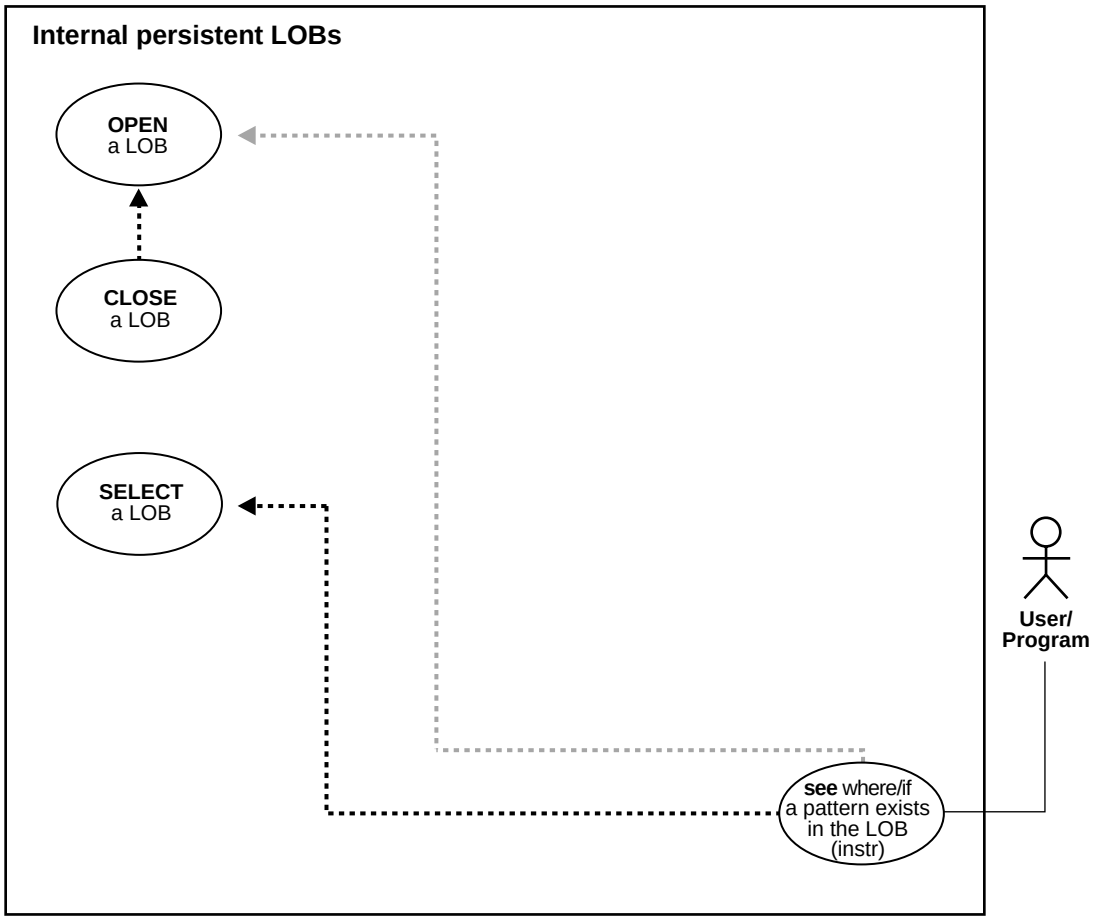
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
```

```
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_87
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BLOB lob_loc1 = null;
     BLOB lob_loc2 = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
   if (rset.next())
   {
     lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
   }

       rset = stmt.executeQuery (
          "SELECT frame FROM multimedia_tab WHERE clip_id = 99");
   if (rset.next())
   {
```

```
      lob_loc2 = ((OracleResultSet)rset).getBLOB (1);
    }

    if (lob_loc1.length() > lob_loc2.length())
      System.out.println("Looking for LOB2 inside LOB1.
         result = " + Long.toString(lob_loc1.position(lob_loc2, 0)));
    else
      System.out.println("Looking for LOB1 inside LOB2.
         result = " + Long.toString(lob_loc2.position(lob_loc1, 0)));

    stmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

See If a Pattern Exists in the LOB (instr)

Figure 3–25 Use Case Diagram: See If a Pattern Exists in the LOB (instr)



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The example examines the storyboard text to see if the string "children" is present.

- "Example: See If a Pattern Exists in the LOB (instr) Using PL/SQL (DBMS_LOB Package)" on page 3-132
- "Example: See If a Pattern Exists in the LOB (instr) Using COBOL (Pro*COBOL)" on page 3-133
- "Example: See If a Pattern Exists in the LOB (instr) Using C++ (Pro*C/C++)" on page 3-134
- "Example: See If a Pattern Exists in the LOB (instr) Using Java (JDBC)" on page 3-136

Example: See If a Pattern Exists in the LOB (instr) Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure instringLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instringLOB_proc IS
   Lob_loc        CLOB;
   Pattern        VARCHAR2(30) := 'children';
   Position       INTEGER := 0;
   Offset         INTEGER := 1;
   Occurrence     INTEGER := 1;
BEGIN
   /* Select the LOB: */
   SELECT Story INTO Lob_loc
      FROM Multimedia_tab
         WHERE Clip_ID = 1;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Seek for the pattern: */
   Position := DBMS_LOB.INSTR(Lob_loc, Pattern, Offset, Occurrence);
   IF Position = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Pattern not found');
   ELSE
      DBMS_OUTPUT.PUT_LINE('The pattern occurs at '|| position);
   END IF;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

Example: See If a Pattern Exists in the LOB (instr) Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CLOB-INSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  CLOB1          SQL-CLOB.
01  PATTERN        PIC X(8) VALUE "children".
01  POS            PIC S9(9) COMP.
01  OFFSET         PIC S9(9) COMP VALUE 1.
01  OCCURRENCE     PIC S9(9) COMP VALUE 1.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
CLOB-INSTR.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.

    EXEC SQL
         SELECT STORY INTO :CLOB1
         FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
    END-EXEC.

* Open the CLOB for READ ONLY:
    EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

* Execute PL/SQL to get INSTR functionality:
    EXEC SQL EXECUTE
      BEGIN
        :POS := DBMS_LOB.INSTR(:CLOB1, :PATTERN,
                               :OFFSET,:OCCURRENCE);
      END;
    END-EXEC.
```

```
            IF POS = 0
        *       Logic for pattern not found here
                DISPLAY "Pattern not found."
            ELSE
        *       Pos contains position where pattern is found
                DISPLAY "Pattern found."
            END-IF.


            EXEC SQL LOB CLOSE :CLOB1 END-EXEC.

        END-OF-CLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :CLOB1 END-EXEC.
            EXEC SQL
                 COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: See If a Pattern Exists in the LOB (instr) Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
```

```
}

void instringLOB_proc()
{
  OCIClobLocator *Lob_loc;
  char *Pattern = "The End";
  int Position = 0;
  int Offset = 1;
  int Occurrence = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Story INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc;
  /* Seek the Pattern using DBMS_LOB.INSTR() in a PL/SQL block: */
  EXEC SQL EXECUTE
    BEGIN
      :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, :Offset, :Occurrence);
    END;
  END-EXEC;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  /* Closing the LOB is mandatory if you have opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  instringLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: See If a Pattern Exists in the LOB (instr) Using Visual Basic (OO4O)

Note: A Visual Basic (OO4O) example will be made available in a subsequent release.

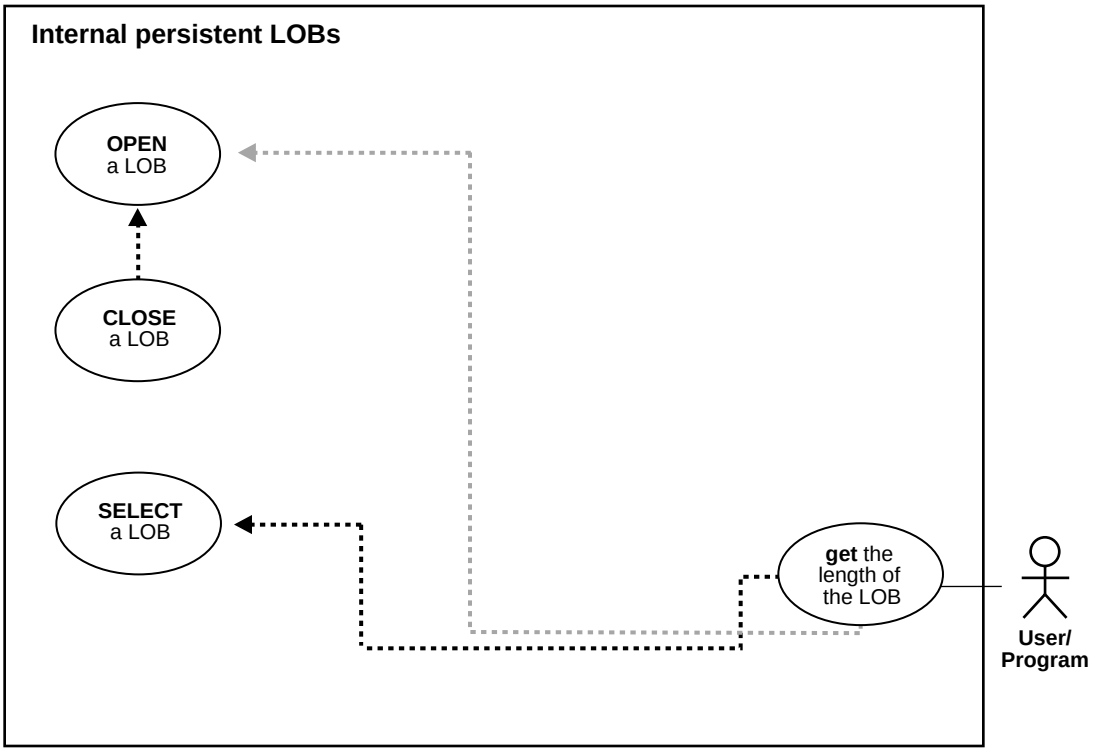
Example: See If a Pattern Exists in the LOB (instr) Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_91
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
```

```
    Statement stmt = conn.createStatement ();

    try
    {
     final int offset = 1;       // Start looking at the first byte
     final int occurrence = 1;  // Start at the 1st occurrence of the pattern
within the CLOB

   CLOB lob_loc = null;
   String pattern = new String("Junk"); // Pattern to look for within the CLOB.

       ResultSet rset = stmt.executeQuery (
          "SELECT story FROM multimedia_tab WHERE clip_id = 2");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   // Search for location of pattern string in the CLOB, starting at offset 1:
   long result = lob_loc.position(pattern, offset);
   System.out.println("Results of Pattern Comparison : " +
      Long.toString(result));

   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Get the Length of a LOB

Figure 3–26 Use Case Diagram: Get the length of a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This example demonstrates how to determine the length of a LOB in terms of the foreign language subtitle (FLSub).

- "Example: Get the Length of a LOB Using PL/SQL (DBMS_LOB Package)" on page 3-139

Example: Get the Length of a LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure getLengthLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE getLengthLOB_proc IS
   Lob_loc     NCLOB;
   Length      INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT FLSub INTO Lob_loc FROM Multimedia_tab
       WHERE Clip_ID = 2;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Get the length of the LOB: */
   length := DBMS_LOB.GETLENGTH(Lob_loc);
   IF length IS NULL THEN
       DBMS_OUTPUT.PUT_LINE('LOB is null.');
   ELSE
       DBMS_OUTPUT.PUT_LINE('The length is '|| length);
   END IF;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;
```

Example: Get the Length of a LOB Using C (OCI)

```
/* Select the locator into a locator variable */
sb4 select_FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;
```

```
  text  *sqlstmt =
    (text *)"SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_CLOB,(dvoid *)0, (ub2 *)0,
                                  (ub2 *)0, (ub4)OCI_DEFAULT));

  /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return 0;
}

/* This function gets the length of the selected LOB */
void getLengthLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  ub4 length;

  OCILobLocator *Lob_loc;

  /* Allocate Locator resources */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select a LOB locator from FLSub */
  printf(" select a FLSub locator\n");
  select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);

  /* Opening the LOB is Optional */
  printf(" Open the locator (optional)\n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY)));

  printf(" get the length of FLSub.\n");
```

```
  checkerr (errhp, OCILobGetLength(svchp, errhp, Lob_loc, &length));

  /* Length is undefined if the LOB is NULL or undefined */
  fprintf(stderr," Length of LOB is %d\n",length);

  /* Closing the LOBs is Mandatory if they have been Opened */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Get the Length of a LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. LOB-LENGTH.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  CLOB1          SQL-CLOB.
     01  LOB-ATTR-GRP.
         05 LEN         PIC S9(9) COMP.

     01  D-LEN          PIC 9(4).
     01  USERID   PIC X(11) VALUES "USER1/USER1".
         EXEC SQL INCLUDE SQLCA END-EXEC.

     PROCEDURE DIVISION.
     LOB-LENGTH.

         EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
         EXEC SQL
              CONNECT :USERID
         END-EXEC.

        * Allocate and initialize the target CLOB:
         EXEC SQL ALLOCATE :CLOB1 END-EXEC.

         EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
         EXEC SQL
            SELECT STORY INTO :CLOB1
```

```
           FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2
       END-EXEC.

   * Obtain the length of the CLOB:
       EXEC SQL
          LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN
       END-EXEC.

       MOVE LEN TO D-LEN.
       DISPLAY "The length is ", D-LEN.

   * Free the resources used by the CLOB:
    END-OF-CLOB.
       EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
       EXEC SQL FREE :CLOB1 END-EXEC.
       EXEC SQL
            COMMIT WORK RELEASE
       END-EXEC.
       STOP RUN.

    SQL-ERROR.
       EXEC SQL
           WHENEVER SQLERROR CONTINUE
       END-EXEC.
       DISPLAY " ".
       DISPLAY "ORACLE ERROR DETECTED:".
       DISPLAY " ".
       DISPLAY SQLERRMC.
       EXEC SQL
           ROLLBACK WORK RELEASE
       END-EXEC.
       STOP RUN.
```

Example: Get the Length of a LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void getLengthLOB_proc()
{
  OCIClobLocator *Lob_loc;
  unsigned int Length;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Story INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Length: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* If the LOB is NULL or unitialized, then Length is Undefined: */
  printf("Length is %d characters\n", Length);
  /* Closing the LOB is mandatory if you have Opened it: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Get the Length of a LOB Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
```

```
'Display out size of the lob:
MsgBox "Length of the lob is " & OraSound1.Size
```

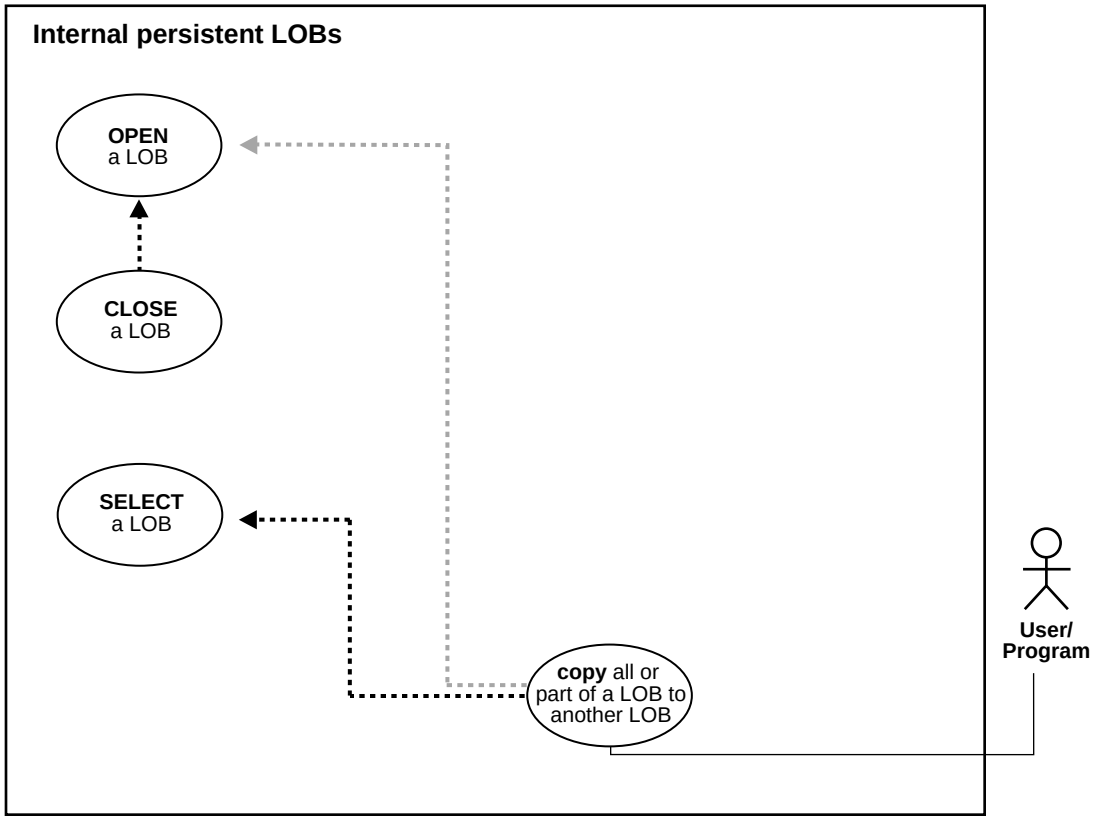
Example: Get the Length of a LOB Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_95
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
```

```
    {
     CLOB lob_loc = null;

      ResultSet rset = stmt.executeQuery ("SELECT story FROM multimedia_tab
WHERE clip_id = 2");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   System.out.println(
      "Length of this column is : " + Long.toString(lob_loc.length()));

   stmt.close();
   conn.commit();
   conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Copy All or Part of a LOB to another LOB

Figure 3–27 Use Case Diagram: Copy all or part of a LOB to another LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Locking the Row Prior to Updating

Prior to updating a `LOB` value via the PL/SQL `DBMS_LOB` package or the OCI, you must lock the row containing the `LOB`. While the SQL `INSERT` and `UPDATE`

statements implicitly lock the row, locking is done explicitly by means of a SQL SELECT FOR UPDATE statement in SQL and PL/SQL programs, or by using an OCI pin or lock function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

The code in this example shows you to copy a portion of Sound from one clip to another.

- "Example: Copy All or Part of a LOB to another LOB Using PL/SQL (DBMS_LOB Package)" on page 3-147
- "Example: Copy All or Part of a LOB to another LOB Using C (OCI)" on page 3-148
- "Example: Copy All or Part of a LOB to another LOB Using COBOL (Pro*COBOL)" on page 3-150
- "Example: Copy All or Part of a LOB to another LOB Using C++ (Pro*C/C++)" on page 3-152
- "Example: Copy All or Part of a LOB to another LOB Using Visual Basic (OO4O)" on page 3-154
- "Example: Copy All or Part of a LOB to another LOB Using Java (JDBC)" on page 3-154

Example: Copy All or Part of a LOB to another LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure copyLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE copyLOB_proc IS
   Dest_loc     BLOB;
   Src_loc      BLOB;
   Amount       NUMBER;
   Dest_pos     NUMBER;
   Src_pos      NUMBER;
BEGIN
   SELECT Sound INTO Dest_loc FROM Multimedia_tab
      WHERE Clip_ID = 2 FOR UPDATE;
   /* Select the LOB: */
   SELECT Sound INTO Src_loc FROM Multimedia_tab
```

```
      WHERE Clip_ID = 1;
   /* Opening the LOBs is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Copies the LOB from the source position to the destination position: */
   DBMS_LOB.COPY(Dest_loc, Src_loc, Amount, Dest_pos, Src_pos);
   /* Closing LOBs is mandatory if you have opened them: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Copy All or Part of a LOB to another LOB Using C (OCI)

```
/* Select the locator */
sb4 select_lock_sound_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1           dest_type;                    /* whether destination locator */
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  char        sqlstmt[150];
  OCIDefine   *defnp1;

  if (dest_type == TRUE)
  {
    strcpy (sqlstmt,
           (char *)"SELECT Sound FROM Multimedia_tab
              WHERE Clip_ID=2 FOR UPDATE");
    printf ("  select destination sound locator\n");
  }
  else
  {
    strcpy(sqlstmt, (char *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1");
    printf ("  select source sound locator\n");
  }
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, (text *)sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
```

```
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                   (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return 0;
}

/* This function copies part of the Source LOB into a specified position
   in the destination LOB
 */
void copyAllPartLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  int Amount = 1000;                                  /* <Amount to Copy> */
  int Dest_pos = 100;                    /*<Position to start copying into> */
  int Src_pos = 1;                       /* <Position to start copying from> */

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf(" select the destination and source locators\n");
  select_lock_sound_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
                                                    /* destination locator */
  select_lock_sound_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
                                                         /* source locator */

  /* Opening the LOBs is Optional */
  printf (" open the destination locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
  printf (" open the source locator (optional)\n");
  checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));
```

```
  printf (" copy the lob (amount) from the source to destination\n");
  checkerr (errhp, OCILobCopy(svchp, errhp, Dest_loc, Src_loc,
                              Amount, Dest_pos, Src_pos));

  /* Closing the LOBs is Mandatory if they have been Opened */
  printf(" close the locators\n");
  checkerr (errhp, OCILobClose(svchp, errhp, Dest_loc));
  checkerr (errhp, OCILobClose(svchp, errhp, Src_loc));

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Dest_loc, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) Src_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Copy All or Part of a LOB to another LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. BLOB-COPY.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID   PIC X(11) VALUES "USER1/USER1".

     01  DEST           SQL-BLOB.
     01  SRC            SQL-BLOB.

    * Define the amount to copy.
    * This value has been chosen arbitrarily:
     01  AMT            PIC S9(9) COMP VALUE 10.

    * Define the source and destination position.
    * These values have been chosen arbitrarily:
     01  SRC-POS        PIC S9(9) COMP VALUE 1.
     01  DEST-POS       PIC S9(9) COMP VALUE 1.

    * The return value from PL/SQL function:
     01  RET            PIC S9(9) COMP.

         EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
PROCEDURE DIVISION.
COPY-BLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :DEST END-EXEC.
    EXEC SQL ALLOCATE :SRC END-EXEC.
    DISPLAY "Source and destination LOBs are open.".

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :SRC
         FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2
    END-EXEC.
    DISPLAY "Source LOB populated.".
    EXEC SQL
         SELECT SOUND INTO :DEST
         FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 3 FOR UPDATE
    END-EXEC.
    DISPLAY "Destination LOB populated.".

* Open the DESTination LOB read/write and SRC LOB read only
    EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
    EXEC SQL LOB OPEN :SRC READ ONLY END-EXEC.
    DISPLAY "Source and destination LOBs are open.".

* Copy the desired amount
    EXEC SQL
         LOB COPY :AMT FROM :SRC AT :SRC-POS
         TO :DEST AT :DEST-POS
    END-EXEC.
    DISPLAY "Src LOB copied to destination LOB.".

* Execute PL/SQL to get COMPARE functionality
* to make sure that the BLOBs are identical
    EXEC SQL EXECUTE
      BEGIN
        :RET := DBMS_LOB.COMPARE(:SRC,:DEST,:AMT,1,1);
      END;
    END-EXEC.
```

```
            IF RET = 0
          *     Logic for equal BLOBs goes here
                DISPLAY "BLOBs are equal"
            ELSE
          *     Logic for unequal BLOBs goes here
                DISPLAY "BLOBs are not equal"
            END-IF.

            EXEC SQL LOB CLOSE :DEST END-EXEC.
            EXEC SQL LOB CLOSE :SRC END-EXEC.

         END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :DEST END-EXEC.
            EXEC SQL FREE :SRC END-EXEC.
            EXEC SQL
                 COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

         SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Copy All or Part of a LOB to another LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void copyLOB_proc()
{
  OCIBlobLocator *Dest_loc, *Src_loc;
  int Amount = 5;
  int Dest_pos = 10;
  int Src_pos = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL ALLOCATE :Src_loc;
  /* Select the LOBs: */
  EXEC SQL SELECT Sound INTO :Dest_loc
           FROM Multimedia_tab WHERE Clip_ID = 2 FOR UPDATE;
  EXEC SQL SELECT Sound INTO :Src_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
  /* Copies the specified Amount from the source position in the source
     LOB to the destination position in the destination LOB: */
  EXEC SQL LOB COPY :Amount
               FROM :Src_loc AT :Src_pos TO :Dest_loc AT :Dest_pos;
  /* Closing the LOBs is mandatory if they have been opened: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;
  /* Release resources held by the locators: */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  copyLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Copy All or Part of a LOB to another LOB Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

Set OraSoundClone = OraSound1

'Go to next row and copy LOB

OraDyn.MoveNext

OraDyn.Edit
OraSound1.Copy OraSoundClone, OraSoundClone.Size, 1, 1
OraDyn.Update
```

Example: Copy All or Part of a LOB to another LOB Using Java (JDBC)

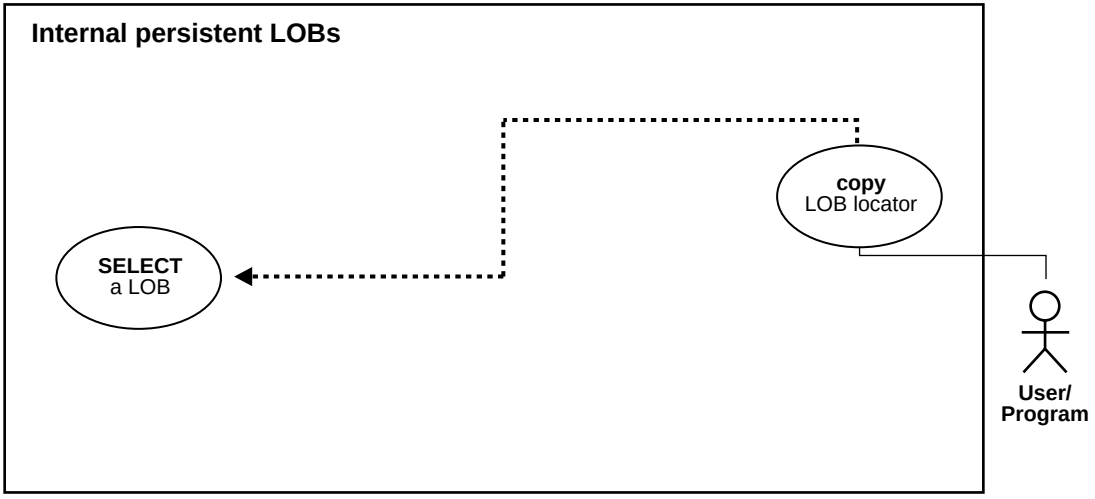
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_100
{
```

```
public static void main (String args [])
     throws Exception
{
  // Load the Oracle JDBC driver:
  Class.forName ("oracle.jdbc.driver.OracleDriver");

  // Connect to the database:
  Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

  // It's faster when auto commit is off:
  conn.setAutoCommit (false);

  // Create a Statement:
  Statement stmt = conn.createStatement ();

  try
  {
   final int AMOUNT_TO_COPY = 2000;

   ResultSet rset = null;
   BLOB dest_loc = null;
   BLOB src_loc = null;
   InputStream in = null;
   OutputStream out = null;
   byte[] buf = new byte[AMOUNT_TO_COPY];

   rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 1");
 if (rset.next())
 {
   src_loc = ((OracleResultSet)rset).getBLOB (1);
 }
 in = src_loc.getBinaryStream();

     rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
 if (rset.next())
 {
   dest_loc = ((OracleResultSet)rset).getBLOB (1);
 }
 out = dest_loc.getBinaryOutputStream();
```

```
   // read AMOUNT_TO_COPY bytes into buf from stream, starting from offset 0:
   in.read(buf, 0, AMOUNT_TO_COPY);

   // write AMOUNT_TO_COPY bytes from buf into output stream, starting at offset
0:
   out.write(buf, 0, AMOUNT_TO_COPY);

   // Close all streams and handles
   in.close();
   out.flush();
   out.close();
   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Copy a LOB Locator

Figure 3–28 Use Case Diagram: Copy a LOB Locator



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This example shows the copying of one locator to another involving the video frame (Frame). Note how different locators may point to the same / different, current / outdated data.

- "Example: Copy a LOB Locator Using PL / SQL" on page 3-158
- "Example: Copy a LOB Locator Using C (OCI)" on page 3-158
- "Example: Copy a LOB Locator Using COBOL (Pro*COBOL)" on page 3-160
- "Example: Copy a LOB Locator Using C++ (Pro*C / C++)" on page 3-161
- "Example: Copy a LOB Locator Using Visual Basic (OO4O)" on page 3-162
- "Example: Copy a LOB Locator Using Java (JDBC)" on page 3-163

Example: Copy a LOB Locator Using PL/SQL

> **Note:** Assigning one LOB to another using PL/SQL entails using the ":=" sign. This is an advanced topic that is discussed in more detail under the heading "Read-Consistent Locators" on page 2-2.

```
/* Note that the example procedure lobAssign_proc is not part of the
   DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE lobAssign_proc IS
  Lob_loc1    blob;
  Lob_loc2    blob;
BEGIN
  SELECT Frame INTO Lob_loc1 FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
   /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the lob
      at this point in time. */
  Lob_loc2 := Lob_loc1;
  /* When you write some data to the lob through Lob_loc1, Lob_loc2 will not see
      the newly written data whereas Lob_loc1 will see the new data. */
END;
```

Example: Copy a LOB Locator Using C (OCI)

```
/* Select the locator */
sb4 select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));
```

```
  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return (0);
}

void assignLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *dest_loc, *src_loc;
  boolean       isEqual;

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf (" select and lock a frame locator\n");
  select_lock_frame_locator(src_loc, errhp, svchp, stmthp);/* source locator */

  /* Assign src_loc to dest_loc thereby saving a copy of the value of the LOB
     at this point in time.
   */
  printf(" assign the src locator to dest locator\n");
  checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));

  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data.
   */

  /* Call OCI to see if the two locators are Equal */

  printf (" check if Lobs are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, src_loc, dest_loc, &isEqual));

  if (isEqual)
  {
    /* ... The LOB locators are Equal */
```

```
    printf(" Lob Locators are equal.\n");
  }
  else
  {
    /* ... The LOB locators are not Equal */
    printf(" Lob Locators are NOT Equal.\n");
  }

  /* Note that in this example, the LOB locators will be Equal */

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) dest_loc, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) src_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Copy a LOB Locator Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. COPY-LOCATOR.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID   PIC X(11) VALUES "USER1/USER1".
     01  DEST          SQL-BLOB.
     01  SRC           SQL-BLOB.

         EXEC SQL INCLUDE SQLCA END-EXEC.
     PROCEDURE DIVISION.
     COPY-BLOB-LOCATOR.

         EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
         EXEC SQL
              CONNECT :USERID
         END-EXEC.
    * Allocate and initialize the BLOB locators:
         EXEC SQL ALLOCATE :DEST END-EXEC.
         EXEC SQL ALLOCATE :SRC END-EXEC.

         EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

         EXEC SQL
```

```
            SELECT FRAME INTO :SRC
            FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2 FOR UPDATE
        END-EXEC.

        EXEC SQL
            LOB ASSIGN :SRC TO :DEST
        END-EXEC.

      * When you write data to the LOB through SRC, DEST will
      * not see the newly written data

       END-OF-BLOB.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :DEST END-EXEC.
        EXEC SQL FREE :SRC END-EXEC.
        EXEC SQL
            COMMIT WORK RELEASE
        END-EXEC.
        STOP RUN.

       SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED:".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Copy a LOB Locator Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void lobAssign_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Frame INTO :Lob_loc1
           FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the
     LOB at this point in time: */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data: */
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  lobAssign_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Copy a LOB Locator Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id ", ORADYN_DEFAULT)

Set OraSound1 = OraDyn.Fields("Sound").Value
Set OraSoundClone = OraSound1

OraDyn.MoveNext
```

```
'Copy 1000 bytes of LOB values OraSoundClone to OraSoundl at OraSoundl
'offset 100:
OraDyn.Edit
OraSound1.Copy OraSoundClone, 1000, 100

OraDyn.Update
```

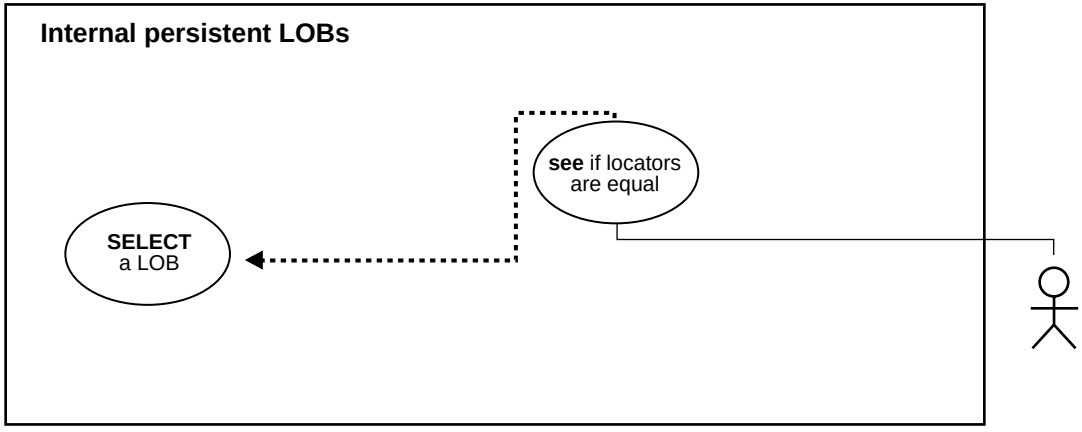
Example: Copy a LOB Locator Using Java (JDBC)

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_104
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BLOB lob_loc1 = null;
```

```
      BLOB lob_loc2 = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT frame FROM multimedia_tab WHERE clip_id = 1");
    if (rset.next())
    {
      lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

    // When you write some data to the LOB through lob_loc1, lob_loc2 will not
see the changes
    lob_loc2 = lob_loc1;

    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

See If One LOB Locator Is Equal to Another

Figure 3–29 Use Case Diagram: See If One LOB Locator Is Equal to Another



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

If two locators are equal, this means that they refer to the same version of the LOB data (see "Read-Consistent Locators" on page 2-2). In this example, the locators are equal. However, it may be as important to determine that locators do not refer to same version of the LOB data.

This functionality is available in only a limited number of environments.

- "Example: See If One LOB Locator Is Equal to Another Using C (OCI)" on page 3-166
- "Example: See If One LOB Locator Is Equal to Another Using C++ (Pro*C/C++)" on page 3-167
- "Example: See If One LOB Locator Is Equal to Another Using Java (JDBC)" on page 3-169

Example: See If One LOB Locator Is Equal to Another Using C (OCI)

```
/* Select the locator: */

sb4 select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return (0);
}

void assignLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *dest_loc, *src_loc;
  boolean       isEqual;

  /* Allocate the LOB locators: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
```

```
  /* Select the LOBs: */
  printf (" select and lock a frame locator\n");
  select_lock_frame_locator(src_loc, errhp, svchp, stmthp);/* source locator */

  /* Assign src_loc to dest_loc thereby saving a copy of the value of the LOB
     at this point in time: */
  printf(" assign the src locator to dest locator\n");
  checkerr (errhp, OCILobAssign(envhp, errhp, src_loc, &dest_loc));

  /* When you write some data to the LOB through Lob_loc1, Lob_loc2 will not
     see the newly written data whereas Lob_loc1 will see the new data: */

  /* Call OCI to see if the two locators are Equal: */

  printf (" check if Lobs are Equal.\n");
  checkerr (errhp, OCILobIsEqual(envhp, src_loc, dest_loc, &isEqual));

  if (isEqual)
  {
    /* ... The LOB locators are Equal: */
    printf(" Lob Locators are equal.\n");
  }
  else
  {
    /* ... The LOB locators are not Equal: */
    printf(" Lob Locators are NOT Equal.\n");
  }

  /* Note that in this example, the LOB locators will be Equal */

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) dest_loc, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) src_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: See If One LOB Locator Is Equal to Another Using C++ (Pro*C/C++)

```
/* Pro*C/C++ does not provide a mechanism to test the equality of two
   locators.  However, by using the OCI directly, two locators can be
   compared to determine whether or not they are equal as this example
   demonstrates: */
```

```
#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void LobLocatorIsEqual_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2;
  OCIEnv *oeh;
  boolean isEqual;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Frame INTO Lob_loc1
           FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
  /* Assign Lob_loc1 to Lob_loc2 thereby saving a copy of the value of the
     LOB at this point in time: */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* When you write some data to the lob through Lob_loc1, Lob_loc2 will
     not see the newly written data whereas Lob_loc1 will see the new
     data. */
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Call OCI to see if the two locators are Equal: */
  (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
  if (isEqual)
    printf("The locators are equal\n");
  else
    printf("The locators are not equal\n");
  /* Note that in this example, the LOB locators will be Equal */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
```

```
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobLocatorIsEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

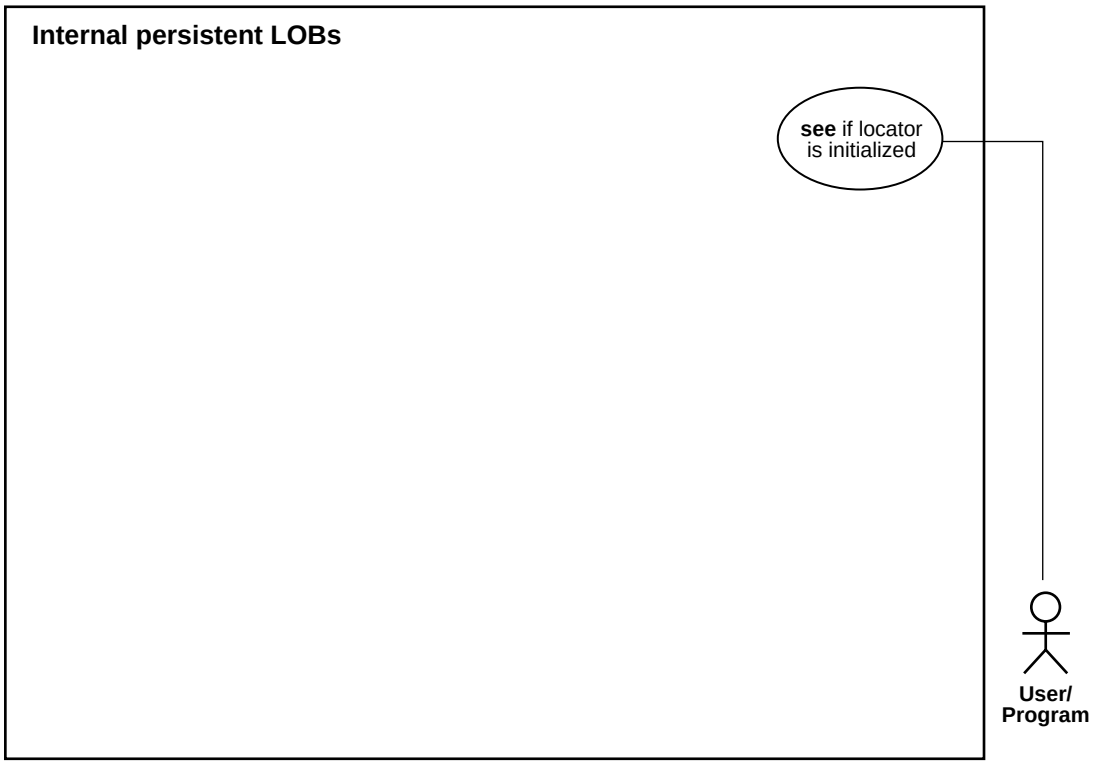
Example: See If One LOB Locator Is Equal to Another Using Java (JDBC)

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_108
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
      BLOB lob_loc1 = null;
      BLOB lob_loc2 = null;
```

```
        ResultSet rset = stmt.executeQuery (
           "SELECT sound FROM multimedia_tab WHERE clip_id = 2");
    if (rset.next())
    {
      lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

    // When you write some data to the LOB through lob_loc1, lob_loc2 will not
see the changes:
    lob_loc2 = lob_loc1;

    // Note that in this example, the Locators will be equal.
    if (lob_loc1.equals(lob_loc2))
    {
       // The Locators are equal:
       System.out.println("Locators are equal");
    }
    else
    {
       // The Locators are different:
       System.out.println("Locators are NOT equal");
    }

    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

See If a LOB Locator Is Initialized

Figure 3–30 Use Case Diagram: See If a LOB Locator Is Initialized



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The operation allows you to determine if the locator has been initialized or not. In the example shown both locators are found to be initialized.

This functionality is currently available in only two environments.

- "Example: See If a LOB Locator Is Initialized Using C (OCI)" on page 3-172
- "Example: See If a LOB Locator Is Initialized Using C++ (Pro*C/C++)" on page 3-173

Example: See If a LOB Locator Is Initialized Using C (OCI)

```
/* Select the locator: */

sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text      *sqlstmt =
                (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return (0);
}

void isInitializedLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc1, *Lob_loc2;
  boolean       isInitialized;
```

```
  /* Allocate the LOB locators: */
  printf(" allocate locator 1 and 2\n");
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc1,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc2,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs: */
  printf (" select a frame locator into locator 1\n");
  select_frame_locator(Lob_loc1, errhp, svchp, stmthp);         /* locator 1 */

  /* Determine if the locator 1 is Initialized -: */
  checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc1, &isInitialized));
                                    /* IsInitialized should return TRUE here */
  printf(" for Locator 1, isInitialized = %d\n", isInitialized);

  /* Determine if the locator 2 is Initialized -: */
  checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, Lob_loc2, &isInitialized));
                                    /* IsInitialized should return TRUE here */
  printf(" for Locator 2, isInitialized = %d\n", isInitialized);

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc1, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) Lob_loc2, (ub4) OCI_DTYPE_LOB);

  return;
}
```

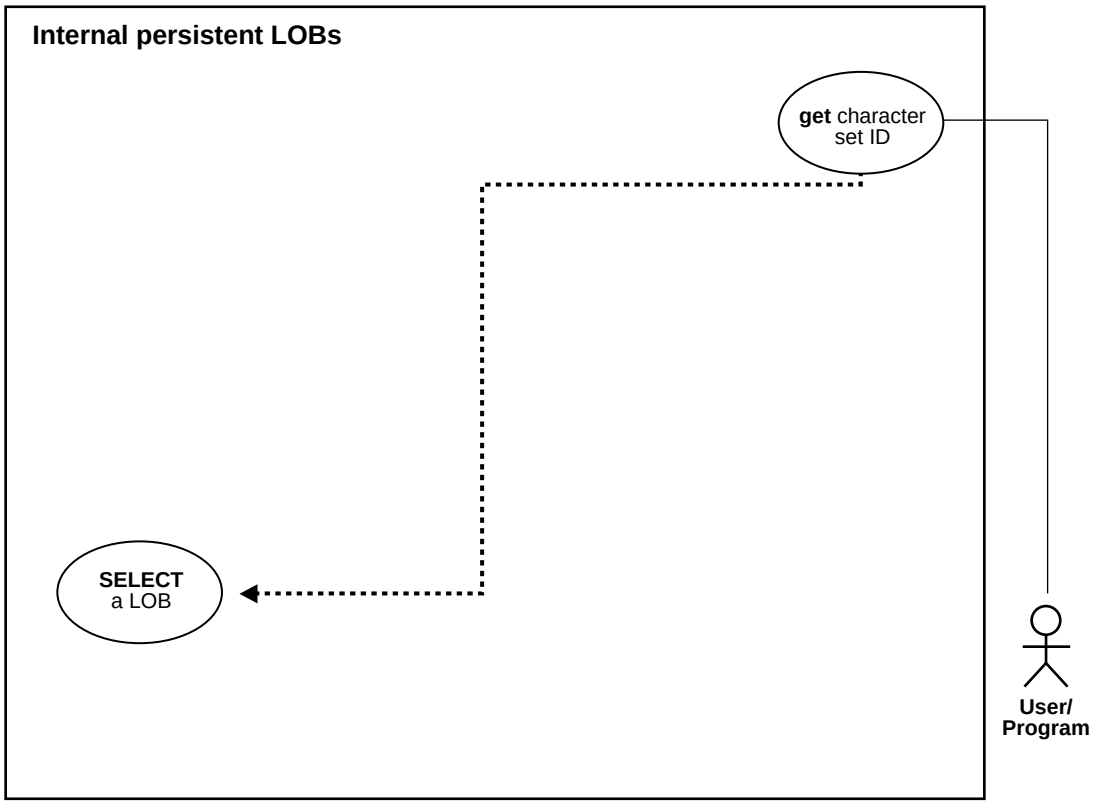
Example: See If a LOB Locator Is Initialized Using C++ (Pro*C/C++)

```
/* Pro*C/C++ has no form of embedded SQL statement to determine if a LOB
   locator is initialized.  Locators in Pro*C/C++ are initialized when they
   are allocated via the EXEC SQL ALLOCATE statement. However, an example
   can be written that uses embedded SQL and the OCI as is shown here: */

#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void LobLocatorIsInit_proc()
{
  OCIBlobLocator *Lob_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Frame INTO Lob_loc
           FROM Multimedia_tab where Clip_ID = 1;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized: */
  (void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
  if (isInitialized)
    printf("The locator is initialized\n");
  else
    printf("The locator is not initialized\n");
  /* Note that in this example, the locator is initialized */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobLocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Get Character Set ID

Figure 3–31 Use Case Diagram: Get Character Set ID



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The use case demonstrates how to determine the characterset ID of the foreign language subtitle (FLSub). This functionality is available only in OCI.

- "Example: Get Character Set ID Using C (OCI)" on page 3-176

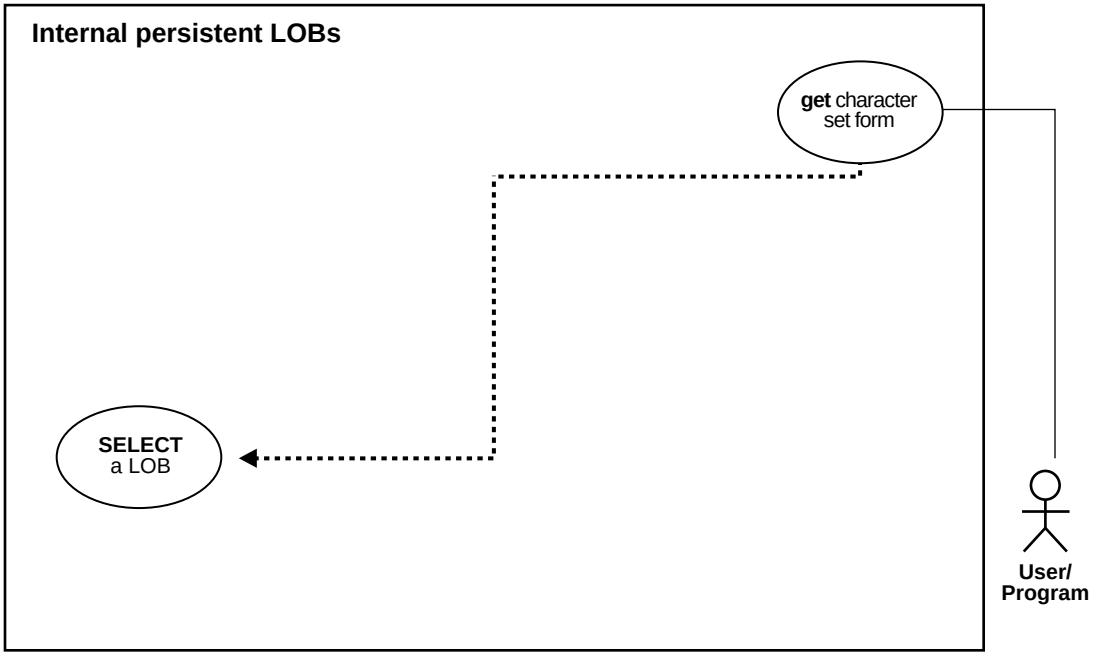
Example: Get Character Set ID Using C (OCI)

```
/* This function takes a valid LOB locator and prints the character set id of
the LOB. */

/* Select the locator */
sb4 select_FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
    (text *)"SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_CLOB,(dvoid *)0, (ub2 *)0,
                                  (ub2 *)0, (ub4)OCI_DEFAULT));

  /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return 0;
}

sb4 getcsidLob (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub2 charsetid =0 ;
```

```
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  printf (" select a FLSub locator\n");
  select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);

  printf (" get the character set id of FLSub_locator\n");

  /* Get the charactersid ID of the LOB*/
  checkerr (errhp, OCILobCharSetId(envhp, errhp, Lob_loc, &charsetid));
  printf(" character Set ID of FLSub is : %d\n", charsetid);

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Get Character Set Form

Figure 3–32 Use Case Diagram: Get Character Set Form



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

The use case demonstrates how to determine the character set form of the foreign language subtitle (FLSub). This functionality is available only in OCI.

- "Example: Get Character Set Form Using C (OCI)" on page 3-179

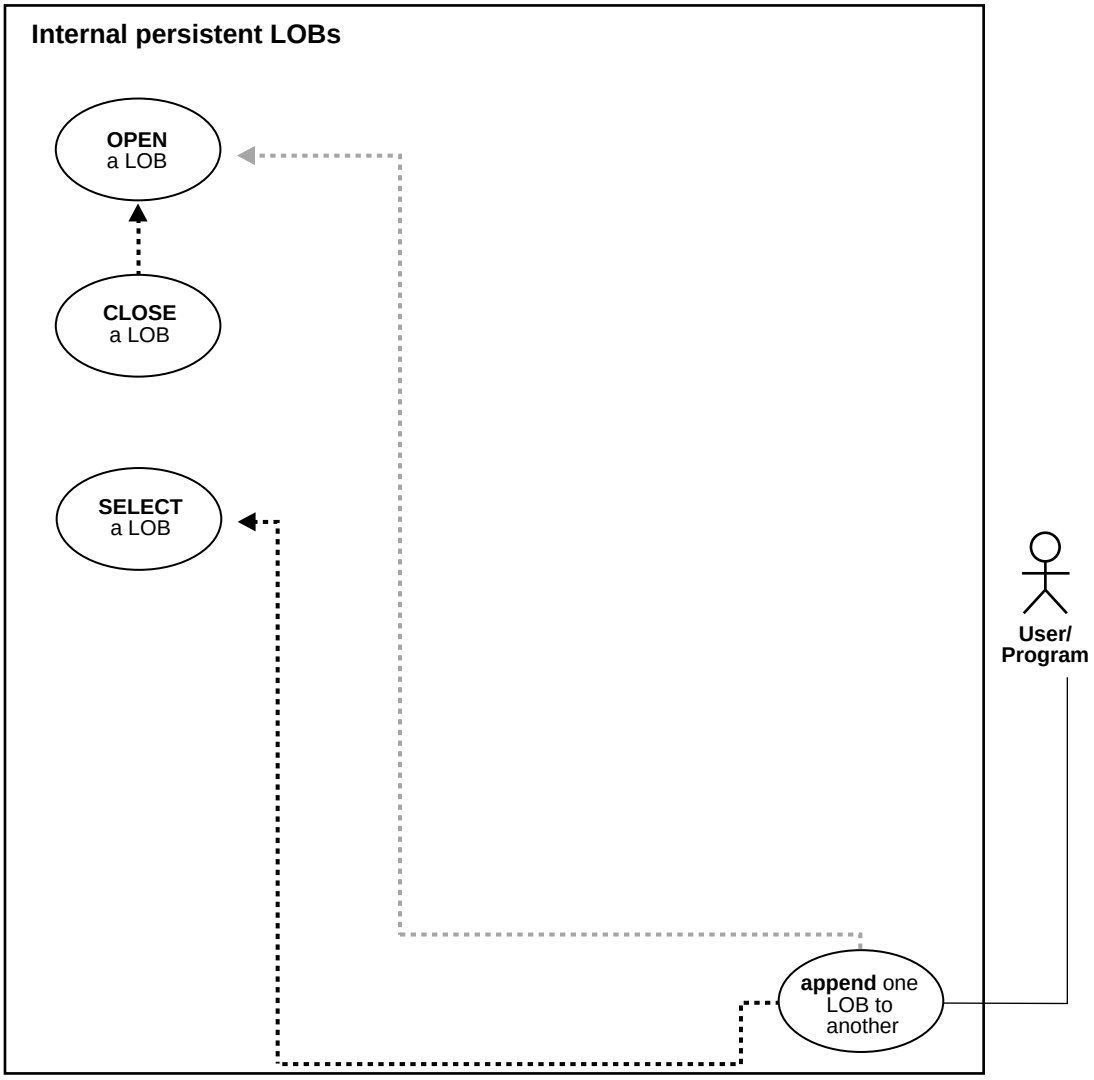
Example: Get Character Set Form Using C (OCI)

```
/* Select the locator */
sb4 select_FLSub_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text  *sqlstmt =
    (text *)"SELECT FLSub FROM Multimedia_tab WHERE Clip_ID = 2";

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Define the column being selected */
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2)SQLT_CLOB,(dvoid *)0, (ub2 *)0,
                                  (ub2 *)0, (ub4)OCI_DEFAULT));

  /* Execute and fetch one row */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return 0;
}

/* This function takes a valid LOB locator and prints the character set form
   of the LOB.
 */

sb4 getcsformLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub1 charset_form = 0 ;
```

```
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  printf (" select a FLSub locator\n");
  select_FLSub_locator(Lob_loc, errhp, svchp, stmthp);

  printf (" get the character set form of FLSub\n");

  /* Get the charactersid ID of the LOB*/
  checkerr (errhp, OCILobCharSetForm(envhp, errhp, Lob_loc, &charset_form));
  printf(" character Set Form of FLSub is : %d\n", charset_form);

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;

}
```

Append One LOB to Another

Figure 3–33 Use Case Diagram: Append one LOB to another



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Locking the Row Prior to Updating

Prior to updating a LOB value via the PL/SQL DBMS_LOB package or the OCI, you must lock the row containing the LOB. While the SQL INSERT and UPDATE statements implicitly lock the row, locking is done explicitly by means of a SQL SELECT FOR UPDATE statement in SQL and PL/SQL programs, or by using an OCI pin or lock function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

This example deals with the task of appending one segment of Sound to another. We assume that you use sound-specific editing tools to match the wave-forms.

- "Example: Append One LOB to Another Using PL/SQL (DBMS_LOB Package)" on page 3-182
- "Example: Append One LOB to Another Using C (OCI)" on page 3-183
- "Example: Append One LOB to Another Using COBOL (Pro*COBOL)" on page 3-185
- "Example: Append One LOB to Another Using C++ (Pro*C/C++)" on page 3-186
- "Example: Append One LOB to Another Using Visual Basic (OO4O)" on page 3-187
- "Example: Append One LOB to Another Using Java (JDBC)" on page 3-188

Example: Append One LOB to Another Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure appendLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE appendLOB_proc IS
   Dest_loc        BLOB;
   Src_loc         BLOB;
BEGIN
```

```
   /* Select the LOB, get the destination LOB locator: */
   SELECT Sound INTO Dest_loc FROM Multimedia_tab
      WHERE Clip_ID = 2
         FOR UPDATE;
   /* Select the LOB, get the destination LOB locator: */
   SELECT Sound INTO Src_loc FROM Multimedia_tab
      WHERE Clip_ID = 1;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);
   DBMS_LOB.APPEND(Dest_loc, Src_loc);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Dest_loc);
   DBMS_LOB.CLOSE (Src_loc);
COMMIT;

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Append One LOB to Another Using C (OCI)

```
/* This function appends the Source LOB to the end of the Destination LOB*/
/* Select the locator */
sb4 select_lock_sound_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1            dest_type;                    /* whether destination locator */
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  char        sqlstmt[150];
  OCIDefine   *defnp1;

  if (dest_type == TRUE)
  {
    strcpy (sqlstmt,
      (char *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=2 FOR UPDATE");
    printf ("  select destination sound locator\n");
  }
  else
  {
    strcpy(sqlstmt, (char *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1");
```

```
    printf ("  select source sound locator\n");
  }
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, (text *)sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                   (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                   (ub4) OCI_DEFAULT));

  return 0;
}
void appendLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Dest_loc, *Src_loc;

  /* Allocate the LOB locators */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Dest_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Src_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the LOBs */
  printf(" select source and destination Lobs\n");
  select_lock_sound_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
                                                    /* destination locator */
  select_lock_sound_locator_2(Src_loc, FALSE, errhp, svchp, stmthp);
                                                         /* source locator */

  /* Opening the LOBs is Optional */
  checkerr (errhp, OCILobOpen(svchp, errhp, Dest_loc, OCI_LOB_READWRITE));
  checkerr (errhp, OCILobOpen(svchp, errhp, Src_loc, OCI_LOB_READONLY));

  /* Append Source LOB to the end of the Destination LOB. */
  printf(" append the source Lob to the destination Lob\n");
```

```
  checkerr(errhp, OCILobAppend(svchp, errhp, Dest_loc, Src_loc));

  /* Closing the LOBs is Mandatory if they have been Opened */
  checkerr (errhp, OCILobClose(svchp, errhp, Dest_loc));
  checkerr (errhp, OCILobClose(svchp, errhp, Src_loc));

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Dest_loc, (ub4) OCI_DTYPE_LOB);
  (void) OCIDescriptorFree((dvoid *) Src_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Append One LOB to Another Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. LOB-APPEND.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID        PIC X(11) VALUES "USER1/USER1".
     01  DEST          SQL-BLOB.
     01  SRC           SQL-BLOB.
         EXEC SQL INCLUDE SQLCA END-EXEC.

     PROCEDURE DIVISION.
     APPEND-BLOB.
         EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
         EXEC SQL CONNECT :USERID END-EXEC.

    * Allocate and initialize the BLOB locators:
         EXEC SQL ALLOCATE :DEST END-EXEC.
         EXEC SQL ALLOCATE :SRC END-EXEC.

         EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

         EXEC SQL
              SELECT SOUND INTO :DEST
              FROM MULTIMEDIA_TAB WHERE CLIP_ID = 2 FOR UPDATE
         END-EXEC.

         EXEC SQL
              SELECT SOUND INTO :SRC
```

```
            FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
        END-EXEC.

    * Open the DESTination LOB read/write and SRC LOB read only:
        EXEC SQL LOB OPEN :DEST READ WRITE END-EXEC.
        EXEC SQL LOB OPEN :SRC READ ONLY END-EXEC.

    * Append the source LOB to the destination LOB:
        EXEC SQL
             LOB APPEND :SRC TO :DEST
        END-EXEC.

        EXEC SQL LOB CLOSE :DEST END-EXEC.
        EXEC SQL LOB CLOSE :SRC END-EXEC.

     END-OF-BLOB.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :DEST END-EXEC.
        EXEC SQL FREE :SRC END-EXEC.
        EXEC SQL COMMIT WORK RELEASE END-EXEC.
        STOP RUN.

     SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED:".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Append One LOB to Another Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void appendLOB_proc()
{
  OCIBlobLocator *Dest_loc, *Src_loc;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the locators: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL ALLOCATE :Src_loc;
  /* Select the destination locator: */
  EXEC SQL SELECT Sound INTO :Dest_loc
           FROM Multimedia_tab WHERE Clip_ID = 2 FOR UPDATE;
  /* Select the source locator: */
  EXEC SQL SELECT Sound INTO :Src_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
  /* Append the source LOB to the end of the destination LOB: */
  EXEC SQL LOB APPEND :Src_loc TO :Dest_loc;
  /* Closing the LOBs is mandatory if they have been opened: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;
  /* Release resources held by the locators: */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  appendLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Append One LOB to Another Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
```

```
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
Set OraSoundClone = OraSound1

OraDyn.MoveNext

OraDyn.Edit
OraSound1.Append OraSoundClone
OraDyn.Update
```

Example: Append One LOB to Another Using Java (JDBC)

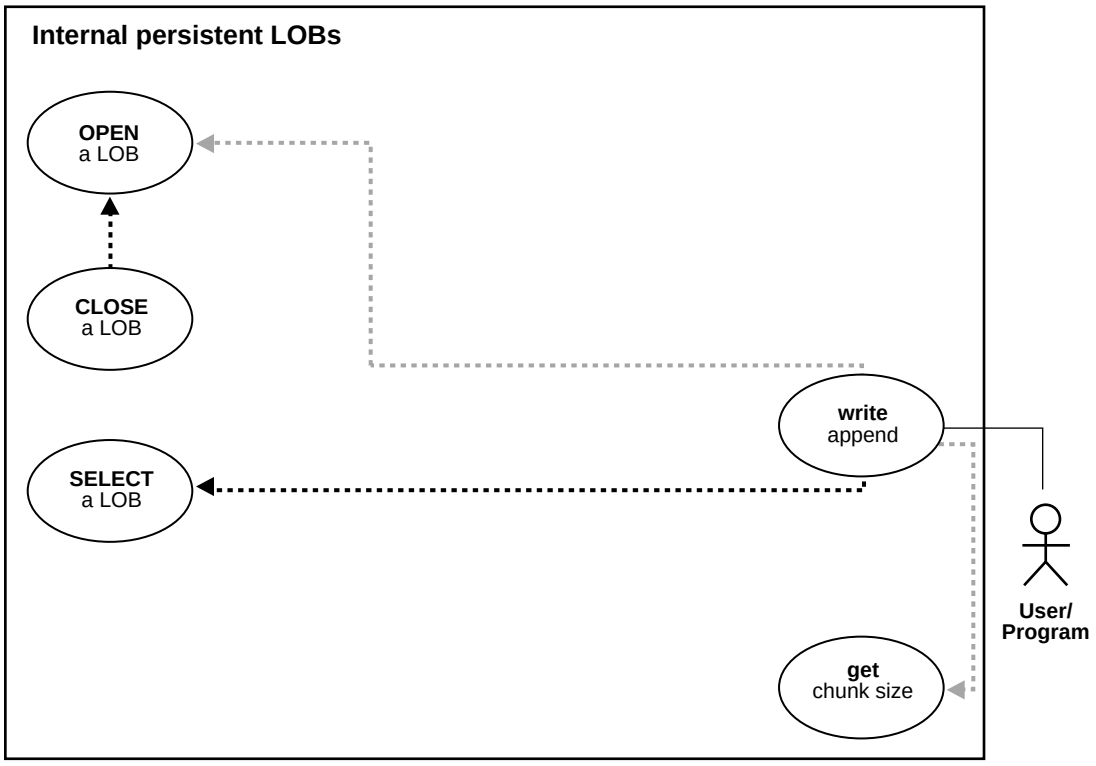
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_121
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
```

```
 Class.forName ("oracle.jdbc.driver.OracleDriver");

 // Connect to the database:
 Connection conn =
   DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

 // It's faster when auto commit is off:
 conn.setAutoCommit (false);

 // Create a Statement:
 Statement stmt = conn.createStatement ();

 try
 {
  ResultSet rset = null;
  BLOB dest_loc = null;
  BLOB src_loc = null;
InputStream in = null;
byte[] buf = new byte[MAXBUFSIZE];
int length = 0;
long pos = 0;

    rset = stmt.executeQuery (
       "SELECT sound FROM multimedia_tab WHERE clip_id = 2");
if (rset.next())
{
  src_loc = ((OracleResultSet)rset).getBLOB (1);
}
in = src_loc.getBinaryStream();

    rset = stmt.executeQuery (
       "SELECT sound FROM multimedia_tab WHERE clip_id = 1 FOR UPDATE");
if (rset.next())
{
  dest_loc = ((OracleResultSet)rset).getBLOB (1);
}

// Start writing at the end of the LOB.  ie. append:
pos = dest_loc.length();


// populate the buffer:
buf = (new String("Hello World")).getBytes();

while ((length = in.read(buf)) != -1)
```

```
      {
         // Write the contents of the buffer into position pos of the output LOB:
         dest_loc.putBytes(pos, buf);
      }

      // Close all streams and handles:
      in.close();
      stmt.close();
      conn.commit();
      conn.close();

       }
       catch (SQLException e)
       {
          e.printStackTrace();
       }
    }
}
```

Write Append to a LOB

Figure 3–34 Use Case Diagram: Write Append to a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Writing Singly or Piecewise

The `writeappend` operation writes a buffer to the end of a LOB. For the OCI, the buffer can be written to the `LOB` in a single piece with this call; alternatively, it can be rendered piecewise using callbacks or a standard polling method. If the value of

the piece parameter is OCI_FIRST_PIECE, data must be provided through callbacks or polling. If a callback function is defined in the cbfp parameter, then this callback function will be invoked to get the next piece after a piece is written to the pipe. Each piece will be written from bufp. If no callback function is defined, then OCILobWriteAppend() returns the OCI_NEED_DATA error code. The application must call OCILobWriteAppend() again to write more pieces of the LOB. In this mode, the buffer pointer and the length can be different in each call if the pieces are of different sizes and from different locations. A piece value of OCI_LAST_PIECE terminates the piecewise write.

Locking the Row Prior to Updating

Prior to updating a LOB value via the PL/SQL DBMS_LOB package or the OCI, you must lock the row containing the LOB. While the SQL INSERT and UPDATE statements implicitly lock the row, locking is done explicitly by means of a SQL SELECT FOR UPDATE statement in SQL and PL/SQL programs, or by using an OCI pin or lock function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

This example demonstrates writing to the end of a video frame (Frame).

- "Example: Write Append to a LOB Using PL/SQL" on page 3-192
- "Example: Write Append to a LOB Using C (OCI)" on page 3-193
- "Example: Write Append to a LOB Using COBOL (Pro*COBOL)" on page 3-195
- "Example: Write Append to a LOB Using Visual Basic (OO4O)" on page 3-197
- "Example: Write Append to a LOB Using Java (JDBC)" on page 3-197

Example: Write Append to a LOB Using PL/SQL

```
/* Note that the example procedure lobWriteAppend_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE lobWriteAppend_proc IS
   Lob_loc    BLOB;
   Buffer     RAW(32767);
   Amount     Binary_integer := 32767;
BEGIN
   SELECT Frame INTO Lob_loc FROM Multimedia_tab where Clip_ID = 1 FOR UPDATE;
   /* Fill the buffer with data... */
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Append the data from the buffer to the end of the LOB: */
   DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Write Append to a LOB Using C (OCI)

```
/* Select the locator into a locator variable: */

sb4 select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *)"SELECT Frame FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return (0);
}

#define MAXBUFLEN 32767

void writeAppendLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
```

```
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIBlobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  OCILobLocator *Lob_Loc;

  /* Allocate the Source (bfile) & destination (blob) locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf(" select and lock a frame locator\n");
  select_lock_frame_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Setting the amt to the buffer length.  Note here that amt is in bytes
     since we are using a BLOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);

  /* Fill bufp with data: */
  /* Write the data from the buffer at the end of the LOB: */
  printf(" write-append data to the frame Lob\n");
  checkerr (errhp, OCILobWriteAppend (svchp, errhp, Lob_loc, &amt,
                            bufp, buflen,
                            OCI_ONE_PIECE, (dvoid *)0,
                            (sb4 (*)(dvoid *, dvoid *, ub4 *, ub1 *))0,
                            0, SQLCS_IMPLICIT));

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Write Append to a LOB Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. WRITE-APPEND-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  BLOB1         SQL-BLOB.
       01  AMT           PIC S9(9) COMP.
       01  BUFFER        PIC X(32767) VARYING.
           EXEC SQL VAR BUFFER IS LONG RAW (32767) END-EXEC.
       01  USERID   PIC X(11) VALUES "USER1/USER1".
           EXEC SQL INCLUDE SQLCA END-EXEC.


       PROCEDURE DIVISION.
       WRITE-APPEND-BLOB.

           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
                CONNECT :USERID
           END-EXEC.

      * Allocate and initialize the BLOB locators:
           EXEC SQL ALLOCATE :BLOB1 END-EXEC.

           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
           EXEC SQL
                SELECT FRAME INTO :BLOB1
                FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1 FOR UPDATE
           END-EXEC.


      * Open the target LOB:
           EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
1
      *    Populate AMT here:
           MOVE 5 TO AMT.
           MOVE "2424242424" to BUFFER.

      * Append the source LOB to the destination LOB:
           EXEC SQL
                LOB WRITE APPEND :AMT FROM :BUFFER INTO :BLOB1
           END-EXEC.
```

```
            EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

        END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BLOB1 END-EXEC.
            EXEC SQL
                 COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED:".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Write Append to a LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 128

void LobWriteAppend_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
```

```
  /* Amount == BufferLength so only a single WRITE is needed: */
  char Buffer[BufferLength];
  /* Datatype equivalencing is mandatory for this datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Frame INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc;
  memset((void *)Buffer, 1, BufferLength);
  /* Write the data from the buffer at the end of the LOB: */
  EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Lob_loc;
  /* Closing the LOB is mandatory if it has been opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  LobWriteAppend_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Write Append to a LOB Using Visual Basic (OO4O)

Note: A Visual Basic example will be made available in a subsequent release.

Example: Write Append to a LOB Using Java (JDBC)

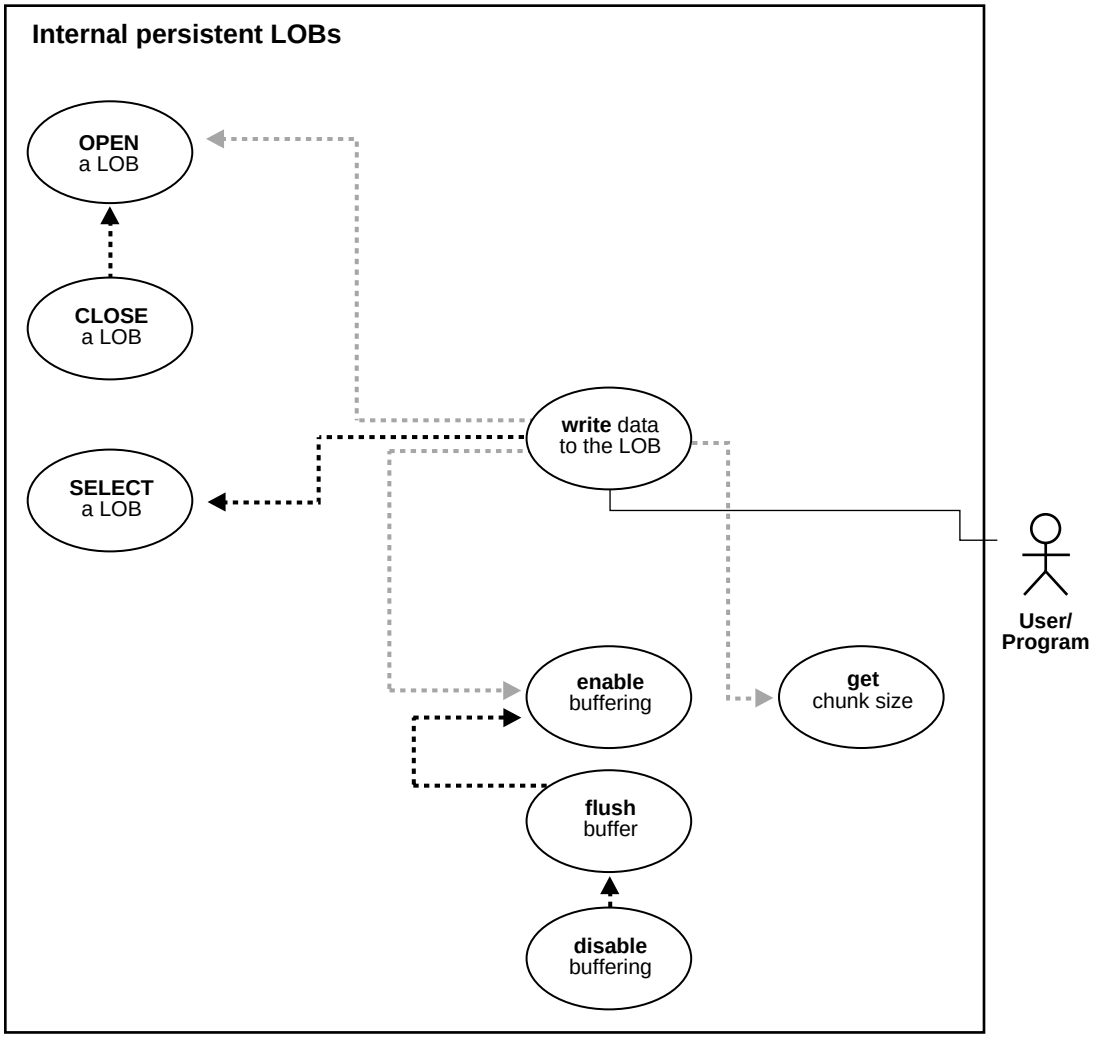
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
```

```
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_126
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BLOB dest_loc = null;
     byte[] buf = new byte[MAXBUFSIZE];
     long pos = 0;

      ResultSet rset = stmt.executeQuery (
         "SELECT frame FROM multimedia_tab WHERE clip_id = 1 FOR UPDATE");
   if (rset.next())
   {
     dest_loc = ((OracleResultSet)rset).getBLOB (1);
   }

   // Start writing at the end of the LOB.  ie. append:
   pos = dest_loc.length();
```

```
   // fill buf with contents to be written:
   buf = (new String("Hello World")).getBytes();

   // Write the contents of the buffer into position pos of the output LOB:
   dest_loc.putBytes(pos, buf);

   // Close all streams and handles:
   stmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Write Data to a LOB

Figure 3–35 Use Case Diagram: Write data to a LOB



To refer to the table of all basic operations having to do with Temporary LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Stream Write

The most efficient way to write large amounts of LOB data is to use `OCILobWrite()` with the streaming mechanism enabled via polling or a callback. If you know how much data will be written to the `LOB`, specify that amount when calling `OCILobWrite()`. This will allow for the contiguity of the `LOB` data on disk. Apart from being spatially efficient, the contiguous structure of the `LOB` data will make for faster reads and writes in subsequent operations.

Chunksize

A chunk is one or more Oracle blocks. As noted previously, you can specify the chunk size for the `LOB` when creating the table that contains the `LOB`. This corresponds to the chunk size used by Oracle when accessing / modifying the `LOB` value. Part of the chunk is used to store system-related information and the rest stores the `LOB` value. The `getchunksize` function returns the amount of space used in the `LOB` chunk to store the `LOB` value.

You will improve performance if the you execute `write` requests using a multiple of this chunk size. The reason for this is that the `LOB` chunk is versioned for every `write` operation. If all `writes` are done on a chunk basis, no extra or excess versioning is incurred or duplicated. If it is appropriate for your application, you should batch writes until you have enough for an entire chunk instead of issuing several `LOB` write calls that operate on the same `LOB` chunk.

Locking the Row Prior to Updating

Prior to updating a `LOB` value via the PL / SQL `DBMS_LOB` package or the OCI, you must lock the row containing the `LOB`. While the SQL `INSERT` and `UPDATE` statements implicitly lock the row, locking is done explicitly by means of a SQL `SELECT FOR UPDATE` statement in SQL and PL / SQL programs, or by using an `OCI pin` or `lock` function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

The following example procedure allows the STORY data (the storyboard for the clip) to be updated by writing data to the LOB.

- "Example: Write Data to a LOB Using the DBMS_LOB Package" on page 3-202
- "Example: Write Data to a LOB Using C (OCI)" on page 3-203
- "Example: Write Data to a LOB Using COBOL (Pro*COBOL)" on page 3-207
- "Example: Write Data to a LOB Using C++ (Pro*C / C++)" on page 3-209
- "Example: Write Data to a LOB Using Visual Basic (OO4O)" on page 3-212
- "Example: Write Data to a LOB Using Java (JDBC)" on page 3-213

Example: Write Data to a LOB Using the DBMS_LOB Package

```
/* Note that the example procedure writeDataToLOB_proc is not part of the
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
   Lob_loc         CLOB;
   Buffer          VARCHAR2(32767);
   Amount          BINARY_INTEGER := 32767;
   Position        INTEGER := 1;
   i               INTEGER;
BEGIN
   /* Select a LOB: */
   SELECT Story INTO Lob_loc
        FROM Multimedia_tab
           WHERE Clip_ID = 1
              FOR UPDATE;
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Fill the buffer with data to write to the LOB: */
   FOR i IN 1..3 LOOP
      DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
      /* Fill the buffer with more data to write to the LOB: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
END;

/* We add a second example to show a case in which the buffer size and amount
   differs from the first example: */
CREATE or REPLACE PROCEDURE writeDataToLOB_proc IS
```

```
   Lob_loc         CLOB;
   Buffer          VARCHAR2(32767);
   Amount          BINARY_INTEGER := 32767;
   Position        INTEGER;
   i               INTEGER;
   Chunk_size      INTEGER;
BEGIN
    SELECT Story INTO Lob_loc
        FROM Multimedia_tab
           WHERE Clip_ID = 1
              FOR UPDATE;
     /* Opening the LOB is optional: */
    DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);

    Chunk_size := DBMS_LOB.GETCHUNKSIZE(Lob_loc);

    /* Fill the buffer with 'Chunk_size' worth of data to write to
       the LOB. Use the chunk size (or a multiple of chunk size) when writing
       data to the LOB.  Make sure that you write within a chunk boundary and
       don't overlap different chunks within a single call to DBMS_LOB.WRITE. */

    Amount := Chunk_size;

    /* Write data starting at the beginning of the second chunk: */
    Position := Chunk_size + 1;

    FOR i IN 1..3 LOOP
        DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
        /* Fill the buffer with more data (of size Chunk_size) to  write to
           the LOB: */
        Position := Position + Amount;
    END LOOP;
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE (Lob_loc);
END;
```

Example: Write Data to a LOB Using C (OCI)

```
/* This example demonstrates how OCI provides for the ability to write
   arbitrary amounts of data to an Internal LOB in either a single piece
   or in multiple pieces using a streaming mechanism that utilizes standard
   polling.  A dynamically allocated Buffer is used to hold the data being
   written to the LOB. */

/* Select the locator into a locator variable */
```

```
sb4 select_lock_story_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
    (text *) "SELECT Story FROM Multimedia_tab m  \
                WHERE m.Clip_ID = 1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));
  return (0);
}

void writeDataToLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCIClobLocator *Lob_loc;
  ub4 Total = 2.5*MAXBUFLEN;
                    /* <total amount of data to write to the CLOB in bytes> */
  unsigned int amt;
  unsigned int offset;
  unsigned int remainder, nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sb4 err;

  /* Allocate the locators desriptors*/
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
```

```
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

/* Select the CLOB */
printf (" select a story Lob\n");
select_lock_story_locator(Lob_loc, errhp, svchp, stmthp);

/* Open the CLOB */
checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

if (Total > MAXBUFLEN)
  nbytes = MAXBUFLEN;   /* We will use streaming via standard polling */
else
  nbytes = Total;                     /* Only a single write is required */

/* Fill the buffer with nbytes worth of data */
remainder = Total - nbytes;

/* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */
amt = 0;
offset = 1;

printf(" write the Lob data in pieces\n");
if (0 == remainder)
{
  amt = nbytes;
  /* Here, (Total <= MAXBUFLEN ) so we can write in one piece */
  checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                                offset, bufp, nbytes,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));
}
else
{
  /* Here (Total > MAXBUFLEN ) so we use streaming via standard polling */
  /* write the first piece.  Specifying first initiates polling. */
  err =  OCILobWrite (svchp, errhp, Lob_loc, &amt,
                      offset, bufp, nbytes,
                      OCI_FIRST_PIECE, (dvoid *)0,
                      (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                      0, SQLCS_IMPLICIT);

  if (err != OCI_NEED_DATA)
    checkerr (errhp, err);
```

```
  last = FALSE;
  /* Write the next (interim) and last pieces */
  do
  {
    if (remainder > MAXBUFLEN)
      nbytes = MAXBUFLEN;            /* Still have more pieces to go */
    else
    {
      nbytes = remainder;       /* Here, (remainder <= MAXBUFLEN) */
      last = TRUE;              /* This is going to be the final piece */
    }

    /* Fill the Buffer with nbytes worth of data */

    if (last)
    {
      /* Specifying LAST terminates polling */
      err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                         offset, bufp, nbytes,
                         OCI_LAST_PIECE, (dvoid *)0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_SUCCESS)
        checkerr(errhp, err);
    }
    else
    {
      err = OCILobWrite (svchp, errhp, Lob_loc, &amt,
                         offset, bufp, nbytes,
                         OCI_NEXT_PIECE, (dvoid *)0,
                         (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                         0, SQLCS_IMPLICIT);
      if (err != OCI_NEED_DATA)
        checkerr (errhp, err);
    }
    /* Determine how much is left to write */
    remainder = remainder - nbytes;
  } while (!last);
}

/* At this point, (remainder == 0) */

/* Closing the LOB is mandatory if you have opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));
```

```
  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Write Data to a LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-CLOB.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT INFILE
      ASSIGN TO "datfile.dat"
      ORGANIZATION IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.

FD INFILE
   RECORD CONTAINS 5 CHARACTERS.
01 INREC      PIC X(5).

WORKING-STORAGE SECTION.

01  CLOB1          SQL-CLOB.
01  BUFFER         PIC X(5) VARYING.
01  AMT            PIC S9(9) COMP VALUES 321.
01  OFFSET         PIC S9(9) COMP VALUE 1.
01  END-OF-FILE    PIC X(1) VALUES "N".

01  D-BUFFER-LEN   PIC 9.
01  D-AMT          PIC 9.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
WRITE-CLOB.

    EXEC SQL WHENEVER SQLERROR GOTO SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.
```

```
* Open the input file:
     OPEN INPUT INFILE.
* Allocate and initialize the CLOB locator:
     EXEC SQL ALLOCATE :CLOB1 END-EXEC.

     EXEC SQL
          SELECT STORY INTO :CLOB1 FROM MULTIMEDIA_TAB
          WHERE CLIP_ID = 1 FOR UPDATE
     END-EXEC.

* Either write entire record or write first piece
* Read a data file here and populate BUFFER-ARR and BUFFER-LEN
* END-OF-FILE will be set to "Y" when the entire file has been
* read.
     PERFORM READ-NEXT-RECORD.
     MOVE INREC TO BUFFER-ARR.
     MOVE 5 TO BUFFER-LEN.
     IF (END-OF-FILE = "Y")
        EXEC SQL
             LOB WRITE ONE :AMT FROM :BUFFER
             INTO :CLOB1 AT :OFFSET
        END-EXEC
     ELSE
        DISPLAY "LOB WRITE FIRST: ", BUFFER-ARR
        EXEC SQL
           LOB WRITE FIRST :AMT FROM :BUFFER
           INTO :CLOB1 AT :OFFSET
        END-EXEC.

* Continue reading from the input data file
* and writing to the CLOB:
     PERFORM READ-NEXT-RECORD.
     PERFORM WRITE-TO-CLOB
        UNTIL END-OF-FILE = "Y".

     MOVE INREC TO BUFFER-ARR.
     MOVE 1 TO BUFFER-LEN.
     DISPLAY "LOB WRITE LAST: ", BUFFER-ARR(1:BUFFER-LEN).
     EXEC SQL
          LOB WRITE LAST :AMT FROM :BUFFER INTO :CLOB1
     END-EXEC.
     EXEC SQL
          COMMIT WORK RELEASE
     END-EXEC.
     STOP RUN.
```

```
WRITE-TO-CLOB.
   MOVE INREC TO BUFFER-ARR.
   MOVE 5 TO BUFFER-LEN.
   DISPLAY "LOB WRITE NEXT: ", BUFFER-ARR(1:BUFFER-LEN).
   EXEC SQL
        LOB WRITE NEXT :AMT FROM :BUFFER INTO :CLOB1
   END-EXEC.
   PERFORM READ-NEXT-RECORD.

READ-NEXT-RECORD.
   MOVE SPACES TO INREC.
   READ INFILE NEXT RECORD
        AT END
         MOVE "Y" TO END-OF-FILE.

SQL-ERROR.
   EXEC SQL
       WHENEVER SQLERROR CONTINUE
   END-EXEC.
   DISPLAY " ".
   DISPLAY "ORACLE ERROR DETECTED:".
   DISPLAY " ".
   DISPLAY SQLERRMC.
   EXEC SQL
       ROLLBACK WORK RELEASE
   END-EXEC.
   STOP RUN.
```

Example: Write Data to a LOB Using C++ (Pro*C/C++)

```
/* This example demonstrates how Pro*C/C++ provides for the ability to write
   arbitrary amounts of data to an Internal LOB in either a single piece
   of in multiple pieces using a Streaming Mechanism that utilizes standard
   polling.  A dynamically allocated Buffer is used to hold the data being
   written to the LOB: */
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void writeDataToLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Lob_loc;
  varchar Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Story INTO Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Open the CLOB: */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  Total = Amount = (multiple * BufferLength);
  if (Total > BufferLength)
    nbytes = BufferLength;   /* We will use streaming via standard polling */
  else
    nbytes = Total;                     /* Only a single write is required */
  /* Fill the buffer with nbytes worth of data: */
  memset((void *)Buffer.arr, 32, nbytes);
  Buffer.len = nbytes;       /* Set the Length */
  remainder = Total - nbytes;
  if (0 == remainder)
    {
      /* Here, (Total <= BufferLength) so we can write in one piece: */
      EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Lob_loc;
      printf("Write ONE Total of %d characters\n", Amount);
    }
  else
    {
      /* Here (Total > BufferLength) so we streaming via standard polling */
      /* write the first piece.  Specifying first initiates polling: */
      EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Lob_loc;
      printf("Write first %d characters\n", Buffer.len);
      last = FALSE;
```

```
      /* Write the next (interim) and last pieces: */
      do
        {
          if (remainder > BufferLength)
            nbytes = BufferLength;        /* Still have more pieces to go */
          else
            {
              nbytes = remainder;    /* Here, (remainder <= BufferLength) */
              last = TRUE;         /* This is going to be the Final piece */
            }
          /* Fill the buffer with nbytes worth of data: */
          memset((void *)Buffer.arr, 32, nbytes);
          Buffer.len = nbytes;       /* Set the Length */
          if (last)
            {
              EXEC SQL WHENEVER SQLERROR DO Sample_Error();
              /* Specifying LAST terminates polling: */
              EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Lob_loc;
              printf("Write LAST Total of %d characters\n", Amount);
            }
          else
            {
              EXEC SQL WHENEVER SQLERROR DO break;
              EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Lob_loc;
              printf("Write NEXT %d characters\n", Buffer.len);
            }
          /* Determine how much is left to write: */
          remainder = remainder - nbytes;
        } while (!last);
    }
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* At this point, (Amount == Total), the total amount that was written */
  /* Close the CLOB: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeDataToLOB_proc(1);
  EXEC SQL ROLLBACK WORK;
  writeDataToLOB_proc(4);
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Write Data to a LOB Using Visual Basic (OO4O)

```
'Note that this code fragment assumes an orablob object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1. There are two ways
'of writing a lob using  orablob.write or orablob.copyfromfile

'Using OraBlob.Write mechanism
Dim OraDyn as OraDynaset, OraSound as OraBlob, amount_written%, chunksize%,
curchunk

chunksize = 32768
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound")

OraSound.offset = 1
OraSound.pollingAmount = LOF(fnum)

Dim piece As Byte
Get #fnum, , curchunk

piece = ORALOB_FIRST_PIECE
amount_written = OraSound.Write(curchunk, chunksize, ORALOB_FIRST_PIECE)

While OraSound.Status = ORALOB_NEED_DATA
   If amount_written <= chunksize Then
      piece = ORALOB_LAST_PIECE
   Else
      piece = ORALOB_NEXT_PIECE
   End If

   Get #fnum, , curchunk
   amount_written = OraSound.Write(curchunk, chunksize, piece)

Wend

'Using OraBlob.CopyFromFile mechanism
Dim OraDyn as OraDynaset, OraSound as OraBlob, amount_read%, chunksize%, chunk

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraSound = OraDyn.Fields("Sound").Value
```

```
OraSound.CopyFromFile "c:\mysound.aud"
```

Example: Write Data to a LOB Using Java (JDBC)

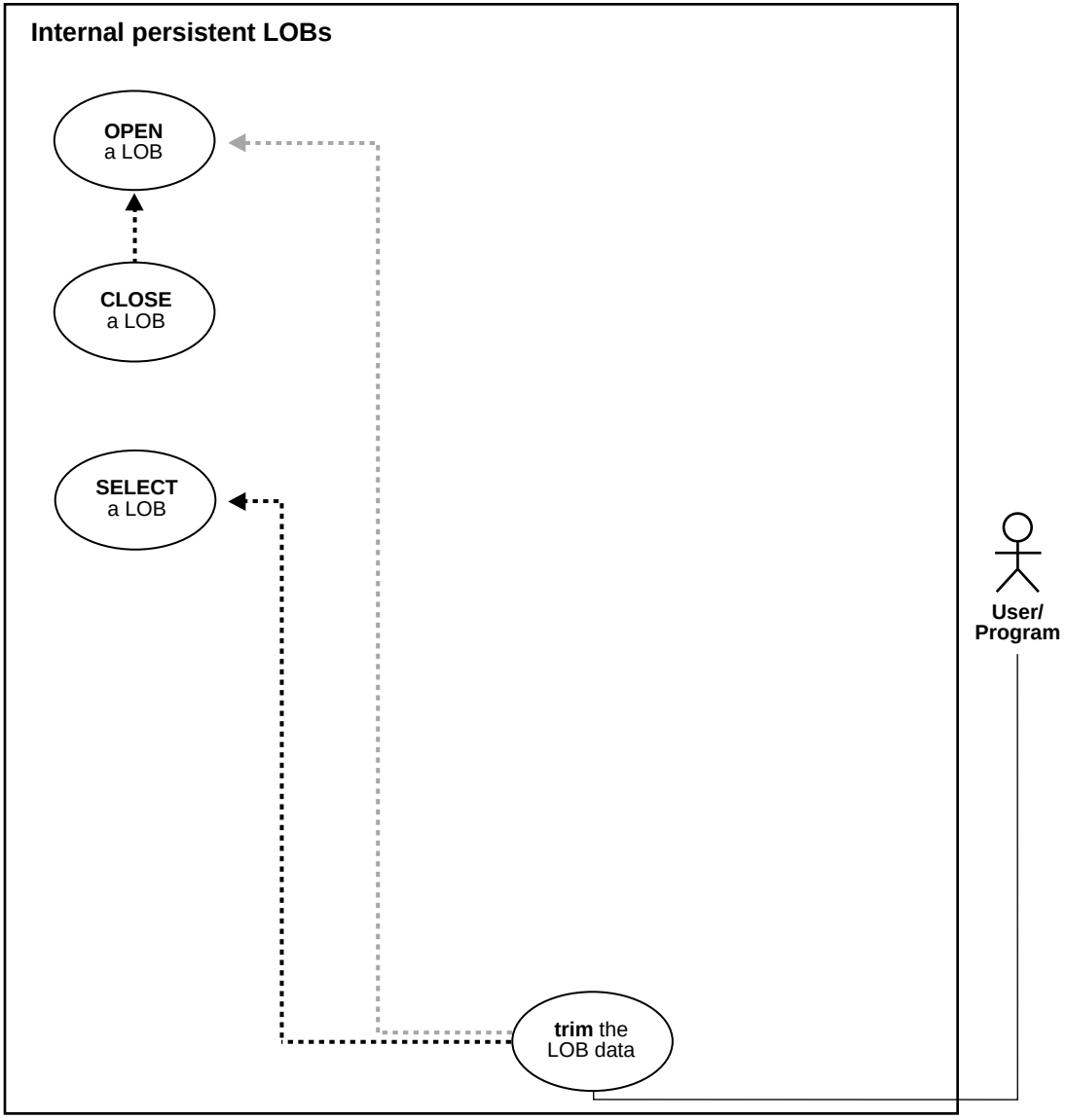
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_66
{
  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     CLOB lob_loc = null;
```

```
      String buf = new String ("Some Text To Write");

      ResultSet rset = stmt.executeQuery (
         "SELECT intab.transcript FROM TABLE(
             SELECT mtab.inseg_ntab FROM multimedia_tab mtab
                WHERE mtab.clip_id = 1) intab WHERE intab.segment=1 FOR UPDATE");

   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

       OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("BEGIN DBMS_LOB.OPEN( ?,
                            DBMS_LOB.LOB_READWRITE); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   long pos = 0;       // This is the offset within the CLOB where the data is
to be written
   long length = 0;        // This is the size of the buffer to be written.

   // This loop writes the buffer three times consecutively:
   for (int i = 0; i < 3; i++)
   {
      // Fill the buffer with some data to be written:
      length = buf.length();
      pos += length;
      // This is an Oracle-specific method:
      lob_loc.plsql_write(pos, buf.toCharArray());
   }

   // All OPENed LOBS must be CLOSEd:
       cstmt = (OracleCallableStatement) conn.prepareCall (
          "BEGIN DBMS_LOB.CLOSE(?); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   stmt.close();
   cstmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
```

```
    {
       e.printStackTrace();
    }
  }
}
```

Trim the LOB Data

Figure 3–36 Use Case Diagram: Trim the LOB data



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Locking the Row Prior to Updating

Prior to updating a LOB value via the PL/SQL DBMS_LOB package or the OCI, you must lock the row containing the LOB. While the SQL INSERT and UPDATE statements implicitly lock the row, locking is done explicitly by means of a SQL SELECT FOR UPDATE statement in SQL and PL/SQL programs, or by using an OCI pin or lock function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

Our example accesses text (CLOB data) that is referenced in the Script column of the table Voiceover_tab, and trims it.

- "Example: Trim the LOB Data Using PL/SQL (DBMS_LOB Package)" on page 3-217
- "Example: Trim the LOB Data Using C (OCI)" on page 3-218
- "Example: Trim the LOB Data Using COBOL (Pro*COBOL)" on page 3-219
- "Example: Trim the LOB Data Using C++ (Pro*C/C++)" on page 3-221
- "Example: Trim the LOB Data Using Visual Basic (OO4O)" on page 3-223
- "Example: Trim the LOB Data Using Java (JDBC)" on page 3-223

Example: Trim the LOB Data Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure trimLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE trimLOB_proc IS
   Lob_loc        CLOB;
BEGIN
   /* Select the LOB, get the LOB locator: */
   SELECT Mtab.Voiced_ref.Script INTO Lob_loc FROM Multimedia_tab Mtab
      WHERE Mtab.Clip_ID = 2
         FOR UPDATE;
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Trim the LOB data: */
   DBMS_LOB.TRIM(Lob_loc,100);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
COMMIT;
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Trim the LOB Data Using C (OCI)

```
/* Select the locator into a locator variable */
sb4 select_lock_voice_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
      (text *) "SELECT Mtab.Voiced_ref.Script \
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2 FOR UPDATE";

  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_CLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return 0;
}
```

```
void trimLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  unsigned int trimLength;

  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the CLOB */
  printf( " select a voice LOB\n");
  select_lock_voice_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the CLOB */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Trim the LOB to its new length */
  trimLength = 100;                       /* <New truncated length of the LOB>*/

  printf (" trim the lob to %d bytes\n", trimLength);
  checkerr (errhp, OCILobTrim (svchp, errhp, Lob_loc, trimLength ));

  /* Closing the CLOB is mandatory if you have opend it */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  /* Free resources held by the locators*/
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);
}
```

Example: Trim the LOB Data Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TRIM-CLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  CLOB1          SQL-CLOB.
01  NEW-LEN        PIC S9(9) COMP.
* Define the source and destination position and location:
```

```
01  SRC-POS        PIC S9(9) COMP.
01  DEST-POS       PIC S9(9) COMP.
01  SRC-LOC        PIC S9(9) COMP.
01  DEST-LOC       PIC S9(9) COMP.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
TRIM-CLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locators:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
    EXEC SQL
         SELECT MTAB.STORY INTO :CLOB1
         FROM MULTIMEDIA_TAB MTAB
         WHERE MTAB.CLIP_ID = 2 FOR UPDATE
    END-EXEC.

* Open the CLOB:
    EXEC SQL LOB OPEN :CLOB1 READ WRITE END-EXEC.

* Move some value to  NEW-LEN:
    MOVE 3 TO NEW-LEN.
    EXEC SQL
         LOB TRIM :CLOB1 TO :NEW-LEN
    END-EXEC.

    EXEC SQL LOB CLOSE :CLOB1 END-EXEC.

END-OF-CLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :CLOB1 END-EXEC.
    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
```

```
    WHENEVER SQLERROR CONTINUE
END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

Example: Trim the LOB Data Using C++ (Pro*C/C++)

Note: In addition to the data structures set up above in the section "Example: Create a Table Containing One or More LOB Columns using SQL DDL" on page 3-15, you should use DML like this:

```
INSERT INTO multimedia_tab VALUES (
  2, 'The quick brown fox jumped over the lazy dog',
  empty_clob(), NULL, empty_blob(), empty_blob(), NULL, NULL,
NULL, NULL);

INSERT INTO voiceover_tab VALUES (
  voiced_typ('hello',
  (SELECT story FROM multimedia_tab WHERE clip_id = 2),
  'world', 1, NULL))

UPDATE multimedia_tab SET voiced_ref =
  (SELECT REF(r) FROM voiceover_tab r WHERE r.take = 1)
  WHERE clip_id = 2
```

Then create this text file, pers_trim.typ, containing:

```
case=lower
type voiced_typ
```

Then run this Object Type Translator command:

```
ott intyp=pers_trim.typ outtyp=pers_trim_o.typ
   hfile=pers_trim.h code=c user=samp/samp
```

```
#include "pers_trim.h"
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("sqlcode = %ld\n", sqlca.sqlcode);
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void trimLOB_proc()
{
  voiced_typ_ref *vt_ref;
  voiced_typ *vt_typ;
  OCIClobLocator *Lob_loc;
  unsigned int Length, trimLength;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :vt_ref;
  EXEC SQL ALLOCATE :vt_typ;
  /* Retrieve the REF using Associative SQL */
  EXEC SQL SELECT Mtab.Voiced_ref INTO :vt_ref
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2 FOR UPDATE;
  /* Dereference the Object using the Navigational Interface */
  EXEC SQL OBJECT DEREF :vt_ref INTO :vt_typ FOR UPDATE;
  Lob_loc = vt_typ->script;
  /* Opening the LOB is Optional */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  printf("Old length was %d\n", Length);
  trimLength = (unsigned int)(Length / 2);
  /* Trim the LOB to its new length */
  EXEC SQL LOB TRIM :Lob_loc TO :trimLength;
  /* Closing the LOB is mandatory if it has been opened */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Mark the Object as Modified (Dirty) */
  EXEC SQL OBJECT UPDATE :vt_typ;
  /* Flush the changes to the LOB in the Object Cache */
  EXEC SQL OBJECT FLUSH :vt_typ;
  /* Display the new (modified) length */
  EXEC SQL SELECT Mtab.Voiced_ref.Script INTO :Lob_loc
```

```
          FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 2;
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  printf("New length is now %d\n", Length);
  /* Free the Objects and the LOB Locator */
  EXEC SQL FREE :vt_ref;
  EXEC SQL FREE :vt_typ;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  trimLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Trim the LOB Data Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value

OraDyn.Edit
OraSound1.Trim 10
OraDyn.Update
```

Example: Trim the LOB Data Using Java (JDBC)

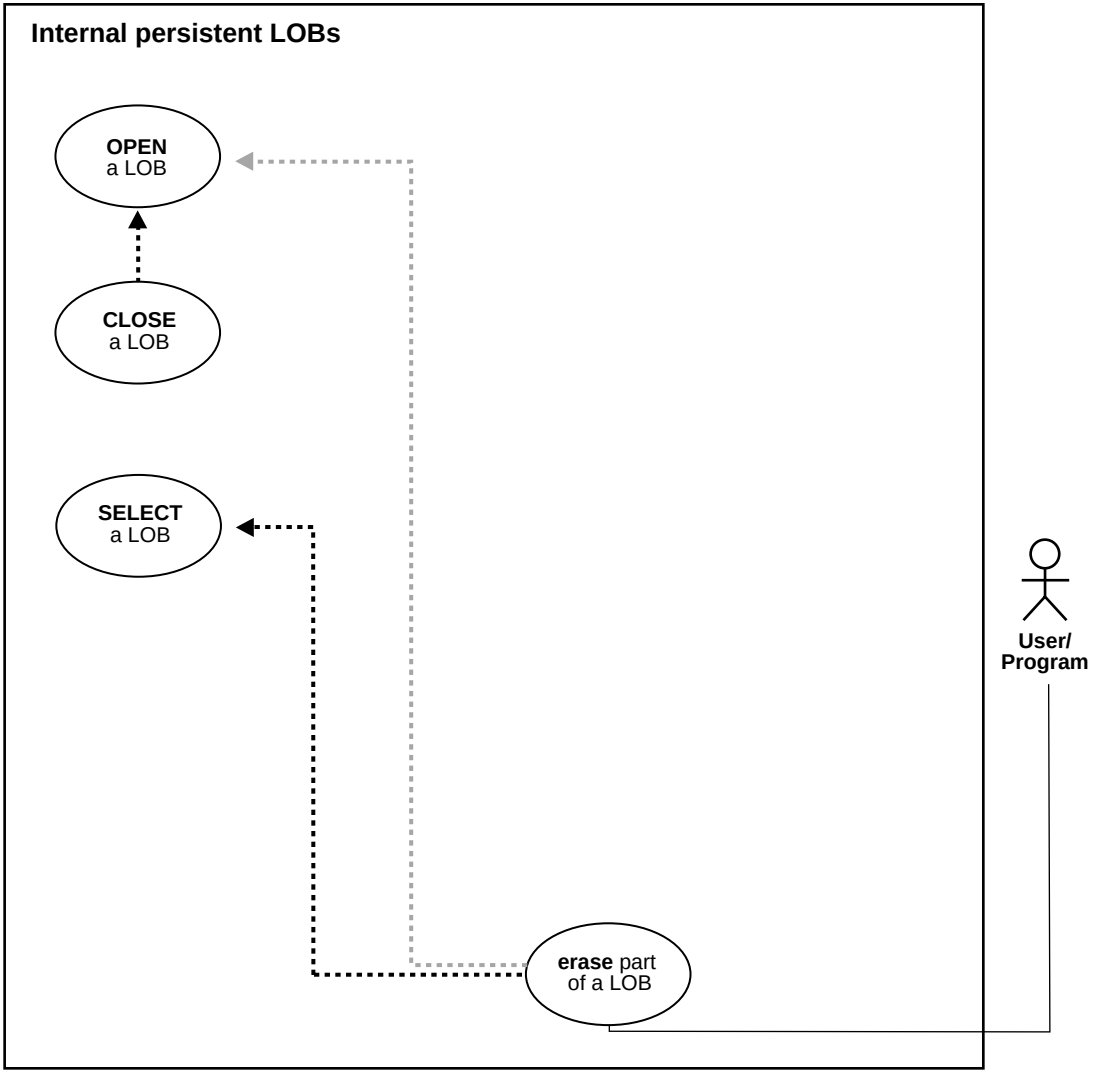
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
```

```
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_141
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     CLOB lob_loc = null;

     ResultSet rset = stmt.executeQuery (
        "SELECT mtab.voiced_ref.script FROM multimedia_tab mtab
            WHERE mtab.clip_id = 2 FOR UPDATE");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getCLOB (1);
   }

   // Open the LOB for READWRITE:
```

```
       OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READWRITE);
END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   // Trim the LOB to length of 400:
   cstmt = (OracleCallableStatement)
      conn.prepareCall ("BEGIN DBMS_LOB.TRIM(?, 400); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   // Close the LOB:
   cstmt = (OracleCallableStatement) conn.prepareCall (
      "BEGIN DBMS_LOB.CLOSE(?); END;");
   cstmt.setCLOB(1, lob_loc);
   cstmt.execute();

   stmt.close();
   cstmt.close();
   conn.commit();
   conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Erase Part of a LOB

Figure 3–37 Use Case Diagram: Erase part of a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Locking the Row Prior to Updating

Prior to updating a LOB value via the PL/SQL DBMS_LOB package or the OCI, you must lock the row containing the LOB. While the SQL INSERT and UPDATE statements implicitly lock the row, locking is done explicitly by means of a SQL SELECT FOR UPDATE statement in SQL and PL/SQL programs, or by using an OCI pin or lock function in OCI programs. For more details on the state of the locator after an update, refer to "Updated locators" on page 2-5 in Chapter 2, "Advanced Topics".

Scenario

The example demonstrates erasing a portion of sound (Sound).

- "Example: Erase Part of a LOB Using PL/SQL (DBMS_LOB Package)" on page 3-227
- "Example: Erase Part of a LOB Using C (OCI)" on page 3-228
- "Example: Erase Part of a LOB Using COBOL (Pro*COBOL)" on page 3-229
- "Example: Erase Part of a LOB Using C++ (Pro*C/C++)" on page 3-231
- "Example: Erase Part of a LOB Using Visual Basic (OO4O)" on page 3-232
- "Example: Erase Part of a LOB Using Java (JDBC)" on page 3-232

Example: Erase Part of a LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure eraseLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE eraseLOB_proc IS
   Lob_loc        BLOB;
   Amount         INTEGER := 3000;
BEGIN
   /* Select the LOB, get the LOB locator: */
   SELECT Sound INTO lob_loc FROM Multimedia_tab
      WHERE Clip_ID = 1
         FOR UPDATE;
```

```
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Erase the data: */
   DBMS_LOB.ERASE(Lob_loc, Amount, 2000);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
COMMIT;
/* Exception handling: */
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Erase Part of a LOB Using C (OCI)

```
/* Select the locator into a locator variable: */

sb4 select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return 0;
}
```

```
void eraseLob(envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amount = 3000;
  ub4 offset = 2000;

  OCILobLocator *Lob_Loc;

  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the CLOB: */
  printf( " select and lock a sound LOB\n");
  select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Erase the data starting at the specified Offset: */
  printf(" erase %d bytes from the sound Lob\n", amount);
  checkerr (errhp, OCILobErase (svchp, errhp, Lob_loc, &amount, offset ));

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;
}
```

Example: Erase Part of a LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ERASE-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
```

```
01  BLOB1          SQL-BLOB.
01  AMT            PIC S9(9) COMP.
01  OFFSET         PIC S9(9) COMP.
    EXEC SQL INCLUDE SQLCA END-EXEC.


PROCEDURE DIVISION.
ERASE-BLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.
* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :BLOB1
         FROM MULTIMEDIA_TAB MTAB
         WHERE MTAB.CLIP_ID = 2 FOR UPDATE
    END-EXEC.

* Open the BLOB:
    EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.

* Move some value to AMT and OFFSET:
    MOVE 2 TO AMT.
    MOVE 1 TO OFFSET.
    EXEC SQL
         LOB ERASE :AMT FROM :BLOB1 AT :OFFSET
    END-EXEC.

    EXEC SQL LOB CLOSE :BLOB1 END-EXEC.

END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.
    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
```

```
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED:".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Erase Part of a LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void eraseLob_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = 5;
  int Offset = 5;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Opening the LOB is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ WRITE;
  /* Erase the data starting at the specified Offset: */
  EXEC SQL LOB ERASE :Amount FROM :Lob_loc AT :Offset;
  printf("Erased %d bytes\n", Amount);
  /* Closing the LOB is mandatory if it has been opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  eraseLob_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Erase Part of a LOB Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab ORDER BY clip_
id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
'Erase 10 bytes begining from the 100th byte:
OraDyn.Edit
OraSound1.Erase 10, 100
OraDyn.Update
```

Example: Erase Part of a LOB Using Java (JDBC)

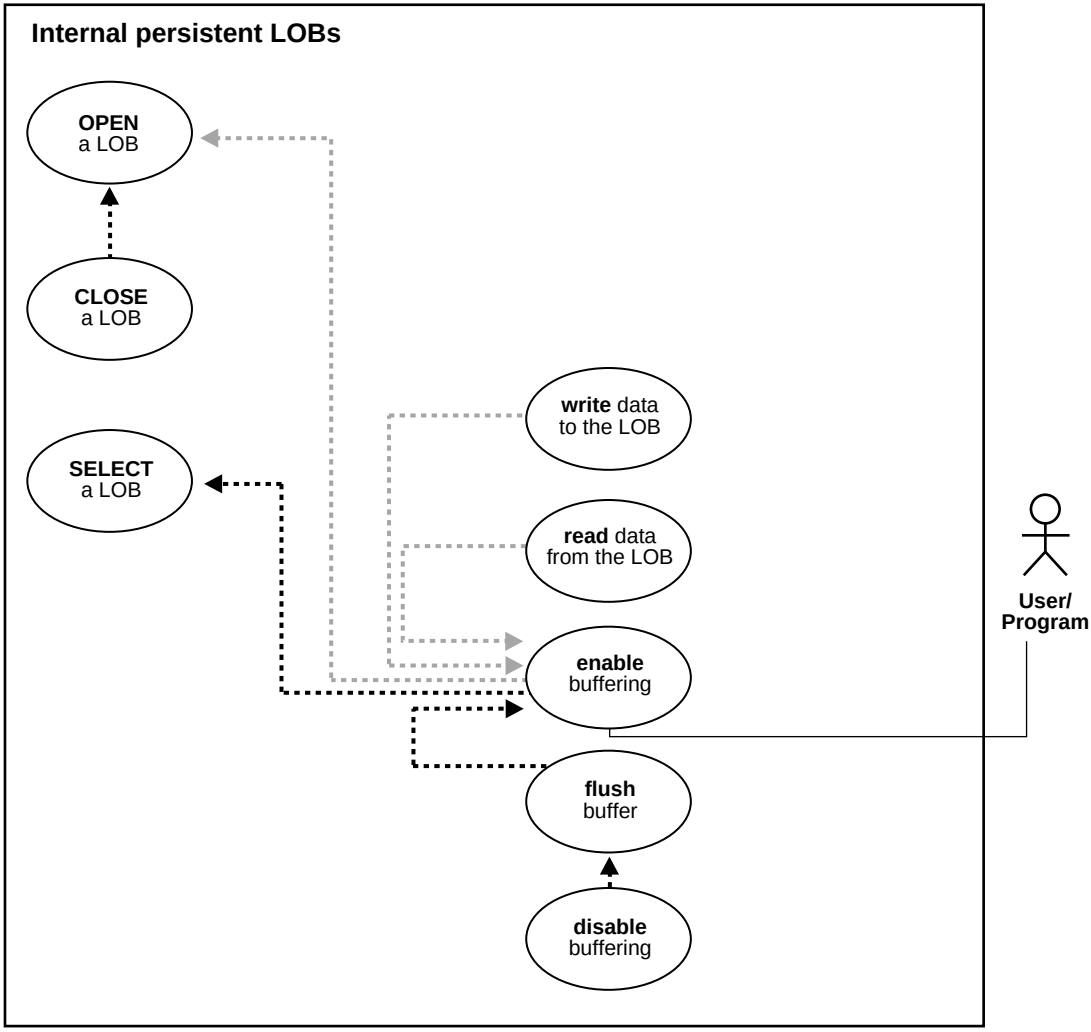
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
```

```
import oracle.jdbc.driver.*;

public class Ex2_145
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     BLOB lob_loc = null;
     int eraseAmount = 30;


     ResultSet rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
   if (rset.next())
   {
     lob_loc = ((OracleResultSet)rset).getBLOB (1);
   }

   // Open the LOB for READWRITE:
       OracleCallableStatement cstmt = (OracleCallableStatement)
          conn.prepareCall ("BEGIN DBMS_LOB.OPEN(?,
                            DBMS_LOB.LOB_READWRITE); END;");
   cstmt.setBLOB(1, lob_loc);
   cstmt.execute();

   // Erase eraseAmount bytes starting at offset 2000:
   cstmt = (OracleCallableStatement)
```

```
      conn.prepareCall ("BEGIN DBMS_LOB.ERASE(?, ?, 1); END;");
    cstmt.registerOutParameter (1, OracleTypes.BLOB);
    cstmt.registerOutParameter (2, Types.INTEGER);
    cstmt.setBLOB(1, lob_loc);
    cstmt.setInt(2, eraseAmount);
    cstmt.execute();
    lob_loc = cstmt.getBLOB(1);
    eraseAmount = cstmt.getInt(2);

    // Close the LOB:
    cstmt = (OracleCallableStatement) conn.prepareCall (
       "BEGIN DBMS_LOB.CLOSE(?); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

    conn.commit();
    stmt.close();
    cstmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

Enable LOB Buffering

Figure 3–38 Use Case Diagram: Enable LOB Buffering



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This scenario is part of the management of a buffering example related to Sound that is developed in this and related methods.

You enable buffering in order to perform a small read or write of the data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations. Note that you must flush the buffer in order to make your modifications persistent. For more information, refer to "LOB Buffering Subsystem" on page 2-14 in Chapter 2, "Advanced Topics".

Please note that you would not enable buffering to perform the stream read and write involved in checkin and checkout.

- "Example: Enable LOB Buffering Using COBOL (Pro*COBOL)" on page 3-236
- "Example: Enable LOB Buffering Using C++ (Pro*C/C++)" on page 3-238
- "Example: Enable LOB Buffering Using Visual Basic (OO4O)" on page 3-239

Example: Enable LOB Buffering Using C (OCI)

See:

- "Disable LOB Buffering" on page 3-246

Example: Enable LOB Buffering Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BLOB1          SQL-BLOB.
01  BUFFER         PIC X(10).
01  AMT            PIC S9(9) COMP.
```

```
    EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 LOB-BUFFERING.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locator:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :BLOB1
         FROM MULTIMEDIA_TAB
         WHERE CLIP_ID = 1 FOR UPDATE
    END-EXEC.

* Open the BLOB and enable buffering:
    EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
    EXEC SQL
         LOB ENABLE BUFFERING :BLOB1
    END-EXEC.

* Write some data to the BLOB:
    MOVE "242424" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

    MOVE "212121" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

* Now flush the buffered writes:
    EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
    EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.

    EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
```

```
END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.
    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED:".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

Example: Enable LOB Buffering Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 256

void enableBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;
```

```
  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
            FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  enableBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

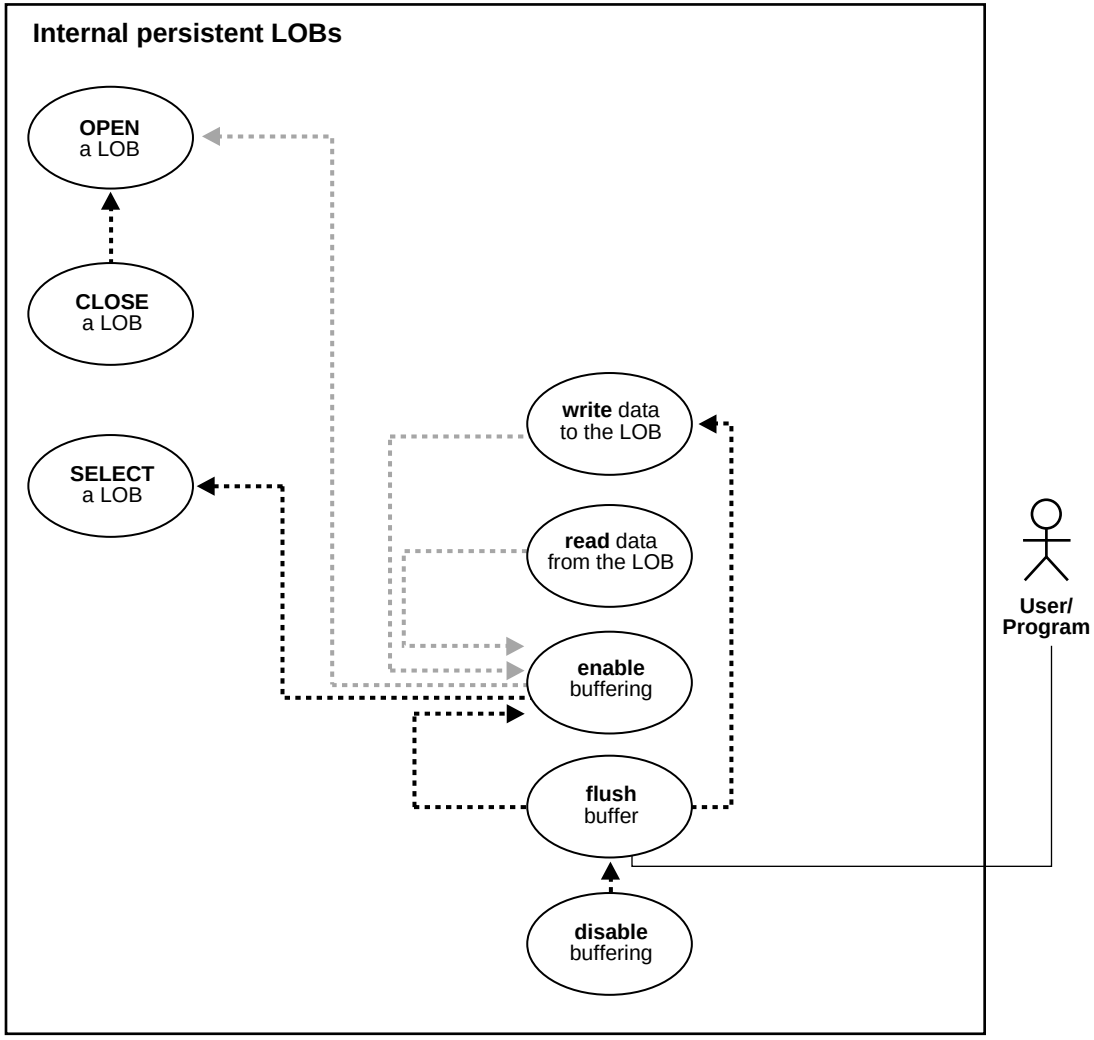
Example: Enable LOB Buffering Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
```

```
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)
Set OraSound1 = OraDyn.Fields("Sound").Value
'Enable buffering:
OraSound1.EnableBuffering
```

Flush Buffer

Figure 3–39 Use Case Diagram: Flush Buffer



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This scenario is part of the management of a buffering example related to Sound that is developed in this and related methods.

You enable buffering in order to perform a small read or write of the data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations. Note that you must flush the buffer in order to make your modifications persistent. For more information, refer to "LOB Buffering Subsystem" on page 2-14 in Chapter 2, "Advanced Topics".

Please note that you would not enable buffering to perform the stream read and write involved in checkin and checkout.

- "Example: Flush Buffer Using C (OCI)" on page 3-242
- "Example: Flush Buffer Using COBOL (Pro*COBOL)" on page 3-242
- "Example: Flush Buffer Using C++ (Pro*C/C++)" on page 3-244

Example: Flush Buffer Using C (OCI)

See:

- "Disable LOB Buffering" on page 3-246

Example: Flush Buffer Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BLOB1          SQL-BLOB.
01  BUFFER         PIC X(10).
01  AMT            PIC S9(9) COMP.
```

```
    EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 LOB-BUFFERING.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locator:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :BLOB1
         FROM MULTIMEDIA_TAB
         WHERE CLIP_ID = 1 FOR UPDATE
    END-EXEC.

* Open the BLOB and enable buffering:
    EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
    EXEC SQL
         LOB ENABLE BUFFERING :BLOB1
    END-EXEC.

* Write some data to the BLOB:
    MOVE "242424" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

    MOVE "212121" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

* Now flush the buffered writes:
    EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
    EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.

    EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
```

```
END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.
    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED:".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

Example: Flush Buffer Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 256

void flushBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;
```

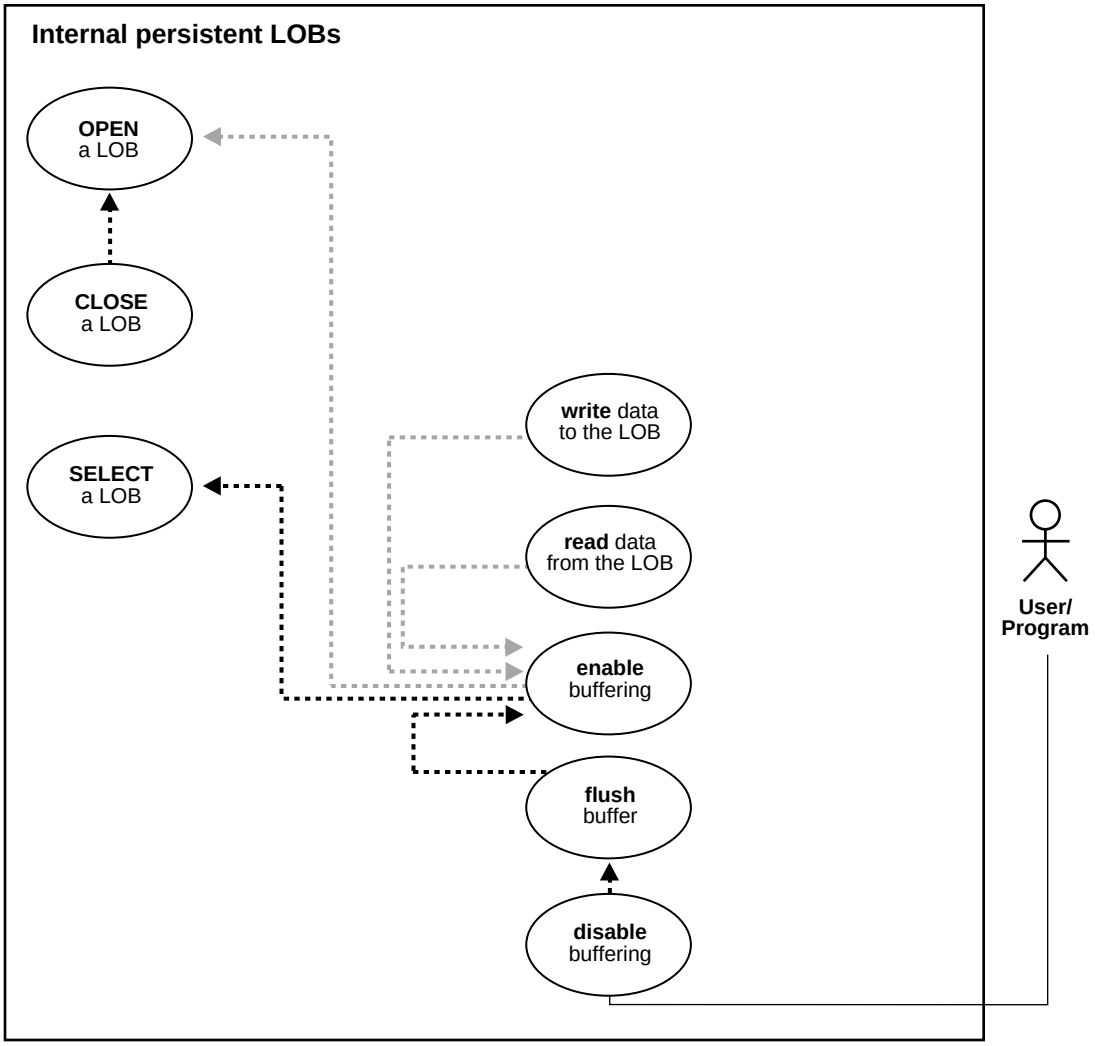
```
  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
            FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  flushBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Flush Buffer Using Visual Basic (OO4O)

Note: A Visual Basic (OO4O) example will be made available in a subsequent release.

Disable LOB Buffering

Figure 3–40 Use Case Diagram: Disable LOB Buffering



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This scenario is part of the management of a buffering example related to Sound that is developed in this and related methods.

You enable buffering in order to perform a small read or write of the data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations. Note that you must flush the buffer in order to make your modifications persistent.

Please note that you would not enable buffering to perform the stream read and write involved in checkin and checkout.

- "Example: Disable LOB Buffering Using C (OCI)" on page 3-247
- "Example: Disable LOB Buffering Using COBOL (Pro*COBOL)" on page 3-249
- "Example: Disable LOB Buffering Using C++ (Pro*C / C++)" on page 3-251
- "Example: Disable LOB Buffering Using Visual Basic (OO4O)" on page 3-252
- "Three Ways to Update a LOB" on page 3-254

Example: Disable LOB Buffering Using C (OCI)

```
/* Select the locator into a locator variable: */

sb4 select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=1 FOR UPDATE";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
```

```
  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                  (dvoid *)&Lob_loc, (sb4)0,
                                  (ub2) SQLT_BLOB,(dvoid *) 0,
                                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return 0;
}

void lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;


  /* Allocate the locator desriptor: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" select a sound Lob\n");
  select_lock_sound_locator(Lob_loc, errhp, svchp, stmthp);

  /* Open the BLOB: */
  checkerr (errhp, (OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READWRITE)));

  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, Lob_loc));

  printf (" write data to LOB\n");
```

```
  /* Write data into the LOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);
  offset = 1;

  checkerr (errhp, OCILobWrite (svchp, errhp, Lob_loc, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));

  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, Lob_loc,
                                     (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, Lob_loc));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem: */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

  return;

}
```

Example: Disable LOB Buffering Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BLOB1          SQL-BLOB.
01  BUFFER         PIC X(10).
01  AMT            PIC S9(9) COMP.
```

```
    EXEC SQL VAR BUFFER IS RAW(10) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.

 PROCEDURE DIVISION.
 LOB-BUFFERING.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locator:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :BLOB1
         FROM MULTIMEDIA_TAB
         WHERE CLIP_ID = 1 FOR UPDATE
    END-EXEC.

* Open the BLOB and enable buffering:
    EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.
    EXEC SQL
         LOB ENABLE BUFFERING :BLOB1
    END-EXEC.

* Write some data to the BLOB:
    MOVE "242424" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

    MOVE "212121" TO BUFFER.
    MOVE 3 TO AMT.
    EXEC SQL
         LOB WRITE :AMT FROM :BUFFER INTO :BLOB1
    END-EXEC.

* Now flush the buffered writes:
    EXEC SQL LOB FLUSH BUFFER :BLOB1 END-EXEC.
    EXEC SQL LOB DISABLE BUFFERING :BLOB1 END-EXEC.

    EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
```

```
END-OF-BLOB.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BLOB1 END-EXEC.
    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED:".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

Example: Disable LOB Buffering Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 256

void disableBufferingLOB_proc()
{
  OCIBlobLocator *Lob_loc;
  int Amount = BufferLength;
  int multiple, Position = 1;
```

```
  /* Datatype equivalencing is mandatory for this datatype: */
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer is RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Lob_loc;
  memset((void *)Buffer, 0, BufferLength);
  for (multiple = 0; multiple < 7; multiple++)
    {
      /* Write data to the LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                   FROM :Buffer INTO :Lob_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Lob_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Lob_loc;
  /* Write APPEND can only be done when Buffering is Disabled: */
  EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Lob_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  disableBufferingLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

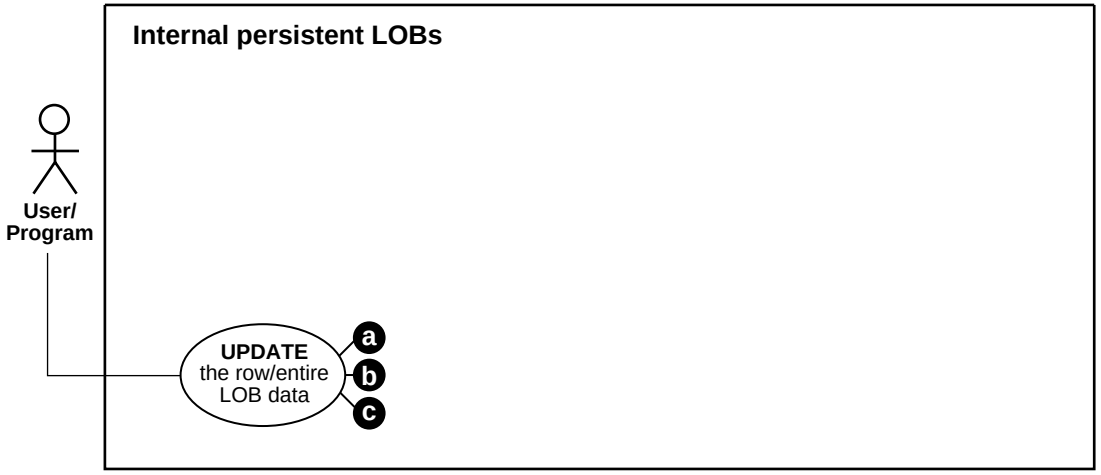
Example: Disable LOB Buffering Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraSound1 As OraBlob, OraSoundClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
```

```
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
   "SELECT * FROM Multimedia_tab ORDER BY clip_id", ORADYN_DEFAULT)

Set OraSound1 = OraDyn.Fields("Sound").Value
'Disable buffering:
OraSound1.DisableBuffering
```

Three Ways to Update a LOB

Figure 3–41 Use Case Diagram: Three Ways to Update a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

---

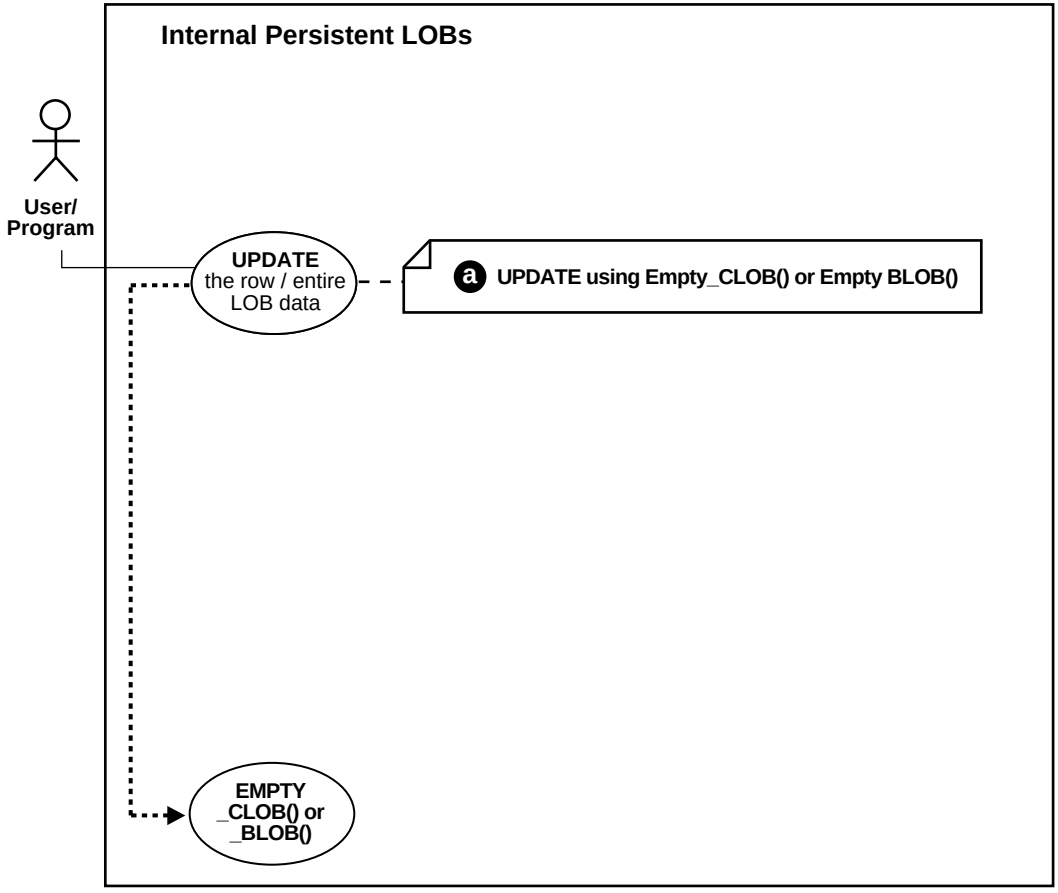
a. "UPDATE a LOB with EMPTY_CLOB() or EMPTY_BLOB()" on page 3-256

b. "UPDATE as SELECT" on page 3-257

c. "UPDATE by Initializing a LOB Locator Bind Variable" on page 3-259

UPDATE a LOB with EMPTY_CLOB() or EMPTY_BLOB()

Figure 3–42 Use Case Diagram: UPDATE using EMPTY_CLOB() or EMPTY_BLOB()



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

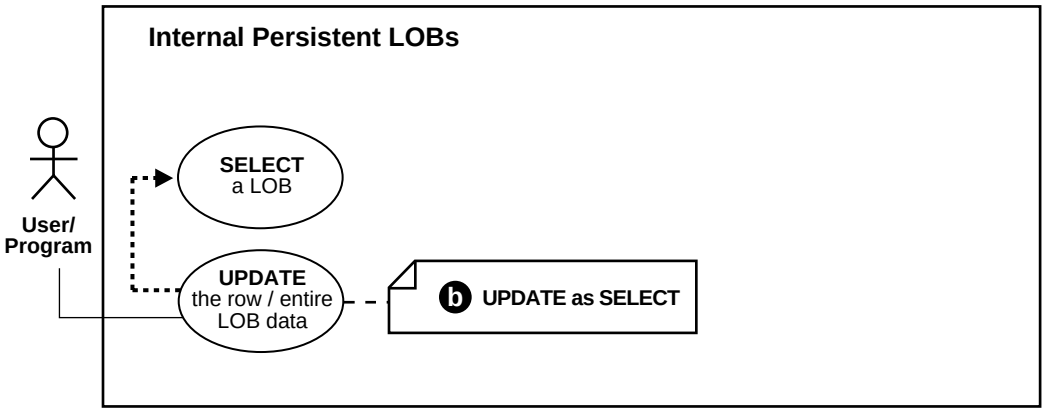
This example shows a series of updates via the EMPTY_CLOB operation to different data types of the first clip.

Example: UPDATE a LOB with EMPTY_CLOB() or EMPTY_BLOB() Using SQL

```
UPDATE Multimedia_tab SET Story = EMPTY_CLOB() WHERE Clip_ID = 1;
UPDATE Multimedia_tab SET FLSub = EMPTY_CLOB() WHERE Clip_ID = 1;
UPDATE multimedia_tab SET Sound = EMPTY_BLOB() WHERE Clip_ID = 1;
```

UPDATE as SELECT

Figure 3–43 Use Case Diagram: UPDATE as SELECT



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

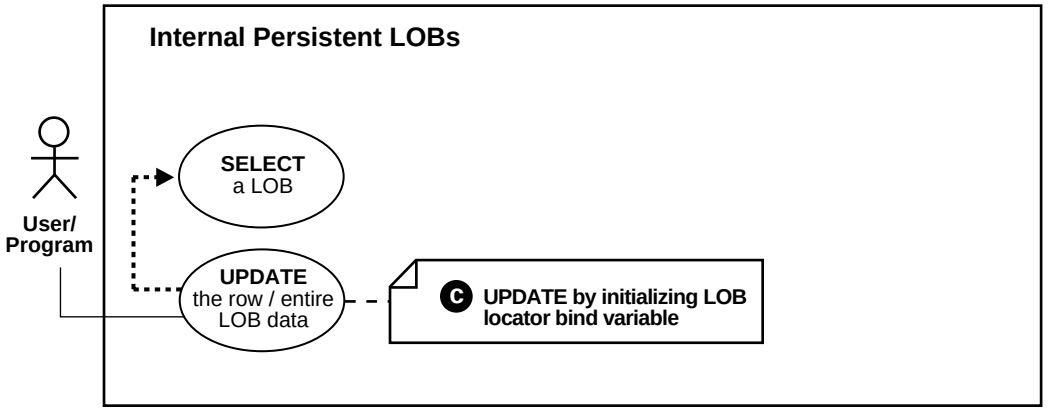
This example updates voice-over data from archival storage (VoiceoverLib_tab) by means of a reference.

Example: Update as Select Using SQL DML

```
UPDATE Voiceover_tab SET (Originator, Script, Actor, Take, Recording) =
   (SELECT * FROM VoiceoverLib_tab T2 WHERE T2.Take = 101);
UPDATE Multimedia_tab Mtab
   SET Voiced_ref =
      (SELECT REF(Vref) FROM Voiceover_tab Vref
         WHERE Vref.Actor = 'James Earl Jones' AND Vref.Take = 1)
            WHERE Mtab.Clip_ID = 1;
```

UPDATE by Initializing a LOB Locator Bind Variable

Figure 3–44 Use Case Diagram: UPDATE by Initializing a LOB Locator Bind Variable



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

This example updates Sound data by means of a locator bind variable.

- "Example: Update by Initializing a LOB Locator Bind Variable Using SQL DML" on page 3-259
- "Example: Update by Initializing a LOB Locator Bind Variable Using C (OCI)" on page 3-259
- "Example: Update by Initializing a LOB Locator Bind Variable Using COBOL (Pro*COBOL)" on page 3-261
- "Example: Update by Initializing a LOB Locator Bind Variable Using C++ (Pro*C/C++)" on page 3-262
- "Example: Update by Initializing a LOB Locator Bind Variable Using C++ (Pro*C/C++)" on page 3-262

- "Example: Update by Initializing a LOB Locator Bind Variable Using Java (JDBC)" on page 3-264

Example: Update by Initializing a LOB Locator Bind Variable Using SQL DML

```
/* Note that the example procedure updateUseBindVariable_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (Lob_loc BLOB) IS
BEGIN
   UPDATE Multimedia_tab SET Sound = lob_loc WHERE Clip_ID = 2;
END;

DECLARE
   Lob_loc     BLOB;
BEGIN
   /* Select the LOB: */
   SELECT Sound INTO Lob_loc
      FROM Multimedia_tab
         WHERE Clip_ID = 1;
   updateUseBindVariable_proc (Lob_loc);
   COMMIT;
END;
```

Example: Update by Initializing a LOB Locator Bind Variable Using C (OCI)

```
/* Select the locator into a locator variable: */

sb4 select_sound_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  text  *sqlstmt =
     (text *)"SELECT Sound FROM Multimedia_tab WHERE Clip_ID=2";
  OCIDefine *defnp1;

  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                  (ub4)strlen((char *)sqlstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));

  checkerr (errhp, OCIDefineByPos(stmthp, &defnp1, errhp, (ub4) 1,
                                   (dvoid *)&Lob_loc, (sb4)0,
```

```
                                    (ub2) SQLT_BLOB,(dvoid *) 0,
                                    (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the select and fetch one row: */
  checkerr(errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                 (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                 (ub4) OCI_DEFAULT));

  return 0;
}

/* Update the LOB  in the selected row in the table: */
void updateLobUsingBind (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  text  *updstmt =
    (text *) "UPDATE Multimedia_tab SET Sound = :1 WHERE Clip_ID = 1";
  OCILobLocator *Lob_loc;
  OCIBind       *bndhp1;

  /* Allocate locator resources: */
  (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **)&Lob_loc,
                            (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid **)0);

  /* Select the locator: */
  printf(" select a sound locator\n");
  (void)select_sound_locator(Lob_loc, errhp, svchp, stmthp);

  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, updstmt, (ub4)
                                  strlen((char *) updstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Binds the bind positions: */
  printf(" bind locator to bind position\n");

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp1, errhp, (ub4) 1,
                                (dvoid *) &Lob_loc, (sb4)0, SQLT_BLOB,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the SQL statement: */
```

```
  printf ("update LOB column in another row using this locator\n");
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

  return;
}
```

Example: Update by Initializing a LOB Locator Bind Variable Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDATE-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  BLOB1          SQL-BLOB.
01  NEW-LEN        PIC S9(9) COMP.
01  AMT            PIC S9(9) COMP.
01  OFFSET         PIC S9(9) COMP.

* Define the source and destination position and location:
01  SRC-POS        PIC S9(9) COMP.
01  DEST-POS       PIC S9(9) COMP.
01  SRC-LOC        PIC S9(9) COMP.
01  DEST-LOC       PIC S9(9) COMP.
01  USERID   PIC X(11) VALUES "USER1/USER1".
    EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
UPDATE-BLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC SQL
         SELECT SOUND INTO :BLOB1
```

```
            FROM MULTIMEDIA_TAB
            WHERE CLIP_ID = 1
        END-EXEC.

        EXEC SQL
            UPDATE MULTIMEDIA_TAB
            SET SOUND = :BLOB1 WHERE CLIP_ID = 2
        END-EXEC.

    END-OF-BLOB.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :BLOB1 END-EXEC.
        EXEC SQL
            COMMIT WORK RELEASE
        END-EXEC.
        STOP RUN.

    SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED:".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Update by Initializing a LOB Locator Bind Variable Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}
```

```
void updateUseBindVariable_proc(Lob_loc)
  OCIBlobLocator *Lob_loc;
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL UPDATE Multimedia_tab SET Sound = :Lob_loc WHERE Clip_ID = 2;
}

void updateLOB_proc()
{
  OCIBlobLocator *Lob_loc;

  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Sound INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  updateUseBindVariable_proc(Lob_loc);
  EXEC SQL FREE :Lob_loc;
  EXEC SQL COMMIT WORK;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  updateLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Update by Initializing a LOB Locator Bind Variable Using Visual Basic (OO4O)

```
Dim OraDatabase as OraDatabase,OraDyn as OraDynaset, OraSound as OraBLOB,

'Select a column with clip_id  = 1:
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Multimedia_tab WHERE
    clip_id = 1", ORADYN_DEFAULT)

'Get the OraBlob object from the field:
Set OraSound = OraDyn.Fields("Sound").Value

'Create a parameter for OraBlob object:
OraDatabase.Parameters.Add "SOUND",NULL,ORAPARM_INPUT,ORATYPE_BLOB

'Set the value of SOUND parameter to OraSound:
```

```
OraDatabase.Parameters("SOUND").Value = OraSound

'Update the Multimedia_tab with OraSound for clip_id = 2:
OraDatabase.ExecuteSQL("Update  Multimedia_tab SET Sound = :SOUND
   WHERE Clip_id = 2")
```

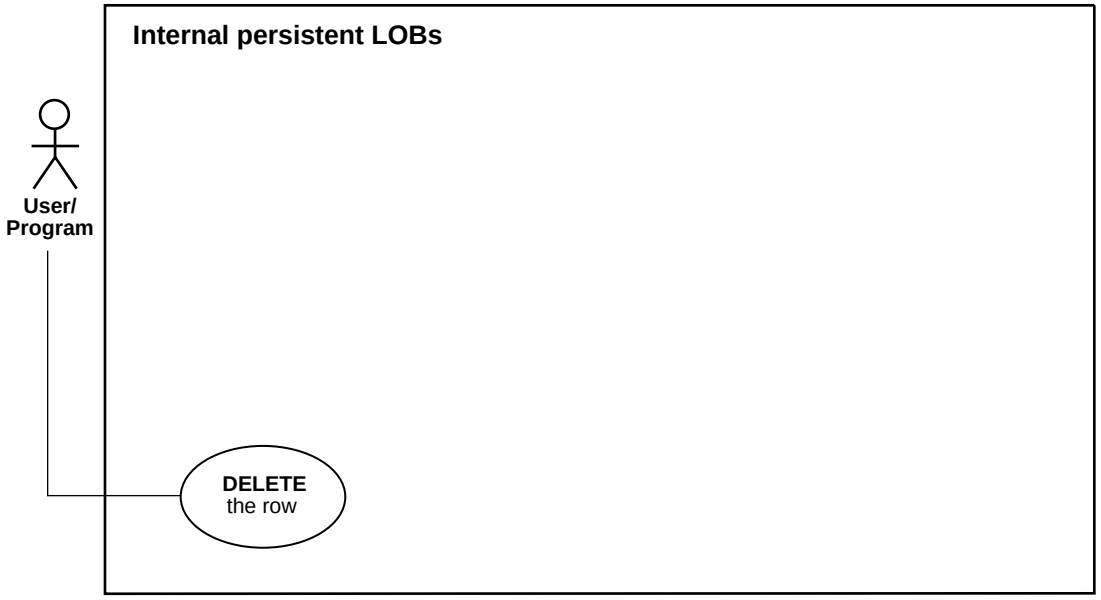
Example: Update by Initializing a LOB Locator Bind Variable Using Java (JDBC)

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_163
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
     ResultSet rset = stmt.executeQuery (
        "SELECT sound FROM multimedia_tab WHERE clip_id = 1");
      if (rset.next())
```

```
      {
       // retrieve the LOB locator from the ResultSet:
       BLOB sound_blob = ((OracleResultSet)rset).getBLOB (1);

        OraclePreparedStatement ops =
          (OraclePreparedStatement) conn.prepareStatement(
             "UPDATE multimedia_tab SET SOUND = ? WHERE clip_id = 2");

          ops.setBlob(1, sound_blob);
          ops.execute();
     conn.commit();
     conn.close();
   }
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

DELETE the Row of a Table Containing a LOB

Figure 3–45 Use Case Diagram: DELETE the row of a table containing a LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: Internal Persistent LOBs" on page 3-2

Scenario

You delete a row that contains an internal LOB column / attribute by using

- the explicit SQL DML command DELETE, or
- a SQL DDL command that effectively deletes it, such as DROP TABLE, TRUNCATE TABLE, or DROP TABLESPACE.

In either case you delete the LOB locator *and the LOB value as well.*

But note that due to the consistent read mechanism, the old LOB value remains accessible with the value that it had at the time of execution of the statement (such as SELECT) that returned the LOB locator.

> **Note:** This is an advanced topic that is discussed in more detail with regard to "Read-Consistent Locators" on page 2-2.

Of course, two distinct rows of a table with a LOB column have their own distinct LOB locators and distinct copies of the LOB values irrespective of whether the LOB values are the same or different. This means that deleting one row has no effect on the data or LOB locator in another row even if one LOB was originally copied from another row.

In this case we delete all the data associated with Clip 10.

Example: Delete a LOB Using SQL DML

```
DELETE FROM Multimedia_tab
   WHERE Clip_ID = 10;

DROP TABLE Multimedia_tab;

TRUNCATE TABLE Multimedia_tab;
```

4

Temporary LOBs

In this chapter we describe how to work with Temporary LOBs in terms of use cases. That is, we discuss each operation on a LOB (such as "See If a Temporary LOB is Open") in terms of a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: Internal Temporary LOBs" on page 4-2). A summary figure, "Use Case Model Diagram: Temporary LOBs", locates all the use cases in single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case in which you are interested by clicking on the relevant use case title.

The individual use cases are themselves laid out as follows:

- A figure that depicts the use case (see "Preface" for a description of how to interpret these diagrams).
- A scenario that portrays one implementation of the use case in terms of the hypothetical multimedia application described above (see "An Example Application" on page 1-39 in Chapter 1, "Introduction to Working With LOBs").
- Code examples in each of the programmatic environments which can be utilized to implement the use case (see "Programmatic Environments for Operating on LOBs" on page 1-9 in Chapter 1, "Introduction to Working With LOBs").

Use Case Model: Internal Temporary LOBs

Table 4–1 Use Case Model Overview: Internal Temporary LOBs

Use Case and Page
Create a Temporary LOB on page 4-11
See If a LOB is Temporary on page 4-18
Free a Temporary LOB on page 4-23
Load a Temporary LOB with Data from a BFILE on page 4-28
See If a Temporary LOB Is Open on page 4-36
Display the Temporary LOB Data on page 4-42
Read Data from a Temporary LOB on page 4-52
Read a Portion of the Temporary LOB (substr) on page 4-61
Compare All or Part of Two (Temporary) LOBs on page 4-67
See If a Pattern Exists in a Temporary LOB (instr) on page 4-74
Get the Length of a Temporary LOB on page 4-80
Copy All or Part of One (Temporary) LOB to Another on page 4-88
Copy a LOB Locator for a Temporary LOB on page 4-98
See If One LOB Locator for a Temporary LOB Is Equal to Another on page 4-107
See If a LOB Locator for a Temporary LOB Is Initialized on page 4-111
Get Character Set ID of a Temporary LOB on page 4-114
Get Character Set Form of a Temporary LOB on page 4-116
Append One (Temporary) LOB to Another on page 4-118
Write Append to a Temporary LOB on page 4-127
Write Data to a Temporary LOB on page 4-134
Trim the Temporary LOB Data on page 4-144
Erase Part of a Temporary LOB on page 4-152
Enable LOB Buffering for a Temporary LOB on page 4-160
Flush Buffer for a Temporary LOB on page 4-166
Disable LOB Buffering for a Temporary LOB on page 4-172

Figure 4–1 Use Case Model Diagram: Internal Temporary LOBs (part 1 of 2)

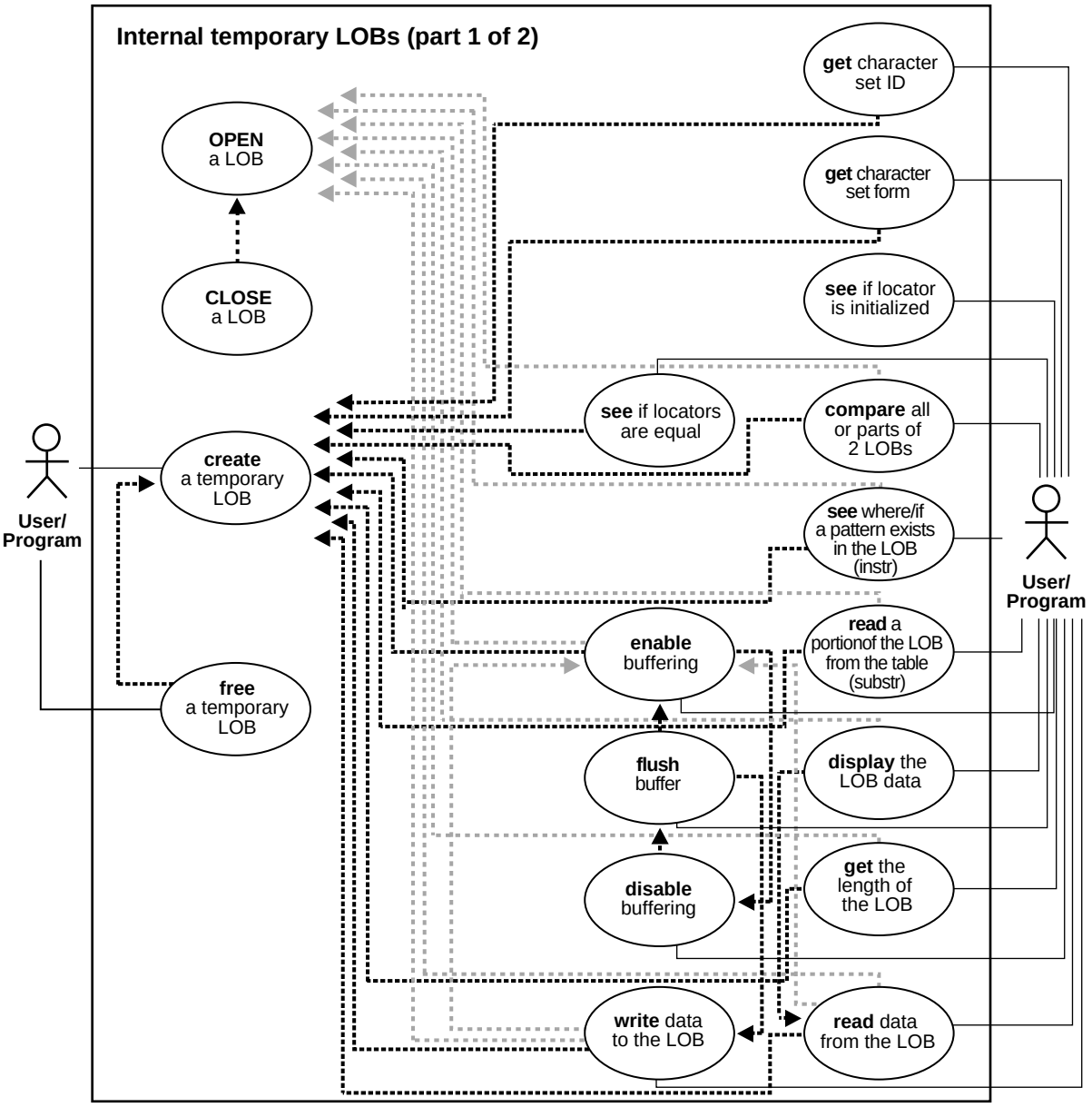
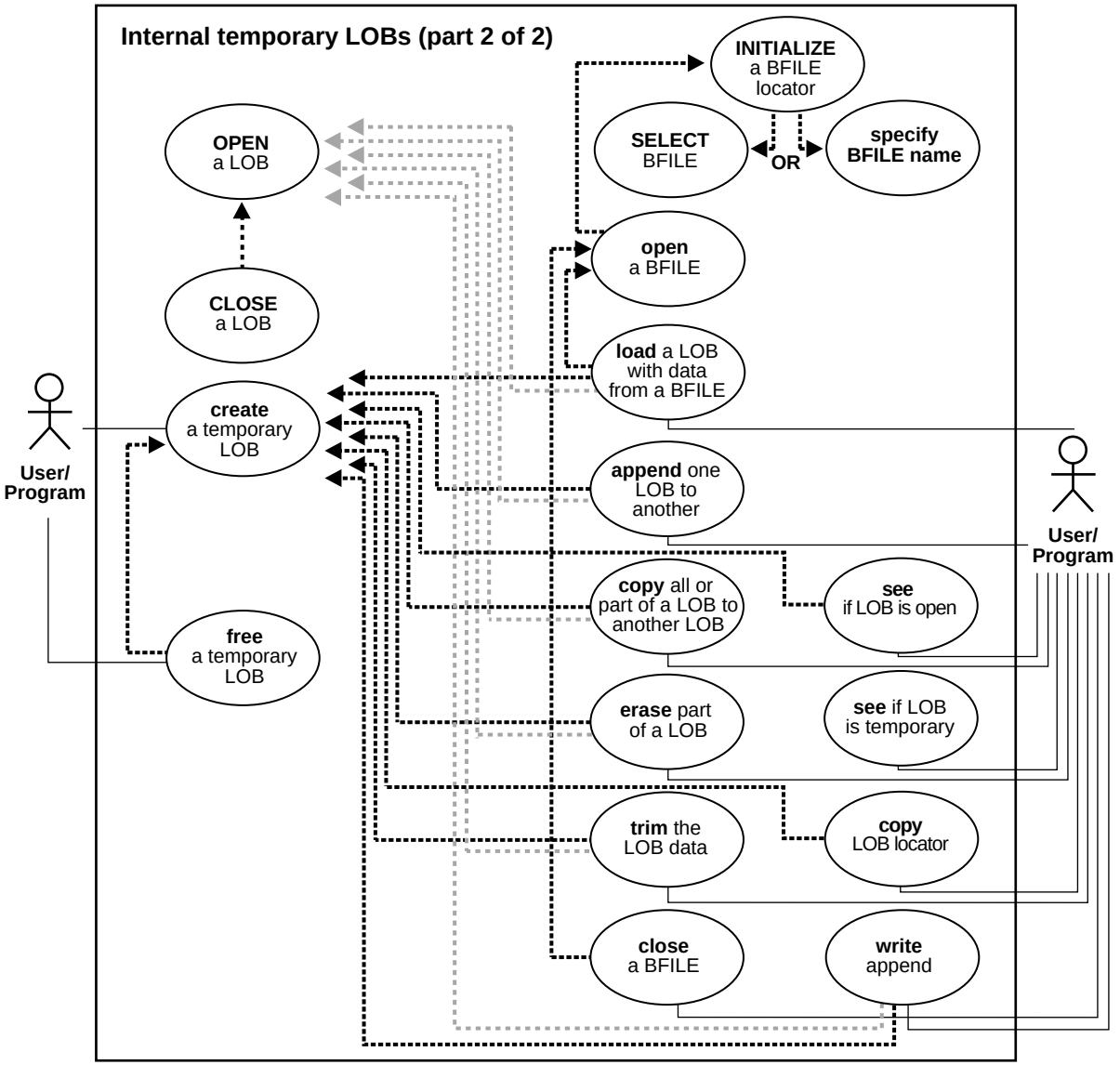


Figure 4–2 Use Case Model Diagram: Internal temporary LOBs (part 2 of 2)



Programmatic Environments

> **Note:** No Visual Basic or Java support for temporary LOBs is planned for the 8.1 time-frame.

Oracle8*i* supports the definition, creation, deletion, access, and update of temporary LOBs in PL/SQL (using the DBMS_LOB package), C/C++ (using PRO*C), and C (using the OCI).

These interfaces operate on temporary LOBs through locators in the same way that they do for permanent LOBs. Since temporary lobs are never part of any table, you cannot use SQL DML to operate on them. They must be manipulated using the DBMS_LOB package, the OCI, or the other programmatic interfaces.

SQL support for temporary LOBs is available in that temporary LOB locators can be used as IN values, with values accessed through a locator. Specifically, they can be used

- as a value in a WHERE clause for INSERT, UPDATE, DELETE, or SELECT such as

```
SELECT pattern FROM composite_image WHERE temp_lob_pattern_id  =
somepattern_match_function(lobvalue);
```

and

- as a variable in a SELECT INTO... statement such as

```
SELECT PermanentLob INTO TemporaryLob_loc FROM Demo_tab WHERE Column1 := 1;
```

Note that selecting a permanent LOB into a temporary LOB locator will cause the temporary LOB locator to point to a permanent LOB. It does not cause a copy of the permanent LOB to be put in the temporary LOB.

Examining the use case model diagrams for temporary LOBs, and comparing it to the "Use Case Model Diagram: Internal Persistent LOBs (part 1 of 2)", and "Use Case Model Diagram: Internal Persistent LOBs (part 2 of 2)", you can see that you can utilize many of the same functions that apply to persistent LOBs for operating on temporary LOBs:

- DBMS_LOB package PL/SQL procedures (Compare, Instr, Substr)
- DBMS_LOB package PL/SQL procedures and corresponding OCI functions (Append, Copy, Erase, Getlength, Loadfromfile, Read, Trim, Write, WriteAppend).

- OCI functions (OCILobAssign, OCILobLocatorIsInit, etc.).

In addition, you can use the ISTEMPORARY function to determine if a LOB is temporary based on its locator.

The Location of Temporary LOBs

Temporary LOBs are not stored permanently in the database like other data. The data is stored in temporary tablespaces, but is not stored in any tables. This means you can CREATE an internal temporary LOB (BLOB,CLOB, NCLOB) on the server independent of any table, but you cannot store that LOB. Since temporary LOBs are not associated with a table schema, there are no meanings to the terms "inline" and "out-of-line" for temporary LOBs. However, note that all temporary LOBs reside in the server; there is no support for client-side temporary LOBs.

The Lifetime and Duration of Temporary LOBs

The default lifetime of a temporary LOB is a session.

The interface for creating temporary LOBs includes a parameter that lets you specify the default scope of the life of the temporary LOB. By default, all temporary LOBs are deleted at the end of the session in which they were created. If a process dies unexpectedly or the database crashes, all temporary LOBs are deleted.

OCI users can group temporary LOBs together into a logical bucket. The OCIDuration will represent a store for temporary LOBs. There will be a default duration for every session into which temporary LOBs will be placed if the user doesn't specify a specific duration. The default duration will end when the user's session ends. Also, the user will be able to perform an OCIDuration operation which will cause all contents in the OCIDuration to be freed.

Memory Handling

Temporary LOBs are especially useful when you want to perform some transformational operation on a LOB — such as morphing an image, or changing a LOB from one format to another — and then return it to the database. In doing this you can utilize LOB Buffering support for temporary LOBs, you can specify CACHE / NOCACHE for each temporary LOB, and you can FREE an individual temporary internal LOB when you have no further need of it.

Your temporary tablespace is used to store the temporary LOB data. Data storage resources will be controlled by the DBA through control of a user's access to temporary tablespaces, and by the creation of different temporary tablespaces.

Memory usage will increase incrementally as the number of temporary LOBs grows. You can reuse temporary LOB space in your session by freeing temporary LOBs explicitly. Freeing one or more temporary LOBs does not result in all of the space being returned to the temporary tablespace for general re-consumption. Instead, it remains available for reuse in the session. If a process dies unexpectedly or the database crashes, the space for temporary LOBs is freed along with the deletion of the temporary LOBs. In all cases, when a user's session ends, space is returned to the temporary tablespace for general reuse.

We previously noted that if you perform a

```
SELECT permanent_lob INTO temporary_lob_locator FROM y_blah WHERE x_blah
```

the temporary_lob_locator will get overwritten with the permanent_lob's locator. This will result in creating a copy of the LOB pointed at by permanent_lob, and temporary_lob_locator will represent this newly created temporary LOB. Note that unless you had saved the temporary_lob's locator in another variable, you will lose track of the LOB that temporary_lob_locator originally pointed at before the SELECT INTO operation.

In this case the temporary LOB will not get implicitly freed. If you do not wish to waste space, you will explicitly free a temporary LOB before overwriting it with a permanent LOB locator.

Since CR and rollbacks will not be supported for temporary LOBs, you will have to free the temporary LOB and start over again if you run into an error.

Locators and Semantics

Creation of a temporary LOB instance by a user causes the engine to create, and return a locator to the LOB data. Temporary LOBs do not support any operations that are not supported for persistent LOB locators, but temporary LOB locators have certain specific features. For instance, when you perform the following query

```
SELECT permanent_lob INTO temporary_lob_locator FROM y_blah
   WHERE x_blah := a_number;
```

temporary_lob_locator is overwritten with the permanent_lob's locator. This means that unless you have a copy of temporary_lob's locator that points to the temporary LOB that was overwritten, you no longer have a locator with which to access the temporary LOB.

Temporary LOBs adhere to value semantics in order to be consistent with permanent LOBs and to conform to the ANSI standard for LOBs. Since CR, undo,

and versions are not generated for temporary LOBs, there may be an impact on performance if you assign multiple locators to the same temporary LOB because semantically each locator will have its own copy of the temporary LOB. Each time a user does an OCILobAssign, or the equivalent assignment in PL/SQL, the database will make a copy of the temporary LOB (although it may be done lazily for performance reasons). Each locator will point to its own LOB value. If one locator is used to create a temporary LOB, and another LOB locator is assigned to that temporary LOB using OCILobAssign, the database will copy the original temporary LOB and cause the second locator to point to the copy, not the original temporary LOB.

In order for multiple users to modify the same LOB, they must go through the same locator. Although temporary LOBs use value semantics, you can apply pseudo-reference semantics by using pointers to locators in OCI, and having multiple pointers to locators point to the same temporary LOB locator if necessary. In PL/SQL, you can have the same effect by passing the temporary LOB locator "by reference" between modules. This will help avoid using more than one locator per temporary LOB, and prevent these modules from making local copies of the temporary LOB.

Here are two examples of situations where a user will incur a copy, or at least an extra roundtrip to the server:

- **Assigning one temporary LOB to another.**

```
DECLARE
  Va BLOB;
  Vb BLOB;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Vb,TRUE,DBMS_LOB.SESSION);
  DBMS_LOB.CREATETEMPORARY(Va,TRUE,DBMS_LOB.SESSION);
  Va := Vb;
END;
```

  This will cause Oracle to create a copy of Vb and point the locator Va to it. We will also free the temporary LOB that Va used to point to.

- **Assigning one collection to another collection.**

  If a temporary LOB is an element in a collection and you assign one collection to another, you will incur copy overhead and free overhead for the temporary LOB locators that get updated. This is also true for the case where you assign an object type containing a temporary LOB as an attribute to another such object type, and they have temporary LOB locators that get assigned to each other

because the object types have LOB attributes that are pointing to temporary LOB locators.

For information about with collections, see:

- *Oracle8i Concepts*
- *Oracle8i Application Developer's Guide - Fundamentals*

If your application involves several such assignments and copy operations of collections or complex objects, and you seek to avoid the above overheads, then persistent internal LOBs may be more suitable for such applications. More precisely: you should not use temporary LOBs inside collections or complex objects when you are doing assignments or copies of those collections or complex objects. Also, you should not select LOB values into temporary LOB locators.

You will incur overhead if you have a temporary LOB in a duration, you call OCIDurationEnd on that duration, and then subsequently reassign the locator for that temporary LOB to another LOB. Irrespective of whether there was a previous OCIDurationEnd call, Oracle will attempt to free the temporary LOB to which the locator pointed. Or if the user tries to access the temporary LOB with that locator they will incur an error. Once a user issues OCIDurationEnd, all temporary LOBs in that duration will be freed regardless of the fact that locators may still exist which used to refer to the now freed LOBs.

In PL/SQL, user-defined durations are not exposed. However, users may specify either session scope or call scopes using the predefined duration parameters dbms_lob.session, or dbms_lob.call.

User-defined OCIDurations can be created using the OCIDurationBegin call when the database is using the object option. The user can end the OCIDuration with a call to OCIDurationEnd. Any temporary LOBs that existed in the duration will be freed.

Security Issues with Temporary LOBs

Security is provided through the LOB locator. Only the user who created the temporary LOB can access it. Locators are not designed to be passed from one user's session to another. If you did manage to pass a locator from one session to another, you would no longer be able to access the temporary LOBs in the new session from the original session. By the same token, you would not be able to access a temporary LOB in the original session from the new (current) session to which the locator was migrated.

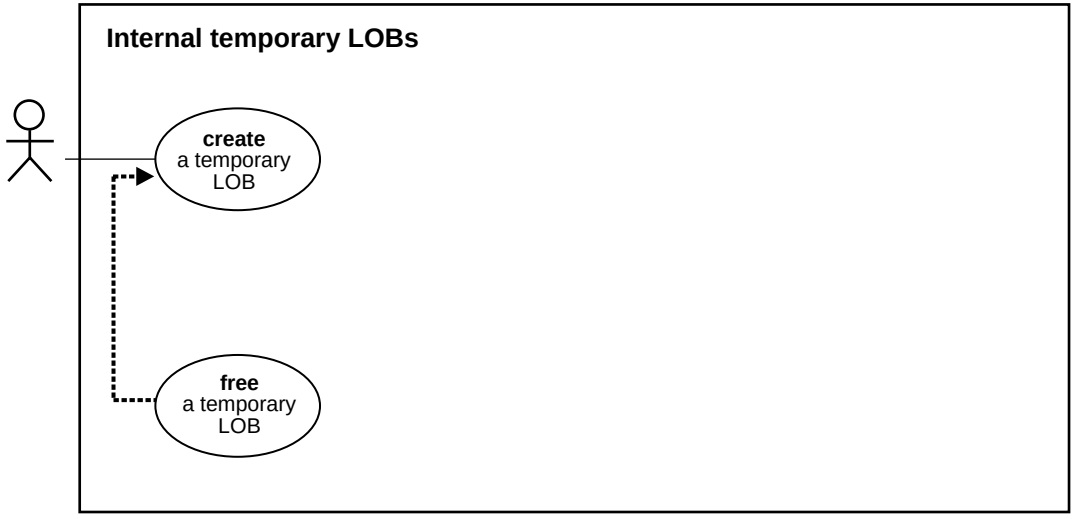
Temporary LOB lookup will be localized to each user's own session. Someone using a locator from another session would only be able to access LOBs within his own session that had the same lobid. Users of your application should not try to do this, but if they do, they will still not be able to affect anyone else's data.

Managing Temporary LOBs

Oracle keeps track of temporary LOBs per session, and provides a v$ view called v$temporary_lobs. From the session the application can determine which user owns the temporary LOBs. This table can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

Create a Temporary LOB

Figure 4–3 Use Case Diagram: Create a temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

A temporary LOB will be empty when it is created.

Temporary LOBs do not support the `empty_blob()` or `empty_clob()` functions that are supported for permanent LOBs. The `empty_blob()` function specifies the fact that the LOB is initialized, but not populated with any data.

This example reads in a single video `Frame` from the `Multimedia_tab` table. Then it creates a temporary LOB so that we can use the temporary LOB to convert the video image from MPEG to JPEG format. The Temporary LOB which is created will be read through the CACHE, and it will be automatically cleaned up at the end of the user's session, if it is not explicitly freed sooner.

Example: Create a Temporary LOB Using PL/SQL (DBMS_LOB Package)

Note: You may need to set up the following data structures for certain examples to work:

```
CREATE TABLE long_raw_tab (id number, long_raw_col long raw);
INSERT INTO long_raw_tab VALUES (1,HEXTORAW('7D'));
INSERT INTO multimedia_tab (clip_id,frame) SELECT
   id,TO_LOB(long_raw_col) FROM long_raw_tab;
```

```
DECLARE
   Dest_loc       BLOB;
   Src_loc        BLOB;
   Amount         INTEGER := 4000;
BEGIN
   SELECT Frame INTO Src_loc FROM Multimedia_tab WHERE Clip_ID = 1;
    /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
   /* Copy the entire frame from the Src_loc to the Temporary Lob: */
   DBMS_LOB.COPY(Dest_loc,Src_loc,DBMS_LOB.GETLENGTH(Src_loc),1,1);
   DBMS_LOB.FREETEMPORARY(Dest_loc);
END;
```

Example: Create a Temporary LOB Using C (OCI)

```
/* This function reads in a single video Frame from the Multimedia_tab table.
  Then it creates a temporary LOB so that we can use the temporary LOB to
  convert the video image from MPEG to JPEG format.. The Temporary LOB which is
  created will be read through the CACHE, and it will be automatically cleaned
  up at the end of the user's session, if it is not explicitly freed sooner.
  This function returns 0 if it completes successfully, and -1 if it fails: */
sb4 select_and_createtemp (OCILobLocator *lob_loc,
                           OCIError      *errhp,
                           OCISvcCtx     *svchp,
```

```
                          OCIStmt       *stmthp,
                          OCIEnv        *envhp)
{
  OCIDefine     *defnp1;
  OCIBind       *bndhp;
  text          *sqlstmt;
  int rowind =1;
  ub4 loblen = 0;
  OCILobLocator *tblob;

  printf ("in select_and_createtemp \n");

  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
  {
    printf("failed in OCIDescriptor Alloc in select_and_createtemp \n");
    return -1;
  }

  /* Arbitrarily select where Clip_ID =1: */
  sqlstmt = (text *)
     "SELECT Frame FROM Multimedia_tab WHERE Clip_ID = 1 FOR UPDATE";

  if (OCIStmtPrepare(stmthp, errhp, sqlstmt,
                     (ub4) strlen((char *)sqlstmt),
                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT))
  {
      (void) printf("FAILED: OCIStmtPrepare() sqlstmt\n");
      return -1;
  }

  /* Define for BLOB: */
  if (OCIDefineByPos(stmthp,
             &defnp1, errhp, (ub4) 1, (dvoid *) &lob_loc, (sb4)0,
             (ub2) SQLT_BLOB, (dvoid *) 0, (ub2 *) 0,
             (ub2 *) 0, (ub4) OCI_DEFAULT))
  {
    (void) printf("FAILED: Select locator: OCIDefineByPos()\n");
    return -1;
  }

  /* Execute the select and fetch one row: */
  if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                     (ub4) OCI_DEFAULT))
```

```
  {
    (void) printf("FAILED: OCIStmtExecute() sqlstmt\n");
    return -1;
  }

  if(OCILobCreateTemporary(svchp,
                           errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  if (OCILobGetLength(svchp, errhp, lob_loc, &loblen) != 0)
  {
    printf("OCILobGetLength FAILED\n");
    return -1;
  }
  if (OCILobCopy(svchp, errhp, tblob,lob_loc,(ub4)loblen, (ub4) 1, (ub4) 1))
  {
    printf( "OCILobCopy FAILED \n");
  }
  if(OCILobFreeTemporary(svchp,errhp,tblob))
  {
    printf ("FAILED: OCILobFreeTemporary call \n");
    return -1;
  }

  return 0;
}
```

Example: Create a Temporary LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. CREATE-TEMPORARY.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID   PIC X(11) VALUES "USER1/USER1".

     01  BLOB1          SQL-BLOB.
     01  TEMP-BLOB      SQL-BLOB.
```

```
01  LEN            PIC S9(9) COMP.
01  D-LEN          PIC 9(9).
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
CREATE-TEMPORARY.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locators:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.

    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
    EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
    EXEC SQL
         SELECT FRAME INTO :BLOB1
         FROM MULTIMEDIA_TAB
         WHERE CLIP_ID = 1
    END-EXEC.

* Get the length of the persistent BLOB:
    EXEC SQL
         LOB DESCRIBE :BLOB1
         GET LENGTH INTO :LEN
    END-EXEC.

* Copy the entire length from persistent to temporary:
    EXEC SQL
         LOB COPY :LEN FROM :BLOB1 TO :TEMP-BLOB
    END-EXEC.

* Free the temporary LOB:
    EXEC SQL
```

```
            LOB FREE TEMPORARY :TEMP-BLOB
        END-EXEC.

    END-OF-BLOB.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :BLOB1 END-EXEC.
        EXEC SQL FREE :TEMP-BLOB END-EXEC.
        STOP RUN.

    SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

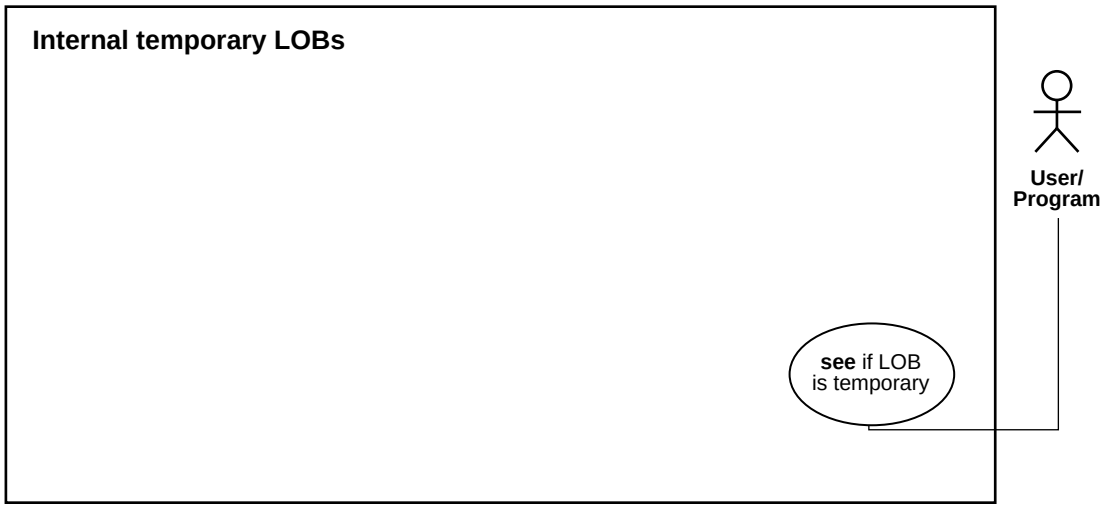
Example: Create a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void createTempLOB_proc()
{
  OCIBlobLocator *Lob_loc, *Temp_loc;
  int Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB Locators: */
```

```
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :Temp_loc;
  /* Create the Temporary LOB: */
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  EXEC SQL SELECT Frame INTO :Lob_loc FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Copy the full length of the source LOB into the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
  EXEC SQL LOB COPY :Amount FROM :Lob_loc TO :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  createTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

See If a LOB is Temporary

Figure 4–4 Use Case Diagram: See If a LOB is Temporary



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This is a generic example that queries whether the locator is associated with a temporary LOB or not.

- "Example: See If a LOB is Temporary Using PL/SQL (DBMS_LOB Package)" on page 4-19
- "Example: See If a LOB is Temporary Using C (OCI)" on page 4-19
- "Example: See If a LOB is Temporary Using COBOL (Pro*COBOL)" on page 4-20
- "Example: See If a LOB is Temporary Using C++ (Pro*C/C++)" on page 4-21

Example: See If a LOB is Temporary Using PL/SQL (DBMS_LOB Package)

```
/* This is also an example of freeing a temporary LOB. First we test to make
   sure that the LOB locator points to a temporary LOB, then we free it.
   Otherwise, we issue an error: */
CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS
BEGIN
   /* Free the temporary LOB locator passed in. */
   /* First check to make sure that the locator is pointing to a temporary
      LOB:*/
    IF DBMS_LOB.ISTEMPORARY(Lob_loc) = 1 THEN
       /* Free the temporary LOB locator: */
        DBMS_LOB.FREETEMPORARY(Lob_loc);
        DBMS_OUTPUT.PUT_LINE(' temporary LOB was freed');
    ELSE
        /* Print an error: */
        DBMS_OUTPUT.PUT_LINE(
           'Locator passed in was not a temporary LOB locator');
    END IF;
END;
```

Example: See If a LOB is Temporary Using C (OCI)

```
/* This function also frees a temporary LOB. It takes a locator as an argument,
   checks to see if it is a temporary LOB, and if it is the function will free
   the temporary LOB. Otherwise, it will print out a message saying the locator
   wasn't a temporary LOB locator. This function returns 0 if it
   completes successfully, and -1 otherwise: */

sb4 check_and_free_temp(OCILobLocator *tblob,
                        OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCIStmt       *stmthp,
                        OCIEnv        *envhp)
{
  boolean is_temp;
  is_temp = FALSE;

  if (OCILobIsTemporary(envhp, errhp, tblob, &is_temp))
  {
    printf ("FAILED: OCILobIsTemporary call\n");
    return -1;
  }
  if(is_temp)
  {
```

```
        if(OCILobFreeTemporary(svchp, errhp, tblob))
        {
            printf ("FAILED: OCILobFreeTemporary call\n");
            return -1;

        }else
        {
            printf("Temporary LOB freed\n");
        }
    }else
    {
        printf("locator is not a temporary LOB locator\n");
    }
    return 0;
}
```

Example: See If a LOB is Temporary Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-ISTEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB      SQL-BLOB.
01  IS-TEMP        PIC S9(9) COMP.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
CREATE-TEMPORARY.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.

    EXEC SQL
```

```
            LOB CREATE TEMPORARY :TEMP-BLOB
        END-EXEC.

    * Check if the LOB is temporary:
        EXEC SQL
            LOB DESCRIBE :TEMP-BLOB
            GET ISTEMPORARY INTO :IS-TEMP
        END-EXEC.

        IF IS-TEMP = 1
    *     Logic for a temporary LOB goes here
          DISPLAY "LOB is temporary."
        ELSE
    *     Logic for a persistent LOB goes here.
          DISPLAY "LOB is persistent."
        END-IF.

        EXEC SQL
            LOB FREE TEMPORARY :TEMP-BLOB
        END-EXEC.
        EXEC SQL FREE :TEMP-BLOB END-EXEC.
        STOP RUN.

     SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

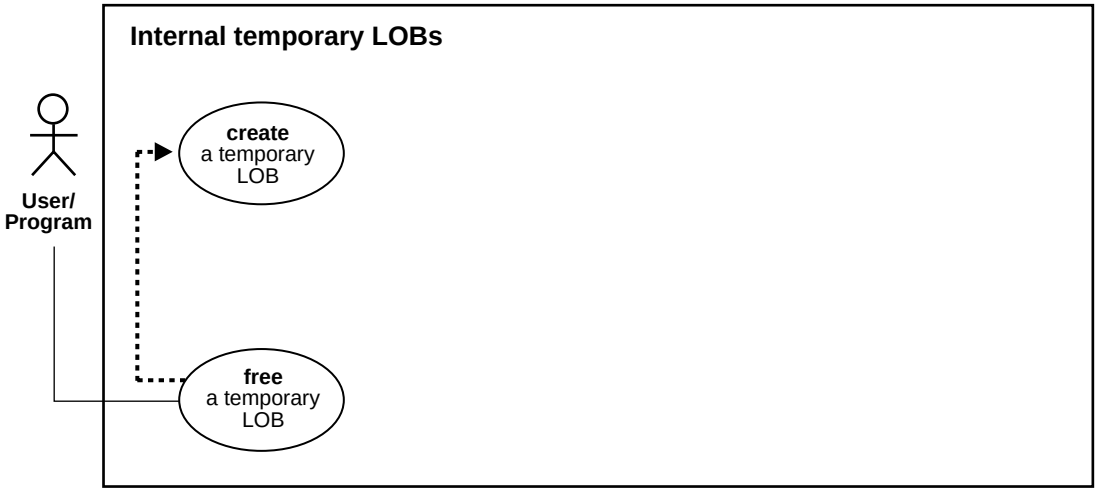
Example: See If a LOB is Temporary Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
```

```
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void lobIsTemp_proc()
{
  OCIBlobLocator *Temp_loc;
  int isTemporary = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Determine if the Locator is a Temporary LOB Locator: */
  EXEC SQL LOB DESCRIBE :Temp_loc GET ISTEMPORARY INTO :isTemporary;
  if (isTemporary)
    printf("Locator is a Temporary LOB locator\n");
  else
    printf("Locator is not a Temporary LOB locator \n");
  /* Note that in this example, isTemporary should be 1 (TRUE) */
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  lobIsTemp_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Free a Temporary LOB

Figure 4–5 Use Case Diagram: Free a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

A temporary `LOB` instance can only be destroyed by using OCI or the `DBMS_LOB` package by using the appropriate `FREETEMPORARY` or `OCIDurationEnd` or `OCILOBFreeTemporary` statements.

To make a temporary `LOB` permanent, the user must explicitly use the `OCI` or `DBMS_LOB` `copy`() command and copy the temporary `LOB` into a permanent one.

- "Example: Free a Temporary LOB Using PL/SQL (DBMS_LOB Package)" on page 4-24
- "Example: Free a Temporary LOB Using C (OCI)" on page 4-24
- "Example: Free a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-25

- "Example: Free a Temporary LOB Using C++ (Pro*C/C++)" on page 4-26

Example: Free a Temporary LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure freeTempLob_proc is not part of the
   DBMS_LOB package: */
CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS

BEGIN
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* Use the temporary LOB locator here, then free it.*/
   /* Free the temporary LOB locator: */
   DBMS_LOB.FREETEMPORARY(Lob_loc);
   DBMS_OUTPUT.PUT_LINE('Temporary LOB was freed');
END;
```

Example: Free a Temporary LOB Using C (OCI)

```
/* This function creates a temporary LOB and then frees it:
   This function returns 0 if it completes successfully, and -1 otherwise:  */

sb4 freeTempLob(OCIError      *errhp,
                OCISvcCtx     *svchp,
                OCIStmt       *stmthp,
                OCIEnv        *envhp)
{
  OCILobLocator *tblob;

  checkerr (errhp,OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                                     (ub4)OCI_DTYPE_LOB, (size_t)0,
                                     (dvoid**)0));

  if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0,SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
      (void) printf("FAILED:CreateTemporary():check_and_free_temp2\n");
      return -1;
  }

  if(OCILobFreeTemporary(svchp,errhp,tblob))
  {
      printf ("FAILED: OCILobFreeTemporary call in check_and_free_temp2\n");
      return -1;
```

```
  }else
  {
   printf("Temporary LOB freed in check_and_free_temp2\n");
  }
  return 0;
}
```

Example: Free a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".

01  TEMP-BLOB      SQL-BLOB.
01  IS-TEMP        PIC S9(9) COMP.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
FREE-TEMPORARY.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.

    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

* Do something with the temporary LOB here:

* Free the temporary LOB:
    EXEC SQL
```

```
            LOB FREE TEMPORARY :TEMP-BLOB
        END-EXEC.
        EXEC SQL FREE :TEMP-BLOB END-EXEC.
        STOP RUN.

    SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

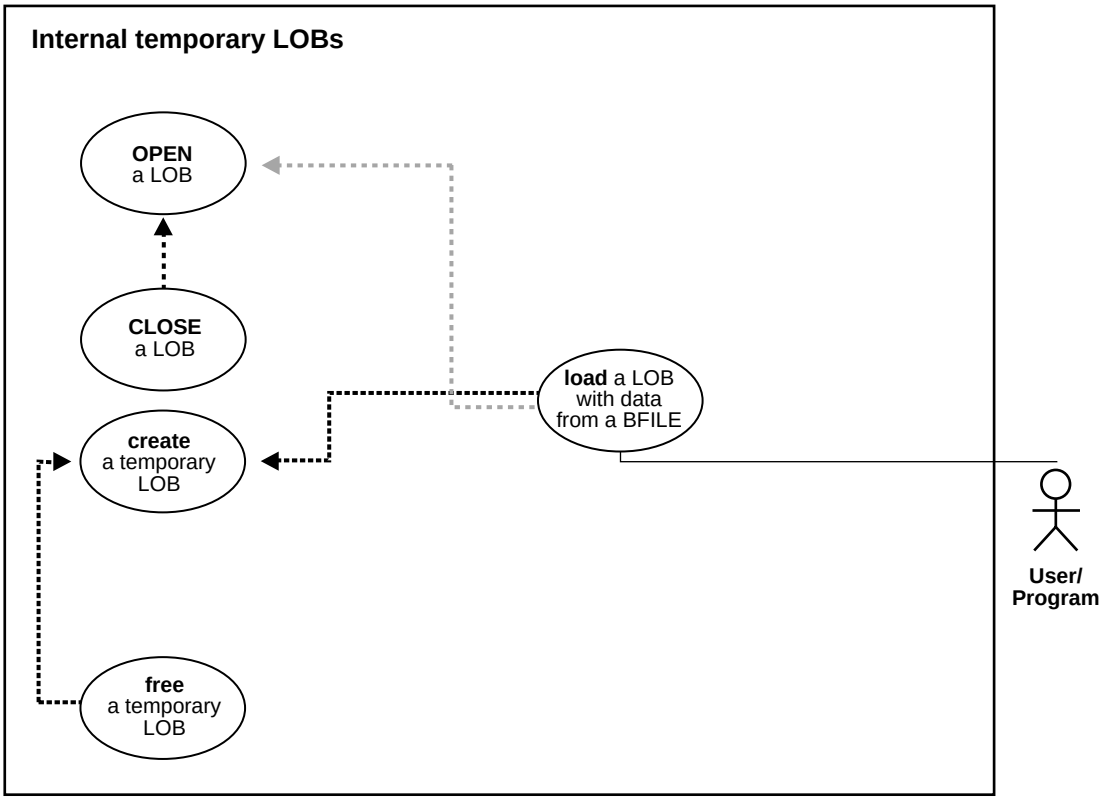
Example: Free a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void freeTempLob_proc()
{
  OCIBlobLocator *Temp_loc;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Do something with the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  EXEC SQL FREE :Temp_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  freeTempLob_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Load a Temporary LOB with Data from a BFILE

Figure 4–6 Use Case Diagram: Load a LOB with data from a BFILE



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

In using the OCI, or any of the programmatic environments that access OCI functionality, character set conversions are implicitly performed when translating

from one character set to another. However, no implicit translation is ever performed from binary data to a character set. When you use the `loadfromfile` operation to populate a `CLOB` or `NCLOB`, you are populating the `LOB` with binary data from the `BFILE`. In that case, you will need to perform character set conversions on the `BFILE` data before executing `loadfromfile`.

The example procedure assumes that there is an operating system source directory (`AUDIO_DIR`) that contains the `LOB` data to be loaded into the target `LOB`.

- "Example: Load a Temporary LOB with Data from a BFILE Using PL/SQL (DBMS_LOB Package)" on page 4-29
- "Example: Load a Temporary LOB with Data from a BFILE Using C (OCI)" on page 4-30
- "Example: Load a Temporary LOB with Data from a BFILE Using COBOL (Pro*COBOL)" on page 4-32
- "Example: Load a Temporary LOB with Data from a BFILE Using C++ (Pro*C/C++)" on page 4-33

Example: Load a Temporary LOB with Data from a BFILE Using PL/SQL (DBMS_LOB Package)

```
DECLARE
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount         INTEGER := 4000;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
   /* Opening the BFILE is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   /* 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;
```

Example: Load a Temporary LOB with Data from a BFILE Using C (OCI)

```
/* Here is a section of code which shows how to create a temporary LOB, and load
   the contents of a BFILE into the temporary LOB: */

sb4 load_temp(OCIError *errhp,
              OCISvcCtx *svchp,
              OCIStmt   *stmthp,
              OCIEnv    *envhp)
{
  OCILobLocator *bfile;
  int amount =100;
  OCILobLocator *tblob;

  printf("in load_temp\n");
  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
  {
    printf("OCIDescriptorAlloc failed in load_temp\n");
    return -1;
  }
  if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                        (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
  {
    printf("OCIDescriptorAlloc failed in load_temp\n");
    return -1;
  }

  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"), (text *)"Washington_audio",
                       (ub2)strlen("Washington_audio")))
  {
    printf("OCILobFileSetName FAILED in load_temp\n");
    return -1;
  }
```

```
   /* Opening the BFILE is mandatory: */
   if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
  {
    printf( "OCILobFileOpen FAILED for the bfile load_temp \n");
    return -1;
  }

   /* Opening the LOB is optional: */
   if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp LOB \n");
    return -1;
  }

  if(OCILobLoadFromFile(svchp,
         errhp,
         tblob,
         (OCILobLocator*)bfile,
         (ub4)amount,
         (ub4)1,(ub4)1))
  {
    printf( "OCILobLoadFromFile FAILED\n");
    return -1;
  }

  /* Close the lobs: */
  if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
  {
    printf( "OCILobClose FAILED for bfile \n");
    return -1;
  }

  checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));


  /* Free the temporary LOB now that we are done using it */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }
}
```

Example: Load a Temporary LOB with Data from a BFILE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOAD-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
LOAD-TEMPORARY.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE and BLOB locators:
    EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

* Set up the directory and file information:
    MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "washington_audio" TO FNAME-ARR.
    MOVE 16 TO FNAME-LEN.

    EXEC SQL
       LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
```

```
           FILENAME = :FNAME
       END-EXEC.

   * Open source BFILE and destination temporary BLOB:
       EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
       EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

       EXEC SQL
            LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
       END-EXEC.

   * Close the LOBs:
       EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
       EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

   * Free the temporary LOB:
       EXEC SQL
            LOB FREE TEMPORARY :TEMP-BLOB
       END-EXEC.

   * And free the LOB locators:
       EXEC SQL FREE :TEMP-BLOB END-EXEC.
       EXEC SQL FREE :SRC-BFILE END-EXEC.
       STOP RUN.

    SQL-ERROR.
       EXEC SQL
           WHENEVER SQLERROR CONTINUE
       END-EXEC.
       MOVE ORASLNR TO ORASLNRD.
       DISPLAY " ".
       DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
       DISPLAY " ".
       DISPLAY SQLERRMC.
       EXEC SQL
           ROLLBACK WORK RELEASE
       END-EXEC.
       STOP RUN.
```

Example: Load a Temporary LOB with Data from a BFILE Using C++ (Pro*C/C++)

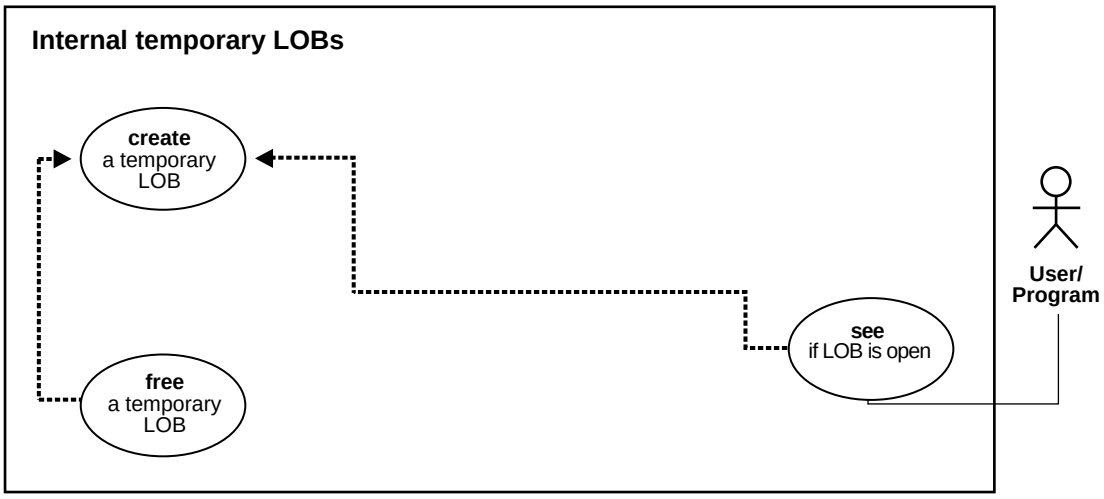
```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>
```

```
void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void loadTempLobFromBFILE_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the BFILE is mandatory; */
  /* Opening the LOB is optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load the data from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Temp_loc;
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Temp_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  loadTempLobFromBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
}
```

See If a Temporary LOB Is Open

Figure 4–7 Use Case Diagram: See If a Temporary LOB Is Open... 206



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This is a generic example takes a locator as input, creates a temporary LOB, opens it and tests if the LOB is open.

- "Example: See If a Temporary LOB Is Open Using PL/SQL" on page 4-37
- "Example: See If a Temporary LOB Is Open Using C (OCI)" on page 4-37
- "Example: See If a Temporary LOB Is Open Using COBOL (Pro*COBOL)" on page 4-38
- "Example: See If a Temporary LOB Is Open Using C++ (Pro*C/C++)" on page 4-40

Example: See If a Temporary LOB Is Open Using PL/SQL

```
/* Note that the example procedure seeTempLOBIsOpen_proc is not part of the
   DBMS_LOB package. This procedure takes a locator as input, creates a
   temporary LOB, opens it and tests if the LOB is open. */
CREATE OR REPLACE PROCEDURE seeTempLOBIsOpen_proc(Lob_loc IN OUT BLOB,
                                                  Retval OUT INTEGER) IS
BEGIN
   /* Create the temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* See If the LOB is open: */
   Retval := DBMS_LOB.ISOPEN(Lob_loc);
  /* The value of Retval will be 1  if  the LOB is open. */
   /* Free the temporary LOB: */
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

Example: See If a Temporary LOB Is Open Using C (OCI)

```
/* This function takes a locator and returns 0 if the function
   completes successfully. The function prints out "Temporary LOB is open" or
  "Temporary LOB is closed". It does not check whether or not the locator is
   actually pointing to a temporary LOB or not, but the open or close test will
   work either way. The function returns 0 if it completes
   successfully, and -1 if it fails. */

sb4 seeTempLOBIsOpen (OCILobLocator *lob_loc,
                      OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
    boolean is_open = FALSE;
    OCILobLocator *tblob;


    printf("in seeTempLOBIsOpen \n");

    if(OCILobCreateTemporary(svchp,
              errhp,
              lob_loc,
              (ub2)0,
              SQLCS_IMPLICIT,
              OCI_TEMP_BLOB,
```

```
                OCI_ATTR_NOCACHE,
                OCI_DURATION_SESSION))
        {
            (void) printf("FAILED: CreateTemporary() \n");
            return -1;
        }

        if(OCILobIsOpen(svchp, errhp, lob_loc, &is_open))
        {
            printf("OCILobIsOpen FAILED\n");
            return -1;
        }
        if(is_open)
        {
            printf("Temporary LOB is open\n");

        }else
        {
            printf("Temporary LOB is closed\n");

        }

        if(OCILobFreeTemporary(svchp,errhp,tblob))
        {
            printf("OCILobFreeTemporary FAILED \n");
            return -1;
        }

        return 0;
    }
```

Example: See If a Temporary LOB Is Open Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. TEMP-LOB-ISOPEN.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID   PIC X(11) VALUES "USER1/USER1".

       01  TEMP-BLOB      SQL-BLOB.
       01  SRC-BFILE      SQL-BFILE.
       01  DIR-ALIAS      PIC X(30) VARYING.
```

```
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP.
01  IS-OPEN        PIC S9(9) COMP.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-ISOPEN.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

* Open temporary LOB:
    EXEC SQL LOB OPEN :TEMP-BLOB READ ONLY END-EXEC.

    EXEC SQL
         LOB DESCRIBE :TEMP-BLOB GET ISOPEN INTO :IS-OPEN
    END-EXEC.

    IF IS-OPEN = 1
*       Logic for an open temporary LOB goes here:
        DISPLAY "Temporary LOB is OPEN."
    ELSE
*       Logic for a closed temporary LOB goes here:
        DISPLAY "Temporary LOB is CLOSED."
    END-IF.

* Close the temporary LOB:
    EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
    EXEC SQL
```

```
            LOB FREE TEMPORARY :TEMP-BLOB
        END-EXEC.

      * And free the LOB locators:
        EXEC SQL FREE :TEMP-BLOB END-EXEC.
        STOP RUN.

       SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

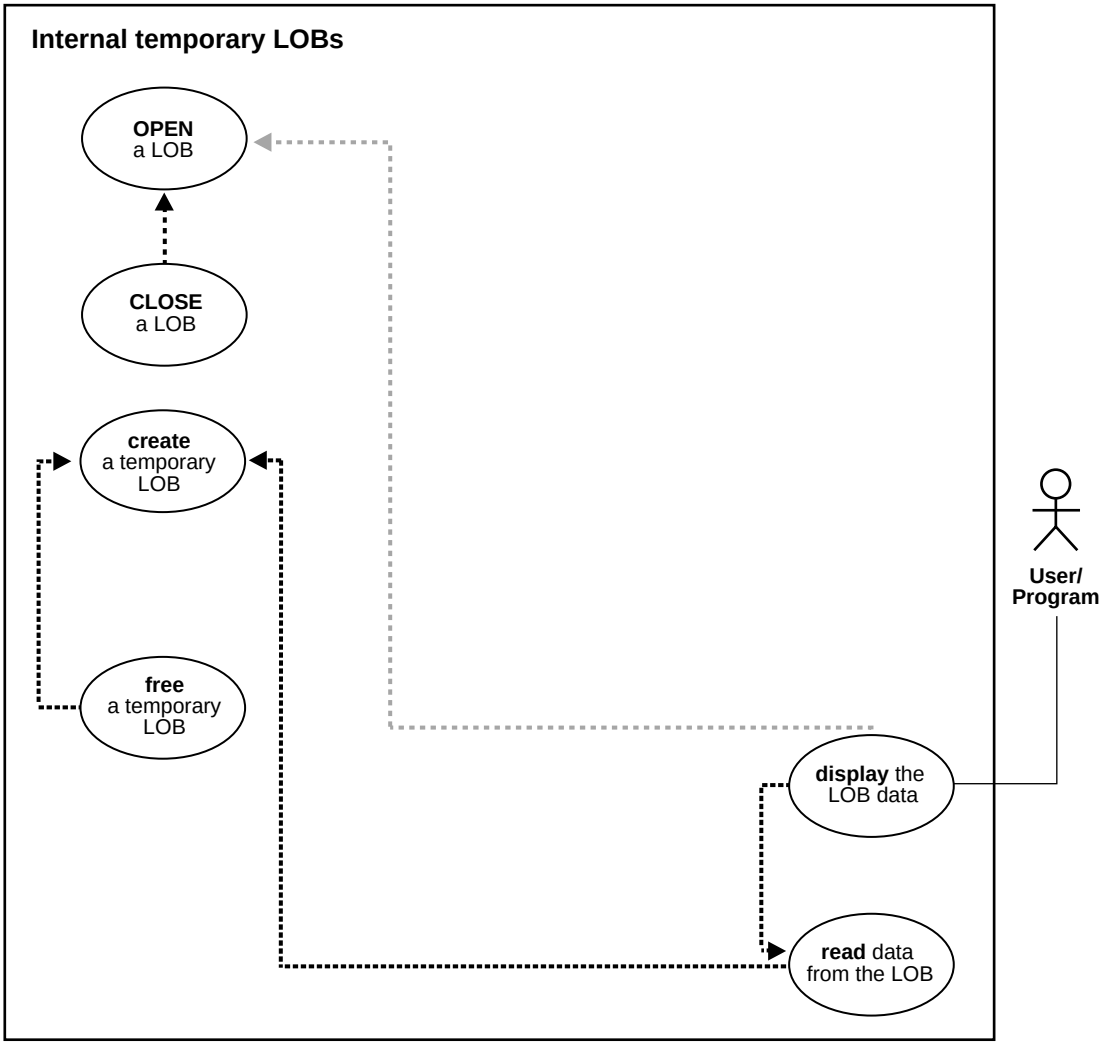
Example: See If a Temporary LOB Is Open Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void tempLobIsOpen_proc()
{
  OCIBlobLocator *Temp_loc;
  int isOpen = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
```

```
  /* Open the Temporary LOB */
  EXEC SQL LOB OPEN :Temp_loc READ ONLY;
  /* Determine if the LOB is Open */
  EXEC SQL LOB DESCRIBE :Temp_loc GET ISOPEN INTO :isOpen;
  if (isOpen)
    printf("Temporary LOB is open\n");
  else
    printf("Temporary LOB is not open\n");
  /* Note that in this example, the LOB is Open so isOpen == 1 (TRUE) */
  /* Close the LOB */
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  tempLobIsOpen_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Display the Temporary LOB Data

Figure 4–8 Use Case Diagram: Display the Temporary LOB data



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

As an instance of displaying a LOB, our example stream-reads the image Drawing from the column object Map_obj onto the client-side in order to view the data.

- "Example: Display the Temporary LOB Data Using C (OCI)" on page 4-44
- "Example: Display the Temporary LOB Data Using COBOL (Pro*COBOL)" on page 4-47
- "Example: Display the Temporary LOB Data Using C++ (Pro*C/C++)" on page 4-49

Example: Display the Temporary LOB Data Using PL/SQL (DBMS_LOB Package)

```
/* The following function acceses the Washington_audio file, creates a temporary
  LOB, loads some data from the file, and then reads it back and
  displays it. */
DECLARE
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount         INTEGER := 128;
   Bbuf           RAW(128);
   Position       INTEGER :=1;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc,Src_loc,Amount);

   LOOP
      DBMS_LOB.READ (Dest_loc, Amount, Position, Bbuf);
      /* Display the buffer contents: */
      DBMS_OUTPUT.PUT_LINE('Result :'|| utl_raw.cast_to_varchar2(Bbuf));
      Position := Position + Amount;
   END LOOP;
EXCEPTION
```

```
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data loaded into temp LOB');

    DBMS_LOB.CLOSE(Dest_loc);
    DBMS_LOB.FREETEMPORARY(Dest_loc);
    /* Closing the file is mandatory unless you close the files later: */
    DBMS_LOB.CLOSE(Src_loc);
END;
```

Example: Display the Temporary LOB Data Using C (OCI)

```
/* The following function acceses the Washington_audio file, creates a temporary
  LOB, loads some data from the file, and then reads it back and
  displays it. The reading is done in a streaming fashion. This function assumes
  that the file specified is kept in the directory known by the directory alias
  "AUDIO_DIR". It also assumes that the file is at least 14000 bytes long, which
  is the amount specified to be read and loaded. These amounts are arbitrary for
  this example. This function uses fprintf() to display the contents of the
  file. This works well for text data, but you may wish to change the method for
  binary data. For audio  data, you could, for instance, call an audio function.
  The function returns 0 if it completes successfully, and -1 if it fails. */

#define MAXBUFLEN 32767

sb4 display_file_to_lob( OCIError      *errhp,
                         OCISvcCtx     *svchp,
                         OCIStmt       *stmthp,
                         OCIEnv        *envhp)
{
  int rowind;
  char *binfile;
  OCILobLocator *tblob;
  OCILobLocator *bfile;


  ub4 amount = 14000;
  ub4 offset = 0;
  ub4 loblen = 0;
  ub4 amtp   = 0;
  sword retval;
  ub4 piece  = 1;
  ub4 remainder= 0;
  ub1 bufp[MAXBUFLEN];
  sb4 return_code = 0;
```

```
  (void) printf("\n===> Testing loading files into lobs and displaying
them\n\n");

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp, errhp, tblob,(ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
      (void) printf("FAILED: CreateTemporary() \n");
      return -1;
  }


  if(OCILobFileSetName(envhp, errhp, &bfile, (text*)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"),(text*)"Washington_audio",
                       (ub2)strlen("Wasthington_audio")))
  {
      printf("OCILobFileSetName FAILED\n");
      return_code = -1;
  }

  /* Open the BFILE: */
  if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
  {
     printf( "OCILobFileOpen FAILED \n");
     return_code = -1;
  }
```

```
if(OCILobLoadFromFile(svchp,errhp,tblob,(OCILobLocator*)bfile,(ub4)amount,
                      (ub4)1,(ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
}

offset = 1;
memset(bufp, '\0', MAXBUFLEN);

retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                   (dvoid *) bufp, (amount < MAXBUFLEN ? amount : MAXBUFLEN),
                    (dvoid *)0, (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                    (ub2) 0, (ub1) SQLCS_IMPLICIT);

printf("1st piece read from file is %s\n",bufp);

switch (retval)
{
  case OCI_SUCCESS:              /* Only one piece */
    (void) printf("stream read piece # %d \n", ++piece);
    (void)printf("piece read was %s\n",bufp);
    break;
  case OCI_FAILURE:
    /*  report_error();  function not shown here */
    break;
  case OCI_NEED_DATA:            /* There are 2 or more pieces */
    remainder = amount;
    printf("remainder is %d \n",remainder);
    do
    {
         memset(bufp, '\0', MAXBUFLEN);
         amtp = 0;
         remainder -= MAXBUFLEN;
         printf("remainder is %d \n",remainder);
         retval = OCILobRead(svchp, errhp, tblob,  &amtp, offset,
                             (dvoid *) bufp, (ub4) MAXBUFLEN, (dvoid *)0,
                             (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                             (ub2) 0, (ub1) SQLCS_IMPLICIT);

      /* The amount read returned is undefined for FIRST, NEXT pieces: */
          (void)fprintf(stderr,"stream read %d th piece, amtp = %d\n",
                        ++piece, amtp);
          (void)fprintf(stderr,"piece  of length read was %d\n",
              strlen((const char*)bufp));
```

```
            (void)fprintf(stderr,"piece read was %s\n",bufp);
         } while (retval == OCI_NEED_DATA);
         break;
      default:
        (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
         break;
    }

     /* Close the audio file: */
    if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
    {
       printf( "OCILobFileClose FAILED\n");
       return_code =  -1;
    }
    /* clean up the temp LOB now that we are done with it */

    if(check_and_free_temp(tblob, errhp,  svchp,stmthp, envhp))
    {
        printf("check and free failed in load test\n");
        return_code = -1;
    }
    return return_code;
}
```

Example: Display the Temporary LOB Data Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. ONE-READ-BLOB.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID         PIC X(9) VALUES "SAMP/SAMP".
     01  TEMP-BLOB      SQL-BLOB.
     01  SRC-BFILE      SQL-BFILE.
     01  DIR-ALIAS      PIC X(30) VARYING.
     01  FNAME          PIC X(20) VARYING.
     01  BUFFER2        PIC X(32767) VARYING.
     01  AMT            PIC S9(9) COMP.
     01  OFFSET         PIC S9(9) COMP VALUE 1.
     01  ORASLNRD       PIC 9(4).
     01  ISTEMP         PIC S9(9) COMP.

         EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
           EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
           EXEC SQL INCLUDE ORACA END-EXEC.

           EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

       PROCEDURE DIVISION.
       ONE-READ-BLOB.

           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
                CONNECT :USERID
           END-EXEC.

      * Allocate and initialize the BLOB locator:
           EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
           EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
           EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.

           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

      * Set up the directory and file information:
           MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
           MOVE 9 TO DIR-ALIAS-LEN.
           MOVE "Washington_audio" TO FNAME-ARR.
           MOVE 16 TO FNAME-LEN.

           EXEC SQL
              LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
              FILENAME = :FNAME
           END-EXEC.

           EXEC SQL
              LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
           END-EXEC.

      * Open source BFILE and destination temporary BLOB:
           EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
           EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

           EXEC SQL
                LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
           END-EXEC.

      * Perform a single read:
```

```
            EXEC SQL
                 LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
            END-EXEC.

            DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

        END-OF-BLOB.
            EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
            EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
            EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Display the Temporary LOB Data Using C++ (Pro*C/C++)

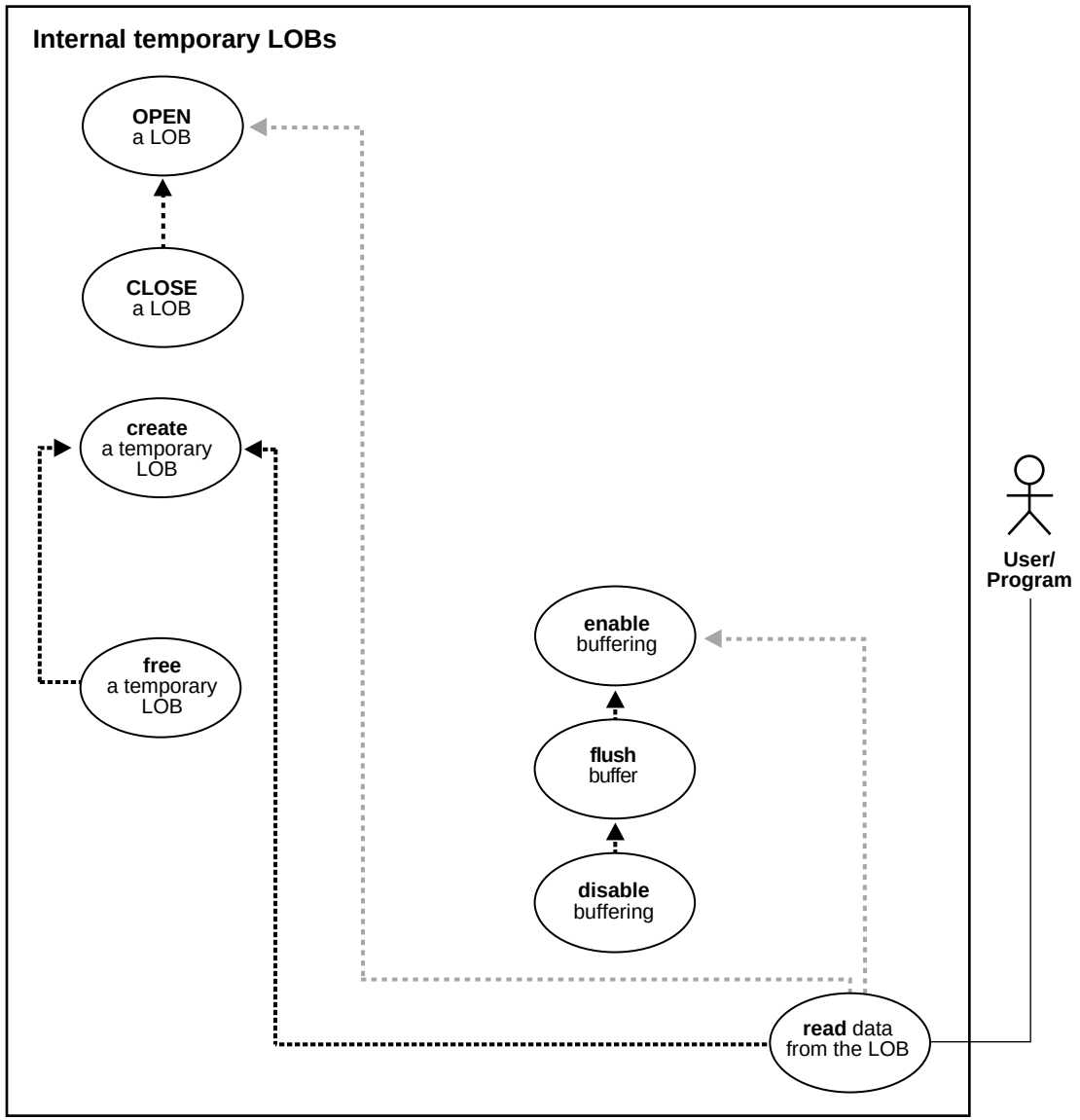
```
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void displayTempLOB_proc()
```

```
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  int Position = 1;
  /* Datatype Equivalencing is Mandatory for this Datatype */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the LOB Locators */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Amount;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc;
  /* Setting Amount = 0 will initiate the polling method */
  Amount = 0;
  /* Set the maximum size of the Buffer */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the BLOB into the Buffer */
      EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
      printf("Display %d bytes\n", Buffer.Length);
    }
  printf("Display %d bytes\n", Amount);
  /* Closing the LOBs is mandatory if you have opened them */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator */
  EXEC SQL FREE :Temp_loc;
}
```

```
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Read Data from a Temporary LOB

Figure 4–9 Use Case Diagram: Read Data from a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Stream Read

The most efficient way to read large amounts of LOB data is to use `OCILobRead`() with the streaming mechanism enabled via polling or a callback.

When reading the `LOB` value, it is not an error to try to read beyond the end of the `LOB`. This means that you can always specify an input amount of 4 gigabytes regardless of the starting offset and the amount of data in the LOB. You do not need to incur a round-trip to the server to call `OCILobGetLength`() to find out the length of the LOB value in order to determine the amount to read.

For example, assume that the length of a LOB is 5,000 bytes and you want to read the entire LOB value starting at offset 1,000. Also assume that you do not know the current length of the LOB value. Here's the OCI read call, excluding the initialization of the parameters:

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

When using polling mode, be sure to look at the value of the '`amount`' parameter after each `OCILobRead`() call to see how many bytes were read into the buffer since the buffer may not be entirely full.

When using callbacks, the '`len`' parameter, which is input to the callback, will indicate how many bytes are filled in the buffer. Be sure to check the '`len`' parameter during your callback processing since the entire buffer may not be filled with data (see the*Oracle Call Interface Programmer's Guide*.).

Scenario

Our example reads the data from a single video Frame.

- "Example: Read Data from a Temporary LOB Using PL/SQL (DBMS_LOB Package)" on page 4-54
- "Example: Read Data from a Temporary LOB Using C (OCI)" on page 4-54

- "Example: Read Data from a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-57
- "Example: Read Data from a Temporary LOB Using C++ (Pro*C/C++)" on page 4-59

Example: Read Data from a Temporary LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that PL/SQL does not support streaming reads. The OCI example will
   illustrate streaming reads: */
DECLARE
  Dest_loc       BLOB;
  Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
  Amount         INTEGER := 4000;
  Bbuf           RAW(32767);
  Position       INTEGER :=1;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
    /* Opening the FILE is mandatory: */
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    /* Opening the LOB is optional: */
    DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
    DBMS_LOB.READ (Dest_loc, Amount, Position, Bbuf);
    /* Closing the LOB is mandatory if you have opened it: */
    DBMS_LOB.CLOSE(Src_loc);
```

Example: Read Data from a Temporary LOB Using C (OCI)

```
/* This is the same example as was shown for reading and displaying data from a
   temporary LOB. This function takes the Washinton_audio file, opens that file
   as a BFILE as input, loads that file data into a temporary LOB and then reads
   the data from the temporary LOB 5000 or less bytes at a time.
   5000 bytes was an arbitrary maximum buffer length chosen for this example.
   The function returns 0 if it completes successfully, and -1 if it fails. */

#define MAXBUFLEN 32767

sb4 test_file_to_lob (OCILobLocator *lob_loc,
                      OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  int rowind;
```

```
 OCILobLocator *tblob;
 OCILobLocator *bfile;

 ub4 amount = 14000;
 ub4 offset =0;
 ub4   loblen = 0;
 ub4   amtp = 0;
 sword retval;
 ub4   piece = 1;
 ub4   remainder=0;
 ub1 bufp[MAXBUFLEN];

 (void) printf(
            "\n===> Testing loading files into lobs and displaying them\n\n");

/* Create a temporary LOB: */
if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
}
if(OCILobFileSetName(envhp, errhp, &bfile,(text*)"AUDIO_DIR",
                     (ub2)strlen("AUDIO_DIR"),
                     (text*)"Washington_audio",
                     (ub2)strlen("Washington_audio")))
 {
     printf("OCILobFileSetName FAILED\n");
     return -1;
 }
 if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
{
    printf( "OCILobFileOpen FAILED \n");
    return -1;
}
if(OCILobLoadFromFile(svchp,errhp,tblob,(OCILobLocator*)bfile,(ub4)amount,
                      (ub4)1,(ub4)1))
{
    printf( "OCILobLoadFromFile FAILED\n");
    return -1;
}

offset = 1;
memset(bufp, '\0', MAXBUFLEN);
```

```
retval = OCILobRead(svchp, errhp, tblob, &amtp, offset, (dvoid *) bufp,
                    (amount < MAXBUFLEN ? amount : MAXBUFLEN), (dvoid *)0,
                    (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                    (ub2) 0, (ub1) SQLCS_IMPLICIT);
 fprintf(stderr,"1st piece read from file is %s\n",bufp);

 switch (retval)
 {
   case OCI_SUCCESS:              /* Only one piece */
     (void) printf("stream read piece # %d \n", ++piece);
     (void)printf("piece read was %s\n",bufp);
     break;
   case OCI_FAILURE:
     /*  report_error();  function not shown here */
     break;
   case OCI_NEED_DATA:            /* There are 2 or more pieces */
     remainder = amount;
     fprintf(stderr,"remainder is %d \n",remainder);
     do
     {
          memset(bufp, '\0', MAXBUFLEN);
          amtp = 0;
          remainder -= MAXBUFLEN;
          fprintf(stderr,"remainder is %d \n",remainder);

           retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                              (dvoid *) bufp,(ub4) MAXBUFLEN, (dvoid *)0,
                              (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                              (ub2) 0, (ub1) SQLCS_IMPLICIT);

       /* The amount read returned is undefined for FIRST, NEXT pieces: */
           (void)fprintf(stderr,"stream read %d th piece, amtp = %d\n",
                        ++piece, amtp);
           (void)fprintf(stderr,
               "piece  of length read was %d\n",strlen((const  char *)bufp));
           (void)fprintf(stderr,"piece read was %s\n",bufp);
      } while (retval == OCI_NEED_DATA);
      break;
   default:
     (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
      break;
 }

  /* Close the audio file: */
```

```
  if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
  {
     printf( "OCILobFileClose FAILED\n");
     return -1;
  }

  /* Clean up the temp LOB now that we are done with it: */
  if(check_and_free_temp(lob_loc, errhp, svchp,stmthp, envhp))
  {
      printf("check and free failed in load test\n");
      return -1;
  }
  return 0;
}
```

Example: Read Data from a Temporary LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. ONE-READ-BLOB.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID          PIC X(9) VALUES "SAMP/SAMP".
     01  TEMP-BLOB       SQL-BLOB.
     01  SRC-BFILE       SQL-BFILE.
     01  DIR-ALIAS       PIC X(30) VARYING.
     01  FNAME           PIC X(20) VARYING.
     01  BUFFER2         PIC X(32767) VARYING.
     01  AMT             PIC S9(9) COMP.
     01  OFFSET          PIC S9(9) COMP VALUE 1.
     01  ORASLNRD        PIC 9(4).
     01  ISTEMP          PIC S9(9) COMP.

         EXEC SQL INCLUDE SQLCA END-EXEC.
         EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
         EXEC SQL INCLUDE ORACA END-EXEC.

         EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

     PROCEDURE DIVISION.
     ONE-READ-BLOB.

         EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
```

```
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "Washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

     EXEC SQL
        LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Perform a single read:

     EXEC SQL
          LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
     END-EXEC.

     DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

 END-OF-BLOB.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
```

```
            EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

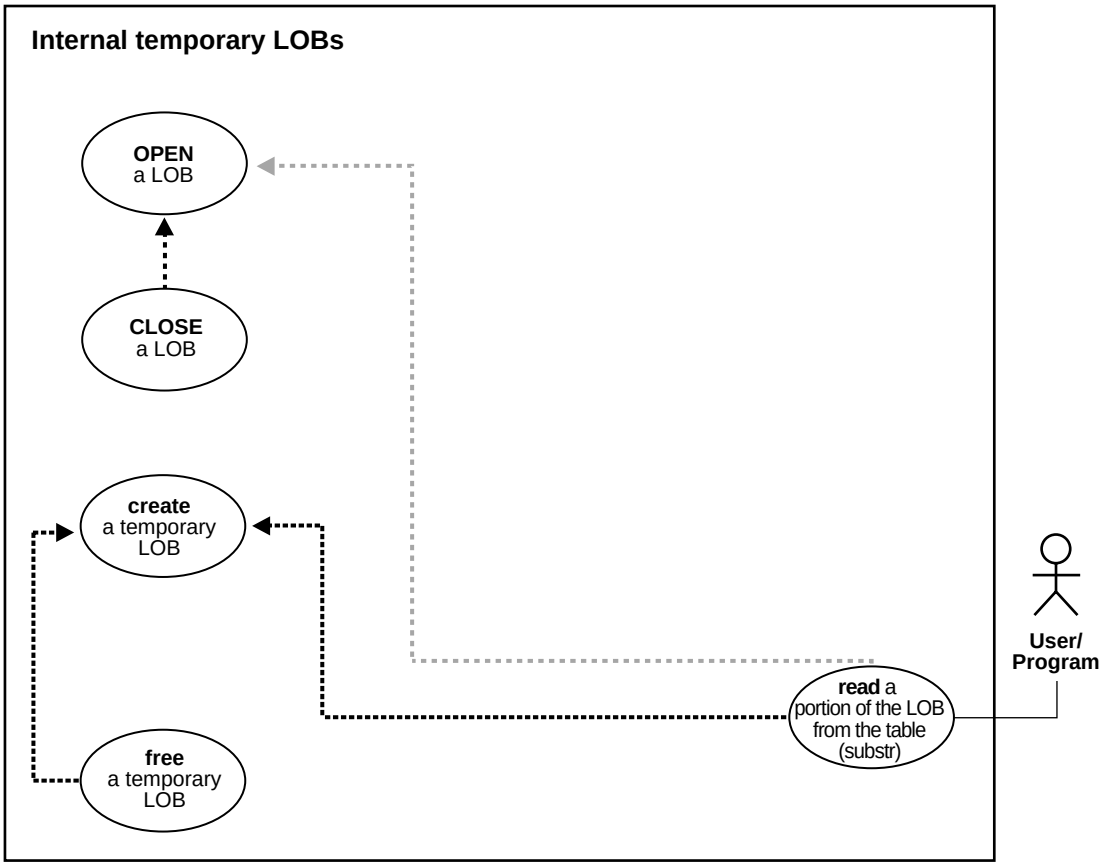
Example: Read Data from a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void readTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Length, Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
```

```
  } Buffer;
  /* Datatype Equivalencing is Mandatory for this Datatype */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the BFILE Locator */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Determine the Length of the BFILE */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* Allocate and Create the Temporary LOB */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Open the BFILE for Reading */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Load the BFILE into the Temporary LOB */
  Amount = Length;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Close the BFILE */
  EXEC SQL LOB CLOSE :Lob_loc;
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the Temporary LOB into the Buffer */
      EXEC SQL LOB READ :Amount FROM :Temp_loc INTO :Buffer;
      printf("Read %d bytes\n", Buffer.Length);
    }
  printf("Read %d bytes\n", Amount);
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators */
  EXEC SQL FREE :Temp_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  readTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Read a Portion of the Temporary LOB (substr)

Figure 4–10 Use Case Diagram: Read a portion of the Temporary LOB from the Table (substr)



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This example shows the operation in terms of reading a portion from sound-effect Sound.

- "Example: Read a Portion of the Temporary LOB (substr) Using PL/SQL (DBMS_LOB Package)" on page 4-62
- "Example: Read a Portion of the Temporary LOB (substr) Using COBOL (Pro*COBOL)" on page 4-62
- "Example: Read a Portion of the Temporary LOB (substr) Using C++ (Pro*C/C++)" on page 4-65

Example: Read a Portion of the Temporary LOB (substr) Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure substringTempLOB_proc is not part of the
   DBMS_LOB package. */
/* This example assumes the user has a 'Washington_audio' file in a
   directory which has a AUDIO_DIR alias */
CREATE or REPLACE PROCEDURE substringTempLOB_proc IS
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount         INTEGER := 32767;
   Bbuf           RAW(32767);
   Position       INTEGER :=128;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   Bbuf := DBMS_LOB.SUBSTR(Dest_loc, Amount, Position);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.CLOSE(Dest_loc);
END;
```

Example: Read a Portion of the Temporary LOB (substr) Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ONE-READ-BLOB.
```

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(9) VALUES "SAMP/SAMP".
01  TEMP-BLOB       SQL-BLOB.
01  SRC-BFILE       SQL-BFILE.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  BUFFER2         PIC X(32767) VARYING.
01  AMT             PIC S9(9) COMP.
01  OFFSET          PIC S9(9) COMP VALUE 1.
01  ORASLNRD        PIC 9(4).
01  ISTEMP          PIC S9(9) COMP.

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

    EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

PROCEDURE DIVISION.
ONE-READ-BLOB.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locator
    EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL LOB CREATE TEMPORARY :TEMP-BLOB END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.

* Set up the directory and file information
    MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "Washington_audio" TO FNAME-ARR.
    MOVE 16 TO FNAME-LEN.

    EXEC SQL
       LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
       FILENAME = :FNAME
```

```
        END-EXEC.

        EXEC SQL
           LOB DESCRIBE :SRC-BFILE GET LENGTH INTO :AMT
        END-EXEC.

    * Open source BFILE and destination temporary BLOB.
        EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
        EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

        EXEC SQL
             LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
        END-EXEC.

    * Perform a single read

        EXEC SQL
             LOB READ :AMT FROM :TEMP-BLOB INTO :BUFFER2
        END-EXEC.

        DISPLAY "Read ", BUFFER2, " from TEMP-BLOB".

     END-OF-BLOB.
        EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
        EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
        EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.
        EXEC SQL FREE :TEMP-BLOB END-EXEC.
        EXEC SQL FREE :SRC-BFILE END-EXEC.
        STOP RUN.

     SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

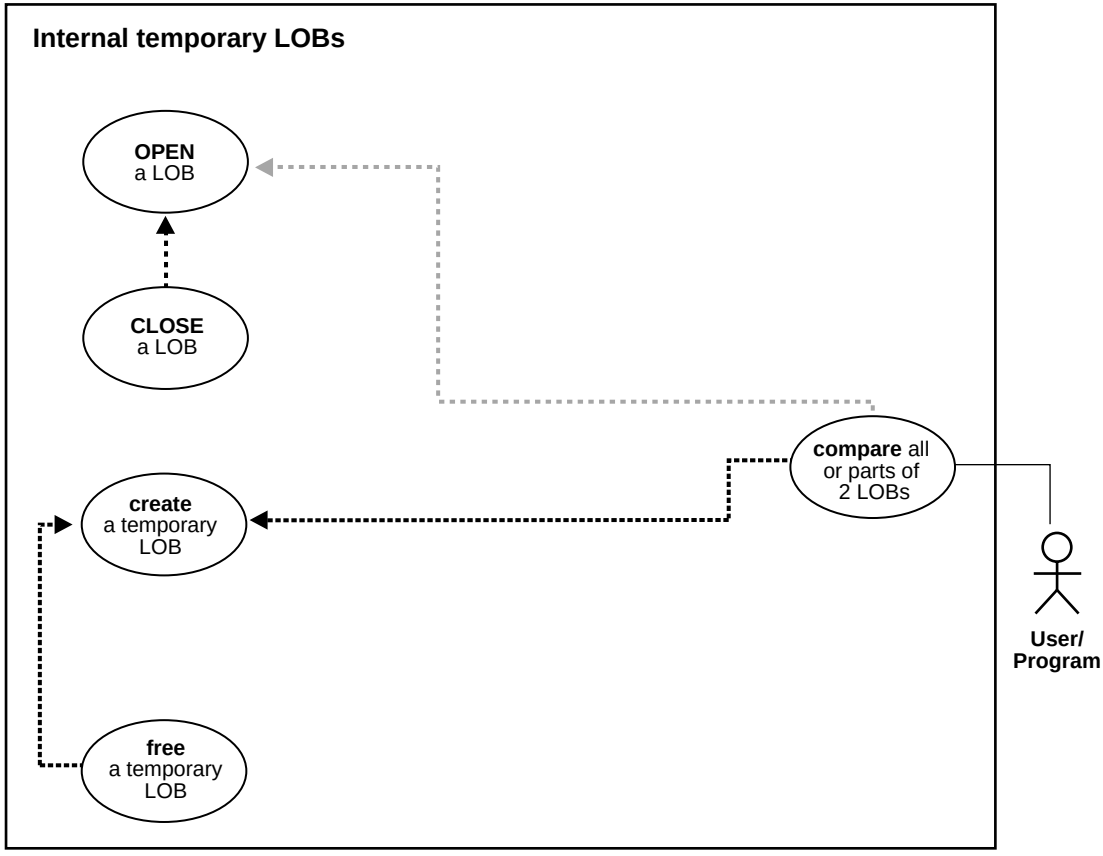
Example: Read a Portion of the Temporary LOB (substr) Using C++ (Pro*C/C++)

```
/* Pro*C/C++ lacks an equivalent embedded SQL form for the DBMS_LOB.SUBSTR()
   function.  However, Pro*C/C++ can interoperate with PL/SQL using
   anonymous PL/SQL blocks embedded in a Pro*C/C++ program as this example
   shows. */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 4096

void substringTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Position = 1024;
  unsigned int Length;
  int Amount = BufferLength;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Open the LOBs: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
```

```
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Determine the length of the BFILE and load it into the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  EXEC SQL LOB LOAD :Length FROM FILE :Lob_loc INTO :Temp_loc;
  /* Invoke SUBSTR() on the Temporary LOB inside a PL/SQL block: */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Temp_loc, :Amount, :Position);
    END;
  END-EXEC;
  /* Process the Data in the Buffer. */
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources used by the locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Compare All or Part of Two (Temporary) LOBs

Figure 4–11 Use Case Diagram: Compare All or Part of Two Temporary LOBs



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

The following example compares two frames from the archival table `VideoframesLib_tab` to see whether they are different and, depending on the result of comparison, inserts the Frame into the `Multimedia_tab`.

- "Example: Compare All or Part of Two (Temporary) LOBs Using PL/SQL (DBMS_LOB Package)" on page 4-68
- "Example: Compare All or Part of Two (Temporary) LOBs Using COBOL (Pro*COBOL)" on page 4-69
- "Example: Compare All or Part of Two (Temporary) LOBs Using C++ (Pro*C/C++)" on page 4-71

Example: Compare All or Part of Two (Temporary) LOBs Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure compareTwoTemporPersistLOBs_proc is not part
   of the DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE compareTwoTemporPersistLOBs_proc IS
    Lob_loc1 BLOB;
    Lob_loc2 BLOB;
    Temp_loc BLOB;
    Amount   INTEGER := 32767;
    Retval   INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT Frame INTO Lob_loc1 FROM Multimedia_tab
        WHERE Clip_ID = 1;
    SELECT Frame INTO Lob_loc2 FROM Multimedia_tab
        WHERE Clip_ID = 2;
    /* Copy a frame into a temp LOB and convert it to a different format */
    /* before comparing the frames : */
    DBMS_LOB.CREATETEMPORARY(Temp_loc, TRUE, DBMS_LOB.SESSION);
    DBMS_LOB.OPEN(Temp_loc, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
    DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READONLY);
    /* Copy the persistent LOB into the temp LOB: */
    DBMS_LOB.COPY(Temp_loc,Lob_loc2,DBMS_LOB.GETLENGTH(Lob_loc2),1,1);
    /* Perform some conversion function on the temp LOB before comparing it*/
    /* ...some_conversion_format_function(Temp_loc); */
    retval := DBMS_LOB.COMPARE(Lob_loc1, Temp_loc, Amount, 1, 1);
    IF retval = 0 THEN
       DBMS_OUTPUT.PUT_LINE('Processing for equal frames');
```

```
ELSE
   DBMS_OUTPUT.PUT_LINE('Processing for non-equal frames');
END IF;
DBMS_LOB.CLOSE(Temp_loc);
DBMS_LOB.CLOSE(Lob_loc1);
DBMS_LOB.CLOSE(Lob_loc2);
/* Free the temporary LOB now that we are done using it: */
DBMS_LOB.FREETEMPORARY(Temp_loc);
END;
```

Example: Compare All or Part of Two (Temporary) LOBs Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".

01  BLOB1          SQL-BLOB.
01  BLOB2          SQL-BLOB.
01  TEMP-BLOB      SQL-BLOB.
01  RET            PIC S9(9) COMP.
01  AMT            PIC S9(9) COMP VALUE 5.
01 ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.


PROCEDURE DIVISION.
BLOB-COMPARE.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :BLOB1 END-EXEC.
    EXEC SQL ALLOCATE :BLOB2 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BLOB END-EXEC.
```

```
     EXEC SQL
          SELECT FRAME INTO :BLOB1
          FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 1
     END-EXEC.

     EXEC SQL
          SELECT FRAME INTO :BLOB2
          FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 2
     END-EXEC.

* Allocate and create a temporary LOB:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Open the BLOBs for READ ONLY, Open temp LOB READ/WRITE:
     EXEC SQL LOB OPEN :BLOB1 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :BLOB2 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.

* Copy data from BLOB2 to the temporary BLOB:
     EXEC SQL
          LOB COPY :AMT FROM :BLOB2 TO :TEMP-BLOB
     END-EXEC.

* Execute PL/SQL to use its COMPARE functionality:
     MOVE 5 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BLOB1,:TEMP-BLOB,:AMT,1,1);
       END;
     END-EXEC.

     IF RET = 0
*         Logic for equal BLOBs goes here
          DISPLAY "BLOBs are equal"
     ELSE
*         Logic for unequal BLOBs goes here
          DISPLAY "BLOBs are not equal"
     END-IF.

     EXEC SQL LOB CLOSE :BLOB1 END-EXEC.
     EXEC SQL LOB CLOSE :BLOB2 END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.
```

```
            EXEC SQL LOB FREE TEMPORARY :TEMP-BLOB END-EXEC.

            EXEC SQL FREE :TEMP-BLOB END-EXEC.

        END-OF-BLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BLOB1 END-EXEC.
            EXEC SQL FREE :BLOB2 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Compare All or Part of Two (Temporary) LOBs Using C++ (Pro*C/C++)

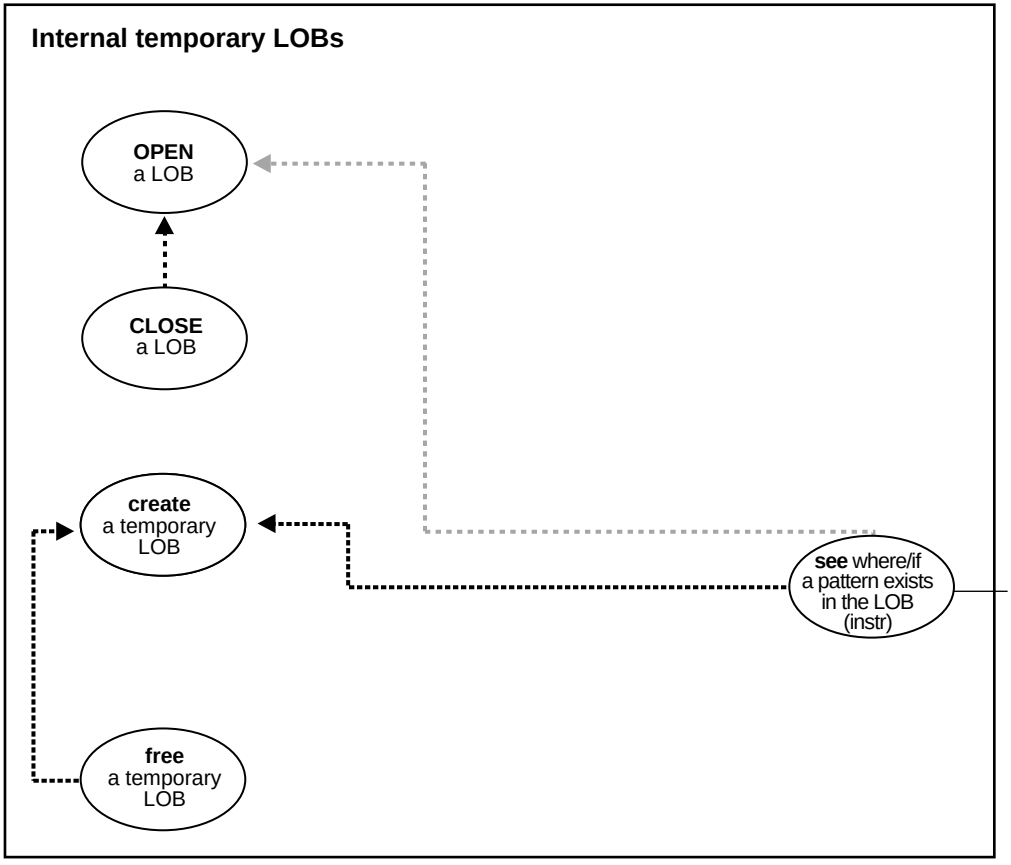
```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void compareTwoTempOrPersistLOBs_proc()
{
  OCIBlobLocator *Lob_loc1, *Lob_loc2, *Temp_loc;
  int Amount = 128;
  int Retval;
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the LOB locators: */
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Select the LOBs: */
  EXEC SQL SELECT Frame INTO :Lob_loc1
     FROM Multimedia_tab WHERE Clip_ID = 1;
  EXEC SQL SELECT Frame INTO :Lob_loc2
     FROM Multimedia_tab WHERE Clip_ID = 2;
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Copy the Persistent LOB into the Temporary LOB: */
  EXEC SQL LOB COPY :Amount FROM :Lob_loc2 TO :Temp_loc;
  /* Compare the two Frames using DBMS_LOB.COMPARE() from within PL/SQL: */
  EXEC SQL EXECUTE
     BEGIN
       :Retval := DBMS_LOB.COMPARE(:Lob_loc1, :Temp_loc, :Amount, 1, 1);
     END;
  END-EXEC;
  if (0 == Retval)
    printf("Frames are equal\n");
  else
    printf("Frames are not equal\n");
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareTwoTempOrPersistLOBs_proc();
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

See If a Pattern Exists in a Temporary LOB (instr)

Figure 4–12 Use Case Diagram: See If a Pattern Exists in a Temporary LOB (instr)



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs"

Scenario

The following example examines the storyboard text to see if the string "children" is present.

- "Example: See If a Pattern Exists in a Temporary LOB (instr) Using PL/SQL (DBMS_LOB Package)" on page 4-75
- "Example: See If a Pattern Exists in a Temporary LOB (instr) Using COBOL (Pro*COBOL)" on page 4-76
- "Example: See If a Pattern Exists in a Temporary LOB (instr) Using C++ (Pro*C/C++)" on page 4-78

Example: See If a Pattern Exists in a Temporary LOB (instr) Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure instringTempLOB_proc is not part of the
   DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE instringTempLOB_proc IS
   Lob_loc        CLOB;
   Temp_clob      CLOB;
   Pattern        VARCHAR2(30) := 'children';    Position       INTEGER := 0;
   Offset         INTEGER := 1;
   Occurrence     INTEGER := 1;
BEGIN
   /* Create the temp LOB and copy a CLOB into it: */
   DBMS_LOB.CREATETEMPORARY(Temp_clob,TRUE, DBMS_LOB.SESSION);
   SELECT Story INTO Lob_loc
      FROM Multimedia_tab
         WHERE Clip_ID = 1;

   DBMS_LOB.OPEN(Temp_clob,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Lob_loc,DBMS_LOB.LOB_READONLY);
   /* Copy the CLOB into the temp CLOB: */
   DBMS_LOB.COPY(Temp_clob,Lob_loc,DBMS_LOB.GETLENGTH(Lob_loc),1,1);
   /* Seek the pattern in the temp CLOB: */
   Position := DBMS_LOB.INSTR(Temp_clob, Pattern, Offset, Occurrence);
   IF Position = 0 THEN
      DBMS_OUTPUT.PUT_LINE('Pattern not found');
   ELSE
      DBMS_OUTPUT.PUT_LINE('The pattern occurs at '|| position);
   END IF;
   DBMS_LOB.CLOSE(Lob_loc);
   DBMS_LOB.CLOSE(Temp_clob);
```

```
   /* Free the temporary LOB: */
   DBMS_LOB.FREETEMPORARY(Temp_clob);
END;
```

Example: See If a Pattern Exists in a Temporary LOB (instr) Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CLOB-INSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".

01  CLOB1          SQL-CLOB.
01  TEMP-CLOB      SQL-CLOB.
01  PATTERN        PIC X(8) VALUE "children".
01  BUFFER2        PIC X(32767) VARYING.
01  OFFSET         PIC S9(9) COMP VALUE 1.
01  OCCURRENCE     PIC S9(9) COMP VALUE 1.
01  LEN            PIC S9(9) COMP.
01  POS            PIC S9(9) COMP.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

    EXEC SQL VAR BUFFER2 IS LONG RAW(32767) END-EXEC.

PROCEDURE DIVISION.
CLOB-INSTR.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :CLOB1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
```

```
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-CLOB END-EXEC.
     EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
     EXEC SQL
          SELECT STORY INTO :CLOB1
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
     END-EXEC.
     EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-CLOB
     END-EXEC.

* Open the CLOB for READ ONLY:
     EXEC SQL LOB OPEN :CLOB1 READ ONLY END-EXEC.

* Use LOB describe to get the length of CLOB1:
     EXEC SQL
          LOB DESCRIBE :CLOB1 GET LENGTH INTO :LEN
     END-EXEC.
     EXEC SQL
          LOB COPY :LEN FROM :CLOB1 TO :TEMP-CLOB
     END-EXEC.

* Execute PL/SQL to get INSTR functionality:
     EXEC SQL EXECUTE
       BEGIN
         :POS := DBMS_LOB.INSTR(:TEMP-CLOB,:PATTERN,
                                :OFFSET, :OCCURRENCE);
       END;
     END-EXEC.

     IF POS = 0
*         Logic for pattern not found here
          DISPLAY "Pattern was not found"
     ELSE
*         Pos contains position where pattern is found
          DISPLAY "Pattern was found"
     END-IF.

* Close and free the LOBs:
     EXEC SQL LOB CLOSE :CLOB1 END-EXEC.
     EXEC SQL FREE :TEMP-CLOB END-EXEC.
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-CLOB
     END-EXEC.
```

```
            EXEC SQL FREE :TEMP-CLOB END-EXEC.

        END-OF-CLOB.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :CLOB1 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

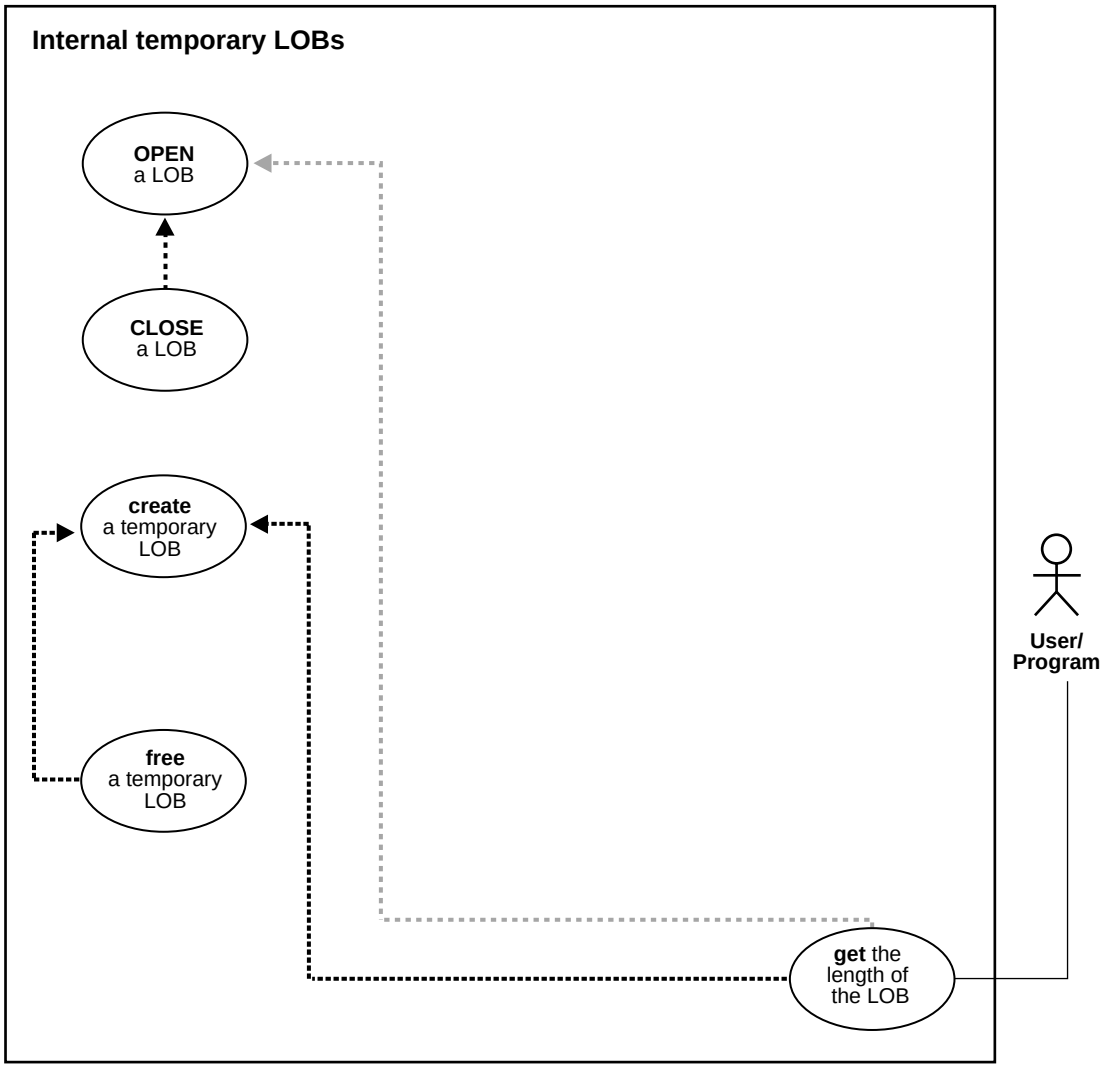
Example: See If a Pattern Exists in a Temporary LOB (instr) Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void instringTempLOB_proc()
{
  OCIClobLocator *Lob_loc, *Temp_loc;
  char *Pattern = "The End";
  unsigned int Length;
  int Position = 0;
  int Offset = 1;
  int Occurrence = 1;
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Persistent LOB: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Story INTO :Lob_loc
     FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Determine the Length of the Persistent LOB: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH into :Length;
  /* Copy the Persistent LOB into the Temporary LOB: */
  EXEC SQL LOB COPY :Length FROM :Lob_loc TO :Temp_loc;
  /* Seek the Pattern using DBMS_LOB.INSTR() in a PL/SQL block: */
  EXEC SQL EXECUTE
     BEGIN
        :Position :=
            DBMS_LOB.INSTR(:Temp_loc, :Pattern, :Offset, :Occurrence);
     END;
  END-EXEC;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  /* Closing the LOBs is mandatory if you have opened them: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  instringTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Get the Length of a Temporary LOB

Figure 4–13 Use Case Diagram: Get the length of a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

The following example gets the length of interview to see if it will run over the 4 gigabyte limit.

- "Example: Get the Length of a Temporary LOB Using PL/SQL (DBMS_LOB Package)" on page 4-81
- "Example: Get the Length of a Temporary LOB Using C (OCI)" on page 4-82
- "Example: Get the Length of a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-84
- "Example: Get the Length of a Temporary LOB Using C++ (Pro*C/C++)" on page 4-86

Example: Get the Length of a Temporary LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure getLengthTempCLOB_proc is not part of the
   DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE getLengthTempCLOB_proc IS
   Length      INTEGER;
   tlob        CLOB;
   bufc        VARCHAR2(8);
   Amount      NUMBER;
   pos         NUMBER;
   Src_loc     BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
BEGIN
    DBMS_LOB.CREATETEMPORARY(tlob,TRUE,DBMS_LOB.SESSION);
    /* Opening the LOB is optional: */
    DBMS_LOB.OPEN(tlob,DBMS_LOB.LOB_READWRITE);
    /* Opening the file is mandatory: */
    DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
    Amount := 32767;
    DBMS_LOB.LOADFROMFILE(tlob, Src_loc, Amount);
    /* Get the length of the LOB: */
    length := DBMS_LOB.GETLENGTH(tlob);
    IF length = 0 THEN
        DBMS_OUTPUT.PUT_LINE('LOB is empty.');
```

```
    ELSE
        DBMS_OUTPUT.PUT_LINE('The length is ' || length);
    END IF;
     /* Must close any lobs that were opened: */
    DBMS_LOB.CLOSE(tlob);
    DBMS_LOB.CLOSE(Src_loc);
    /* Free the temporary LOB now that we are done with it: */
    DBMS_LOB.FREETEMPORARY(tlob);
END;
```

Example: Get the Length of a Temporary LOB Using C (OCI)

```
/* This function takes a temporary LOB locator as an amount as argument and
   prints out the length of the corresponding LOB. The function returns
   0 if it completes successfully, and -1 if it fails.*/
sb4 print_length( OCIError      *errhp,
                  OCISvcCtx     *svchp,
                  OCIStmt       *stmthp,
                  OCIEnv        *envhp)

{
  ub4 length=0;
  ub4 amount = 4;
  ub4 pos = 1;
  OCILobLocator *bfile;
  OCILobLocator *tblob;
  sb4 return_code = 0;

  printf("in print_length\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }
```

```
 if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                      (ub2)strlen("AUDIO_DIR"),
                      (text *)"Washington_audio",
                      (ub2)strlen("Washington_audio")))
 {
    printf("OCILobFileSetName FAILED\n");
    return_code = -1;
 }

 checkerr(errhp,(OCILobFileOpen(svchp, errhp,
                                (OCILobLocator *) bfile,
                                 OCI_LOB_READONLY)));

 /* Create a temporary BLOB: */
 if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                          OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                          OCI_DURATION_SESSION))
 {
     (void) printf("FAILED: CreateTemporary() \n");
     return_code = -1 ;
 }

 if(OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
 {
     (void) printf("FAILED: Open Temporary \n");
     return_code = -1;
 }

if(OCILobLoadFromFile(svchp, errhp, tblob,(OCILobLocator*)bfile,
                      (ub4)amount, (ub4)1,(ub4)1))
 {
     (void) printf("FAILED: Open Temporary \n");
     return_code = -1;
 }

 if (OCILobGetLength(svchp, errhp, tblob,&length))
 {
     printf ("FAILED: OCILobGetLength in print_length\n");
     return_code = -1;
 }

 /* Close the bfile and the temp LOB */
 checkerr(errhp,OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile));
```

```
  checkerr(errhp,OCILobClose(svchp, errhp, (OCILobLocator *) tblob));

  /* Free the temporary LOB now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  fprintf(stderr,"Length of LOB is %d\n",length);
  return return_code;
}
```

Example: Get the Length of a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-LENGTH.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
01  LEN            PIC S9(9) COMP.
01  LEN-D          PIC 9(4).
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-LENGTH.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.
```

```
* Allocate and initialize the BFILE and BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Get the length of the temporary LOB:
     EXEC SQL
          LOB DESCRIBE :TEMP-BLOB GET LENGTH INTO :LEN
     END-EXEC.
     MOVE LEN TO LEN-D.
     DISPLAY "Length of TEMPORARY LOB is ", LEN-D.
* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.
```

```
SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

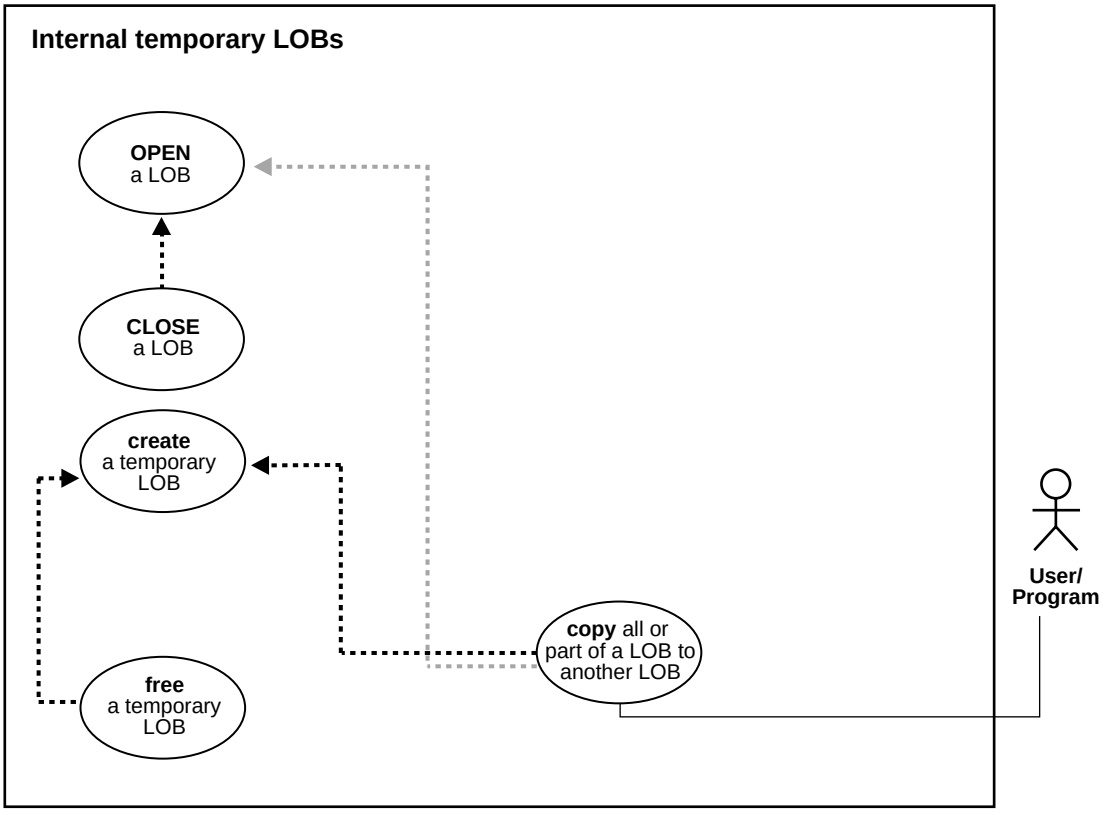
Example: Get the Length of a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void getLengthTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Length, Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
```

```
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Get the length of the Temporary LOB: */
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  /* Note that in this example, Length == Amount == 4096: */
  printf("Length is %d bytes\n", Length);
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Copy All or Part of One (Temporary) LOB to Another

Figure 4–14 Use Case Diagram: Copy All or Part of One (Temporary) LOB to Another



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

Assume the following table:

```
CREATE TABLE VoiceoverLib_tab of VOICED_TYP;
```

Note that this VoiceoverLib_tab is of the same type as the Voiceover_tab which is referenced by the Voiced_ref column of the multimedia table.

```
INSERT INTO Voiceover_tab
    (SELECT * FROM VoiceoverLib_tab Vtab1
       WHERE T2.Take = 101);
```

creates a new LOB locator in the table Voiceover_tab, and copies the LOB data from Vtab1 to the location pointed to by a new LOB locator which is inserted into table Voiceover_tab.

- "Example: Copy All or Part of One (Temporary) LOB to Another Using PL/SQL (DBMS_LOB Package)" on page 4-89
- "Example: Copy All or Part of One (Temporary) LOB to Another Using C (OCI)" on page 4-90
- "Example: Copy All or Part of One (Temporary) LOB to Another Using COBOL (Pro*COBOL)" on page 4-93
- "Example: Copy All or Part of One (Temporary) LOB to Another Using C++ (Pro*C/C++)" on page 4-95

Example: Copy All or Part of One (Temporary) LOB to Another Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure copyTempLOB_proc is not part of the
   DBMS_LOB package.* /
CREATE OR REPLACE PROCEDURE copyTempLOB_proc IS
   Dest_pos     NUMBER;
   Src_pos      NUMBER;
   Dest_loc     BLOB;
   Dest_loc2    BLOB;
   Src_loc      BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount       INTEGER := 32767;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE,DBMS_LOB.SESSION);
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE, DBMS_LOB.SESSION);
   /* Opening the FILE  is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the temporary  LOBs is optional: */
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Dest_loc2,DBMS_LOB.LOB_READWRITE);
```

```
      DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
        /* Set Dest_pos to the position at which we should start writing in the
           target temp LOB */
        /* Copies the LOB from the source position to the destination
           position:*/
        /* Set amount to the amount you want copied */
           Amount := 328;
           Dest_pos := 1000;
           Src_pos := 1000;
        /* Set Src_pos to the position from which we should start copying data
           from  tclob_src: */
           DBMS_LOB.COPY(Dest_loc2,Dest_loc, Amount, Dest_pos, Src_pos);
      COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
      DBMS_LOB.CLOSE(Dest_loc);
      DBMS_LOB.CLOSE(Dest_loc2);
      DBMS_LOB.CLOSE(Src_loc);
      DBMS_LOB.FREETEMPORARY(Dest_loc);
      DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

Example: Copy All or Part of One (Temporary) LOB to Another Using C (OCI)

```
/* This function takes two temporary LOB locators as arguments and copies 4000
   bytes from one temporary LOB to another. It reads the source LOB starting at
   offset 1, and writes to the destination at offset 2. The function returns
   0 if it completes successfully, and -1 otherwise. */
sb4 copy_temp_lobs (OCILobLocator *lob_loc,
                    OCIError      *errhp,
                    OCISvcCtx     *svchp,
                    OCIStmt       *stmthp,
                    OCIEnv        *envhp)
{
  OCIDefine *defnp1;
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
  OCILobLocator *bfile;
  int rowind =1;
  ub4 amount=4000;
  ub4 src_offset=1;
  ub4 dest_offset=2;
  sb4 return_code = 0;
```

```
printf("in copy_temp_lobs \n");

if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob,
                      (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
{
  printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
  return -1;
}

if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&bfile,
                      (ub4)OCI_DTYPE_FILE, (size_t)0, (dvoid**)0))
{
  printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
  return -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if(OCIDescriptorAlloc((dvoid*)envhp, (dvoid **)&tblob2,
                      (ub4)OCI_DTYPE_LOB, (size_t)0, (dvoid**)0))
{
  printf("OCIDescriptorAlloc failed in copy_temp_lobs\n");
  return_code =  -1;
}

if(OCILobCreateTemporary(svchp, errhp, tblob2, (ub2)0,  SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                     (ub2)strlen("AUDIO_DIR"),
                     (text *)"Washington_audio",
                     (ub2)strlen("Washington_audio")))
{
  printf("OCILobFileSetName FAILED\n");
```

```
  return_code = -1;
 }

 if(OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
 {
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
 }

 if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
 {
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
 }
 if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE ))
 {
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
 }

 if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                       (ub4)amount, (ub4)1,(ub4)1))
 {
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;
 }

if (OCILobCopy(svchp, errhp, tblob2, tblob,  amount, dest_offset,
              src_offset))
 {
  printf ("FAILED: OCILobCopy in copy_temp_lobs\n");
  return -1;
 }
 /* Close LOBs here */

 if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
 {
  printf( "OCILobFileClose FAILED for bfile \n");
  return_code =  -1;
 }
 if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
 {
  printf( "OCILobClose FAILED for temporary LOB \n");
  return_code =  -1;
 }
```

```
  if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob2))
  {
    printf( "OCILobClose FAILED for temporary LOB \n");
    return_code =  -1;
  }
  /* free the temporary lobs now that we are done using them */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  if(OCILobFreeTemporary(svchp, errhp, tblob2))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  return return_code;
}
```

Example: Copy All or Part of One (Temporary) LOB to Another Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. TEMP-BLOB-COPY.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID   PIC X(11) VALUES "USER1/USER1".

     01  TEMP-DEST      SQL-BLOB.
     01  TEMP-SRC       SQL-BLOB.
     01  SRC-BFILE      SQL-BFILE.
     01  DIR-ALIAS      PIC X(30) VARYING.
     01  FNAME          PIC X(30) VARYING.
     01  AMT            PIC S9(9) COMP.

    * Define the source and destination position and location:
     01  SRC-POS        PIC S9(9) COMP VALUE 1.
     01  DEST-POS       PIC S9(9) COMP VALUE 1.

     01 ORASLNRD        PIC 9(4).

         EXEC SQL INCLUDE SQLCA END-EXEC.
```

```
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 TEMP-BLOB-COPY.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
    EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
    EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-DEST
    END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-SRC
    END-EXEC.

* Set up the directory and file information:
    MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "washington_audio" TO FNAME-ARR.
    MOVE 16 TO FNAME-LEN.

    EXEC SQL
       LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
       FILENAME = :FNAME
    END-EXEC.

* Open source BFILE and destination temporary BLOB:
    EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
    EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
    EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* MOVE the desired amount to copy to AMT:
    MOVE 5 TO AMT.
    EXEC SQL
         LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
    END-EXEC.

* Copy data from BFILE to temporary LOB:
```

```
           EXEC SQL
                LOB COPY :AMT FROM :TEMP-SRC AT :SRC-POS
                TO :TEMP-DEST AT :DEST-POS
           END-EXEC.

           EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
           EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
           EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
           EXEC SQL
                LOB FREE TEMPORARY :TEMP-SRC
           END-EXEC.
           EXEC SQL
                LOB FREE TEMPORARY :TEMP-DEST
           END-EXEC.
           EXEC SQL FREE :TEMP-SRC END-EXEC.
           EXEC SQL FREE :TEMP-DEST END-EXEC.
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

Example: Copy All or Part of One (Temporary) LOB to Another Using C++ (Pro*C/C++)

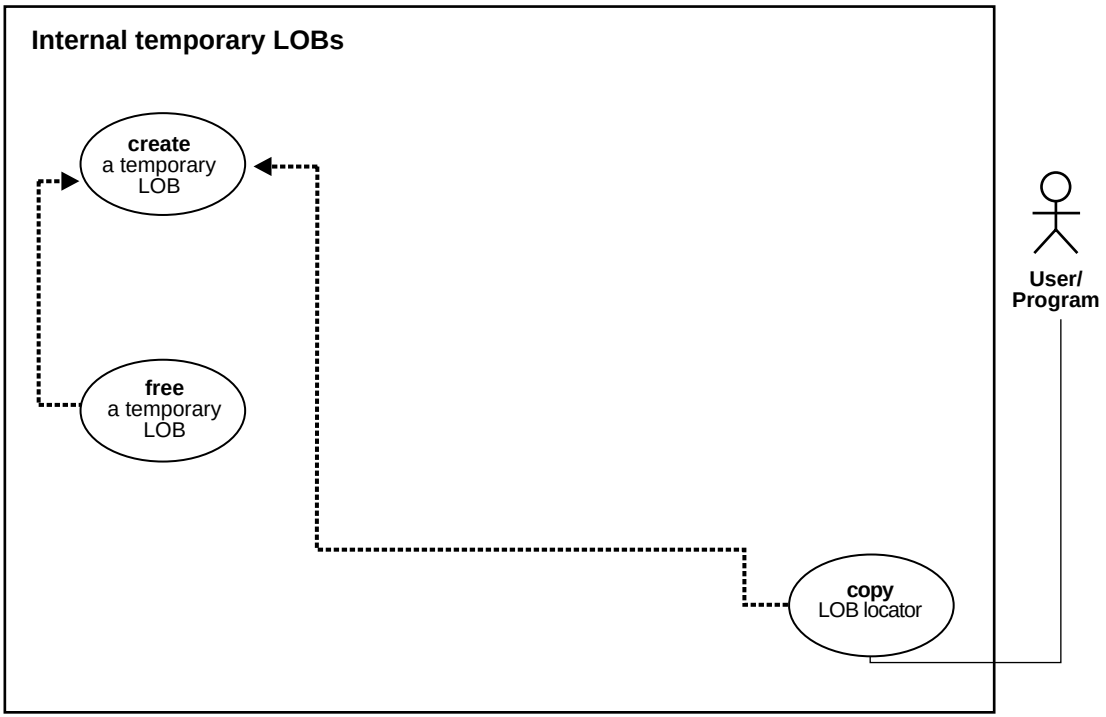
```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void copyTempLOB_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
  /* Load a specified amount from the BFILE into one of the
     Temporary LOBs: */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Copy a specified amount from one Temporary LOB to another: */
  EXEC SQL LOB COPY :Amount FROM :Temp_loc1 TO :Temp_loc2;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
```

```
  EXEC SQL CONNECT :samp;
  copyTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Copy a LOB Locator for a Temporary LOB

Figure 4–15 Use Case Diagram: Copy a LOB Locator for a Temporary LOB...208



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This generic operation copies one temporary LOB locator to another.

- "Example: Copy a LOB Locator (Temporary LOBs) Using PL/SQL" on page 4-99
- "Example: Copy a LOB Locator for a Temporary LOB Using C (OCI)" on page 4-100

- "Example: Copy a LOB Locator for a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-102
- "Example: Copy a LOB Locator for a Temporary LOB Using C++ (Pro*C/C++)" on page 4-104

Example: Copy a LOB Locator (Temporary LOBs) Using PL/SQL

> **Note:** Assigning one LOB to another using PL/SQL entails using the "=" sign. This is an advanced topic that is discussed in more detail above with regard to "Read-Consistent Locators" on page 2-2.

```
/* Note that the example procedure copyTempLOBLocator_proc is not part of the
   DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE copyTempLOBLocator_proc(
   Lob_loc1 IN OUT CLOB, Lob_loc2 IN OUT CLOB) IS

   bufp     VARCHAR2(4);
   Amount   NUMBER  := 32767;
   Src_loc  BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc1,TRUE,DBMS_LOB.SESSION);
    DBMS_LOB.CREATETEMPORARY(Lob_loc2,TRUE,DBMS_LOB.SESSION);
    /* Populate the first temporary LOB with some data. */
    /* Opening file is mandatory: */
    DBMS_LOB.OPEN(Src_loc,DBMS_LOB.LOB_READONLY);
    /* Opening LOB is optional: */
    DBMS_LOB.OPEN(Lob_loc1,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.OPEN(Lob_loc2,DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(Lob_loc1,Src_loc,Amount);

   /* Assign Lob_loc1 to Lob_loc2 thereby creating  a copy of the value of
      the temporary LOB referenced by Lob_loc1 at this point in time: */
   Lob_loc2 := Lob_loc1;

   /* When you write some data to the LOB through Lob_loc1, Lob_loc2
      will not see the newly written data whereas Lob_loc1 will see
      the new data: */
   /*Closing LOBs is mandatory if they were opened: */
    DBMS_LOB.CLOSE (Src_loc);
    DBMS_LOB.CLOSE (Lob_loc1);
```

```
    DBMS_LOB.CLOSE (Lob_loc2);
    DBMS_LOB.FREETEMPORARY(Lob_loc1);
    DBMS_LOB.FREETEMPORARY(Lob_loc2);
END;
```

Example: Copy a LOB Locator for a Temporary LOB Using C (OCI)

```
/* This function creates two temporary lobs. It populates one and
   then copies the locator of that one to the other temporary
   LOB locator: */

sb4 copy_locators( OCIError    *errhp,
                   OCISvcCtx   *svchp,
                   OCIEnv      *envhp)
{
  sb4 return_code = 0;
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
  OCILobLocator *bfile;
  ub4 amount = 4000;

 checkerr(errhp,OCIDescriptorAlloc((dvoid*)envhp,(dvoid**)&tblob,
                                   (ub4) OCI_DTYPE_LOB,
                                   (size_t) 0, (dvoid **) 0));

 checkerr(errhp,OCIDescriptorAlloc((dvoid*)envhp,(dvoid**)&tblob2,
                                   (ub4) OCI_DTYPE_LOB,
                                   (size_t) 0, (dvoid **) 0));

 checkerr(errhp,OCIDescriptorAlloc((dvoid*)envhp,(dvoid**)&bfile,
                                   (ub4) OCI_DTYPE_FILE,
                                   (size_t) 0, (dvoid **) 0));

  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"),
                       (text *)"Washington_audio",
                       (ub2)strlen("Washington_audio")))
  {
    printf("OCILobFileSetName FAILED in load_temp\n");
    return -1;
  }

  if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_FILE_READONLY))
  {
```

```
  printf( "OCILobFileOpen FAILED for the bfile load_temp \n");
  return -1;
}


if(OCILobCreateTemporary(svchp,errhp, tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
  (void) printf("FAILED: CreateTemporary() \n");
  return -1;
}

if(OCILobCreateTemporary(svchp,errhp, tblob2,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
  (void) printf("FAILED: CreateTemporary() \n");
  return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return -1;
}

if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amount, (ub4)1,(ub4)1))
{
  printf("OCILobLoadFromFile failed \n");
  return_code = -1;
}

if(OCILobLocatorAssign(svchp,errhp, (CONST OCILobLocator *)tblob,&tblob2))
{

  printf("OCILobLocatorAssign failed \n");
  return_code = -1;
```

```
  }

  /* Close the lobs */
  if (OCILobFileClose(svchp, errhp, (OCILobLocator *) bfile))
  {
    printf( "OCILobClose FAILED for bfile \n");
    return -1;
  }

  checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob)));
  checkerr(errhp,(OCILobClose(svchp, errhp, (OCILobLocator *) tblob2)));


  /* Free the temporary lobs now that we are done using it */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

  if(OCILobFreeTemporary(svchp, errhp, tblob2))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }
}
```

Example: Copy a LOB Locator for a Temporary LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. TEMP-BLOB-COPY-LOCATOR.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  USERID   PIC X(11) VALUES "USER1/USER1".

     01  TEMP-DEST      SQL-BLOB.
     01  TEMP-SRC       SQL-BLOB.
     01  SRC-BFILE      SQL-BFILE.
     01  DIR-ALIAS      PIC X(30) VARYING.
     01  FNAME          PIC X(30) VARYING.
     01  AMT            PIC S9(9) COMP.
```

```
01 ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-BLOB-COPY-LOCATOR.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :TEMP-DEST END-EXEC.
    EXEC SQL ALLOCATE :TEMP-SRC END-EXEC.
    EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-DEST
    END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-SRC
    END-EXEC.

* Set up the directory and file information:
    MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "washington_audio" TO FNAME-ARR.
    MOVE 16 TO FNAME-LEN.

    EXEC SQL
       LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
       FILENAME = :FNAME
    END-EXEC.

* Open source BFILE and destination temporary BLOB:
    EXEC SQL LOB OPEN :TEMP-SRC READ WRITE END-EXEC.
    EXEC SQL LOB OPEN :TEMP-DEST READ WRITE END-EXEC.
    EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* MOVE the desired amount to copy to AMT:
    MOVE 5 TO AMT.
    EXEC SQL
         LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-SRC
```

```
            END-EXEC.

        * Assign source BLOB locator to destination BLOB locator:
            EXEC SQL
                 LOB ASSIGN :TEMP-SRC TO :TEMP-DEST
            END-EXEC.

            EXEC SQL LOB CLOSE :TEMP-SRC END-EXEC.
            EXEC SQL LOB CLOSE :TEMP-DEST END-EXEC.
            EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
            EXEC SQL
                 LOB FREE TEMPORARY :TEMP-SRC
            END-EXEC.
            EXEC SQL
                 LOB FREE TEMPORARY :TEMP-DEST
            END-EXEC.
            EXEC SQL FREE :TEMP-SRC END-EXEC.
            EXEC SQL FREE :TEMP-DEST END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

         SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Copy a LOB Locator for a Temporary LOB Using C++ (Pro*C/C++)

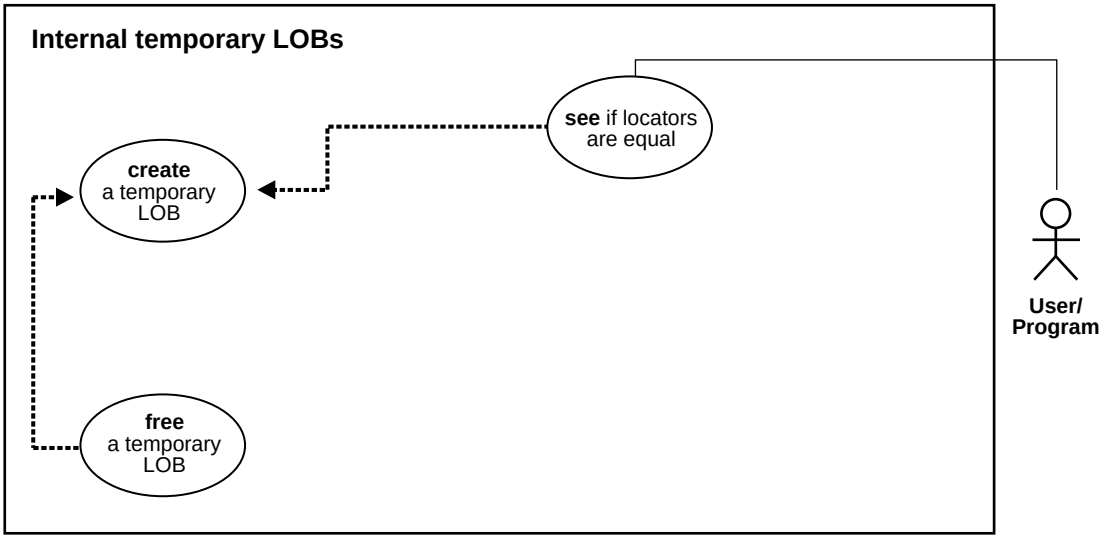
```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
```

```
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void copyTempLobLocator_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Assign Temp_loc1 to Temp_loc2 thereby creating a copy of the value of
     the Temporary LOB referenced by Temp_loc1 at this point in time: */
  EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;
  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
}

void main()
{
```

```
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  copyTempLobLocator_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

See If One LOB Locator for a Temporary LOB Is Equal to Another

Figure 4–16 Use Case Diagram: See If One (Temporary) LOB Locator Is Equal to Another



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

If two locators are equal, this means that they refer to the same version of the LOB data (see "Read-Consistent Locators" on page 2-2)

- "Example: See If One LOB Locator for a Temporary LOB Is Equal to Another Using C (OCI)" on page 4-108
- "Example: See If One LOB Locator for a Temporary LOB Is Equal to Another Using C++ (Pro*C/C++)" on page 4-109

Example: See If One LOB Locator for a Temporary LOB Is Equal to Another Using C (OCI)

```
sb4 ck_isequal (OCILobLocator *lob_loc,
                OCIError      *errhp,
                OCISvcCtx     *svchp,
                OCIStmt       *stmthp,
                OCIEnv        *envhp)
{
  OCILobLocator *loc1;f
  OCILobLocator *loc2;
  boolean is_equal;
  is_equal= FALSE;
  if(OCILobCreateTemporary(svchp, errhp, loc1, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }
   if(OCILobCreateTemporary(svchp, errhp, loc2, (ub2)0, SQLCS_IMPLICIT,
                            OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                            OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  if (OCILobIsEqual(envhp,loc1,loc2, &is_equal))
  {
    printf ("FAILED: OCILobLocatorIsEqual call\n");
    return -1;
  }
  if(is_equal)
  {
      fprintf (stderr,"LOB loators are equal \n");
      return -1;

   }else
   {
      fprintf(stderr,"LOB locators are not equal \n");
   }
   if(OCILobFreeTemporary(svchp,errhp,loc1))
   {
        printf("FAILED: OCILobFreeTemporary for temp LOB #1\n");
```

```
        return -1;
    }
  if(OCILobFreeTemporary(svchp,errhp,loc2))
    {
        printf("FAILED: OCILobFreeTemporary for temp LOB #2\n");
        return -1;
    }

    return 0;
}
```

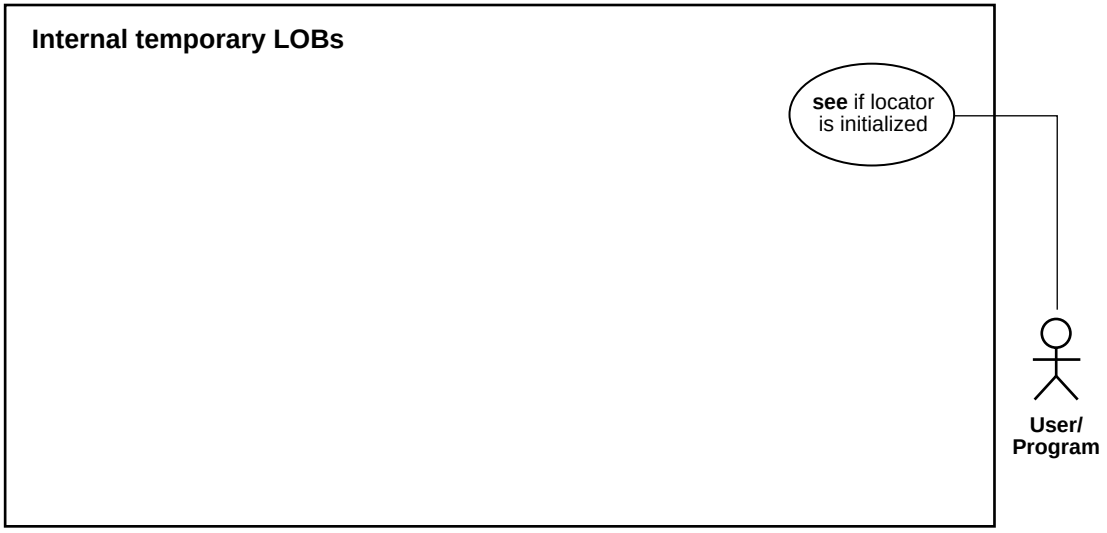
Example: See If One LOB Locator for a Temporary LOB Is Equal to Another Using C++ (Pro*C/C++)

```
#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("sqlcode = %ld\n", sqlca.sqlcode);
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void seeTempLobLocatorsAreEqual_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 4096;
  OCIEnv *oeh;
  int isEqual = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
```

```
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
  /* Load a specified amount from the BFILE into one of the Temporary LOBs: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc1;
  /* Retrieve the OCI Environment Handle: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Now assign Temp_loc1 to Temp_loc2 using Embedded SQL: */
  EXEC SQL LOB ASSIGN :Temp_loc1 TO :Temp_loc2;
  /* Determine if the Temporary LOBs are Equal: */
  (void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);
  /* This time, isEqual should be 0 (FALSE): */
  printf("Locators %s equal\n", isEqual ? "are" : "are not");
  /* Assign Temp_loc1 to Temp_loc2 using C pointer assignment: */
  Temp_loc2 = Temp_loc1;
  /* Determine if the Temporary LOBs are Equal again: */
  (void) OCILobIsEqual(oeh, Temp_loc1, Temp_loc2, &isEqual);
  /* The value of isEqual should be 1 (TRUE) in this case: */
  printf("Locators %s equal\n", isEqual ? "are" : "are not");
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  /* Note that because Temp_loc1 and Temp_loc2 are now equal, closing
     and freeing one will implicitely do the same to the other: */
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeTempLobLocatorsAreEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

See If a LOB Locator for a Temporary LOB Is Initialized

Figure 4–17 Use Case Diagram: See If a LOB Locator for a Temporary LOB Is Initialized



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This generic function takes a LOB locator and checks if it is initialized. If it is initialized, then it prints out a message saying "LOB is initialized". Otherwise, it reports "LOB is not initialized".

- "Example: See If a LOB Locator for a Temporary LOB Is Initialized Using C (OCI)" on page 4-112
- "Example: See If a LOB Locator for a Temporary LOB Is Initialized Using C++ (Pro*C/C++)" on page 4-112

Example: See If a LOB Locator for a Temporary LOB Is Initialized Using C (OCI)

```
/* This function takes a LOB locator and checks if it is initialized. If it is
   initalized, then it prints out a message saying "LOB is initialized".
   Otherwise, it says "LOB is not initialized". This function returns
   0 if it completes successfully, and -1 if it doesn't. */

sb4 ck_isinit (OCILobLocator *lob_loc,
               OCIError      *errhp,
               OCISvcCtx     *svchp,
               OCIStmt       *stmthp,
               OCIEnv        *envhp)

{
  boolean is_init;

  is_init= FALSE;
  if (OCILobLocatorIsInit(envhp,errhp, lob_loc, &is_init))
  {
    printf ("FAILED: OCILobLocatorIsInit call\n");
    return -1;
  }
  if(is_init)
  {
      printf ("LOB is initialized\n");

  }else
  {
      printf("LOB is not initialized\n");
   }
   return 0;
}
```

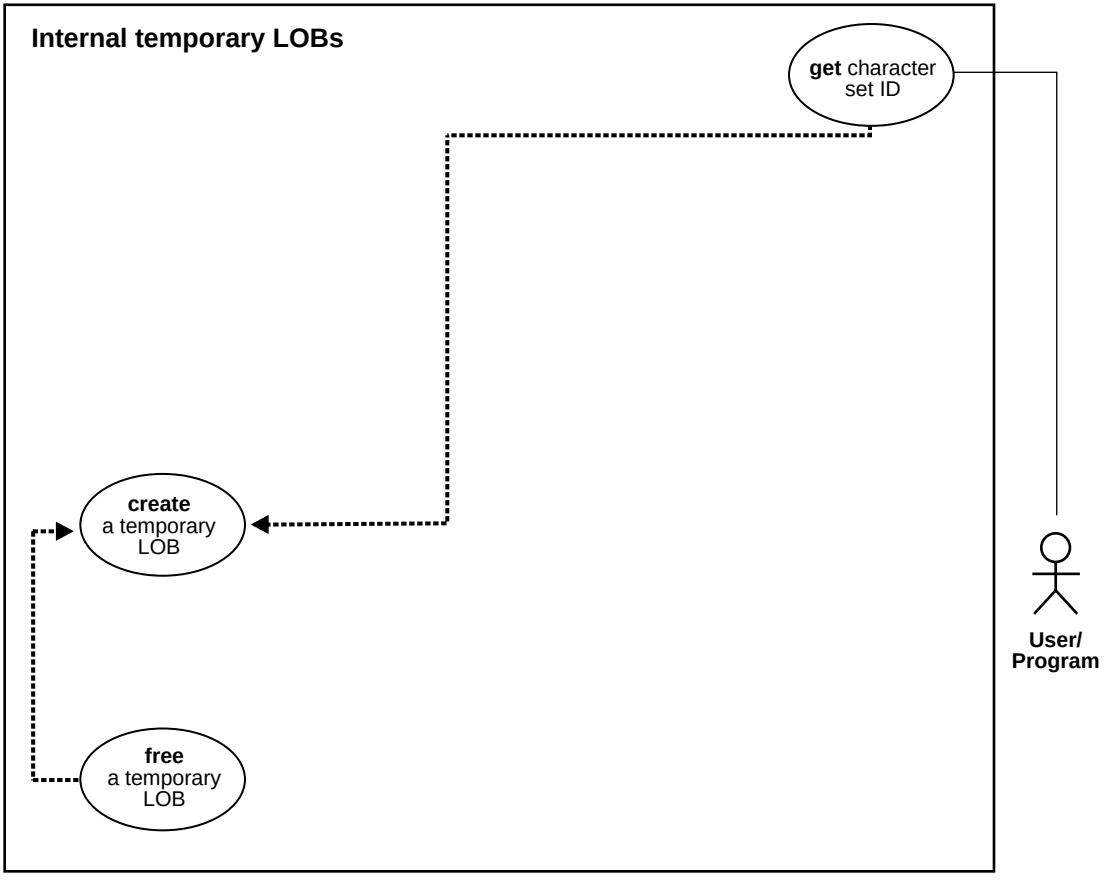
Example: See If a LOB Locator for a Temporary LOB Is Initialized Using C++ (Pro*C/C++)

```
#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}
void tempLobLocatorIsInit_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized */
  (void) OCILobLocatorIsInit(oeh, err, Temp_loc, &isInitialized);
  if (isInitialized)
    printf("Locator is initialized\n");
  else
    printf("Locator is not initialized\n");
  /* Note that in this example, the locator is initialized. */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  tempLobLocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Get Character Set ID of a Temporary LOB

Figure 4–18 Use Case Diagram: Get Character Set ID for a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This function takes a LOB locator and prints the character set id of the LOB.

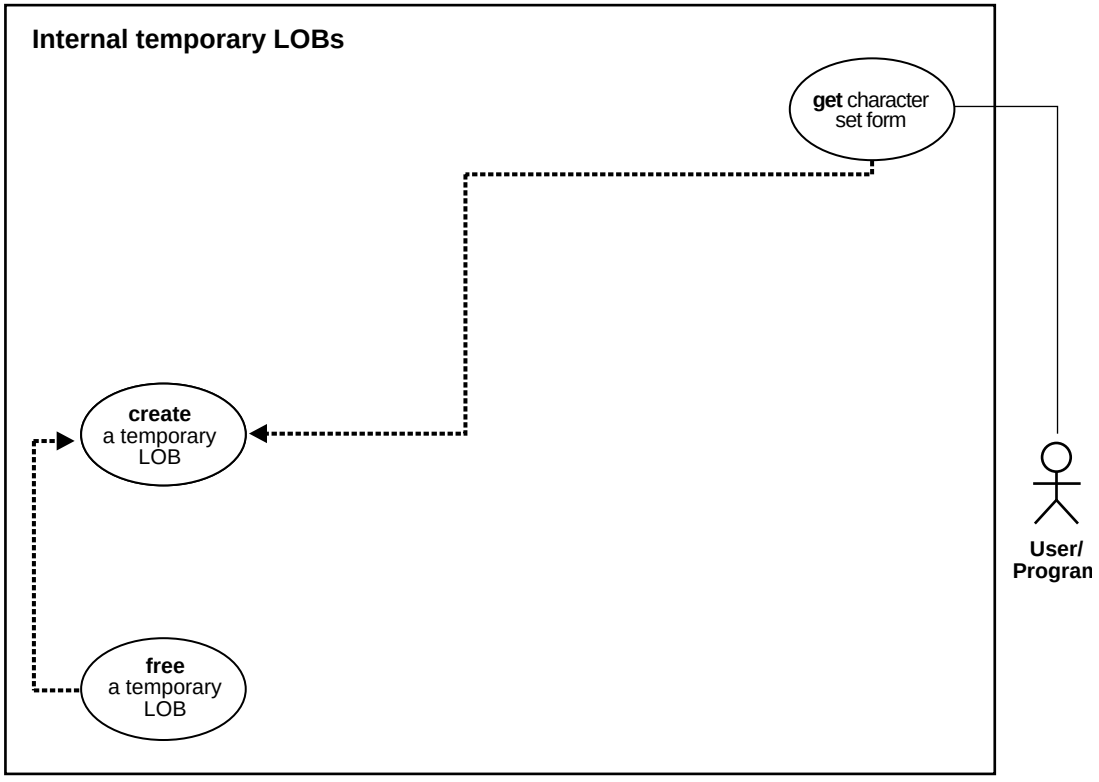
- "Example: Get Character Set ID of a Temporary LOB Using C (OCI)" on page 4-115

Example: Get Character Set ID of a Temporary LOB Using C (OCI)

```
/* This function takes a LOB locator and prints the character set id of the LOB.
   This function returns 0 if it completes successfully, and -1
   if it doesn't. */

sb4 get_charsetid (OCILobLocator *lob_loc,
                   OCIError      *errhp,
                   OCISvcCtx     *svchp,
                   OCIStmt       *stmthp,
                   OCIEnv        *envhp)
{
  ub2 charsetid=199;
  if(OCILobCreateTemporary(svchp, errhp, lob_loc, (ub2)0,  SQLCS_IMPLICIT,
                           OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  if (OCILobCharSetId(envhp, errhp, lob_loc, &charsetid))
  {
    printf ("FAILED: OCILobCharSetId call\n");
    return -1;
  }
  fprintf (stderr,"LOB charsetid is %d\n",charsetid);
  if(OCILobFreeTemporary(svchp,errhp,lob_loc))
  {
        printf("FAILED: OCILobFreeTemporary \n");
        return -1;
  }

  return 0;

}
```

Get Character Set Form of a Temporary LOB

Figure 4–19 Use Case Diagram: Get Character Set Form of a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This function takes a LOB locator and prints the character set form for the LOB.

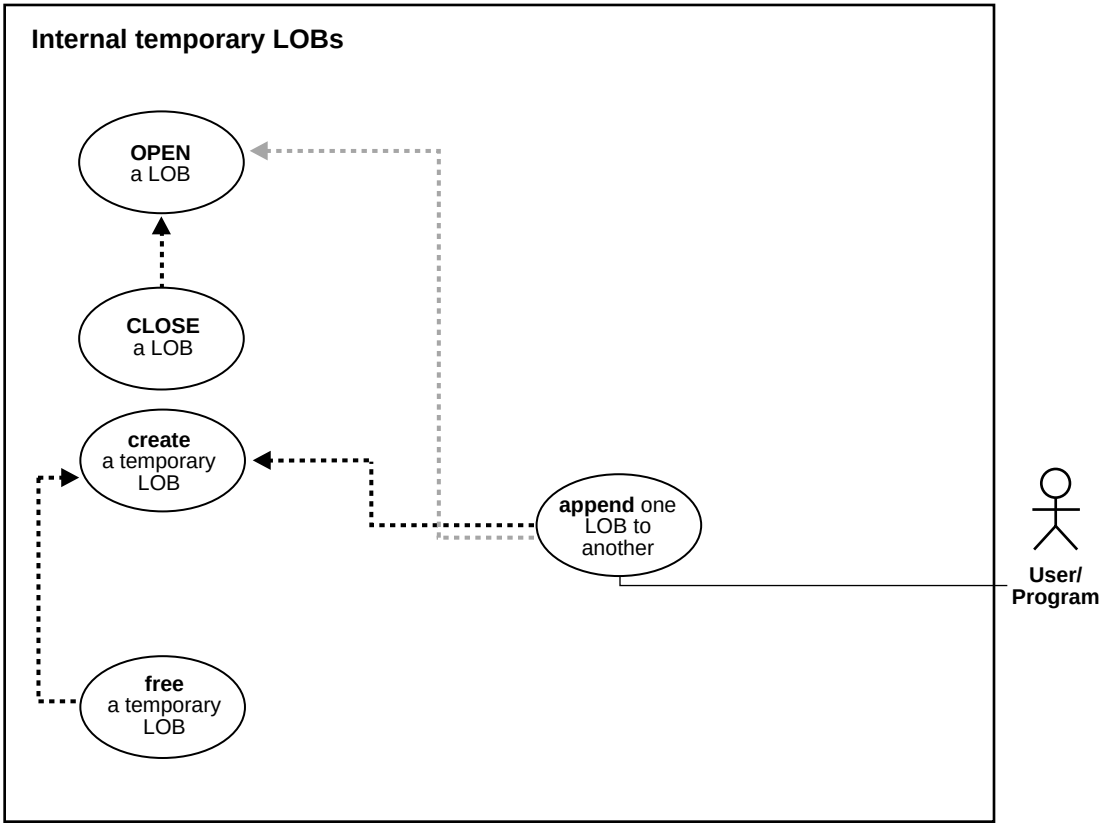
- "Example: Get Character Set Form of a Temporary LOB Using C (OCI)" on page 4-117

Example: Get Character Set Form of a Temporary LOB Using C (OCI)

```
/* This function takes a LOB locator and prints out the character set form for
the LOB. It returns 0 if it completes successfully, and it returns -1 if it
doesn't. */

sb4 get_charsetform (OCILobLocator *lob_loc,
                     OCIError      *errhp,
                     OCISvcCtx     *svchp,
                     OCIStmt       *stmthp,
                     OCIEnv        *envhp)
{
  ub1 charsetform =0;
  if(OCILobCreateTemporary(svchp,errhp,lob_loc,(ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  if (OCILobCharSetForm(envhp,errhp, lob_loc, &charsetform))
  {
    printf ("FAILED: OCILobCharSetForm call\n");
    return -1;
  }
  fprintf (stderr,"LOB charsetform is %d\n",charsetform);

  if(OCILobFreeTemporary(svchp,errhp,lob_loc))
  {
        printf("FAILED: OCILobFreeTemporary \n");
        return -1;
  }
  return 0;
}
```

Append One (Temporary) LOB to Another

Figure 4–20 Use Case Diagram: Append one (Temporary) LOB to another



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This example deals with the task of appending one segment of sound to another. We assume that you use sound-specific editing tools to match the wave-forms.

- "Example: Append One (Temporary) LOB to Another Using PL/SQL (DBMS_LOB Package)" on page 4-119
- "Example: Append One (Temporary) LOB to Another Using C (OCI)" on page 4-120
- "Example: Append One (Temporary) LOB to Another Using COBOL (Pro*COBOL)"
- "Example: Append One (Temporary) LOB to Another Using C++ (Pro*C/C++)"

Example: Append One (Temporary) LOB to Another Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure appendTempLOB_proc is not part of the
   DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE appendTempLOB_proc IS
   Dest_loc2 CLOB;
   Dest_loc  CLOB;
   Amount    NUMBER;
   Src_loc   BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
BEGIN
   DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE,DBMS_LOB.SESSION);
   DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE,DBMS_LOB.SESSION);
   DBMS_LOB.OPEN(Dest_loc,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Dest_loc2,DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.OPEN(Src_loc,DBMS_LOB.LOB_READWRITE);
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   DBMS_LOB.LOADFROMFILE(Dest_loc2, Src_loc, Amount);
   DBMS_LOB.APPEND(Dest_loc, Dest_loc2);
   /* Close the temporary lobs and then free them: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Dest_loc2);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.FREETEMPORARY(Dest_loc);
   DBMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

Example: Append One (Temporary) LOB to Another Using C (OCI)

```
/* This function takes two temporary LOB locators and appends the second LOB to
   the first one. It returns 0 if it completes successfully, and
   -1, otherwise.*/

sb4 append_temp_lobs (OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *tblob2;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  sb4 return_code = 0;

  printf("in append \n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob2,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in print_length\n");
     return -1;
  }

  /* Set the BFILE to point to the Washington_audio file */
  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"),
                       (text *)"Washington_audio",
```

```
                    (ub2)strlen("Washington_audio")))
{
  printf("OCILobFileSetName FAILED\n");
  return -1;
}

if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob2,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_CLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return_code = -1;
}

/* Open the lobs: */
if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB tblob \n");
  return_code =  -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob2, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB, tblob2 \n");
  return_code =  -1;
}

/* Populate the source temporary LOB with some data: */

If(OCILobLoadFromFile(svchp, errhp, tblob,(OCILobLocator*)bfile,
                    (ub4)amt, (ub4)1,(ub4)1))
```

```
  {
    printf( "OCILobLoadFromFile FAILED\n");
    return_code = -1;
  }

  /* Append the source LOB to the dest temp LOB: */
  if (OCILobAppend(svchp, errhp,tblob2,tblob))
  {
    printf ("FAILED: OCILobAppend in append_temp_lobs\n");
    return_code = -1;
  }else
  {
    printf("Append succeeded\n");
  }

  if(OCILobFreeTemporary(svchp,errhp,tblob))
  {
        printf("FAILED: OCILobFreeTemporary \n");
        return_code = -1;
  }
 if(OCILobFreeTemporary(svchp,errhp,tblob2))
 {
      printf("FAILED: OCIlobFreeTemporary\n");
      return_code = -1;
 }
 return return_code;
}
```

Example: Append One (Temporary) LOB to Another Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. APPEND-TEMP-BLOB.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

      * Define the username and password:
       01  USERID   PIC X(11) VALUES "USER1/USER1".

      * Define the temporary LOBs and the source BFILE:
       01  TEMP-BLOB1     SQL-BLOB.
       01  TEMP-BLOB2     SQL-BLOB.
       01  SRC-BFILE      SQL-BFILE.
       01  AMT            PIC S9(9) COMP.
```

```
 01  DIR-ALIAS      PIC X(30) VARYING.
 01  FNAME          PIC X(30) VARYING.

* Define the source position in BFILE:
 01  SRC-POS        PIC S9(9) COMP.

* Define the line number in case of error:
 01  ORASLNRD       PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 APPEND-TEMP-BLOB.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB1 END-EXEC.
     EXEC SQL ALLOCATE :TEMP-BLOB2 END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB1
     END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB2
     END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB2 READ WRITE END-EXEC.
```

```
     EXEC SQL LOB OPEN :TEMP-BLOB1 READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     DISPLAY "LOBs opened.".

* Move the desired amount to copy to AMT:
     MOVE 5 TO AMT.
     MOVE 1 TO SRC-POS.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE
          AT :SRC-POS INTO :TEMP-BLOB1
     END-EXEC.

     ADD 1 TO AMT GIVING SRC-POS.
     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE
          AT :SRC-POS INTO :TEMP-BLOB2
     END-EXEC.
     DISPLAY "Temporary LOBs loaded".

     EXEC SQL
          LOB APPEND :TEMP-BLOB2 TO :TEMP-BLOB1
     END-EXEC.
     DISPLAY "LOB APPEND complete.".

     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB1
     END-EXEC.
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB2
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB1 END-EXEC.
     EXEC SQL FREE :TEMP-BLOB2 END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
```

```
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

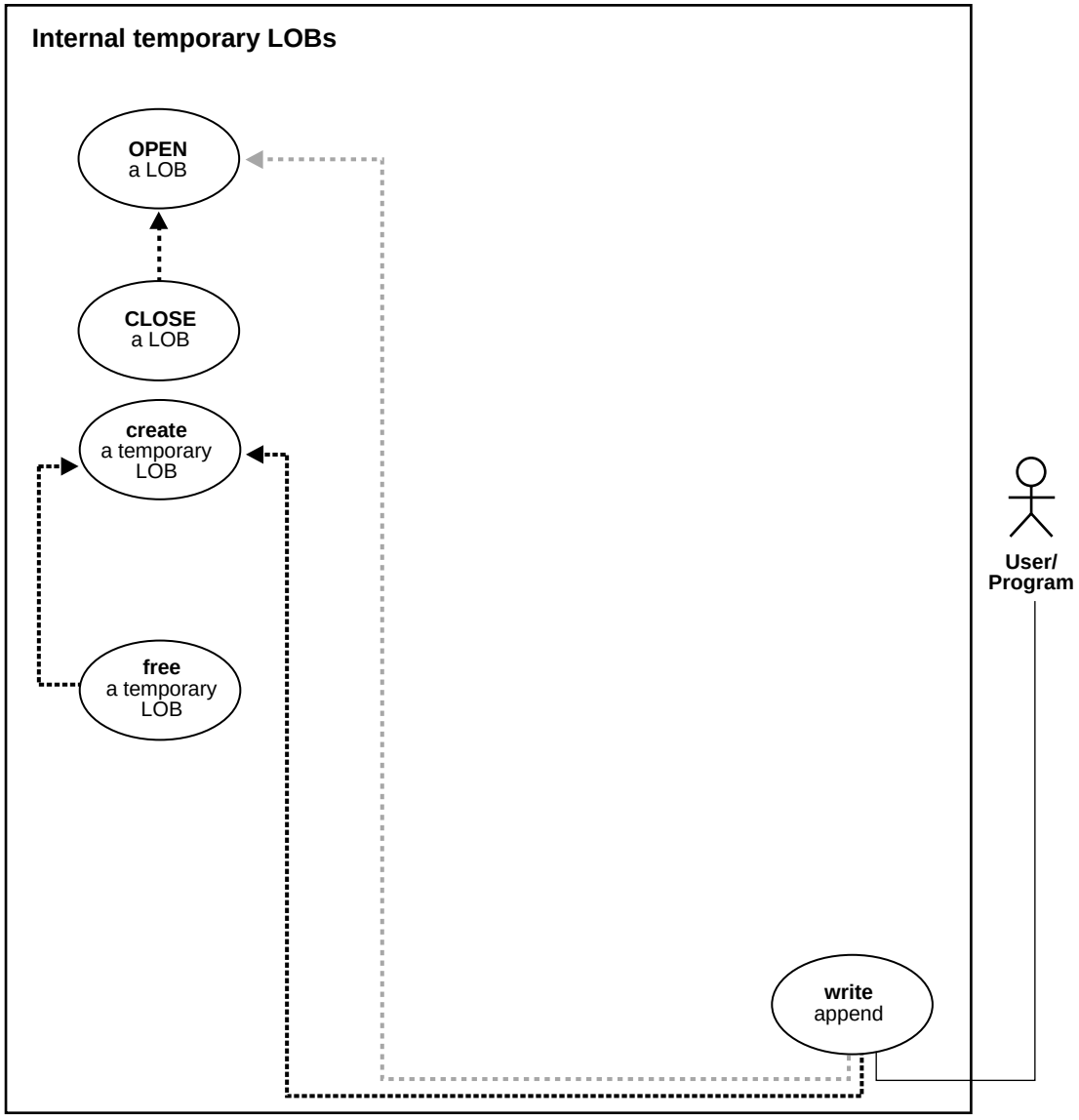
Example: Append One (Temporary) LOB to Another Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void appendTempLOB_proc()
{
  OCIBlobLocator *Temp_loc1, *Temp_loc2;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 2048;
  int Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOBs: */
  EXEC SQL ALLOCATE :Temp_loc1;
  EXEC SQL ALLOCATE :Temp_loc2;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc1;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
  EXEC SQL LOB OPEN :Temp_loc2 READ WRITE;
  /* Load a specified amount from the BFILE into the first Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc1;
  /* Set the Position for the next load from the same BFILE: */
  Position = Amount + 1;
  /* Load a second amount from the BFILE into the second Temporary LOB: */
```

```
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc AT :Position INTO :Temp_loc2;
  /* Append the second Temporary LOB to the end of the first one: */
  EXEC SQL LOB APPEND :Temp_loc2 TO :Temp_loc1;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc1;
  EXEC SQL LOB CLOSE :Temp_loc2;
  /* Free the Temporary LOBs: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc1;
  EXEC SQL LOB FREE TEMPORARY :Temp_loc2;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc1;
  EXEC SQL FREE :Temp_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  appendTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Write Append to a Temporary LOB

Figure 4–21 Use Case Diagram: Write Append to a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

This example procedure will read in 32767 bytes of data from the `Washington_audio` file starting at offset 128 and append it to a temporary LOB.

- "Example: Write Append to a Temporary LOB Using PL/SQL" on page 4-128
- "Example: Write Append to a Temporary LOB Using C (OCI)" on page 4-129
- "Example: Write Append to a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-130
- "Example: Write Append to a Temporary LOB Using C++ (Pro*C/C++)" on page 4-132

Example: Write Append to a Temporary LOB Using PL/SQL

```
/* Note that the example procedure writeAppendTempLOB_proc is not part of the
   DBMS_LOB package. This example procedure will read in 32767 bytes of  data
   from the Washington_audio file starting at offset 128 and append it to a
   temporary LOB. */

CREATE OR REPLACE PROCEDURE writeAppendTempLOB_proc IS
   Lob_loc    BLOB;
   Buffer     RAW(32767);
   Src_loc    BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount     Binary_integer := 32767;
   Position   Binary_integer := 128;
BEGIN
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* Opening the temporary LOB is optional: */
   DBMS_LOB.OPEN(Lob_loc,DBMS_LOB.LOB_READWRITE);
   /* Opening the FILE is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Fill the buffer with data: */
   DBMS_LOB.LOADFROMFILE (Lob_loc,Src_loc, Amount);

   /* Append the data from the buffer to the end of the LOB: */
   DBMS_LOB.WRITEAPPEND(Lob_loc, Amount, Buffer);
   DBMS_LOB.CLOSE(Src_loc);
```

```
   DBMS_LOB.CLOSE(Lob_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

Example: Write Append to a Temporary LOB Using C (OCI)

```
sb4 write_append_temp_lobs (OCIError      *errhp,
                            OCISvcCtx     *svchp,
                            OCIStmt       *stmthp,
                            OCIEnv        *envhp)
{
  OCIClobLocator *tclob;
  unsigned int Total = 40000;
  unsigned int amtp;
  unsigned int  nbytes;
  ub1 bufp[MAXBUFLEN];

  /* Allocate the locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  if(OCILobCreateTemporary(svchp, errhp, tclob, (ub2)0,  SQLCS_IMPLICIT,
          OCI_TEMP_CLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
     return -1;
  }

  /* Open the CLOB */
  printf("calling open \n");
  checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

  nbytes = MAXBUFLEN;   /* We will use Streaming via Standard Polling */

  /* Fill the Buffer with nbytes worth of Data */
  memset(bufp,'a',32767);

  amtp = sizeof(bufp);
  /* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE */

  printf("calling write append \n");
  checkerr (errhp, OCILobWriteAppend (svchp, errhp, tclob, &amtp,
                                      bufp, nbytes, OCI_ONE_PIECE, (dvoid *)0,
                                      (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
```

```
                                        0, SQLCS_IMPLICIT));

  printf("calling close \n");
  /* Closing the LOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tclob));

  /* Free the temporary LOB: */
  printf("calling free\n");
  checkerr(errhp,OCILobFreeTemporary(svchp,errhp,tclob));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);
}
```

Example: Write Append to a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-APPEND-TEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  BUFFER         PIC X(2048).
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
    EXEC SQL VAR BUFFER IS RAW(2048) END-EXEC.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
WRITE-APPEND-TEMP.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
```

```
     END-EXEC.

* Allocate and initialize the BFILE and BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

  MOVE "262626" TO BUFFER.
  MOVE 3 TO AMT.
* Append the data in BUFFER to TEMP-BLOB:
     EXEC SQL
          LOB WRITE APPEND :AMT FROM :BUFFER INTO :TEMP-BLOB
     END-EXEC.

* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

* And free the LOB locators:
```

```
            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

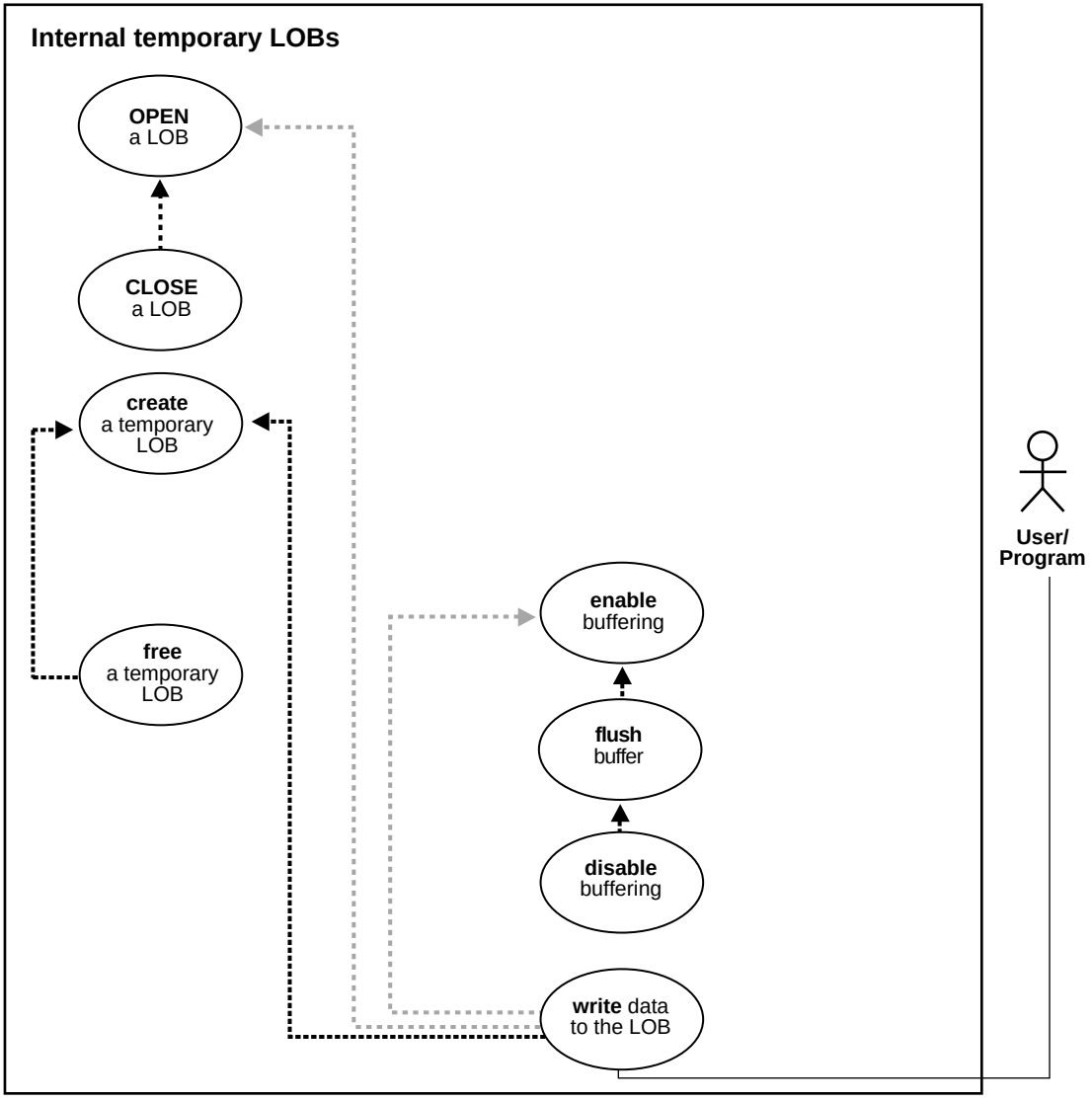
Example: Write Append to a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 256

void writeAppendTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount;
  struct {
    unsigned short Length;
    char Data[BufferLength];
  } Buffer;
```

```
  EXEC SQL VAR Buffer IS VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB: */
  Amount = 2048;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  strcpy((char *)Buffer.Data, "afafafafafaf");
  Buffer.Length = 6;
  /* Write the contents of the Buffer to the end of the Temporary LOB: */
  Amount = Buffer.Length;
  EXEC SQL LOB WRITE APPEND :Amount FROM :Buffer INTO :Temp_loc;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeAppendTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Write Data to a Temporary LOB

Figure 4–22 Use Case Diagram: Write data to a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Stream Write

The most efficient way to write large amounts of LOB data is to use `OCILobWrite()` with the streaming mechanism enabled via polling or a callback. If you know how much data will be written to the `LOB` specify that amount when calling `OCILobWrite()`. This will allow for the contiguity of the `LOB` data on disk. Apart from being spatially efficient, contiguous structure of the `LOB` data will make for faster reads and writes in subsequent operations.

Scenario

This example procedure allows the `STORY` data (the storyboard for the clip) to be updated by writing data to the `LOB`.

- "Example: Write Data to a Temporary LOB Using the DBMS_LOB Package" on page 4-135
- "Example: Write Data to a Temporary LOB Using C (OCI)" on page 4-136
- "Example: Write Data to a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-139
- "Example: Write Data to a Temporary LOB Using C++ (Pro*C/C++)" on page 4-140

Example: Write Data to a Temporary LOB Using the DBMS_LOB Package

```
/* Note that the example procedure writeDataToTempLOB_proc is not part of the
   DBMS_LOB package. */
CREATE or REPLACE PROCEDURE writeDataToTempLOB_proc IS
   Lob_loc         CLOB;
   Buffer          VARCHAR2(26);
   Amount          BINARY_INTEGER := 26;
   Position        INTEGER := 1;
   i               INTEGER;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
```

```
   /* Fill the buffer with data to write to the LOB: */
   Buffer := 'abcdefghijklmnopqrstuvwxyz';

   FOR i IN 1..3 LOOP
      DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
      /* Fill the buffer with more data to write to the LOB: */
      Position := Position + Amount;
   END LOOP;
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
END;
```

Example: Write Data to a Temporary LOB Using C (OCI)

```
/* This example illustrates streaming writes with polling */
sb4 write_temp_lobs (OCIError      *errhp,
                     OCISvcCtx     *svchp,
                     OCIStmt       *stmthp,
                     OCIEnv        *envhp)
{
  OCIClobLocator *tclob;
  unsigned int Total = 40000;
  unsigned int amtp;
  unsigned int offset;
  unsigned int remainder, nbytes;
  boolean last;
  ub1 bufp[MAXBUFLEN];
  sb4     err;

  /* Allocate the locators desriptors: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tclob ,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  if(OCILobCreateTemporary(svchp,
           errhp,
           tclob,
           (ub2)0,
           SQLCS_IMPLICIT,
           OCI_TEMP_CLOB,
           OCI_ATTR_NOCACHE,
           OCI_DURATION_SESSION))
  {
     (void) printf("FAILED: CreateTemporary() \n");
```

```
   return -1;
}

/* Open the CLOB: */
checkerr (errhp, (OCILobOpen(svchp, errhp, tclob, OCI_LOB_READWRITE)));

if (Total > MAXBUFLEN)
  nbytes = MAXBUFLEN;   /* We will use Streaming via Standard Polling */
else
  nbytes = Total;       /* Only a single WRITE is required */

/* Fill the Buffer with nbytes worth of Data: */
memset(bufp,'a',32767);

remainder = Total - nbytes;
amtp = 0;
offset = 1;
/* Setting Amount to 0 streams the data until use specifies OCI_LAST_PIECE: */

if (0 == remainder)
{
  amtp = nbytes;
  /* Here, (Total <= MAXBUFLEN ) so we can WRITE in ONE piece: */
  checkerr (errhp, OCILobWrite (svchp, errhp, tclob, &amtp,
                                offset, bufp, nbytes,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));
}
else
{
    /* Here (Total > MAXBUFLEN ) so we use Streaming via Standard Polling: */
    /* WRITE the FIRST piece.  Specifying FIRST initiates Polling: */
    err = OCILobWrite (svchp, errhp, tclob, &amtp,
                       offset, bufp, nbytes,
                       OCI_FIRST_PIECE, (dvoid *)0,
                       (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                       0, SQLCS_IMPLICIT);

    if (err != OCI_NEED_DATA)
      checkerr (errhp, err);

    last = FALSE;
    /* WRITE the NEXT (interim) and LAST pieces: */
    do
```

```
        {
          if (remainder > MAXBUFLEN)
            nbytes = MAXBUFLEN;            /* Still have more pieces to go */
          else
          {
            nbytes = remainder;         /* Here, (remainder <= MAXBUFLEN) */
            last = TRUE;              /* This is going to be the Final piece */
          }

          /* Fill the Buffer with nbytes worth of Data */

          if (last)
          {
            /* Specifying LAST terminates Polling */
            err = OCILobWrite (svchp, errhp, tclob, &amtp,
                               offset, bufp, nbytes,
                               OCI_LAST_PIECE, (dvoid *)0,
                               (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                               0, SQLCS_IMPLICIT);

            if (err != 0)
              checkerr (errhp, err);

          } else

          {
            err =  OCILobWrite (svchp, errhp, tclob, &amtp,
                                offset, bufp, nbytes,
                                OCI_NEXT_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT);

            if (err != OCI_NEED_DATA)
              checkerr (errhp, err);

          }
          /* Determine how much is left to WRITE: */
          remainder = remainder - nbytes;
        } while (!last);
      }
     /* At this point, (remainder == 0) */

   /* Closing the LOB is mandatory if you have opened it: */
      checkerr (errhp, OCILobClose(svchp, errhp, tclob));
```

```
  /* Free the temporary LOB: */
  checkerr(errhp,OCILobFreeTemporary(svchp,errhp,tclob));

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tclob, (ub4) OCI_DTYPE_LOB);
}
```

Example: Write Data to a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-TEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-CLOB      SQL-CLOB.
01  BUFFER         PIC X(20) VARYING.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
WRITE-TEMP.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE and BLOB locators:
    EXEC SQL ALLOCATE :TEMP-CLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-CLOB
    END-EXEC.

    EXEC SQL LOB OPEN :TEMP-CLOB READ WRITE END-EXEC.
```

```
      MOVE "ABCDE12345ABCDE12345" TO BUFFER-ARR.
      MOVE 20 TO BUFFER-LEN.
      MOVE 20 TO AMT.
    * Append the data in BUFFER to TEMP-CLOB:
          EXEC SQL
               LOB WRITE :AMT FROM :BUFFER INTO :TEMP-CLOB
          END-EXEC.

    * Close the LOBs:
          EXEC SQL LOB CLOSE :TEMP-CLOB END-EXEC.

    * Free the temporary LOB:
          EXEC SQL
               LOB FREE TEMPORARY :TEMP-CLOB
          END-EXEC.

    * And free the LOB locators:
          EXEC SQL FREE :TEMP-CLOB END-EXEC.
          STOP RUN.

     SQL-ERROR.
          EXEC SQL
              WHENEVER SQLERROR CONTINUE
          END-EXEC.
          MOVE ORASLNR TO ORASLNRD.
          DISPLAY " ".
          DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
          DISPLAY " ".
          DISPLAY SQLERRMC.
          EXEC SQL
              ROLLBACK WORK RELEASE
          END-EXEC.
          STOP RUN.
```

Example: Write Data to a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
```

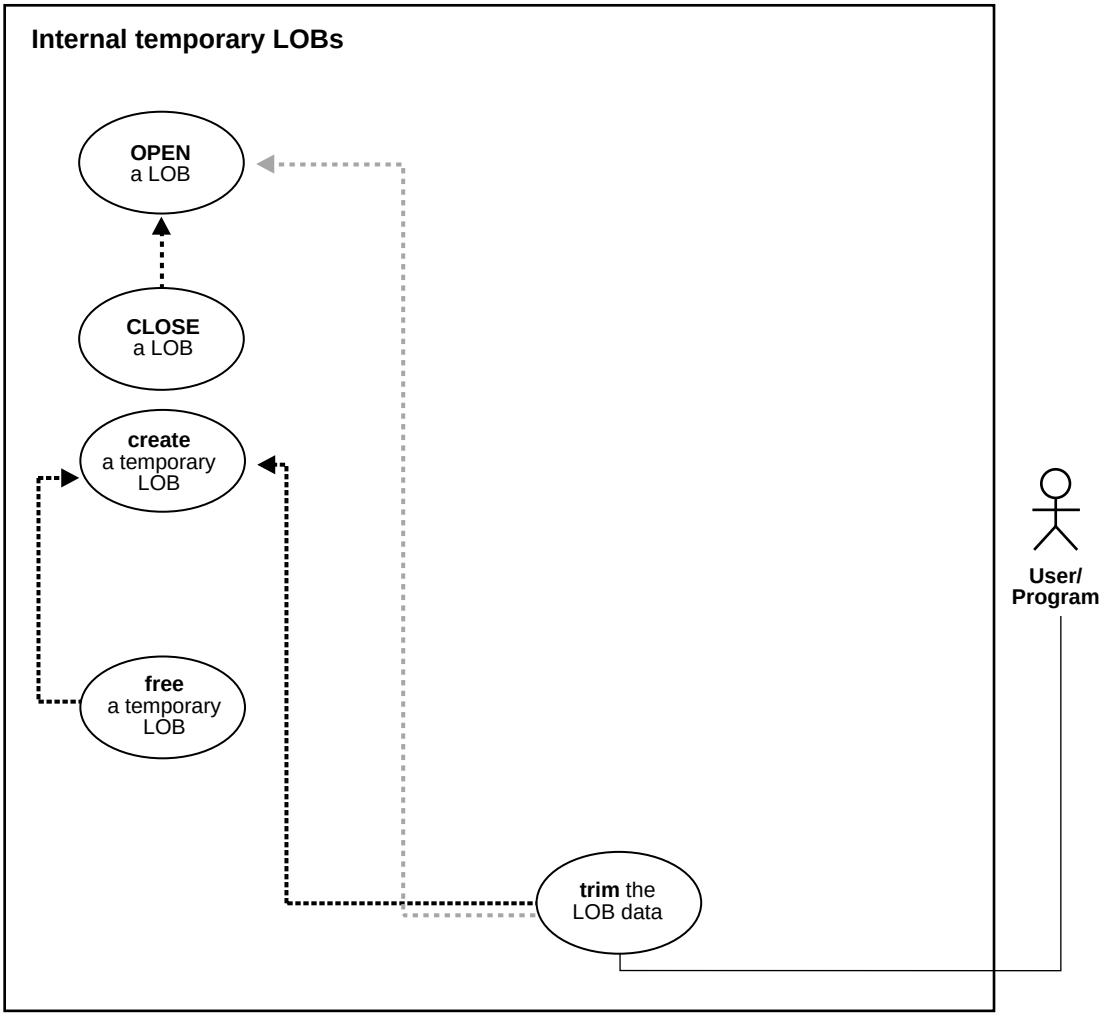
```
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void writeDataToTempLOB_proc(multiple) int multiple;
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  unsigned int Total;
  unsigned int Amount;
  unsigned int remainder, nbytes;
  boolean last;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Initialize the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Open the Temporary LOB: */
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  Total = Amount = (multiple * BufferLength);
  if (Total > BufferLength)
    nbytes = BufferLength;   /* We will use Streaming via Standard Polling */
  else
    nbytes = Total;          /* Only a single WRITE is required */
  /* Fill the Buffer with nbytes worth of Data: */
  memset((void *)Buffer.arr, 32, nbytes);
  Buffer.len = nbytes;       /* Set the Length */
  remainder = Total - nbytes;
  if (0 == remainder)
    {
      /* Here, (Total <= BufferLength) so we can WRITE in ONE piece: */
      EXEC SQL LOB WRITE ONE :Amount FROM :Buffer INTO :Temp_loc;
      printf("Write ONE Total of %d characters\n", Amount);
    }
  else
    {
      /* Here (Total > BufferLength) so use Streaming via Standard Polling */
      /* WRITE the FIRST piece.  Specifying FIRST initiates Polling: */
      EXEC SQL LOB WRITE FIRST :Amount FROM :Buffer INTO :Temp_loc;
      printf("Write FIRST %d characters\n", Buffer.len);
      last = FALSE;
```

```
      /* WRITE the NEXT (interim) and LAST pieces: */
      do
        {
          if (remainder > BufferLength)
            nbytes = BufferLength;         /* Still have more pieces to go */
          else
            {
              nbytes = remainder;     /* Here, (remainder <= BufferLength) */
              last = TRUE;          /* This is going to be the Final piece */
            }
          /* Fill the Buffer with nbytes worth of Data: */
          memset((void *)Buffer.arr, 32, nbytes);
          Buffer.len = nbytes;       /* Set the Length */
          if (last)
            {
              EXEC SQL WHENEVER SQLERROR DO Sample_Error();
              /* Specifying LAST terminates Polling: */
              EXEC SQL LOB WRITE LAST :Amount FROM :Buffer INTO :Temp_loc;
              printf("Write LAST Total of %d characters\n", Amount);
            }
          else
            {
              EXEC SQL WHENEVER SQLERROR DO break;
              EXEC SQL LOB WRITE NEXT :Amount FROM :Buffer INTO :Temp_loc;
              printf("Write NEXT %d characters\n", Buffer.len);
            }
          /* Determine how much is left to WRITE: */
          remainder = remainder - nbytes;
        } while (!last);
    }
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* At this point, (Amount == Total), the total amount that was written. */
  /* Close the Temporary LOB: */
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Free resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  writeDataToTempLOB_proc(1);                          /* Write One Piece */
```

```
  writeDataToTempLOB_proc(4);     /* Write Multiple Pieces using Polling */
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Trim the Temporary LOB Data

Figure 4–23 Use Case Diagram: Trim the Temporary LOB data



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

Our example accesses text (CLOB data) that is referenced in the Script column of the table Voiceover_tab, and trims it.

- "Example: Trim the Temporary LOB Data Using PL/SQL (DBMS_LOB Package)" on page 4-145
- "Example: Trim the Temporary LOB Data Using C (OCI)" on page 4-146
- "Example: Trim the Temporary LOB Data Using COBOL (Pro*COBOL)" on page 4-148
- "Example: Trim the Temporary LOB Data Using C++ (Pro*C/C++)" on page 4-150

Example: Trim the Temporary LOB Data Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure trimTempLOB_proc is not part of the
   DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
   Lob_loc        CLOB;
   Amount         number;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   TrimAmount     number := 100;
BEGIN
   /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Opening the file is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Populate the temporary LOB with some data: */
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Lob_loc, Src_loc, Amount);
   DBMS_LOB.TRIM(Lob_loc,TrimAmount);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.CLOSE(Src_loc);
```

```
   DBMS_LOB.FREETEMPORARY(Lob_loc);
COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Trim the Temporary LOB Data Using C (OCI)

```
sb4 trim_temp_lobs (  OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{
  OCILobLocator *tblob;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  ub4 trim_size = 2;
  sb4 return_code = 0;

  printf("in trim\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in trim\n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED in trim\n");
     return -1;
  }

  /* Set the BFILE to point to the Washington_audio file: */
  if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"),
                       (text *)"Washington_audio",
                       (ub2)strlen("Washington_audio")))
  {
    printf("OCILobFileSetName FAILED\n");
```

```
  return -1;
}

if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
                         OCI_TEMP_BLOB, OCI_ATTR_NOCACHE,
                         OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
}

/* populate the temp LOB with 4000 bytes of data */
if(OCILobLoadFromFile(svchp, errhp, tblob, (OCILobLocator*)bfile,
                      (ub4)amt,(ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;
}

if (OCILobTrim(svchp, errhp, (OCILobLocator *) tblob, trim_size))
{
    printf( "OCILobTrim FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobTrim succeeded for temp LOB \n");
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
  return_code =  -1;
}
```

```
  if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
  {
    printf( "OCILobClose FAILED for temporary LOB \n");
    return_code =  -1;
  }
  /* Free the temporary LOB now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  return return_code;
}
```

Example: Trim the Temporary LOB Data Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-TRIM.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-TRIM.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.
```

```
* Allocate and initialize the BFILE and BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Trim the last half of the data:
     MOVE 5 TO AMT.
     EXEC SQL
          LOB TRIM :TEMP-BLOB TO :AMT
     END-EXEC.

* Close the LOBs:
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.

* And free the LOB locators:
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     EXEC SQL FREE :SRC-BFILE END-EXEC.
```

```
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

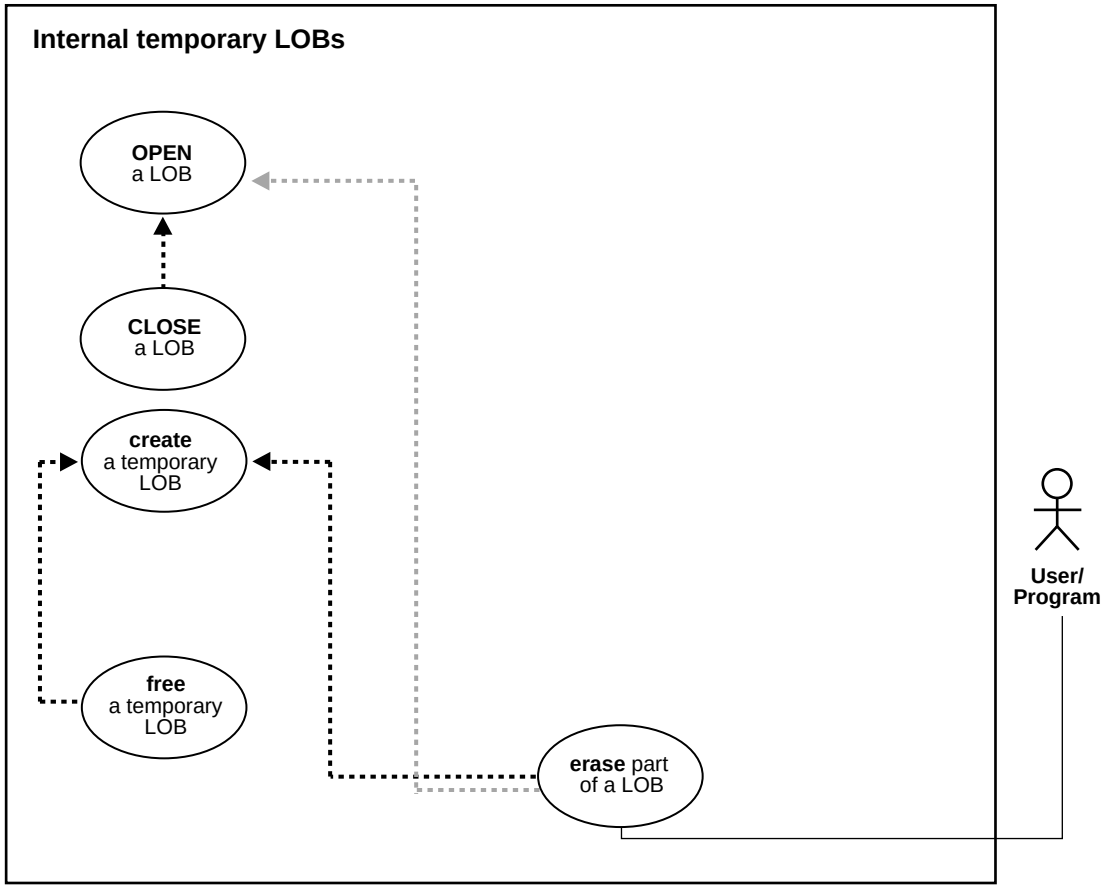
Example: Trim the Temporary LOB Data Using C++ (Pro*C/C++)

```
void trimTempLOB_proc()
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void trimTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 4096;
  int trimLength;

  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
```

```
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load the specified amount from the BFILE into the Temporary LOB: */
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Set the new length of the Temporary LOB: */
  trimLength = (int) (Amount / 2);
  /* Trim the Temporary LOB to its new length: */
  EXEC SQL LOB TRIM :Temp_loc TO :trimLength;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  trimTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Erase Part of a Temporary LOB

Figure 4–24 Use Case Diagram: Erase part of a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

Example: Erase Part of a Temporary LOB Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure eraseTempLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
   Lob_loc        CLOB;
   amt            number;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount         INTEGER := 32767;
BEGIN
   /* Create a temporary LOB: */
   DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE,DBMS_LOB.SESSION);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);
   /* Populate the temporary LOB with some data: */
   Amount := 32767;
   DBMS_LOB.LOADFROMFILE(Lob_loc, Src_loc, Amount);
   /* Erase  the LOB data: */
   amt := 3000;
   DBMS_LOB.ERASE(Lob_loc, amt, 2);
    /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE (Lob_loc);
   DBMS_LOB.CLOSE(Src_loc);
   DBMS_LOB.FREETEMPORARY(Lob_loc);
COMMIT;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: Erase Part of a Temporary LOB Using C (OCI)

```
/* Erase 2 bytes at offset 100 in a temporary LOB: */

sb4 erase_temp_lobs ( OCIError      *errhp,
                      OCISvcCtx     *svchp,
                      OCIStmt       *stmthp,
                      OCIEnv        *envhp)
{

  OCILobLocator *tblob;
  OCILobLocator *bfile;
  ub4 amt = 4000;
  ub4 erase_size = 2;
  ub4 erase_offset = 100;
  sb4 return_code = 0;

  printf("in erase\n");
  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &tblob,
                        (ub4) OCI_DTYPE_LOB,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED \n");
     return -1;
  }

  if(OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &bfile,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0))
  {
     printf("OCIDescriptor Alloc FAILED \n");
     return -1;
  }

  /* Set the BFILE to point to the Washington_audio file: */
  if(OCILobFileSetName(envhp, errhp, &bfile,
                       (text *)"AUDIO_DIR",
                       (ub2)strlen("AUDIO_DIR"),
                       (text *)"Washington_audio",
                       (ub2)strlen("Washington_audio")))
  {
    printf("OCILobFileSetName FAILED\n");
    return -1;
  }
```

```
if (OCILobFileOpen(svchp, errhp, (OCILobLocator *) bfile, OCI_LOB_READONLY))
{
  printf( "OCILobFileOpen FAILED  for the bfile\n");
  return_code = -1;
}

if(OCILobCreateTemporary(svchp,errhp,tblob,(ub2)0, SQLCS_IMPLICIT,
    OCI_TEMP_BLOB, OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
{
   (void) printf("FAILED: CreateTemporary() \n");
   return -1;
}

if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
{
  printf( "OCILobOpen FAILED for temp LOB \n");
  return_code =  -1;
}

/* Populate the temp LOB with 4000 bytes of data: */
if(OCILobLoadFromFile(svchp,
                      errhp,
                      tblob,
                      (OCILobLocator*)bfile,
                      (ub4)amt,
                      (ub4)1,(ub4)1))
{
  printf( "OCILobLoadFromFile FAILED\n");
  return_code = -1;

}

if (OCILobErase(svchp, errhp, (OCILobLocator *) tblob, &erase_size,
               erase_offset))
{
    printf( "OCILobErase FAILED for temp LOB \n");
    return_code = -1;
} else
{
    printf( "OCILobErase succeeded for temp LOB \n");
}

if (OCILobClose(svchp, errhp, (OCILobLocator *) bfile))
{
  printf( "OCILobClose FAILED for bfile \n");
```

```
    return_code =  -1;
  }

  if (OCILobClose(svchp, errhp, (OCILobLocator *) tblob))
  {
    printf( "OCILobClose FAILED for temporary LOB \n");
    return_code =  -1;
  }
  /* free the temporary LOB now that we are done using it */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return_code = -1;
  }
  return return_code;

}
```

Example: Erase Part of a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-ERASE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".

01  TEMP-BLOB      SQL-BLOB.
01  SRC-BFILE      SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  DIR-IND        PIC S9(4) COMP.
01  FNAME-IND      PIC S9(4) COMP.
01  AMT            PIC S9(9) COMP VALUE 10.
01  POS            PIC S9(9) COMP VALUE 1.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-BLOB-ERASE.
```

```
     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locator:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.
     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

* Open source BFILE and destination temporary BLOB:
     EXEC SQL LOB OPEN :TEMP-BLOB READ WRITE END-EXEC.
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

     EXEC SQL
          LOB LOAD :AMT FROM FILE :SRC-BFILE INTO :TEMP-BLOB
     END-EXEC.

* Erase some of the LOB data:
     EXEC SQL
          LOB ERASE :AMT FROM :TEMP-BLOB AT :POS
     END-EXEC.

* Close the LOBs
     EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
     EXEC SQL LOB CLOSE :TEMP-BLOB END-EXEC.

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
```

```
      * And free the LOB locators:
           EXEC SQL FREE :TEMP-BLOB END-EXEC.
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

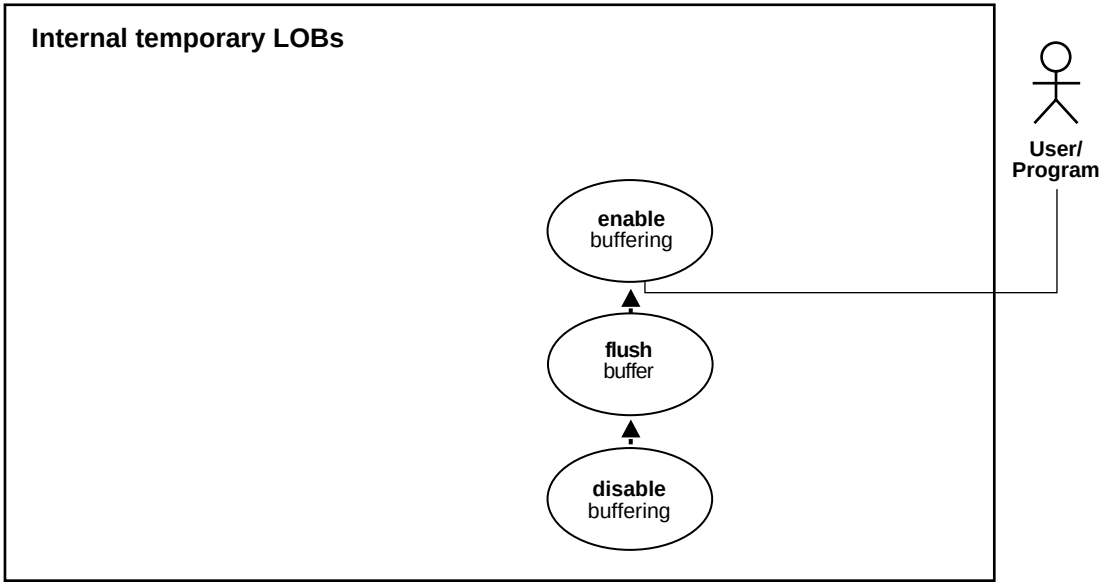
Example: Erase Part of a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void eraseTempLOB_proc()
{
  OCIBlobLocator *Temp_loc;
  OCIBFileLocator *Lob_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount;
  int Position = 1024;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
```

```
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Allocate and Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Opening the LOBs is Optional: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL LOB OPEN :Temp_loc READ WRITE;
  /* Load a specified amount from the BFILE into the Temporary LOB: */
  Amount = 4096;
  EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
  /* Erase a specified amount from the Temporary LOB at a given position: */
  Amount = 2048;
  EXEC SQL LOB ERASE :Amount FROM :Temp_loc AT :Position;
  /* Closing the LOBs is Mandatory if they have been Opened: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL LOB CLOSE :Temp_loc;
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Lob_loc;
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  eraseTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Enable LOB Buffering for a Temporary LOB

Figure 4–25 Use Case Diagram: Enable LOB Buffering for a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

You enable buffering in order to perform a small series of reads or writes. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations.

Please note that you would not enable buffering to perform the stream read and write involved in checkin and checkout.

- "Example: Enable LOB Buffering for a Temporary LOB Using C (OCI)" on page 4-161

- "Example: Enable LOB Buffering for a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-163
- "Example: Enable LOB Buffering for a Temporary LOB Using C++ (Pro*C/C++)" on page 4-163

Example: Enable LOB Buffering for a Temporary LOB Using C (OCI)

```
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *tblob;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  /* Open the BLOB: */
  if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp LOB \n");
    return -1;
  }

  /* Enable LOB Buffering: */
```

```
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);
  offset = 1;
  checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));


  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                     (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tblob));

  /* Free the temporary LOB now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

  return;

}
```

Example: Enable LOB Buffering for a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB          SQL-BLOB.
01  BUFFER             PIC X(80).
01  AMT                PIC S9(9) COMP VALUE 10.
01  ORASLNRD        PIC 9(4).

    EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-BUFFERING.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locators:
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

* Enable buffering for the temporary LOB:
    EXEC SQL
         LOB ENABLE BUFFERING :TEMP-BLOB
    END-EXEC.
*
* Write some data to the temporary LOB here:
*
    MOVE '252525262626252525' TO BUFFER.
    EXEC SQL
         LOB WRITE ONE :AMT FROM :BUFFER
         INTO :TEMP-BLOB
    END-EXEC
```

```
       * Flush the buffered writes:
            EXEC SQL
                 LOB FLUSH BUFFER :TEMP-BLOB FREE
            END-EXEC.

       * Disable buffering for the temporary LOB:
            EXEC SQL
                 LOB DISABLE BUFFERING :TEMP-BLOB
            END-EXEC.

            EXEC SQL
                 LOB FREE TEMPORARY :TEMP-BLOB
            END-EXEC.

            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

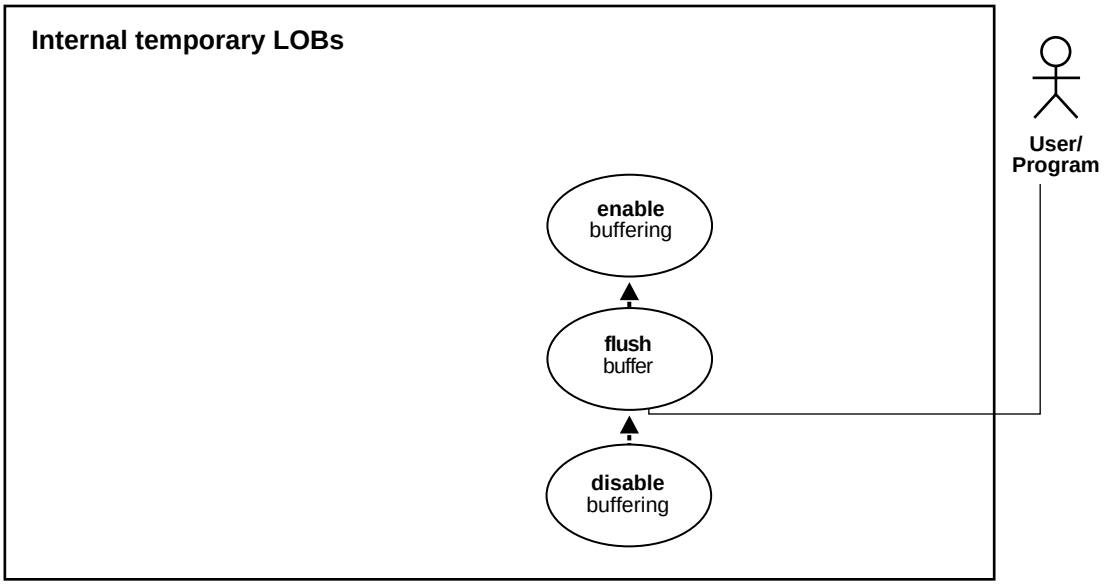
Example: Enable LOB Buffering for a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
```

```
}

#define BufferLength 1024

void enableBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  printf("Wrote %d characters using the Buffering Subsystem\n", Length);
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}
void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  enableBufferingTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Flush Buffer for a Temporary LOB

Figure 4–26 Use Case Diagram: Flush Buffer for a Temporary LOB



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

- "Example: Flush Buffer for a Temporary LOB Using C (OCI)" on page 4-167
- "Example: Flush Buffer for a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-168
- "Example: Flush Buffer for a Temporary LOB Using C++ (Pro*C/C++)" on page 4-170

Example: Flush Buffer for a Temporary LOB Using C (OCI)

```
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *tblob;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;

  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary lob :*/
  if(OCILobCreateTemporary(svchp, errhp, tblob, (ub2)0,
                           SQLCS_IMPLICIT, OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE, OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  /* Open the BLOB: */
  if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp lob \n");
    return -1;
  }

  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
```

```
  buflen = sizeof(bufp);
  offset = 1;
  checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));

  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                     (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tblob));

  /* Free the temporary lob now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

  return;

}
```

Example: Flush Buffer for a Temporary LOB Using COBOL (Pro*COBOL)

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID. FREE-TEMPORARY.
     ENVIRONMENT DIVISION.
     DATA DIVISION.
     WORKING-STORAGE SECTION.
```

```
 01  USERID   PIC X(11) VALUES "USER1/USER1".

 01  TEMP-BLOB      SQL-BLOB.
 01  IS-TEMP        PIC S9(9) COMP.
 01  ORASLNRD        PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 FREE-TEMPORARY.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BLOB locators:
     EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.

     EXEC SQL
          LOB CREATE TEMPORARY :TEMP-BLOB
     END-EXEC.

* Do something with the temporary LOB here:

* Free the temporary LOB:
     EXEC SQL
          LOB FREE TEMPORARY :TEMP-BLOB
     END-EXEC.
     EXEC SQL FREE :TEMP-BLOB END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
         ROLLBACK WORK RELEASE
```

```
           END-EXEC.
           STOP RUN.
```

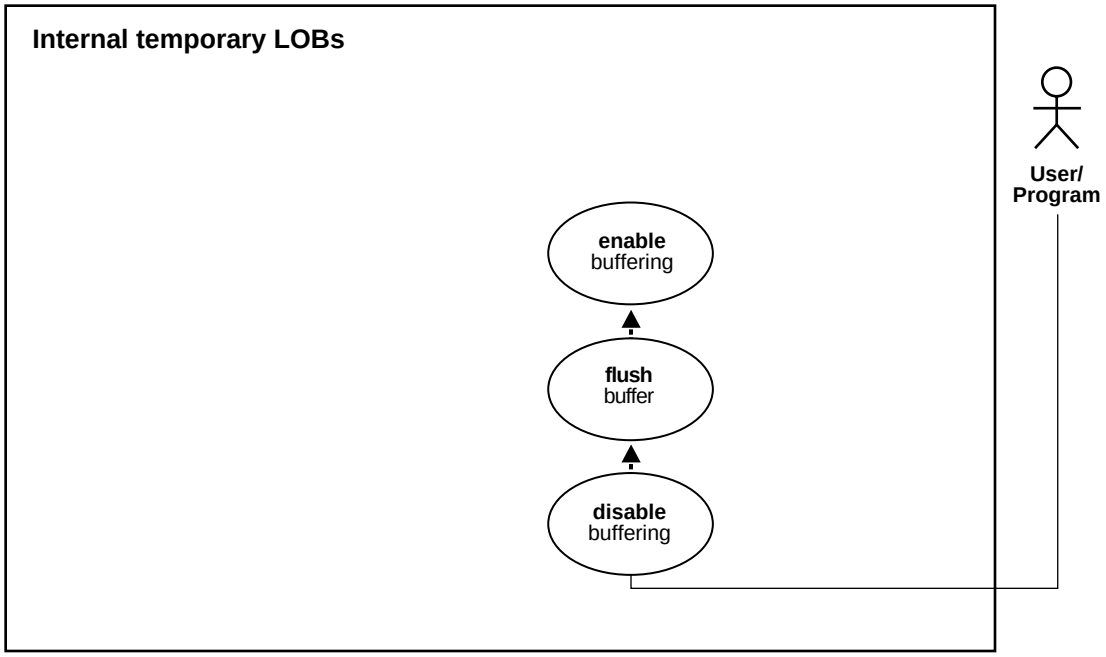
Example: Flush Buffer for a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void flushBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 8; multiple++)
    {
      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
         FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem: */
```

```
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  printf("Wrote %d characters using the Buffering Subsystem\n", Length);
  /* Free the Temporary LOB */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  flushBufferingTempLOB_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Disable LOB Buffering for a Temporary LOB

Figure 4–27 Use Case Diagram: Disable LOB Buffering



To refer to the table of all basic operations having to do with Internal Temporary LOBs see:

- "Use Case Model: Internal Temporary LOBs" on page 4-2

Scenario

You enable buffering in order to perform a small series of reads or writes. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations.

Please note that you would not enable buffering to perform the stream read and write involved in checkin and checkout.

- "Example: Disable LOB Buffering Using C (OCI)" on page 4-173

- "Example: Disable LOB Buffering for a Temporary LOB Using COBOL (Pro*COBOL)" on page 4-175
- "Example: Disable LOB Buffering for a Temporary LOB Using C++ (Pro*C/C++)" on page 4-176

Example: Disable LOB Buffering Using C (OCI)

```
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *tblob;
  ub4 amt;
  ub4 offset;
  sword retval;
  ub1 bufp[MAXBUFLEN];
  ub4 buflen;


  /* Allocate the descriptor for the lob locator: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &tblob,
                            (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

  /* Select the BLOB: */
  printf (" create a temporary Lob\n");
  /* Create a temporary LOB: */
  if(OCILobCreateTemporary(svchp,errhp, tblob, (ub2)0, SQLCS_IMPLICIT,
                           OCI_TEMP_BLOB,
                           OCI_ATTR_NOCACHE,
                           OCI_DURATION_SESSION))
  {
    (void) printf("FAILED: CreateTemporary() \n");
    return -1;
  }

  /* Open the BLOB: */
  if (OCILobOpen(svchp, errhp, (OCILobLocator *) tblob, OCI_LOB_READWRITE))
  {
    printf( "OCILobOpen FAILED for temp LOB \n");
    return -1;
  }
```

```
  /* Enable LOB Buffering: */
  printf (" enable LOB buffering\n");
  checkerr (errhp, OCILobEnableBuffering(svchp, errhp, tblob));

  printf (" write data to LOB\n");

  /* Write data into the LOB: */
  amt    = sizeof(bufp);
  buflen = sizeof(bufp);
  offset = 1;
  checkerr (errhp, OCILobWrite (svchp, errhp, tblob, &amt,
                                offset, bufp, buflen,
                                OCI_ONE_PIECE, (dvoid *)0,
                                (sb4 (*)(dvoid*,dvoid*,ub4*,ub1 *))0,
                                0, SQLCS_IMPLICIT));


  /* Flush the buffer: */
  printf(" flush the LOB buffers\n");
  checkerr (errhp, OCILobFlushBuffer(svchp, errhp, tblob,
                                   (ub4)OCI_LOB_BUFFER_FREE));

  /* Disable Buffering: */
  printf (" disable LOB buffering\n");
  checkerr (errhp, OCILobDisableBuffering(svchp, errhp, tblob));

  /* Subsequent LOB WRITEs will not use the LOB Buffering Subsystem */

  /* Closing the BLOB is mandatory if you have opened it: */
  checkerr (errhp, OCILobClose(svchp, errhp, tblob));

  /* Free the temporary LOB now that we are done using it: */
  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf("OCILobFreeTemporary FAILED \n");
    return -1;
  }

  /* Free resources held by the locators: */
  (void) OCIDescriptorFree((dvoid *) tblob, (ub4) OCI_DTYPE_LOB);

  return;

}
```

Example: Disable LOB Buffering for a Temporary LOB Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB          SQL-BLOB.
01  BUFFER             PIC X(80).
01  AMT                PIC S9(9) COMP VALUE 10.
01  ORASLNRD        PIC 9(4).

    EXEC SQL VAR BUFFER IS RAW(80) END-EXEC.
    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
TEMP-LOB-BUFFERING.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the CLOB locators:
    EXEC SQL ALLOCATE :TEMP-BLOB END-EXEC.
    EXEC SQL
         LOB CREATE TEMPORARY :TEMP-BLOB
    END-EXEC.

* Enable buffering for the temporary LOB:
    EXEC SQL
         LOB ENABLE BUFFERING :TEMP-BLOB
    END-EXEC.

* Write some data to the temporary LOB here:

    MOVE '252525262626252525' TO BUFFER.
    EXEC SQL
         LOB WRITE ONE :AMT FROM :BUFFER
         INTO :TEMP-BLOB
    END-EXEC
```

```
       * Flush the buffered writes:
            EXEC SQL
                 LOB FLUSH BUFFER :TEMP-BLOB FREE
            END-EXEC.

       * Disable buffering for the temporary LOB:
            EXEC SQL
                 LOB DISABLE BUFFERING :TEMP-BLOB
            END-EXEC.

            EXEC SQL
                 LOB FREE TEMPORARY :TEMP-BLOB
            END-EXEC.

            EXEC SQL FREE :TEMP-BLOB END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Disable LOB Buffering for a Temporary LOB Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
```

```
}

#define BufferLength 1024

void disableBufferingTempLOB_proc()
{
  OCIClobLocator *Temp_loc;
  varchar Buffer[BufferLength];
  int Amount = BufferLength;
  int multiple, Length = 0, Position = 1;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate and Create the Temporary LOB: */
  EXEC SQL ALLOCATE :Temp_loc;
  EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
  /* Enable use of the LOB Buffering Subsystem: */
  EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
  memset((void *)Buffer.arr, 42, BufferLength);
  Buffer.len = BufferLength;
  for (multiple = 0; multiple < 7; multiple++)
    {
      /* Write Data to the Temporary LOB: */
      EXEC SQL LOB WRITE ONE :Amount
                    FROM :Buffer INTO :Temp_loc AT :Position;
      Position += BufferLength;
    }
  /* Flush the contents of the buffers and Free their resources: */
  EXEC SQL LOB FLUSH BUFFER :Temp_loc FREE;
  /* Turn off use of the LOB Buffering Subsystem:  */
  EXEC SQL LOB DISABLE BUFFERING :Temp_loc;
  /* Write APPEND can only be done when Buffering is Disabled: */
  EXEC SQL LOB WRITE APPEND ONE :Amount FROM :Buffer INTO :Temp_loc;
  EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
  printf("Wrote a total of %d characters\n", Length);
  /* Free the Temporary LOB: */
  EXEC SQL LOB FREE TEMPORARY :Temp_loc;
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Temp_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  disableBufferingTempLOB_proc();
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

5

External LOBs (BFILEs)

In this chapter we describe how to work with External LOBs (BFILEs) in terms of use cases. That is, we discuss each operation on a LOB (such as "See If a Temporary LOB is Open") in terms of a use case by that name. The table listing all the use cases is provided at the head of the chapter (see "Use Case Model: External LOBs" on page 5-2). A summary figure, "Use Case Model Diagram: External LOBs", locates all the use cases in single drawing. If you are using the HTML version of this document, you can use this figure to navigate to the use case in which you are interested by clicking on the relevant use case title.

The individual use cases are themselves laid out as follows:

- A figure that depicts the use case (see "Preface" for a description of how to interpret these diagrams).
- A scenario that portrays one implementation of the use case in terms of the hypothetical multimedia application described above (see "An Example Application" on page 1-39 in Chapter 1, "Introduction to Working With LOBs").
- Code examples in each of the programmatic environments which can be utilized to implement the use case (see "Programmatic Environments for Operating on LOBs" on page 1-9 in Chapter 1, "Introduction to Working With LOBs").

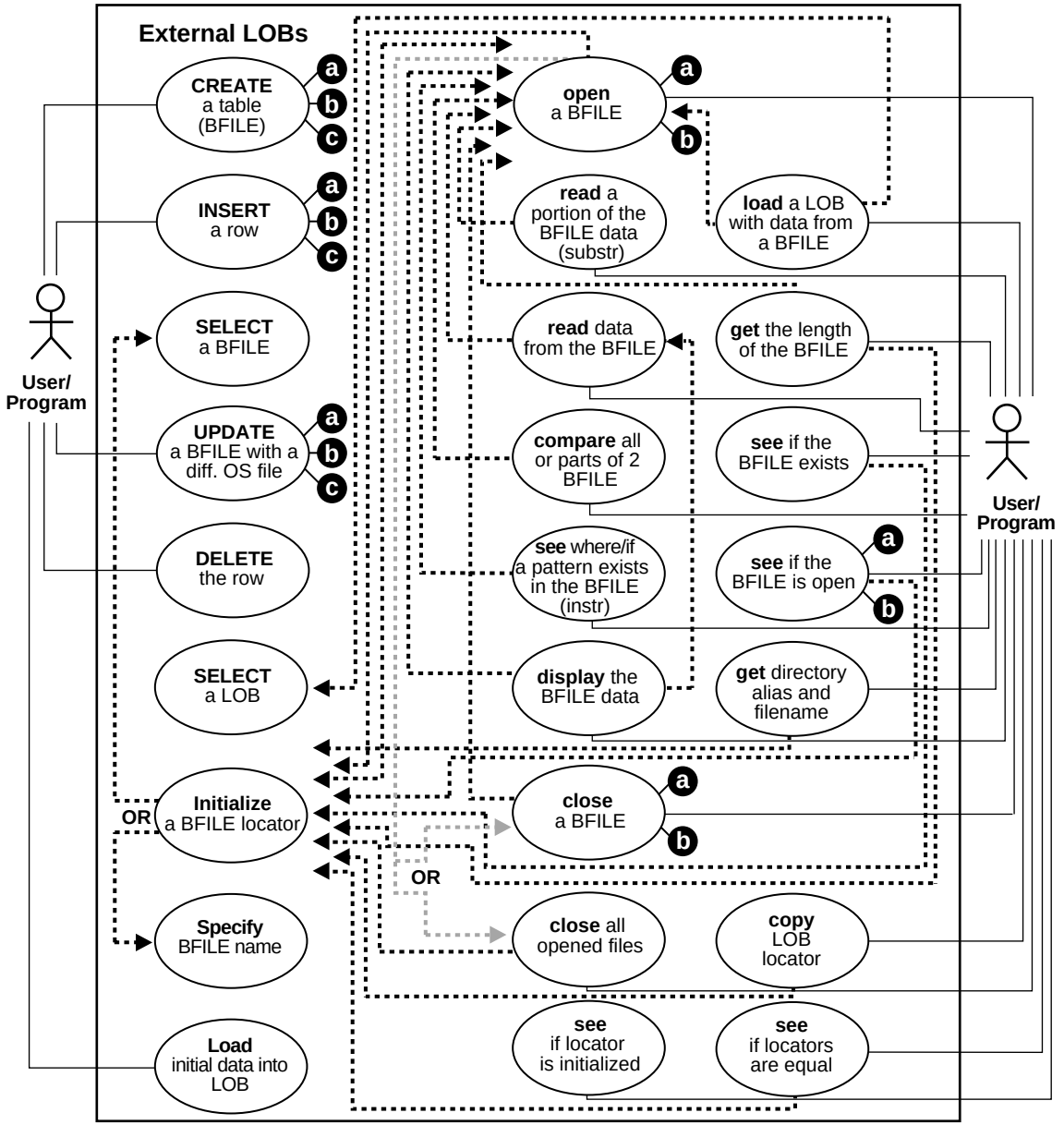
Use Case Model: External LOBs

Table 5–1 Use Case Model: External LOBs

Use Case and Page
"Three Ways to Create a Table Containing a BFILE" on page 5-12
CREATE a Table Containing a BFILE on page 5-13
CREATE a Table of an Object Type with a BFILE Attribute on page 5-16
CREATE a Table with a Nested Table Containing a BFILE on page 5-19 on page 5-19
Three Ways to Insert a Row Containing a BFILE on page 5-21
INSERT a Row by means of BFILENAME() on page 5-22
INSERT a Row Containing a BFILE as SELECT on page 5-29
INSERT a Row Containing a BFILE by Initializing a BFILE Locator on page 5-30
Load External LOB (BFILE) Data into a Table on page 5-38
Load a LOB with Data from a BFILE on page 5-41
Two Ways to Open a BFILE on page 5-51
Open a BFILE with FILEOPEN on page 5-53
Open a BFILE with OPEN on page 5-59
Two Ways to See If a BFILE is Open on page 5-67
See If the BFILE is Open with FILEISOPEN on page 5-69
See If the BFILE is Open Using ISOPEN on page 5-74
Display the BFILE Data on page 5-82
Read the Data from a BFILE on page 5-93
Read a Portion of the BFILE Data (substr) on page 5-103
Compare All or Parts of Two BFILES on page 5-110
See If a Pattern Exists (instr) in the BFILE on page 5-119
See If the BFILE Exists on page 5-127
Get the Length of a BFILE on page 5-136
Copy a LOB Locator for a BFILE on page 5-145
See If a LOB Locator for a BFILE Is Initialized on page 5-153
See If One LOB Locator for a BFILE Is Equal to Another on page 5-156

Use Case and Page
Get Directory Alias and Filename on page 5-161
Three Ways to Update a Row Containing a BFILE on page 5-169
UPDATE a BFILE Using BFILENAME() on page 5-170
UPDATE a BFILE as SELECT on page 5-173
UPDATE a BFILE by Initializing a BFILE Locator on page 5-174
Two Ways to Close a BFILE on page 5-182
Close a BFILE with FILECLOSE on page 5-184
Close a BFILE with CLOSE on page 5-189
Close All Open BFILEs on page 5-197
DELETE the Row of a Table Containing a BFILE on page 5-205

Figure 5–1 Use Case Model Diagram: External LOBs



Accessing External LOBs (SQL DML)

Directory Object

The DIRECTORY object enables administering the access and usage of BFILEs in an Oracle Server (see the CREATE DIRECTORY command in the *Oracle8i SQL Reference).* A DIRECTORY specifies a *logical alias name* for a physical directory on the server's filesystem under which the file to be accessed is located. You can access a file in the server's filesystem only if granted the required access privilege on the DIRECTORY object.

The DIRECTORY object also provides the flexibility to manage the locations of the files, instead of forcing you to hardcode the absolute pathnames of the physical files in your applications. A DIRECTORY alias is used in conjunction with the BFILENAME() function (in SQL and PL/SQL), or the OCILobFileSetName() (in OCI) for initializing a BFILE locator.

Note: Oracle does not verify that the directory and pathname you specify actually exist. You should take care to specify a valid directory in your operating system. If your operating system uses case-sensitive pathnames, be sure you specify the directory in the correct format. There is no need to specify a terminating slash (e.g., ***/tmp/*** is not necessary, simply use ***/tmp***).

Initializing BFILES using BFILENAME()

In order to associate an operating system file to a BFILE, it is necessary to first create a DIRECTORY object which is an alias for the full pathname to the operating system file.

You use Oracle SQL DML to associate existing operating system files with the relevant database records of a particular table. You can use the SQL INSERT statement to initialize a BFILE column to point to an existing file in the server's filesystem, and you can use a SQL UPDATE statement to change the reference target of the BFILE. You can also initialize a BFILE to NULL and then update it later to refer to an operating system file via the BFILENAME() function. OCI users can also use OCILobFIleSetName() to initialize a BFILE locator variable that is then used in the VALUES clause of an INSERT statement.

For example, the following statements associate the files *Image1.gif* and *image2.gif* with records having key_value of 21 and 22 respectively. 'IMG' is a DIRECTORY

object that represents the physical directory under which *Image1.dif* and i*mage2.dif* are stored.

Note: You may need to set up data structures similar to the following for certain examples to work:

```
CREATE TABLE Lob_table (
   Key_value NUMBER NOT NULL,
   F_lob BFILE)
```

```
INSERT INTO Lob_table VALUES
  (21,  BFILENAME('IMG', 'Image1.gif'));
INSERT INTO Lob_table VALUES
  (22, BFILENAME('IMG', 'image2.gif'));
```

The UPDATE statement below changes the target file to *image3.gif* for the row with key_value 22**.**

```
UPDATE Lob_table SET f_lob = BFILENAME('IMG', 'image3.gif')
   WHERE Key_value = 22;
```

BFILENAME() is a built-in function that is used to initialize the BFILE column to point to the external file.

Once physical files are associated with records using SQL DML, subsequent read operations on the BFILE can be performed using PL/SQL DBMS_LOB package and OCI. However, these files are read-only when accessed through BFILES, and so they cannot be updated or deleted through BFILES.

As a consequence of the reference-based semantics for BFILEs, it is possible to have multiple BFILE columns in the same record or different records referring to the same file. For example, the UPDATE statements below set the BFILE column of the row with key_value 21 in lob_table to point to the same file as the row with key_value 22.

```
UPDATE lob_table
   SET f_lob = (SELECT f_lob FROM lob_table WHERE key_value = 22)
      WHERE key_value = 21;
```

You should think of BFILENAME() in terms of initialization — it can initialize the value for both a BFILE column and that of a BFILE (automatic) variable declared inside a PL/SQL module. This has the unique advantage that if your need for a particular BFILE is temporary, and scoped just within the module on which you are working, you can utilize the BFILE related APIs on the variable without ever

having to associate this with a column in the database. There is a further advantage as well. Since you are not forced to create a BFILE column in a server side table, initialize this column value, and then retrieve this column value via a SELECT, you save a roundtrip to the server.

For more information, refer to the example given for DBMS_LOB.LOADFROMFILE (see "Load a LOB with Data from a BFILE" on page 5-41). The OCI counterpart for BFILENAME() is OCILobFileSetName(), which can be used in a similar fashion.

DIRECTORY Name Specification

The naming convention for DIRECTORY objects is the same as that done for tables and indexes. That is, normal identifiers are interpreted in uppercase, but delimited identifiers are interpreted as is. For example, the following statement

```
CREATE DIRECTORY scott_dir AS '/usr/home/scott';
```

creates a directory object whose name is 'SCOTT_DIR' (in uppercase). But if a delimited identifier is used for the DIRECTORY name, as shown in the following statement

```
CREATE DIRECTORY "Mary_Dir" AS '/usr/home/mary';
```

the directory object's name is 'Mary_Dir'. Use 'SCOTT_DIR' and 'Mary_Dir' when calling BFILENAME(). For example:

```
BFILENAME('SCOTT_DIR', 'afile')
BFILENAME('Mary_Dir', 'afile')
```

BFILE Security

This section introduces the BFILE security model and the associated SQL DDL and DML. The main features for BFILE security are:

- SQL DDL to CREATE and REPLACE / ALTER a DIRECTORY object.
- SQL DML to GRANT and REVOKE the READ system and object privileges on DIRECTORY objects.

Ownership and Privileges

The DIRECTORY is a *system owned* object. For more information on system owned objects, see *Oracle8i SQL Reference*. Oracle8*i* supports two new system privileges, which are granted only to the DBA account:

- CREATE ANY DIRECTORY — for creating or altering the directory object creation
- DROP ANY DIRECTORY — for deleting the directory object

The READ privilege on the DIRECTORY object allows you to read files located under that directory. The creator of the DIRECTORY object automatically earns the READ privilege. If you have been granted the READ privilege with GRANT option, you may in turn grant this privilege to other users/roles and add them to your privilege domains.

It is important to note that the READ privilege is defined only on the DIRECTORY object. The physical directory that it represents may or may not have the corresponding operating system privileges (*read* in this case) for the Oracle Server process. It is the DBA's responsibility to ensure that the physical directory exists, and *read* permission for the Oracle Server process is enabled on the file, the directory, and the path leading to it. It is also the DBA's responsibility to make sure that the directory remains available, and the *read* permission remains enabled, for the entire duration of file access by database users.

The privilege just implies that as far as the Oracle Server is concerned, you may read from files in the directory. These privileges are checked and enforced by the PL/SQL DBMS_LOB package and OCI APIs at the time of the actual file operations.

WARNING: Because the CREATE ANY DIRECTORY and DROP ANY DIRECTORY privileges potentially expose the server filesystem to all database users, the DBA should be prudent in granting these privileges to normal database users to prevent any accidental or malicious security breach.

SQL DDL for BFILE security

Refer to the *Oracle8i SQL Reference* for information about the following SQL DDL commands that create, replace, and drop directory objects:

- CREATE DIRECTORY
- DROP DIRECTORY

SQL DML for BFILE security

Refer to the *Oracle8i SQL Reference* for information about the following SQL DML commands that provide security for BFILEs:

- GRANT (system privilege)

- GRANT (object privilege)
- REVOKE (system privilege)
- REVOKE (object privilege)
- AUDIT (new statements)
- AUDIT (schema objects)

Catalog Views on Directories

Catalog views are provided for directory objects to enable users to view object names and their corresponding paths and privileges. The supported views are:

- ALL_DIRECTORIES (OWNER, DIRECTORY_NAME, DIRECTORY_PATH)

  This view describes all the directories accessible to the user.

- DBA_DIRECTORIES(OWNER, DIRECTORY_NAME, DIRECTORY_PATH)

  This view describes all the directories specified for the entire database.

Guidelines for DIRECTORY Usage

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 filesystem. But to realize this goal, it is very important that the DBA follow these guidelines when using directory objects:

- A DIRECTORY should not be mapped to physical directories which contain Oracle data files, control files, log files, and other system files. Tampering with these files (accidental or otherwise) could potentially corrupt the database or the server operating system.
- The system privileges such as CREATE 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.
- 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.
- DIRECTORY objects should not be arbitrarily dropped or replaced when the database is in operation. If this were to happen, DBMS_LOB or OCI operations *from all sessions* on all files associated with this directory object will fail. Further,

if a DROP or REPLACE command is executed before these files could be successfully closed, the references to these files will be lost in the programs, and system resources associated with these files will not be released until the session(s) is shutdown.

The only recourse left to PL/SQL users, for example, will be to either execute 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.

- Similarly, revoking an user's privilege on a directory using the REVOKE statement causes all subsequent operations on dependent files from the user's session to fail. Either you must re-acquire the privileges to close the file, or execute 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.

BFILEs in Multi-Threaded Server (MTS) Mode

Oracle8*i* does not support session migration for BFILEs in MTS mode. This implies that operations on open BFILEs can persist beyond the end of a call to an MTS server. Sessions involving BFILE operations need to be bound to one shared server, they cannot migrate from one server to another.

External LOB Locators (BFILE Locators)

For BFILEs, the value is stored in a server-side operating system file; i.e., external to the database. The BFILE locator that refers to that file is stored in the row. If a BFILE locator variable that is used in a DBMS_LOB.FILEOPEN() (for example L1) is assigned to another locator variable, (for example L2), both L1 and L2 point to the same file. This means that two rows in a table with a BFILE column can refer to the same file or to two distinct files — a fact that the canny developer might turn to advantage, but which could well be a pitfall for the unwary.

A BFILE locator variable in a PL/SQL or OCI program behaves like any other automatic variable. With respect to file operations, it behaves like a *file descriptor* available as part of the standard I/O library of most conventional programming languages. This implies that once you define and initialize a BFILE locator, and

open the file pointed to by this locator, all subsequent operations until the closure of this file must be done from within the same program block using this locator or local copies of this locator.

The BFILE locator variable can be used, just as any scalar, as a parameter to other procedures, member methods, or external function callouts. However, it is recommended that you open and close a file from the same program block at the same nesting level, in PL/SQL and OCI programs.

If the object contains a BFILE, you must set the BFILE value before flushing the object to the database, thereby inserting a new row. In other words, you must call OCILobFileSetName() after OCIObjectNew() and before OCIObjectFlush(). It is an error to INSERT/UPDATE a BFILE without indicating a directory alias and filename.

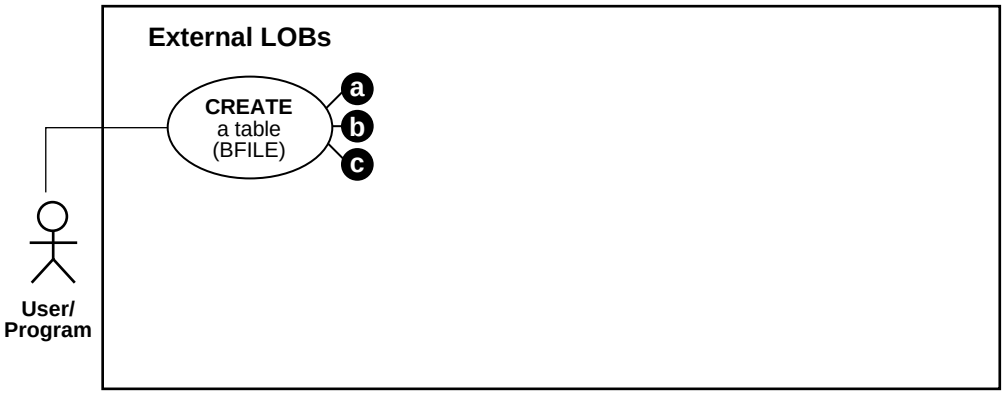
This rule also applies to users using an OCI bind variable for a BFILE in an insert/update statement. The OCI bind variable must be initialized with a directory alias and filename before issuing the insert or update statement. Note that OCISetAttr() does not allow the user to set a BFILE locator to NULL.

General rule: Before using SQL to insert or update a row with a BFILE, the user must either initialize the BFILE

- to NULL (not possible if using an OCI bind variable) or
- to a directory alias and filename

Three Ways to Create a Table Containing a BFILE

Figure 5–2 Use Case Diagram: Three Ways to Create a Table Containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

It is possible to incorporate `BFILE`s into tables in three ways.

a. `BFILE`s may be columns in a table — see "CREATE a Table of an Object Type with a BFILE Attribute" on page 5-13

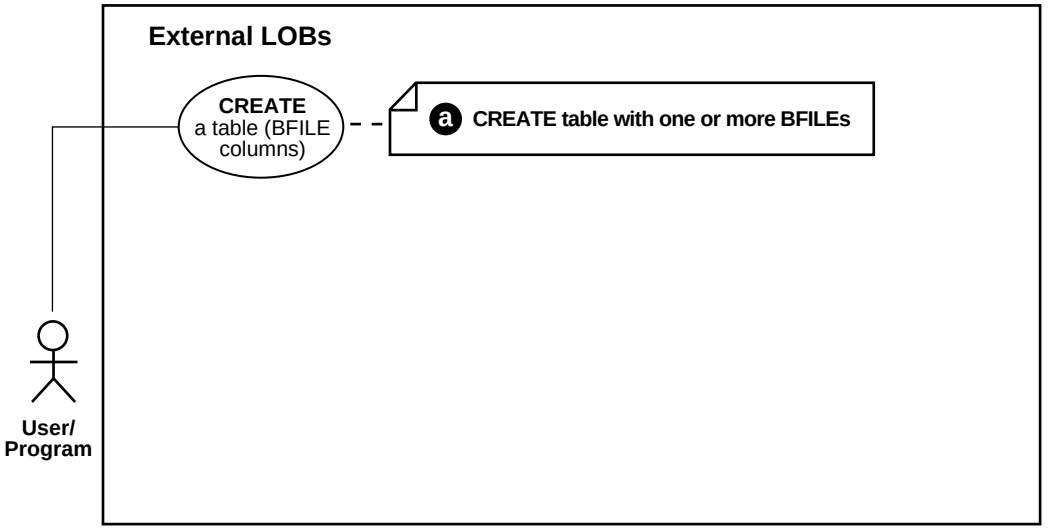
b. `BFILE`s may be attributes of an object type — see "CREATE a Table of an Object Type with a BFILE Attribute" on page 5-16

c. `BFILE`s may be contained within a nested table — see "CREATE a Table with a Nested Table Containing a BFILE" on page 5-19

In all cases SQL DDL is used — to define `BFILE` columns in a table and `BFILE` attributes in an object type.

CREATE a Table Containing a BFILE

Figure 5–3 Use Case Diagram: CREATE a table containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The heart of our hypothetical application is the table `Multimedia_tab`. The varied types which make up the columns of this table make it possible to collect together the many different kinds multimedia elements used in the composition of clips.

Example: Create a Table Containing a BFILE Using SQL DDL

Note: You may need to set up the following data structures for certain examples to work:

```
CONNECT system/manager;
DROP USER samp CASCADE;
DROP DIRECTORY AUDIO_DIR;
DROP DIRECTORY FRAME_DIR;
DROP DIRECTORY PHOTO_DIR;

CREATE USER samp identified by samp;
GRANT CONNECT, RESOURCE to samp;
CREATE DIRECTORY AUDIO_DIR AS '/tmp/';
CREATE DIRECTORY FRAME_DIR AS '/tmp/';
CREATE DIRECTORY PHOTO_DIR AS '/tmp/';
GRANT READ ON DIRECTORY AUDIO_DIR to samp;
GRANT READ ON DIRECTORY FRAME_DIR to samp;
GRANT READ ON DIRECTORY PHOTO_DIR to samp;

CREATE TABLE VoiceoverLib_tab of Voiced_typ (
Script DEFAULT EMPTY_CLOB(),
   CONSTRAINT TakeLib CHECK (Take IS NOT NULL),
   Recording DEFAULT NULL
);
CONNECT samp/samp
CREATE TABLE a_table (blob_col BLOB);
CREATE TYPE Voiced_typ AS OBJECT (
   Originator      VARCHAR2(30),
   Script          CLOB,
   Actor           VARCHAR2(30),
   Take            NUMBER,
   Recording       BFILE );
```

Note (continued):

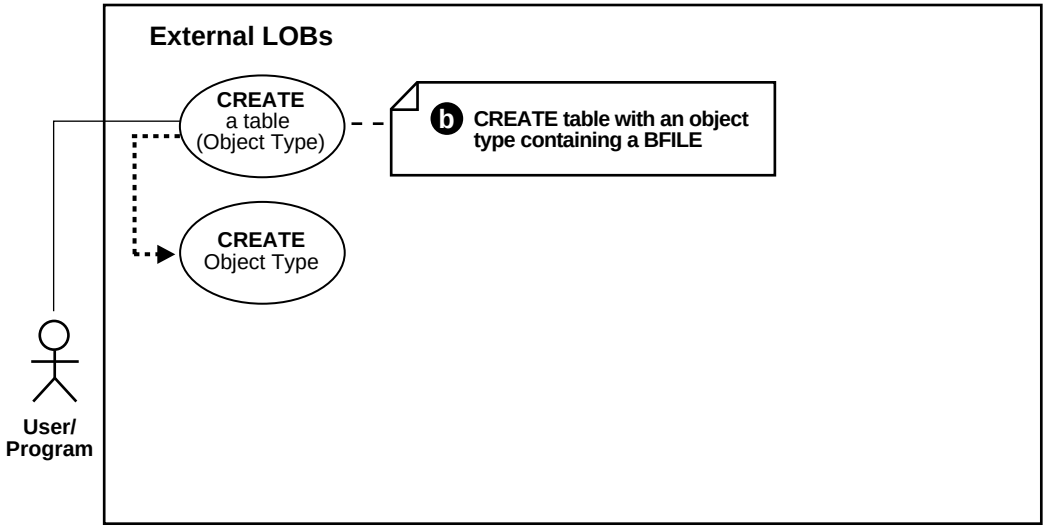
```
CREATE TYPE InSeg_typ AS OBJECT (
   Segment         NUMBER,
   Interview_Date  DATE,
   Interviewer     VARCHAR2(30),
   Interviewee     VARCHAR2(30),
   Recording       BFILE,
   Transcript      CLOB );

CREATE TYPE InSeg_tab AS TABLE of InSeg_typ;

CREATE TYPE Map_typ AS OBJECT (
   Region          VARCHAR2(30),
   NW              NUMBER,
   NE              NUMBER,
   SW              NUMBER,
   SE              NUMBER,
   Drawing         BLOB,
   Aerial          BFILE);
CREATE TABLE Map_Libtab of Map_typ;
CREATE TABLE Voiceover_tab of Voiced_typ (
Script DEFAULT EMPTY_CLOB(),
   CONSTRAINT Take CHECK (Take IS NOT NULL),
   Recording DEFAULT NULL);
```

Because you can use SQL DDL directly to create a table containing one or more LOB columns, it is not necessary to use the DBMS_LOB package.

```
CREATE TABLE Multimedia_tab (
   Clip_ID         NUMBER NOT NULL,
   Story           CLOB default EMPTY_CLOB(),
   FLSub           NCLOB default EMPTY_CLOB(),
   Photo           BFILE default NULL,
   Frame           BLOB default EMPTY_BLOB(),
   Sound           BLOB default EMPTY_BLOB(),
   Voiced_ref      REF Voiced_typ,
   InSeg_ntab      InSeg_tab,
   Music           BFILE default NULL,
   Map_obj         Map_typ
) NESTED TABLE InSeg_ntab STORE AS InSeg_nestedtab;
```

CREATE a Table of an Object Type with a BFILE Attribute

Figure 5–4 Use Case Diagram: CREATE a table containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

As shown in the diagram, you must create the object type that contains the `BFILE` attributes before you can proceed to create a table that makes use of that object type.

Our example application contains examples of two different ways in which object types can contain `BFILE`s:

- `Multimedia_tab` contains a column `Voiced_ref` that references row objects in the table `VoiceOver_tab` which is based on the type `Voiced_typ`. This type contains two kinds of `LOB`s — a `CLOB` to store the script that's read by the actor, and a `BFILE` to hold the audio recording.

- The table `Multimedia_tab` contains a column `Map_obj` that contains column objects of the type Map_typ. This type utilizes the `BFILE` datatype for storing aerial pictures of the region.

Example: Create a Table of an Object Type with a BFILE Attribute Using SQL DDL

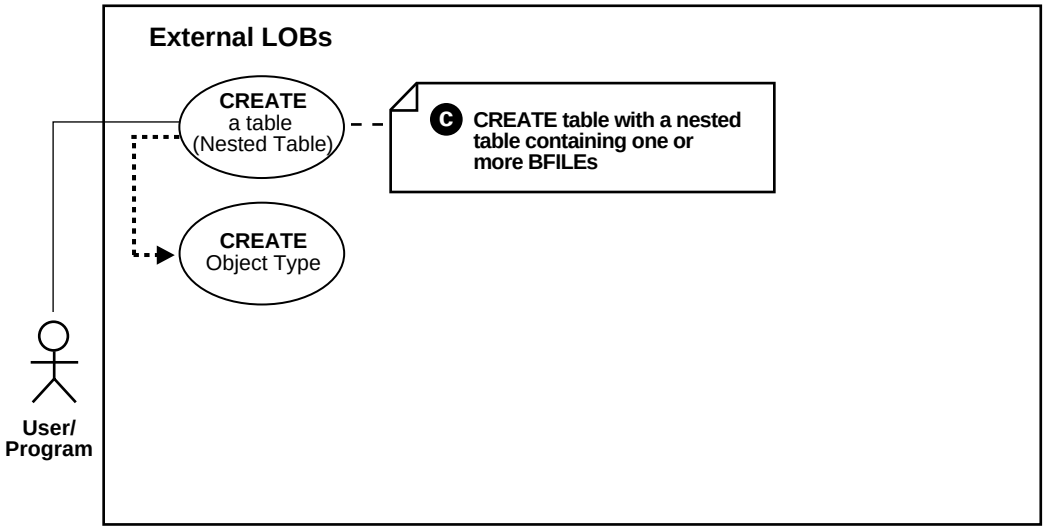
```
/* Create type Voiced_typ as a basis for tables that can contain recordings of
   voice-over readings using SQL DDL: */
CREATE TYPE Voiced_typ AS OBJECT
(
   Originator      VARCHAR2(30),
   Script          CLOB,
   Actor           VARCHAR2(30),
   Take            NUMBER,
   Recording       BFILE
);

/* Create table Voiceover_tab Using SQL DDL: */
CREATE TABLE Voiceover_tab OF Voiced_typ
(
    Script DEFAULT EMPTY_CLOB(),
    CONSTRAINT Take CHECK (Take IS NOT NULL),
    Recording DEFAULT NULL
);

/* Create Type Map_typ using SQL DDL as a basis for the table that will contain
   the column object: */
CREATE TYPE Map_typ AS OBJECT (
   Region          VARCHAR2(30),
   NW              NUMBER,
   NE              NUMBER,
   SW              NUMBER,
   SE              NUMBER,
   Drawing         BLOB,
   Aerial          BFILE
);

/* Create support table MapLib_tab as an archive of maps using SQL DDL: */
CREATE TABLE Map_tab of MapLib_typ;
```

For more information see:

— *Oracle8i SQL Reference* for a complete specification of the syntax for using LOBs in the DDL commands CREATE TYPE and ALTER TYPE with BLOB, CLOB, and BFILE attributes (noting that NCLOBs cannot be attributes of an object type).

CREATE a Table with a Nested Table Containing a BFILE

Figure 5–5 Use Case Diagram: CREATE a Table with a Nested Table Containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

As shown in the diagram, you must create the object type that contains the BFILE attributes before you can proceed to create a nested table based on that object type.

In our example, `Multimedia_tab` contains a nested table `Inseg_ntab` that is based on the type `InSeg_typ`. This type makes use of two LOB datatypes — a `BFILE` for audio recordings of the interviews, and a `CLOB` should the user wish to make transcripts of the recordings.

We have already described how to create a table with BFILE columns (see "CREATE a Table Containing a BFILE" on page 5-13), so here we only describe the SQL DDL syntax the creating the underlying type:

Example: Create a Table with a Nested Table Containing a BFILE Using SQL DDL

Because you use SQL DDL directly to create a table, the DBMS_LOB package is not relevant.

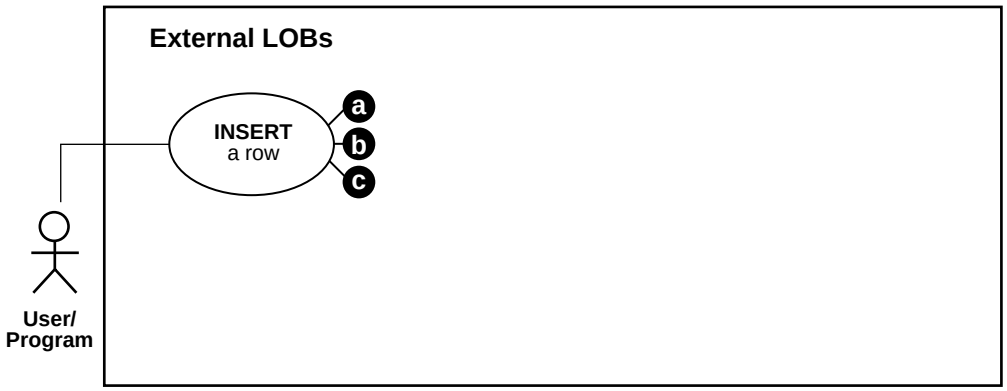
```
CREATE TYPE InSeg_typ AS OBJECT
(
   Segment            NUMBER,
   Interview_Date     DATE,
   Interviewer        VARCHAR2(30),
   Interviewee        VARCHAR2(30),
   Recording          BFILE,
   Transcript         CLOB
);
```

The actual embedding of the nested table is accomplished when the structure of the containing table is defined. In our example, this is effected by means of the following statement at the time that Multimedia_tab is created.

```
NESTED TABLE InSeg_ntab STORE AS InSeg_nestedtab;
```

Three Ways to Insert a Row Containing a BFILE

Figure 5–6 Use Case Diagram: Three Ways to Insert a Row Containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Note that before you insert, you must initialize the `BFILE` either to `NULL` or to a directory alias and filename.

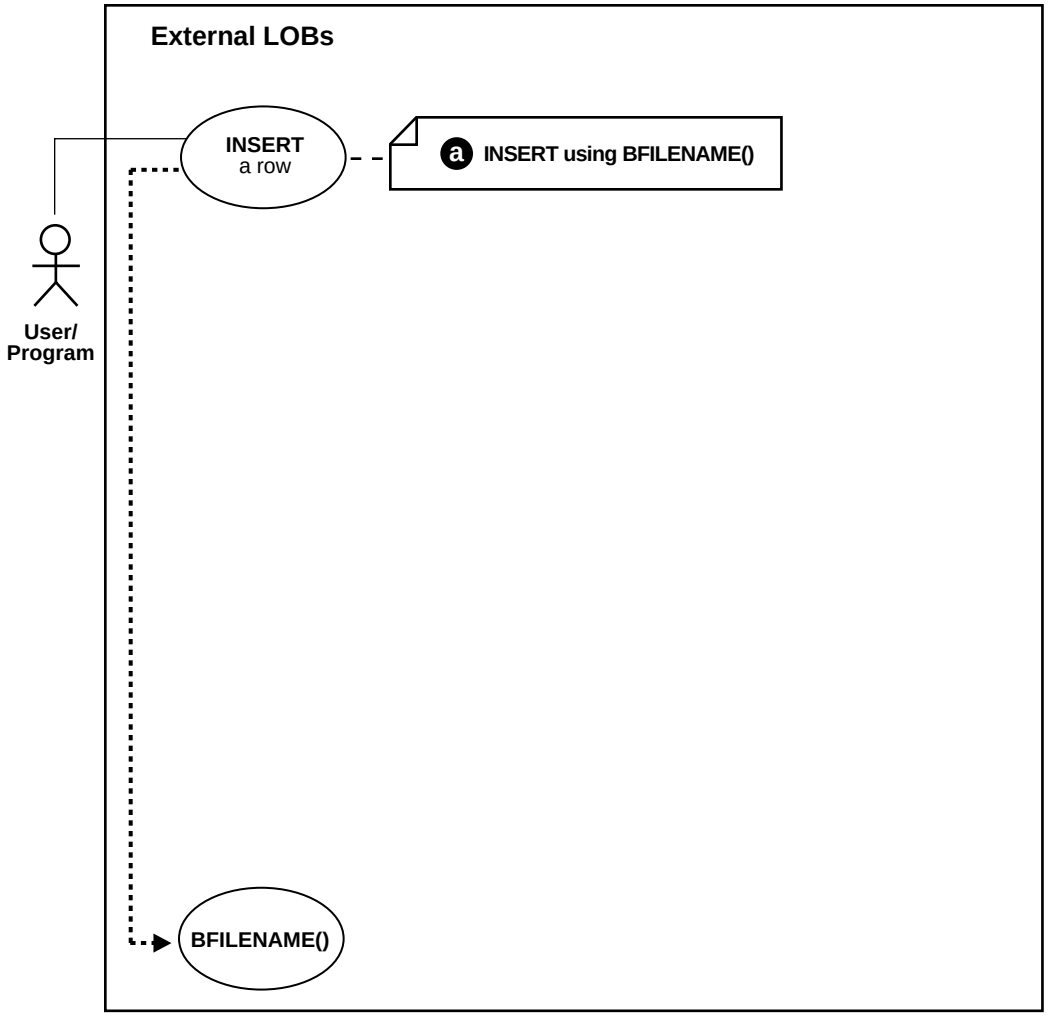
a. "INSERT a Row by means of BFILENAME()" on page 5-22

b. "INSERT a Row Containing a BFILE as SELECT" on page 5-30

c. "INSERT a Row Containing a BFILE by Initializing a BFILE Locator" on page 5-30

INSERT a Row by means of BFILENAME()

Figure 5–7 Use Case Diagram: INSERT a Row by means of BILENAME()



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The BFILENAME() function should be called as part of a SQL INSERT to initialize a BFILE column or attribute for a particular row by associating it with a physical file in the server's filesystem.

The DIRECTORY object represented by the directory_alias parameter to this function need not already be defined using SQL DDL before the BFILENAME() function is called in SQL DML or a PL/SQL program. However, the directory object and operating system file must exist by the time you actually use the BFILE locator (for example, as having been used as a parameter to an operation such as OCILobFileOpen(), DBMS_LOB.FILEOPEN(), OCILobOpen(), or DBMS_LOB.OPEN()).

Note that BFILENAME() does not validate privileges on this DIRECTORY object, or check if the physical directory that the DIRECTORY object represents actually exists. These checks are performed only during file access using the BFILE locator that was initialized by the BFILENAME() function.

You can use BFILENAME() as part of a SQL INSERT and UPDATE statement to initialize a BFILE column. You can also use it to initialize a BFILE locator variable in a PL/SQL program, and use that locator for file operations. However, if the corresponding directory alias and/or filename does not exist, then PL/SQL DBMS_LOB routines that use this variable will generate errors.

The directory_alias parameter in the BFILENAME() function must be specified taking case-sensitivity of the directory name into consideration.

See Also: "DIRECTORY Name Specification". on page 5-7

Example: Insert a Row by means of BFILENAME() Using SQL

```
/* Note that this is the same insert statement as applied to internal persistent
   LOBs but with the BFILENAME() function added to initialize the BFILE columns:
*/

INSERT INTO Multimedia_tab VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(),
```

```
                                    BFILENAME('PHOTO_DIR', 'LINCOLN_PHOTO'),
                                    EMPTY_BLOB(), EMPTY_BLOB(),
                                    VOICED_TYP('Abraham Lincoln', EMPTY_CLOB(),
                                            'James Earl Jones', 1, NULL),
                                    NULL, BFILENAME('AUDIO_DIR',
                                                    'LINCOLN_AUDIO'),
                                    MAP_TYP('Gettysburg', 23, 34, 45, 56,
                                            EMPTY_BLOB(), NULL));
```

Example: Insert a Row by means of BFILENAME() Using C (OCI)

```
/* Insert a row using BFILENAME: */
void insertUsingBfilename(svchp, stmthp, errhp)
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
{
  text  *insstmt =
     (text *) "INSERT INTO Multimedia_tab VALUES (3, EMPTY_CLOB(),
         EMPTY_CLOB(), BFILENAME('PHOTO_DIR', 'Lincoln_photo'),
         EMPTY_BLOB(), EMPTY_BLOB(),
         VOICED_TYP('Abraham Lincoln', EMPTY_CLOB(),
                    'James Earl Jones', 1, NULL),
         NULL, BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
         MAP_TYP('Gettysburg', 23, 34, 45, 56, EMPTY_BLOB(), NULL))";

  /* Prepare the SQL statement */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt, (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
  /* Execute the SQL statement */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));
}
```

Example: Insert a Row by means of BFILENAME() Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BFILE-INSERT.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
```

```
WORKING-STORAGE SECTION.

01  USERID         PIC X(11) VALUES "USER1/USER1".
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-INSERT.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

    EXEC SQL
         INSERT INTO MULTIMEDIA_TAB (CLIP_ID, PHOTO)
          VALUES (1, BFILENAME('PHOTO_DIR', 'LINCOLN_PHOTO'))
    END-EXEC.

    EXEC SQL
         COMMIT WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNRD.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.
```

Example: Insert a Row by means of BFILENAME() Using C++ (Pro*C/C++)

```
#include <oci.h>
```

```
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void BFILENAMEInsert_proc()
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  /* Delete any existing row: */
  EXEC SQL DELETE FROM Multimedia_tab WHERE Clip_ID = 1;
  /* Insert a new row using the BFILENAME() function for BFILEs: */
  EXEC SQL INSERT INTO Multimedia_tab
    VALUES (1, EMPTY_CLOB(), EMPTY_CLOB(),
            BFILENAME('PHOTO_DIR', 'Lincoln_photo'),
            EMPTY_BLOB(), EMPTY_BLOB(), NULL,
            InSeg_tab(InSeg_typ(1, NULL, 'Ted Koppell', 'Abraham Lincoln',
                      BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
                      EMPTY_CLOB())),
            BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
            Map_typ('Moon Mountain', 23, 34, 45, 56, EMPTY_BLOB(),
                    BFILENAME('PHOTO_DIR', 'Lincoln_photo')));
  printf("Inserted %d row\n", sqlca.sqlerrd[2]);
}

void main()
{
   char *samp = "samp/samp";
   EXEC SQL CONNECT :samp;
   BFILENAMEInsert_proc();
   EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Insert a Row by means of BFILENAME() Using Visual Basic (OO4O)

```
Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
```

```
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value
OraDyn.AddNew

OraDyn.Fields("Clip_ID").value = 1
OraDyn.Fields("Story").value = Empty 'This is equivalent to EMPTY_BLOB() in SQL
OraDyn.Fields("FLSub").value = Empty
'Initialize BFile Data:
OraPhoto.Directory = "PHOTO_DIR"
OraPhoto.FileName = "LINCOLN_PHOTO"
OraDyn.Fields("Frame").Value = Empty
OraDyn.Fields("Sound").Value = Empty
'Initialize BFile Data:
OraMusic.DirectoryName = "AUDIO_DIR"
OraMusic.FileName = "LINCOLN_AUDIO"
OraDyn.Edit
OraDyn.Update
'Add the row to the table
```

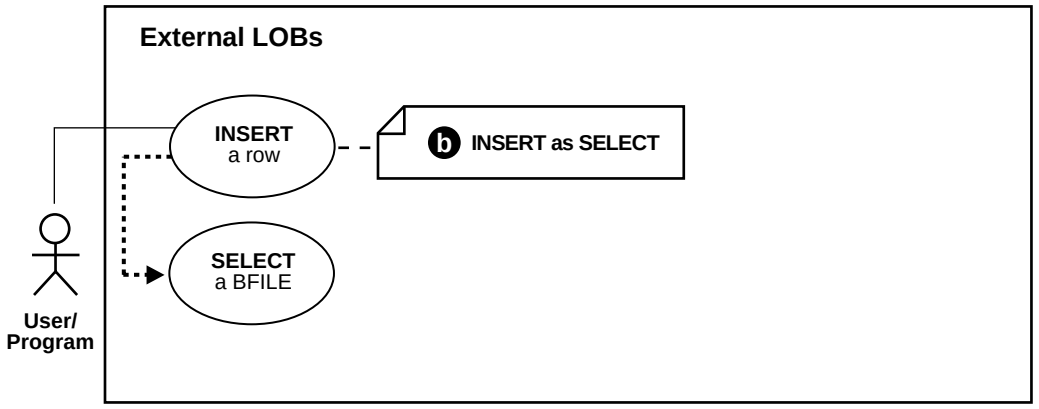
Example: Insert a Row by means of BFILENAME() Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_21
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
```

```
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {

       stmt.execute("INSERT INTO multimedia_tab
          VALUES (99, EMPTY_CLOB(), EMPTY_CLOB(),
          BFILENAME ('PHOTO_DIR','Lincoln_photo'),
          EMPTY_BLOB(), EMPTY_BLOB(),
          (SELECT REF(Vref) FROM Voiceover_tab Vref
             WHERE Actor = 'James Earl Jones'), NULL,
          BFILENAME('AUDIO_DIR', 'Lincoln_audio'),
          MAP_TYP('Gettysburg', 23, 34, 45, 56, EMPTY_BLOB(), NULL))");

   // Commit the transaction:
   conn.commit();
    }
    catch (SQLException e)
    {
   e.printStackTrace();
    }
  }
}
```

INSERT a Row Containing a BFILE as SELECT

Figure 5–8 Use Case Diagram: INSERT a Row Containing a BFILE as SELECT



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

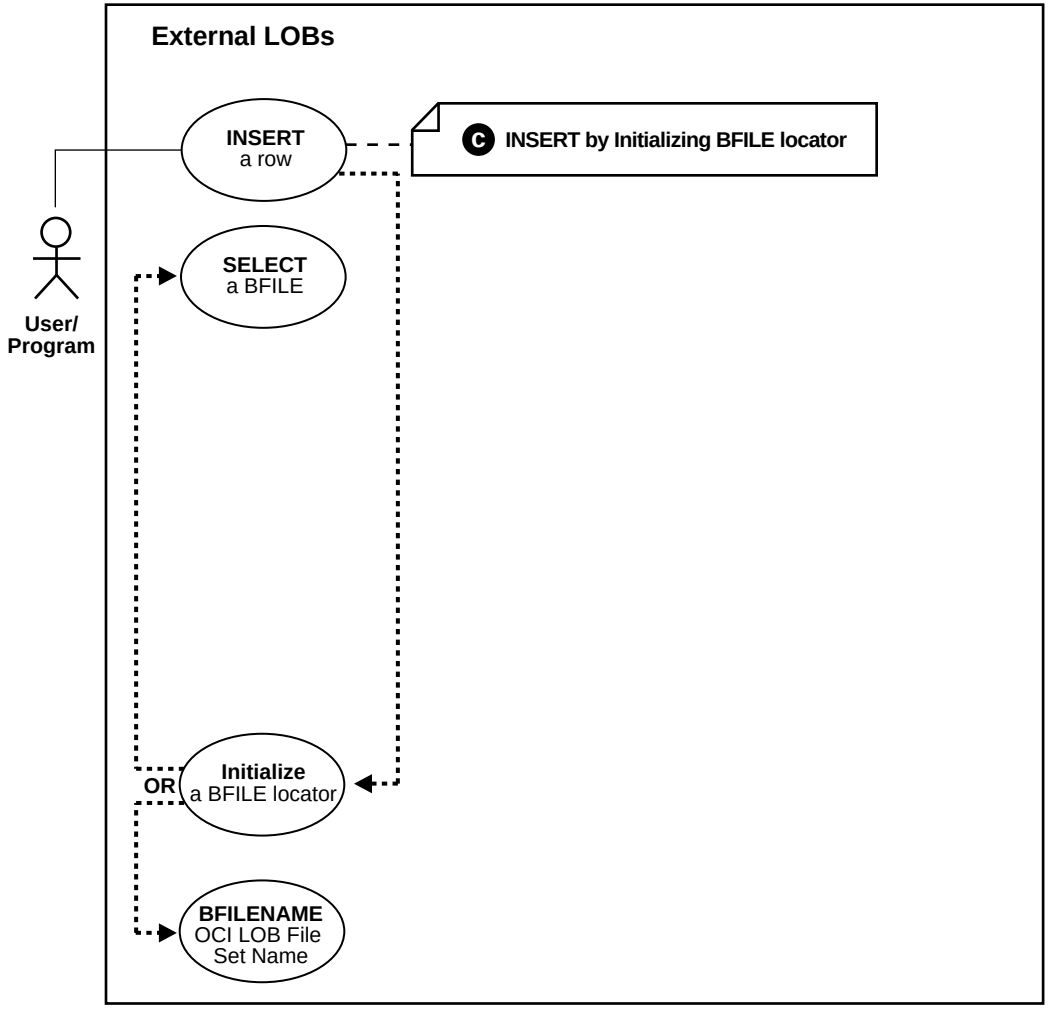
With regard to LOBs, one of the advantages of utilizing an object-relational approach is that you can define a type as a common template for related tables. For instance, it makes sense that both the tables that store archival material and the working tables that use those libraries share a common structure. The following code fragment is based on the fact that a library table `VoiceoverLib_tab` is of the same type (`Voiced_typ`) as `Voiceover_tab` referenced by the `Voiced_ref` column of the `Multimedia_tab` table. It inserts values from the library table into `Multimedia_tab` by means of a `SELECT` operation.

Example: Insert a Row Containing a BFILE as Select Using SQL

```
INSERT INTO Voiceover_tab
  (SELECT * from VoiceoverLib_tab
     WHERE Take = 12345);
```

INSERT a Row Containing a BFILE by Initializing a BFILE Locator

Figure 5–9 Use ***Case Diagram: INSERT a Row by Initializing a BFILE Locator***



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

Note that you must initialize the `BFILE` locator bind variable to a directory alias and filename before issuing the insert statement. In this case we insert a `Photo` from an operating system source file (`PHOTO_DIR`).

- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using PL/SQL" on page 5-31
- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using C (OCI)" on page 5-31
- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using C (OCI)" on page 5-31
- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using C++ (Pro*C/C++)" on page 5-34
- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using Visual Basic (OO4O)" on page 5-35
- "Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using Java (JDBC)" on page 5-35

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using PL/SQL

```
DECLARE
  /* Initialize the BFILE locator: */
  Lob_loc  BFILE := BFILENAME('PHOTO_DIR', 'Washington_photo');
BEGIN
    INSERT INTO Multimedia_tab (Clip_ID, Photo) VALUES (3, Lob_loc);
    COMMIT;
END;
```

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using C

(OCI)

```
/* Insert a row using BFILE Locator: */
void insertUsingBfileLocator(envhp, svchp, stmthp, errhp)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
  text  *insstmt =
     (text *) "INSERT INTO Multimedia_tab (Clip_ID, Photo)
         VALUES (3, :Lob_loc)";
  OCIBind *bndhp;
  OCILobLocator *Lob_loc;
  OraText *Dir = (OraText *)"PHOTO_DIR", *Name = (OraText *)"Washington_photo";


  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, insstmt,  (ub4)
                                  strlen((char *) insstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  * Allocate Locator resources: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0))

  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (dvoid *) &Lob_Loc, (sb4) 0,  SQLT_BFILE,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

  /* Execute the SQL statement: */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

 /* Free LOB resources: */
  OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}
```

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-INSERT-INIT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "USER1/USER1".
01  TEMP-BLOB       SQL-BLOB.
01  SRC-BFILE       SQL-BFILE.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  DIR-IND         PIC S9(4) COMP.
01  FNAME-IND       PIC S9(4) COMP.
01  AMT             PIC S9(9) COMP.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-INSERT-INIT.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator:
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* Set up the directory and file information:
     MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_photo" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

* Set the directory alias and filename in locator:
     EXEC SQL
        LOB FILE SET :SRC-BFILE DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.
```

```
            EXEC SQL
                 INSERT INTO MULTIMEDIA_TAB (CLIP_ID, PHOTO)
                 VALUES (6, :SRC-BFILE)
            END-EXEC.

            EXEC SQL COMMIT WORK END-EXEC.
            EXEC SQL FREE :SRC-BFILE END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void insertBFILELocator_proc()
{
  OCIBFileLocator *Lob_loc;
  char *Dir = "PHOTO_DIR", *Name = "Washington_photo";
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Allocate the input Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  /* Set the Directory and Filename in the Allocated (Initialized) Locator: */
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  EXEC SQL INSERT INTO Multimedia_tab (Clip_ID, Photo) VALUES (4, :Lob_loc);
  /* Release resources held by the Locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  insertBFILELocator_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using Visual Basic (OO4O)

```
Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile

Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value

'Edit the first row and initiliaze the "Photo" column:
OraDyn.Edit
OraPhoto.DirectoryName = "PHOTO_DIR"
OraPhoto.Filename = "Washington_photo"
OraDynaset.Update
```

Example: Insert a Row Containing a BFILE by Initializing a BFILE Locator Using Java (JDBC)

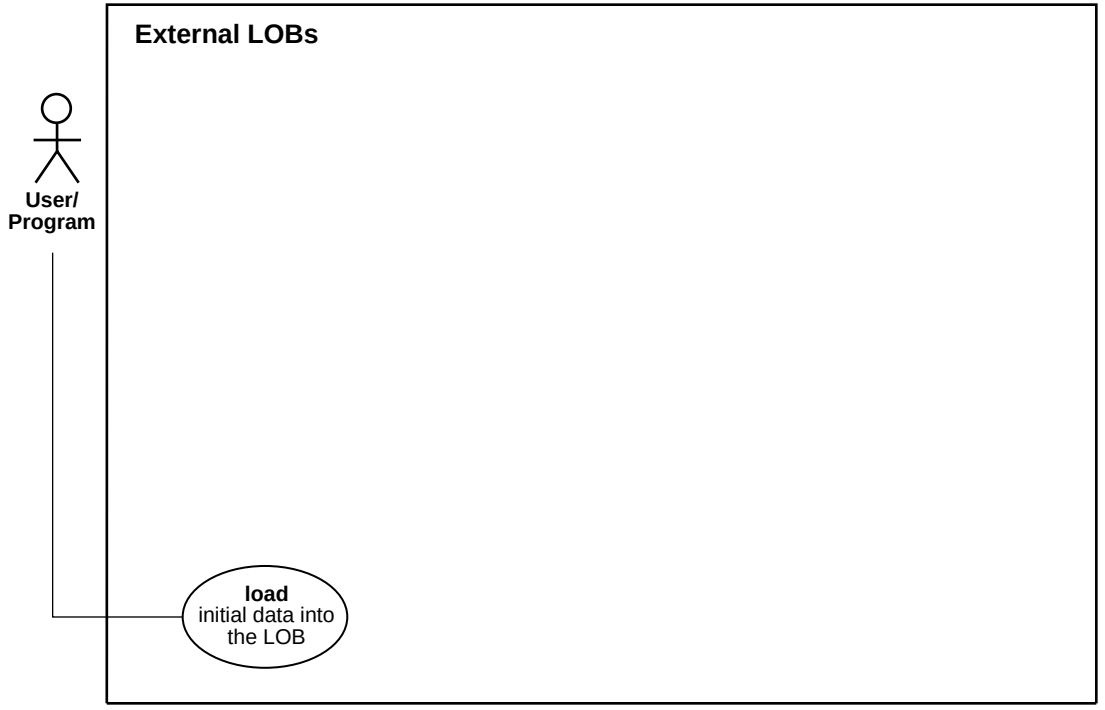
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
```

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_26
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       OracleCallableStatement cstmt = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('PHOTO_DIR', 'Washington_photo') FROM DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);
       }

       // Prepare a CallableStatement to OPEN the LOB for READWRITE:
       cstmt = (OracleCallableStatement) conn.prepareCall (
          "INSERT INTO   multimedia_tab (clip_id, photo) VALUES (3, ?)");
       cstmt.setBFILE(1, src_lob);
```

```
        cstmt.execute();

    //Close the statements and commit the transaction:
    stmt.close();
    cstmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

Load External LOB (BFILE) Data into a Table

Figure 5–10 Use Case Diagram: Load the Initial Data into the External LOB



To refer to the table of all basic operations having to do with Internal Persistent LOBs see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The `BFILE` datatype stores unstructured binary data in operating-system files outside of the database. A `BFILE` column or attribute stores a file locator that points to a server-side external file containing the data

Note: A particular file which is to be loaded as a BFILE does not have to actually exist at the time of loading.

The SQL Loader assumes that the necessary directory objects (a logical alias name for a physical directory on the server's filesystem) have already been created.

For more information on BFILES: See the *Oracle8i Application Developer's Guide - Fundamentals*

A control file field corresponding to a BFILE column consists of 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.

The following two examples illustrate the loading of BFILES. In the first example only the file name is specified dynamically. In the second example, the BFILE and the DIRECTORY OBJECT are specified dynamically.

Note: You may need to set up the following data structures for certain examples to work:

```
CONNECT system/manager
GRANT CREATE ANY DIRECTORY to samp;
CONNECT samp/samp
CREATE OR REPLACE DIRECTORY detective_photo as '/tmp';
CREATE OR REPLACE DIRECTORY photo_dir as '/tmp';
```

Control File:

```
LOAD DATA
INFILE sample9.dat
INTO TABLE Multimedia_tab
FIELDS TERMINATED BY ','
(Clip_ID     INTEGER EXTERNAL(5),
 FileName    FILLER CHAR(30),
 Photo       BFILE(CONSTANT "DETECTIVE_PHOTO", FileName))
```

Data file (sample9.dat):

```
007,/tmp/JamesBond.jpeg,
```

```
008,/tmp/SherlockHolmes.jpeg,
009,/tmp/MissMarple.jpeg,
```

Note:

`Clip_ID` defaults to (255) if a size is not specified; it is mapped to the file names in the datafile. `Detectivel_dir` is the directory where all the files are stored (`Detectivel_dir` is a directory object created previously).

Control File:

```
LOAD DATA
INFILE sample10.dat
INTO TABLE Multimedia_tab
replace
FIELDS TERMINATED BY ','
(
 Clip_ID   INTEGER EXTERNAL(5),
 Photo     BFILE (DirName, FileName),
 FileName  FILLER CHAR(30),
 DirName   FILLER CHAR(30)
)
```

Data file (sample10.dat):

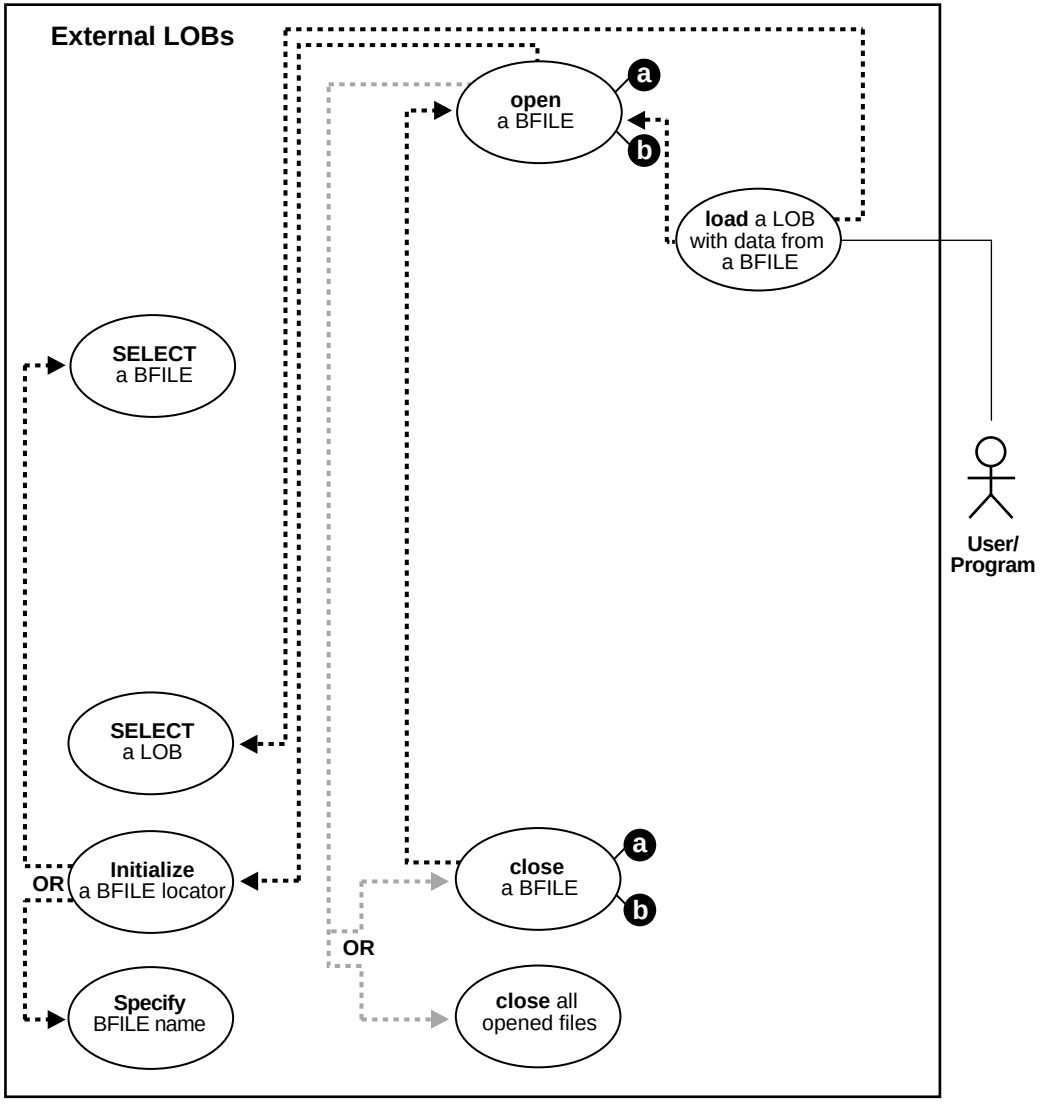
```
007,JamesBond.jpeg,DETECTIVE_PHOTO,
008,SherlockHolmes.jpeg,DETECTIVE_PHOTO,
009,MissMarple.jpeg,PHOTO_DIR,
```

Note:

`DirName FILLER CHAR (30)` is mapped to the datafile field containing the directory name corresponding to the file being loaded.

Load a LOB with Data from a BFILE

Figure 5–11 Use Case Diagram: Load a LOB with data from a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

In using the OCI, or any of the programmatic environments that access OCI functionality, character set conversions are implicitly performed when translating from one character set to another. However, no implicit translation is ever performed from binary data to a character set. When you use the `loadfromfile` operation to populate a `CLOB` or `NCLOB`, you are populating the `LOB` with binary data from the `BFILE`. In that case, you will need to perform character set conversions on the `BFILE` data before executing `loadfromfile`.

The example procedure assumes that there is an operating system source file (`AUDIO_DIR`) that contains the `LOB` data to be loaded into the target `LOB` (`Music`).

- "Example: Load a LOB with Data from a BFILE Using PL/SQL (DBMS_LOB Package)" on page 5-42
- "Example: Load a LOB with Data from a BFILE Using C (OCI)" on page 5-43
- "Example: Load a LOB with Data from a BFILE Using COBOL (Pro*COBOL)" on page 5-44
- "Example: Load a LOB with Data from a BFILE Using C++ (Pro*C/C++)" on page 5-46
- "Example: Load a LOB with Data from a BFILE Using Visual Basic (OO4O)" on page 5-47
- "Example: Load a LOB with Data from a BFILE Using Java (JDBC)" on page 5-48

Example: Load a LOB with Data from a BFILE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure loadLOBFromBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
   Dest_loc       BLOB;
   Src_loc        BFILE := BFILENAME('AUDIO_DIR', 'Washington_audio');
   Amount         INTEGER := 4000;
BEGIN
    SELECT Music INTO Dest_loc FROM Multimedia_tab
       WHERE Clip_ID = 3
```

```
          FOR UPDATE;
    /* Opening the LOB is mandatory: */
   DBMS_LOB.OPEN(Src_loc, DBMS_LOB.LOB_READONLY);
   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN(Dest_loc, DBMS_LOB.LOB_READWRITE);
   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);
   /* Closing the LOB is mandatory if you have opened it: */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);
   COMMIT;
END;
```

Example: Load a LOB with Data from a BFILE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
OCILobLocator *Lob_loc;
text *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

      /* Define the column being selected */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}

/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx *svchp;
OCIStatement *stmthp;
```

```
OCIError *errhp;
OCIDefine *dfnhp;
OCILobLocator *Lob_loc;
text *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

      /* Define the column being selected */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
     /* Free the locator descriptors */
     OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_BLOB);
     OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_FILE);
}

void loadLobFromBfile(svchp, errhp, dest_loc, src_loc)
OCISvcCtx *svchp;
OCIError *errhp;
OCILobLocator *dest_loc;        /* These locators have been already allocated */
OCILobLocator *src_loc;         /* This is the BFILE locator. */
{
  checkerr(errhp, OCILobFileOpen(svchp, errhp, src_loc,
                                 (ub1)OCI_FILE_READONLY));
  checkerr(errhp, OCILobOpen(svchp, errhp, dest_loc, (ub1)OCI_FILE_READWRITE));
  checkerr (errhp, OCILobLoadFromFile(svchp, errhp, dest_loc, src_loc,
                                      (ub4)4000, (ub4)0, (ub4)0));
  checkerr(errhp, OCILobClose(svchp, errhp, dest_loc));
  checkerr(errhp, OCILobFileClose(svchp, errhp, src_loc));
}
```

Example: Load a LOB with Data from a BFILE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LOAD-BFILE.
```

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "USER1/USER1".
01  DEST-BLOB       SQL-BLOB.
01  SRC-BFILE       SQL-BFILE.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  DIR-IND         PIC S9(4) COMP.
01  FNAME-IND       PIC S9(4) COMP.
01  AMT             PIC S9(9) COMP.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
LOAD-BFILE.

* Allocate and initialize the LOB locators:
     EXEC SQL ALLOCATE :DEST-BLOB END-EXEC.
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

* Populate the BFILE:
     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
     EXEC SQL
          SELECT PHOTO INTO :SRC-BFILE
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
     END-EXEC.

* Open the source BFILE READ ONLY.
* Open the destination BLOB READ/WRITE:
     EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :DEST-BLOB READ WRITE END-EXEC.

* Load BFILE data into the BLOB:
     EXEC SQL
```

```
            LOB LOAD :AMT FROM FILE :SRC-BFILE
            INTO :DEST-BLOB
        END-EXEC.

    * Close the LOBs:
        EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.
        EXEC SQL LOB CLOSE :DEST-BLOB END-EXEC.

    * And free the LOB locators:
     END-OF-BFILE.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
        EXEC SQL FREE :DEST-BLOB END-EXEC.
        EXEC SQL FREE :SRC-BFILE END-EXEC.
        EXEC SQL
             COMMIT WORK RELEASE
        END-EXEC.
        STOP RUN.

     SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Load a LOB with Data from a BFILE Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
```

```
}

void loadLOBFromBFILE_proc()
{
  OCIBlobLocator *Dest_loc;
  OCIBFileLocator *Src_loc;
  char *Dir = "AUDIO_DIR", *Name = "Washington_audio";
  int Amount = 4096;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Initialize the BFILE Locator: */
  EXEC SQL ALLOCATE :Src_loc;
  EXEC SQL LOB FILE SET :Src_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Initialize the BLOB Locator: */
  EXEC SQL ALLOCATE :Dest_loc;
  EXEC SQL SELECT Sound INTO :Dest_loc FROM Multimedia_tab
           WHERE Clip_ID = 3 FOR UPDATE;
  /* Opening the BFILE is Mandatory: */
  EXEC SQL LOB OPEN :Src_loc READ ONLY;
  /* Opening the BLOB is Optional: */
  EXEC SQL LOB OPEN :Dest_loc READ WRITE;
  EXEC SQL LOB LOAD :Amount FROM FILE :Src_loc INTO :Dest_loc;
  /* Closing LOBs and BFILEs is Mandatory if they have been OPENed: */
  EXEC SQL LOB CLOSE :Dest_loc;
  EXEC SQL LOB CLOSE :Src_loc;
  /* Release resources held by the Locators: */
  EXEC SQL FREE :Dest_loc;
  EXEC SQL FREE :Src_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  loadLOBFromBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Load a LOB with Data from a BFILE Using Visual Basic (OO4O)

```
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1.
```

```
Dim OraDyn as OraDynaset, OraDyn2 as OraDynaset, OraPhoto as OraBFile
Dim OraImage as OraLob

chunksize = 32768
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraDyn2 = OraDb.CreateDynaset("select * from Images", ORADYN_DEFAULT)

Set OraPhoto = OraDyn.Fields("Photo").value
Set OraImage = OraDyn2.Fields("Image").value

OraDyn2.Edit
'Load LOB with data from BFILE:
OraImage.CopyFromBFile (OraPhoto)
OraDyn2.Update
```

Example: Load a LOB with Data from a BFILE Using Java (JDBC)

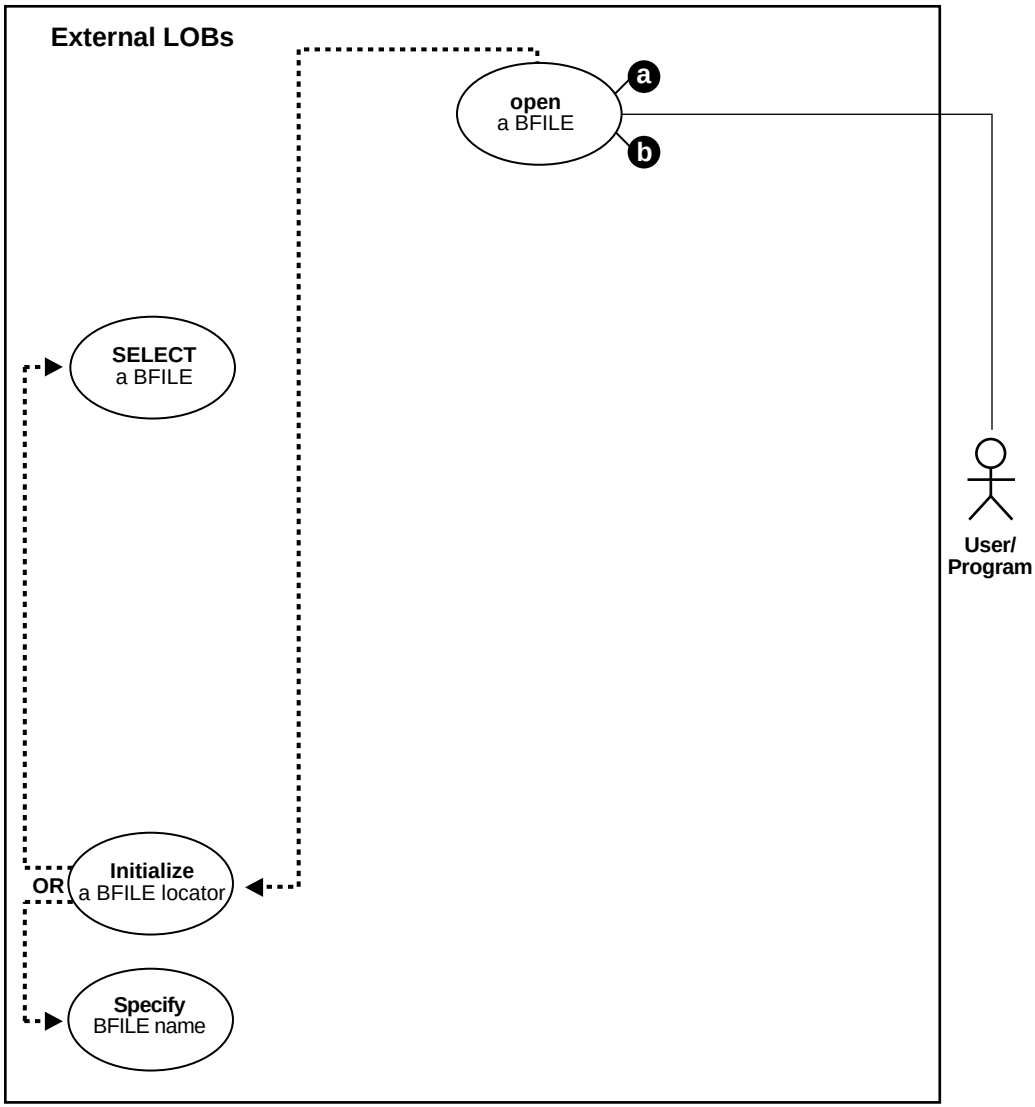
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex2_45
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
```

```
  DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

conn.setAutoCommit (false);

// Create a Statement:
Statement stmt = conn.createStatement ();

try
{
   BFILE src_lob = null;
   BLOB dest_lob = null;
   InputStream in = null;
   OutputStream out = null;
   byte buf[] = new byte[1000];
   ResultSet rset = null;
   OracleCallableStatement cstmt = null;

  // Prepare a CallableStatement to OPEN the LOB for READWRITE:
  cstmt = (OracleCallableStatement) conn.prepareCall (
    "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READWRITE); END;");

   rset = stmt.executeQuery (
      "SELECT BFILENAME('AUDIO_DIR', 'Washington_audio') FROM DUAL");
   if (rset.next())
   {
      src_lob = ((OracleResultSet)rset).getBFILE (1);
      src_lob.openFile();
      in = src_lob.getBinaryStream();
   }

   rset = stmt.executeQuery (
      "SELECT sound FROM multimedia_tab WHERE clip_id = 2 FOR UPDATE");
   if (rset.next())
   {
      dest_lob = ((OracleResultSet)rset).getBLOB (1);

      // Bind the dest_lob to the prepared statement and execute it:
      cstmt.setBLOB(1, dest_lob);
      cstmt.execute();

      // Fetch the output stream for dest_lob:
      out = dest_lob.getBinaryOutputStream();
   }

   int length = 0;
```

```
            int pos = 0;
            while ((in != null) && (out != null) && ((length = in.read(buf)) != -1))
            {
               System.out.println("Pos = " + Integer.toString(pos) +
                  ".  Length = " + Integer.toString(length));
               pos += length;
               out.write(buf, pos, length);
            }

            // Close all streams and file handles:
            in.close();
            out.flush();
            out.close();
            src_lob.closeFile();

            // All OPENed LOBS must be CLOSEd:
            cstmt = (OracleCallableStatement) conn.prepareCall (
               "BEGIN DBMS_LOB.CLOSE(?); END;");
            cstmt.setBLOB(1, dest_lob);
            cstmt.execute();

           // Commit the transaction:
           conn.commit();
         }
         catch (SQLException e)
         {
            e.printStackTrace();
         }
      }
   }
```

Two Ways to Open a BFILE

Figure 5–12 Use Case Diagram: Two Ways to Open a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

As you can see by comparing the code, these alternative methods are very similar. However, while you can continue to use the older `FILEOPEN` form, we *strongly recommend* that you switch to using `OPEN` because this facilitates future extensibility.

a. "Open a BFILE with FILEOPEN" on page 5-53

b. "Open a BFILE with OPEN" on page 5-59

Maximum Number of Open BFILEs

A limited number of `BFILE`s can be open simultaneously per session. The maximum number is specified by using the initialization parameter `SESSION_MAX_OPEN_FILES`.

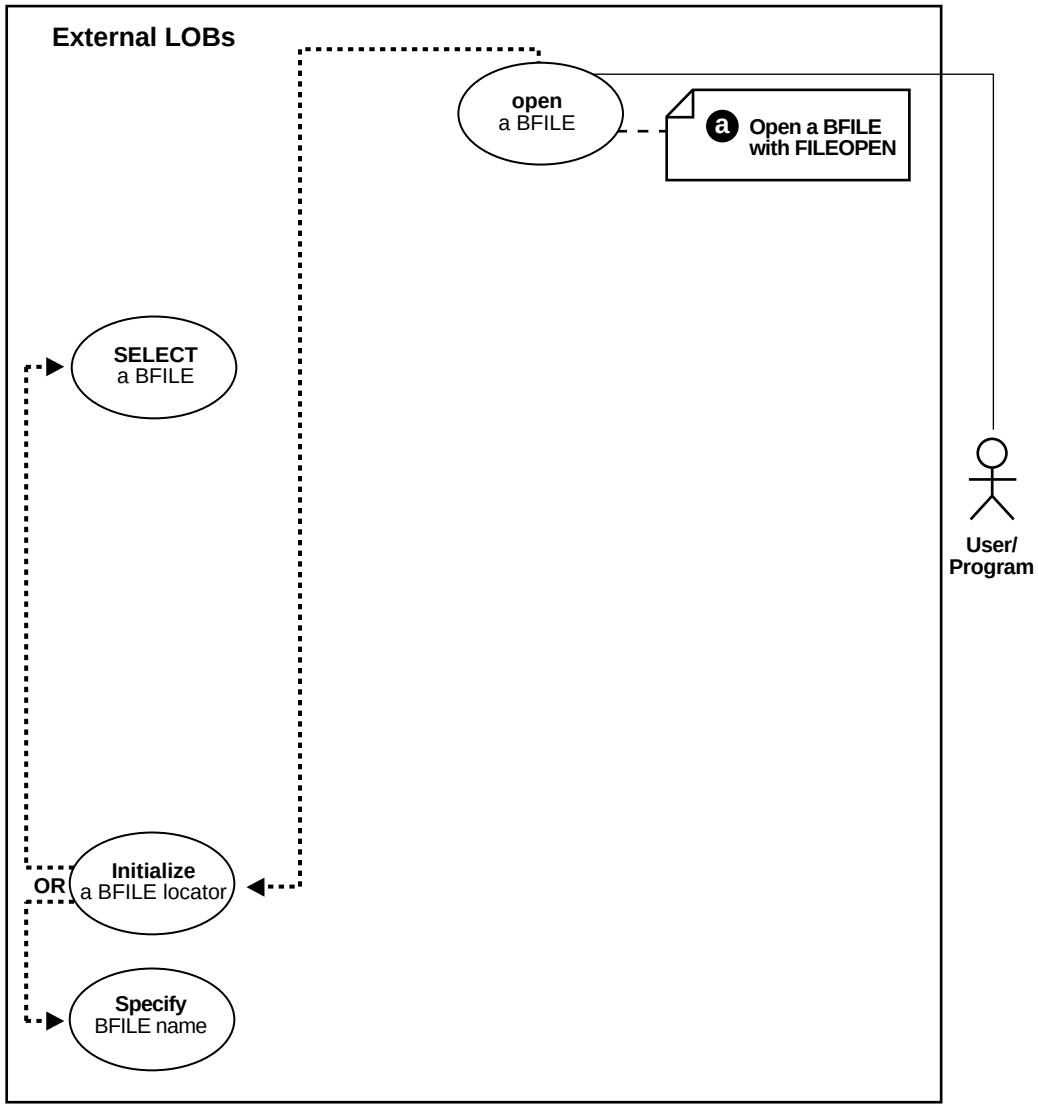
`SESSION_MAX_OPEN_FILES` defines an upper limit on the number of simultaneously open files in a session. The default value for this parameter is 10. That is, a maximum of 10 files can be opened simultaneously per session if the default value is utilized. The database administrator can change the value of this parameter in the `init.ora` file**.** For example:

```
SESSION_MAX_OPEN_FILES=20
```

If the number of unclosed files exceeds the `SESSION_MAX_OPEN_FILES` value then you will not be able to open any more files in the session. To close all open files, use the `FILECLOSEALL` call.

Open a BFILE with FILEOPEN

Figure 5–13 Use Case Diagram: Open a BFILE with FILEOPEN



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

While you can continue to use the older `FILEOPEN` form, we *strongly recommend* that you switch to using `OPEN`, because this facilitates future extensibility. This example opens a `Lincoln_photo` in operating system file `PHOTO_DIR`.

- "Example: Open a BFILE with FILEOPEN Using PL/SQL" on page 5-54
- "Example: Open a BFILE with FILEOPEN Using C (OCI)" on page 5-54
- "Example: Open a BFILE with FILEOPEN Using Visual Basic (OO4O)" on page 5-56
- "Example: Open a BFILE with FILEOPEN Using Java (JDBC)" on page 5-56

Example: Open a BFILE with FILEOPEN Using PL/SQL

```
/* Note that the example procedure openBFILE_procOne is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE openBFILE_procOne IS
   Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
   /* Open the BFILE: */
   DBMS_LOB.FILEOPEN (Lob_loc, DBMS_LOB.FILE_READONLY)
   /* ... Do some processing. */
   DBMS_LOB.FILECLOSE(Lob_loc);
END;
```

Example: Open a BFILE with FILEOPEN Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
OCILobLocator *Lob_loc;
text *selstmt;
```

```
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

      /* Define the column being selected */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 ,
                                     SQLT_BFILE, (dvoid *)0, (ub2 *)0,
                                     (ub2 *)0, OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileOpen(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                               (ub4) OCI_DTYPE_FILE,
                               (size_t) 0, (dvoid **) 0)

     /* Set the bfile locator information */
     checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                                        (OraText *)"PHOTO_DIR",
                                        (ub2)strlen("PHOTO_DIR"),
                                        (OraText *)"Lincoln_photo",
                                        (ub2)strlen("Lincoln_photo"))));
  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                 (ub1)OCI_FILE_READONLY));
  /* ... Do some processing. */
  checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));
```

```
     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

void BfileOpen(envhp, errhp, svchp, stmthp, bfile_loc)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
OCILobLocator *bfile_loc;  /* This is the BFILE locator that is already
                              allocated and initialized. */
{
  checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                 (ub1)OCI_FILE_READONLY));
  /* ... Do some processing. */
  checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));
}
```

Example: Open a BFILE with FILEOPEN Using Visual Basic (OO4O)

Note: At the present time, OO4O only offers `BFILE` opening with `OPEN` (see "Example: Open a BFILE with OPEN Using Visual Basic (OO4O)" on page 5-64).

Example: Open a BFILE with FILEOPEN Using Java (JDBC)

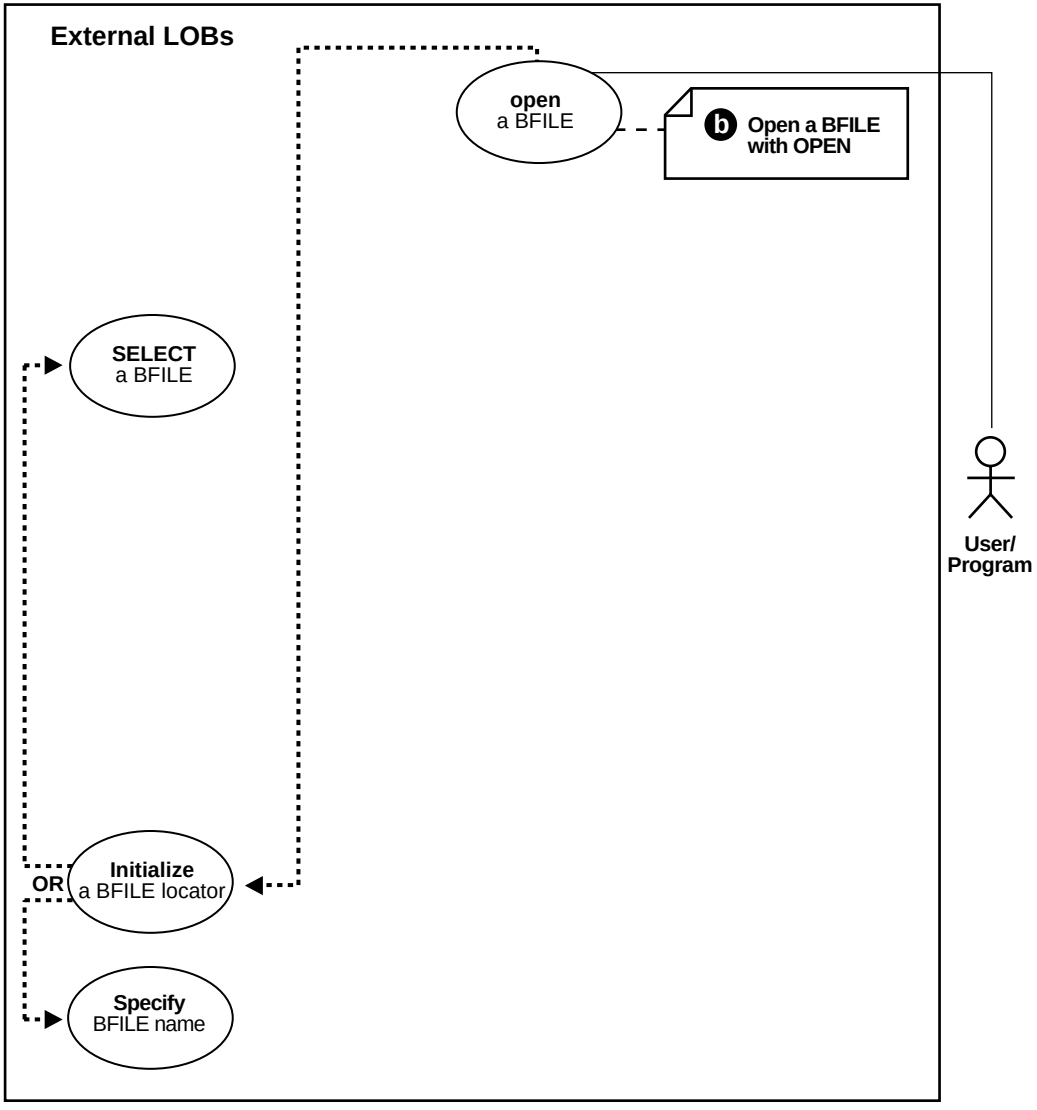
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
```

```
public class Ex4_38
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
       "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
       if (rset.next())
       {
           src_lob = ((OracleResultSet)rset).getBFILE (1);

          // plsql_fileOpen() wraps a call to dbms_lob.fileopen():
          src_lob.plsql_fileOpen();

           System.out.println("The file is now open");
        }

        // Close the BFILE,  statement and connection:
        src_lob.plsql_fileClose();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
```

```
        }
      }
    }
```

Open a BFILE with OPEN

Figure 5–14 Use Case Diagram: Open a BFILE with OPEN



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example opens a Lincoln_photo in operating system file PHOTO_DIR.

- "Example: Open a BFILE with OPEN Using PL/SQL" on page 5-60
- "Example: Open a BFILE with OPEN Using C (OCI)" on page 5-60
- "Example: Open a BFILE with OPEN Using COBOL (Pro*COBOL)" on page 5-62
- "Example: Open a BFILE with OPEN Using C++ (Pro*C/C++)" on page 5-63
- "Example: Open a BFILE with OPEN Using Visual Basic (OO4O)" on page 5-64
- "Example: Open a BFILE with OPEN Using Java (JDBC)" on page 5-64

Example: Open a BFILE with OPEN Using PL/SQL

```
/* Note that the example procedure openBFILE_procTwo is not part of the
   DBMS_LOB package:  */
CREATE OR REPLACE PROCEDURE openBFILE_procTwo IS
  Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
   /* Open the BFILE: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY)
   /* ... Do some processing: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Open a BFILE with OPEN Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
```

```
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileFileOpen(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Set the Bfile Locator Information */
     checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                          (OraText *)"PHOTO_DIR", (ub2)strlen("PHOTO_DIR"),
                          (OraText *)"Lincoln_photo",
                          (ub2)strlen("Lincoln_photo"))));
     checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                               (ub1)OCI_FILE_READONLY));
     /* ... Do some processing. */
     checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));
```

```
    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Example: Open a BFILE with OPEN Using COBOL (Pro*COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. OPEN-BFILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  USERID         PIC X(11) VALUES "USER1/USER1".
 01  SRC-BFILE      SQL-BFILE.
 01  DIR-ALIAS      PIC X(30) VARYING.
 01  FNAME          PIC X(20) VARYING.
 01  ORASLNRD       PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 OPEN-BFILE.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BFILE locator:
     EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* Set up the directory and file information:
     MOVE "AUDIO_DIR" TO DIR-ALIAS-ARR.
     MOVE 9 TO DIR-ALIAS-LEN.
     MOVE "washington_audio" TO FNAME-ARR.
     MOVE 16 TO FNAME-LEN.

* Assign directory alias and file name to BFILE:
     EXEC SQL
          LOB FILE SET :SRC-BFILE
          DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME
     END-EXEC.
```

```
      * Open the BFILE read only:
           EXEC SQL
                LOB OPEN :SRC-BFILE READ ONLY
           END-EXEC.

      * Close the LOB:
           EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

      * And free the LOB locator:
           EXEC SQL FREE :SRC-BFILE END-EXEC.
           EXEC SQL
                COMMIT WORK RELEASE
           END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

Example: Open a BFILE with OPEN Using C++ (Pro*C/C++)

```
/* In Pro*C/C++ there is only one form of OPEN that is used for OPENing
   BFILEs.  There is no FILE OPEN, only a simple OPEN statement: */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
```

```
  exit(1);
}

void openBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  /* Initialize the Locator: */
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* ... Do some processing: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  openBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Open a BFILE with OPEN Using Visual Basic (OO4O)

```
Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab",ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
Set OraPhoto = OraDyn.Fields("Photo").Value

'Go to the last rowand open Bfile for reading:
OraDyn.MoveLast
OraPhoto.Open 'Open Bfile for reading
'Do some processing:
OraPhoto.Close
```

Example: Open a BFILE with OPEN Using Java (JDBC)

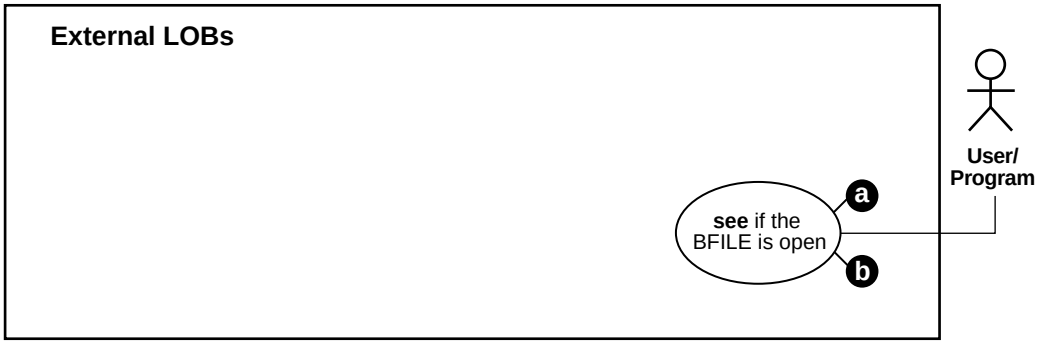
```
// Java IO classes:
```

```
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_41
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
       "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);

          // openFile() delegates to oracle.jdbc.dbaccess.DBAccess.fileOpen():
         src_lob.openFile();
```

```
            System.out.println ("the file is now open");
          }

          // Close the BFILE,  statement and connection:
          src_lob.closeFile();
          stmt.close();
          conn.commit();
          conn.close();
        }
        catch (SQLException e)
        {
          e.printStackTrace();
        }
      }
    }
```

Two Ways to See If a BFILE is Open

Figure 5–15 Use Case Diagram: Two Ways to See If a BFILE is Open



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

As you can see by comparing the code, these alternative methods are very similar. However, while you can continue to use the older `FILEISOPEN` form, we strongly recommend that you switch to using `ISOPEN`, because this facilitates future extensibility.

a. "See If the BFILE is Open with FILEISOPEN" on page 5-69

b. "See If the BFILE is Open Using ISOPEN" on page 5-74

Maximum Number of Open BFILEs

A limited number of `BFILE`s can be open simultaneously per session. The maximum number is specified by using the `SESSION_MAX_OPEN_FILES` initialization parameter.

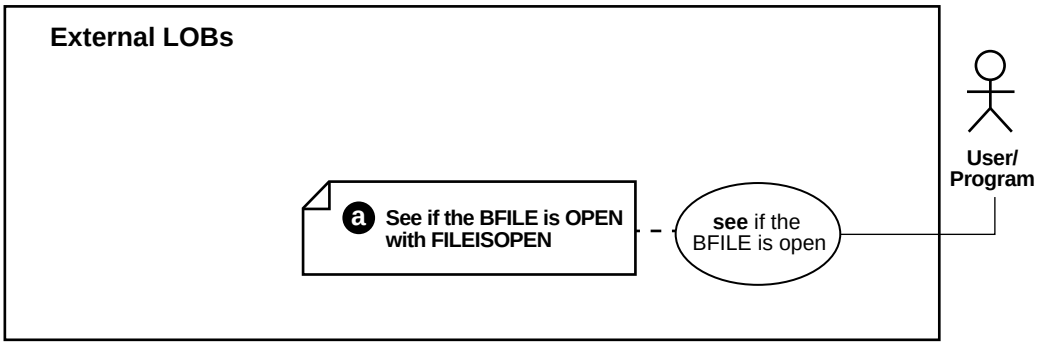
`SESSION_MAX_OPEN_FILES` defines an upper limit on the number of simultaneously open files in a session. The default value for this parameter is 10. That is, a maximum of 10 files can be opened simultaneously per session if the default value is utilized. The database administrator can change the value of this parameter in the `init.ora` file**.** For example:

```
SESSION_MAX_OPEN_FILES=20
```

If the number of unclosed files exceeds the SESSION_MAX_OPEN_FILES value then you will not be able to open any more files in the session. To close all open files, use the FILECLOSEALL call.

See If the BFILE is Open with FILEISOPEN

Figure 5–16 Use Case Diagram: See If the BFILE is Open Using FILEISOPEN



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

While you can continue to use the older `FILEISOPEN` form, we *strongly recommend* that you switch to using `ISOPEN`, because this facilitates future extensibility. his example queries whether the a BFILE associated with `Music` is open that is.

- "Example: See If the BFILE is Open with FILEISOPEN Using PL/SQL (DBMS_LOB Package)" on page 5-70
- "Example: See If the BFILE is Open with FILEISOPEN Using C (OCI)" on page 5-70
- "Example: See If the BFILE is Open with FILEISOPEN Using Visual Basic (OO4O)" on page 5-72
- "Example: See If the BFILE is Open with FILEISOPEN Using Java (JDBC)" on page 5-72

Example: See If the BFILE is Open with FILEISOPEN Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure seeIfOpenBFILE_procOne is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procOne IS
   Lob_loc      BFILE;
   RetVal       INTEGER;
BEGIN
   /* Select the LOB, initializing the BFILE locator: */
   SELECT Music INTO Lob_loc FROM Multimedia_tab
      WHERE Clip_ID = 3;
   RetVal := DBMS_LOB.FILEISOPEN(Lob_loc);
   IF (RetVal = 1)
      THEN
      DBMS_OUTPUT.PUT_LINE('File is open');
   ELSE
      DBMS_OUTPUT.PUT_LINE('File is not open');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: See If the BFILE is Open with FILEISOPEN Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)

OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
```

```
                                        (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                        OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                        (ub4) OCI_DEFAULT));
}
boolean BfileIsOpen(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     boolean flag;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Select the bfile */
     selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
               "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3");
     boolean flag;
     checkerr(errhp, OCILobFileIsOpen(svchp, errhp, bfile_loc,
                                   &flag));
     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
      return(flag);
}
```

Example: See If the BFILE is Open with FILEISOPEN Using Visual Basic (OO4O)

Note: At the present time, OO4O only offers ISOPEN to test whether or not a BFILE is open (see "Example: See If the BFILE is Open with FILEISOPEN Using Visual Basic (OO4O)" on page 5-72).

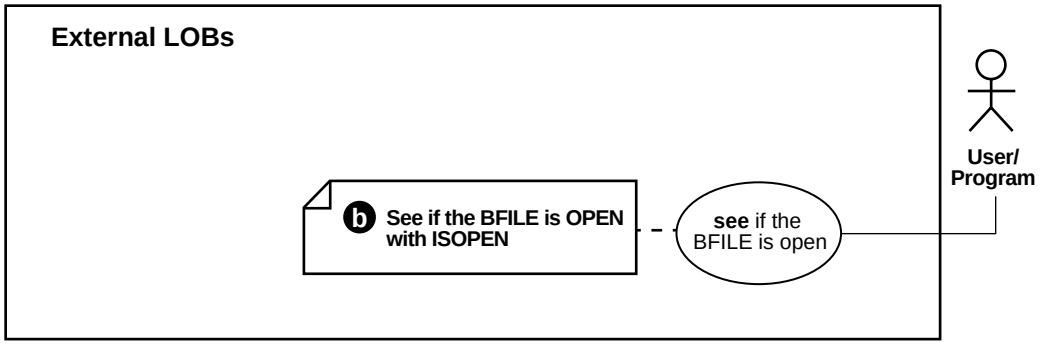
Example: See If the BFILE is Open with FILEISOPEN Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_45
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
```

```
    {
        BFILE src_lob = null;
        ResultSet rset = null;
        Boolean result = null;

        rset = stmt.executeQuery (
           "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
        if (rset.next())
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        result = new Boolean(src_lob.plsql_fileIsOpen());
        System.out.println(
           "result of fileIsOpen() before opening file : " + result.toString());

        src_lob.plsql_fileOpen();

        result = new Boolean(src_lob.plsql_fileIsOpen());
        System.out.println(
           "result of fileIsOpen() after opening file : " + result.toString());

        // Close the BFILE, statement and connection:
        src_lob.plsql_fileClose();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

See If the BFILE is Open Using ISOPEN

Figure 5–17 Use Case Diagram: See If the BFILE is Open Using FILEISOPEN



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example queries whether the a BFILE is open that is associated with `Music`.

- "Example: See If the BFILE is Open with ISOPEN Using PL/SQL (DBMS_LOB Package)" on page 5-75
- "Example: See If the BFILE is Open with ISOPEN Using C (OCI)" on page 5-75
- "Example: See If the BFILE is Open with ISOPEN Using COBOL (Pro*COBOL)" on page 5-76
- "Example: See If the BFILE is Open with ISOPEN Using C++ (Pro*C/C++)" on page 5-78
- "Example: See If the BFILE is Open with ISOPEN Using Visual Basic (OO4O)" on page 5-79
- "Example: See If the BFILE is Open with ISOPEN Using Java (JDBC)" on page 5-80

Example: See If the BFILE is Open with ISOPEN Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure seeIfOpenBFILE_procTwo is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procTwo IS
   Lob_loc      BFILE;
   RetVal       INTEGER;
BEGIN
   /* Select the LOB, initializing the BFILE locator: */
   SELECT Music INTO Lob_loc FROM Multimedia_tab
      WHERE Clip_ID = 3;
   RetVal := DBMS_LOB.ISOPEN(Lob_loc);
   IF (RetVal = 1)
   THEN
      DBMS_OUTPUT.PUT_LINE('File is open');
   ELSE
      DBMS_OUTPUT.PUT_LINE('File is not open');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: See If the BFILE is Open with ISOPEN Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));
```

```
     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
boolean BfileIsOpen(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     boolean flag;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Select the bfile */
     selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
              "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3");


     boolean flag;
     checkerr(errhp, OCILobFileIsOpen(svchp, errhp, bfile_loc,
                                      &flag));
     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
     return(flag);
}
```

Example: See If the BFILE is Open with ISOPEN Using COBOL (Pro*COBOL)

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. BFILE-IS-OPEN.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
```

```
 WORKING-STORAGE SECTION.

 01  USERID          PIC X(11) VALUES "USER1/USER1".
 01  BFILE1          SQL-BFILE.
 01  DIR-ALIAS       PIC X(30) VARYING.
 01  FNAME           PIC X(20) VARYING.
 01  IS-OPEN         PIC S9(9) COMP.
 01  ORASLNRD        PIC 9(4).

     EXEC SQL INCLUDE SQLCA END-EXEC.
     EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
     EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 BFILE-IS-OPEN.

     EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
     EXEC SQL
          CONNECT :USERID
     END-EXEC.

* Allocate and initialize the BFILE locator:
     EXEC SQL ALLOCATE :BFILE1 END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
     EXEC SQL
          SELECT PHOTO INTO :BFILE1
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
     END-EXEC.

* Use the LOB DESCRIBE to see if lob is open:
     EXEC SQL
          LOB DESCRIBE :BFILE1 GET ISOPEN INTO :IS-OPEN
     END-EXEC.
     IF IS-OPEN = 1
*       Logic for an open BFILE goes here
        DISPLAY "BFILE is open."
     ELSE
*       Logic for a closed BFILE goes here
        DISPLAY "BFILE is closed."
     END-IF.

* And free the LOB locator:
 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
```

```
            EXEC SQL FREE :BFILE1 END-EXEC.
            EXEC SQL
                 COMMIT WORK RELEASE
            END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: See If the BFILE is Open with ISOPEN Using C++ (Pro*C/C++)

```
/* In Pro*C/C++, there is only one form of ISOPEN used to determine whether
   or not a BFILE is OPEN.  There is no FILE IS OPEN, only a simple ISOPEN.
   This is an attribute used in the DESCRIBE statement: */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void seeIfOpenBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int isOpen;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BFILE into the locator: */
  EXEC SQL SELECT Music INTO :Lob_loc FROM Multimedia_tab
           WHERE Clip_ID = 3;
  /* Determine if the BFILE is OPEN or not: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET ISOPEN into :isOpen;
  if (isOpen)
    printf("BFILE is open\n");
  else
    printf("BFILE is not open\n");
  /* Note that in this example, the BFILE is not open: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeIfOpenBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: See If the BFILE is Open with ISOPEN Using Visual Basic (OO4O)

```
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1:

Dim OraDyn as OraDynaset, OraMusic as OraBFile, amount_read%, chunksize%, chunk

chunksize = 32768
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music")

If OraMusic.IsOpen then
  'Processing given that the file is already open:
Else
   'Processing given that the file is not open, or return an error:
End If
```

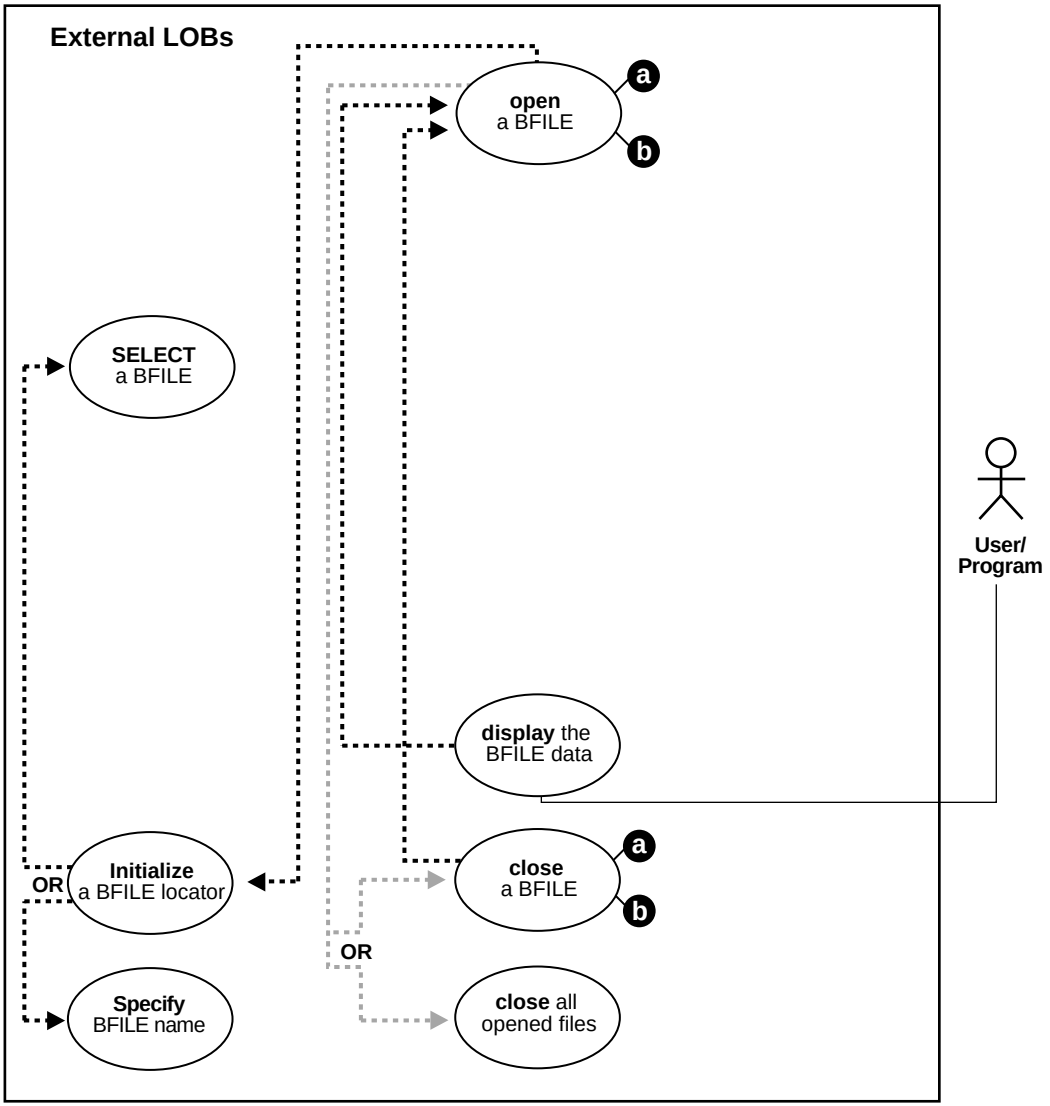
Example: See If the BFILE is Open with ISOPEN Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_48
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
       if (rset.next())
```

```
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        result = new Boolean(src_lob.isFileOpen());
        System.out.println(
           "result of fileIsOpen() before opening file : " + result.toString());

        src_lob.openFile();

        result = new Boolean(src_lob.isFileOpen());
        System.out.println(
           "result of fileIsOpen() after opening file : " + result.toString());

        // Close the BFILE,  statement and connection:
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
  }
}
```

Display the BFILE Data

Figure 5–18 Use Case Diagram: Display the BFILE data



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example opens and displays a BFILE is open that is associated with `Music`.

- "Example: Display the BFILE Data Using PL/SQL" on page 5-83
- "Example: Display the BFILE Data Using C (OCI)" on page 5-84
- "Example: Display the BFILE Data Using COBOL (Pro*COBOL)" on page 5-86
- "Example: Display the BFILE Data Using C++ (Pro*C/C++)" on page 5-88
- "Example: Display the BFILE Data Using Visual Basic (OO4O)" on page 5-90
- "Example: Display the BFILE Data Using Java (JDBC)" on page 5-90

Example: Display the BFILE Data Using PL/SQL

```
/* Note that the example procedure displayBFILE_proc is not part of the
DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE displayBFILE_proc IS
   Lob_loc  BFILE;
   Buffer   RAW(1024);
   Amount   BINARY_INTEGER := 1024;
   Position INTEGER        := 1;
BEGIN
   /* Select the LOB: */
   SELECT Music INTO Lob_loc
   FROM Multimedia_tab WHERE Clip_ID = 1;
   /* Opening the BFILE: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);
   LOOP
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Display the buffer contents: */
      DBMS_OUTPUT.PUT_LINE(utl_raw.cast_to_varchar2(Buffer));
      Position := Position + Amount;
   END LOOP;
   /* Closing the BFILE: */
   DBMS_LOB.CLOSE (Lob_loc);
   EXCEPTION
```

```
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

Example: Display the BFILE Data Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
#define MAXBUFLEN 32767

void BfileDisplay(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
   /* Assume all handles passed as input to this routine have been
      allocated and initialized.
   */
```

```
OCILobLocator *bfile_loc;
ub1 bufp[MAXBUFLEN];
ub4 buflen, amt, offset;
boolean done;
ub4 retval;

/* Allocate the locator descriptor */
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                          (ub4) OCI_DTYPE_FILE,
                          (size_t) 0, (dvoid **) 0)

 /* Select the bfile */
 selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
           "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3");

 ub1 bufp[MAXBUFLEN];
 ub4 buflen, amt, offset;
 boolean done;
 ub4 retval;

 checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                OCI_FILE_READONLY));
 /* This example will READ the entire contents of a BFILE piecewise into a
    buffer using a standard polling method, processing each buffer piece
    after every READ operation until the entire BFILE has been read. */
 /* Setting amt = 0 will read till the end of LOB*/
 amt = 0;
 buflen = sizeof(bufp);
 /* Process the data in pieces */
 offset = 1;
 memset(bufp, '\0', MAXBUFLEN);
 done = FALSE;
 while (!done)
 {
     retval = OCILobRead(svchp, errhp, bfile_loc,
                         &amt, offset, (dvoid *) bufp,
                         buflen, (dvoid *)0,
                         (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                         (ub2) 0, (ub1) SQLCS_IMPLICIT);
     switch (retval)
     {
         case 0:             /* Only one piece or last piece*/
            /* process the data in bufp. amt will give the amount of data
              just read in bufp. This is in bytes for BLOBs and in characters
                for fixed  width CLOBS and in bytes for variable width CLOBs*/
```

```
            done = TRUE;
            break;
          case -1:
            /*  report_error();         this function is not shown here */
            done = TRUE;
            break;
          case OCI_NEED_DATA:           /* There are 2 or more pieces */
            /* process the data in bufp. amt will give the amount of
               data just read in bufp. This is in bytes for BFILEs and i
               characters for fixed width CLOBS and in bytes for variable
               width CLOBs */
            break;
          default:
            (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
            done = TRUE;
            break;
        }  /* switch */
      } /* while */

      /* Closing the BFILE is mandatory if you have opened it */
      checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

      /* Free the locator descriptor */
      OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Example: Display the BFILE Data Using COBOL (Pro*COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  DISPLAY-BFILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  USERID         PIC X(9) VALUES "SAMP/SAMP".

     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
 01  DEST-BLOB      SQL-BLOB.
 01  SRC-BFILE      SQL-BFILE.
 01  BUFFER         PIC X(5) VARYING.
 01  OFFSET  PIC S9(9) COMP VALUE 1.
 01  AMT            PIC S9(9) COMP.
 01  ORASLNRD       PIC  9(4).
```

```
    EXEC SQL END DECLARE SECTION END-EXEC.

    01  D-AMTPIC 99,999,99.

    EXEC SQL VAR BUFFER IS LONG RAW (100) END-EXEC.

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

 PROCEDURE DIVISION.
 DISPLAY-BFILE-DATA.

* Connect to ORACLE
    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator
    EXEC SQL ALLOCATE :SRC-BFILE END-EXEC.

* Select the BFILE
    EXEC SQL SELECT PHOTO INTO :SRC-BFILE
       FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
    END-EXEC.

* Open the BFILE
    EXEC SQL LOB OPEN :SRC-BFILE READ ONLY END-EXEC.

* Set the amount = 0 will initiate the polling method
    MOVE 0 TO AMT;
    EXEC SQL
          LOB READ :AMT FROM :SRC-BFILE INTO :BUFFER
    END-EXEC.

*    DISPLAY "BFILE DATA".
*    MOVE AMT TO D-AMT.
*    DISPLAY "First READ (", D-AMT, "): " BUFFER.

* Do READ-LOOP until the whole BFILE is read.
    EXEC SQL WHENEVER NOT FOUND GO TO END-LOOP END-EXEC.

 READ-LOOP.
    EXEC SQL
```

```
            LOB READ :AMT FROM :SRC-BFILE INTO :BUFFER
        END-EXEC.

*       MOVE AMT TO D-AMT.
*       DISPLAY "Next READ (", D-AMT, "): " BUFFER.

        GO TO READ-LOOP.

 END-LOOP.
        EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.

* Close the LOB
        EXEC SQL LOB CLOSE :SRC-BFILE END-EXEC.

* And free the LOB locator
        EXEC SQL FREE :SRC-BFILE END-EXEC.
        EXEC SQL ROLLBACK RELEASE END-EXEC.
        STOP RUN.

 SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Display the BFILE Data Using C++ (Pro*C/C++)

```
/* This example will READ the entire contents of a BFILE piecewise into a
   buffer using a streaming mechanism via standard polling, displaying each
   buffer piece after every READ operation until the entire BFILE has been
   read: */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
```

```
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 1024

void displayBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int Amount;
  struct {
    short Length;
    char Data[BufferLength];
  } Buffer;
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer is VARRAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BFILE: */
  EXEC SQL SELECT Music INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 3;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Setting Amount = 0 will initiate the polling method: */
  Amount = 0;
  /* Set the maximum size of the Buffer: */
  Buffer.Length = BufferLength;
  EXEC SQL WHENEVER NOT FOUND DO break;
  while (TRUE)
    {
      /* Read a piece of the BFILE into the Buffer: */
      EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
      printf("Display %d bytes\n", Buffer.Length);
    }
  printf("Display %d bytes\n", Amount);
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
```

```
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  displayBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Display the BFILE Data Using Visual Basic (OO4O)

```
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1:
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk
As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

OraMusic.offset = 1
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents

'Open the Bfile for reading:
OraMusic.Open
amount_read = OraMusic.Read(chunk, chunksize)

While OraMusic.Status = ORALOB_NEED_DATA
    amount_read = OraMusic.Read(chunk, chunksize)
Wend

OraMusic.Close
```

Example: Display the BFILE Data Using Java (JDBC)

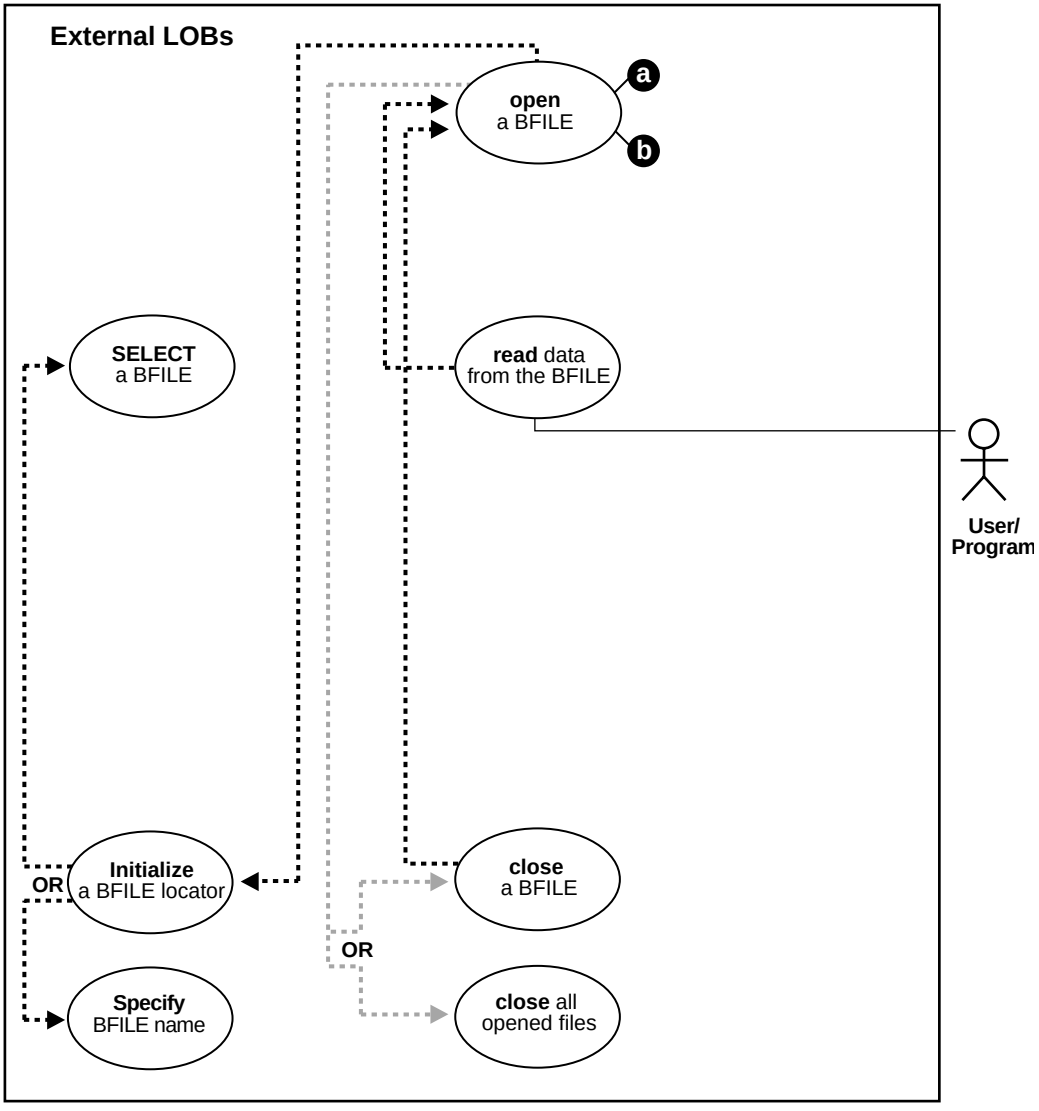
```
// Java IO classes
import java.io.InputStream;
import java.io.OutputStream;
```

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_53
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;
       InputStream in = null;
       byte buf[] = new byte[1000];
       int length = 0;
       boolean alreadyDisplayed = false;

       rset = stmt.executeQuery (
          "SELECT music FROM multimedia_tab WHERE clip_id = 2");

       if (rset.next())
       {
```

```
            src_lob = ((OracleResultSet)rset).getBFILE (1);
          }

          // Open the BFILE:
          src_lob.openFile();

          // Get a handle to stream the data from the BFILE:
          in = src_lob.getBinaryStream();

          // This loop fills the buf iteratively, retrieving data
          // from the InputStream:
          while ((in != null) && ((length = in.read(buf)) != -1))
          {
             // the data has already been read into buf

            // We will only display the first CHUNK in this example:
            if (! alreadyDisplayed)
            {
               System.out.println("Bytes read in: " + Integer.toString(length));
               System.out.println(new String(buf));
               alreadyDisplayed = true;
            }
          }

          // Close the stream, BFILE, statement and connection:
          in.close();
          src_lob.closeFile();
          stmt.close();
          conn.commit();
          conn.close();
        }
        catch (SQLException e)
        {
           e.printStackTrace();
        }
      }
    }
```

Read the Data from a BFILE

Figure 5–19 Use Case Diagram: Read the data from a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

When reading the LOB value, it is not an error to try to read beyond the end of the LOB. This means that you can always specify an input amount of 4 gigabytes regardless of the starting offset and the amount of data in the LOB. You do not need to incur a round-trip to the server to call OCILobGetLength() to find out the length of the LOB value in order to determine the amount to read.

For example, assume that the length of a LOB is 5,000 bytes and you want to read the entire LOB value starting at offset 1,000. Also assume that you do not know the current length of the LOB value. Here is the OCI read call, excluding the initialization of all parameters:

```
#define MAX_LOB_SIZE 4294967295
ub4  amount =  MAX_LOB_SIZE;
ub4  offset = 1000;
OCILobRead(svchp, errhp, locp, &amount, offset, bufp, bufl, 0, 0, 0, 0)
```

Note: The most efficient way to read large amounts of LOB data is to use OCILobRead() with the streaming mechanism enabled via polling or a callback.

The following example considers reading a photograph into PHOTO from a BFILE 'PHOTO_DIR'.

- "Example: Read the Data from a BFILE Using PL/SQL (DBMS_LOB Package)" on page 5-95
- "Example: Read the Data from a BFILE Using C (OCI)" on page 5-95
- "Example: Read the Data from a BFILE Using COBOL (Pro*COBOL)" on page 5-97
- "Example: Read the Data from a BFILE Using C++ (Pro*C/C++)" on page 5-98
- "Example: Read the Data from a BFILE Using Visual Basic (OO4O)" on page 5-99

- "Example: Read the Data from a BFILE Using Java (JDBC)" on page 5-100

Example: Read the Data from a BFILE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure readBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE readBFILE_proc IS
   Lob_loc       BFILE := BFILENAME('PHOTO_DIR', 'Jefferson_photo');
   Amount        INTEGER := 32767;
   Position      INTEGER := 1;
   Buffer        RAW(32767);
BEGIN
   /* Select the LOB: */
   SELECT Photo INTO Lob_loc FROM Multimedia_tab
      WHERE Clip_ID = 3;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Read data: */
   DBMS_LOB.READ(Lob_loc, Amount, Position, Buffer);
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Read the Data from a BFILE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
```

```
                                        OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                        (ub4) OCI_DEFAULT));
}
#define MAXBUFLEN 32767
void BfileRead(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     ub1 bufp[MAXBUFLEN];
     ub4 buflen, amt, offset;
     boolean done;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                               (ub4) OCI_DTYPE_FILE,
                               (size_t) 0, (dvoid **) 0)

     /* Select the bfile */
     selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
               "SELECT Photo FROM Multimedia_tab WHERE Clip_ID=3");


     ub1 bufp[MAXBUFLEN];
     ub4 buflen, amt, offset;
     boolean done;

     checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                    OCI_FILE_READONLY));
     amt = MAXBUFLEN;
     buflen = sizeof(bufp);
     /* Process the data in pieces */
     offset = 1;
     memset(bufp, '\0', MAXBUFLEN);
```

```
    done = FALSE;
    checkerr(errhp, OCILobRead(svchp, errhp, bfile_loc, &amt, offset,
                               (dvoid *) bufp, buflen, (dvoid *)0,
                               (sb4 (*)(dvoid *, dvoid *, ub4, ub1)) 0,
                               (ub2) 0, (ub1) SQLCS_IMPLICIT));

    /* Closing the BFILE is mandatory if you have opened it */
    checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Example: Read the Data from a BFILE Using COBOL (Pro*COBOL)

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. READ-BFILE.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.

 01  BFILE1         SQL-BFILE.
 01  BUFFER2        PIC X(5) VARYING.
 01  AMT            PIC S9(9) COMP.
 01  OFFSET         PIC S9(9) COMP VALUE 1.

     EXEC SQL INCLUDE SQLCA END-EXEC.

     EXEC SQL VAR BUFFER2 IS LONG RAW(5) END-EXEC.

 PROCEDURE DIVISION.
 READ-BFILE.

* Allocate and initialize the CLOB locator
     EXEC SQL ALLOCATE :BFILE1 END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.

     EXEC SQL
          SELECT MUSIC INTO :BFILE1
          FROM MULTIMEDIA_TAB M WHERE M.CLIP_ID = 3
     END-EXEC.
* Open the BFILE
     EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.
```

```
      * Initiate polling read
           MOVE 0 TO AMT.

           EXEC SQL LOB READ :AMT FROM :BFILE1
                 INTO :BUFFER2 END-EXEC.
      *
      *     Display the data here.
      *

      * Close and free the locator
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

       END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
```

Example: Read the Data from a BFILE Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 4096

void readBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  /* Amount and BufferLength are equal so only one READ is necessary: */
  int Amount = BufferLength;
  char Buffer[BufferLength];
  /* Datatype Equivalencing is Mandatory for this Datatype: */
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
```

```
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Photo INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 3;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  EXEC SQL WHENEVER NOT FOUND CONTINUE;
  /* Read data: */
  EXEC SQL LOB READ :Amount FROM :Lob_loc INTO :Buffer;
  printf("Read %d bytes\n", Amount);
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  readBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Read the Data from a BFILE Using Visual Basic (OO4O)

```
'Example: Read the Data from a BFILE Using Visual Basic (OO4O)
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1:
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk
As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

OraMusic.offset = 1
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents
```

```
'Open the Bfile for reading:
OraMusic.Open
amount_read = OraMusic.Read(chunk, chunksize)

While OraMusic.Status = ORALOB_NEED_DATA
    amount_read = OraMusic.Read(chunk, chunksize)
Wend

OraMusic.Close
```

Example: Read the Data from a BFILE Using Java (JDBC)

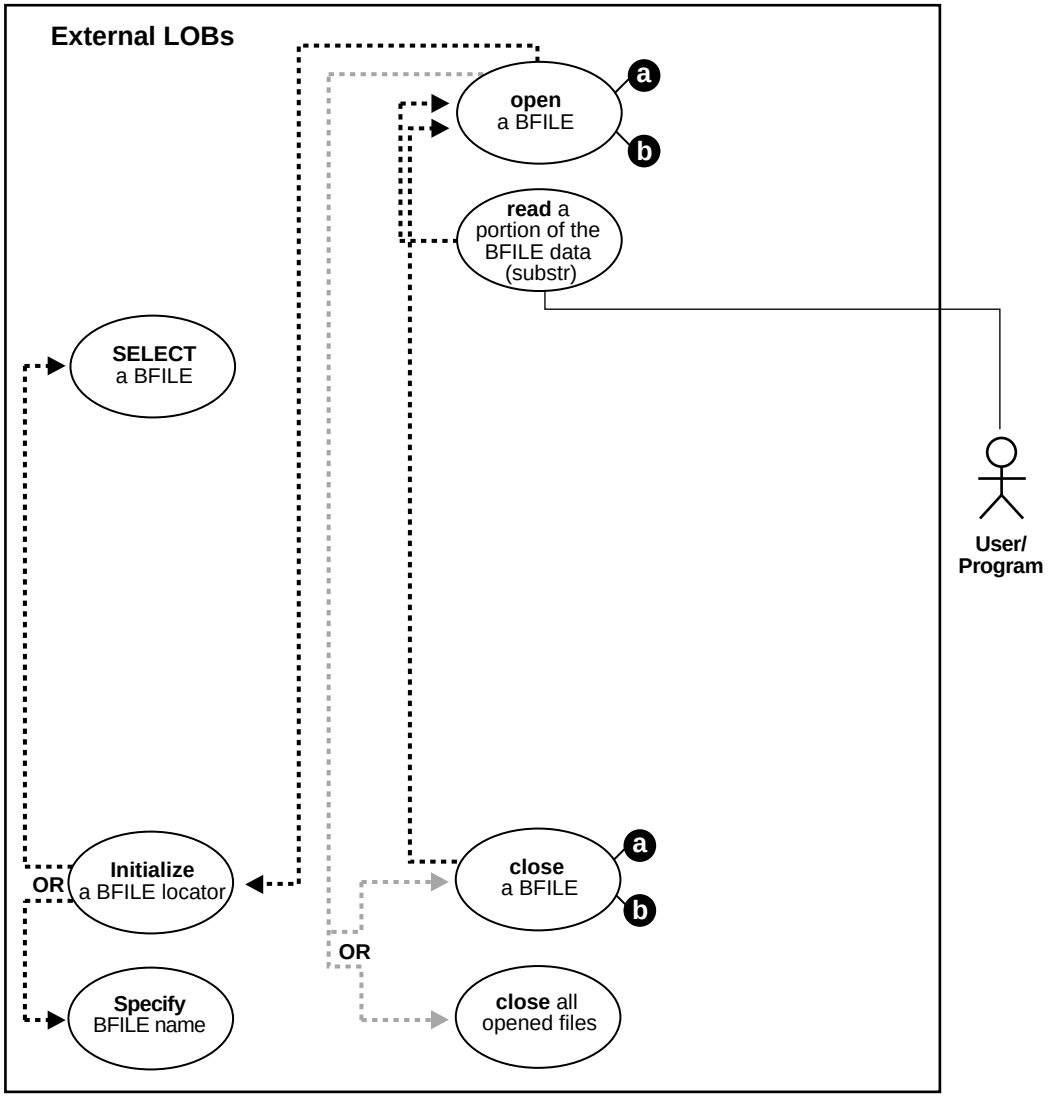
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_53
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement
```

```
Statement stmt = conn.createStatement ();

try
{
   BFILE src_lob = null;
   ResultSet rset = null;
   Boolean result = null;
   InputStream in = null;
   byte buf[] = new byte[1000];
   int length = 0;
   boolean alreadyDisplayed = false;
   rset = stmt.executeQuery (
      "SELECT music FROM multimedia_tab WHERE clip_id = 2");
   if (rset.next())
   {
      src_lob = ((OracleResultSet)rset).getBFILE (1);
   }

   // Open the BFILE:
   src_lob.openFile();

   // Get a handle to stream the data from the BFILE:
   in = src_lob.getBinaryStream();

   // This loop fills the buf iteratively, retrieving data
   // from the InputStream:
   while ((in != null) && ((length = in.read(buf)) != -1))
   {
      // the data has already been read into buf

      // We will only display the first CHUNK in this example:
      if (! alreadyDisplayed)
      {
         System.out.println("Bytes read in: " + Integer.toString(length));
         System.out.println(new String(buf));
         alreadyDisplayed = true;
      }
   }

   // Close the stream, BFILE, statement and connection:
   in.close();
   src_lob.closeFile();
   stmt.close();
   conn.commit();
   conn.close();
```

```
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
      }
    }
```

Read a Portion of the BFILE Data (substr)

Figure 5–20 Use Case Diagram: Read a portion of the BFILE data (substr)



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The following example considers reading an audio recording into `RECORDING` from a `BFILE 'AUDIO_DIR'`.

- "Example: Read a Portion of the BFILE Data (substr) Using PL/SQL (DBMS_LOB Package)" on page 5-104
- "Example: Read a Portion of the BFILE Data (substr) Using COBOL (Pro*COBOL)" on page 5-105
- "Example: Read a Portion of the BFILE Data (substr) Using C++ (Pro*C/C++)" on page 5-106
- "Example: Read a Portion of the BFILE Data (substr) Using Visual Basic (OO4O)" on page 5-107
- "Example: Read a Portion of the BFILE Data (substr) Using Java (JDBC)" on page 5-107

Example: Read a Portion of the BFILE Data (substr) Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure substringBFILE_proc is not part of the
   DBMS_LOB package:  */
CREATE OR REPLACE PROCEDURE substringBFILE_proc IS
   Lob_loc         BFILE;
   Position        INTEGER := 1;
   Buffer          RAW(32767);
BEGIN
   /* Select the LOB: */
   SELECT Mtab.Voiced_ref.Recording INTO Lob_loc FROM Multimedia_tab Mtab
      WHERE Mtab.Clip_ID = 3;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   Buffer := DBMS_LOB.SUBSTR(Lob_loc, 255, Position);
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Read a Portion of the BFILE Data (substr) Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  BFILE1         SQL-BFILE.
01  BUFFER2        PIC X(32767) VARYING.
01  AMT            PIC S9(9) COMP.
01  POS            PIC S9(9) COMP VALUE 1024.
01  OFFSET         PIC S9(9) COMP VALUE 1.

    EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

PROCEDURE DIVISION.
BFILE-SUBSTR.

* Allocate and initialize the CLOB locator:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.

    EXEC SQL
         SELECT MTAB.VOICED_REF.RECORDING INTO :BFILE1
         FROM MULTIMEDIA_TAB MTAB WHERE MTAB.CLIP_ID = 3
    END-EXEC.

* Open the BFILE for READ ONLY:
    EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* Execute PL/SQL to use its SUBSTR functionality:
    MOVE 32767 TO AMT.
    EXEC SQL EXECUTE
      BEGIN
        :BUFFER2 := DBMS_LOB.SUBSTR(:BFILE1,:AMT,:POS);
      END;
    END-EXEC.

* Close and free the locators:
    EXEC SQL LOB CLOSE :BFILE1 END-EXEC.
```

```
END-OF-BFILE.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXC SQL FREE :BFILE1 END-EXEC.
```

Example: Read a Portion of the BFILE Data (substr) Using C++ (Pro*C/C++)

```
/* Pro*C/C++ lacks an equivalent embedded SQL form for the DBMS_LOB.SUBSTR()
   function.  However, Pro*C/C++ can interoperate with PL/SQL using anonymous
   PL/SQL blocks embedded in a Pro*C/C++ program as this example shows: */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define BufferLength 256

void substringBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  int Position = 1;
  char Buffer[BufferLength];
  EXEC SQL VAR Buffer IS RAW(BufferLength);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Invoke SUBSTR() from within an anonymous PL/SQL block:  */
  EXEC SQL EXECUTE
    BEGIN
      :Buffer := DBMS_LOB.SUBSTR(:Lob_loc, 256, :Position);
    END;
  END-EXEC;
  /* Close the BFILE:  */
  EXEC SQL LOB CLOSE :Lob_loc;
```

```
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  substringBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Read a Portion of the BFILE Data (substr) Using Visual Basic (OO4O)

```
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1:
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunk_size = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value
OraMusic.PollingAmount = OraMusic.Size 'Read entire BFILE contents
OraMusic.offset = 255 'Read from the 255th position
'Open the Bfile for reading:
OraMusic.Open
amount_read = OraMusic.Read(chunk, chunk_size) 'chunk returned is a variant of
type byte array
 If amount_read <> chunk_size Then
    'Do error processing
 Else
     'Process the data
 End If
```

Example: Read a Portion of the BFILE Data (substr) Using Java (JDBC)

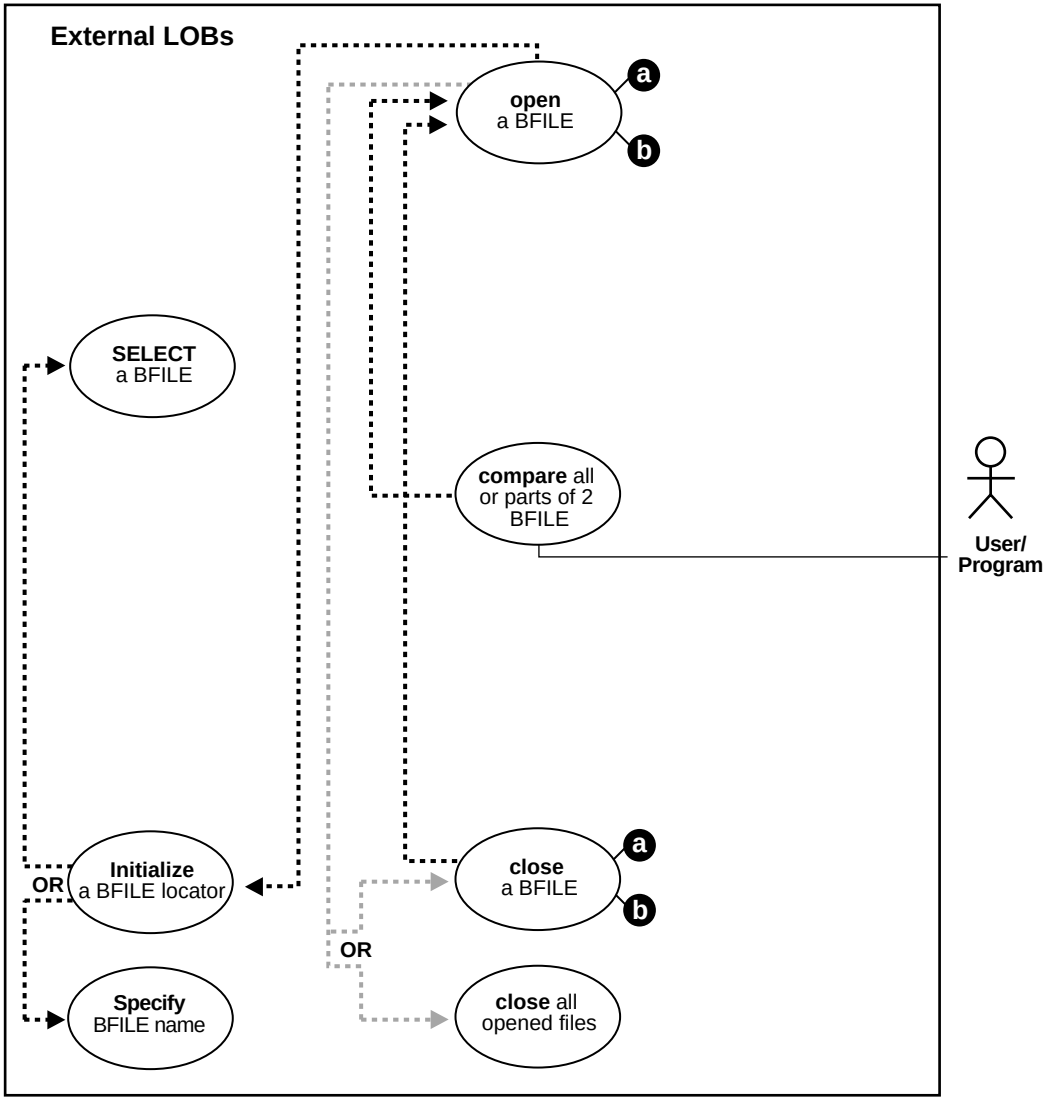
```
// Java IO classes:
import java.io.InputStream;
```

```
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_62
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
   BFILE src_lob = null;
       ResultSet rset = null;
   InputStream in = null;
   byte buf[] = new byte[1000];
   int length = 0;

   rset = stmt.executeQuery (
      "SELECT music FROM multimedia_tab WHERE clip_id = 2");
   if (rset.next())
   {
     src_lob = ((OracleResultSet)rset).getBFILE (1);
```

```
    }

    // Open the BFILE:
    src_lob.openFile();

    // Get a handle to stream the data from the BFILE
    in = src_lob.getBinaryStream();

    if (in != null)
    {
       // request 255 bytes into buf, starting from offset 1.
       // length = # bytes actually returned from stream:
       length = in.read(buf, 1, 255);

       System.out.println("Bytes read in: " + Integer.toString(length));

       // Process the buf:
       System.out.println(new String(buf));
    }

    // Close the stream, BFILE, statement and connection:
    in.close();
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
     }
     catch (SQLException e)
     {
        e.printStackTrace();
     }
   }
}
```

Compare All or Parts of Two BFILES

Figure 5–21 Use Case Diagram: Compare all or parts of 2 BFILES



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The following example deals with the problem of determining whether a photograph in the file 'PHOTO_DIR' has already been used as a specific PHOTO by comparing each data entity bit by bit. Note that LOBMAXSIZE is set at 4 gigabytes so that you do not have to find out the length of each BFILE before beginning the comparison.

Example: Compare All or Parts of Two BFILES Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure compareBFILEs_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE compareBFILEs_proc IS
   /* Initialize the BFILE locator: */
   Lob_loc1       BFILE := BFILENAME('PHOTO_DIR', 'RooseveltFDR_photo');
   Lob_loc2       BFILE;
   Retval         INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT Photo INTO Lob_loc2 FROM Multimedia_tab
      WHERE Clip_ID = 3;
   /* Open the BFILEs: */
```

```
   DBMS_LOB.OPEN(Lob_loc1, DBMS_LOB.LOB_READONLY);
   DBMS_LOB.OPEN(Lob_loc2, DBMS_LOB.LOB_READONLY);
   Retval := DBMS_LOB.COMPARE(Lob_loc2, Lob_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
   /* Close the BFILEs: */
   DBMS_LOB.CLOSE(Lob_loc1);
   DBMS_LOB.CLOSE(Lob_loc2);
END;
```

Example: Compare All or Parts of Two BFILES Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "USER1/USER1".
01  BFILE1          SQL-BFILE.
01  BFILE2          SQL-BFILE.
01  RET             PIC S9(9) COMP.
01  AMT             PIC S9(9) COMP.
01  DIR-ALIAS       PIC X(30) VARYING.
01  FNAME           PIC X(20) VARYING.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-COMPARE.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BLOB locators:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.
    EXEC SQL ALLOCATE :BFILE2 END-EXEC.

* Set up the directory and file information:
    MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
```

```
     MOVE "fdroosevelt_photo" TO FNAME-ARR.
     MOVE 17 TO FNAME-LEN.

     EXEC SQL
        LOB FILE SET :BFILE1 DIRECTORY = :DIR-ALIAS,
        FILENAME = :FNAME
     END-EXEC.

     EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
     EXEC SQL
          SELECT PHOTO INTO :BFILE2
          FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
     END-EXEC.

* Open the BLOBs for READ ONLY:
     EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.
     EXEC SQL LOB OPEN :BFILE2 READ ONLY END-EXEC.

* Execute PL/SQL to get COMPARE functionality:
     MOVE 5 TO AMT.
     EXEC SQL EXECUTE
       BEGIN
         :RET := DBMS_LOB.COMPARE(:BFILE1,:BFILE2,
                                  :AMT,1,1);
       END;
     END-EXEC.

     IF RET = 0
*         Logic for equal BFILEs goes here
          DISPLAY "BFILES are equal"
     ELSE
*         Logic for unequal BFILEs goes here
          DISPLAY "BFILEs are not equal"
     END-IF.

     EXEC SQL LOB CLOSE :BFILE1 END-EXEC.
     EXEC SQL LOB CLOSE :BFILE2 END-EXEC.

 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL FREE :BFILE2 END-EXEC.
     STOP RUN.

 SQL-ERROR.
```

```
EXEC SQL
    WHENEVER SQLERROR CONTINUE
END-EXEC.
MOVE ORASLNR TO ORASLNRD.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.
```

Example: Compare All or Parts of Two BFILES Using C++ (Pro*C/C++)

```
/* Pro*C/C++ lacks an equivalent embedded SQL form for the
   DBMS_LOB.COMPARE() function.  Like the DBMS_LOB.SUBSTR() function,
   however, Pro*C/C++ can invoke DBMS_LOB.COMPARE() in an anonymous PL/SQL
   block as is shown here:  */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void compareBFILEs_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;
  int Retval = 1;
  char *Dir1 = "PHOTO_DIR", *Name1 = "RooseveltFDR_photo";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL LOB FILE SET :Lob_loc1 DIRECTORY = :Dir1, FILENAME = :Name1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Photo INTO :Lob_loc2 FROM Multimedia_tab
           WHERE Clip_ID = 3;
```

```
  /* Open the BFILEs: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Compare the BFILEs in PL/SQL using DBMS_LOB.COMPARE() */
  EXEC SQL EXECUTE
    BEGIN
      :Retval := DBMS_LOB.COMPARE(
                  :Lob_loc2, :Lob_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
    END;
  END-EXEC;
  /* Close the BFILEs:  */
  EXEC SQL LOB CLOSE :Lob_loc1;
  EXEC SQL LOB CLOSE :Lob_loc2;
  if (0 == Retval)
    printf("BFILEs are the same\n");
  else
    printf("BFILEs are not the same\n");
  /* Release resources used by the locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  compareBFILEs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Compare All or Parts of Two BFILES Using Visual Basic (OO4O)

```
'Note that the PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraMusic As OraBfile, OraMyMusic As OraBfile, OraSql
As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans
```

```
Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 1001, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyMusic", Empty, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
         END;", ORASQL_FAILEXEC)

Set OraMyMusic = OraParameters("MyMusic").Value

'Create dynaset:
Set OraDyn =
   OraDb.CreateDynaset(
      "SELECT * FROM Multimedia_tab WHERE Clip_Id = 1001", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

'Open the Bfile for reading:
OraMusic.Open
OraMyMusic.Open

If OraMusic.Compare(OraMyMusic) Then
    'Process the data
Else
   'Do error processing
End If
OraDb.Connection.CommitTrans
```

Example: Compare All or Parts of Two BFILES Using Java (JDBC)

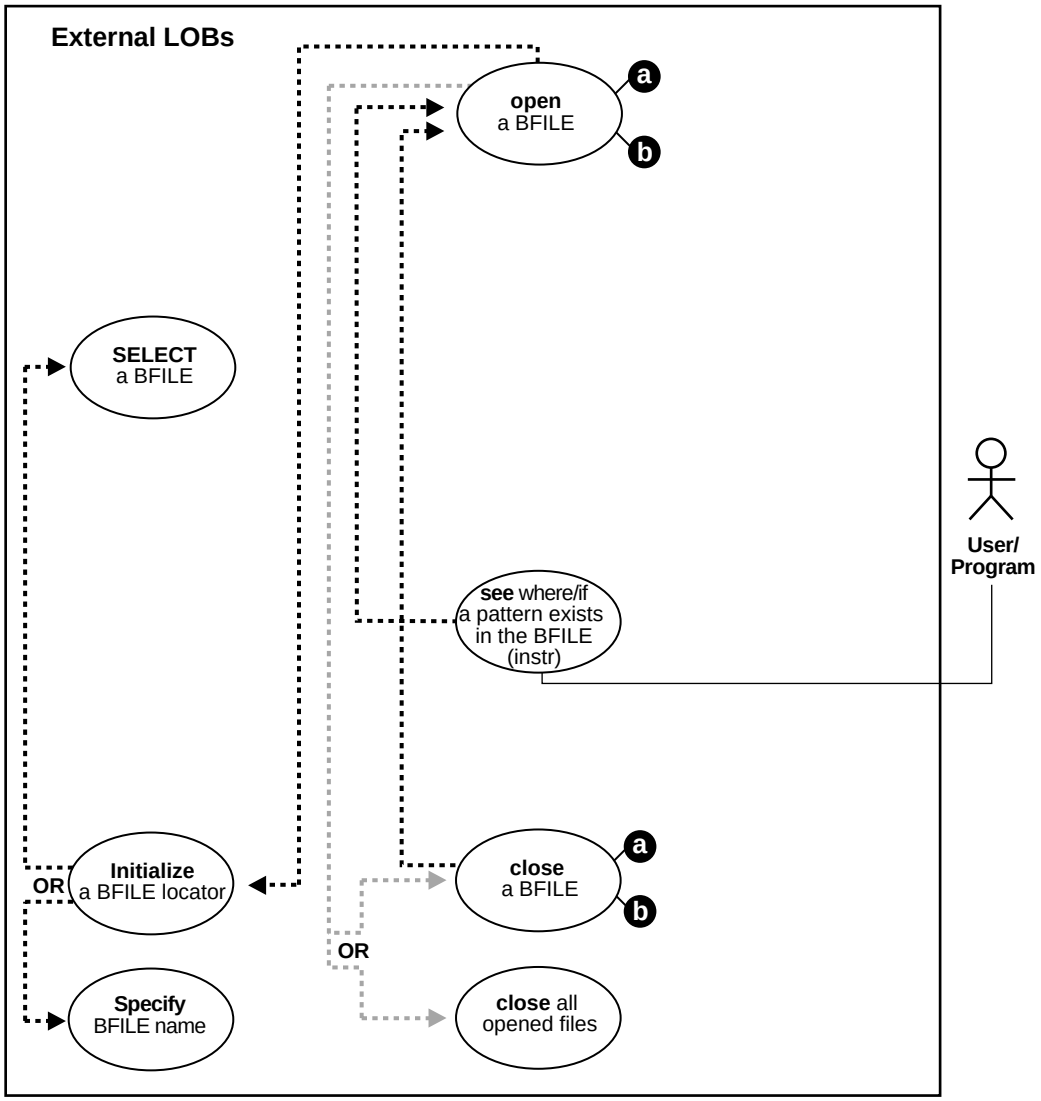
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_66
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;
       ResultSet rset = null;

       rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 2");
       if (rset.next())
       {
          lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
       }

       rset = stmt.executeQuery (
          "SELECT BFILENAME('PHOTO_DIR', 'music') FROM DUAL");
       if (rset.next())
       {
```

```
            lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
         }

         if (lob_loc1.length() > lob_loc2.length())
         System.out.println("Looking for LOB2 inside LOB1.  result = " +
            Long.toString(lob_loc1.position(lob_loc2, 0)));
    else
      System.out.println("Looking for LOB1 inside LOB2.  result = " +
         Long.toString(lob_loc2.position(lob_loc1, 0)));

    stmt.close();
    conn.commit();
    conn.close();

     }
     catch (SQLException e)
     {
         e.printStackTrace();
     }
   }
}
```

See If a Pattern Exists (instr) in the BFILE

Figure 5–22 Use Case Diagram: See If a Pattern Exists in the BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

The following example searches for the occurrence of a pattern of audio data within an interview `Recording`. This assumes that an audio signature is represented by an identifiable bit pattern.

- "Example: See If a Pattern Exists (instr) in the BFILE Using PL/SQL (DBMS_LOB Package)" on page 5-120
- "Example: See If a Pattern Exists (instr) in the BFILE Using COBOL (Pro*COBOL)" on page 5-121
- "Example: See If a Pattern Exists (instr) in the BFILE Using C++ (Pro*C/C++)" on page 5-123
- "Example: See If a Pattern Exists (instr) in the BFILE Using Visual Basic (OO4O)" on page 5-124
- "Example: See If a Pattern Exists (instr) in the BFILE Using Java (JDBC)" on page 5-124

Example: See If a Pattern Exists (instr) in the BFILE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure instringBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instringBFILE_proc IS
   Lob_loc        BFILE;
   Pattern        RAW(32767);
   Position       INTEGER;
BEGIN
   /* Select the LOB: */
   SELECT Intab.Recording INTO Lob_loc
      FROM THE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
         WHERE Clip_ID = 3) Intab
            WHERE Segment = 1;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /*  Initialize the pattern for which to search, find the 2nd occurrence of
```

```
        the pattern starting from the beginning of the BFILE: */
    Position := DBMS_LOB.INSTR(Lob_loc, Pattern, 1, 2);
    /* Close the BFILE: */
    DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: See If a Pattern Exists (instr) in the BFILE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-INSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.

* The length of pattern was chosen arbitrarily:
01  PATTERN        PIC X(4) VALUE "2424".
    EXEC SQL VAR PATTERN IS RAW(4) END-EXEC.
01  POS            PIC S9(9) COMP.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-INSTR.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
    EXEC SQL
         SELECT PHOTO INTO :BFILE1
         FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
    END-EXEC.
```

```
* Open the CLOB for READ ONLY:
     EXEC SQL LOB OPEN :BFILE1 READ ONLY END-EXEC.

* Execute PL/SQL to get INSTR functionality:
     EXEC SQL EXECUTE
       BEGIN
         :POS := DBMS_LOB.INSTR(:BFILE1,:PATTERN, 1, 2);
       END;
     END-EXEC.

     IF POS = 0
*        Logic for pattern not found here
         DISPLAY "Pattern is not found."
     ELSE
*        Pos contains position where pattern is found
         DISPLAY "Pattern is found."
     END-IF.

* Close and free the LOB:
     EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

 END-OF-BFILE.
     EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
     EXEC SQL FREE :BFILE1 END-EXEC.
     EXEC SQL
          COMMIT WORK RELEASE
     END-EXEC.
     STOP RUN.

 SQL-ERROR.
     EXEC SQL
         WHENEVER SQLERROR CONTINUE
     END-EXEC.
     MOVE ORASLNR TO ORASLNRD.
     DISPLAY " ".
     DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
     DISPLAY " ".
     DISPLAY SQLERRMC.
     EXEC SQL
         ROLLBACK WORK RELEASE
     END-EXEC.
     STOP RUN.
```

Example: See If a Pattern Exists (instr) in the BFILE Using C++ (Pro*C/C++)

```
/* Pro*C lacks an equivalent embedded SQL form of the DBMS_LOB.INSTR()
   function.  However, like SUBSTR() and COMPARE(), Pro*C/C++ can call
   DBMS_LOB.INSTR() from within an anonymous PL/SQL block as shown here: */
#include <sql2oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

#define PatternSize 5

void instringBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Position = 0;
  int Clip_ID = 3, Segment = 1;
  char Pattern[PatternSize];
  /* Datatype Equivalencing is Mandatory for this Datatype:  */
  EXEC SQL VAR Pattern IS RAW(PatternSize);

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Use Dynamic SQL to retrieve the BFILE Locator:  */
  EXEC SQL PREPARE S FROM
    'SELECT Intab.Recording \
       FROM TABLE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab \
          WHERE Clip_ID = :cid) Intab \
             WHERE Intab.Segment = :seg';
  EXEC SQL DECLARE C CURSOR FOR S;
  EXEC SQL OPEN C USING :Clip_ID, :Segment;
  EXEC SQL FETCH C INTO :Lob_loc;
  EXEC SQL CLOSE C;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  memset((void *)Pattern, 0, PatternSize);
  /* Find the first occurrance of the pattern starting from the
```

```
    beginning of the BFILE using PL/SQL: */
  EXEC SQL EXECUTE
    BEGIN
      :Position := DBMS_LOB.INSTR(:Lob_loc, :Pattern, 1, 1);
    END;
  END-EXEC;
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  if (0 == Position)
    printf("Pattern not found\n");
  else
    printf("The pattern occurs at %d\n", Position);
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  instringBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: See If a Pattern Exists (instr) in the BFILE Using Visual Basic (OO4O)

Note: A Visual Basic (OO4O) example will made available in a subsequent release.

Example: See If a Pattern Exists (instr) in the BFILE Using Java (JDBC)

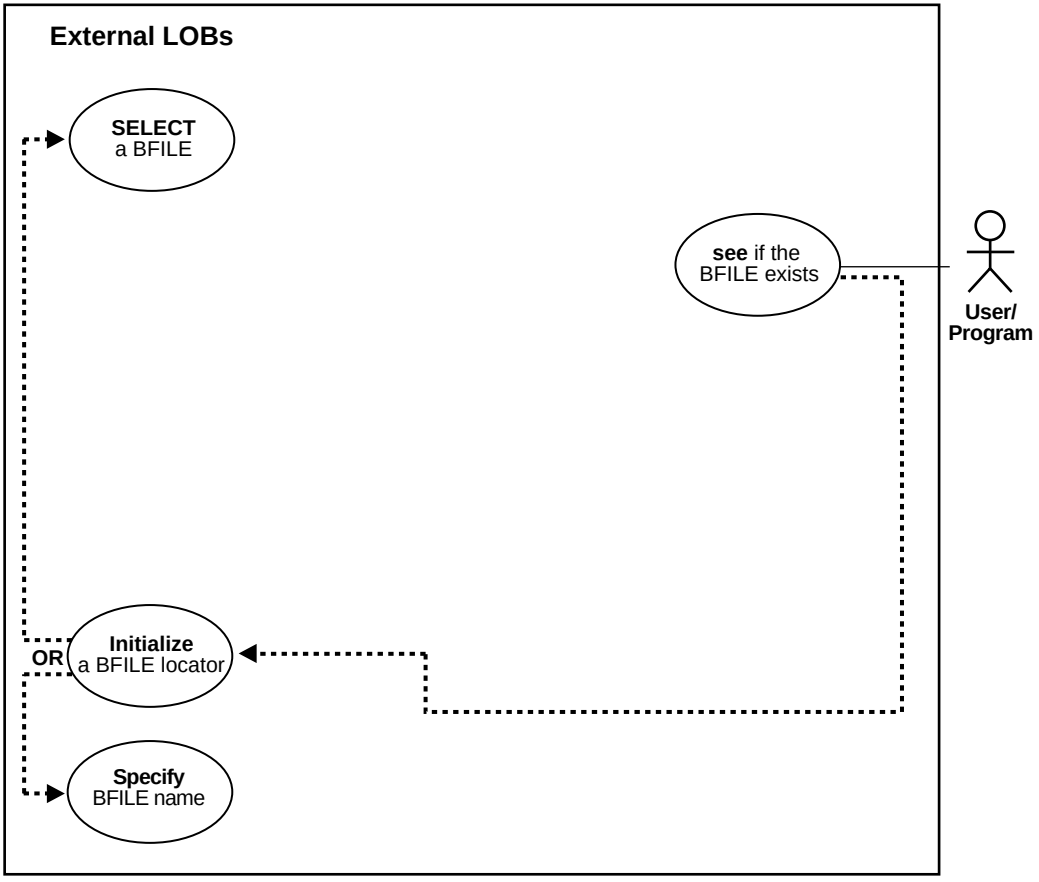
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_70
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE lob_loc = null;
       // Pattern to look for within the BFILE:
       String pattern = new String("children");

       ResultSet rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }

       // Open the LOB:
       lob_loc.openFile();

       // Search for the location of pattern string in the BFILE,
       // starting at offset 1:
      long result = lob_loc.position(pattern.getBytes(), 1);
      System.out.println(
```

```
            "Results of Pattern Comparison : " + Long.toString(result));

          // Close the LOB:
          lob_loc.closeFile();

          stmt.close();
          conn.commit();
          conn.close();

        }
        catch (SQLException e)
        {
           e.printStackTrace();
        }
      }
    }
```

See If the BFILE Exists

Figure 5–23 Use Case Diagram: See If the BFILE exists



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example queries whether a BFILE that is associated with `Recording`.

Example: See If the BFILE Exists Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure seeIfExistsBFILE_proc is not part of the
   DBMS_LOB package:  */
CREATE OR REPLACE PROCEDURE seeIfExistsBFILE_proc IS
   Lob_loc      BFILE;
BEGIN
   /* Select the LOB: */
   SELECT Intab.Recording INTO Lob_loc
      FROM THE(SELECT Mtab.InSeg_ntab FROM Multimedia_tab Mtab
         WHERE Mtab.Clip_ID = 3) Intab
            WHERE Intab.Segment = 1;
   /* See If the BFILE exists: */
   IF (DBMS_LOB.FILEEXISTS(Lob_loc) != 0)
   THEN
      DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE exists');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Processing given that the BFILE does not exist');
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Example: See If the BFILE Exists Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
```

```
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
boolean BfileExists(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     boolean is_exists;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Select the bfile */
     selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
               "SELECT Intab.Recording FROM THE(
                SELECT Mtab.InSeg_ntab FROM
```

```
                    Multimedia_tab Mtab WHERE Mtab.Clip_ID=3) Intab
                    WHERE Intab.Segment = 1");


     boolean is_exists;
     checkerr(errhp, OCILobFileExists(svchp, errhp, bfile_loc,
                                      &is_exists));
     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
     return(is_exists);
}
```

Example: See If the BFILE Exists Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BFILE-EXISTS.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID         PIC X(11) VALUES "USER1/USER1".
       01  BFILE1         SQL-BFILE.
       01  FEXISTS        PIC S9(9) COMP.
       01  ORASLNRD       PIC 9(4).

           EXEC SQL INCLUDE SQLCA END-EXEC.
           EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
           EXEC SQL INCLUDE ORACA END-EXEC.

       PROCEDURE DIVISION.
       BFILE-EXISTS.

           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
                CONNECT :USERID
           END-EXEC.

      * Allocate and initialize the BFILE locator:
           EXEC SQL ALLOCATE :BFILE1 END-EXEC.

           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
           EXEC SQL
                SELECT PHOTO INTO :BFILE1
                FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
```

```
           END-EXEC.

           EXEC SQL
                LOB DESCRIBE :BFILE1 GET FILEEXISTS INTO :FEXISTS
           END-EXEC.

           IF FEXISTS = 1
       *       Logic for file exists here
               DISPLAY "File exists"
           ELSE
       *       Logic for file does not exist here
               DISPLAY "File does not exist"
           END-IF.

       END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
           EXEC SQL
                COMMIT WORK RELEASE
           END-EXEC.
           STOP RUN.

       SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

Example: See If the BFILE Exists Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
```

```
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void seeIfBFILEExists_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Exists = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* See if the BFILE Exists:  */
  EXEC SQL LOB DESCRIBE :Lob_loc GET FILEEXISTS INTO :Exists;
  printf("BFILE %s exist\n", Exists ? "does" : "does not");
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  seeIfBFILEExists_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: See If the BFILE Exists Using Visual Basic (OO4O)

```
'Note that the PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraMusic As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters
```

```
OraParameters.Add "id", 1001, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyMusic", Empty, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
         END;", ORASQL_FAILEXEC)

Set OraMusic = OraParameters("MyMusic").Value

If OraMusic.Exists Then
    'Process the data
Else
   'Do error processing
End If
OraDb.Connection.CommitTrans
```

Example: See If the BFILE Exists Using Java (JDBC)

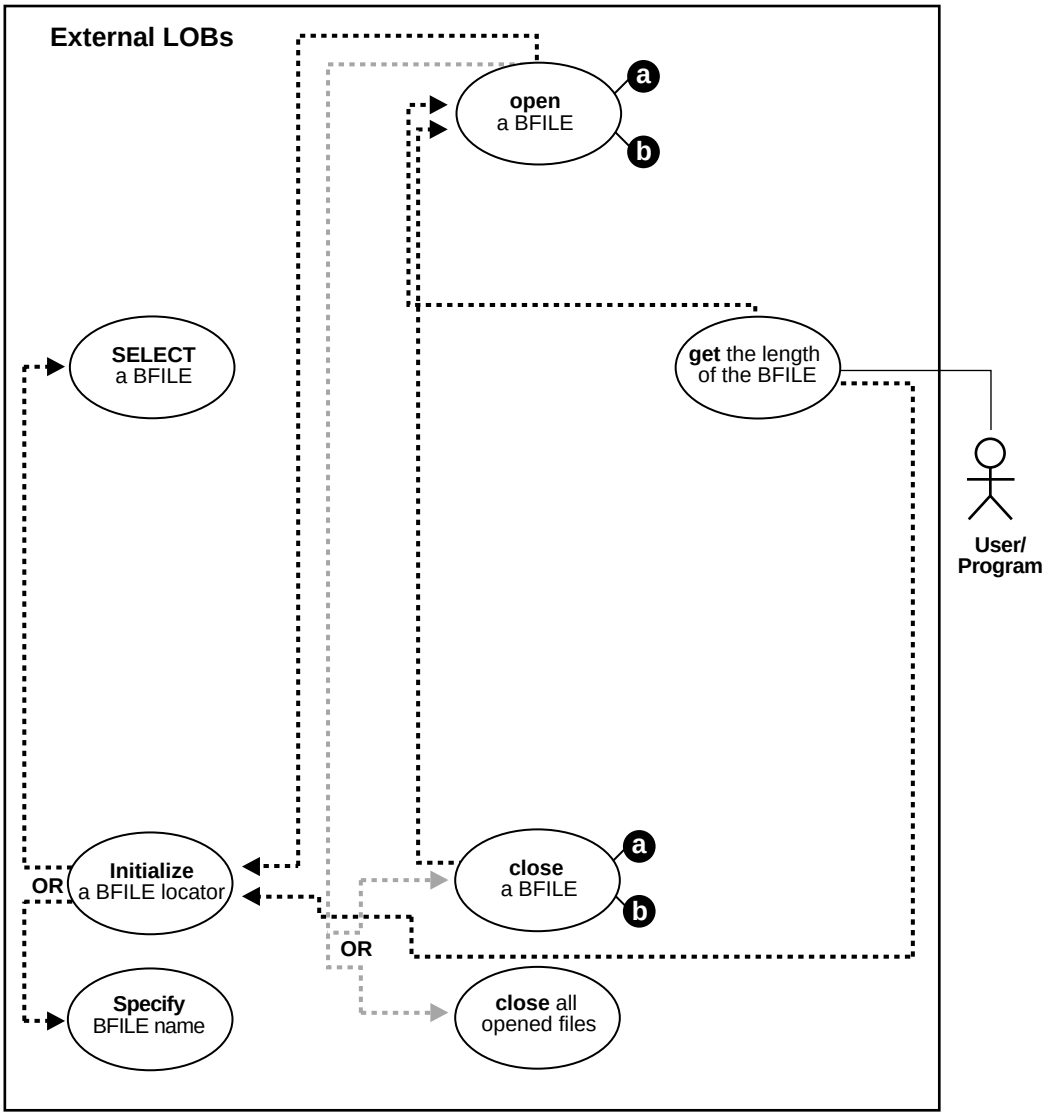
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_74
{

  static final int MAXBUFSIZE = 32767;
```

```
public static void main (String args [])
     throws Exception
{
  // Load the Oracle JDBC driver:
  Class.forName ("oracle.jdbc.driver.OracleDriver");

  // Connect to the database:
  Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

  // It's faster when auto commit is off:
  conn.setAutoCommit (false);

  // Create a Statement
  Statement stmt = conn.createStatement ();

  try
  {
     BFILE lob_loc = null;

     ResultSet rset = stmt.executeQuery (
        "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
     if (rset.next())
     {
        lob_loc = ((OracleResultSet)rset).getBFILE (1);
     }

     // See if the BFILE exists:
     Boolean exists = new Boolean(lob_loc.fileExists());
     System.out.println("Result from fileExists(): " + exists.toString());

     // Return the length of the BFILE:
     long length = lob_loc.length();
     System.out.println("Length of BFILE: " + Long.toString(length));

     // Get the directory alias for this BFILE:
     System.out.println("Directory alias: " + lob_loc.getDirAlias());

     // Get the file name for this BFILE:
     System.out.println("File name: " + lob_loc.getName());

     stmt.close();
     conn.commit();
     conn.close();
```

```
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Get the Length of a BFILE

Figure 5–24 Use Case Diagram: Get the length of the BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example gets the length of a BFILE that is associated with `Recording`.

- "Example: Get the Length of a BFILE Using PL/SQL (DBMS_LOB Package)" on page 5-137
- "Example: Get the Length of a BFILE Using C (OCI)" on page 5-138
- "Example: Get the Length of a BFILE Using COBOL (Pro*COBOL)" on page 5-139
- "Example: Get the Length of a BFILE Using COBOL (Pro*COBOL)" on page 5-139
- "Example: Get the Length of a BFILE Using C++ (Pro*C/C++)" on page 5-140
- "Example: Get the Length of a BFILE Using Visual Basic (OO4O)" on page 5-141
- "Example: Get the Length of a BFILE Using Java (JDBC)" on page 5-142

Example: Get the Length of a BFILE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure getLengthBFILE_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE getLengthBFILE_proc IS
   Lob_loc      BFILE;
   Length       INTEGER;
BEGIN
   /* Initialize the BFILE locator by selecting the LOB: */
   SELECT Mtab.Voiced_ref.Recording INTO Lob_loc FROM Multimedia_tab Mtab
      WHERE Mtab.Clip_ID = 3;
   /* Open the BFILE: */
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /* Get the length of the LOB: */
   Length := DBMS_LOB.GETLENGTH(Lob_loc);
   IF Length IS NULL THEN
       DBMS_OUTPUT.PUT_LINE('BFILE is null.');
   ELSE
       DBMS_OUTPUT.PUT_LINE('The length is ' || length);
   END IF;
```

```
   /* Close the BFILE: */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Get the Length of a BFILE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
ub4 BfileLength(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
      * allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     ub4 len;
```

```
    /* Allocate the locator descriptor */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
               (ub4) OCI_DTYPE_FILE,
               (size_t) 0, (dvoid **) 0)

    /* Select the bfile */
    selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
              "SELECT Mtab.Voiced_ref.Recording FROM Multimedia_tab Mtab
               WHERE Mtab.Clip_ID = 3");

    ub4 len;
    checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                   (ub1)OCI_FILE_READONLY));
    checkerr(errhp, OCILobGetLength(svchp, errhp, bfile_loc,
                                  &len));
    /* ... Do some processing. */
    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
    return(len);
}
```

Example: Get the Length of a BFILE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-LENGTH.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID         PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.
01  LEN            PIC S9(9) COMP.
01  D-LEN          PIC 9(4).
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-LENGTH.
```

```
           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
                CONNECT :USERID
           END-EXEC.

       * Allocate and initialize the BFILE locator:
           EXEC SQL ALLOCATE :BFILE1 END-EXEC.
           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
           EXEC SQL
                SELECT PHOTO INTO :BFILE1
                FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
           END-EXEC.

       * Use LOB DESCRIBE to get length of lob:
           EXEC SQL
                LOB DESCRIBE :BFILE1 GET LENGTH INTO :LEN
           END-EXEC.

           MOVE LEN TO D-LEN.
           DISPLAY "Length of BFILE is ", D-LEN.

        END-OF-BFILE.
           EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
           EXEC SQL FREE :BFILE1 END-EXEC.
           STOP RUN.

        SQL-ERROR.
           EXEC SQL
               WHENEVER SQLERROR CONTINUE
           END-EXEC.
           MOVE ORASLNR TO ORASLNRD.
           DISPLAY " ".
           DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
           DISPLAY " ".
           DISPLAY SQLERRMC.
           EXEC SQL
               ROLLBACK WORK RELEASE
           END-EXEC.
           STOP RUN.
```

Example: Get the Length of a BFILE Using C++ (Pro*C/C++)

```
#include <oci.h>
```

```
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void getLengthBFILE_proc()
{
  OCIBFileLocator *Lob_loc;
  unsigned int Length = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Length: */
  EXEC SQL LOB DESCRIBE :Lob_loc GET LENGTH INTO :Length;
  /* If the BFILE is NULL or unitialized, then Length is Undefined: */
  printf("Length is %d bytes\n", Length);
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getLengthBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Get the Length of a BFILE Using Visual Basic (OO4O)

```
'Note that the PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:
```

```
Dim MySession As OraSession
Dim OraDb As OraDatabase

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 1001, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyMusic", Empty, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
         END;", ORASQL_FAILEXEC)

Set OraMusic = OraParameters("MyMusic").Value

If OraMusic.Size = 0 Then
    MsgBox "BFile size is 0"
Else
    MsgBox "BFile size is " & OraMusic.Size
End If
OraDb.Connection.CommitTrans
```

Example: Get the Length of a BFILE Using Java (JDBC)

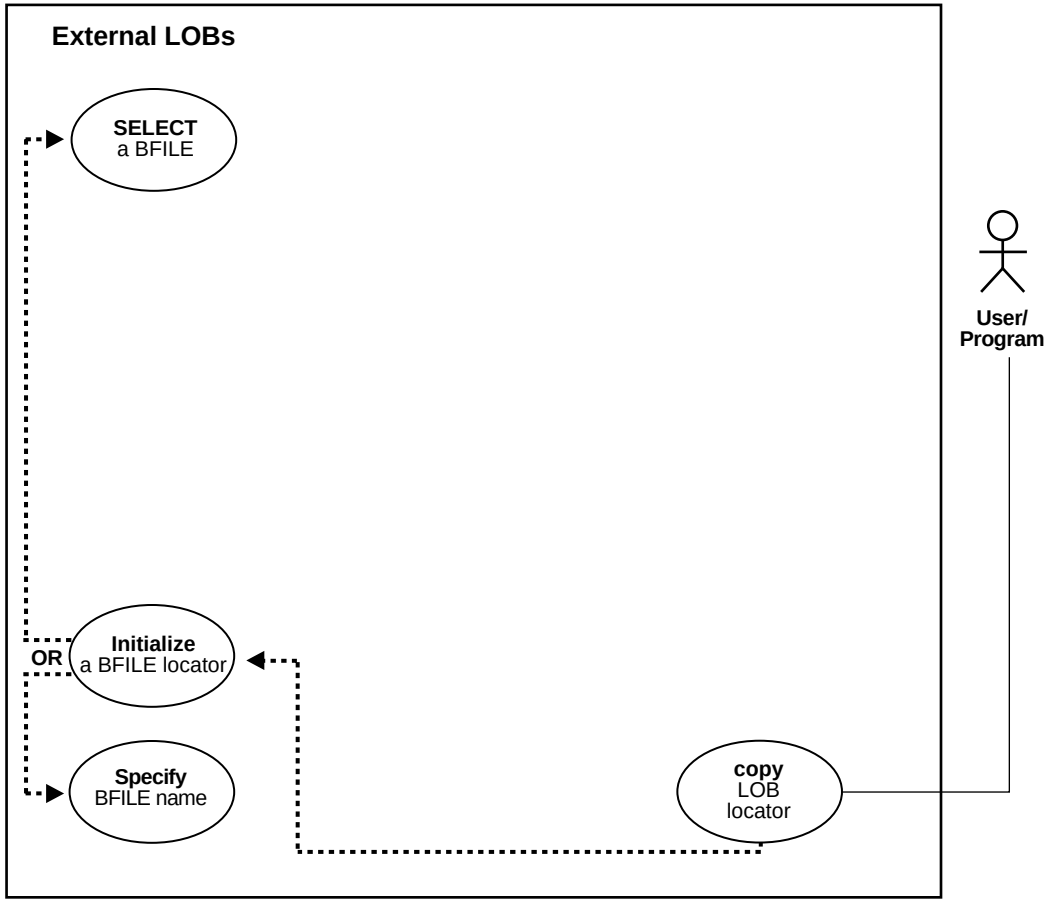
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_74
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE lob_loc = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }

       // See if the BFILE exists:
       Boolean exists = new Boolean(lob_loc.fileExists());
       System.out.println("Result from fileExists(): " + exists.toString());

       // Return the length of the BFILE:
       long length = lob_loc.length();
       System.out.println("Length of BFILE: " + Long.toString(length));

       // Get the directory alias for this BFILE:
```

```
            System.out.println("Directory alias: " + lob_loc.getDirAlias());

            // Get the file name for this BFILE:
            System.out.println("File name: " + lob_loc.getName());

            stmt.close();
            conn.commit();
            conn.close();

        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

Copy a LOB Locator for a BFILE

Figure 5–25 Use Case Diagram: Copy a LOB Locator for a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example assigns one BFILE locator to another related to `Photo`.

Example: Copy a LOB Locator for a BFILE Using PL/SQL

Note: Assigning one BFILE to another using PL/SQL entails using the "=" sign. This is an advanced topic that is discussed in more detail above with regard to "Read-Consistent Locators". on page 2-2

```
/* Note that the example procedure BFILEAssign_proc is not part of the
   DBMS_LOB package:  */
CREATE OR REPLACE PROCEDURE BFILEAssign_proc IS
   Lob_loc1    BFILE;
   Lob_loc2    BFILE;
BEGIN
   SELECT Photo INTO Lob_loc1 FROM Multimedia_tab WHERE Clip_ID = 3
      FOR UPDATE;
   /* Assign Lob_loc1 to Lob_loc2 so that they both refer to the same operating
      system file: */
   Lob_loc2 := Lob_loc1;
   /* Now you can read the bfile from either Lob_loc1 or Lob_loc2. */
END;
```

Example: Copy a LOB Locator for a BFILE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table: */
```

```
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement: */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column: */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement: */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
sword BfileAssign(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
      * allocated and initialized:
      */

     OCILobLocator *src_loc;
     OCILobLocator *dest_loc;

     /* Allocate the locator descriptors: */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &src_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &dest_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)
```

```
        /* Select the bfile: */
        selectLob(svchp, stmthp, errhp, dfnhp, src_loc,
                  "SELECT Photo FROM Multimedia_tab WHERE Clip_ID=3");

        /* Free the locator descriptors: */
        OCIDescriptorFree((dvoid *)src_loc, (ub4)OCI_DTYPE_FILE);
        OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_FILE);
        return (OCILobLocatorAssign(svchp, errhp, src_loc, &dst_loc));
        /* Note:  it is the caller's responsibilit to free the source
           and destination locator descriptors once the caller is done using them.
         */
}
```

Example: Copy a LOB Locator for a BFILE Using COBOL (Pro*COBOL)

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. BFILE-COPY-LOCATOR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  USERID         PIC X(11) VALUES "USER1/USER1".
       01  BFILE1         SQL-BFILE.
       01  BFILE2         SQL-BFILE.
       01  ORASLNRD       PIC 9(4).

           EXEC SQL INCLUDE SQLCA END-EXEC.
           EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
           EXEC SQL INCLUDE ORACA END-EXEC.

       PROCEDURE DIVISION.
       BILFE-COPY-LOCATOR.

           EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
           EXEC SQL
                CONNECT :USERID
           END-EXEC.

      * Allocate and initialize the BFILE locator:
           EXEC SQL ALLOCATE :BFILE1 END-EXEC.
           EXEC SQL ALLOCATE :BFILE2 END-EXEC.

           EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
```

```
            EXEC SQL
                 SELECT PHOTO INTO :BFILE1
                 FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
            END-EXEC.

            EXEC SQL
                 LOB ASSIGN :BFILE1 TO :BFILE2
            END-EXEC.

        END-OF-BFILE.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BFILE1 END-EXEC.
            EXEC SQL FREE :BFILE2 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Copy a LOB Locator for a BFILE Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void BFILEAssign_proc()
```

```
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Photo INTO :Lob_loc1
           FROM Multimedia_tab WHERE Clip_ID = 3;
  /* Assign Lob_loc1 to Lob_loc2 so that they both refer to the same
     operating system file:  */
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* Now you can read the BFILE from either Lob_loc1 or Lob_loc2 */
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILEAssign_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Copy a LOB Locator for a BFILE Using Visual Basic (OO4O)

Note: A Visual Basic (OO4O) example will made available in a subsequent release.

Example: Copy a LOB Locator for a BFILE Using Java (JDBC)

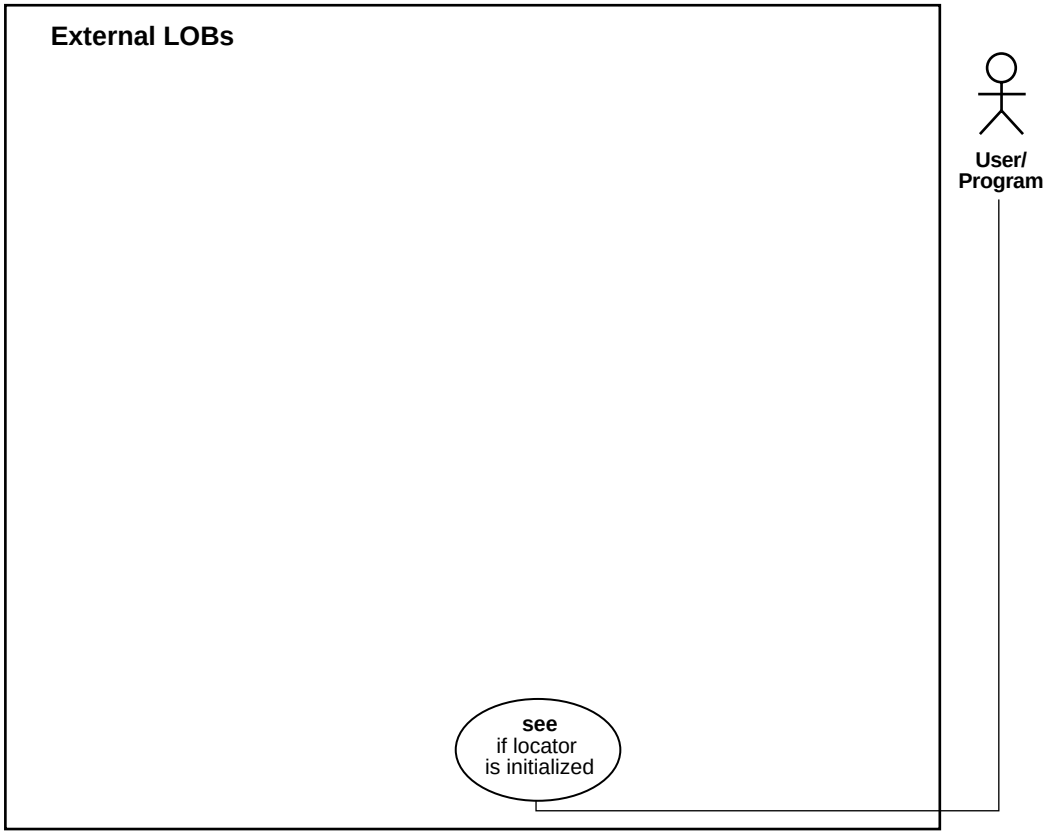
```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;
```

```
public class Ex4_81
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
       if (rset.next())
       {
          lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
       }

       // Assign lob_loc1 to lob_loc2 so that they both refer
       // to the same operating system file.
       // Now the BFILE can be read through either of the locators:
       lob_loc2 = lob_loc1;

       stmt.close();
       conn.commit();
       conn.close();

    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
```

```
    }
}
```

See If a LOB Locator for a BFILE Is Initialized

Figure 5–26 Use Case Diagram: See If a LOB Locator Is Initialized



To refer to the table of all basic operations having to do with External LOBs (BFILES):

- "Use Case Model: External LOBs" on page 5-2

Scenario

Before you call any of the OCILob* interfaces (such as OCILobWrite), or any of the programmatic environments that make use of the OCILob* interfaces, you must first initialize the LOB locator, via a SELECT, for example. So, if your application requires for a locator to be passed from one function to another, you may want to verify that the locator has already been initialized. If it turns out the locator is not initialized, you could design your application either to return an error or to perform the SELECT before calling the OCILob* interface.

- "Example: See If a LOB Locator for a BFILE Is Initialized Using C (OCI)" on page 5-154
- "Example: See If a LOB Locator for a BFILE Is Initialized Using C++ (Pro*C/C++)" on page 5-154

Example: See If a LOB Locator for a BFILE Is Initialized Using C (OCI)

```
boolean BfileIsInit(envhp, svchp, errhp, bfile_loc)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIError *errhp;
OCILobLocator *bfile_loc;  /* This is the BFILE locator that is already
                              allocated and initialized. */
{
   boolean is_init;
   checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, bfile_loc, &is_init));
   return(is_init);
}
```

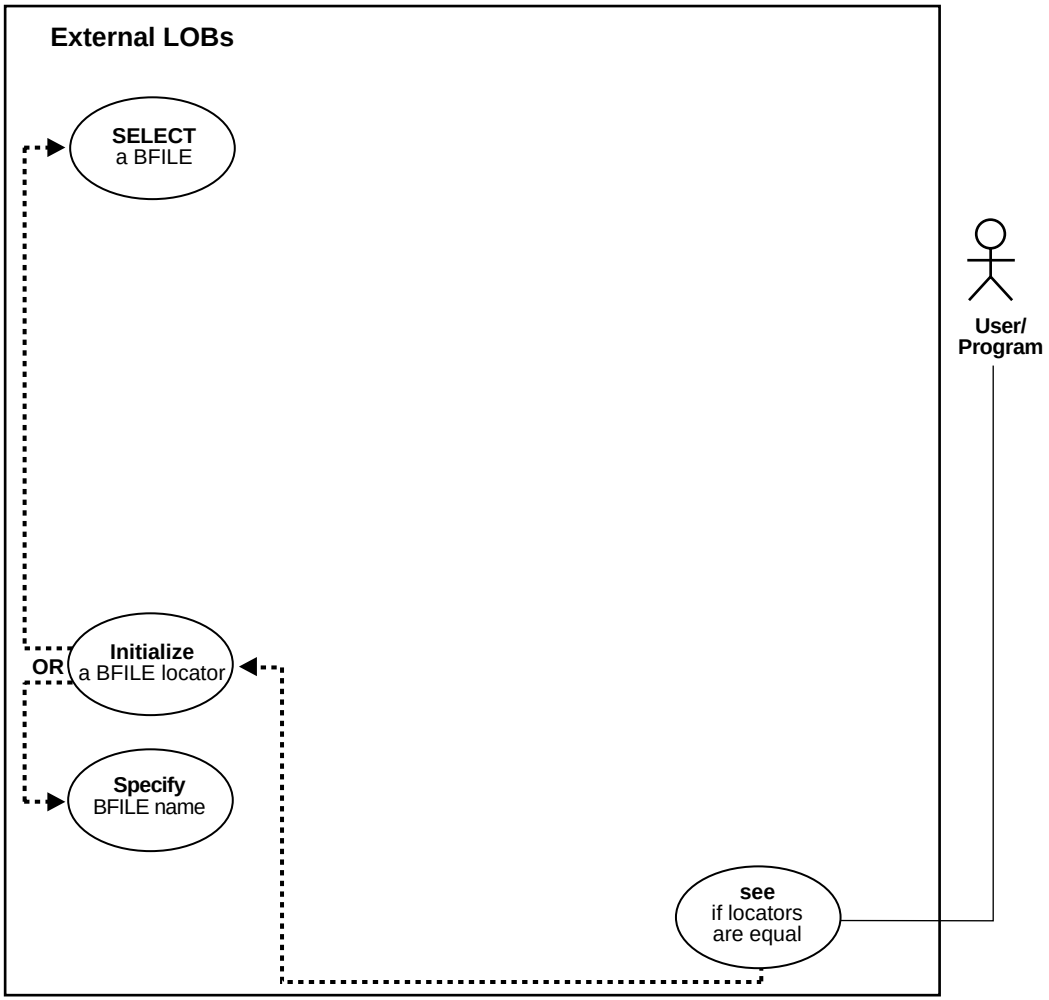
Example: See If a LOB Locator for a BFILE Is Initialized Using C++ (Pro*C/C++)

```
/* Pro*C/C++ has no form of embedded SQL statement to determine if a BFILE
   locator is initialized.  Locators in Pro*C/C++ are initialized when they
   are allocated via the EXEC SQL ALLOCATE statement. However, an example
   can be written that uses embedded SQL and the OCI as is shown here: */
#include <sql2oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
```

```
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void BFILELocatorIsInit_proc()
{
  OCIBFileLocator *Lob_loc;
  OCIEnv *oeh;
  OCIError *err;
  boolean isInitialized = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO :Lob_loc
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Allocate the OCI Error Handle: */
  (void) OCIHandleAlloc((dvoid *)oeh, (dvoid **)&err,
                        (ub4)OCI_HTYPE_ERROR, (ub4)0, (dvoid **)0);
  /* Use the OCI to determine if the locator is Initialized: */
  (void) OCILobLocatorIsInit(oeh, err, Lob_loc, &isInitialized);
  if (isInitialized)
    printf("Locator is initialized\n");
  else
    printf("Locator is not initialized\n");
  /* Note that in this example, the locator is initialized: */
  /* Deallocate the OCI Error Handle: */
  (void) OCIHandleFree(err, OCI_HTYPE_ERROR);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILELocatorIsInit_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

See If One LOB Locator for a BFILE Is Equal to Another

Figure 5–27 Use Case Diagram: See If One LOB Locator for a BFILE Is Equal to Another



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

If two locators are equal, this means that they refer to the same version of the LOB data (see "Read-Consistent Locators" on page 2-2).

- "Example: See If One LOB Locator for a BFILE Is Equal to Another Using C (OCI)" on page 5-157
- "Example: See If One LOB Locator for a BFILE Is Equal to Another Using C++ (Pro*C / C++)" on page 5-157
- "Example: See If One LOB Locator for a BFILE Is Equal to Another Using Java (JDBC)" on page 5-159

Example: See If One LOB Locator for a BFILE Is Equal to Another Using C (OCI)

```
boolean BfileIsEqual(envhp, errhp, bfile_loc1, bfile_loc2)
OCIEnv *envhp;
OCIError *errhp;
OCILobLocator *bfile_loc1;    /* BFILE Locator 1 that is already allocated */
OCILobLocator *bfile_loc2;    /* BFILE Locator 2 that is already allocated */
{
  boolean is_equal;
  OCILobIsEqual(envhp, bfile_loc1, bfile_loc2, &is_equal);
  return(is_equal);
}
```

Example: See If One LOB Locator for a BFILE Is Equal to Another Using C++ (Pro*C/C++)

```
/* Pro*C/C++ does not provide a mechanism to test the equality of two
   locators However, by using the OCI directly, two locators can be
   compared to determine whether or not they are equal as this example
   demonstrates: */

#include <sql2oci.h>
#include <stdio.h>
```

```
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void BFILELocatorIsEqual_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;
  OCIEnv *oeh;
  boolean isEqual = 0;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  EXEC SQL SELECT Photo INTO :Lob_loc1
           FROM Multimedia_tab WHERE Clip_ID = 3;
  EXEC SQL LOB ASSIGN :Lob_loc1 TO :Lob_loc2;
  /* Now you can read the BFILE from either Lob_loc1 or Lob_loc2 */
  /* Get the OCI Environment Handle using a SQLLIB Routine: */
  (void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
  /* Call OCI to see if the two locators are Equal: */
  (void) OCILobIsEqual(oeh, Lob_loc1, Lob_loc2, &isEqual);
  if (isEqual)
    printf("Locators are equal\n");
  else
    printf("Locators are not equal\n");
  /* Note that in this example, the LOB locators will be Equal: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  BFILELocatorIsEqual_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

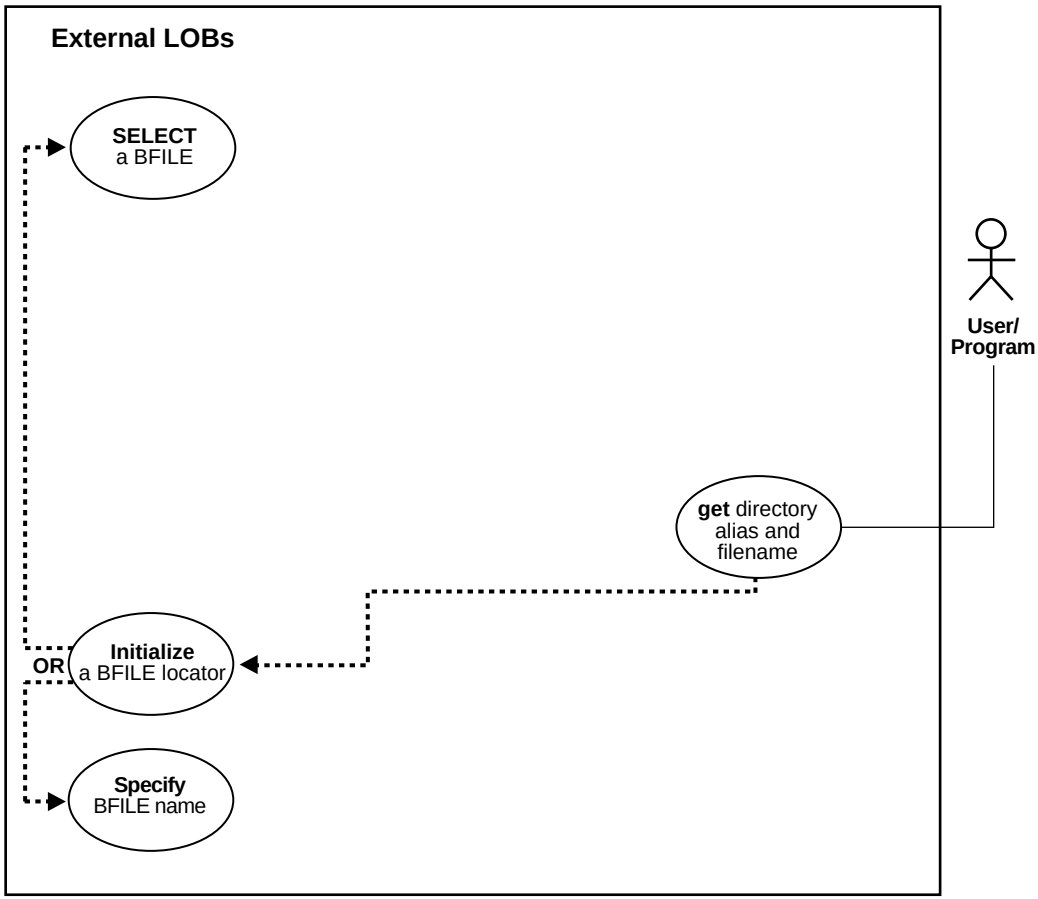
Example: See If One LOB Locator for a BFILE Is Equal to Another Using Java (JDBC)

```
// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_89
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE lob_loc1 = null;
       BFILE lob_loc2 = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
      if (rset.next())
      {
          lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
      }
```

```
      // Set both LOBS to reference the same BFILE:
      lob_loc2 = lob_loc1;

      // Note that in this example, the Locators will be equal:
      if (lob_loc1.equals(lob_loc2))
      {
         // The Locators are equal:
         System.out.println("The BFILEs are equal");
      }
      else
      {
         // The Locators are different:
         System.out.println("The BFILEs are NOT equal");
      }

      stmt.close();
      conn.commit();
      conn.close();

    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Get Directory Alias and Filename

Figure 5–28 Use Case Diagram: Get Directory Alias and Filename



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example retrieves the directory alias and filename related to the `BFILE`, `Music`.

Example: Get Directory Alias and Filename Using PL/SQL

```
CREATE OR REPLACE PROCEDURE getNameBFILE_proc IS
   Lob_loc         BFILE;
   DirAlias_name   VARCHAR2(30);
   File_name       VARCHAR2(40);
BEGIN
   SELECT Music INTO Lob_loc FROM Multimedia_tab WHERE Clip_ID = 3;
   DBMS_LOB.FILEGETNAME(Lob_loc, DirAlias_name, File_name);
   /* do some processing based on the directory alias and file names */
END;
```

Example: Get Directory Alias and Filename Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table: */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement: */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
```

```
                                        (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column: */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                        (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                        (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                        OCI_DEFAULT));

     /* Execute the SQL select statement: */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                        (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                        (ub4) OCI_DEFAULT));
}
void BfileGetDirFile(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;
     OraText dir_alias[32] = NULL;
     ub2     d_length = 32;
     OraText filename[256] = NULL;
     ub2     f_length = 256;

     /* Allocate the locator descriptor: */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                               (ub4) OCI_DTYPE_FILE,
                               (size_t) 0, (dvoid **) 0)

     /* Select the bfile: */
     selectLob(svchp, stmthp, errhp, dfnhp, bfile_loc,
               "SELECT Music FROM Multimedia_tab WHERE Clip_ID=3");

     OCILobFileGetName(envhp, errhp, bfile_loc, dir_alias, &d_length,
                       filename, &f_length);

     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Example: Get Directory Alias and Filename Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-DIR-ALIAS.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID         PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(30) VARYING.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-DIR-ALIAS.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.

* Populate the BFILE locator:
    EXEC SQL
         SELECT PHOTO INTO :BFILE1
         FROM MULTIMEDIA_TAB WHERE CLIP_ID = 3
    END-EXEC.

* Use the LOB DESCRIBE functionality to get
* the directory alias and the filename:
    EXEC SQL
         LOB DESCRIBE :BFILE1
         GET DIRECTORY, FILENAME INTO :DIR-ALIAS, :FNAME
    END-EXEC.
```

```
            DISPLAY "DIRECTORY: ", DIR-ALIAS-ARR, "FNAME: ", FNAME-ARR.

        END-OF-BFILE.
            EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
            EXEC SQL FREE :BFILE1 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Get Directory Alias and Filename Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void getBFILEDirectoryAndFilename_proc()
{
  OCIBFileLocator *Lob_loc;
  char Directory[31], Filename[255];
  /* Datatype Equivalencing is Optional: */
  EXEC SQL VAR Directory IS STRING;
  EXEC SQL VAR Filename IS STRING;
```

```
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  /* Select the BFILE: */
  EXEC SQL SELECT Photo INTO :Lob_loc
     FROM Multimedia_tab WHERE Clip_ID = 3;
  /* Open the BFILE: */
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* Get the Directory Alias and Filename: */
  EXEC SQL LOB DESCRIBE :Lob_loc
     GET DIRECTORY, FILENAME INTO :Directory, :Filename;
  /* Close the BFILE: */
  EXEC SQL LOB CLOSE :Lob_loc;
  printf("Directory Alias: %s\n", Directory);
  printf("Filename: %s\n", Filename);
  /* Release resources held by the locator: */
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  getBFILEDirectoryAndFilename_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Get Directory Alias and Filename Using Visual Basic (OO4O)

```
'Note that the PL/SQL packages and the tables mentioned here are not part of the
'standard OO4O installation:

Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraMusic1 As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 1001, ORAPARM_INPUT
```

```
'Define out parameter of BFILE type:
OraParameters.Add "MyMusic", Empty, ORAPARM_OUTPUT
OraParameters("MyMusic").ServerType = ORATYPE_BFILE

Set OraSql =
   OraDb.CreateSql(
      "BEGIN SELECT music INTO :MyMusic FROM multimedia_tab WHERE clip_id = :id;
         END;", ORASQL_FAILEXEC)

Set OraMusic1 = OraParameters("MyMusic").Value
'Get Directory alias and filename:
MsgBox " Directory alias is " & OraMusic1.DirectoryName &
   " Filename is " & OraMusic1.filename

OraDb.Connection.CommitTrans
```

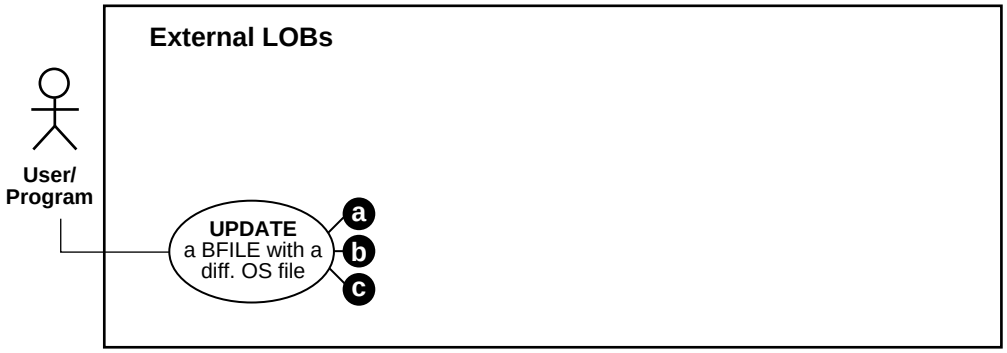
Example: Get Directory Alias and Filename Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_74
{

  static final int MAXBUFSIZE = 32767;

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
```

```
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
       BFILE lob_loc = null;

       ResultSet rset = stmt.executeQuery (
          "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
       if (rset.next())
       {
          lob_loc = ((OracleResultSet)rset).getBFILE (1);
       }
       // See if the BFILE exists:
       Boolean exists = new Boolean(lob_loc.fileExists());
       System.out.println("Result from fileExists(): " + exists.toString());

       // Return the length of the BFILE:
       long length = lob_loc.length();
       System.out.println("Length of BFILE: " + Long.toString(length));

       // Get the directory alias for this BFILE:
       System.out.println("Directory alias: " + lob_loc.getDirAlias());

       // Get the file name for this BFILE:
       System.out.println("File name: " + lob_loc.getName());
       stmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
  }
}
```

Three Ways to Update a Row Containing a BFILE

Figure 5–29 Use Case Diagram: Three Ways to Update a Row Containing a BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Note that you must initialize the `BFILE` either to `NULL` or to a directory alias and filename.

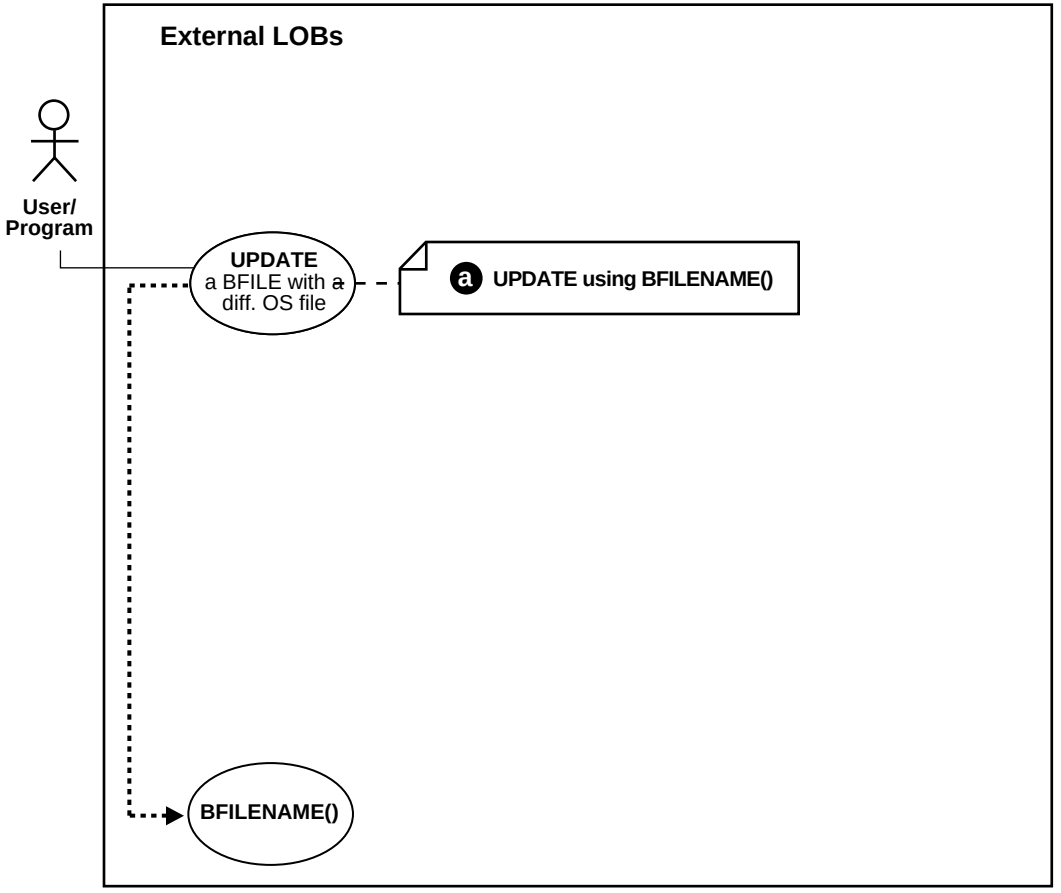
a. "UPDATE a BFILE Using BFILENAME()" on page 5-170

b. "UPDATE a BFILE as SELECT" on page 5-173

c. "UPDATE a BFILE by Initializing a BFILE Locator" on page 5-174

UPDATE a BFILE Using BFILENAME()

Figure 5–30 Use Case Diagram: UPDATE a BFILE Using BFILENAME()



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

BFILENAME() Function

The BFILENAME() function can be called as part of SQL INSERT or UPDATE to initialize a BFILE column or attribute for a particular row by associating it with a physical file in the server's filesystem.

The DIRECTORY object represented by the directory_alias parameter to this function need not already be defined using SQL DDL before the BFILENAME() function is called in SQL DML or a PL/SQL program. However, the directory object and operating system file must exist by the time you actually use the BFILE locator (for example, as having been used as a parameter to an operation such as OCILobFileOpen(), DBMS_LOB.FILEOPEN(), OCILobOpen(), or DBMS_LOB.OPEN()).

Note that BFILENAME() does not validate privileges on this DIRECTORY object, or check if the physical directory that the DIRECTORY object represents actually exists. These checks are performed only during file access using the BFILE locator that was initialized by the BFILENAME() function.

You can use BFILENAME() as part of a SQL INSERT and UPDATE statement to initialize a BFILE column. You can also use it to initialize a BFILE locator variable in a PL/SQL program, and use that locator for file operations. However, if the corresponding directory alias and/or filename does not exist, then PL/SQL DBMS_LOB routines that use this variable will generate errors.

The directory_alias parameter in the BFILENAME() function must be specified taking case-sensitivity of the directory name into consideration.

See Also: "DIRECTORY Name Specification" on page 5-7

Syntax

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

See Also: "DIRECTORY Name Specification" on page 5-7 for information about the use of uppercase letters in the directory name, and OCILobFileSetName() in *Oracle Call Interface Programmer's Guide* for an equivalent OCI based routine.

Scenario

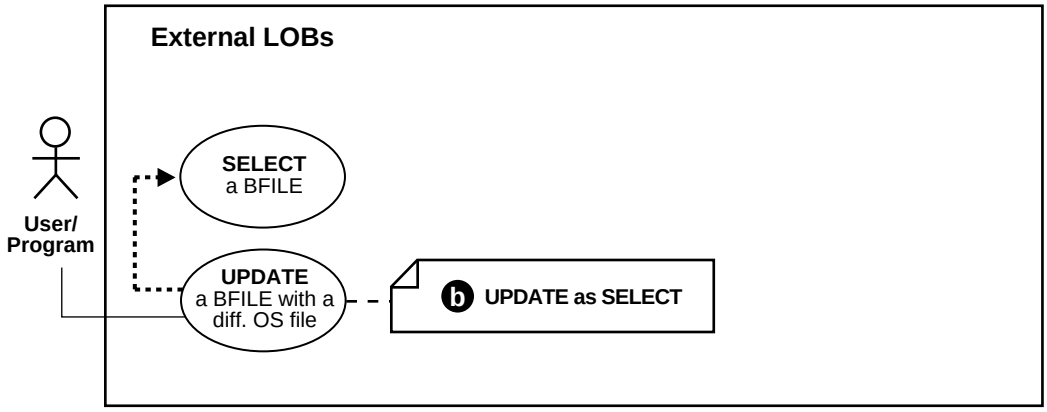
This example updates Multimedia_tab by means of the BFILENAME function.

Example: Update a BFILE by means of BFILENAME() Using SQL

```
UPDATE Multimedia_tab
   SET Photo = BFILENAME('PHOTO_DIR', 'Nixon_photo') where Clip_ID = 3;
```

UPDATE a BFILE as SELECT

Figure 5–31 Use Case Diagram: UPDATE a BFILE as SELECT



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

There is no copy function for BFILEs, so you have to use UPDATE as SELECT if you want to copy a BFILE from one location to another. Because BFILEs use reference semantics instead of copy semantics, only the BFILE locator is copied from one row to another row. This means that you cannot make a copy of an external LOB value without issuing an operating system command to copy the operating system file.

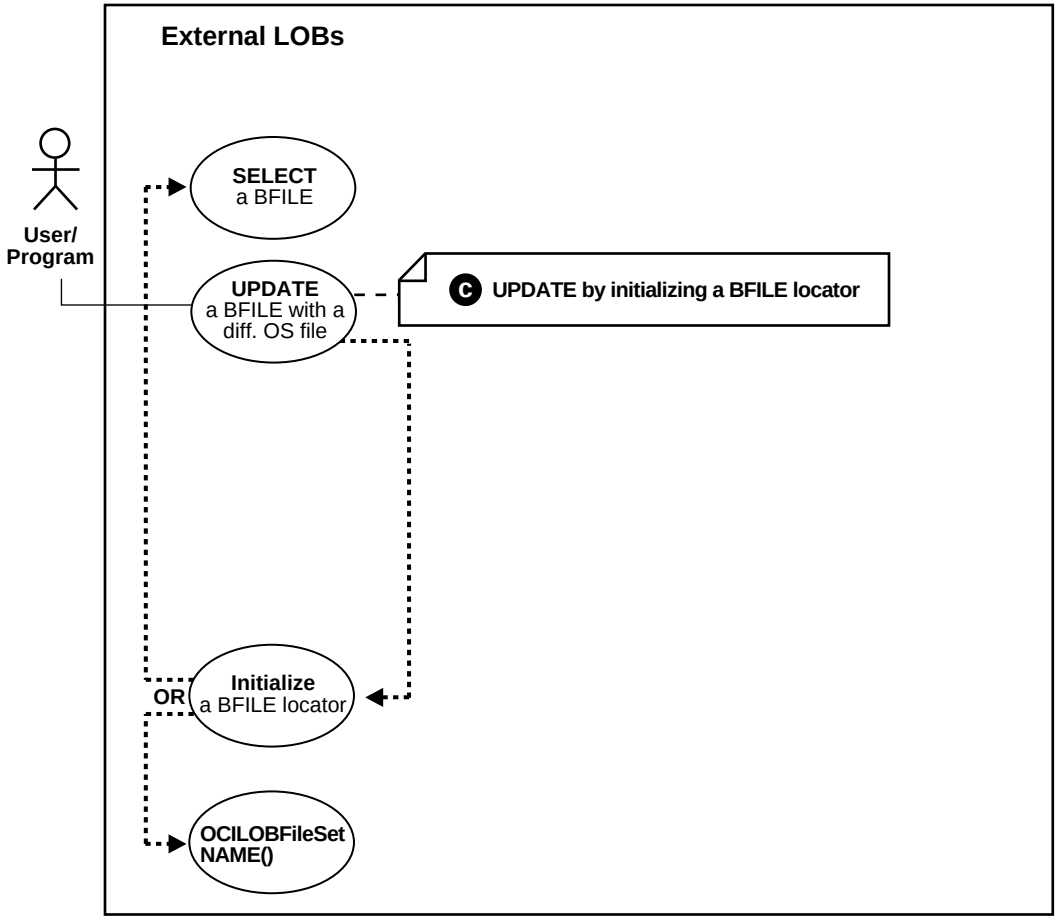
This example updates the table, Voiceover_tab by selecting from the archival storage table, VoiceoverLib_tab

Example: Update a BFILE as Select Using SQL

```
UPDATE Voiceover_tab
    SET (originator,script,actor,take,recording) =
       (SELECT * FROM VoiceoverLib_tab VLtab WHERE VLtab.Take = 101);
```

UPDATE a BFILE by Initializing a BFILE Locator

Figure 5–32 Use Case Diagram: UPDATE a BFILE by Initializing a BFILE Locator



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

Note that you must initialize the `BFILE` locator bind variable to a directory alias and filename before issuing the update statement.

- "Example: Update a BFILE by Initializing a BFILE Locator Using PL/SQL" on page 5-175
- "Example: Update a BFILE by Initializing a BFILE Locator Using C (OCI)" on page 5-175
- "Example: Update a BFILE by Initializing a BFILE Locator Using COBOL (Pro*COBOL)" on page 5-176
- "Example: Update a BFILE by Initializing a BFILE Locator Using C++ (Pro*C/C++)" on page 5-178
- "Example: Update a BFILE by Initializing a BFILE Locator Using Visual Basic (OO4O)" on page 5-179
- "Example: Update a BFILE by Initializing a BFILE Locator Using Java (JDBC)" on page 5-180

Example: Update a BFILE by Initializing a BFILE Locator Using PL/SQL

```
/* Note that the example procedure updateUseBindVariable_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (Lob_loc BFILE) IS
BEGIN
   UPDATE Multimedia_tab SET Photo = Lob_loc WHERE Clip_ID = 3;
END;

DECLARE
   Lob_loc  BFILE;
BEGIN
   SELECT Photo INTO Lob_loc
      FROM Multimedia_tab
         WHERE Clip_ID = 1;
   updateUseBindVariable_proc (Lob_loc);
   COMMIT;
END;
```

Example: Update a BFILE by Initializing a BFILE Locator Using C (OCI)

```
void BfileUpdate(envhp, errhp, svchp, stmthp)
```

```
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
  OCILobLocator *Lob_loc;
  OCIBind *bndhp;

  text  *updstmt =
     (text *) "UPDATE Multimedia_tab SET Photo = :Lob_loc WHERE Clip_ID = 1";

  OraText *Dir = (OraText *)"PHOTO_DIR", *Name = (OraText *)"Washington_photo";

  /* Prepare the SQL statement: */
  checkerr (errhp, OCIStmtPrepare(stmthp, errhp, updstmt,  (ub4)
                                  strlen((char *) updstmt),
                                  (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

  /* Allocate Locator resources: */
  (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                            (ub4)OCI_DTYPE_FILE, (size_t) 0, (dvoid **) 0);

  checkerr (errhp, OCILobFileSetName(envhp, errhp, &Lob_loc,
                                     Dir, (ub2)strlen((char *)Dir),
                                     Name,(ub2)strlen((char *)Name)));

  checkerr (errhp, OCIBindByPos(stmthp, &bndhp, errhp, (ub4) 1,
                                (dvoid *) &Lob_loc, (sb4) 0,  SQLT_BFILE,
                                (dvoid *) 0, (ub2 *)0, (ub2 *)0,
                                (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT));

 /* Execute the SQL statement: */
  checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                  (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                  (ub4) OCI_DEFAULT));

 /* Free LOB resources: */
  OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_FILE);
}
```

Example: Update a BFILE by Initializing a BFILE Locator Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-UPDATE.
```

```
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID         PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.
01  BFILE-IND      PIC S9(4) COMP.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(30) VARYING.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-UPDATE.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locator:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* Populate the BFILE:
    EXEC SQL WHENEVER NOT FOUND GOTO END-OF-BFILE END-EXEC.
    EXEC ORACLE OPTION (SELECT_ERROR=NO) END-EXEC.
    EXEC SQL
         SELECT PHOTO INTO :BFILE1:BFILE-IND
         FROM MULTIMEDIA_TAB WHERE CLIP_ID = 1
    END-EXEC.

* Make photo associated with clip_id=3 same as clip_id=1:
    EXEC SQL
         UPDATE MULTIMEDIA_TAB SET PHOTO = :BFILE1:BFILE-IND
         WHERE CLIP_ID = 3
    END-EXEC.

* Free the BFILE:
END-OF-BFILE.
    EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
    EXEC SQL FREE :BFILE1 END-EXEC.
    EXEC SQL
```

```
            COMMIT WORK RELEASE
        END-EXEC.
        STOP RUN.

    SQL-ERROR.
        EXEC SQL
            WHENEVER SQLERROR CONTINUE
        END-EXEC.
        MOVE ORASLNR TO ORASLNRD.
        DISPLAY " ".
        DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
        DISPLAY " ".
        DISPLAY SQLERRMC.
        EXEC SQL
            ROLLBACK WORK RELEASE
        END-EXEC.
        STOP RUN.
```

Example: Update a BFILE by Initializing a BFILE Locator Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void updateUseBindVariable_proc(Lob_loc)
  OCIBFileLocator *Lob_loc;
{
  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL UPDATE Multimedia_tab SET Photo = :Lob_loc WHERE Clip_ID = 3;
}

void updateBFILE_proc()
{
  OCIBFileLocator *Lob_loc;

  EXEC SQL ALLOCATE :Lob_loc;
```

```
  EXEC SQL SELECT Photo INTO :Lob_loc
           FROM Multimedia_tab WHERE Clip_ID = 1;
  updateUseBindVariable_proc(Lob_loc);
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  updateBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Update a BFILE by Initializing a BFILE Locator Using Visual Basic (OO4O)

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraParameters As OraParameters, OraPhoto As OraBfile

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

'Define in out parameter of BFILE type:
OraParameters.Add "MyPhoto", Empty, ORAPARM_BOTH, ORATYPE_BFILE

'Define out parameter of BFILE type:
OraDb.ExecuteSQL (
   "BEGIN SELECT Photo INTO :MyPhoto FROM Multimedia_tab WHERE Clip_ID = 1;
      END;")

'Update the photo BFile for clip_id=1 to clip_id=1001:
OraDb.ExecuteSQL (
   "UPDATE Multimedia_tab SET Photo = :MyPhoto WHERE Clip_ID = 1001")

'Get Directory alias and filename
'MsgBox " Directory alias is " & OraMusic1.DirectoryName & " Filename is " &
OraMusic1.filename

OraDb.Connection.CommitTrans
```

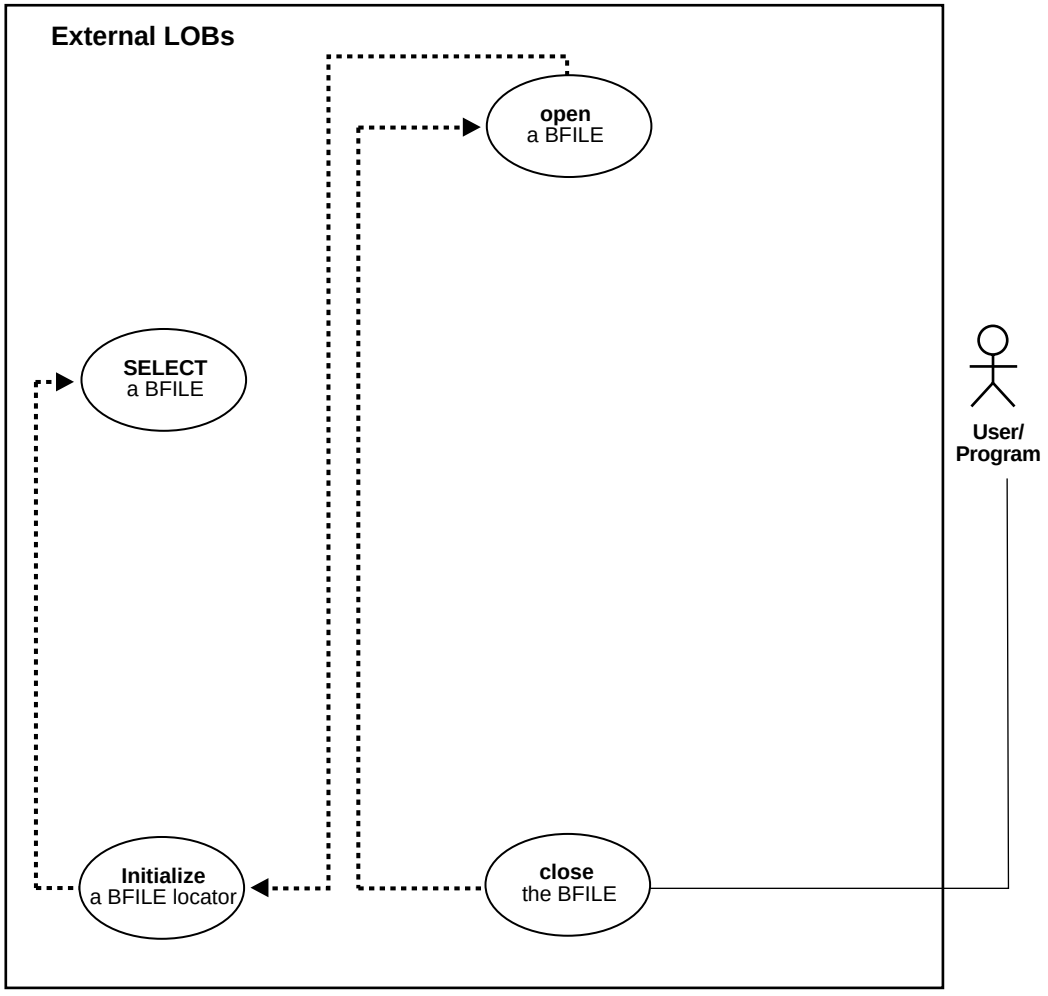
Example: Update a BFILE by Initializing a BFILE Locator Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_100
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
        BFILE src_lob = null;
        ResultSet rset = null;
        OracleCallableStatement cstmt = null;

        rset = stmt.executeQuery (
           "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
        if (rset.next())
```

```
        {
           src_lob = ((OracleResultSet)rset).getBFILE (1);
        }

        // Prepare a CallableStatement to OPEN the LOB for READWRITE:
        cstmt = (OracleCallableStatement) conn.prepareCall (
           "UPDATE multimedia_tab SET photo = ? WHERE clip_id = 1");
        cstmt.setBFILE(1, src_lob);
        cstmt.execute();

       //Close the statements and commit the transaction:
       stmt.close();
       cstmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Two Ways to Close a BFILE

Figure 5–33 Use Case Diagram: Two Ways to See If a BFILE is Open



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

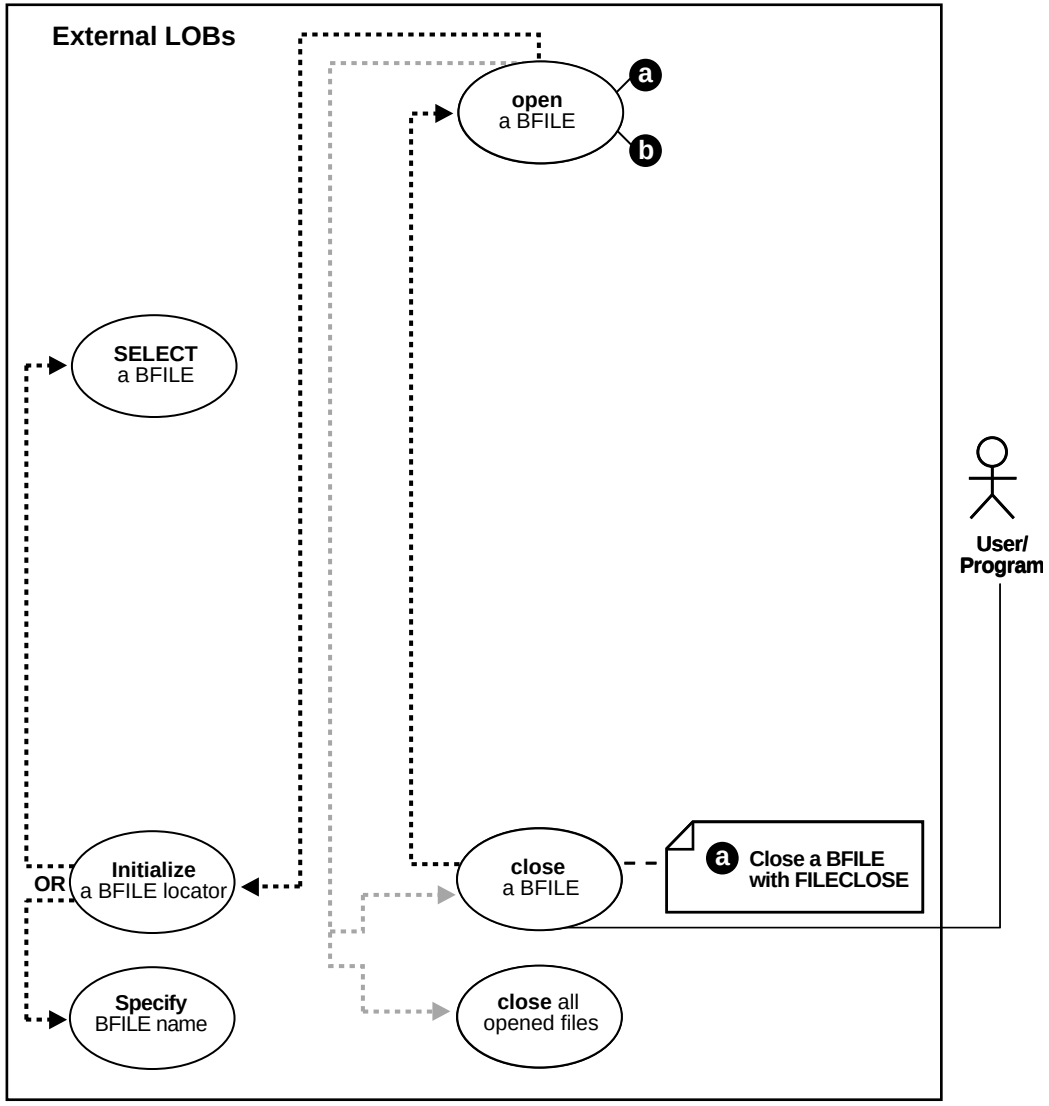
As you can see by comparing the code, these alternative methods are very similar. However, while you can continue to use the older `FILECLOSE` form, we strongly recommend that you switch to using `CLOSE`, because this facilitates future extensibility.

a. "Close a BFILE with FILECLOSE" on page 5-184

b. "Close a BFILE with CLOSE" on page 5-189

Close a BFILE with FILECLOSE

Figure 5–34 Use Case Diagram: Close an Open BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

While you can continue to use the older `FILECLOSE` form, we *strongly recommend* that you switch to using `CLOSE`, because this facilitate future extensibility. This example can be read in conjunction with the example of opening a `BFILE`.

- "Example: Close a BFile with FILECLOSE Using PL/SQL (DBMS_LOB Package)" on page 5-185
- "Example: Close a BFile with FILECLOSE Using C (OCI)" on page 5-185
- "Example: Close a BFile with FILECLOSE Using Visual Basic (OO4O)" on page 5-187
- "Example: Close a BFile with FILECLOSE Using Java (JDBC)" on page 5-187

Example: Close a BFile with FILECLOSE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure closeBFILE_procOne is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE closeBFILE_procOne IS
   Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
   DBMS_LOB.FILEOPEN(Lob_loc, DBMS_LOB.FILE_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.FILECLOSE(Lob_loc);
END;
```

Example: Close a BFile with FILECLOSE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
OCILobLocator *Lob_loc;
text          *selstmt;
```

```
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileClose(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv       *envhp;
OCISvcCtx    *svchp;
OCIStatement *stmthp;
OCIError     *errhp;
OCIDefine    *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
      * allocated and initialized.
      */

     OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Set the bfile locator information */
     checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                          (OraText *)"PHOTO_DIR", (ub2)strlen("PHOTO_DIR"),
                          (OraText *)"Lincoln_photo",
                          (ub2)strlen("Lincoln_photo"))));

     checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

     /* Free the locator descriptor */
     OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
```

```
}
```

Example: Close a BFile with FILECLOSE Using Visual Basic (OO4O)

Note: At the present time, OO4O only offers BFILE closing with CLOSE (see below).

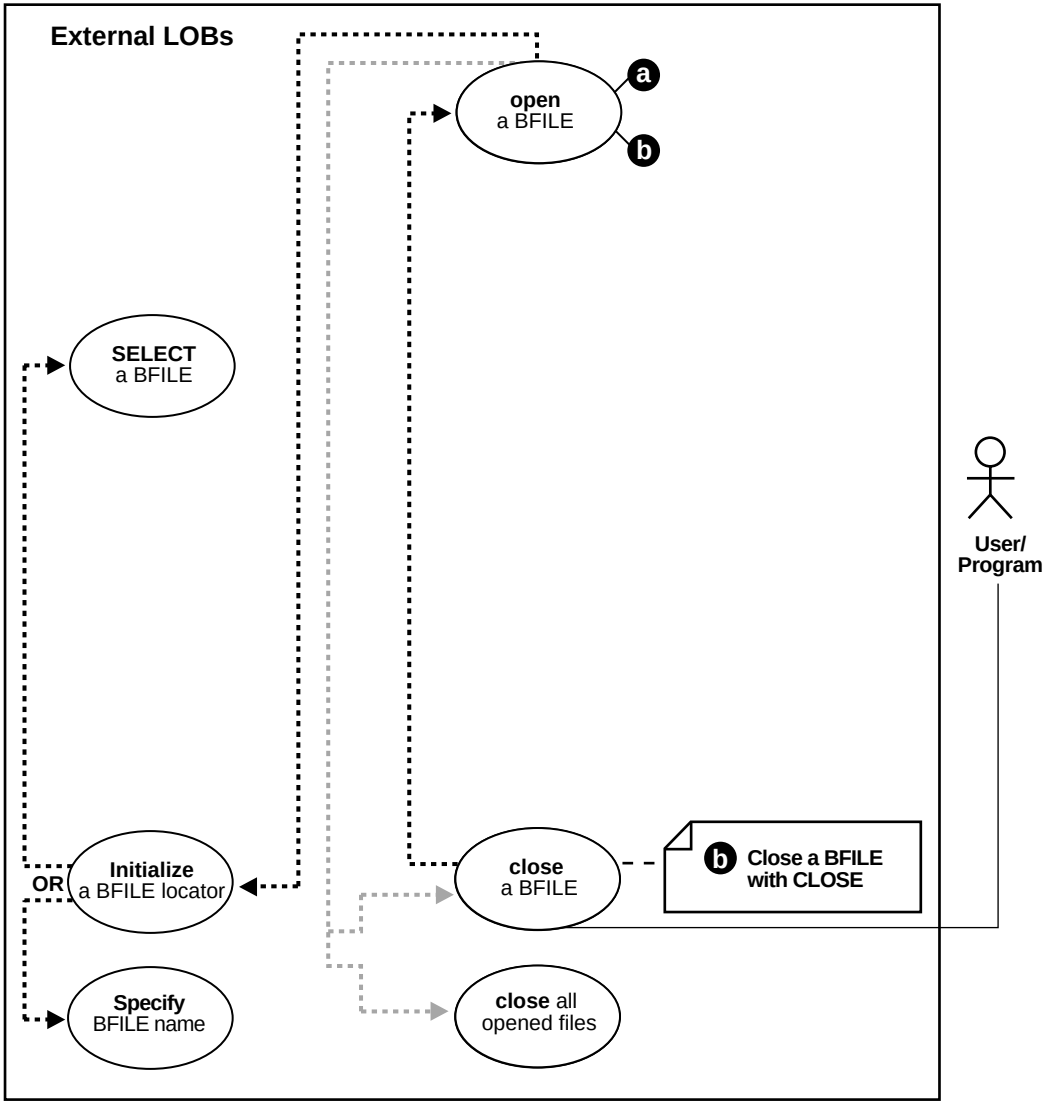
Example: Close a BFile with FILECLOSE Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_45
{
  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
```

```
    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
       if (rset.next())
       {
          src_lob = ((OracleResultSet)rset).getBFILE (1);
       }

       result = new Boolean(src_lob.plsql_fileIsOpen());
       System.out.println(
          "result of fileIsOpen() before opening file : " + result.toString());

       src_lob.plsql_fileOpen();

       result = new Boolean(src_lob.plsql_fileIsOpen());
       System.out.println(
          "result of fileIsOpen() after opening file : " + result.toString());

       // Close the BFILE,  statement and connection:
       src_lob.plsql_fileClose();
       stmt.close();
       conn.commit();
       conn.close();
    }
    catch (SQLException e)
    {
       e.printStackTrace();
    }
  }
}
```

Close a BFILE with CLOSE

Figure 5–35 Use Case Diagram: Close an Open BFILE



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

This example should be read in conjunction with the example of opening a BFILE — in this case, closing the BFILE associated with Lincoln_photo.

- "Example: Close a BFile with CLOSE Using PL/SQL (DBMS_LOB Package)" on page 5-190
- "Example: Close a BFile with CLOSE Using C (OCI)" on page 5-190
- "Example: Close a BFile with CLOSE Using COBOL (Pro*COBOL)" on page 5-192
- "Example: Close a BFile with CLOSE Using C++ (Pro*C/C++)" on page 5-193
- "Example: Close a BFile with CLOSE Using Visual Basic (OO4O)" on page 5-194
- "Example: Close a BFile with CLOSE Using Java (JDBC)" on page 5-195

Example: Close a BFile with CLOSE Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure closeBFILE_procTwo is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE closeBFILE_procTwo IS
   Lob_loc    BFILE := BFILENAME('PHOTO_DIR', 'Lincoln_photo');
BEGIN
   DBMS_LOB.OPEN(Lob_loc, DBMS_LOB.LOB_READONLY);
   /* ...Do some processing. */
   DBMS_LOB.CLOSE(Lob_loc);
END;
```

Example: Close a BFile with CLOSE Using C (OCI)

```
/* Select the lob/bfile from the Multimedia table */
void selectLob(svchp, stmthp, errhp, dfnhp, Lob_loc, selstmt)
OCISvcCtx     *svchp;
OCIStatement  *stmthp;
OCIError      *errhp;
OCIDefine     *dfnhp;
```

```
OCILobLocator *Lob_loc;
text          *selstmt;
{
     /* Prepare the SQL select statement */
     checkerr (errhp, OCIStmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

     /* Call define for the bfile column */
     checkerr (errhp, OCIDefineByPos(stmhp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT));

     /* Execute the SQL select statement */
     checkerr (errhp, OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                     (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                                     (ub4) OCI_DEFAULT));
}
void BfileClose(envhp, svchp, stmthp, errhp, dfnhp)
OCIEnv  *envhp;
OCISvcCtx *svchp;
OCIStatement *stmthp;
OCIError *errhp;
OCIDefine *dfnhp;
{
     /* Assume all handles passed as input to this routine have been
        allocated and initialized.
      */

     OCILobLocator *bfile_loc;

     /* Allocate the locator descriptor */
     (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                (ub4) OCI_DTYPE_FILE,
                (size_t) 0, (dvoid **) 0)

     /* Set the bfile locator information */
     checkerr(errhp, (OCILobFileSetName(envhp, errhp, &bfile_loc,
                          (OraText *)"PHOTO_DIR", (ub2)strlen("PHOTO_DIR"),
                          (OraText *)"Lincoln_photo",
                          (ub2)strlen("Lincoln_photo"))));

     checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));
```

```
    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

Example: Close a BFile with CLOSE Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-CLOSE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.
01  DIR-ALIAS      PIC X(30) VARYING.
01  FNAME          PIC X(20) VARYING.
01  ORASLNRD        PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-CLOSE.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate and initialize the BFILE locators:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.

* Set up the directory and file information:
    MOVE "PHOTO_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "lincoln_photo" TO FNAME-ARR.
    MOVE 13 TO FNAME-LEN.

    EXEC SQL
         LOB FILE SET :BFILE1
         DIRECTORY = :DIR-ALIAS, FILENAME = :FNAME
    END-EXEC.
```

```
            EXEC SQL
                 LOB OPEN :BFILE1 READ ONLY
            END-EXEC.

       * Close the LOB:
            EXEC SQL LOB CLOSE :BFILE1 END-EXEC.

       * And free the LOB locator:
            EXEC SQL FREE :BFILE1 END-EXEC.
            STOP RUN.

        SQL-ERROR.
            EXEC SQL
                WHENEVER SQLERROR CONTINUE
            END-EXEC.
            MOVE ORASLNR TO ORASLNRD.
            DISPLAY " ".
            DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
            DISPLAY " ".
            DISPLAY SQLERRMC.
            EXEC SQL
                ROLLBACK WORK RELEASE
            END-EXEC.
            STOP RUN.
```

Example: Close a BFile with CLOSE Using C++ (Pro*C/C++)

```
/* Pro*C/C++ has only one form of CLOSE for BFILEs.  Pro*C/C++ has no
   FILE CLOSE statement.  A simple CLOSE statement is used instead: */

#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void closeBFILE_proc()
{
```

```
  OCIBFileLocator *Lob_loc;
  char *Dir = "PHOTO_DIR", *Name = "Lincoln_photo";

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc;
  EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
  EXEC SQL LOB OPEN :Lob_loc READ ONLY;
  /* ... Do some processing */
  EXEC SQL LOB CLOSE :Lob_loc;
  EXEC SQL FREE :Lob_loc;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  closeBFILE_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Close a BFile with CLOSE Using Visual Basic (OO4O)

```
'Note that this code fragment assumes a ORABFILE object as the result of a
'dynaset operation. This object could have been an OUT parameter of a PL/SQL
'procedure. For more information please refer to chapter 1:
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraMusic As OraBfile, amount_read%, chunksize%, chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "scott/tiger", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Multimedia_tab", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("Music").Value

If OraMusic.IsOpen Then
   'Processing given that the file is already open
   OraMusic.Close
End If
```

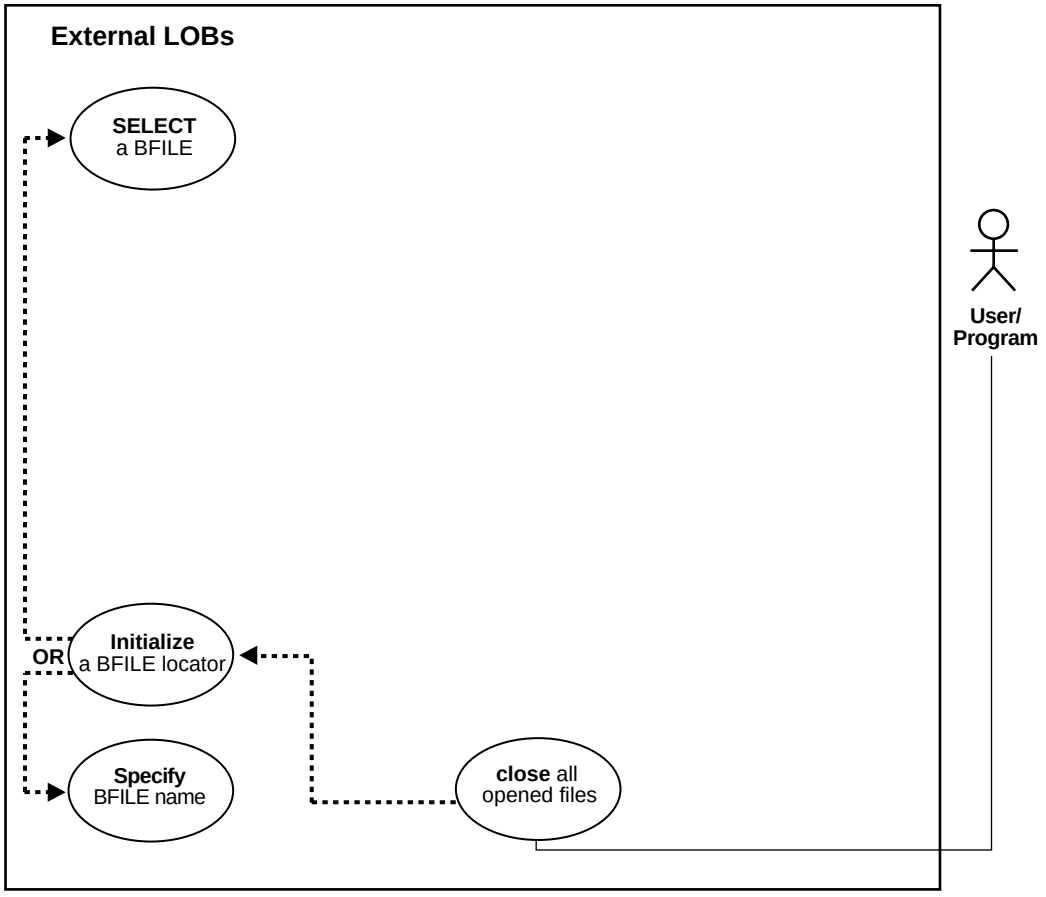
Example: Close a BFile with CLOSE Using Java (JDBC)

```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_48
{

  public static void main (String args [])
       throws Exception
  {
    // Load the Oracle JDBC driver:
    Class.forName ("oracle.jdbc.driver.OracleDriver");

    // Connect to the database:
    Connection conn =
      DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();

    try
    {
       BFILE src_lob = null;
       ResultSet rset = null;
       Boolean result = null;

       rset = stmt.executeQuery (
          "SELECT BFILENAME('PHOTO_DIR', 'Lincoln_photo') FROM DUAL");
       if (rset.next())
```

```
            {
               src_lob = ((OracleResultSet)rset).getBFILE (1);
            }

            result = new Boolean(src_lob.isFileOpen());
            System.out.println(
               "result of fileIsOpen() before opening file : " + result.toString());

            src_lob.openFile();

            result = new Boolean(src_lob.isFileOpen());
            System.out.println(
               "result of fileIsOpen() after opening file : " + result.toString());

            // Close the BFILE, statement and connection:
            src_lob.closeFile();
            stmt.close();
            conn.commit();
            conn.close();
         }
         catch (SQLException e)
         {
            e.printStackTrace();
         }
      }
   }
```

Close All Open BFILEs

Figure 5–36 Use Case Diagram: Close All Open BFILEs



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

It is the user's responsibility to close any opened file(s) after normal or abnormal

termination of a PL/SQL program block or OCI program. So, for instance, for every DBMS_LOB.FILEOPEN() or DBMS_LOB.OPEN() call on a BFILE, there must be a matching DBMS_LOB.FILECLOSE() or DBMS_LOB.CLOSE() call. You should close open files before the termination of a PL/SQL block or OCI program, and also in situations which have raised errors. The exception handler should make provisions to close any files that were opened before the occurrence of the exception or abnormal termination.

If this is not done, Oracle will consider these files unclosed.

See Also: "Maximum Number of Open BFILEs" on page 5-52

Scenario

Example: Close All Open BFiles Using PL/SQL (DBMS_LOB Package)

```
/* Note that the example procedure closeAllOpenFilesBFILE_proc is not part of
   the DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE closeAllOpenFilesBFILE_proc IS
BEGIN
   /* Close all open BFILEs: */
   DBMS_LOB.FILECLOSEALL;
END;
```

Example: Close All Open BFiles Using C (OCI)

```
void BfileCloseAll(svchp, errhp)
OCISvcCtx *svchp;
OCIError *errhp;
{
  /* Close all open files on the service context */
```

```
  checkerr(errhp, OCILobFileCloseAll(svchp, errhp));
}
```

Example: Close All Open BFiles Using COBOL (Pro*COBOL)

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-CLOSE-ALL.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "USER1/USER1".
01  BFILE1         SQL-BFILE.
01  BFILE2         SQL-BFILE.
01  DIR-ALIAS1     PIC X(30) VARYING.
01  FNAME1         PIC X(20) VARYING.
01  DIR-ALIAS2     PIC X(30) VARYING.
01  FNAME2         PIC X(20) VARYING.
01  ORASLNRD       PIC 9(4).

    EXEC SQL INCLUDE SQLCA END-EXEC.
    EXEC ORACLE OPTION (ORACA=YES) END-EXEC.
    EXEC SQL INCLUDE ORACA END-EXEC.

PROCEDURE DIVISION.
BFILE-CLOSE-ALL.

    EXEC SQL WHENEVER SQLERROR DO PERFORM SQL-ERROR END-EXEC.
    EXEC SQL
         CONNECT :USERID
    END-EXEC.

* Allocate the BFILEs:
    EXEC SQL ALLOCATE :BFILE1 END-EXEC.
    EXEC SQL ALLOCATE :BFILE2 END-EXEC.

* Set up the directory and file information:
    MOVE "AUDIO_DIR" TO DIR-ALIAS1-ARR.
    MOVE 9 TO DIR-ALIAS1-LEN.
    MOVE "washington_audio" TO FNAME1-ARR.
    MOVE 16 TO FNAME1-LEN.

    EXEC SQL
         LOB FILE SET :BFILE1
```

```
           DIRECTORY = :DIR-ALIAS1, FILENAME = :FNAME1
      END-EXEC.

      EXEC SQL
           LOB OPEN :BFILE1 READ ONLY
      END-EXEC.

* Set up the directory and file information:
      MOVE "PHOTO_DIR" TO DIR-ALIAS2-ARR.
      MOVE 9 TO DIR-ALIAS2-LEN.
      MOVE "lincoln_photo" TO FNAME2-ARR.
      MOVE 13 TO FNAME2-LEN.

      EXEC SQL
           LOB FILE SET :BFILE2
           DIRECTORY = :DIR-ALIAS2, FILENAME = :FNAME2
      END-EXEC.

      EXEC SQL
           LOB OPEN :BFILE2 READ ONLY
      END-EXEC.

* Close both BFILE1 and BFILE2:
      EXEC SQL LOB FILE CLOSE ALL END-EXEC.
      STOP RUN.

 SQL-ERROR.
      EXEC SQL
          WHENEVER SQLERROR CONTINUE
      END-EXEC.
      MOVE ORASLNR TO ORASLNRD.
      DISPLAY " ".
      DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNRD, ":".
      DISPLAY " ".
      DISPLAY SQLERRMC.
      EXEC SQL
          ROLLBACK WORK RELEASE
      END-EXEC.
      STOP RUN.
```

Example: Close All Open BFiles Using C++ (Pro*C/C++)

```
#include <oci.h>
#include <stdio.h>
```

```
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLERROR CONTINUE;
  printf("%.*s\n", sqlca.sqlerrm.sqlerrml, sqlca.sqlerrm.sqlerrmc);
  EXEC SQL ROLLBACK WORK RELEASE;
  exit(1);
}

void closeAllOpenBFILEs_proc()
{
  OCIBFileLocator *Lob_loc1, *Lob_loc2;

  EXEC SQL WHENEVER SQLERROR DO Sample_Error();
  EXEC SQL ALLOCATE :Lob_loc1;
  EXEC SQL ALLOCATE :Lob_loc2;
  /* Populate the Locators: */
  EXEC SQL SELECT Music INTO :Lob_loc1
           FROM Multimedia_tab WHERE Clip_ID = 3;
  EXEC SQL SELECT Mtab.Voiced_ref.Recording INTO Lob_loc2
           FROM Multimedia_tab Mtab WHERE Mtab.Clip_ID = 3;
  /* Open both BFILEs: */
  EXEC SQL LOB OPEN :Lob_loc1 READ ONLY;
  EXEC SQL LOB OPEN :Lob_loc2 READ ONLY;
  /* Close all open BFILEs: */
  EXEC SQL LOB FILE CLOSE ALL;
  /* Free resources held by the Locators: */
  EXEC SQL FREE :Lob_loc1;
  EXEC SQL FREE :Lob_loc2;
}

void main()
{
  char *samp = "samp/samp";
  EXEC SQL CONNECT :samp;
  closeAllOpenBFILEs_proc();
  EXEC SQL ROLLBACK WORK RELEASE;
}
```

Example: Close All Open BFiles Using Visual Basic (OO4O)

```
Dim OraParameters as OraParameters, OraPhoto as OraBFile
OraConnection.BeginTrans
```

```
Set OraParameters = OraDatabase.Parameters

'Define in out parameter of BFILE type:
OraParameters.Add "MyPhoto", Empty,ORAPARAM_BOTH,ORATYPE_BFILE

'Select the photo BFile for clip_id 1:
OraDatabase.ExecuteSQL("Begin SELECT Photo INTO :MyPhoto FROM
Multimedia_tab WHERE Clip_ID = 1; END " )

'Get the BFile photo column:
set OraPhoto = OraParameters("MyPhoto").Value

'Open the OraPhoto:
OraPhoto.Open

'Do some processing on OraPhoto

'Close all the BFILEs associated with OraPhoto:
OraPhoto.CloseAll
```

Example: Close All Open BFiles Using Java (JDBC)

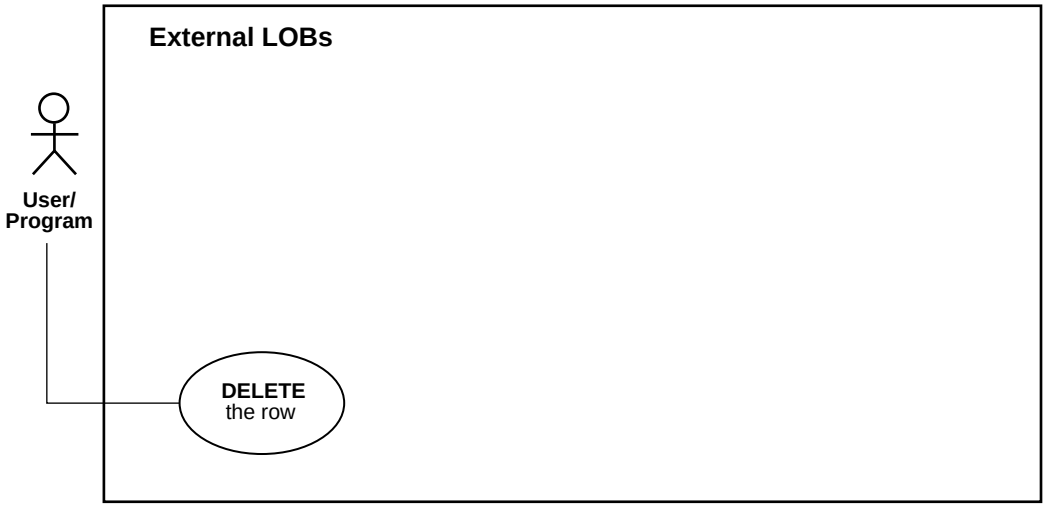
```
// Java IO classes:
import java.io.InputStream;
import java.io.OutputStream;

// Core JDBC classes:
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

// Oracle Specific JDBC classes:
import oracle.sql.*;
import oracle.jdbc.driver.*;

public class Ex4_66
{

  static final int MAXBUFSIZE = 32767;
```

```
public static void main (String args [])
     throws Exception
{
  // Load the Oracle JDBC driver:
  Class.forName ("oracle.jdbc.driver.OracleDriver");

  // Connect to the database:
  Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "samp", "samp");

  // It's faster when auto commit is off:
  conn.setAutoCommit (false);

  // Create a Statement:
  Statement stmt = conn.createStatement ();

  try
  {
     BFILE lob_loc1 = null;
     BFILE lob_loc2 = null;
     ResultSet rset = null;
     OracleCallableStatement cstmt = null;

     rset = stmt.executeQuery (
        "SELECT photo FROM multimedia_tab WHERE clip_id = 3");
     if (rset.next())
     {
        lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
     }

     rset = stmt.executeQuery (
        "SELECT BFILENAME('PHOTO_DIR', 'RooseveltFDR_photo') FROM DUAL");
     if (rset.next())
     {
        lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
     }

     cstmt = (OracleCallableStatement) conn.prepareCall (
        "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
     // Open the first LOB:
     cstmt.setBFILE(1, lob_loc1);
     cstmt.execute();

     cstmt = (OracleCallableStatement) conn.prepareCall (
        "BEGIN DBMS_LOB.FILEOPEN(?,DBMS_LOB.LOB_READONLY); END;");
```

```
            // Use the same CallableStatement to open the second LOB:
            cstmt.setBFILE(1, lob_loc2);
            cstmt.execute();

            // Compare MAXBUFSIZE bytes starting at the first byte of
            // both lob_loc1 and lob_loc2:
            cstmt = (OracleCallableStatement) conn.prepareCall (
               "BEGIN ? := DBMS_LOB.COMPARE(?, ?, ?, 1, 1); END;");
            cstmt.registerOutParameter (1, Types.NUMERIC);
            cstmt.setBFILE(2, lob_loc1);
            cstmt.setBFILE(3, lob_loc2);
            cstmt.setInt(4, MAXBUFSIZE);
            cstmt.execute();

            int result = cstmt.getInt(1);

            System.out.println("Comparison result: " + Integer.toString(result));

            // Close all BFILEs:
            stmt.execute("BEGIN DBMS_LOB.FILECLOSEALL; END;");

            stmt.close();
            cstmt.close();
            conn.commit();
            conn.close();
         }
         catch (SQLException e)
         {
            e.printStackTrace();
         }
      }
   }
```

DELETE the Row of a Table Containing a BFILE

Figure 5–37 Use Case Diagram: DELETE the Row of a Table Containing a LOB (BFILE)



To refer to the table of all basic operations having to do with External LOBs (BFILES) see:

- "Use Case Model: External LOBs" on page 5-2

Scenario

Unlike internal persistent LOBs, the LOB value in a BFILE does not get deleted by using SQL DDL or SQL DML commands — only the BFILE locator is deleted. Deletion of a record containing a BFILE column amounts to de-linking that record from an existing file, *not* deleting the physical operating system file itself. An SQL DELETE statement on a particular row deletes the BFILE locator for the particular row, thereby removing the reference to the operating system file.

The following DELETE, DROP TABLE, or TRUNCATE TABLE statements delete the row, and hence the BFILE locator that refers to Image1.gif, but leave the operating system file undeleted in the filesystem.

Example: Delete a Row from a Table Using SQL

```
DELETE FROM Multimedia_tab
   WHERE Clip_ID = 3;

DROP TABLE Multimedia_tab;

TRUNCATE TABLE Multimedia_tab;
```

6

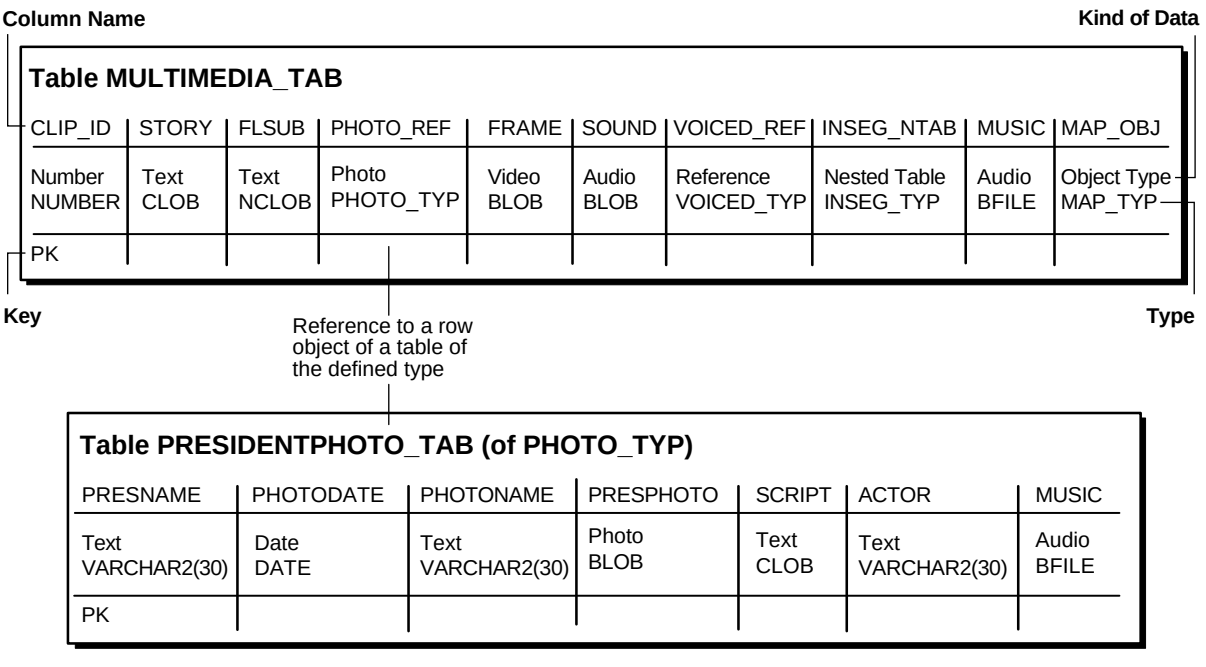
LOBs and Partitioned Tables

Using LOBs in Partitions

You can partition tables with LOBs. As a result, LOBs can take advantage of all of the benefits of partitioning. For example, LOB segments can be spread between several tablespaces to balance I/O load and to make backup and recovery more manageable. LOBs in a partitioned table also become easier to maintain. This section describes some of the ways you can manipulate LOBs in partitioned tables.

As an extension to the example multimedia application described in Chapter 1, "Introduction to Working With LOBs", let us suppose that makers of a documentary are producing multiple clips relating to different Presidents of the United States. The clips consist of photographs of the presidents accompanied by spoken text and background music. The photographs will come from the PhotoLib_Tab archive. To make the most efficient use of the presidents' photographs, they are loaded into a database according to the schema illustrated in Figure 6–1.

Figure 6–1 Schema Design for Inclusion of PHOTO_REF Reference



PRESNAME: A column on the president's name lets the documentary producers select data for clips organized around specific presidents. PRESNAME is also chosen as a primary key because it holds unique values.

PRESPHOTO: This column contains photographs in which a president appears. This category also contains photographs of paintings and engravings of presidents who lived before the advent of photography.

PHOTODATE: This column contains the date on which the photograph was taken. In the case of presidents who lived before the advent of photography, PHOTODATE pertains to the date when the painting or engraving was created. This column is chosen as the partition key to make it easier to add partitions and to perform MERGEs and SPLITs of the data based on some given date such as the end of a president's first term. This will be illustrated later in this section.

PHOTONAME: This column contains the name of the photograph. An example name might be something as precise as "Bush Addresses UN - June 1990" or as general as "Franklin Roosevelt - Inauguration".

SCRIPT: This column contains written text associated with the photograph. This could be text describing the event portrayed by the photograph or perhaps segments of a speech by the president.

ACTOR: This column contains the name of the actor reading the script.

MUSIC: This column contains background music to be played during the viewing of the photographs.

Creating and Partitioning a Table Containing LOB Data

To isolate the photographs associated with a given president, a partition is created for each president by the ending dates of their terms of office. For example, a president who served two terms would have two partitions: the first partition bounded by the end date of the first term and a second partition bounded by the end date of the second term.

Note that in the following examples, the extension 1 refers to a president's first term and 2 refers to a president's second term. For example, GeorgeWashington1_part refers to the partition created for George Washington's first term and RichardNixon2_part refers to the partition created for Richard Nixon's second term.

Note: You may need to set up data structures for certain examples to work; such as:

```
CONNECT system/manager
GRANT CREATE TABLESPACE, DROP TABLESPACE TO scott;
CONNECT scott/tiger
CREATE TABLESPACE EarlyPresidents_tbs DATAFILE
'disk1:moredata01' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsPhotos_tbs DATAFILE
'disk1:moredata99' SIZE 1M;
CREATE TABLESPACE EarlyPresidentsScripts_tbs DATAFILE
'disk1:moredata03' SIZE 1M;
CREATE TABLESPACE RichardNixon1_tbs DATAFILE
'disk1:moredata04' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsPhotos_tbs DATAFILE
'disk1:moredata05' SIZE 1M;
CREATE TABLESPACE Post1960PresidentsScripts_tbs DATAFILE
'disk1:moredata06' SIZE 1M;
CREATE TABLESPACE RichardNixon2_tbs DATAFILE
'disk1:moredata07' SIZE 1M;
CREATE TABLESPACE GeraldFord1_tbs DATAFILE
'disk1:moredata97' SIZE 1M;
CREATE TABLESPACE RichardNixonPhotos_tbs DATAFILE
'disk1:moredata08' SIZE 2M;
CREATE TABLESPACE RichardNixonBigger2_tbs DATAFILE
'disk1:moredata48' SIZE 2M;
CREATE TABLE Mirrorlob_tab(
   PresName VARCHAR2(30),
   PhotoDate DATE,
   PhotoName VARCHAR2(30),
   PresPhoto BLOB,
   Script CLOB,
   Actor VARCHAR2(30),
   Music BFILE);
```

```
CREATE TABLE Presidentphoto_tab(PresName VARCHAR2(30), PhotoDate DATE,
                                PhotoName VARCHAR2(30), PresPhoto BLOB,
                                Script CLOB, Actor VARCHAR2(30), Music BFILE)
   STORAGE (INITIAL 100K  NEXT 100K PCTINCREASE 0)
   LOB (PresPhoto) STORE AS (CHUNK 4096)
   LOB (Script) STORE AS (CHUNK 2048)
   PARTITION BY RANGE(PhotoDate)
(PARTITION GeorgeWashington1_part
```

```
   /* Use photos to the end of Washington's first term */
   VALUES LESS THAN (TO_DATE('19-mar-1792', 'DD-MON-YYYY'))
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION GeorgeWashington2_part
   /* Use photos to the end of Washington's second term */
   VALUES LESS THAN (TO_DATE('19-mar-1796', 'DD-MON-YYYY'))
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
PARTITION JohnAdams1_part
   /* Use photos to the end of Adams' only term */
   VALUES LESS THAN (TO_DATE('19-mar-1800', 'DD-MON-YYYY'))
   TABLESPACE EarlyPresidents_tbs
   LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs),
/* ...intervening presidents... */
PARTITION RichardNixon1_part
   /* Use photos to the end of Nixon's first term */
   VALUES LESS THAN (TO_DATE('20-jan-1972', 'DD-MON-YYYY'))
   TABLESPACE RichardNixon1_tbs
   LOB (PresPhoto) store as (TABLESPACE Post1960PresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE Post1960PresidentsScripts_tbs)
);
```

Creating an Index on a Table Containing LOB Columns

To improve the performance of queries which access records by a President's name and possibly the names of photographs, a UNIQUE local index is created:

```
CREATE UNIQUE INDEX PresPhoto_idx
   ON PresidentPhoto_tab (PresName, PhotoName, Photodate) LOCAL;
```

Exchanging Partitions Containing LOB Data

As a part of upgrading from Oracle8.0 to 8.1, data was exchanged from an existing non-partitioned table containing photos of Bill Clinton's first term into the appropriate partition:

```
ALTER TABLE PresidentPhoto_tab EXCHANGE PARTITION RichardNixon1_part
   WITH TABLE Mirrorlob_tab INCLUDING INDEXES;
```

Adding Partitions to Tables Containing LOB Data

To account for Richard Nixon's second term, a new partition was added to `PresidentPhoto_tab`:

```
ALTER TABLE PresidentPhoto_tab ADD PARTITION RichardNixon2_part
   VALUES LESS THAN (TO_DATE('20-jan-1976', 'DD-MON-YYYY'))
   TABLESPACE RichardNixon2_tbs
   LOB (PresPhoto) store as (TABLESPACE Post1960PresidentsPhotos_tbs)
   LOB (Script) store as (TABLESPACE Post1960PresidentsScripts_tbs);
```

Moving Partitions Containing LOBs

During his second term, Richard Nixon had so many photo-ops, that the partition containing information on his second term is no longer adequate. It was decided to move the data partition and the corresponding `LOB` partition of `PresidentPhoto_tab` into a different tablespace, with the corresponding `LOB` partition of `Script` remaining in the original tablespace:

```
ALTER TABLE PresidentPhoto_tab MOVE PARTITION RichardNixon2_part
   TABLESPACE RichardNixonBigger2_tbs
   LOB (PresPhoto) STORE AS (TABLESPACE RichardNixonPhotos_tbs);
```

Splitting Partitions Containing LOBs

When Richard Nixon was re-elected for his second term, a partition with bounds equal to the expected end of his term (20-jan-1976) was added to the table (see above example.) Since Nixon resigned from office on 9 August 1974, that partition had to be split to reflect the fact that the remainder of the term was served by Gerald Ford:

```
ALTER TABLE PresidentPhoto_tab SPLIT PARTITION RichardNixon2_part
   AT (TO_DATE('09-aug-1974'), 'DD-MON-YYYY'))
   INTO (PARTITION RichardNixon2_part),
      PARTITION GeraldFord1_part TABLESPACE GeraldFord1_tbs
         LOB (PresPhoto) STORE AS (TABLESPACE Post1960PresidentsPhotos_tbs)
         LOB (Script) STORE AS (TABLESPACE Post1960PresidentsScripts_tbs)));
```

Merging Partitions Containing LOBs

Despite the best efforts of the documentary producers in searching for photographs of paintings or engravings of George Washington, the number of photographs that were found was inadequate to justify a separate partition for each of his two terms.

Accordingly, it was decided to merge these two partition into one named GeorgeWashington8Years_part:

```
ALTER TABLE PresidentPhoto_tab
   MERGE PARTITIONS GeorgeWashington1_part, GeorgeWashington2_part
   INTO PARTITION GeorgeWashington8Years_part TABLESPACE EarlyPresidents_tbs
      LOB (PresPhoto) store as (TABLESPACE EarlyPresidentsPhotos_tbs)
      LOB (Script) store as (TABLESPACE EarlyPresidentsScripts_tbs);
```

Populating the Script CLOB and Photo BLOB

The documentary producers have found a photograph Bill Clinton during his trip to Florida on 22 March 1993. They will add it to the PresidentPhoto_tab table, and then fill the PresPhoto column with the photograph BLOB data and the Script column with the text CLOB data. This section illustrates populating the Script CLOB and the Photo BLOB.

Assume that the following directory objects for the music audio files and the presidential photographs were already created,

```
CREATE DIRECTORY Music_dir as '/audio/presidents';
CREATE DIRECTORY Image_dir as '/image/presidents';
```

and that READ permission has been granted to the user who will use it:

```
GRANT READ ON DIRECTORY Music_dir TO a_user;
GRANT READ ON DIRECTORY Image_dir TO a_user;
```

```
INSERT INTO PresidentPhoto_tab VALUES (
   'RichardNixon', TO_DATE('22-mar-1973', 'DD-MON-YYYY'), 'NixonFlorida1993',
   EMPTY_BLOB(), EMPTY_CLOB(), 'Warren Beatty', BFILENAME('MUSIC_DIR',
   'TropicalMusic'));
```

Populating the BLOB:

The following code segment uses the LOADFROMFILE command to populate the PresPhoto BLOB with data:

```
CREATE OR REPLACE PROCEDURE loadPartLOBFromBFILE_proc IS
   Dest_loc   BLOB;
   Src_loc    BFILE := BFILENAME('IMAGE_DIR', 'FloridaTrip');
   Amount     INTEGER := 4000;
BEGIN
```

```
   /* Select the LOB from the partitioned table: */
   SELECT PresPhoto INTO Dest_loc FROM PresidentPhoto_tab WHERE
      PresName  = 'RichardNixon' AND
      PhotoName = 'NixonFlorida1993'
      FOR UPDATE;

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Dest_loc, DBMS_LOB.LOB_READWRITE);
   /* Opening the BFILE is mandatory */
   DBMS_LOB.OPEN (Src_loc, DBMS_LOB.LOB_READONLY);

   DBMS_LOB.LOADFROMFILE(Dest_loc, Src_loc, Amount);

   /* Closing the LOB is mandatory if you have opened it */
   DBMS_LOB.CLOSE(Dest_loc);
   DBMS_LOB.CLOSE(Src_loc);

    COMMIT;
END;
```

Populate the CLOB:

The following code segment uses the `CHECKIN` method to load data into the `Script CLOB`:

```
CREATE OR REPLACE PROCEDURE checkinPartLOB_proc IS
   Lob_loc       CLOB;
   Buffer        VARCHAR2(32767);
   Amount        BINARY_INTEGER := 32767;
   Position      INTEGER := 1;
   i             INTEGER;
BEGIN
   /* Select the LOB from the partitioned table: */
   SELECT script INTO Lob_loc FROM PresidentPhoto_tab where
      PresName = 'RichardNixon' AND
      PhotoName = 'NixonFlorida1993';

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READWRITE);

   /* Fill the buffer with data */

   FOR i IN 1..3 LOOP
      /* Write data: */
      DBMS_LOB.WRITE (Lob_loc, Amount, Position, Buffer);
```

```
      /* Fill in more data: */
      Position := Position + Amount;
   END LOOP;

   /* Closing the LOB is mandatory if you have opened it */
   DBMS_LOB.CLOSE(Lob_loc);
   COMMIT;

EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Operation failed');
END;
```

Reading the LOB Value:

The following code segment uses the CHECKOUT command to READ the LOB value:

```
CREATE OR REPLACE PROCEDURE checkoutPartLOB_proc is
   Lob_loc       CLOB;
   Buffer        VARCHAR2(32767);
   Amount        BINARY_INTEGER := 32767;
   Position      INTEGER := 1;
BEGIN
   /* Select the LOB from the partitioned table: */
   SELECT Script INTO Lob_loc FROM PresidentPhoto_tab WHERE
      PresName = 'RichardNixon' AND
      PhotoName = 'NixonFlorida1993';

   /* Opening the LOB is optional: */
   DBMS_LOB.OPEN (Lob_loc, DBMS_LOB.LOB_READONLY);

   LOOP
      /* Read data: */
      DBMS_LOB.READ (Lob_loc, Amount, Position, Buffer);
      /* Process the data in the buffer. */
      Position := Position + Amount;
   END LOOP;

   /* Closing the LOB is mandatory if you have opened it */
   DBMS_LOB.CLOSE(Lob_loc);

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

Index

M

N

O

P

R

S

T

U

V