

**Oracle9i**

Application Developer's Guide - Large Objects (LOBs)

Release 2 (9.2)

March 2002

Part No. A96591-01

**ORACLE®**

Part No. A96591-01

Copyright © 1996, 2002 Oracle Corporation. All rights reserved.

Primary Authors: Eric Paapanen, Shelley Higgins, Susan Kotsovolos, Den Raphaely

Contributing Authors: K. Akiyama, Geeta Arora, S. Banerjee, Yujie Cao, T. H. Chang, E. Chong, S. Das, C. Freiwald, C. Iyer, M. Jagannath, R. Krishnan, M. Krishnaprasad, S. Lari, Li-Sen Liu, D. Mullen, V. Nimani, A. Roy, S. Shah, A. Shivarudraiah, J. Srinivasan, R. Toohey, Anh-Tuan Tran, G. Viswana, A. Yalamanchi

Contributors: J. Balaji, D. Cruceanu, M. Chien, G. Edmiston, M. Fry, J. Kalogeropoulos, V. Karra, P. Manavazhi, S. Muthulingam, R. Ratnam, C. Shay, A. Shehade, E. Shirk, Jan Syssauw, S. Vedala, E. Wan, J. Yang

Graphics: Valerie Moore, Charles Keller

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

**Restricted Rights Notice** Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Gist, Oracle Store, Oracle7, Oracle8, Oracle8i, Oracle9i, PL/SQL, Pro\*C, Pro\*C/C++, Pro\*COBOL, and SQL\*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

---

---

# Contents

|   |               |
|---|---------------|
| <b>Send Us Your Comments .....</b>                            | <b>xxxii</b>  |
| <b>Preface.....</b>   | <b>xxxiii</b> |
| Audience .....  | xxxiv         |
| Organization.....   | xxxiv         |
| Related Documents.....  | xxxvii        |
| Conventions.....  | xxxix         |
| Documentation Accessibility .....                             | xlii          |
| <b>What's New with Large Objects (LOBs)? .....</b>            | <b>xlv</b>    |
| LOB Features Introduced with Oracle9i, Release 2 (9.2).....   | xlv           |
| LOB Features Introduced with Oracle9i, Release 1 (9.0.1)..... | xlvii         |
| LOB Features Introduced with Oracle8i Release 2 (8.1.6).....  | xlix          |
| LOB Features Introduced with Oracle8i, Release 8.1.5 .....    | l             |
| <b>1 Introduction to LOBs</b>                                 |               |
| <b>Why Use LOBs?.....</b>                                     | <b>1-2</b>    |
| Unstructured Data.....  | 1-2           |
| LOB Datatype Helps Support Internet Applications .....        | 1-3           |
| Using XML, LOBs, and Oracle Text (interMedia Text) .....      | 1-3           |
| <b>Why Not Use LONGs?.....</b>                                | <b>1-4</b>    |
| <b>LONG-to-LOB Migration API.....</b>                         | <b>1-5</b>    |
| <b>SQL Semantics Support for LOBs.....</b>                    | <b>1-5</b>    |
| <b>Partitioned Index-Organized Tables and LOBs.....</b>       | <b>1-6</b>    |

|  |     |
|--|-----|
| <b>Extensible Indexing on LOBs</b> .....                             | 1-6 |
| <b>Function-Based Indexing on LOBs</b> .....                         | 1-7 |
| <b>XML Documents Can be Stored in XMLType Columns as CLOBs</b> ..... | 1-8 |
| <b>Compatibility and Migration Issues</b> .....                      | 1-8 |
| <b>Examples in This Guide</b> .....                                  | 1-9 |

## 2 Basic LOB Components

|   |      |
|---|------|
| <b>The LOB Datatype</b> .....   | 2-2  |
| Internal LOBs.....  | 2-2  |
| External LOBs (BFILES).....   | 2-2  |
| Internal LOBs Use Copy Semantics, External LOBs Use Reference Semantics ..... | 2-3  |
| <b>Varying-Width Character Data</b> .....                                     | 2-4  |
| Using DBMS_LOB.LOADFROMFILE and Functions that Access OCI.....                | 2-4  |
| <b>LOB Value and Locators</b> .....   | 2-5  |
| Inline storage of the LOB value .....   | 2-5  |
| LOB Locators .....  | 2-5  |
| Setting the LOB Column/Attribute to Contain a Locator .....                   | 2-6  |
| Accessing a LOB Through a Locator.....  | 2-7  |
| <b>Creating Tables that Contain LOBs</b> .....                                | 2-8  |
| Initializing Internal LOBs to NULL or Empty.....                              | 2-8  |
| Initializing LOBs Example Using Table Multimedia_tab .....                    | 2-10 |
| Initializing Internal LOB Columns to a Value .....                            | 2-10 |
| Initializing External LOBs to NULL or a File Name.....                        | 2-10 |

## 3 LOB Support in Different Programmatic Environments

|   |     |
|---|-----|
| <b>Eight Programmatic Environments Operate on LOBs</b> .....                          | 3-2 |
| <b>Comparing the LOB Interfaces</b> .....   | 3-3 |
| <b>Using PL/SQL (DBMS_LOB Package) To Work With LOBs</b> .....                        | 3-6 |
| Provide a LOB Locator Before Invoking the DBMS_LOB Routine .....                      | 3-6 |
| PL/SQL - LOB Guidelines.....  | 3-7 |
| PL/SQL Functions and Procedures that Operate on LOBs.....                             | 3-8 |
| PL/SQL Functions/Procedures To Modify BLOB, CLOB, and NCLOB Values .....              | 3-8 |
| PL/SQL Functions/Procedures To Read or Examine Internal and External LOB Values ..... | 3-9 |
| PL/SQL Functions/Procedures To Operate on Temporary LOBs.....                         | 3-9 |

|  |      |
|--|------|
| PL/SQL Read-Only Functions/Procedures for BFILES .....                                     | 3-10 |
| PL/SQL Functions/Procedures To Open and Close Internal and External LOBs .....             | 3-10 |
| <b>Using C (OCI) To Work With LOBs</b> .....   | 3-10 |
| Set CSID Parameter To OCI_UCS2ID to Read/Write in UCS2 .....                               | 3-11 |
| Offset and Amount Parameters: Fixed-Width Versus Varying-Width,<br>Character or Byte ..... | 3-11 |
| OCILobLoadFromFile: Specify Amount Parameter to be Less than Length of BFILE.....          | 3-13 |
| OCILobRead: Specify Amount Parameter to be 4 gigabytes - 1 .....                           | 3-13 |
| OCI LOB Examples.....  | 3-13 |
| Further Information About OCI.....   | 3-13 |
| OCI Functions that Operate on BLOBs, BLOBs, NCLOBs, and BFILES .....                       | 3-14 |
| OCI Functions To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values.....                   | 3-14 |
| OCI Functions To Read or Examine Internal LOB and External LOB (BFILE) Values ....         | 3-15 |
| OCI Functions For Temporary LOBs.....  | 3-15 |
| OCI Read-Only Functions For BFILES.....  | 3-15 |
| OCI LOB Locator Functions.....   | 3-16 |
| OCI LOB-Buffering Functions .....  | 3-16 |
| OCI Functions To Open and Close Internal and External LOBs .....                           | 3-16 |
| OCI Example — Is the LOB Open: main() and seeIfLOBIsOpen .....                             | 3-17 |
| <b>Using C++ (OCCI) To Work With LOBs</b> .....  | 3-22 |
| Distinct Classes for Each LOB Type.....  | 3-23 |
| Offset and Amount Parameters: Fixed-Width Versus Varying-Width,<br>Character or Byte ..... | 3-24 |
| Loading from Files with OCCIClob.copy() and OCCIBlob.copy(): Amount Parameter .            | 3-25 |
| OCCIClob.read(), OCCIBlob.read(), and OCCIBfile.read():Amount Parameter .....              | 3-25 |
| Further Information About OCCI .....   | 3-26 |
| OCCI Methods that Operate on BLOBs, BLOBs, NCLOBs, and BFILES .....                        | 3-26 |
| OCCI Methods To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values .....                   | 3-26 |
| OCCI Methods To Read or Examine Internal LOB and BFILE Values.....                         | 3-27 |
| OCCI Read-Only Methods For BFILES .....  | 3-27 |
| Other OCCI LOB Methods .....   | 3-27 |
| OCCI Methods To Open and Close Internal and External LOBs .....                            | 3-28 |
| <b>Using C/C++ (Pro*C) To Work with LOBs</b> .....   | 3-29 |
| First Provide an Allocated Input Locator Pointer that Represents LOB .....                 | 3-29 |
| Pro*C/C++ Statements that Operate on BLOBs, CLOBs, NCLOBs, and BFILES .....                | 3-29 |
| Pro*C/C++ Embedded SQL Statements To Modify Internal LOB Values .....                      | 3-30 |

|   |             |
|---|-------------|
| Pro*C/C++ Embedded SQL Statements To Read or Examine Internal and External LOB Values ..... | 3-30        |
| Pro*C/C++ Embedded SQL Statements For Temporary LOBs.....                                   | 3-31        |
| Pro*C/C++ Embedded SQL Statements For BFILES.....   | 3-31        |
| Pro*C/C++ Embedded SQL Statements For LOB Locators .....                                    | 3-31        |
| Pro*C/C++ Embedded SQL Statements For LOB Buffering .....                                   | 3-32        |
| Pro*C/C++ Embedded SQL Statements To Open and Close Internal and External LOBs .....        | 3-32        |
| <b>Using COBOL (Pro*COBOL) to Work with LOBs .....</b>                                      | <b>3-33</b> |
| First Provide an Allocated Input Locator Pointer that Represents LOB .....                  | 3-33        |
| Pro*COBOL Statements that Operate on BLOBs, CLOBs, NCLOBs, and BFILES .....                 | 3-33        |
| Pro*COBOL Embedded SQL Statements To Modify Internal LOB Values .....                       | 3-34        |
| Pro*COBOL Embedded SQL Statements To Read or Examine Internal and External LOB Values.....  | 3-35        |
| Pro*COBOL Embedded SQL Statements For Temporary LOBs.....                                   | 3-35        |
| Pro*COBOL Embedded SQL Statements For BFILES.....   | 3-35        |
| Pro*COBOL Embedded SQL Statements For LOB Locators .....                                    | 3-35        |
| Pro*COBOL Embedded SQL Statements For LOB Buffering .....                                   | 3-36        |
| Pro*COBOL Embedded SQL Statements To Open and Close Internal LOBs and BFILES.....           | 3-36        |
| <b>Using Visual Basic (Oracle Objects for OLE (OO4O)) to Work with LOBs .....</b>           | <b>3-37</b> |
| OO4O Syntax Reference .....   | 3-37        |
| OraBlob, OraClob, and OraBfile Object Interfaces Encapsulate Locators.....                  | 3-38        |
| Example of OraBlob and OraBfile .....   | 3-38        |
| OO4O Methods and Properties to Access Data Stored in LOBs.....                              | 3-39        |
| OO4O Methods To Modify BLOB, CLOB, and NCLOB Values .....                                   | 3-41        |
| OO4O Methods To Read or Examine Internal and External LOB Values.....                       | 3-42        |
| OO4O Methods To Open and Close External LOBs (BFILES) .....                                 | 3-42        |
| OO4O Methods For Internal LOB-Buffering .....   | 3-43        |
| OO4O Properties For Operating on LOBs .....   | 3-43        |
| OO4O Read-Only Methods For External Lobs (BFILES) .....                                     | 3-43        |
| OO4O Properties For Operating on External LOBs (BFILES) .....                               | 3-44        |
| <b>Using Java (JDBC) To Work with LOBs.....</b>   | <b>3-45</b> |
| Changing Internal Persistent LOBs Using Java.....   | 3-45        |
| Reading Internal Persistent LOBs and External LOBs (BFILES) with Java .....                 | 3-45        |
| Calling DBMS_LOB Package From Java (JDBC).....  | 3-46        |

|   |             |
|---|-------------|
| Referencing LOBs Using Java (JDBC) .....  | 3-46        |
| JDBC Syntax References and Further Information .....                                      | 3-46        |
| JDBC Methods for Operating on LOBs .....  | 3-46        |
| JDBC oracle.sql.BLOB Methods To Modify BLOB Values .....                                  | 3-48        |
| JDBC oracle.sql.BLOB Methods To Read or Examine BLOB Values .....                         | 3-48        |
| JDBC oracle.sql.BLOB Methods and Properties for BLOB-Buffering .....                      | 3-49        |
| JDBC oracle.sql.CLOB Methods To Modify CLOB Values .....                                  | 3-49        |
| JDBC oracle.sql.CLOB Methods To Read or Examine CLOB Value .....                          | 3-49        |
| JDBC oracle.sql.CLOB Methods and Properties for CLOB-Buffering .....                      | 3-50        |
| JDBC oracle.sql.BFILE Methods To Read or Examine External LOB (BFILE) Values.....         | 3-50        |
| JDBC oracle.sql.BFILE Methods and Properties for BFILE-Buffering .....                    | 3-51        |
| JDBC: OracleBlob and OracleClob Do Not Work in Oracle8i 8.1.x and<br>Higher Releases..... | 3-51        |
| JDBC Temporary LOB APIs.....  | 3-51        |
| JDBC: Opening and Closing LOBs.....   | 3-52        |
| JDBC: Opening and Closing BLOBs .....   | 3-53        |
| JDBC: Opening and Closing CLOBs.....  | 3-54        |
| JDBC: Opening and Closing BFILES.....   | 3-56        |
| Trimming LOBs using JDBC.....   | 3-60        |
| JDBC: New LOB Streaming APIs .....  | 3-61        |
| New CLOB Streaming APIs.....  | 3-62        |
| New BFILE Streaming APIs.....   | 3-63        |
| JDBC and Empty LOBs .....   | 3-68        |
| <b>OLEDB (Oracle Provider for OLEDB — OraOLEDB)</b> .....                                 | <b>3-70</b> |

## 4 Managing LOBs

|  |            |
|--|------------|
| <b>Rules for using Directory Objects and BFILES</b> .....    | <b>4-1</b> |
| <b>DBA Actions Required Prior to Working with LOBs</b> ..... | <b>4-2</b> |
| Set Maximum Number of Open BFILES .....                      | 4-2        |
| Using SQL DML for Basic Operations on LOBs .....             | 4-2        |
| Changing Tablespace Storage for a LOB.....                   | 4-3        |
| <b>Managing Temporary LOBs</b> .....                         | <b>4-5</b> |
| <b>Using SQL*Loader to Load LOBs</b> .....                   | <b>4-5</b> |
| LOBFILES.....  | 4-5        |
| <b>Inline versus Out-of-Line LOBs</b> .....                  | <b>4-6</b> |

|  |      |
|--|------|
| <b>Loading InLine and Out-Of-Line Data into Internal LOBs Using SQL*Loader</b> ..... | 4-7  |
| SQL*Loader Performance: Loading Into Internal LOBs .....                             | 4-7  |
| <b>Loading Inline LOB Data</b> .....   | 4-8  |
| Loading Inline LOB Data in Predetermined Size Fields .....                           | 4-8  |
| Loading Inline LOB Data in Delimited Fields .....                                    | 4-9  |
| Loading Inline LOB Data in Length-Value Pair Fields .....                            | 4-9  |
| <b>Loading Out-Of-Line LOB Data</b> .....  | 4-10 |
| Loading One LOB Per File.....  | 4-11 |
| Loading Out-of-Line LOB Data in Predetermined Size Fields .....                      | 4-12 |
| Loading Out-of-Line LOB Data in Delimited Fields .....                               | 4-13 |
| Loading Out-of-Line LOB Data in Length-Value Pair Fields.....                        | 4-14 |
| <b>SQL*Loader LOB Loading Tips</b> .....   | 4-15 |
| <b>LOB Restrictions</b> .....  | 4-16 |

## 5 Large Objects: Advanced Topics

|  |      |
|--|------|
| <b>Introducing Large Objects: Advanced Topics</b> .....                          | 5-1  |
| <b>Read Consistent Locators</b> .....  | 5-1  |
| A Selected Locator Becomes a Read Consistent Locator .....                       | 5-2  |
| Updating LOBs and Read-Consistency .....   | 5-2  |
| Updating LOBs Via Updated Locators.....  | 5-5  |
| Example of Updating a LOB Using SQL DML and DBMS_LOB .....                       | 5-6  |
| Example of Using One Locator to Update the Same LOB Value.....                   | 5-8  |
| Example of Updating a LOB with a PL/SQL (DBMS_LOB) Bind Variable.....            | 5-10 |
| LOB Locators Cannot Span Transactions.....                                       | 5-12 |
| Example of Locator Not Spanning a Transaction .....                              | 5-13 |
| <b>LOB Locators and Transaction Boundaries</b> .....                             | 5-14 |
| Transaction IDs: Reading and Writing to a LOB Using Locators.....                | 5-15 |
| Non-Serializable Example: Selecting the Locator with No Current Transaction..... | 5-15 |
| Non-Serializable Example: Selecting the Locator within a Transaction.....        | 5-16 |
| <b>LOBs in the Object Cache</b> .....  | 5-17 |
| <b>LOB Buffering Subsystem</b> .....   | 5-18 |
| Advantages of LOB Buffering.....   | 5-18 |
| Guidelines for Using LOB Buffering.....  | 5-19 |
| LOB Buffering Usage Notes .....  | 5-21 |
| Flushing the LOB Buffer .....  | 5-22 |



|   |             |
|---|-------------|
| Flushing the Updated LOB .....  | 5-23        |
| Using Buffer-Enabled Locators.....                                      | 5-24        |
| Saving Locator State to Avoid a Reselect.....                           | 5-25        |
| OCI Example of LOB Buffering.....                                       | 5-25        |
| <b>Creating a Varray Containing References to LOBs .....</b>            | <b>5-28</b> |
| <b>LOBs in Partitioned Index-Organized Tables .....</b>                 | <b>5-29</b> |
| Example of LOB Columns in Partitioned Index-Organized Tables.....       | 5-29        |
| <b>Restrictions for LOBs in Partitioned Index-Organized Tables.....</b> | <b>5-31</b> |
| Range Partitioned Index-Organized Table LOB Restrictions .....          | 5-31        |
| Hash Partitioned Index-Organized Table LOB Restrictions .....           | 5-31        |

## 6 Frequently Asked Questions about LOBs

|  |            |
|--|------------|
| <b>Converting Data Types to LOB Data Types.....</b>                        | <b>6-2</b> |
| Can I Insert or Update Any Length Data Into a LOB Column? .....            | 6-2        |
| Does COPY LONG to LOB Work if Data is > 64K?.....                          | 6-2        |
| <b>General.....</b>  | <b>6-3</b> |
| How Do I Determine if the LOB Column with a Trigger is Being Updated?..... | 6-3        |
| Reading and Loading LOB Data: What Should Amount Parameter Size Be?.....   | 6-3        |
| Is LOB Data Lost After a Crash?.....                                       | 6-5        |
| <b>Index-Organized Tables (IOTs) and LOBs.....</b>                         | <b>6-5</b> |
| Is Inline Storage Allowed for LOBs in Index-Organized Tables? .....        | 6-5        |
| <b>Initializing LOB Locators.....</b>                                      | <b>6-6</b> |
| When Do I Use EMPTY_BLOB() and EMPTY_CLOB()? .....                         | 6-6        |
| How Do I Initialize a BLOB Attribute Using EMPTY_BLOB() in Java?.....      | 6-7        |
| <b>JDBC, JPublisher and LOBs .....</b>                                     | <b>6-7</b> |
| How Do I Insert a Row With Empty LOB Locator into Table Using JDBC? .....  | 6-7        |
| How Do I setData to EMPTY_BLOB() Using JPublisher? .....                   | 6-8        |
| JDBC: Do OracleBlob and OracleClob Work in 8.1.x? .....                    | 6-8        |
| How Do I Manipulate LOBs With the 8.1.5 JDBC Thin Driver?.....             | 6-9        |
| Is the FOR UPDATE Clause Needed on SELECT When Writing to a LOB? .....     | 6-9        |
| What Does DBMS_LOB.ERASE Do? .....   | 6-10       |
| Can I Use putChars()?.....   | 6-10       |
| Manipulating CLOB CharSetId in JDBC.....                                   | 6-11       |
| Why is Inserting into BLOBs Slower than into LONG Rows? .....              | 6-11       |
| Why Do I Get an ORA-03127 Error with LobLength on a LONG Column? .....     | 6-11       |

|  |      |
|--|------|
| How Do I Create a CLOB Object in a Java Program?.....  | 6-12 |
| How do I Load a 1MB File into a CLOB Column?.....  | 6-13 |
| How Do We Improve BLOB and CLOB Performance When Using<br>JDBC Driver To Load?.....  | 6-13 |
| <b>LOB Indexing</b> .....  | 6-16 |
| Is LOB Index Created in Same Tablespace as LOB Data?.....  | 6-16 |
| Indexing: Why is a BLOB Column Removed on DELETing but not a BFILE Column? ..  | 6-16 |
| Which Views Can I Query to Find Out About a LOB Index? .....   | 6-17 |
| <b>LOB Storage and Space Issues</b> .....  | 6-17 |
| What Happens If I Specify LOB Tablespace and ENABLE STORAGE IN ROW?.....   | 6-17 |
| What Are the Pros and Cons of Storing Images in a BFILE Versus a BLOB?.....  | 6-18 |
| When Should I Specify DISABLE STORAGE IN ROW?.....   | 6-19 |
| Do <4K BLOBs Go Into the Same Segment as Table Data,<br>>4K BLOBs Go Into a Specified Segment? .....   | 6-19 |
| Is 4K LOB Stored Inline?.....  | 6-19 |
| How is a LOB Locator Stored If the LOB Column is EMPTY_CLOB() or<br>EMPTY_BLOB() Instead of NULL? Are Extra Data Blocks Used For This? ..... | 6-20 |
| Storing CLOBs Inline: DISABLING STORAGE and Space Used .....   | 6-21 |
| Should I Include a LOB Storage Clause When Creating Tables With<br>Varray Columns? .....   | 6-21 |
| <b>LONG to LOB Migration</b> .....   | 6-23 |
| How Can We Migrate LONGs to LOBs, If Our Application Cannot Go Down?.....  | 6-23 |
| <b>Converting Between Different LOB Types</b> .....  | 6-24 |
| Is Implicit LOB Conversion Between Different LOB Types Allowed? .....  | 6-24 |
| <b>Performance</b> .....   | 6-24 |
| What Can We Do To Improve the Poor LOB Loading Performance When Using<br>Veritas File System on Disk Arrays, UNIX, and Oracle? .....         | 6-24 |
| Is There a Difference in Performance When Using DBMS_LOB.SUBSTR<br>Versus DBMS_LOB.READ?.....  | 6-26 |
| Are There Any White Papers or Guidelines on Tuning LOB Performance? .....  | 6-26 |
| When Should I Use Chunks Over Reading the Whole Thing? .....   | 6-26 |
| Is Inlining the LOB a Good Idea and If So When? .....  | 6-27 |
| How Can I Store LOBs >4Gb in the Database? .....   | 6-27 |
| Why is Performance Affected When Temporary LOBs are Created<br>in a Called Routine?.....   | 6-27 |
| <b>PL/SQL</b> .....  | 6-30 |

|                      |      |
|----------------------|------|
| UPLOAD_AS_BLOB ..... | 6-30 |
|----------------------|------|

## 7 Modeling and Design

|   |      |
|---|------|
| <b>Selecting a Datatype</b> .....   | 7-2  |
| LOBs Compared to LONG and LONG RAW Types.....   | 7-2  |
| Character Set Conversions: Varying-Width and Multi-byte Fixed-Width<br>Character Data ..... | 7-3  |
| <b>Selecting a Table Architecture</b> .....   | 7-4  |
| <b>LOB Storage</b> .....  | 7-4  |
| Where are NULL Values in a LOB Column Stored? .....   | 7-4  |
| Defining Tablespace and Storage Characteristics for Internal LOBs.....                      | 7-5  |
| LOB Storage Characteristics for LOB Column or Attribute.....                                | 7-6  |
| TABLESPACE and LOB Index .....  | 7-7  |
| PCTVERSION .....  | 7-7  |
| CACHE / NOCACHE / CACHE READS.....  | 7-8  |
| LOGGING / NOLOGGING .....   | 7-9  |
| CHUNK.....  | 7-10 |
| ENABLE   DISABLE STORAGE IN ROW .....   | 7-11 |
| Guidelines for ENABLE or DISABLE STORAGE IN ROW .....                                       | 7-11 |
| <b>How to Create Gigabyte LOBs</b> .....  | 7-12 |
| Example 1: Creating a Tablespace and Table to Store Gigabyte LOBs.....                      | 7-13 |
| Example 2: Creating a Tablespace and Table to Store Gigabyte LOBs.....                      | 7-13 |
| <b>LOB Locators and Transaction Boundaries</b> .....  | 7-14 |
| <b>Binds Greater Than 4,000 Bytes in INSERTs and UPDATEs</b> .....                          | 7-14 |
| Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs ...              | 7-14 |
| Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion .....             | 7-15 |
| 4,000 Byte Limit On Results of SQL Operator .....   | 7-16 |
| Binds of More Than 4,000 Bytes: Restrictions.....   | 7-16 |
| Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE... ..           | 7-16 |
| Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported .....             | 7-18 |
| Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes.....            | 7-18 |
| Example: C (OCI) - Binds of More than 4,000 Bytes For INSERT and UPDATE .....               | 7-19 |
| <b>OPEN, CLOSE, and ISOPEN Interfaces for Internal LOBs</b> .....                           | 7-22 |
| Example 1: Correct Use of OPEN/CLOSE Calls to LOBs in a Transaction .....                   | 7-23 |
| Example 2: Incorrect Use of OPEN/CLOSE Calls to a LOB in a Transaction .....                | 7-24 |

|   |      |
|---|------|
| <b>LOBs in Index Organized Tables (IOT)</b> .....                                     | 7-24 |
| Example of Index Organized Table (IOT) with LOB Columns.....                          | 7-25 |
| <b>Manipulating LOBs in Partitioned Tables</b> .....                                  | 7-26 |
| Creating and Partitioning a Table Containing LOB Data.....                            | 7-28 |
| Creating an Index on a Table Containing LOB Columns.....                              | 7-30 |
| Exchanging Partitions Containing LOB Data.....  | 7-30 |
| Adding Partitions to Tables Containing LOB Data.....                                  | 7-31 |
| Moving Partitions Containing LOBs.....  | 7-31 |
| Splitting Partitions Containing LOBs.....   | 7-31 |
| <b>Indexing a LOB Column</b> .....  | 7-32 |
| Functional Indexes on LOB Columns.....  | 7-32 |
| <b>SQL Semantics Support for LOBs</b> .....   | 7-33 |
| Improved LOB Usability: Accessing LOBs Using SQL “Character” Functions.....           | 7-33 |
| SQL and PL/SQL VARCHAR2 Functions/Operators Now Allowed for CLOBs.....                | 7-34 |
| PL/SQL Relational Operators Now Allowed for LOBs.....                                 | 7-34 |
| SQL and PL/SQL CHAR to CLOB Conversion Functions.....                                 | 7-34 |
| Non-Supported SQL Functionality for LOBs.....   | 7-35 |
| Using SQL Functions and Operators for VARCHAR2s on CLOBs.....                         | 7-35 |
| UNICODE Support for VARCHAR2 and CLOB.....  | 7-38 |
| SQL Features Where LOBs Cannot be Used.....   | 7-39 |
| <b>How SQL VARCHAR2/RAW Semantics Apply to CLOBs/BLOBs</b> .....                      | 7-40 |
| Defining CHAR Buffer on CLOB.....   | 7-40 |
| Accepting CLOBs in VARCHAR2 Operators/Functions.....                                  | 7-40 |
| Returning CLOB Values from SQL Functions/Operators.....                               | 7-41 |
| IS [NOT] NULL in VARCHAR2s and CLOBs.....   | 7-43 |
| <b>SQL RAW Type and BLOBs</b> .....   | 7-43 |
| <b>SQL DML Changes For LOBs</b> .....   | 7-43 |
| <b>SQL Functions/Operators for VARCHAR2s/RAWs and CLOBs/BLOBs</b> .....               | 7-44 |
| <b>PL/SQL Statements and Variables: New Semantics Changes</b> .....                   | 7-45 |
| Implicit Conversions Between CLOB and VARCHAR2.....                                   | 7-45 |
| PL/SQL Example 1: Prior Release SQL Interface for a<br>CLOB/VARCHAR2 Application..... | 7-46 |
| PL/SQL Example 2: Accessing CLOB Data When Treated as VARCHAR2s.....                  | 7-46 |
| PL/SQL Example 3: Defining a CLOB Variable on a VARCHAR2.....                         | 7-46 |
| Explicit Conversion Functions.....  | 7-47 |
| VARCHAR2 and CLOB in PL/SQL Built-in Functions.....                                   | 7-47 |

|   |             |
|---|-------------|
| PL/SQL Example 4: CLOB Variables in PL/SQL.....                           | 7-48        |
| PL/SQL Example 5: Change in Locator-Data Linkage.....                     | 7-48        |
| PL/SQL Example 6: Freeing Temporary LOBs Automatically and Manually ..... | 7-49        |
| <b>PL/SQL CLOB Comparison Rules.....</b>                                  | <b>7-50</b> |
| <b>Interacting with SQL and PL/SQL in OCI and Java Interfaces .....</b>   | <b>7-50</b> |
| <b>Performance Attributes When Using SQL Semantics with LOBs .....</b>    | <b>7-51</b> |
| Inserting More than 4K Bytes Data Into LOB Columns.....                   | 7-51        |
| Temporary LOB Creation/Deallocation .....                                 | 7-51        |
| Performance Measurement .....   | 7-52        |
| <b>User-Defined Aggregates and LOBs .....</b>                             | <b>7-53</b> |
| UDAGs: DDL Support .....  | 7-54        |
| UDAGs: DML and Query Support .....  | 7-54        |

## 8 Migrating From LONGs to LOBs

|  |             |
|--|-------------|
| <b>Introducing LONG-to-LOB Migration.....</b>  | <b>8-2</b>  |
| Using the LONG-to-LOB API Results in an Easy Migration .....                           | 8-2         |
| <b>Guidelines for Using LONG-to-LOB API .....</b>                                      | <b>8-3</b>  |
| Using ALTER TABLE.....   | 8-3         |
| LONG-to-LOB API and OCI.....   | 8-3         |
| LONG-to-LOB API and PL/SQL .....   | 8-5         |
| <b>Migrating Existing Tables from LONG to LOB.....</b>                                 | <b>8-6</b>  |
| Migrating LONGs to LOBs: Using ALTER TABLE to Change<br>LONG Column to LOB Types ..... | 8-6         |
| <b>LONG-to-LOB Migration Limitations.....</b>  | <b>8-10</b> |
| LONGs, LOBs, and NULLs.....  | 8-11        |
| <b>Using LONG-to-LOB API with OCI.....</b>   | <b>8-12</b> |
| Guidelines for Using LONG-to-LOB API for LOBs with OCI.....                            | 8-13        |
| <b>Using OCI Functions to Perform INSERT or UPDATE on LOBs .....</b>                   | <b>8-13</b> |
| <b>Using OCI Functions to Perform FETCH on LOBs.....</b>                               | <b>8-14</b> |
| <b>Using SQL and PL/SQL to Access LONGs and LOBs.....</b>                              | <b>8-16</b> |
| Using SQL and PL/SQL to Access LOBs.....   | 8-16        |
| Implicit Assignment and Parameter Passing .....  | 8-17        |
| Explicit Conversion Functions.....   | 8-18        |
| VARCHAR2 and CLOB in PL/SQL Built-In Functions .....                                   | 8-19        |
| PL/SQL and C Binds from OCI .....  | 8-20        |

|  |             |
|--|-------------|
| Calling PL/SQL and C Procedures from SQL or PL/SQL.....  | 8-21        |
| <b>Applications Requiring Changes When Converting From LONGs to LOBs .....</b>   | <b>8-22</b> |
| <b>Overloading with Anchored Types .....</b>   | <b>8-22</b> |
| <b>Implicit Conversion of NUMBER, DATE, ROW_ID, BINARY_INTEGER,<br/>    and PLS_INTEGER to LOB is Not Supported 8-23</b> |             |
| No Implicit Conversions of BLOB to VARCHAR2, CHAR, or CLOB to RAW<br>or LONG RAW 8-23                                    |             |
| <b>Using utldtree.sql to Determine Where Your Application Needs Change.....</b>  | <b>8-23</b> |
| <b>Examples of Converting from LONG to LOB Using Table Multimedia_tab.....</b>   | <b>8-24</b> |
| Converting LONG to LOB Example 1: More than 4K Binds and Simple INSERTs.....   | 8-25        |
| Converting LONG to LOB Example 2: Piecewise INSERT with Polling.....   | 8-26        |
| Converting LONG to LOB Example 3: Piecewise INSERT with Callback .....   | 8-27        |
| Converting LONG to LOB Example 4: Array insert.....  | 8-29        |
| Converting LONG to LOB Example 5: Simple Fetch .....   | 8-31        |
| Converting LONG to LOB Example 6: Piecewise Fetch with Polling.....  | 8-31        |
| Converting LONG to LOB Example 7: Piecewise Fetch with Callback.....   | 8-32        |
| Converting LONG to LOB Example 8: Array Fetch .....  | 8-34        |
| Converting LONG to LOB Example 9: Using PL/SQL in INSERT, UPDATE<br>and SELECT.....                                      | 8-36        |
| Converting LONG to LOB Example 10: Assignments and Parameter Passing<br>in PL/SQL.....                                   | 8-36        |
| Converting LONG to LOB Example 11: CLOBs in PL/SQL Built-In Functions .....  | 8-37        |
| Converting LONG to LOB Example 12: Using PL/SQL Binds from OCI on LOBs .....   | 8-37        |
| Converting LONG to LOB Example 13: Calling PL/SQL and C Procedures<br>from PL/SQL.....                                   | 8-39        |
| <b>Summary of New Functionality Associated with the LONG-to-LOB API.....</b>   | <b>8-40</b> |
| OCI Functions.....   | 8-40        |
| SQL Statements .....   | 8-40        |
| PL/SQL Interface.....  | 8-40        |
| <b>Compatibility and Migration .....</b>   | <b>8-41</b> |
| <b>Performance .....</b>   | <b>8-42</b> |
| <b>Frequently Asked Questions (FAQs): LONG to LOB Migration .....</b>  | <b>8-42</b> |
| <b>Moving From LOBs Back to LONGs .....</b>  | <b>8-42</b> |
| Is CREATE VIEW Needed? .....   | 8-43        |
| Are OCI LOB Routines Obsolete? .....   | 8-43        |
| PL/SQL Issues.....   | 8-43        |

|   |      |
|---|------|
| Retrieving an Entire Image if Less Than 32K ..... | 8-44 |
| Triggers in LONGs and LOBs.....                   | 8-44 |

## 9 LOBS: Best Practices

|  |      |
|--|------|
| <b>Using SQL*Loader</b> .....                              | 9-2  |
| Loading XML Documents Into LOBs With SQL*Loader.....       | 9-2  |
| <b>LOB Performance Guidelines</b> .....                    | 9-5  |
| General Performance Guidelines .....                       | 9-5  |
| Temporary LOB Performance Guidelines .....                 | 9-7  |
| <b>Moving Data to LOBs in a Threaded Environment</b> ..... | 9-9  |
| Incorrect procedure .....                                  | 9-9  |
| The Correct Procedure.....                                 | 9-10 |
| <b>Migrating from LONGs to LOBs</b> .....                  | 9-10 |

## 10 Internal Persistent LOBs

|  |       |
|--|-------|
| <b>Use Case Model: Internal Persistent LOBs Operations</b> .....                             | 10-2  |
| <b>Creating a Table Containing One or More LOB Columns</b> .....                             | 10-5  |
| SQL: Create a Table Containing One or More LOB Columns .....                                 | 10-7  |
| <b>Creating a Table Containing an Object Type with a LOB Attribute</b> .....                 | 10-10 |
| SQL: Creating a Table Containing an Object Type with a LOB Attribute .....                   | 10-11 |
| <b>Creating a Nested Table Containing a LOB</b> .....  | 10-13 |
| SQL: Creating a Nested Table Containing a LOB.....   | 10-14 |
| <b>Inserting a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()</b> .....                        | 10-16 |
| SQL: Inserting a Value Using EMPTY_CLOB() / EMPTY_BLOB() .....                               | 10-19 |
| <b>Inserting a Row by Selecting a LOB From Another Table</b> .....                           | 10-20 |
| SQL: Inserting a Row by Selecting a LOB from Another Table.....                              | 10-22 |
| <b>Inserting a Row by Initializing a LOB Locator Bind Variable</b> .....                     | 10-23 |
| PL/SQL (DBMS_LOB Package): Inserting a Row by Initializing a LOB Locator Bind Variable ..... | 10-25 |
| C (OCI): Inserting a Row by Initializing a LOB Locator Bind Variable.....                    | 10-26 |
| COBOL (Pro*COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable ..            | 10-28 |
| C/C++ (ProC/C++): Inserting a Row by Initializing a LOB Locator Bind Variable.....           | 10-29 |
| Visual Basic (OO4O): Inserting a Row by Initializing a LOB Locator Bind Variable .....       | 10-30 |
| Java (JDBC): Inserting a Row by Initializing a LOB Locator Bind Variable .....               | 10-30 |
| <b>Loading a LOB with Data From a BFILE</b> .....  | 10-32 |

|  |              |
|--|--------------|
| PL/SQL (DBMS_LOB Package): Loading a LOB with Data from a BFILE.....                   | 10-35        |
| C (OCI): Loading a LOB with Data from a BFILE.....                                     | 10-36        |
| COBOL (Pro*COBOL): Loading a LOB with Data from a BFILE.....                           | 10-38        |
| Visual Basic (OO4O): Loading a LOB with Data from a BFILE.....                         | 10-39        |
| Java (JDBC): Loading a LOB with Data from a BFILE .....                                | 10-40        |
| <b>Loading an Internal Persistent BLOB with Binary Data from a BFILE.....</b>          | <b>10-43</b> |
| PL/SQL (DBMS_LOB Package): Loading an Internal Persistent BLOB<br>with BFILE Data..... | 10-45        |
| <b>Loading an Internal Persistent CLOB with BFILE Data.....</b>                        | <b>10-47</b> |
| PL/SQL (DBMS_LOB Package): Loading an Internal Persistent CLOB with<br>BFILE Data..... | 10-49        |
| <b>Open: Checking If a LOB Is Open.....</b>  | <b>10-53</b> |
| PL/SQL (DBMS_LOB Package): Checking if a LOB is Open.....                              | 10-54        |
| C (OCI): Checking if a LOB is Open .....   | 10-55        |
| COBOL (Pro*COBOL): Checking if a LOB is Open.....                                      | 10-56        |
| C/C++ (ProC/C++): Checking if a LOB is Open.....                                       | 10-58        |
| Java (JDBC): Checking if a LOB is Open .....   | 10-59        |
| <b>LONG to LOB Migration Using the LONG-to-LOB API.....</b>                            | <b>10-62</b> |
| C (OCI): LONG to LOB Migration .....   | 10-63        |
| <b>LONG to LOB Copying, Using the TO_LOB Operator.....</b>                             | <b>10-66</b> |
| SQL: Copying LONGs to LOBs Using TO_LOB Operator .....                                 | 10-67        |
| <b>Checking Out a LOB .....</b>  | <b>10-71</b> |
| PL/SQL (DBMS_LOB Package): Checking Out a LOB.....                                     | 10-73        |
| C (OCI): Checking Out a LOB.....   | 10-74        |
| COBOL (Pro*COBOL): Checking Out a LOB .....  | 10-76        |
| C/C++ (ProC/C++): Checking Out a LOB.....  | 10-78        |
| Visual Basic (OO4O): Checking Out a LOB .....  | 10-80        |
| Java (JDBC): Checking Out a LOB.....   | 10-80        |
| <b>Checking In a LOB.....</b>  | <b>10-83</b> |
| PL/SQL (DBMS_LOB Package): Checking in a LOB .....                                     | 10-85        |
| C (OCI): Checking in a LOB .....   | 10-86        |
| COBOL (Pro*COBOL): Checking in a LOB.....  | 10-89        |
| C/C++ (ProC/C++): Checking in a LOB .....  | 10-92        |
| Visual Basic (OO4O): Checking in a LOB .....   | 10-94        |
| Java (JDBC): Checking in a LOB.....  | 10-95        |
| <b>Displaying LOB Data .....</b>   | <b>10-98</b> |



|   |        |
|---|--------|
| PL/SQL (DBMS_LOB Package): Displaying LOB Data.....                     | 10-100 |
| C (OCI): Displaying LOB Data .....                                      | 10-101 |
| COBOL (Pro*COBOL): Displaying LOB Data.....                             | 10-103 |
| C/C++ (ProC/C++): Displaying LOB Data .....                             | 10-105 |
| Visual Basic (OO4O): Displaying LOB Data.....                           | 10-106 |
| Java (JDBC): Displaying LOB Data .....                                  | 10-107 |
| <b>Reading Data from a LOB</b> .....                                    | 10-109 |
| PL/SQL (DBMS_LOB Package): Reading Data from a LOB.....                 | 10-112 |
| C (OCI): Reading Data from a LOB .....                                  | 10-113 |
| COBOL (Pro*COBOL): Reading Data from a LOB.....                         | 10-115 |
| C/C++ (Pro*C/C++): Reading Data from a LOB.....                         | 10-116 |
| Visual Basic (OO4O): Reading Data from a LOB.....                       | 10-118 |
| Java (JDBC): Reading Data from a LOB .....                              | 10-118 |
| <b>Reading a Portion of the LOB (substr)</b> .....                      | 10-120 |
| PL/SQL (DBMS_LOB Package): Reading a Portion of the LOB (substr) .....  | 10-122 |
| COBOL (Pro*COBOL): Reading a Portion of the LOB (substr) .....          | 10-123 |
| C/C++ (Pro*C/C++): Reading a Portion of the LOB (substr) .....          | 10-124 |
| Visual Basic (OO4O): Reading a Portion of the LOB (substr) .....        | 10-125 |
| Java (JDBC): Reading a Portion of the LOB (substr).....                 | 10-126 |
| <b>Comparing All or Part of Two LOBs</b> .....                          | 10-128 |
| PL/SQL (DBMS_LOB Package): Comparing All or Part of Two LOBs .....      | 10-130 |
| COBOL (Pro*COBOL): Comparing All or Part of Two LOBs.....               | 10-130 |
| C/C++ (Pro*C/C++): Comparing All or Part of Two LOBs.....               | 10-132 |
| Visual Basic (OO4O): Comparing All or Part of Two LOBs.....             | 10-133 |
| Java (JDBC): Comparing All or Part of Two LOBs .....                    | 10-134 |
| <b>Patterns: Checking for Patterns in the LOB (instr)</b> .....         | 10-137 |
| PL/SQL (DBMS_LOB Package): Checking for Pattern in the LOB (instr)..... | 10-139 |
| COBOL (Pro*COBOL): Checking for Patterns in the LOB (instr) .....       | 10-139 |
| C/C++ (Pro*C/C++): Checking for Patterns in the LOB (instr).....        | 10-141 |
| Java (JDBC): Checking for Patterns in the LOB (instr) .....             | 10-142 |
| <b>Length: Determining the Length of a LOB</b> .....                    | 10-144 |
| PL/SQL (DBMS_LOB Package): Determining the Length of a LOB .....        | 10-146 |
| C (OCI): Determining the Length of a LOB .....                          | 10-146 |
| COBOL (Pro*COBOL): Determining the Length of a LOB .....                | 10-148 |
| C/C++ (Pro*C/C++): Determining the Length of a LOB.....                 | 10-149 |

|   |        |
|---|--------|
| Visual Basic (OO4O): Determining the Length of a LOB.....                   | 10-150 |
| Java (JDBC): Determining the Length of a LOB .....                          | 10-150 |
| <b>Copying All or Part of One LOB to Another LOB</b> .....                  | 10-153 |
| PL/SQL (DBMS_LOB Package): Copying All or Part of One LOB to Another LOB... | 10-155 |
| C (OCI): Copying All or Part of One LOB to Another LOB .....                | 10-156 |
| COBOL (Pro*COBOL): Copying All or Part of One LOB to Another LOB.....       | 10-158 |
| C/C++ (Pro*C/C++): Copy All or Part of a LOB to Another LOB .....           | 10-160 |
| Visual Basic (OO4O): Copying All or Part of One LOB to Another LOB.....     | 10-161 |
| Java (JDBC): Copying All or Part of One LOB to Another LOB .....            | 10-162 |
| <b>Copying a LOB Locator</b> .....  | 10-164 |
| PL/SQL (DBMS_LOB Package): Copying a LOB Locator .....                      | 10-166 |
| C (OCI): Copying a LOB Locator.....   | 10-166 |
| COBOL (Pro*COBOL): Copying a LOB Locator .....                              | 10-168 |
| C/C++ (Pro*C/C++): Copying a LOB Locator .....                              | 10-169 |
| Visual Basic (OO4O): Copying a LOB Locator .....                            | 10-170 |
| Java (JDBC): Copying a LOB Locator .....                                    | 10-170 |
| <b>Equality: Checking If One LOB Locator Is Equal to Another</b> .....      | 10-173 |
| C (OCI): Checking If One LOB Locator Is Equal to Another.....               | 10-174 |
| C/C++ (Pro*C/C++): Checking If One LOB Locator Is Equal to Another .....    | 10-176 |
| Java (JDBC): Checking If One LOB Locator Is Equal to Another .....          | 10-177 |
| <b>Initialized Locator: Checking If a LOB Locator Is Initialized</b> .....  | 10-180 |
| C (OCI): Checking If a LOB Locator Is Initialized .....                     | 10-181 |
| C/C++ (Pro*C/C++): Checking If a LOB Locator Is Initialized .....           | 10-183 |
| <b>Character Set ID: Determining Character Set ID</b> .....                 | 10-185 |
| C (OCI): Determining Character Set ID.....                                  | 10-187 |
| <b>Character Set Form: Determining Character Set Form</b> .....             | 10-189 |
| C (OCI): Determining Character Set Form.....                                | 10-190 |
| <b>Appending One LOB to Another</b> .....                                   | 10-193 |
| PL/SQL (DBMS_LOB Package): Appending One LOB to Another.....                | 10-195 |
| C (OCI): Appending One LOB to Another .....                                 | 10-196 |
| COBOL (Pro*COBOL): Appending One LOB to Another.....                        | 10-198 |
| C/C++ (Pro*C/C++): Appending One LOB to Another .....                       | 10-199 |
| Visual Basic (OO4O): Appending One LOB to Another.....                      | 10-200 |
| Java (JDBC): Appending One LOB to Another .....                             | 10-201 |
| <b>Append-Writing to the End of a LOB</b> .....                             | 10-203 |

|   |        |
|---|--------|
| PL/SQL (DBMS_LOB Package): Writing to the End of (Appending to) a LOB ..... | 10-205 |
| C (OCI): Writing to the End of (Appending to) a LOB .....                   | 10-206 |
| COBOL (Pro*COBOL): Writing to the End of (Appending to) a LOB.....          | 10-208 |
| C/C++ (Pro*C/C++): Writing to the End of (Appending to) a LOB.....          | 10-209 |
| Java (JDBC): Writing to the End of (Append-Write to) a LOB.....             | 10-210 |
| <b>Writing Data to a LOB</b> .....  | 10-212 |
| PL/SQL (DBMS_LOB Package): Writing Data to a LOB.....                       | 10-215 |
| C (OCI): Writing Data to a LOB.....   | 10-217 |
| COBOL (Pro*COBOL): Writing Data to a LOB.....                               | 10-220 |
| C/C++ (Pro*C/C++): Writing Data to a LOB .....                              | 10-222 |
| Visual Basic (OO4O):Writing Data to a LOB .....                             | 10-225 |
| Java (JDBC): Writing Data to a LOB .....                                    | 10-226 |
| <b>Trimming LOB Data</b> .....  | 10-229 |
| PL/SQL (DBMS_LOB Package): Trimming LOB Data .....                          | 10-231 |
| C (OCI): Trimming LOB Data.....   | 10-232 |
| COBOL (Pro*COBOL): Trimming LOB Data .....                                  | 10-233 |
| C/C++ (Pro*C/C++): Trimming LOB Data .....                                  | 10-234 |
| Visual Basic (OO4O): Trimming LOB Data .....                                | 10-236 |
| Java (JDBC): Trimming LOB Data.....   | 10-236 |
| <b>Erasing Part of a LOB</b> .....  | 10-241 |
| PL/SQL (DBMS_LOB Package): Erasing Part of a LOB .....                      | 10-243 |
| C (OCI): Erasing Part of a LOB.....   | 10-244 |
| COBOL (Pro*COBOL): Erasing Part of a LOB .....                              | 10-245 |
| C/C++ (Pro*C/C++): Erasing Part of a LOB.....                               | 10-246 |
| Visual Basic (OO4O): Erasing Part of a LOB .....                            | 10-247 |
| Java (JDBC): Erasing Part of a LOB.....                                     | 10-248 |
| <b>Enabling LOB Buffering</b> .....   | 10-251 |
| C (OCI): Enabling LOB Buffering.....  | 10-253 |
| COBOL (Pro*COBOL): Enabling LOB Buffering .....                             | 10-253 |
| C/C++ (Pro*C/C++): Enabling LOB Buffering .....                             | 10-255 |
| Visual Basic (OO4O): Enabling LOB Buffering.....                            | 10-256 |
| <b>Flushing the Buffer</b> .....  | 10-257 |
| C (OCI): Flushing the Buffer .....  | 10-259 |
| COBOL (Pro*COBOL): Flushing the Buffer.....                                 | 10-259 |
| C/C++ (Pro*C/C++): Flushing the Buffer.....                                 | 10-261 |

|   |        |
|---|--------|
| Visual Basic (OO4O): Flushing the Buffer .....                                  | 10-262 |
| <b>Disabling LOB Buffering</b> .....  | 10-263 |
| C (OCI): Disabling LOB Buffering .....  | 10-265 |
| COBOL (Pro*COBOL): Disabling LOB Buffering .....                                | 10-267 |
| C/C++ (Pro*C/C++): Disabling LOB Buffering .....                                | 10-269 |
| Visual Basic (OO4O): Disabling LOB Buffering .....                              | 10-270 |
| <b>Updating a LOB with EMPTY_CLOB() or EMPTY_BLOB()</b> .....                   | 10-271 |
| SQL: UPDATE a LOB with EMPTY_CLOB() or EMPTY_BLOB() .....                       | 10-273 |
| <b>Updating a Row by Selecting a LOB From Another Table</b> .....               | 10-275 |
| SQL: Update a Row by Selecting a LOB From Another Table .....                   | 10-276 |
| <b>Updating by Initializing a LOB Locator Bind Variable</b> .....               | 10-277 |
| PL/SQL: Updating by Initializing a LOB Locator Bind Variable .....              | 10-279 |
| C (OCI): Updating by Initializing a LOB Locator Bind Variable .....             | 10-279 |
| COBOL (Pro*COBOL): Updating by Initializing a LOB Locator Bind Variable .....   | 10-281 |
| C/C++ (Pro*C/C++): Updating by Initializing a LOB Locator Bind Variable .....   | 10-282 |
| Visual Basic (OO4O): Updating by Initializing a LOB Locator Bind Variable ..... | 10-283 |
| Java (JDBC): Updating by Initializing a LOB Locator Bind Variable .....         | 10-284 |
| <b>Deleting the Row of a Table Containing a LOB</b> .....                       | 10-286 |
| SQL: Delete a LOB .....   | 10-287 |

## 11 Temporary LOBs

|  |       |
|--|-------|
| <b>Use Case Model: Internal Temporary LOBs</b> .....                             | 11-1  |
| <b>Programmatic Environments</b> .....   | 11-4  |
| Locators .....   | 11-4  |
| Temporary LOB Locators Can be IN Values .....                                    | 11-4  |
| Can You Use the Same Functions for Temporary and Internal Persistent LOBs? ..... | 11-5  |
| DBMS_LOB.createtemporary() Parameter is a Hint .....                             | 11-5  |
| Temporary LOB Data is Stored in Temporary Tablespace .....                       | 11-5  |
| Lifetime and Duration of Temporary LOBs .....                                    | 11-5  |
| Memory Handling .....  | 11-6  |
| Locators and Semantics .....   | 11-7  |
| <b>Features Specific to Temporary LOBs</b> .....                                 | 11-7  |
| Security Issues with Temporary LOBs .....  | 11-10 |
| NOCOPY Restrictions .....  | 11-10 |
| Managing Temporary LOBs .....  | 11-10 |

|   |              |
|---|--------------|
| Using JDBC and Temporary LOBs .....   | 11-11        |
| <b>Creating a Temporary LOB .....</b>   | <b>11-13</b> |
| PL/SQL (DBMS_LOB Package): Creating a Temporary LOB .....                         | 11-15        |
| C (OCI): Creating a Temporary LOB .....   | 11-16        |
| COBOL (Pro*COBOL): Creating a Temporary LOB .....                                 | 11-18        |
| C/C++ (Pro*C/C++): Creating a Temporary LOB .....                                 | 11-19        |
| Java (JDBC): Creating a Temporary BLOB .....                                      | 11-20        |
| Java (JDBC): Creating a Temporary CLOB .....                                      | 11-21        |
| <b>Checking If a LOB is Temporary .....</b>                                       | <b>11-23</b> |
| PL/SQL (DBMS_LOB Package): Checking If a LOB is Temporary .....                   | 11-24        |
| C (OCI): Checking If a LOB is Temporary .....                                     | 11-25        |
| COBOL (Pro*COBOL): Checking If a LOB is Temporary .....                           | 11-26        |
| C/C++ (Pro*C/C++): Checking If a LOB is Temporary .....                           | 11-27        |
| Java (JDBC): Checking if a BLOB is Temporary .....                                | 11-28        |
| Java (JDBC): Checking if a CLOB is Temporary .....                                | 11-29        |
| <b>Freeing a Temporary LOB .....</b>  | <b>11-31</b> |
| PL/SQL (DBMS_LOB Package): Freeing a Temporary LOB .....                          | 11-32        |
| C (OCI): Freeing a Temporary LOB .....  | 11-33        |
| COBOL (Pro*COBOL): Freeing a Temporary LOB .....                                  | 11-34        |
| C/C++ (Pro*C/C++): Freeing a Temporary LOB.....                                   | 11-35        |
| Java (JDBC): Freeing a Temporary BLOB .....                                       | 11-36        |
| Java (JDBC): Freeing a Temporary CLOB .....                                       | 11-37        |
| Java (JDBC): Creating and Freeing a Temporary CLOB Using TemporaryClob.java....   | 11-38        |
| <b>Loading a Temporary LOB with Data from a BFILE.....</b>                        | <b>11-39</b> |
| PL/SQL (DBMS_LOB Package): Loading a Temporary LOB with Data from a BFILE.        | 11-41        |
| C (OCI): Loading a Temporary LOB with Data from a BFILE.....                      | 11-42        |
| COBOL (Pro*COBOL): Loading a Temporary LOB with Data from a BFILE.....            | 11-44        |
| C/C++ (Pro*C/C++): Loading a Temporary LOB with Data from a BFILE.....            | 11-46        |
| <b>Loading a Temporary BLOB with Binary Data from a BFILE.....</b>                | <b>11-48</b> |
| PL/SQL (DBMS_LOB Package): Loading a Temporary BLOB with BFILE data.....          | 11-50        |
| <b>Loading a Temporary CLOB/NCLOB with Character Data from a File.....</b>        | <b>11-52</b> |
| PL/SQL (DBMS_LOB Package): Loading a Temporary CLOB/NCLOB<br>with BFILE Data..... | 11-54        |
| <b>Determining If a Temporary LOB Is Open.....</b>                                | <b>11-57</b> |
| PL/SQL: Determining if a Temporary LOB is Open.....                               | 11-58        |
| C (OCI) Determining if a Temporary LOB is Open .....                              | 11-59        |

|  |               |
|--|---------------|
| COBOL (Pro*COBOL): Determining if a Temporary LOB is Open .....                        | 11-60         |
| C/C++ (Pro*C/C++): Determining if a Temporary LOB is Open .....                        | 11-62         |
| <b>Displaying Temporary LOB Data .....</b>   | <b>11-64</b>  |
| PL/SQL (DBMS_LOB Package): Displaying Temporary LOB Data .....                         | 11-66         |
| C (OCI): Displaying Temporary LOB Data .....   | 11-67         |
| COBOL (Pro*COBOL): Displaying Temporary LOB Data .....                                 | 11-70         |
| C/C++ (Pro*C/C++): Displaying Temporary LOB Data .....                                 | 11-72         |
| <b>Reading Data from a Temporary LOB .....</b>   | <b>11-74</b>  |
| PL/SQL (DBMS_LOB Package): Reading Data from a Temporary LOB .....                     | 11-76         |
| C (OCI): Reading Data from a Temporary LOB .....                                       | 11-77         |
| COBOL (Pro*COBOL): Reading Data from a Temporary LOB .....                             | 11-80         |
| C/C++ (Pro*C/C++): Reading Data from a Temporary LOB .....                             | 11-82         |
| <b>Reading Portion of Temporary LOB (Substr) .....</b>                                 | <b>11-85</b>  |
| PL/SQL (DBMS_LOB Package): Reading a Portion of Temporary LOB (substr) .....           | 11-87         |
| COBOL (Pro*COBOL): Reading a Portion of Temporary LOB (substr) .....                   | 11-87         |
| C/C++ (Pro*C/C++): Reading a Portion of Temporary LOB (substr) .....                   | 11-89         |
| <b>Comparing All or Part of Two Temporary LOBs .....</b>                               | <b>11-92</b>  |
| PL/SQL (DBMS_LOB Package): Comparing All or Part of Two Temporary LOBs .....           | 11-94         |
| COBOL (Pro*COBOL): Comparing All or Part of Two Temporary LOBs .....                   | 11-95         |
| C/C++ (Pro*C/C++): Comparing All or Part of Two Temporary LOBs .....                   | 11-97         |
| <b>Determining If a Pattern Exists in a Temporary LOB (instr) .....</b>                | <b>11-99</b>  |
| PL/SQL (DBMS_LOB Package): Determining If a Pattern Exists in a<br>Temporary LOB ..... | 11-101        |
| COBOL (Pro*COBOL): Determining If a Pattern Exists in a Temporary LOB (instr) .        | 11-101        |
| C/C++ (Pro*C/C++): Determining If a Pattern Exists in a Temporary LOB (instr) ...      | 11-104        |
| <b>Finding the Length of a Temporary LOB .....</b>                                     | <b>11-106</b> |
| PL/SQL (DBMS_LOB Package): Finding the Length of a Temporary LOB .....                 | 11-108        |
| C (OCI): Finding the Length of a Temporary LOB .....                                   | 11-108        |
| COBOL (Pro*COBOL): Finding the Length of a Temporary LOB .....                         | 11-110        |
| C/C++ (Pro*C/C++): Finding the Length of a Temporary LOB .....                         | 11-112        |
| <b>Copying All or Part of One Temporary LOB to Another .....</b>                       | <b>11-115</b> |
| PL/SQL (DBMS_LOB Package): Copying All or Part of One Temporary LOB .....              | 11-117        |
| C (OCI): Copying All or Part of One Temporary LOB to Another .....                     | 11-118        |
| COBOL (Pro*COBOL): Copying All or Part of One Temporary LOB to Another .....           | 11-121        |
| C/C++ (Pro*C/C++): Copying All or Part of One Temporary LOB to Another .....           | 11-123        |
| <b>Copying a LOB Locator for a Temporary LOB .....</b>                                 | <b>11-125</b> |

|  |               |
|--|---------------|
| PL/SQL (DBMS_LOB Package): Copying a LOB Locator for a Temporary LOB.....      | 11-127        |
| C (OCI): Copying a LOB Locator for a Temporary LOB .....                       | 11-128        |
| COBOL (Pro*COBOL): Copying a LOB Locator for a Temporary LOB .....             | 11-130        |
| C/C++ (Pro*C/C++): Copying a LOB Locator for a Temporary LOB.....              | 11-132        |
| <b>Is One Temporary LOB Locator Equal to Another .....</b>                     | <b>11-135</b> |
| C (OCI): Is One LOB Locator for a Temporary LOB Equal to Another .....         | 11-136        |
| C/C++ (Pro*C/C++): Is One LOB Locator for a Temporary LOB Equal to Another .   | 11-138        |
| <b>Determining if a LOB Locator for a Temporary LOB Is Initialized.....</b>    | <b>11-140</b> |
| C (OCI): Determining If a LOB Locator for a Temporary LOB Is Initialized ..... | 11-141        |
| C/C++ (Pro*C/C++): Determining If a LOB Locator Is Initialized .....           | 11-142        |
| <b>Finding Character Set ID of a Temporary LOB.....</b>                        | <b>11-144</b> |
| C (OCI): Finding Character Set ID of a Temporary LOB .....                     | 11-145        |
| <b>Finding Character Set Form of a Temporary LOB .....</b>                     | <b>11-147</b> |
| C (OCI): Finding Character Set Form of a Temporary LOB.....                    | 11-148        |
| <b>Appending One Temporary LOB to Another.....</b>                             | <b>11-150</b> |
| PL/SQL (DBMS_LOB Package): Appending One Temporary LOB to Another.....         | 11-152        |
| C (OCI): Appending One Temporary LOB to Another .....                          | 11-152        |
| COBOL (Pro*COBOL): Appending One Temporary LOB to Another.....                 | 11-155        |
| C/C++ (Pro*C/C++): Appending One Temporary LOB to Another .....                | 11-157        |
| <b>Write-Appending to a Temporary LOB .....</b>                                | <b>11-160</b> |
| PL/SQL (DBMS_LOB Package): Writing-Appending to a Temporary LOB.....           | 11-162        |
| C (OCI): Writing-Appending to a Temporary LOB .....                            | 11-162        |
| COBOL (Pro*COBOL): Write-Appending to a Temporary LOB .....                    | 11-164        |
| C/C++ (Pro*C/C++): Write-Appending to a Temporary LOB.....                     | 11-166        |
| <b>Writing Data to a Temporary LOB.....</b>                                    | <b>11-168</b> |
| PL/SQL (DBMS_LOB Package): Writing Data to a Temporary LOB.....                | 11-171        |
| C (OCI): Writing Data to a Temporary LOB.....                                  | 11-171        |
| COBOL (Pro*COBOL): Writing Data to a Temporary LOB .....                       | 11-174        |
| C/C++ (Pro*C/C++): Writing Data to a Temporary LOB .....                       | 11-176        |
| <b>Trimming Temporary LOB Data .....</b>                                       | <b>11-179</b> |
| PL/SQL (DBMS_LOB Package): Trimming Temporary LOB Data .....                   | 11-181        |
| C (OCI): Trimming Temporary LOB Data.....                                      | 11-181        |
| COBOL (Pro*COBOL): Trimming Temporary LOB Data .....                           | 11-184        |
| C/C++ (Pro*C/C++): Trimming Temporary LOB Data.....                            | 11-185        |
| <b>Erasing Part of a Temporary LOB .....</b>                                   | <b>11-188</b> |

|  |               |
|--|---------------|
| PL/SQL (DBMS_LOB Package): Erasing Part of a Temporary LOB .....     | 11-190        |
| (OCI): Erasing Part of a Temporary LOB .....                         | 11-190        |
| COBOL (Pro*COBOL): Erasing Part of a Temporary LOB.....              | 11-193        |
| C/C++ (Pro*C/C++): Erasing Part of a Temporary LOB.....              | 11-195        |
| <b>Enabling LOB Buffering for a Temporary LOB .....</b>              | <b>11-197</b> |
| C (OCI): Enabling LOB Buffering for a Temporary LOB.....             | 11-199        |
| COBOL (Pro*COBOL): Enabling LOB Buffering for a Temporary LOB .....  | 11-200        |
| C/C++ (Pro*C/C++): Enabling LOB Buffering for a Temporary LOB .....  | 11-202        |
| <b>Flushing Buffer for a Temporary LOB.....</b>                      | <b>11-204</b> |
| C (OCI): Flushing Buffer for a Temporary LOB.....                    | 11-205        |
| COBOL (Pro*COBOL): Flushing Buffer for a Temporary LOB .....         | 11-207        |
| C/C++ (Pro*C/C++): Flushing Buffer for a Temporary LOB .....         | 11-209        |
| <b>Disabling LOB Buffering for a Temporary LOB.....</b>              | <b>11-211</b> |
| C (OCI): Disabling LOB Buffering.....                                | 11-213        |
| COBOL (Pro*COBOL): Disabling LOB Buffering for a Temporary LOB ..... | 11-215        |
| C/C++ (Pro*C/C++): Disabling LOB Buffering for a Temporary LOB ..... | 11-216        |

## 12 External LOBs (BFILEs)

|   |              |
|---|--------------|
| <b>Use Case Model: External LOBs (BFILEs) .....</b>                 | <b>12-2</b>  |
| <b>Accessing External LOBs (BFILEs) .....</b>                       | <b>12-4</b>  |
| <b>Directory Object.....</b>  | <b>12-4</b>  |
| Initializing a BFILE Locator.....                                   | 12-4         |
| How to Associate Operating System Files with Database Records ..... | 12-5         |
| BFILENAME() and Initialization.....                                 | 12-6         |
| DIRECTORY Name Specification.....                                   | 12-7         |
| <b>BFILE Security .....</b>   | <b>12-7</b>  |
| Ownership and Privileges .....                                      | 12-7         |
| Read Permission on Directory Object .....                           | 12-8         |
| SQL DDL for BFILE Security.....                                     | 12-8         |
| SQL DML for BFILE Security.....                                     | 12-9         |
| Catalog Views on Directories.....                                   | 12-9         |
| Guidelines for DIRECTORY Usage.....                                 | 12-9         |
| BFILEs in Shared Server (Multi-Threaded Server — MTS) Mode.....     | 12-10        |
| External LOB (BFILE) Locators.....                                  | 12-11        |
| <b>Creating a Table Containing One or More BFILE Columns.....</b>   | <b>12-13</b> |



|   |              |
|---|--------------|
| SQL: Creating a Table Containing One or More BFILE Columns.....                               | 12-14        |
| <b>Creating a Table of an Object Type with a BFILE Attribute .....</b>                        | <b>12-17</b> |
| SQL: Creating a Table of an Object Type with a BFILE Attribute.....                           | 12-18        |
| <b>Creating a Table with a Nested Table Containing a BFILE .....</b>                          | <b>12-20</b> |
| SQL: Creating a Table with a Nested Table Containing a BFILE .....                            | 12-21        |
| <b>INSERT a Row Using BFILENAME() .....</b>   | <b>12-22</b> |
| SQL: Inserting a Row by means of BFILENAME().....   | 12-25        |
| C (OCI): Inserting a Row by means of BFILENAME().....   | 12-25        |
| COBOL (Pro*COBOL): Inserting a Row by means of BFILENAME() .....                              | 12-26        |
| C/C++ (Pro*C/C++): Inserting a Row by means of BFILENAME() .....                              | 12-27        |
| Visual Basic (OO4O): Inserting a Row by means of BFILENAME() .....                            | 12-28        |
| Java (JDBC): Inserting a Row by means of BFILENAME() .....                                    | 12-29        |
| <b>INSERT a BFILE Row by Selecting a BFILE From Another Table.....</b>                        | <b>12-31</b> |
| SQL: Inserting a Row Containing a BFILE by Selecting a BFILE From Another Table .             | 12-32        |
| <b>Inserting a Row With BFILE by Initializing a BFILE Locator.....</b>                        | <b>12-33</b> |
| PL/SQL: Inserting a Row Containing a BFILE by Initializing a BFILE Locator .....              | 12-35        |
| C (OCI): Inserting a Row Containing a BFILE by Initializing a BFILE Locator .....             | 12-36        |
| C/C++ (Pro*C/C++): Inserting a Row Containing a BFILE by Initializing a<br>BFILE Locator..... | 12-38        |
| Java (JDBC): Inserting a Row Containing a BFILE by Initializing a BFILE Locator .....         | 12-39        |
| <b>Loading Data Into External LOB (BFILE) .....</b>   | <b>12-42</b> |
| Loading Data Into BFILES: File Name Only is Specified Dynamically.....                        | 12-44        |
| Loading Data into BFILES: File Name and DIRECTORY Object<br>Dynamically Specified .....       | 12-44        |
| <b>Loading a LOB with BFILE Data .....</b>  | <b>12-46</b> |
| PL/SQL (DBMS_LOB Package): Loading a LOB with BFILE Data .....                                | 12-49        |
| C (OCI): Loading a LOB with BFILE Data .....  | 12-49        |
| COBOL (Pro*COBOL): Loading a LOB with BFILE Data .....  | 12-51        |
| C/C++ (Pro*C/C++): Loading a LOB with BFILE Data.....   | 12-53        |
| Visual Basic (OO4O): Loading a LOB with BFILE Data .....                                      | 12-54        |
| <b>Loading a BLOB with BFILE Data.....</b>  | <b>12-55</b> |
| PL/SQL (DBMS_LOB Package): Loading an Internal Persistent BLOB with<br>BFILE Data.....        | 12-57        |
| <b>Loading a CLOB with BFILE Data .....</b>   | <b>12-59</b> |
| PL/SQL (DBMS_LOB Package): Loading a CLOB/NCLOB with BFILE Data.....                          | 12-61        |
| <b>Ways to Open a BFILE .....</b>   | <b>12-63</b> |

|   |              |
|---|--------------|
| Recommendation: Use OPEN to Open BFILE.....                                   | 12-64        |
| Specify the Maximum Number of Open BFILES: SESSION_MAX_OPEN_FILES .....       | 12-64        |
| <b>Opening a BFILE with FILEOPEN .....</b>                                    | <b>12-65</b> |
| PL/SQL: Opening a BFILE with FILEOPEN .....                                   | 12-67        |
| C (OCI): Opening a BFILE with FILEOPEN .....                                  | 12-67        |
| Visual Basic (OO4O): Opening a BFILE with FILEOPEN .....                      | 12-68        |
| Java (JDBC): Opening a BFILE with FILEOPEN .....                              | 12-68        |
| <b>Opening a BFILE with OPEN .....</b>  | <b>12-70</b> |
| PL/SQL: Opening a BFILE with OPEN .....                                       | 12-72        |
| C (OCI): Opening a BFILE with OPEN.....                                       | 12-72        |
| COBOL (Pro*COBOL): Opening a BFILE with OPEN .....                            | 12-73        |
| C/C++ (Pro*C/C++): Opening a BFILE with OPEN .....                            | 12-74        |
| Visual Basic (OO4O) Opening a BFILE with OPEN .....                           | 12-75        |
| Java (JDBC): Opening a BFILE with OPEN .....                                  | 12-76        |
| <b>Ways to See If a BFILE is Open .....</b>                                   | <b>12-78</b> |
| Recommendation: Use OPEN to Open BFILE.....                                   | 12-78        |
| Specify the Maximum Number of Open BFILES: SESSION_MAX_OPEN_FILES .....       | 12-78        |
| <b>Checking If the BFILE is Open with FILEISOPEN.....</b>                     | <b>12-80</b> |
| PL/SQL (DBMS_LOB Package): Checking If the BFILE is Open with FILEISOPEN..... | 12-81        |
| C (OCI): Checking If the BFILE is Open with FILEISOPEN.....                   | 12-82        |
| Visual Basic (OO4O): Checking If the BFILE is Open with FILEISOPEN .....      | 12-84        |
| Java (JDBC): Checking If the BFILE is Open with FILEISOPEN .....              | 12-84        |
| <b>Checking If a BFILE is Open Using ISOPEN.....</b>                          | <b>12-86</b> |
| PL/SQL (DBMS_LOB Package): Checking If the BFILE is Open with ISOPEN .....    | 12-88        |
| C (OCI): Checking If the BFILE is Open with ISOPEN .....                      | 12-88        |
| COBOL (Pro*COBOL): Checking If the BFILE is Open with ISOPEN .....            | 12-90        |
| C/C++ (Pro*C/C++): Checking If the BFILE is Open with ISOPEN.....             | 12-91        |
| Visual Basic (OO4O): Checking If the BFILE is Open with ISOPEN .....          | 12-92        |
| Java (JDBC): Checking If the BFILE is Open with ISOPEN .....                  | 12-93        |
| <b>Displaying BFILE Data .....</b>  | <b>12-96</b> |
| PL/SQL: Displaying BFILE Data.....  | 12-98        |
| C (OCI): Displaying BFILE Data.....   | 12-98        |
| COBOL (Pro*COBOL): Displaying BFILE Data .....                                | 12-101       |
| C/C++ (Pro*C/C++): Displaying BFILE Data .....                                | 12-103       |
| Visual Basic (OO4O): Displaying BFILE Data .....                              | 12-104       |

|   |        |
|---|--------|
| Java (JDBC): Displaying BFILE Data .....  | 12-105 |
| <b>Reading Data from a BFILE</b> .....  | 12-108 |
| PL/SQL (DBMS_LOB Package): Reading Data from a BFILE.....                         | 12-111 |
| C (OCI): Reading Data from a BFILE.....   | 12-111 |
| COBOL (Pro*COBOL): Reading Data from a BFILE .....                                | 12-113 |
| C/C++ (Pro*C/C++): Reading Data from a BFILE .....                                | 12-114 |
| Visual Basic (OO4O): Reading Data from a BFILE .....                              | 12-115 |
| Java (JDBC): Reading Data from a BFILE .....                                      | 12-116 |
| <b>Reading a Portion of BFILE Data (substr)</b> .....                             | 12-119 |
| PL/SQL (DBMS_LOB Package): Reading a Portion of BFILE Data (substr) .....         | 12-121 |
| COBOL (Pro*COBOL): Reading a Portion of BFILE Data (substr) .....                 | 12-121 |
| C/C++ (Pro*C/C++): Reading a Portion of BFILE Data (substr) .....                 | 12-123 |
| Visual Basic (OO4O): Reading a Portion of BFILE Data (substr) .....               | 12-124 |
| Java (JDBC): Reading a Portion of BFILE Data (substr) .....                       | 12-124 |
| <b>Comparing All or Parts of Two BFILES</b> .....                                 | 12-127 |
| PL/SQL (DBMS_LOB Package): Comparing All or Parts of Two BFILES.....              | 12-129 |
| COBOL (Pro*COBOL): Comparing All or Parts of Two BFILES .....                     | 12-130 |
| C/C++ (Pro*C/C++): Comparing All or Parts of Two BFILES .....                     | 12-132 |
| Visual Basic (OO4O): Comparing All or Parts of Two BFILES .....                   | 12-133 |
| Java (JDBC): Comparing All or Parts of Two BFILES .....                           | 12-134 |
| <b>Checking If a Pattern Exists (instr) in the BFILE</b> .....                    | 12-137 |
| PL/SQL (DBMS_LOB Package): Checking If a Pattern Exists (instr) in the BFILE..... | 12-139 |
| COBOL (Pro*COBOL): Checking If a Pattern Exists (instr) in the BFILE .....        | 12-139 |
| C/C++ (Pro*C/C++): Checking If a Pattern Exists (instr) in the BFILE .....        | 12-141 |
| Java (JDBC): Checking If a Pattern Exists (instr) in the BFILE .....              | 12-143 |
| <b>Checking If the BFILE Exists</b> .....   | 12-145 |
| PL/SQL (DBMS_LOB Package): Checking If the BFILE Exists .....                     | 12-147 |
| C (OCI): Checking If the BFILE Exists.....  | 12-147 |
| COBOL (Pro*COBOL): Checking If the BFILE Exists .....                             | 12-149 |
| C/C++ (Pro*C/C++): Checking If the BFILE Exists .....                             | 12-150 |
| Visual Basic (OO4O): Checking If the BFILE Exists .....                           | 12-151 |
| Java (JDBC): Checking If the BFILE Exists .....                                   | 12-152 |
| <b>Getting the Length of a BFILE</b> .....  | 12-154 |
| PL/SQL (DBMS_LOB Package): Getting the Length of a BFILE.....                     | 12-156 |
| C (OCI): Getting the Length of a BFILE .....                                      | 12-156 |

|  |        |
|--|--------|
| COBOL (Pro*COBOL): Getting the Length of a BFILE .....                               | 12-158 |
| C/C++ (Pro*C/C++): Getting the Length of a BFILE .....                               | 12-159 |
| Visual Basic (OO4O): Getting the Length of a BFILE .....                             | 12-160 |
| Java (JDBC): Getting the Length of a BFILE .....                                     | 12-161 |
| <b>Copying a LOB Locator for a BFILE</b> .....                                       | 12-164 |
| PL/SQL: Copying a LOB Locator for a BFILE .....                                      | 12-166 |
| C (OCI): Copying a LOB Locator for a BFILE.....                                      | 12-166 |
| COBOL (Pro*COBOL): Copying a LOB Locator for a BFILE .....                           | 12-168 |
| C/C++ (Pro*C/C++): Copying a LOB Locator for a BFILE .....                           | 12-169 |
| Java (JDBC): Copying a LOB Locator for a BFILE .....                                 | 12-170 |
| <b>Determining If a LOB Locator for a BFILE Is Initialized</b> .....                 | 12-172 |
| C (OCI): Determining If a LOB Locator for a BFILE Is Initialized.....                | 12-174 |
| C/C++ (Pro*C/C++): Determining If a LOB Locator for a BFILE Is Initialized .....     | 12-175 |
| <b>Determining If One LOB Locator for a BFILE Is Equal to Another</b> .....          | 12-177 |
| C (OCI): Determining If One LOB Locator for a BFILE Is Equal to Another .....        | 12-179 |
| C/C++ (Pro*C/C++): Determining If One BFILE LOB Locator Is Equal<br>to Another ..... | 12-181 |
| Java (JDBC): Determining If One LOB Locator for a BFILE Is Equal to Another .....    | 12-182 |
| <b>Getting DIRECTORY Alias and Filename</b> .....                                    | 12-184 |
| PL/SQL (DBMS_LOB Package): Getting Directory Alias and Filename .....                | 12-186 |
| C (OCI): Getting Directory Alias and Filename .....                                  | 12-186 |
| COBOL (Pro*COBOL): Getting Directory Alias and Filename .....                        | 12-188 |
| C/C++ (Pro*C/C++): Getting Directory Alias and Filename .....                        | 12-189 |
| Visual Basic (OO4O): Getting Directory Alias and Filename.....                       | 12-190 |
| Java (JDBC): Getting Directory Alias and Filename .....                              | 12-191 |
| <b>Updating a BFILE Using BFILENAME()</b> .....                                      | 12-193 |
| SQL: Updating a BFILE by means of BFILENAME() .....                                  | 12-195 |
| <b>Updating a BFILE by Selecting a BFILE From Another Table</b> .....                | 12-196 |
| SQL: Updating a BFILE by Selecting a BFILE From Another Table .....                  | 12-197 |
| <b>Updating a BFILE by Initializing a BFILE Locator</b> .....                        | 12-198 |
| PL/SQL: Updating a BFILE by Initializing a BFILE Locator .....                       | 12-200 |
| C (OCI): Updating a BFILE by Initializing a BFILE Locator .....                      | 12-201 |
| COBOL (Pro*COBOL): Updating a BFILE by Initializing a BFILE Locator .....            | 12-202 |
| C/C++ (Pro*C/C++): Updating a BFILE by Initializing a BFILE Locator .....            | 12-203 |
| Visual Basic (OO4O): Updating a BFILE by Initializing a BFILE Locator .....          | 12-204 |
| Java (JDBC): Updating a BFILE by Initializing a BFILE Locator .....                  | 12-206 |

|   |        |
|---|--------|
| <b>Closing a BFILE with FILECLOSE</b> .....                     | 12-208 |
| PL/SQL (DBMS_LOB Package): Closing a BFile with FILECLOSE ..... | 12-210 |
| C (OCI): Closing a BFile with FILECLOSE .....                   | 12-210 |
| Visual Basic (OO4O): Closing a Bfile with FILECLOSE.....        | 12-211 |
| Java (JDBC): Closing a BFile with FILECLOSE .....               | 12-211 |
| <b>Closing a BFILE with CLOSE</b> .....                         | 12-214 |
| PL/SQL (DBMS_LOB Package): Closing a BFile with CLOSE.....      | 12-216 |
| C (OCI): Closing a BFile with CLOSE.....                        | 12-216 |
| COBOL (Pro*COBOL): Closing a BFILE with CLOSE .....             | 12-217 |
| C/C++ (Pro*C/C++): Closing a Bfile with CLOSE .....             | 12-218 |
| Visual Basic (OO4O): Closing a BFile with CLOSE .....           | 12-219 |
| Java (JDBC): Closing a BFile with CLOSE .....                   | 12-220 |
| <b>Closing All Open BFILES with FILECLOSEALL</b> .....          | 12-222 |
| PL/SQL (DBMS_LOB Package): Closing All Open BFiles .....        | 12-224 |
| C (OCI): Closing All Open BFiles.....                           | 12-224 |
| COBOL (Pro*COBOL): Closing All Open BFiles .....                | 12-225 |
| C/C++ (Pro*C/C++): Closing All Open BFiles .....                | 12-227 |
| Visual Basic (OO4O): Closing All Open BFiles .....              | 12-228 |
| Java (JDBC): Closing All Open BFiles with FILECLOSEALL.....     | 12-228 |
| <b>Deleting the Row of a Table Containing a BFILE</b> .....     | 12-231 |
| SQL: Deleting a Row from a Table.....                           | 12-232 |

## 13 Using OraOLEDB to Manipulate LOBs

|  |      |
|--|------|
| <b>Introducing OLE DB</b> .....  | 13-2 |
| OraOLEDB: OLE DB and Oracle Large Object (LOB) Support .....           | 13-2 |
| Rowset Object.....   | 13-2 |
| <b>Manipulating LOBs Using ADO Recordsets and OLE DB Rowsets</b> ..... | 13-3 |
| ADO Recordsets and LOBs .....  | 13-3 |
| OLE DB Rowsets and LOBs .....  | 13-4 |
| <b>Manipulating LOBs Using OraOLEDB Commands</b> .....                 | 13-4 |
| <b>ADO and LOBs Example 1: Inserting LOB Data From a File</b> .....    | 13-4 |

## 14 LOBs Case Studies

|   |      |
|---|------|
| <b>Building a Multimedia Repository</b> ..... | 14-2 |
| How this Application Uses LOBs.....           | 14-4 |

|   |              |
|---|--------------|
| Populating the Repository.....  | 14-4         |
| Example 1: Inserting a Word document into a BLOB Column using PL/SQL..... | 14-5         |
| Searching the Repository.....   | 14-6         |
| How the Index Was Built on Table sam_emp, resume Column.....              | 14-7         |
| MyServletCtx Servlet.....   | 14-7         |
| Retrieving Data from the Repository.....                                  | 14-10        |
| Summary.....  | 14-12        |
| <b>Building a LOB-Based Web Site: First Steps .....</b>                   | <b>14-13</b> |

## **A Unified Modeling Language Diagrams**

|                        |     |
|------------------------|-----|
| Use Case Diagrams..... | A-2 |
| State Diagrams.....    | A-7 |

## **B The Multimedia Schema**

|  |     |
|--|-----|
| A Typical Multimedia Application.....          | B-2 |
| The Multimedia Schema.....                     | B-3 |
| Table Multimedia_Tab.....                      | B-5 |
| Script for Creating the Multimedia Schema..... | B-7 |

## **Index**

---

---

# Send Us Your Comments

## Oracle9i Application Developer's Guide - Large Objects (LOBs), Release 2 (9.2)

Part No. A96591-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: [infodev\\_us@oracle.com](mailto:infodev_us@oracle.com)
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:

Oracle Corporation  
Server Technologies Documentation  
500 Oracle Parkway, Mailstop 4op11  
Redwood Shores, CA 94065  
USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.





---

---

# Preface

This Guide describes Oracle9i application development features that deal with *Large Objects (LOBs)*. The information applies to all platforms, and does not include system-specific information.

This preface contains these topics:

- [Audience](#)
- [Organization](#)
- [Related Documents](#)
- [Conventions](#)
- [Documentation Accessibility](#)

## Feature Coverage and Availability

*Oracle9i* Application Developer's Guide - Large Objects (LOBs) contains information that describes the features and functionality of Oracle9i and Oracle9i Enterprise Edition products. Oracle9i and Oracle9i Enterprise Edition have the same basic features; however, several advanced features are available only with the Enterprise Edition, and some of these are optional. To use the Partitioning functionality, select the Partitioning option.

---

---

**Note:** From this release, in Oracle9i Enterprise Edition, you no longer need to select the Objects options to install the Objects and functionality.

---

---

## What You Need To Use LOBs

Although there are no special system restrictions when dealing with LOBs:

**See Also:** The following sections in [Chapter 4, "Managing LOBs"](#):

- ["LOB Restrictions"](#) on page 4-16
- ["Removed Restrictions"](#) on page -xlvi
- ["Restrictions for LOBs in Partitioned Index-Organized Tables"](#) on page 5-31

## Audience

The *Oracle9i 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.

## Organization

*Oracle9i Application Developer's Guide - Large Objects (LOBs)* contains thirteen chapters organized into two volumes. A brief summary of what you will find in each chapter follows:

### **Oracle9i Application Developer's Guide - Large Objects (LOBs) Volume 1**

#### **Chapter 1, "Introduction to LOBs"**

Chapter 1 describes the need for unstructured data and the advantages of using LOBs. It discusses the use of LOBs to promote internationalization by way of CLOBs, and the advantages of using LOBs over LONGs. Chapter 1 also describes the LOB demo file and where to find the supplied LOB sample scripts.

#### **Chapter 2, "Basic LOB Components"**

Chapter 2 describes the LOB datatype, including internal persistent and temporary LOBs and external LOBs, (BFILES). The need to initialize LOBs to NULL or Empty is described. The LOB locator and how to use it is also discussed.

### Chapter 3, "LOB Support in Different Programmatic Environments"

Chapter 3 describes the eight programmatic environments used to operate on LOBs and includes a listing of their available LOB-related methods or procedures:

- PL/SQL by means of the **DBMS\_LOB package** as described in *Oracle9i Supplied PL/SQL Packages and Types Reference*.
- C by means of **Oracle Call Interface (OCI)** described in the *Oracle Call Interface Programmer's Guide*
- C++ by means of **Oracle C++ Call Interface (OCCI)** described in the *Oracle C++ Call Interface Programmer's Guide*
- C/C++ by means of **Pro\*C/C++ precompiler** as described in the *Pro\*C/C++ Precompiler Programmer's Guide*
- COBOL by means of **Pro\*COBOL precompiler** as described in the *Pro\*COBOL Precompiler Programmer's Guide*
- Visual Basic by means of **Oracle Objects For OLE (OO4O)** as described in its accompanying online documentation.
- Java by means of the **JDBC Application Programmers Interface (API)** as described in the *Oracle9i JDBC Developer's Guide and Reference*.
- OLEDB by means of OraOLEDB, as described in the *Oracle Provider for OLE DB User's Guide* at [http://otn.oracle.com/tech/nt/ole\\_db](http://otn.oracle.com/tech/nt/ole_db)

### Chapter 4, "Managing LOBs"

Chapter 4 describes how to use SQL\*Loader, DBA actions required prior to working with LOBs, and LOB restrictions.

### Chapter 5, "Large Objects: Advanced Topics"

Chapter 5 covers advanced topics that touch on all the other chapters. Specifically, it focuses on read consistent locators, the LOB buffering subsystem, LOBs in the object cache, and using Partitioned Index-Organized Tables with LOBs.

### Chapter 6, "Frequently Asked Questions about LOBs"

Chapter 6 includes a list of LOB-related questions and answers received from users.

### Chapter 7, "Modeling and Design"

Chapter 7 covers issues related to selecting a datatype and includes a comparison of LONG and LONG RAW properties. Table architecture design criteria are discussed

and include tablespace and storage issues, reference versus copy semantics, index-organized tables, and partitioned tables. This chapter also describes using SQL semantics for LOBs, and indexing a LOB column.

### **Chapter 8, "Migrating From LONGs to LOBs"**

This chapter describes what you need to know when migrating from LONGs to LOBs using the LONG API for LOBs. This API ensures that when you change your LONG columns to LOBs, your existing applications will require few changes, if any.

### **Chapter 9, "LOBs: Best Practices"**

This chapter describes guidelines for using SQL\*Loader to load LOBs, as well as LOB and temporary LOB performance guidelines.

### **Chapter 10, "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 9. 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 programmatic environment.

## **Oracle9i Application Developer's Guide - Large Objects (LOBs) Volume 2**

### **Chapter 11, "Temporary LOBs"**

This chapter follows the same pattern as Chapter 10 but here focuses on temporary LOBs. New JDBC APIs in this release for Temporary LOBs include Creating a Temporary LOB, Checking if the BLOB/CLOB is temporary, and Freeing a Temporary BLOB/CLOB, comparing and trimming temporary LOBs. Visual Basic (OO4O) examples for temporary LOBs are not provided in this release but will be available in a future release.

### **Chapter 12, "External LOBs (BFILES)"**

This chapter focuses on external LOBs, also known as BFILES. The same treatment is provided here as in Chapters 10 and 11, namely, every operation is treated as a use case, and you will find matching code examples in the available programmatic environments.

### **Chapter 13, "Using OraOLEDB to Manipulate LOBs"**

This chapter describes how to manipulate LOBs using ADO Recordsets and OraOLEDB.

### **Chapter 14, "LOBs Case Studies"**

This chapter describes how to build a multimedia repository using LOBs. It also includes some first steps to consider when building a LOB based Web site.

### **Appendix A, "Unified Modeling Language Diagrams"**

This appendix explains how to use the Universal Modeling Language (UML) syntax used in the use case diagrams in Chapters 10, 11, and 12.

### **Appendix B, "The Multimedia Schema"**

This provides a sample multimedia case study and solution. It includes the design of the multimedia application architecture in the form of table `Multimedia_tab` and associated objects, types, and references.

## **Related Documents**

For more information, see the following manuals:

- *Oracle9i Supplied PL/SQL Packages and Types Reference*: Use this 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.
- *Oracle Call Interface Programmer's Guide*: Describes Oracle Call Interface (OCI). You can use OCI to build third-generation language (3GL) applications in C or C++ that access Oracle Server.
- *Oracle C++ Call Interface Programmer's Guide*
- *Pro\*C/C++ Precompiler Programmer's Guide*: Oracle Corporation also provides the Pro\* series of precompilers, which allow you to embed SQL and PL/SQL in your application programs.
- *Pro\*COBOL Precompiler Programmer's Guide*: The Pro\*COBOL precompiler allows you to embed SQL and PL/SQL in your COBOL programs for access to Oracle Server.
- *Programmer's Guide to the Oracle Precompilers* and *Pro\*Fortran Supplement to the Oracle Precompilers Guide*: Use these manuals for Fortran precompiler programming to access Oracle Server.

- *SQL\*Module for Ada Programmer's Guide*: This is a standalone manual for use when programming in Ada to access Oracle Server.
- **Java**: Oracle9i offers the opportunity of working with Java in the database. The Oracle Java documentation set includes the following:
  - *Oracle9i JDBC Developer's Guide and Reference*
  - *Oracle9i Java Developer's Guide*
  - *Oracle9i JPublisher User's Guide*
  - *Oracle9i Java Stored Procedures Developer's Guide*.

## **Multimedia**

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 *Oracle9i Data Cartridge Developer's Guide*
- To utilize Oracle's *interMedia* applications, refer to the following:
  - *Oracle interMedia User's Guide and Reference*.
  - *Oracle interMedia Java Classes User's Guide and Reference*
  - *Using Oracle8i with the Web*
  - *Oracle Text Reference*
  - *Oracle Text Application Developer's Guide*
  - *Oracle interMedia User's Guide and Reference*

## **Basic References**

- For SQL information, see the *Oracle9i SQL Reference* and *Oracle9i Database Administrator's Guide*
- For information about Oracle XML SQL with LOB data, refer to *Oracle9i Replication. LOBs*
- For basic Oracle concepts, see *Oracle9i Database Concepts*.
- Oracle9i Utilities

## How to Order this Manual

In North America, printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

<http://www.oraclebookshop.com/>

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

<http://otn.oracle.com/admin/account/membership.html>

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

<http://otn.oracle.com/docs/index.htm>

To access the database documentation search engine directly, please visit

<http://tahiti.oracle.com>

## Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- [Conventions in Text](#)
- [Conventions in Code Examples](#)

### Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

| Convention  | Meaning   | Example   |
|---|---|---|
| <b>Bold</b>   | Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.   | When you specify this clause, you create an <b>index-organized table</b> .  |
| <i>Italics</i>  | Italic typeface indicates book titles or emphasis.  | <i>Oracle9i Database Concepts</i><br>Ensure that the recovery catalog and target database do <i>not</i> reside on the same disk.  |
| UPPERCASE<br>monospace<br>(fixed-width<br>font)               | Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.   | You can specify this clause only for a NUMBER column.<br>You can back up the database by using the BACKUP command.<br>Query the TABLE_NAME column in the USER_TABLES data dictionary view.<br>Use the DBMS_STATS.GENERATE_STATS procedure.  |
| lowercase<br>monospace<br>(fixed-width<br>font)               | Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values.<br><b>Note:</b> Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown. | Enter sqlplus to open SQL*Plus.<br>The password is specified in the orapwd file.<br>Back up the datafiles and control files in the /disk1/oracle/dbs directory.<br>The department_id, department_name, and location_id columns are in the hr.departments table.<br>Set the QUERY_REWRITE_ENABLED initialization parameter to true.<br>Connect as oe user.<br>The JRepUtil class implements these methods. |
| lowercase<br>monospace<br>(fixed-width<br>font) <i>italic</i> | Lowercase monospace italic font represents placeholders or variables.   | You can specify the <i>parallel_clause</i> .<br>Run <i>Uold_release</i> .SQL where <i>old_release</i> refers to the release you installed prior to upgrading.   |

## Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL\*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```



The following table describes typographic conventions used in code examples and provides examples of their use.

| Convention     | Meaning  | Example   |
|----------------|--|---|
| [ ]            | Brackets enclose one or more optional items. Do not enter the brackets.  | DECIMAL ( <i>digits</i> [ , <i>precision</i> ])   |
| { }            | Braces enclose two or more items, one of which is required. Do not enter the braces.   | {ENABLE   DISABLE}  |
|                | A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.  | {ENABLE   DISABLE}<br>[COMPRESS   NOCOMPRESS]   |
| ...            | Horizontal ellipsis points indicate either: <ul style="list-style-type: none"> <li>■ That we have omitted parts of the code that are not directly related to the example</li> <li>■ That you can repeat a portion of the code</li> </ul> | CREATE TABLE ... AS <i>subquery</i> ;<br><br>SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> FROM employees;   |
| .              | Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.  | SQL> SELECT NAME FROM V\$DATAFILE;<br>NAME<br>-----<br>/fsl/dbs/tbs_01.dbf<br>/fsl/dbs/tbs_02.dbf<br>.<br>.<br>.<br>/fsl/dbs/tbs_09.dbf<br>9 rows selected. |
| Other notation | You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.   | acctbal NUMBER(11,2);<br>acct CONSTANT NUMBER(4) := 3;  |
| <i>Italics</i> | Italicized text indicates placeholders or variables for which you must supply particular values.   | CONNECT SYSTEM/ <i>system_password</i><br>DB_NAME = <i>database_name</i>  |

| Convention | Meaning  | Example   |
|------------|--|---|
| UPPERCASE  | Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase. | <pre>SELECT last_name, employee_id FROM employees;  SELECT * FROM USER_TABLES;  DROP TABLE hr.employees;</pre>    |
| lowercase  | <p>Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.</p> <p><b>Note:</b> Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.</p>  | <pre>SELECT last_name, employee_id FROM employees;  sqlplus hr/hr  CREATE USER mjones IDENTIFIED BY ty3MU9;</pre> |

## Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

**Accessibility of Code Examples in Documentation** JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

**Accessibility of Links to External Web Sites in Documentation** This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither

evaluates nor makes any representations regarding the accessibility of these Web sites.



---

---

# What's New with Large Objects (LOBs)?

This section describes the new features in the following releases:

- [LOB Features Introduced with Oracle9i, Release 2 \(9.2\)](#)
- [LOB Features Introduced with Oracle9i, Release 1 \(9.0.1\)](#)
- [LOB Features Introduced with Oracle8i Release 2 \(8.1.6\)](#)
- [LOB Features Introduced with Oracle8i, Release 8.1.5](#)

## LOB Features Introduced with Oracle9i, Release 2 (9.2)

This release introduces new PL/SQL APIs with improved features for loading binary and character data from LOBs:

- **DBMS\_LOB.LOADBLOBFROMFILE**

This API allows you to load binary large objects from operating system files into internal persistent LOBs, temporary LOBs, and external LOBs (BFILES).

**See Also:**

- ["Loading an Internal Persistent BLOB with Binary Data from a BFILE" on page 10-43](#)
- ["Loading a Temporary BLOB with Binary Data from a BFILE" on page 11-48](#)
- ["Loading a BLOB with BFILE Data" on page 12-55](#)

- **DBMS\_LOB.LOADCLOBFROMFILE**

This API allows you to load character large objects from operating system files into internal persistent LOBs, temporary LOBs, and external LOBs (BFILES). This API performs the proper character set conversions from the BFILE data character set to the destination CLOB/NCLOB character set.

**See Also:**

- ["Loading an Internal Persistent CLOB with BFILE Data"](#) on page 10-47
- ["Loading a Temporary CLOB/NCLOB with Character Data from a File"](#) on page 11-52
- ["Loading a CLOB with BFILE Data"](#) on page 12-59

## **Removed Restrictions**

The following restrictions are removed in Oracle9i Release 2.

- **Trigger restriction removed**

There is now DML BEFORE ROW Trigger :new support for LOBs. This means that triggers on LOBs follow the same rules as triggers on any other type of column.

- **Locally managed tablespaces restriction removed**

You can now create LOB columns in locally managed tablespaces.

- **LOBs in AUTO segment-managed tablespaces restriction removed**

You can now store LOBs in AUTO segment-managed tablespaces.

- **NCLOB parameters**

NCLOB parameters are now allowed as attributes in object types.

- **Partitioned Index Organized Tables**

Partitioned Index Organized Tables (PIOT) are now supported.

- **Client-side PL/SQL DBMS\_LOB procedures**

Client-side PL/SQL DBMS\_LOB procedures are now supported.

- **PL/SQL trigger body**

Prior to Release 9.2, in a PL/SQL trigger body of an `BEFORE ROW DML` trigger, you could read the `:old` value of the LOB, but you could not read the `:new` value.

- **Trigger support**

In releases prior to 9.2, if a view with a LOB column has an `INSTEAD OF TRIGGER`, then you cannot specify a string `INSERT/UPDATE` into the LOB column. This restriction is removed in release 9.2. For example:

```
CREATE TABLE t(a LONG);
CREATE VIEW v AS SELECT * FROM t;
CREATE TRIGGER trig INSTEAD OF insert on v...;
ALTER TABLE t MODIFY (a CLOB);
INSERT INTO v VALUES ('abc');          /* works now */
```

## LOB Features Introduced with Oracle9i, Release 1 (9.0.1)

The following sections describe the new features in Oracle9i Large Objects (LOBs):

- **LONG-to-LOB Migration API**

To assist you in migrating to LOBs, Oracle now supports the LONG API for LOBs. This API ensures that when you change your LONG columns to LOBs, your existing applications will require few changes, if any. When possible, change your existing applications to use LOBs instead of LONGs because of the added benefits that LOBs provide.

**See Also:** [Chapter 8, "Migrating From LONGs to LOBs"](#)

- **Using SQL Semantics with LOBs**

In this release, for the first time, you can access (internal persistent) LOBs using SQL `VARCHAR2` semantics, such as SQL string operators and functions. By providing you with an SQL interface, which you are familiar with, accessing LOB data can be greatly facilitated. These semantics are recommended when using small-sized LOBs (~ 10-100KB).

**See Also:** [Chapter 7, "Modeling and Design"](#)

- **Using Oracle C++ Call Interface (OCI) with LOBs**

Oracle C++ Call Interface (OCI) is a new C++ API for manipulating data in an Oracle database. OCI is organized as an easy-to-use collection of C++ classes which enable a C++ program to connect to a database, execute SQL statements, insert/update values in database tables, retrieve results of a query, execute stored procedures in the database, and access metadata of database schema objects. OCI API provides advantages over JDBC and ODBC.

**See Also:**

- [Chapter 3, "LOB Support in Different Programmatic Environments"](#)
- [Chapter 10, "Internal Persistent LOBs"](#)

- **New JDBC LOB Functionality**

The following are new JDBC LOB-related functionality:

- Temporary LOB APIs: create temporary LOBs and destroy temporary LOBs
- Trim APIs: trim the LOBs to the specified length
- Open and Close APIs: open and close LOBs explicitly
- New Streaming APIs: read and write LOBs as Java streams from the specified offset.
- Empty LOB instances can now be created with JDBC. The instances do not require database round trips

**See Also:**

- [Chapter 3, "LOB Support in Different Programmatic Environments"](#)
- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 11, "Temporary LOBs"](#)
- [Chapter 12, "External LOBs \(BFILES\)"](#)

- **Support for LOBs in Partitioned Index-Organized Tables**

Oracle9i introduces support for LOB, VARRAY columns stored as LOBs, and BFILES in partitioned index-organized tables. The behavior of LOB columns in



these tables is similar to that of LOB columns in conventional (heap-organized) partitioned tables, except for a few minor differences.

**See Also:** [Chapter 5, "Large Objects: Advanced Topics"](#)

- **Using OLEDB and LOBs (new to this manual)**

OLE DB is an open specification for accessing various types of data from different stores in a uniform way. OLEDB supports the following functions for these LOB types:

- **Persistent LOBs:** READ/WRITE through the rowset.
- **BFILEs:** READ-ONLY through the rowset.

**See Also:** [Chapter 13, "Using OraOLEDB to Manipulate LOBs"](#)

## LOB Features Introduced with Oracle8i Release 2 (8.1.6)

---

---

**Note:** There was no change in LOB functionality between Oracle8i Release 2 (8.1.6) and Oracle8i Release 3 (8.1.7).

---

---

New LOB features introduced in Oracle8i, Release 2 (8.1.6) were:

- **CACHE READS**

A CACHE READS option for LOB columns.

- **4,000 byte restriction removed**

The 4,000 byte restriction for bind variables binding to an internal LOB was removed.

- **Binding More Than 4,000 Bytes of Data**

From Oracle8i Release 2 (8.1.6) and higher, Oracle supports binding more than 4,000 bytes of data to internal LOB columns in INSERT and UPDATE statements.

If a table has LONG and LOB columns, you can bind more than 4,000 bytes of data for either the LONG column or the LOB columns, but not both in the same statement.

## LOB Features Introduced with Oracle8i, Release 8.1.5

New LOB features included in the Oracle8i, Release 8.1.5 are:

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

---

# Introduction to LOBs

This chapter discusses the following topics:

- Why Use LOBs?
- Why Not Use LONGs?
- LONG-to-LOB Migration API
- SQL Semantics Support for LOBs
- Partitioned Index-Organized Tables and LOBs
- Extensible Indexing on LOBs
- Function-Based Indexing on LOBs
- XML Documents Can be Stored in XMLType Columns as CLOBs
- Compatibility and Migration Issues
- Examples in This Guide

## Why Use LOBs?

As applications evolve to encompass increasingly richer semantics, they encounter the need to deal with the following kinds of data:

- Simple structured data
- Complex structured data
- Semi-structured data
- Unstructured data

Traditionally, the Relational model has been very successful at dealing with simple structured data -- the kind which can be fit into simple tables. Oracle has added Object-Relational features so that applications can deal with complex structured data -- collections, references, user-defined types and so on. Our queuing technologies, such as Advanced Queueing, deal with Messages and other semi-structured data.

LOBs are designed to support the last kind of data — *unstructured data*.

## Unstructured Data

### **Unstructured Data Cannot be Decomposed Into Standard Components**

Unstructured data cannot be decomposed into standard components. Data about an Employee can be 'structured' into a Name (probably a character string), an identification (likely a number), a Salary and so on. But if you are given a Photo, you find that the data really consists of a long stream of 0s and 1s. These 0s and 1s are used to switch pixels on or off so that you will see the Photo on a display, but they can't be broken down into any finer structure in terms of database storage.

### **Unstructured Data is Large**

Also interesting is that unstructured data such as text, graphic images, still video clips, full motion video, and sound waveforms tend to be *large* -- a typical employee record may be a few hundred bytes, but even small amounts of multimedia data can be thousands of times larger.

### **Unstructured Data in System Files Need Accessing from the Database**

Finally, some multimedia data may reside on operating system files, and it is desirable to access them from the database.

## LOB Datatype Helps Support Internet Applications

With the growth of the internet and content-rich applications, it has become imperative that the database support a datatype that fulfills the following:

- Can store unstructured data
- Is optimized for large amounts of such data
- Provides a uniform way of accessing large unstructured data within the database or outside

### Two Type of LOBs Supported

Oracle supports the following two types of LOBs

- Those stored in the database either in-line in the table or in a separate segment or tablespace, such as BLOB, CLOB, and NCLOB.
- Those stored as operating system files, such as BFILEs.

## Using XML, LOBs, and Oracle Text (interMedia Text)

### Use CLOBs or BFILEs to Store Unstructured Data

CLOBs can store large amounts of character data and are useful for storing unstructured XML documents. Also useful for storing multimedia data, BFILEs which are external file references can also be used. In this case the XML is stored and managed outside the RDBMS, but can be used in queries on the server.

### Oracle Text (interMedia Text) Indexing Supports Searching Content in XML Elements

You can create Oracle Text (*interMedia Text*) indexes on CLOB columns and perform queries on XML.

#### See Also:

- [Oracle9i XML Developer's Kits Guide - XDK](#)
- [Oracle Text Reference](#)
- [Oracle Text Application Developer's Guide](#)

### **LOBs Enable Oracle Text (*interMEDIA* Text)**

While LOBs provide the infrastructure in the database to store multimedia data, Oracle8i and Oracle9i also provide developers with additional functionality for the most commonly used multimedia types. The multimedia types include text, image, locator, audio, and video data.

Oracle8i introduced the *interMedia* bundle, that supports text data, spatial location, images, audio, and video data. You can access *interMedia* objects using SQL queries, manipulate their contents (such as, trim an image), read and write their content, and convert data from one format to another.

*interMedia* in turn uses Oracle's infrastructure to define object types, methods, and LOBs necessary to represent these specialized types of data in the database. Oracle *interMedia* provide a *predefined set of objects and operations* that facilitate application development.

See also <http://otn.oracle.com/products/text>

## **Why Not Use LONGs?**

In Oracle7, most applications storing large amounts of unstructured data used the LONG or LONG RAW data type.

Oracle8i and Oracle9i's support for LOB data types is preferred over support for LONG and LONG RAWs in Oracle7 in the following ways:

- *LOB Capacity:* With Oracle8 and Oracle8i, LOBs can store up to 4GB of data. This doubles the 2GB of data that LONG and LONG RAW data types could store.
- *Number of LOB columns in a table:* An Oracle8, Oracle8i, or Oracle9i table can have multiple LOB columns. Each LOB column in the same table can be of a different type. In Oracle7 Release 7.3 and higher, tables are limited to a single LONG or LONG RAW column.
- *Random piece-wise access:* LOBs support random access to data, but LONGs support only sequential access.

### **LOB Columns**

---

---

**Note:** LOBs can also be object attributes.

---

---

LOB (BLOB, CLOB, NCLOB, or BFILE) column types store values or references, called locators. Locators specify the location of large objects.

**LOB Columns Do Not Only Store Locators!** In LOB columns, the LOB locator is stored in-line in the row. Depending on the user-specified SQL data definition language (DDL) storage parameters, Oracle9i can store small LOBs, less than approximately 4K in-line *in the table*. Once the LOB grows bigger than approximately 4K Oracle9i moves the LOB out of the table into a different segment and possibly even into a different tablespace. Hence, Oracle9i sometimes stores LOB data, not just LOB locators, in-line in the row.

BLOB, CLOB, and NCLOB data is stored out-of-line inside the database. BFILE data is stored in operating system files outside the database. Oracle9i provides programmatic interfaces and PL/SQL support for access to and operation on LOBs.

## LONG-to-LOB Migration API

Oracle9i supports LONG as well as LOB datatypes. When possible, change your existing applications to use LOBs instead of LONGs because of the added benefits that LOBs provide.

LONG-to-LOB migration allows you to easily migrate your existing applications that access LONG columns, to use LOB columns. The migration has two parts:

- Data migration
- Application migration

### See Also:

- See [Chapter 7, "Modeling and Design", "LOBs Compared to LONG and LONG RAW Types"](#) on page 7-2.
- [Chapter 8, "Migrating From LONGs to LOBs"](#)

## SQL Semantics Support for LOBs

In this release, for the first time, you can access LOBs using SQL VARCHAR2 semantics, such as SQL string operators and functions.

By providing you with an SQL interface, which you are familiar with, accessing LOB data can be greatly facilitated. You can benefit from this added functionality in the following two cases:

- When using small-sized LOBs (~ 10-100K) to store data and you need to access the LOB data in SQL queries, the syntax is the same as that of VARCHAR2's.
- When you have just migrated your LONG columns to LOBs. In this release, you can take advantage of an easier migration process using the LONG-to-LOB migration API described in [Chapter 8, "Migrating From LONGs to LOBs"](#).

**See Also:** [Chapter 7, "Modeling and Design"](#), ["SQL Semantics Support for LOBs"](#).

## Partitioned Index-Organized Tables and LOBs

Oracle9i introduces support for LOB, VARRAY columns stored as LOBs, and BFILES in partitioned index-organized tables. The behavior of LOB columns in these tables is similar to that of LOB columns in conventional (heap-organized) partitioned tables, except for the following differences:

- Tablespace mapping
- Inline as opposed to out-of-line LOBs

LOB columns are supported only in range partitioned index-organized tables.

**See Also:** [Chapter 5, "Large Objects: Advanced Topics"](#), ["LOBs in Partitioned Index-Organized Tables"](#).

## Extensible Indexing on LOBs

Oracle provides an extensible server which provides 'extensible indexing'. This allows you to define new index types as required. This is based on the concept of cooperative indexing where a data cartridge and Oracle9i build and maintain indexes for data types such as text and spatial for example, for On-line-Analytical Processing (OLAP).

The cartridge is responsible for defining the index structure, maintaining the index content during load and update operations, and searching the index during query processing. The index structure can be stored in Oracle as heap-organized, or an index-organized table, or externally as an operating system file.

To this end, Oracle introduces the concept of an *indextype*. The purpose of an indextype is to enable efficient search and retrieval functions for complex domains such as text, spatial, image, and OLAP by means of a data cartridge. An indextype is analogous to the sorted or bit-mapped index types that are built-in within the Oracle Server. The difference is that an indextype is implemented by the data



cartridge developer, whereas the Oracle kernel implements built-in indexes. Once a new indextype has been implemented by a data cartridge developer, end users of the data cartridge can use it just as they would built-in indextypes.

When the database system handles the physical storage of domain indexes, data cartridges

- Define the format and content of an index. This enables cartridges to define an index structure that can accommodate a complex data object.
- Build, delete, and update a domain index. The cartridge handles building and maintaining the index structures. Note that this is a significant departure from the medicine indexing features provided for simple SQL data types. Also, since an index is modeled as a collection of tuples, in-place updating is directly supported.
- Access and interpret the content of an index. This capability enables the data cartridge to become an integral component of query processing. That is, the content-related clauses for database queries are handled by the data cartridge.

By supporting extensible indexes, Oracle9i significantly reduces the effort needed to develop high-performance solutions that access complex datatypes such as LOBs.

### **Extensible Optimizer**

The extensible optimizer functionality allows authors of user-defined functions and indexes to create statistics collection, selectivity, and cost functions. This information is used by the optimizer in choosing a query plan. The cost-based optimizer is thus extended to use the user-supplied information; the rule-based optimizer is unchanged.

Extensible indexing functionality allows you to define new operators, index types, and domain indexes. For such user-defined operators and domain indexes, the extensible optimizer functionality will allow users to control the three main components used by the optimizer to select an execution plan: *statistics*, *selectivity*, and *cost*.

**See Also:** *Oracle9i Data Cartridge Developer's Guide*

## **Function-Based Indexing on LOBs**

A function-based index is an index built on an expression. It extends your indexing capabilities beyond indexing on a column. A function-based index increases the variety of ways in which you can access data.

Function-based indexes cannot currently be built on nested tables. However, you can now build function-based indexes on LOB columns and varrays.

**See Also:** *Oracle9i Application Developer's Guide - Fundamentals* for more information about using function-based indexing.

## XML Documents Can be Stored in XMLType Columns as CLOBs

Composed XML documents can be stored in CLOBs. XMLType columns use CLOBs for storage.

**See Also:** *Oracle9i XML Developer's Kits Guide - XDK*, Chapter 5, for information about XMLType, and how XML is stored in LOBs.

## Compatibility and Migration Issues

The following LOB related compatibility and migration issues are described in detail in *Oracle9i Database Migration* :

- *"Varying Width Character Sets for CLOBs or NCLOBs"*, under "Datatypes, Compatibility and Interpretability Issues".
- *Downgrading with CACHE READs Defined:* See "Downgrading to an Older Version 8 Release", under "Remove Incompatibilities", "Datatypes", "Discontinue Use of Cache Reads Specified for LOBs".
- *Downgrading — Removing LOB Columns from Partitioned Table:* See the chapter, "Downgrading to an Older Version 8 Release", under "Remove Incompatibilities", "Datatypes", "Remove LOB Columns from Partitioned Tables".
- *Downgrading — LOBs and Varrays in Index Organized Tables:* See "Downgrading to an Older Version 8 Release", under "Remove Incompatibilities", "Schema Objects", "Discontinue Use of LOBs and Varrays in Index Organized Tables".
- *Downgrading — Varying Width Character Sets for CLOBs or NCLOBs:* See the chapter, "Downgrading to an Older Version 8 Release", under "Remove Incompatibilities", under "Datatypes", "Remove CLOBs and NCLOBs from Tables in Database with Varying-Width Character Set".
- *Downgrading — Partioned Index Organized Tables (PIOTs) (on LOBs):* See the chapter, "Removing Oracle9i Incompatibilities".

- *Downgrading — Functional Indexes on LOBs*: See the chapter, "Removing Oracle9i Incompatibilities".
- *Downgrading — LONG to LOB data and application migration*: See the chapter, "Removing Oracle9i Incompatibilities".

## Examples in This Guide

Examples in this guide use the following sample schemas:

- Product Media (PM) sample schema
- Multimedia sample schema - *deprecated*

The Multimedia schema is deprecated and is not supplied with Oracle9i sample schemas. The Product Media schema replaces the Multimedia schema as the sample schema for most LOB examples.

Examples in [Chapter 10](#), [Chapter 11](#), and [Chapter 12](#) of this guide have been migrated to the PM schema. Most other examples have not been migrated and still use the Multimedia schema. [Appendix B, "The Multimedia Schema"](#) is included in this guide to provide a context for these examples.

The Product Media sample schema provides tables and sample data that are suited for illustrating LOB database features. Much of this data is unstructured as LOB APIs are designed to handle large unstructured data. "[Unstructured Data](#)" is described earlier in this chapter.

The examples in this guide that are based on the PM schema use the `Print_media` sample table.

**See Also:** For further details on the Product Media sample schema, you must refer to *Oracle9i Sample Schemas*.



---

# Basic LOB Components

This chapter discusses the following topics:

- [The LOB Datatype](#)
- [Varying-Width Character Data](#)
- [LOB Value and Locators](#)
- [Creating Tables that Contain LOBs](#)

## The LOB Datatype

Oracle9i 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

*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<sup>1</sup> 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 Oracle9i database.
- **NCLOB**, a LOB whose value is composed of character data that corresponds to the national character set defined for the Oracle9i database.

Internal LOBs are divided into **persistent** and **temporary** LOBs.

### External LOBs (BFILEs)

*External* LOBs (BFILEs) are large binary data objects stored in operating system files outside 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-ROMs, PhotoCDs and DVDs.

The BFILE *datatype* allows *read-only* byte stream access to large files on the file system of the database server.

---

<sup>1</sup> ACID = Access Control Information Directory. This is the attribute that determines who has what type of access and to what directory data. It contains a set of rules for structural and content access items. For more information see the Oracle Internet Directory Administrators Guide.

Oracle can access BFILES provided the underlying server operating system supports 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 Datatypes

There is one datatype, BFILE, for declaring instances of external SQL LOBs.

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

### Internal LOBs Use Copy Semantics, External LOBs Use Reference Semantics

- Copy semantics: Both LOB locator and value are copied
- Reference semantics: Only LOB locator is copied

#### Copy Semantics

Internal LOBs, namely BLOBs, CLOBs, NCLOBs, whether persistent or temporary, use *copy semantics*.

When you insert or update a LOB with a LOB from another row in the same table, the LOB value is copied so that each row has a *different* copy of the LOB value.

Internal LOBs have *copy semantics* so that if the LOB in the row of the table is copied to another LOB, in a different row or perhaps in the same row but in a different column, then the actual LOB *value* is copied, not just the LOB *locator*. This means in this case that there will be two different LOB locators and two copies of the LOB value.

#### Reference Semantics

External LOBs (BFILES) use *reference semantics*. When the BFILE in the row of the table is copied to another BFILE, only the BFILE *locator* is copied, not the actual BFILE data, that is, not the actual operating system file.

## Varying-Width Character Data

- You can create the following LOB tables:
  - Containing CLOB/NCLOB columns even if you use a varying-width CHAR/NCHAR database character set
  - Containing a type that has a CLOB attribute irrespective of whether you use a varying-width CHAR database character set
- You cannot create the following tables:
  - With NCLOBs as attributes of object types

## Using DBMS\_LOB.LOADFROMFILE and Functions that Access OCI

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.

However, we recommend that you use the sql\*loader instead of loadfromfile to load data into a clob/nclob because the sql\*loader will take care of all necessary character set conversions.

### See Also:

- ["Loading an Internal Persistent CLOB with BFILE Data" on page 10-47](#)
- ["Loading a Temporary CLOB/NCLOB with Character Data from a File" on page 11-52](#)
- ["Loading a CLOB with BFILE Data" on page 12-59](#)

## Converting Between Client Character Set and UCS-2

There are APIs in cartridge service that can convert between client character set and UCS-2:

- OCIUnicodeToCharSet()
- OCICharSetToUnicode()



**See Also:**

- [Chapter 3, "LOB Support in Different Programmatic Environments"](#):
- ["Offset and Amount Parameters: Fixed-Width Versus Varying-Width, Character or Byte For DBMS\\_LOB Package"](#) on page 3-7
- ["Offset and Amount Parameters: Fixed-Width Versus Varying-Width, Character or Byte"](#) on page 3-11

## 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 you do not set `DISABLE STORAGE IN ROW` and the internal LOB value is less than approximately 4,000 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 in [Chapter 7, "Modeling and Design"](#), ["ENABLE | DISABLE STORAGE IN ROW"](#) on page 7-11, the LOB value is automatically moved out of the row once it extends beyond approximately 4,000 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 also a distinct copy of the LOB value stored in the database tablespace.
- **External LOB Locators.** For external LOBs (BFILES), the LOB column stores a BFILE locator to the external operating system file. Each BFILE column/attribute for a given row has its own BFILE locator. However, two

different rows can contain a BFILE locator that points to the same operating system file.

## Setting the LOB Column/Attribute to Contain a Locator

### Internal LOBs

Before you can start writing data to an internal LOB using the supported programmatic environment interfaces<sup>1</sup> (PL/SQL, OCI, OCCI, Pro\*C/C++, Pro\*COBOL, Visual Basic, Java, or OLEDB), the LOB column/attribute must be made non-null, that is, it must contain a locator. 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.

**See Also:** ["Inserting a LOB Value using EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)" in Chapter 10, "Internal Persistent LOBs"](#).

### External LOBs

Before you can start accessing the external LOB (BFILE) value using the supported programmatic environment interfaces, the BFILE column/attribute must be made non-null. You can initialize the BFILE column to point to an external operating system file by using the `BFILENAME()` function.

**See Also:** ["INSERT a Row Using BFILENAME\(\)" in Chapter 12, "External LOBs \(BFILES\)"](#).

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',

---

<sup>1</sup> Note: You could use SQL to populate a LOB column with data even if it contained NULL, for example, unless its a LOB attribute. However, you *cannot* use the supported programmatic environment interfaces on a NULL LOB!

- 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 statement in *Oracle9i Database Reference*).

---



---

**Note:** Character strings are inserted using the default character set for the instance.

---



---

See [Appendix B, "The Multimedia Schema"](#), for the definition of table `Multimedia_tab`.

```
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 in the following.

```
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));
```

---



---

**Note:** You cannot initialize a LOB object attribute with a literal.

---



---

## 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 CLOB;
    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. The OCI LOB interface is described [Chapter 3, "LOB Support in Different Programmatic Environments"](#) and in the *Oracle Call Interface Programmer's Guide*.

Using LOB locators and transaction boundaries, and read consistent locators are described in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Creating Tables that Contain LOBs

When creating tables that contain LOBs use the guidelines described in the following sections:

- [Initializing Internal LOBs to NULL or Empty](#)
- [Initializing Internal LOB Columns to a Value](#)
- [Initializing External LOBs to NULL or a File Name](#)
- Defining tablespace and storage characteristics. See [Chapter 7, "Modeling and Design"](#), ["Defining Tablespace and Storage Characteristics for Internal 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:

- *Setting an Internal LOB to NULL:* A LOB set to NULL has no locator. A NULL value is stored in the row in the table, not a locator. This is the same process as for all other datatypes.
- *Setting an Internal LOB to Empty:* 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 or attribute, you get back a locator which you can use to populate the LOB with data using supported programmatic environments, such as OCI or PL/SQL (DBMS\_LOB). See [Chapter 3, "LOB Support in Different Programmatic Environments"](#).

These options are discussed in more detail in the following.

As discussed in the following, an external LOB (BFILE) can be initialized to NULL or to a filename.

### Setting an Internal 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 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.

Note that you cannot call OCI or DBMS\_LOB functions on a NULL LOB, so you must then issue an SQL UPDATE statement to reset the null LOB column — to EMPTY\_BLOB() or EMPTY\_CLOB() or to a value (for example, 'Denzel Washington') for internal LOBs, or to a filename for external LOBs.

The point is that you cannot make a function call from the supported programmatic environments 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, you can set the LOB value to empty using the function EMPTY\_BLOB() or 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;
```

## Initializing LOBs Example Using Table `Multimedia_tab`

You can initialize the LOBs in `Multimedia_tab` by using the following `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`.

## Initializing Internal LOB Columns to a Value

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.

Note that you can initialize the LOB column to a value that contains more than 4K data.

**See Also:** [Chapter 7, "Modeling and Design"](#)

## Initializing External LOBs to `NULL` or a File Name

An external LOB (BFILE) can be initialized to `NULL` or to a filename using the `BFILENAME()` function.

**See Also:** [Chapter 12, "External LOBs \(BFILES\)", "Directory Object"](#) — ["Initializing a BFILE Locator"](#).

---

# LOB Support in Different Programmatic Environments

This chapter discusses the following topics:

- [Eight Programmatic Environments Operate on LOBs](#)
- [Comparing the LOB Interfaces](#)
- [Using PL/SQL \(DBMS\\_LOB Package\) To Work With LOBs](#)
- [Using C \(OCI\) To Work With LOBs](#)
- [Using C++ \(OCCI\) To Work With LOBs](#)
- [Using C/C++ \(Pro\\*C\) To Work with LOBs](#)
- [Using COBOL \(Pro\\*COBOL\) to Work with LOBs](#)
- [Using Visual Basic \(Oracle Objects for OLE \(OO4O\)\) to Work with LOBs](#)
- [Using Java \(JDBC\) To Work with LOBs](#)
- [Using Java \(JDBC\) To Work with LOBs](#)
- [OLEDB \(Oracle Provider for OLEDB — OraOLEDB\)](#)

## Eight Programmatic Environments Operate on LOBs

**Table 3–1** lists the eight programmatic environments (languages) that support LOB functionality. **Chapter 10**, **Chapter 11**, and **Chapter 12** cover the supported LOB functions in terms of use cases. Examples are provided in each programmatic environment for most LOB use cases.

**Table 3–1** *LOBs' Eight Programmatic Environments*

| Language     | Precompiler or Interface Program              | Syntax Reference  | In This Chapter See...  |
|--------------|---|---|---|
| PL/SQL       | DBMS_LOB Package                              | <i>Oracle9i Supplied PL/SQL Packages and Types Reference</i>  | "Using PL/SQL (DBMS_LOB Package) To Work With LOBs" on page 3-6.                      |
| C            | Oracle Call Interface for C (OCI)             | <i>Oracle Call Interface Programmer's Guide</i>   | "Using C (OCI) To Work With LOBs" on page 3-10.                                       |
| C++          | Oracle Call Interface for C++ (OCCI)          | <i>Oracle C++ Call Interface Programmer's Guide</i>   | "Using C++ (OCCI) To Work With LOBs" on page 3-22.                                    |
| C/C++        | Pro*C/C++ Precompiler                         | <i>Pro*C/C++ Precompiler Programmer's Guide</i>   | "Using C/C++ (Pro*C) To Work with LOBs" on page 3-29.                                 |
| COBOL        | Pro*COBOL Precompiler                         | <i>Pro*COBOL Precompiler Programmer's Guide</i>   | "Using COBOL (Pro*COBOL) to Work with LOBs" on page 3-33.                             |
| Visual Basic | Oracle Objects For OLE (OO4O)                 | Oracle Objects for OLE (OO4O) is a Windows-based product included with Oracle9i Client for Windows NT.<br><br>There are no manuals for this product, only online help. Online help is available through the Application Development submenu of the Oracle9i installation. | "Using Visual Basic (Oracle Objects for OLE (OO4O)) to Work with LOBs" on page 3-37." |
| Java         | JDBC Application Programmatic Interface (API) | <i>Oracle9i SQLJ Developer's Guide and Reference</i> and <i>Oracle9i JDBC Developer's Guide and Reference</i> .   | "Using Java (JDBC) To Work with LOBs" on page 3-45.                                   |
| OLEDB        | OraOLEDB, an OLE DB provider for Oracle.      | <i>Oracle Provider for OLE DB Developer's Guide</i>   |   |



## Comparing the LOB Interfaces

Table 3–2 and Table 3–3 compare the eight LOB programmatic interfaces by listing their functions and methods used to operate on LOBs. The tables are split in two simply to accommodate all eight interfaces. The interfaces' functionality, with regards LOBs, is described in the following sections.

**Table 3–2 Comparing the LOB Interfaces, 1 of 2**

| <b>PL/SQL: DBMS_LOB<br/>(dbmslob.sql)</b> | <b>C (OCI)<br/>(ociap.h)</b> | <b>C++ (OCCI)<br/>(occiData.h). Also for<br/>OCCIClob and OCCIBfile<br/>classes.</b> | <b>Pro*C/C++ and<br/>Pro*COBOL</b> |
|---|------------------------------|--|------------------------------------|
| DBMS_LOB.COMPARE                          | N/A                          | N/A  | N/A                                |
| DBMS_LOB.INSTR                            | N/A                          | N/A  | N/A                                |
| DBMS_LOB.SUBSTR                           | N/A                          | N/A  | N/A                                |
| DBMS_LOB.APPEND                           | OCILob.Append                | OCCIBlob.append()  | APPEND                             |
| N/A [use PL/SQL assign<br>operator]       | OCILob.Assign                |  | ASSIGN                             |
| N/A                                       | OCILob.CharSetForm           | OCCIClob.getCharSetForm<br>(CLOB only)   | N/A                                |
| N/A                                       | OCILob.CharSetId             | OCCIClob.getCharSetId()<br><br>(CLOB only)   | N/A                                |
| DBMS_LOB.CLOSE                            | OCILob.Close                 | OCCIBlob.close()   | CLOSE                              |
| N/A                                       | N/A                          | OCCIClob.closeStream()   | N/A                                |
| DBMS_LOB.COPY                             | OCILob.Copy                  | OCCIBlob.copy()  | COPY                               |
| N/A                                       | OCILob.DisableBuffering      | N/A  | DISABLE BUFFERING                  |
| N/A                                       | OCILob.EnableBuffering       | N/A  | ENABLE BUFFERING                   |
| DBMS_LOB.ERASE                            | OCILob.Erase                 | N/A  | ERASE                              |
| DBMS_LOB.FILECLOSE                        | OCILob.FileClose             | OCCIClob.close()   | CLOSE                              |
| DBMS_LOB.FILECLOSEALL                     | OCILob.FileCloseAll          | N/A  | FILE CLOSE ALL                     |
| DBMS_LOB.FILEEXISTS                       | OCILob.FileExists            | OCCIBfile.fileExists()   | DESCRIBE [FILEEXISTS]              |
| DBMS_LOB.GETCHUNKSIZE                     | OCILob.GetChunkSize          | OCCIBlob.getChunkSize()  | DESCRIBE [CHUNKSIZE]               |
| DBMS_LOB.FILEGETNAME                      | OCILob.FileGetName           | OCCIBfile.getFileName() and<br>OCCIBfile.getDirAlias()                               | DESCRIBE [DIRECTORY,<br>FILENAME]  |
| DBMS_LOB.FILEISOPEN                       | OCILob.FileIsOpen            | OCCIBfile.isOpen()   | DESCRIBE [ISOPEN]                  |
| DBMS_LOB.FILEOPEN                         | OCILob.FileOpen              | OCCIBfile.open()   | OPEN                               |
| N/A (use BFILENAME<br>operator)           | OCILob.FileSetName           | OCCIBfile.setName()  | FILE SET                           |
| N/A                                       | OCILob.FlushBuffer           | N/A  | FLUSH BUFFER                       |
| DBMS_LOB.GETLENGTH                        | OCILob.GetLength             | OCCIBlob.length()  | DESCRIBE [LENGTH]                  |

**Table 3–2 Comparing the LOB Interfaces, 1 of 2 (Cont.)**

| <b>PL/SQL: DBMS_LOB<br/>(dbmslob.sql)</b> | <b>C (OCI)<br/>(ociap.h)</b>               | <b>C++ (OCCI)<br/>(occiData.h). Also for<br/>OCCIClob and OCCIBfile<br/>classes.</b> | <b>Pro*C/C++ and<br/>Pro*COBOL</b> |
|---|--|--|------------------------------------|
| N/A                                       | OCILob.IsEqual                             | use operator = ( )/= !=  | N/A                                |
| DBMS_LOB.ISOPEN                           | OCILob.IsOpen                              | OCCIBlob.isOpen()  | DESCRIBE [ISOPEN]                  |
| DBMS_LOB.LOADFROMFILE                     | OCILob.LoadFromFile                        | Use the overloadedcopy()<br>method.  | LOAD FROM FILE                     |
| N/A                                       | OCILob.LocatorIsInit                       | OCCIClob.isinitialized()   | N/A                                |
| DBMS_LOB.OPEN                             | OCILob.Open                                | OCCIBlob.open  | OPEN                               |
| DBMS_LOB.READ                             | OCILob.Read                                | OCCIBlob.read  | READ                               |
| DBMS_LOB.TRIM                             | OCILob.Trim                                | OCCIBlob.trim  | TRIM                               |
| DBMS_LOB.WRITE                            | OCILob.Write                               | OCCIBlob.write   | WRITEORALOB.                       |
| DBMS_LOB.WRITEAPPEND                      | OCILob.WriteAppend                         | N/A  | WRITE APPEND                       |
| DBMS_LOB.CREATETEMPORARY                  | OCILob.CreateTemporary                     | N/A  | N/A                                |
| DBMS_LOB.FREETEMPORARY                    | OCILob.FreeTemporary                       | N/A  | N/A                                |
| DBMS_LOB.ISTEMPORARY                      | OCILob.IsTemporary<br>OCILob.LocatorAssign | N/A<br>use operator = ( ) or copy<br>constructor                                     | N/A<br>N/A                         |

**Table 3–3 Comparing the LOB Interfaces, 2 of 2**

| <b>PL/SQL: DBMS_LOB<br/>(dbmslob.sql)</b> | <b>Visual Basic (OO4O)</b> | <b>Java (JDBC)</b>                                     | <b>OLEDB</b> |
|---|----------------------------|--|--------------|
| DBMS_LOB.COMPARE                          | ORALOB.Compare             | Use DBMS_LOB.<br>position                              | N/A          |
| DBMS_LOB.INSTR                            | ORALOB.Matchpos            | getBytes for BLOBs or BFILEs<br>getSubString for CLOBs | N/A          |
| DBMS_LOB.SUBSTR                           | N/A                        | Use length and then<br>putBytes or PutString           | N/A          |
| DBMS_LOB.APPEND                           | ORALOB.Append              | N/A [use equal sign]                                   | N/A          |
| N/A [use PL/SQL assign<br>operator]       | ORALOB.Clone               | N/A  | N/A          |
| N/A                                       | N/A                        | N/A  | N/A          |
| N/A                                       | N/A                        | N/A  | N/A          |
| DBMS_LOB.CLOSE                            | N/A                        | use DBMS_LOB.  | N/A          |
| DBMS_LOB.COPY                             | ORALOB.Copy                | Use read and write                                     | N/A          |
| N/A                                       | ORALOB.DisableBuffering    | N/A  | N/A          |
| N/A                                       | ORALOB.EnableBuffering     | N/A  | N/A          |

**Table 3-3 Comparing the LOB Interfaces, 2 of 2 (Cont.)**

| <b>PL/SQL: DBMS_LOB<br/>(dbmslob.sql)</b> | <b>Visual Basic (OO4O)</b>                          | <b>Java (JDBC)</b>  | <b>OLEDB</b>  |
|---|---|---|---|
| DBMS_LOB.ERASE                            | ORALOB.Erase  | Use DBMS_LOB.   | N/A   |
| DBMS_LOB.FILECLOSE                        | ORABFILE.Close                                      | closeFile   | N/A   |
| DBMS_LOB.FILECLOSEALL                     | ORABFILE.CloseAll                                   | Use DBMS_LOB.   | N/A   |
| DBMS_LOB.FILEEXISTS                       | ORABFILE.Exist                                      | fileExists  | N/A   |
| DBMS_LOB.GETCHUNKSIZE                     | N/A   | getChunkSize  | N/A   |
| DBMS_LOB.FILEGETNAME                      | ORABFILE.<br>DirectoryName<br>ORABFILE.<br>FileName | getDirAlias<br>getName  | N/A   |
| DBMS_LOB.FILEISOPEN                       | ORABFILE.IsOpen                                     | Use DBMS_LOB.ISOPEN   | N/A   |
| DBMS_LOB.FILEOPEN                         | ORABFILE.Open                                       | openFile  | N/A   |
| N/A (use BFILENAME<br>operator)           | DirectoryName<br>FileName                           | Use BFILENAME   | N/A   |
| N/A                                       | ORALOB.FlushBuffer                                  | N/A   | N/A   |
| DBMS_LOB.GETLENGTH                        | ORALOB.Size   | length  | N/A   |
| N/A                                       | N/A   | equals  | N/A   |
| DBMS_LOB.ISOPEN                           | ORALOB.IsOpen                                       | use DBMS_LOB. IsOpen  | N/A   |
| DBMS_LOB.LOADFROMFILE                     | ORALOB.<br>CopyFromBfile                            | Use read and then write   | N/A   |
| DBMS_LOB.OPEN                             | ORALOB.open   | Use DBMS_LOB.   | N/A   |
| DBMS_LOB.READ                             | ORALOB.Read   | BLOB or BFILE: getBytes<br>and getBinaryStream<br>CLOB: getString and<br>getSubString and<br>getCharacterStream | IRowset::GetData and<br>ISequentialStream::Read           |
| DBMS_LOB.TRIM                             | ORALOB.Trim   | Use DBMS_LOB.   | N/A   |
| DBMS_LOB.WRITE                            | ORALOB.Write  | BLOB or BFILE: putBytes<br>and<br>getBinaryOutputStream<br>CLOB: putString and<br>getCharacterOutputStream      | IRowsetChange::SetData<br>and<br>ISequentialStream::Write |
| DBMS_LOB.WRITEAPPEND                      | N/A   | Use length and then<br>putString or putBytes  | N/A   |

**Table 3–3 Comparing the LOB Interfaces, 2 of 2 (Cont.)**

| PL/SQL: DBMS_LOB<br>(dbmslob.sql) | Visual Basic (OO4O) | Java (JDBC) | OLEDB |
|-----------------------------------|---------------------|-------------|-------|
| DBMS_LOB.CREATETEMPORARY          | N/A                 | N/A         | N/A   |
| DBMS_LOB.FREETEMPORARY            | N/A                 | N/A         | N/A   |
| DBMS_LOB.ISTEMPORARY              | N/A                 | N/A         | N/A   |

## Using PL/SQL (DBMS\_LOB Package) To Work With LOBs

The PL/SQL DBMS\_LOB package can be used for the following operations:

- **Internal persistent LOBs and Temporary LOBs:** Read and modify operations, either entirely or in a piece-wise manner.
- **BFILES:** Read operations

**See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for detailed documentation, including parameters, parameter types, return values, and example code.

### Provide a LOB Locator Before Invoking the DBMS\_LOB Routine

As described in more detail in the following, DBMS\_LOB routines work based on *LOB locators*. For the successful completion of DBMS\_LOB routines, you must provide an input locator representing a LOB that exists in the database tablespaces or external file system, *before* you invoke the routine.

- **Internal LOBs:** First use SQL to define tables that contain LOB columns, and subsequently you can use SQL to initialize or populate the locators in these LOB columns.
- **External LOBs:** Define a DIRECTORY object that maps to a valid physical directory containing the external LOBs that you intend to access. These files must exist, and have READ permission for Oracle Server to process. If your operating system uses case-sensitive path names, specify the directory in the correct case. See [Chapter 12, "External LOBs \(BFILES\)"](#), "Directory Object" on page 12-4 for more information.

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.

## PL/SQL - LOB Guidelines

### Client PL/SQL Procedures Cannot Call DBMS\_LOB Routines

*Client-side* PL/SQL procedures cannot call `DBMS_LOB` package routines.

However, you can use *server-side* PL/SQL procedures or anonymous blocks in `Pro*C/C++` to call `DBMS_LOB` package routines.

### Offset and Amount Parameters: Fixed-Width Versus Varying-Width, Character or Byte For DBMS\_LOB Package

For `DBMS_LOB` package, for example, for both *fixed and varying-width* character sets, the following rules apply:

- CLOBs and NCLOBs: Offset and amount parameters are always in characters
- BLOBs and BFILES: Offset and amount parameters are always in bytes

### DBMS\_LOB.LOADFROMFILE: Amount Parameter Value

When using `DBMS_LOB.LOADFROMFILE`, you cannot specify an amount parameter value larger than the size of the BFILE. (Although you can specify the `LOBMAXSIZE` constant for the amount parameter value to load the entire BFILE.)

### DBMS\_LOB.READ: Amount Parameter Can be Larger than Data Size

When using `DBMS_LOB.READ`, the amount parameter *can* be larger than the size of the data. In PL/SQL, the amount should be less than or equal to the size of the *buffer*, and the buffer size is limited to *32K*.

**See Also:**

- ["Loading an Internal Persistent BLOB with Binary Data from a BFILE" and "Loading an Internal Persistent CLOB with BFILE Data" in Chapter 10](#)
- ["Loading a Temporary BLOB with Binary Data from a BFILE" and "Loading a Temporary CLOB/NCLOB with Character Data from a File" in Chapter 11](#)
- ["Loading a BLOB with BFILE Data" and "Loading a CLOB with BFILE Data" in Chapter 12](#)

## PL/SQL Functions and Procedures that Operate on LOBs

PL/SQL functions and procedures that operate on BLOBs, CLOBs, NCLOBs, and BFILEs are summarized in the following:

- To modify internal LOB values, see [Table 3-4](#)
- To read or examine LOB values, see [Table 3-5](#)
- To create, free, or check on temporary LOBs, see [Table 3-6](#)
- For read-only functions on external LOBs (BFILEs), see [Table 3-7](#)
- To open or close a LOB, or check if LOB is open, see [Table 3-8](#)

## PL/SQL Functions/Procedures To Modify BLOB, CLOB, and NCLOB Values

**Table 3-4** *PL/SQL: DBMS\_LOB Procedures to Modify BLOB, CLOB, and NCLOB Values*

| Function/Procedure | Description  |
|--------------------|--|
| APPEND()           | Appends the LOB value to another LOB                 |
| COPY()             | Copies all or part 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                 |
| LOADCLOBFROMFILE() | Load character data from a file into a LOB           |
| LOADBLOBFROMFILE() | Load binary data from a file into a LOB              |

**Table 3–4 PL/SQL: DBMS\_LOB Procedures to Modify BLOB, CLOB, and NCLOB Values**

| 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                   |

## PL/SQL Functions/Procedures To Read or Examine Internal and External LOB Values

**Table 3–5 PL/SQL: DBMS\_LOB Procedures to Read or Examine Internal and External LOB values**

| Function/Procedure | Description   |
|--------------------|---|
| COMPARE()          | Compares the value of two LOBs  |
| GETCHUNKSIZE()     | Gets the chunk size used when reading and writing. This only works on internal LOBs and does not apply to external LOBs (BFILES). |
| 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  |

## PL/SQL Functions/Procedures To Operate on Temporary LOBs

**Table 3–6 PL/SQL: DBMS\_LOB Procedures to 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                             |

## PL/SQL Read-Only Functions/Procedures for BFILEs

**Table 3–7** PL/SQL: DBMS\_LOB Read-Only Procedures for BFILEs

| Function/Procedure | Description   |
|--------------------|---|
| FILECLOSE()        | Closes the file. Use CLOSE() instead.   |
| 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. Use ISOPEN() instead. |
| FILEOPEN()         | Opens a file. Use OPEN() instead.   |

## PL/SQL Functions/Procedures To Open and Close Internal and External LOBs

**Table 3–8** PL/SQL: DBMS\_LOB Procedures to Open and Close Internal and External LOBs

| Function/Procedure | Description           |
|--------------------|-----------------------|
| OPEN()             | Opens a LOB           |
| ISOPEN()           | Sees if a LOB is open |
| CLOSE()            | Closes a LOB          |

These procedures are described in detail for specific LOB operations, such as, INSERT a row containing a LOB, in these chapters:

- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 11, "Temporary LOBs"](#)
- [Chapter 12, "External LOBs \(BFILEs\)"](#)

You can access many of the PL/SQL LOB example scripts from these chapters, at \$ORACLE\_HOME/rdbms/demo/lobs/plsql.

## Using C (OCI) To Work With LOBs

Oracle Call Interface (OCI) can be used to make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it through OCI, as follows:

- For reading from internal and external LOBs (BFILEs)



- For writing to internal LOBs

OCI also includes functions that you can use to:

- Access data stored in internal LOBs (BLOBs, CLOBs, NCLOBs) and external LOBs (BFILES)
- Modify internal LOBs (BLOBs, CLOBs, NCLOBs)

These functions are listed in the following tables, and are discussed in greater detail later in this section.

## Set CSID Parameter To OCI\_UCS2ID to Read/Write in UCS2

If you want to read or write data in 2 byte unicode (UCS2) format, set the `csid` (character set ID) parameter in `OCIlobRead` and `OCIlobWrite` to `OCI_UCS2ID`. The `csid` parameter indicates the character set id 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.

### See Also:

- *Oracle Call Interface Programmer's Guide* for detailed documentation, including parameters, parameter types, return values, and example code.
- *Oracle9i Database Globalization Support Guide* for detailed information about implementing applications in different languages.

## Offset and Amount Parameters: Fixed-Width Versus Varying-Width, Character or Byte

### Fixed Width Character Set Rules

In OCI, for *fixed-width* client-side character sets, the following rules apply:

- CLOBs and NCLOBs: offset and amount parameters are always in characters
- BLOBs and BFILES: offset and amount parameters are always in bytes

### Varying-Width Character Set Rules

The following rules apply only to *varying-width* client-side character sets:

- **Offset parameter:** Regardless of whether the client-side character set is varying-width, the offset parameter is always as follows:
  - CLOBs and NCLOBs: in characters
  - BLOBs and BFILES: in bytes
- **Amount parameter:** The amount parameter is always as follows:
  - When referring to a server-side LOB: in characters
  - When referring to a client-side buffer: in bytes
- **OCIlobFileGetLength:** Regardless of whether the client-side character set is varying-width, the output length is as follows:
  - CLOBs and NCLOBs: in characters
  - BLOBs and BFILES: in bytes
- **OCIlobRead:** With client-side character set of varying-width, CLOBs and NCLOBs:
  - *Input amount* is in characters. Input amount refers to the number of characters to read from the server-side CLOB or NCLOB.
  - *Output amount* is in bytes. Output amount indicates how many bytes were read into the buffer 'bufp'.
- **OCIlobWrite:** With client-side character set of varying-width, CLOBs and NCLOBs:
  - *Input amount* is in bytes. Input amount refers to the number of bytes of data in the input buffer 'bufp'.
  - *Output amount* is in characters. 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.

**See also:** *Oracle9i Database Globalization Support Guide*

## NCLOBs

NCLOBs parameters are allowed in methods.

## OCILobLoadFromFile: Specify Amount Parameter to be Less than Length of BFILE

When using OCILobLoadFromFile you cannot specify amount larger than the length of the BFILE.

## OCILobRead: Specify Amount Parameter to be 4 gigabytes - 1

In OCILobRead, you can specify amount = 4 gigabytes-1, and it will read to the end of the LOB.

## OCI LOB Examples

Further OCI examples are provided in:

- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 11, "Temporary LOBs"](#)
- [Chapter 12, "External LOBs \(BFILES\)"](#).

You can access most of the OCI LOB example scripts from your Oracle9i distribution software at \$ORACLE\_HOME/rdbms/demo/lobs/oci.

There are further example OCI scripts:

- For UNIX systems at:
  - /ORACLE\_HOME/rdbms/demo/demolb.c
  - /ORACLE\_HOME/rdbms/demo/demolb2.c
  - /ORACLE\_HOME/rdbms/demo/demolbs.c
- For Windows NT at:
  - \ORACLE\_HOME\Oci\Samples\demolb.c,....

See also Appendix B, "OCI Demonstration Programs" in *Oracle Call Interface Programmer's Guide*, for further OCI demonstration script listings.

## Further Information About OCI

For further information and features of OCI, refer to the OTN Web site, <http://otn.oracle.com/> for OCI features and FAQs.

## OCI Functions that Operate on BLOBs, CLOBs, NCLOBs, and BFILES

OCI functions that operate on BLOBs, CLOBs, NCLOBs, and BFILES are as follows:

- To modify internal LOBs, see [Table 3-9](#)
- To read or examine LOB values, see [Table 3-10](#)
- To create or free temporary LOB, or check if Temporary LOB exists, see [Table 3-11](#)
- For read only functions on external LOBs (BFILES), see [Table 3-12](#)
- To operate on LOB locators, see [Table 3-13](#)
- For LOB buffering, see [Table 3-14](#)
- To open and close LOBs, see [Table 3-15](#)

## OCI Functions To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values

**Table 3-9** *OCI Functions To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values*

| Function/Procedure   | Description  |
|----------------------|--|
| OCILobAppend()       | Appends LOB value to another LOB.                                |
| OCILobCopy()         | Copies all or part of a LOB to 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. |
| OCILobWriteAppend()  | Writes data from a buffer to the end of the LOB.                 |

## OCI Functions To Read or Examine Internal LOB and External LOB (BFILE) Values

**Table 3–10 OCI Functions To Read or Examine internal LOB and external LOB (BFILE) Values**

| Function/Procedure   | Description  |
|----------------------|--|
| OCILobGetChunkSize() | Gets the Chunk size used when reading and writing. This works on internal LOBs and does not apply to external LOBs (BFILES). |
| 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.  |

## OCI Functions For Temporary LOBs

**Table 3–11 OCI Functions for Temporary LOBs**

| Function/Procedure      | Description                    |
|-------------------------|--------------------------------|
| OCILobCreateTemporary() | Creates a temporary LOB        |
| OCILobIsTemporary()     | Sees if a temporary LOB exists |
| OCILobFreeTemporary()   | Frees a temporary LOB          |

## OCI Read-Only Functions For BFILES

**Table 3–12 OCI Read-Only Functions for BFILES**

| Function/Procedure   | Description                     |
|----------------------|---------------------------------|
| OCILobFileClose()    | Closes an open BFILE.           |
| OCILobFileCloseAll() | Closes all open BFILES.         |
| OCILobFileExists()   | Checks whether a BFILE exists.  |
| OCILobFileGetName()  | Returns the name of a BFILE.    |
| OCILobFileIsOpen()   | Checks whether a BFILE is open. |
| OCILobFileOpen()     | Opens a BFILE.                  |

## OCI LOB Locator Functions

**Table 3–13** *OCI LOB-Locator Functions*

| 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.           |

## OCI LOB-Buffering Functions

**Table 3–14** *OCI LOB-Buffering Functions*

| 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 (server)  |

## OCI Functions To Open and Close Internal and External LOBs

**Table 3–15** *OCI Functions To Open and Close Internal and External LOBs*

| Function/Procedure | Description           |
|--------------------|-----------------------|
| OCILobOpen()       | Opens a LOB           |
| OCILobIsOpen()     | Sees if a LOB is open |
| OCILobClose()      | Closes a LOB          |

## OCI Example — Is the LOB Open: main() and selfLOBIsOpen

To work with the OCI examples in the remainder of the book, you can use a main() like the following. Here, it is used with `selfLOBIsOpen` 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;
    OCISstmt    *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 *defnpl;

    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, OCISmtExecute(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;
OCISmt    *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++ (OCCI) To Work With LOBs

Oracle C++ Call Interface (OCCI) is a C++ API for manipulating data in an Oracle database. OCCI is organized as an easy-to-use collection of C++ classes which enable a C++ program to connect to a database, execute SQL statements, insert/update values in database tables, retrieve results of a query, execute stored procedures in the database, and access metadata of database schema objects. OCCI also provides a seamless interface to manipulate objects of user-defined types as C++ class instances.

Oracle C++ Call Interface (OCCI) is designed so that you can use OCI and OCCI together to build applications.

The OCCI API provides the following advantages over JDBC and ODBC:

- OCCI encompasses more Oracle functionality than JDBC. OCCI provides all the functionality of OCI that JDBC does not provide.
- OCCI provides *compiled* performance. With compiled programs, the source code is already written as “close to the machine” as possible. Since JDBC is an *interpreted* API, it cannot provide the performance of a compiled API. With an interpreted program, performance degrades as each line of code must be interpreted individually into code that is close to the machine.
- OCCI provides memory management with smart pointers. You do not have to be concerned about managing memory for OCCI objects. This results in robust higher performance application code.
- Navigational access of OCCI enables you to intuitively access objects and call methods. Changes to objects persist without need to write corresponding SQL statements. If you use the client side cache, the navigational interface performs better than the object interface.
- With respect to ODBC, the OCCI API is simpler to use. Since ODBC is built on the C language, OCCI has all the advantages C++ provides over C. Moreover, ODBC has a reputation as being difficult to learn. The OCCI, by contrast, is designed for ease of use.

You can use Oracle C++ Call Interface (OCCI) to make changes to an entire internal LOB, or to pieces of the beginning, middle, or end of it, as follows:

- For reading from internal and external LOBs (BFILES)
- For writing to internal LOBs

## Distinct Classes for Each LOB Type

Unlike OCI which uses a common API for operations on BLOBs, CLOBs, and BFILEs, OCCI has distinct classes for each LOB types, as follows:

- *OCCIClob* Class to access and modify data stored in internal CLOBs and NCLOBs
- *OCCIBlob* Class to access and modify data stored in internal BLOBs
- *OCCIBfile* Class to access and read data stored in external LOBs (BFILEs)

### OCCIClob Class

The *OCCIClob* driver implements a CLOB object using an SQL `locator(CLOB)`. This means that a CLOB object contains a logical pointer to the SQL CLOB data rather than the data itself.

The CLOB interface provides methods for getting the length of an SQL CLOB (Character Large Object) value, for materializing a CLOB value on the client, and getting a substring. Methods in the interfaces *ResultSet* statement such as `getClob()` and `setClob()` allow you to access SQL CLOB values.

### OCCIBlob Class

Methods in the *OCCIResultSet* and *OCCIStatement* interfaces, such as `getBlob()` and `setBlob()`, allow you to access SQL BLOB values. The *OCCIBlob* interface provides methods for getting the length of a SQL BLOB value, for materializing a BLOB value on the client, and for extracting a part of the BLOB.

These methods are listed in the following tables.

#### See Also:

- *Oracle C++ Call Interface Programmer's Guide* for detailed documentation, including parameters, parameter types, return values, and example code.
- *Oracle9i Database Globalization Support Guide* for detailed information about implementing applications in different languages.

## Offset and Amount Parameters: Fixed-Width Versus Varying-Width, Character or Byte

### Fixed Width Character Set Rules

In OCI, for *fixed-width* client-side character sets, the following rules apply:

- `OCIClob`: offset and amount parameters are always in characters
- `OCIBlob`: offset and amount parameters are always in bytes
- `OCIFile`: offset and amount parameters are always in bytes

### Varying-Width Character Set Rules

The following rules apply only to *varying-width* client-side character sets:

- **Offset parameter**: Regardless of whether the client-side character set is varying-width, the offset parameter is always as follows:
  - `OCIClob()`: in characters
  - `OCIBlob()`: in bytes
  - `OCIFile()`: in bytes
- **Amount parameter**: The amount parameter is always as follows:
  - `OCIClob`: in characters, when referring to a server-side LOB
  - `OCIBlob`: in bytes, when referring to a client-side buffer
  - `OCIFile`: in bytes, when referring to a client-side buffer
- **length()**: Regardless of whether the client-side character set is varying-width, the output length is as follows:
  - `OCIClob.length()`: in characters
  - `OCIBlob.length()`: in bytes
  - `OCIFile.length()`: in bytes
- **`OCIClob.read()` and `OCIBlob.read()`**: With client-side character set of varying-width, CLOBs and NCLOBs:
  - *Input amount* is in characters. Input amount refers to the number of characters to read from the server-side CLOB or NCLOB.
  - *Output amount* is in bytes. Output amount indicates how many bytes were read into the OCI buffer parameter, 'buffer'.

- `OCCIClob.write()` and `OCCIBlob.write()`: With client-side character set of varying-width, CLOBs and NCLOBs:
  - *Input amount* is in bytes. Input amount refers to the number of bytes of data in the OCI input buffer, 'buffer'.
  - *Output amount* is in characters. Output amount refers to the number of characters written into the server-side CLOB or NCLOB.

### OffSet and Amount Parameters for Other OCI Operations

For all other OCI LOB operations, irrespective of the client-side character set, the *amount parameter* is in characters for CLOBs and NCLOBs. These include the following:

- `OCCIClob.copy()`
- `OCCIClob.erase()`
- `OCCIClob.trim()`
- For `LoadFromFile` functionality, overloaded `OCCIClob.copy()`

All these operations refer to the amount of LOB data on the server.

**See also:** *Oracle9i Database Globalization Support Guide*

### NCLOBs

- NCLOBs parameters are allowed in methods
- NCLOBs parameters are *not* allowed as attributes in object types

## Loading from Files with `OCCIClob.copy()` and `OCCIBlob.copy()`: Amount Parameter

The `LoadFromFile` functionality in OCI is provided through the `OCCIClob.copy()` and `OCCIBlob.copy()` methods. These take an `OCCIBfile` argument.

You cannot specify amounts larger than the length of the `BFILE`. You must specify an amount parameter less than length of the `BFILE`.

## `OCCIClob.read()`, `OCCIBlob.read()`, and `OCCIBfile.read()`: Amount Parameter

When reading in `OCCIClobs`, `OCCIBlobs`, and `OCCIBfiles`, specify the amount = 4 gigabytes-1, and it will read to the end of the LOB.

## Further Information About OCCI

### See Also:

- *Oracle C++ Call Interface Programmer's Guide*
- <http://www.oracle.com/> search for articles and product information featuring OCCI.

## OCCI Methods that Operate on BLOBs, CLOBs, NCLOBs, and BFILEs

OCCI methods that operate on BLOBs, CLOBs, NCLOBs, and BFILEs are as follows:

- To modify internal LOBs, see [Table 3-16](#)
- To read or examine LOB values, see [Table 3-17](#)
- For read only methods on external LOBs (BFILEs), see [Table 3-18](#)
- Other LOB OCCI methods are described in [Table 3-19](#)
- To open and close LOBs, see [Table 3-20](#)

## OCCI Methods To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values

**Table 3-16** *OCCI Clob and OCCIBlob Methods To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values*

| Function/Procedure             | Description  |
|--------------------------------|--|
| <code>OCCIBlob.append()</code> | Appends CLOB or BLOB value to another LOB.                       |
| <code>OCCIBlob.copy()</code>   | Copies all or part of a CLOB or BLOB to another LOB.             |
| <code>OCCIBlob.copy()</code>   | Loads BFILE data into an internal LOB.                           |
| <code>OCCIBlob.trim()</code>   | Truncates a CLOB or BLOB.  |
| <code>OCCIBlob.write()</code>  | Writes data from a buffer into a LOB, overwriting existing data. |



## OCCI Methods To Read or Examine Internal LOB and BFILE Values

**Table 3–17** *OCCI Blob/OCCIClob/OCCIBfile Methods To Read or Examine internal LOB and external LOB (BFILE) Values*

| Function/Procedure      | Description  |
|-------------------------|--|
| OCCIBlob.getChunkSize() | Gets the Chunk size used when reading and writing. This works on internal LOBs and does not apply to external LOBs (BFILES). |
| OCCIBlob.length()       | Returns the length of a LOB or a BFILE.  |
| OCCIBlob.read()         | Reads a specified portion of a non-null LOB or a BFILE into a buffer.  |

## OCCI Read-Only Methods For BFILES

**Table 3–18** *OCCI Read-Only Methods For BFILES*

| Function/Procedure      | Description                     |
|-------------------------|---------------------------------|
| OCCIBfile.close()       | Closes an open BFILE.           |
| OCCIBfile.fileExists()  | Checks whether a BFILE exists.  |
| OCCIBfile.getFileName() | Returns the name of a BFILE.    |
| OCCIBfile.getDirAlias() | Gets the directory alias.       |
| OCCIBfile.isOpen()      | Checks whether a BFILE is open. |
| OCCIBfile.open()        | Opens a BFILE.                  |

## Other OCCI LOB Methods

**Table 3–19** *Other OCCI LOB Methods*

| Methods                             | Description  |
|-------------------------------------|--|
| OCCIClob/Blob/Bfile.=               | Assigns one LOB locator to another. Use = or the copy constructor. |
| OCCIClob.getCharSetForm()           | Returns the character set form of a LOB.                           |
| OCCIClob.getCharSetId()             | Returns the character set ID of a LOB.                             |
| OCCIBfile.setName()                 | Sets the name of a BFILE.  |
| OCCIClob/Blob/Bfile.IsEqual()       | Checks whether two LOB refer to the same LOB.                      |
| OCCIClob/Blob/Bfile.isInitialized() | Checks whether a LOB is initialized.                               |

## OCCI Methods To Open and Close Internal and External LOBs

**Table 3–20** *OCCI Methods To Open and Close Internal and External LOBs*

| <b>Function/Procedure</b>    | <b>Description</b>    |
|------------------------------|-----------------------|
| OCCIClob/Blob/Bfile.Open()   | Opens a LOB           |
| OCCIClob/Blob/Bfile.IsOpen() | Sees if a LOB is open |
| OCCIClob/Blob/Bfile.Close()  | Closes a LOB          |

## Using C/C++ (Pro\*C) To Work with LOBs

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of a LOB by using embedded SQL. You can access both internal and external LOBs for read purposes, and you can *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 following tables, and are discussed in greater detail later in the chapter.

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

### First Provide an Allocated Input Locator Pointer that Represents LOB

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.

To successfully complete an embedded SQL LOB statement you must do the following:

1. 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.
2. SELECT a LOB locator into a LOB locator pointer variable
3. Use this variable in the embedded SQL LOB statement to access and manipulate the LOB value

Examples provided with each embedded SQL LOB statement are illustrated in:

- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 11, "Temporary LOBs"](#)
- [Chapter 12, "External LOBs \(BFILES\)"](#).

You can access these Pro\*C/C++ LOB example scripts from \$ORACLE\_HOME/rdbms/demo/lobs/.

### Pro\*C/C++ Statements that Operate on BLOBs, CLOBs, NCLOBs, and BFILES

Pro\*C statements that operate on BLOBs, CLOBs, and NCLOBs are listed in the following tables:

- To modify internal LOBs, see [Table 3-21](#)

- To read or examine LOB values, see [Table 3–22](#)
- To create or free temporary LOB, or check if Temporary LOB exists, see [Table 3–23](#)
- To operate close and 'see if file exists' functions on BFILEs, see [Table 3–24](#)
- To operate on LOB locators, see [Table 3–25](#)
- For LOB buffering, see [Table 3–26](#)
- To open or close LOBs or BFILEs, see [Table 3–27](#)

## Pro\*C/C++ Embedded SQL Statements To Modify Internal LOB Values

**Table 3–21** *Pro\*C/C++: Embedded SQL Statements To Modify Internal LOB (BLOB, CLOB, and NCLOB) Values*

| Statement      | Description  |
|----------------|--|
| APPEND         | Appends a LOB value to another LOB.                          |
| COPY           | Copies all or a part 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.  |

## Pro\*C/C++ Embedded SQL Statements To Read or Examine Internal and External LOB Values

**Table 3–22** *Pro\*C/C++: Embedded SQL Statements To Read or Examine Internal and External LOB Values*

| Statement            | Description  |
|----------------------|--|
| DESCRIBE [CHUNKSIZE] | Gets the Chunk size used when writing. This works for internal LOBs only. It does not apply to external LOBs (BFILEs). |
| 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.  |

## Pro\*C/C++ Embedded SQL Statements For Temporary LOBs

**Table 3–23** *Pro\*C/C++: Embedded SQL Statements For 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.                           |

## Pro\*C/C++ Embedded SQL Statements For BFILES

**Table 3–24** *Pro\*C/C++: Embedded SQL Statements For BFILES*

| Statement                        | Description   |
|----------------------------------|---|
| FILE CLOSE ALL                   | Closes all open BFILES.                                 |
| DESCRIBE [FILEEXISTS]            | Checks whether a BFILE exists.                          |
| DESCRIBE<br>[DIRECTORY,FILENAME] | Returns the directory alias and/or filename of a BFILE. |

## Pro\*C/C++ Embedded SQL Statements For LOB Locators

**Table 3–25** *Pro\*C/C++ Embedded SQL Statements for LOB Locators*

| Statement | Description  |
|-----------|--|
| ASSIGN    | Assigns one LOB locator to another.                            |
| FILE SET  | Sets the directory alias and filename of a BFILE in a locator. |

## Pro\*C/C++ Embedded SQL Statements For LOB Buffering

**Table 3–26** *Pro\*C/C++ Embedded SQL Statements for LOB Buffering*

| 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)  |

## Pro\*C/C++ Embedded SQL Statements To Open and Close Internal and External LOBs

**Table 3–27** *Pro\*C/C++ Embedded SQL Statements To Open and Close Internal LOBs and External LOBs (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 following tables, and are discussed in greater detail later in the manual.

### First Provide an Allocated Input Locator Pointer that Represents LOB

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 perform the following:

1. 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.
2. SELECT a LOB locator into a LOB locator pointer variable
3. Use this variable in an embedded SQL LOB statement to access and manipulate the LOB value.

Examples provided with each embedded SQL LOB statement are illustrated in:

- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 11, "Temporary LOBs"](#)
- [Chapter 12, "External LOBs \(BFILES\)"](#).

You can access these Pro\*COBOL LOB example scripts from \$ORACLE\_HOME/rdbms/demo/lobs/.

Where the Pro\*COBOL interface does not supply the required functionality, you can call OCI using C. Such an example is not provided here because such programs are operating system dependent.

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

### Pro\*COBOL Statements that Operate on BLOBs, CLOBs, NCLOBs, and BFILES

The following Pro\*COBOL statements operate on BLOBs, CLOBs, NCLOBs, and BFILES:

- To modify internal LOBs, see [Table 3–28](#)
- To read or examine internal and external LOB values, see [Table 3–29](#)
- To create or free temporary LOB, or check LOB locator, see [Table 3–30](#)
- To operate close and 'see if file exists' functions on BFILES, see [Table 3–31](#)
- To operate on LOB locators, see [Table 3–32](#)
- For LOB buffering, see [Table 3–33](#)
- To open or close internal LOBs or BFILES, see [Table 3–34](#)

## Pro\*COBOL Embedded SQL Statements To Modify Internal LOB Values

**Table 3–28** *Pro\*COBOL Embedded SQL Statements To Modify BLOB, CLOB, and NCLOB Values*

| <b>Statement</b> | <b>Description</b>   |
|------------------|--|
| APPEND           | Appends a LOB value to another LOB.                          |
| COPY             | Copies all or part 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.  |



## Pro\*COBOL Embedded SQL Statements To Read or Examine Internal and External LOB Values

**Table 3–29** *Pro\*COBOL Embedded SQL Statements To Read or Examine Internal and External LOB Values*

| Statement            | Description   |
|----------------------|---|
| DESCRIBE [CHUNKSIZE] | Gets the Chunk size used when 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. |

## Pro\*COBOL Embedded SQL Statements For Temporary LOBs

**Table 3–30** *Pro\*COBOL Embedded SQL Statements For 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.                           |

## Pro\*COBOL Embedded SQL Statements For BFILES

**Table 3–31** *Pro\*COBOL Embedded SQL Statements For 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. |

## Pro\*COBOL Embedded SQL Statements For LOB Locators

**Table 3–32** *Pro\*COBOL Embedded SQL Statements For LOB Locators Statements*

| Statement | Description  |
|-----------|--|
| ASSIGN    | Assigns one LOB locator to another.                            |
| FILE SET  | Sets the directory alias and filename of a BFILE in a locator. |

## Pro\*COBOL Embedded SQL Statements For LOB Buffering

**Table 3–33** *Pro\*COBOL Embedded SQL Statements For LOB Buffering*

| <b>Statement</b>  | <b>Description</b>  |
|-------------------|---|
| 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)  |

## Pro\*COBOL Embedded SQL Statements To Open and Close Internal LOBs and BFILEs

**Table 3–34** *Pro\*COBOL Embedded SQL Statements To Open and Close Internal LOBs and BFILEs*

| <b>Statement</b>   | <b>Description</b>              |
|--------------------|---------------------------------|
| 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 (Oracle Objects for OLE (OO4O)) to Work with LOBs

Oracle Objects for OLE (OO4O) is a collection of programmable COM objects that simplifies the development of applications designed to communicate with an Oracle database. OO4O offers high performance database access. It also provides easy access to features unique to Oracle, yet otherwise cumbersome or inefficient to use from other ODBC or OLE DB-based components, such as ADO.

You can make changes to an entire internal LOB, or to pieces of the beginning, middle or end of it, with the Oracle Objects for OLE (OO4O) API, by using one of the following objects interfaces:

- **OraBlob**: To provide methods for performing operations on BLOB datatypes in the database
- **OraClob**: To provide methods for performing operations on CLOB datatypes in the database
- **OraBFile**: To provide methods for performing operations on BFILE data stored in operating system files.

---

---

**Note:** *OraBlob* and *OraClob* have been deprecated and no longer work!

---

---

## OO4O Syntax Reference

### Syntax

The OO4O syntax reference and further information is viewed from the OO4O online help. Oracle Objects for OLE (OO4O), a Windows-based product included with Oracle9i Client for Windows NT, has no manuals, only online help.

Its online help is available through the Application Development submenu of the Oracle9i installation. To view specific methods and properties from the Help Topics menu, select the Contents tab > OO4O Automation Server > Methods or Properties.

### Further Information

For further information about OO4O, refer to the following Web site:

- <http://otn.oracle.com> Select Products > Internet Tools > Programmer. Scroll down to "Oracle Objects for OLE". At the bottom of the page is a list of useful articles for using the interfaces.

- <http://www.oracle.com/> Search for articles on OO4O or Oracle Objects for OLE.

## OraBlob, OraClob, and OraBfile Object Interfaces Encapsulate Locators

These interfaces encapsulate LOB locators, so you do not deal directly with locators, but instead, can use methods and properties provided to perform operations and get state information.

### OraBlob and OraClob Objects Are Retrieved as Part of Dynaset and Represent LOB Locators

When `OraBlob` and `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 a move operation, the `OraBlob` and `OraClob` objects represent the LOB locator for the *new* current row.

### Use the Clone Method to Retain Locator Independent of the Dynaset Move

To retain the LOB locator of the `OraBlob` and `OraClob` object independent of the dynaset move operation, use the `Clone` method. This method returns the `OraBlob` and `OraClob` object. You can also use these objects as PL/SQL bind parameters.

## Example of OraBlob and OraBfile

The following example shows usage of `OraBlob` and `OraBfile`. Functions and examples are explained in greater detail in Chapters 10 and 12.

```
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 represeneted by
'OraSoundClone
OraSound1.Append OraSoundClone
OraDyn.Update

OraConnection.CommitTrans
```

In the preceding example:

OraSound1 — represents the locator for the current row in the dynaset

OraSoundClone — represents the locator for the 1st row.

A change in the current row (say a OraDyn.MoveNext) means the following:

OraSound1 — will represent the locator for the 2nd row

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 obtained from an PL/SQL “OUT” parameter as a result of executing a PL/SQL procedure, either by doing an

OraDatabase.ExecuteSQL.

---

---

**Note:** A LOB obtained by executing SQL is only valid for the duration of the transaction. For this reason, “BEGINTRANS” and “COMMITTRANS” are used to specify the duration of the transaction.

---

---

## OO4O Methods and Properties to Access Data Stored in LOBs

Oracle Objects for OLE (OO4O) includes methods and properties that you can use to access data stored in BLOBs, CLOBs, NCLOBs, and BFILES. These methods and

properties are listed in the following tables, and are discussed in greater detail in the following chapters:

- [Chapter 10, "Internal Persistent LOBs"](#)
- [Chapter 12, "External LOBs \(BFILES\)"](#).

You can also access OO4O LOB example scripts at \$ORACLE\_HOME/rdbms/demo/lobs.

**See Also:** The OO4O online help for detailed information including parameters, parameter types, return values, and example code. Oracle Objects for OLE (OO4O), a Windows-based product included with Oracle9i Client for Windows NT, has no manuals, only online help. The OO4O online help is available through the Application Development submenu of the Oracle9i installation.

The following OO4O methods and properties operate on BLOBs, CLOBs, NCLOBs, and BFILES:

- To modify internal LOBs, see [Table 3-35](#)
- To read or examine internal and external LOB values, see [Table 3-36](#)
- To open and close BFILES, see [Table 3-37](#)
- For LOB buffering, see [Table 3-38](#)
- Properties such as to see if LOB is NULL, or to get or set polling amount, see [Table 3-39](#)
- For read-only BFILE methods, see [Table 3-40](#)
- For BFILE properties, see [Table 3-41](#)

## OO4O Methods To Modify BLOB, CLOB, and NCLOB Values

**Table 3–35** *OO4O Methods To Modify BLOB, CLOB, and NCLOB Values*

| <b>Methods</b>       | <b>Description</b>   |
|----------------------|--|
| OraBlob.Append       | Appends BLOB value to another LOB.                             |
| OraClob.Append       | Appends CLOB or NCLOB value to another LOB.                    |
| OraBlob.Copy         | Copies a portion of a BLOB into another LOB                    |
| OraClob.Copy         | Copies a portion of a CLOB or NCLOB into another LOB           |
| OraBlob.Erase        | Erases part of a BLOB, starting at a specified offset          |
| OraClob.Erase        | Erases part of a CLOB or NCLOB, starting at a specified offset |
| OraBlob.CopyFromFile | Loads BFILE data into an internal BLOB                         |
| OraClob.CopyFromFile | Loads BFILE data into an internal CLOB or NCLOB                |
| OraBlob.Trim         | Truncates a BLOB   |
| OraClob.Trim         | Truncates a CLOB or NCLOB                                      |
| OraBlob.CopyFromFile | Writes data from a file to a BLOB                              |
| OraClob.CopyFromFile | Writes data from a file to a CLOB or NCLOB                     |
| OraBlob.Write        | Writes data to the BLOB  |
| OraClob.Write        | Writes data to the CLOB or NCLOB                               |

## OO4O Methods To Read or Examine Internal and External LOB Values

**Table 3–36** *OO4O Methods To Read or Examine Internal and External LOB Values*

| <b>Function/Procedure</b> | <b>Description</b>  |
|---------------------------|---|
| OraBlob.Read              | Reads a specified portion of a non-null BLOB into a buffer  |
| OraClob.Read              | Reads a specified portion of a non-null CLOB into a buffer  |
| OraBFile.Read             | Reads a specified portion of a non-null BFILE into a buffer |
| OraBlob.CopyToFile        | Reads a specified portion of a non-null BLOB to a file      |
| OraClob.CopyToFile        | Reads a specified portion of a non-null CLOB to a file      |

## OO4O Methods To Open and Close External LOBs (BFILES)

**Table 3–37** *OO4O Methods To Open and Close External LOBs (BFILES)*

| <b>Method</b>  | <b>Description</b> |
|----------------|--------------------|
| OraBFile.Open  | Opens BFILE.       |
| OraBFile.Close | Closes BFILE.      |



## OO4O Methods For Internal LOB-Buffering

**Table 3–38** OO4O Methods For Internal LOB-Buffering

| Method                   | Description  |
|--------------------------|--|
| OraBlob.FlushBuffer      | Flushes changes made to the BLOB buffering subsystem to the database |
| OraClob.FlushBuffer      | Flushes changes made to the CLOB buffering subsystem to the database |
| OraBlob.EnableBuffering  | Enables buffering of BLOB operations                                 |
| OraClob.EnableBuffering  | Enables buffering of CLOB operations                                 |
| OraBlob.DisableBuffering | Disables buffering of BLOB operations                                |
| OraClob.DisableBuffering | Disables buffering of CLOB operations                                |

## OO4O Properties For Operating on LOBs

**Table 3–39** OO4O Properties for Operating on LOBs

| Property                  | Description  |
|---------------------------|--|
| IsNull (Read)             | Indicates when a LOB is Null   |
| PollingAmount(Read/Write) | Gets/Sets total amount for Read/Write polling operation  |
| Offset(Read/Write)        | Gets/Sets offset for Read/Write operation. By default, it is set to 1.   |
| Status(Read)              | Returns the polling status. Possible values are <ul style="list-style-type: none"> <li>■ ORALOB_NEED_DATA There is more data to be read or written</li> <li>■ ORALOB_NO_DATA There is no more data to be read or written</li> <li>■ ORALOB_SUCCESS LOB data read/written successfully</li> </ul> |
| Size(Read)                | Returns the length of the LOB data   |

## OO4O Read-Only Methods For External Lobs (BFILES)

**Table 3–40** OO4O Read-Only Methods For External LOBs (BFILES)

| Methods           | Description                   |
|-------------------|-------------------------------|
| OraBFile.Close    | Closes an open BFILE          |
| OraBFile.CloseAll | Closes all open BFILES        |
| OraBFile.Open     | Opens a BFILE                 |
| OraBFile.IsOpen   | Determines if a BFILE is open |

## OO4O Properties For Operating on External LOBs (BFILEs)

**Table 3–41** *OO4O Properties For Operating on External LOBs (BFILEs)*

| <b>Property</b>               | <b>Description</b>                              |
|-------------------------------|---|
| 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 perform the following tasks on LOBs with Java (JDBC):

- [Changing Internal Persistent LOBs Using Java](#)
- [Reading Internal Persistent LOBs and External LOBs \(BFILES\) with Java](#)
- [Calling DBMS\\_LOB Package From Java \(JDBC\)](#)
- [Referencing LOBs Using Java \(JDBC\)](#)

### Changing Internal Persistent LOBs Using Java

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 using the objects:

- `oracle.sql.BLOB`
- `oracle.sql.CLOB`

These objects also implement `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.

### Reading Internal Persistent LOBs and External LOBs (BFILES) with Java

With JDBC you can use Java to *read* both internal persistent LOBs and external LOBs (BFILES).

#### **BLOB, CLOB, and BFILE Classes**

- ***BLOB* and *CLOB* Classes.** In JDBC these classes provide methods for performing operations on large objects in the database including BLOB and CLOB data types.
- ***BFILE* Class.** In JDBC this class provides methods for performing operations on BFILE data in the database.

The BLOB, CLOB, and BFILE classes encapsulate LOB locators, so you do not deal with locators but instead use methods and properties provided to perform operations and get state information.

## Calling DBMS\_LOB Package From Java (JDBC)

Any LOB functionality not provided by these classes can be accessed by a call to the PL/SQL `DBMS_LOB` package. This technique is used repeatedly in the examples throughout this manual.

## Referencing LOBs Using Java (JDBC)

You can get a reference to any of the preceding LOBs in the following two ways:

- As a column of an `OracleResultSet`
- As an “OUT” type PL/SQL parameter from an `OraclePreparedStatement`

### Using `OracleResultSet`: BLOB and CLOB Objects Retrieved Represent LOB Locators of Current Row

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.

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.

## JDBC Syntax References and Further Information

For further JDBC syntax and information about using JDBC with LOBs:

### See:

- *Oracle9i JDBC Developer's Guide and Reference* for detailed documentation, including parameters, parameter types, return values, and example code.
- *Oracle9i SQLJ Developer's Guide and Reference*
- <http://otn.oracle.com/>
- <http://www.oracle.com/>

## JDBC Methods for Operating on LOBs

The following JDBC methods operate on BLOBs, CLOBs, and BFILES:

- BLOBs:

- To modify BLOB values, see [Table 3-42](#)
- To read or examine BLOB values, see [Table 3-43](#)
- For BLOB buffering, see [Table 3-44](#)
- Temporary BLOBs: Creating, checking if LOB is open, and freeing. See [Table 3-52](#)
- Opening, closing, and checking if BLOB is open, see [Table 3-52](#)
- Trimming BLOBs, see [Table 3-55](#)
- BLOB streaming API, see [Table 3-57](#)
- CLOBs:
  - To read or examine CLOB values, see [Table 3-46](#)
  - For CLOB buffering, see [Table 3-47](#)
  - To modify CLOBs, see [Table 3-57](#)
- Temporary CLOBs:
  - Opening, closing, and checking if CLOB is open, see [Table 3-53](#)
  - Trimming CLOBs, see [Table 3-56](#)
  - CLOB streaming API, see [Table 3-58](#)
- BFILEs:
  - To read or examine BFILEs, see [Table 3-48](#)
  - For BFILE buffering, see [Table 3-49](#)
  - Opening, closing, and checking if CLOB is open, see [Table 3-54](#)
  - BFILE streaming API, see [Table 3-59](#)

## JDBC oracle.sql.BLOB Methods To Modify BLOB Values

**Table 3–42** DBC oracle.sql.BLOB Methods To Modify BLOB Values

| Method                                   | Description   |
|--|---|
| <code>int putBytes(long, byte[ ])</code> | Inserts the byte array into the LOB, starting at the given offset |

## JDBC oracle.sql.BLOB Methods To Read or Examine BLOB Values

**Table 3–43** DBC oracle.sql.BLOB Methods to Read or Examine BLOB Values

| Method   | Description  |
|--|--|
| <code>byte[ ] getBytes(long, int)</code>             | Gets the contents of the LOB as an array of bytes, given an offset |
| <code>long position(byte[],long)</code>              | Finds the given byte array within the LOB, given an offset         |
| <code>long position(Blob,long)</code>                | Finds the given BLOB within the LOB                                |
| <code>public boolean equals(java.lang.Object)</code> | Compares this LOB with another. Compares the LOB locators.         |
| <code>public long length()</code>                    | Returns the length of the LOB                                      |
| <code>public int getChunkSize()</code>               | Returns the <code>ChunkSize</code> of the LOB                      |

## JDBC oracle.sql.BLOB Methods and Properties for BLOB-Buffering

**Table 3–44** *JDBC oracle.sql.BLOB Methods and Properties for BLOB-Buffering*

| Method   | Description                        |
|--|------------------------------------|
| <code>public java.io.InputStream getBinaryStream()</code>        | Streams the LOB as a binary stream |
| <code>public java.io.OutputStream getBinaryOutputStream()</code> | Writes to LOB as a binary stream   |

## JDBC oracle.sql.CLOB Methods To Modify CLOB Values

**Table 3–45** *JDBC oracle.sql.CLOB Methods to Modify CLOB Values*

| Method   | Description  |
|--|--|
| <code>int putString(long, java.lang.String)</code> | Inserts the string into the LOB, starting at the given offset          |
| <code>int putChars(long, char[])</code>            | Inserts the character array into the LOB, starting at the given offset |

## JDBC oracle.sql.CLOB Methods To Read or Examine CLOB Value

**Table 3–46** *JDBC oracle.sql.CLOB Methods To Read or Examine CLOB Values*

| Method  | Description  |
|---|--|
| <code>java.lang.String getSubString(long, int)</code> | Returns a substring of the LOB as a string             |
| <code>int getChars(long, int, char[])</code>          | Reads a subset of the LOB into a character array       |
| <code>long position(java.lang.String, long)</code>    | Finds the given String within the LOB, given an offset |
| <code>long position(oracle.jdbc2.Clob, long)</code>   | Finds the given CLOB within the LOB, given an offset   |
| <code>boolean equals(java.lang.Object)</code>         | Compares this LOB with another                         |
| <code>long length()</code>                            | Returns the length of the LOB                          |
| <code>int getChunkSize()</code>                       | Returns the ChunkSize of the LOB                       |

## JDBC oracle.sql.CLOB Methods and Properties for CLOB-Buffering

**Table 3–47** *JDBC oracle.sql.CLOB Methods and Properties for CLOB-Buffering*

| Method                                      | Description                            |
|---|--|
| java.io.InputStream getAsciiStream()        | Reads the LOB as an ASCII stream       |
| java.io.OutputStream getAsciiOutputStream() | Writes to the LOB from an ASCII stream |
| java.io.Reader getCharacterStream()         | Reads the LOB as a character stream    |
| java.io.Writer getCharacterOutputStream()   | Writes to LOB from a character stream  |

## JDBC oracle.sql.BFILE Methods To Read or Examine External LOB (BFILE) Values

**Table 3–48** *JDBC oracle.sql.BFILE Methods To Read or Examine External LOB (BFILE) Values*

| Method                                | Description  |
|---------------------------------------|--|
| byte[] getBytes(long, int)            | Gets the contents of the BFILE as an array of bytes, given an offset                         |
| int getBytes(long, int, byte[])       | Reads a subset of the BFILE into a byte array  |
| long position(oracle.sql.BFILE, long) | Finds the first appearance of the given BFILE contents within the LOB, from the given offset |
| long position(byte[], long)           | Finds the first appearance of the given byte array within the BFILE, from the given offset   |
| boolean equals(java.lang.Object)      | Compares this BFILE with another. Compares locator bytes.                                    |
| long length()                         | Returns the length of the BFILE  |
| boolean fileExists()                  | Checks if the operating system (OS) file referenced by this BFILE exists                     |
| public void openFile()                | Opens the OS file referenced by this BFILE   |
| 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  |



## JDBC oracle.sql.BFILE Methods and Properties for BFILE-Buffering

**Table 3–49** *JDBC oracle.sql.BFILE Methods and Properties for BFILE-Buffering*

| Method                                       | Description                        |
|--|------------------------------------|
| public java.io.InputStream getBinaryStream() | Reads the BFILE as a binary stream |

## JDBC: OracleBlob and OracleClob Do Not Work in Oracle8i 8.1.x and Higher Releases

OracleBlob and OracleClob were Oracle specific functions used in JDBC 8.0.x drivers to access LOB data. In Oracle8i Releases 8.1.x and higher, OracleBlob and OracleClob are deprecated.

If you use OracleBlob or OracleClob to access LOB data, you will receive the following typical error message, for example, when attempting to manipulate LOBs with Oracle8i Release 8.1.5 JDBC Thin Driver:

```
"Dumping lob java.sql.SQLException: ORA-03115: unsupported network datatype or representation etc."
```

See *Oracle9i JDBC Developer's Guide and Reference* for a description of these non-supported functions and alternative and improved JDBC methods.

For further ideas on working with LOBs with Java, refer to the LOB examples shipped with Oracle9i or see <http://www.oracle.com/>.

## JDBC Temporary LOB APIs

Oracle9i JDBC drivers contain APIs to create and close temporary LOBs. These APIs can replace prior releases' workaround of using the following procedures from the DBMS\_LOB PL/SQL package:

- DBMS\_LOB.createTemporary()
- DBMS\_LOB.isTemporary()
- DBMS\_LOB.freeTemporary()

**Table 3–50 JDBC: Temporary BLOB APIs**

| Methods   | Description   |
|---|---|
| <code>public static BLOB createTemporary(Connection conn, boolean cache, int duration) throws SQLException</code> | Creates a temporary BLOB  |
| <code>public static boolean isTemporary(BLOB blob) throws SQLException</code>                                     | Checks if the specified BLOB locator refers to a temporary BLOB |
| <code>public boolean isTemporary() throws SQLException</code>   | Checks if the current BLOB locator refers to a temporary BLOB   |
| <code>public static void freeTemporary(BLOB temp_blob) throws SQLException</code>                                 | Frees the specified temporary BLOB                              |
| <code>public void freeTemporary() throws SQLException</code>  | Frees the temporary BLOB  |

**Table 3–51 JDBC: Temporary CLOB APIs**

| Methods   | Description   |
|---|---|
| <code>public static CLOB createTemporary(Connection conn, boolean cache, int duration) throws SQLException</code> | Creates a temporary CLOB  |
| <code>public static boolean isTemporary(CLOB clob) throws SQLException</code>                                     | Checks if the specified CLOB locator refers to a temporary CLOB |
| <code>public boolean isTemporary() throws SQLException</code>   | Checks if the current CLOB locator refers to a temporary CLOB   |
| <code>public static void freeTemporary(CLOB temp_clob) throws SQLException</code>                                 | Frees the specified temporary CLOB                              |
| <code>public void freeTemporary() throws SQLException</code>  | Frees the temporary CLOB  |

**See Also:** [Chapter 11, "Temporary LOBs"](#)

## JDBC: Opening and Closing LOBs

`oracle.sql.CLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Clob` interface. [Table 3–51](#) lists the new Oracle extension APIs in `oracle.sql.CLOB` for accessing temporary CLOBs.

Oracle9i JDBC drivers contain APIs to explicitly open and close LOBs. These APIs replace previous techniques that use `DBMS_LOB.open()` and `DBMS_LOB.close()`.

## JDBC: Opening and Closing BLOBs

`oracle.sql.BLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Blob` interface. [Table 3-52](#) lists the Oracle extension APIs in `oracle.sql.BLOB` that open and close BLOBs. These are new for this release.

**Table 3-52 JDBC: Opening and Closing BLOBs**

| Methods   | Description              |
|---|--------------------------|
| <code>public void open(int mode) throws SQLException</code> | Opens the BLOB           |
| <code>public boolean isOpen() throws SQLException</code>    | Sees if the BLOB is open |
| <code>public void close() throws SQLException</code>        | Closes the BLOB          |

### Opening the BLOB

To open a BLOB, your JDBC application can use the `open` method as defined in `oracle.sql.BLOB` class as follows:

```
/**
 * Open a BLOB in the indicated mode. Valid modes include MODE_READONLY,
 * and MODE_READWRITE. It is an error to open the same LOB twice.
 */
public void open (int mode) throws SQLException
```

Possible values of the mode parameter are:

```
public static final int MODE_READONLY
public static final int MODE_READWRITE
```

Each call to `open` opens the BLOB. For example:

```
BLOB blob = ...
blob.open (BLOB.MODE_READWRITE);
```

### Checking If the BLOB is Opened

To see if a BLOB is opened, your JDBC application can use the `isOpen` method defined in `oracle.sql.BLOB`. The return boolean value indicates whether the BLOB has been previously opened or not. The `isOpen` method is defined as follows:

```
/**
 * Check whether the BLOB is opened.
 * @return true if the LOB is opened.
 */
public boolean isOpen () throws SQLException
```

The usage example is:

```
BLOB blob = ...
// See if the BLOB is opened
boolean isOpen = blob.isOpen ();
```

### Closing the BLOB

To close a BLOB, your JDBC application can use the `close` method defined in `oracle.sql.BLOB`. The `close` API is defined as follows:

```
/**
 * Close a previously opened BLOB.
 */
public void close () throws SQLException
```

The usage example is:

```
BLOB blob = ...
// close the BLOB
blob.close ();
```

## JDBC: Opening and Closing CLOBs

Class, `oracle.sql.CLOB`, is the Oracle JDBC driver's implementation of the standard JDBC `java.sql.Clob` interface. [Table 3-53](#) lists the new Oracle extension APIs in `oracle.sql.CLOB` to open and close CLOBs.

**Table 3–53 JDBC: Opening and Closing CLOBs**

| Methods  | Description               |
|--|---------------------------|
| public void open(int mode) throws SQLException | Open the CLOB             |
| public boolean isOpen() throws SQLException    | See if the CLOB is opened |
| public void close() throws SQLException        | Close the CLOB            |

### Opening the CLOB

To open a CLOB, your JDBC application can use the `open` method defined in `oracle.sql.CLOB` class as follows:

```
/**
 * Open a CLOB in the indicated mode. Valid modes include MODE_READONLY,
 * and MODE_READWRITE. It is an error to open the same LOB twice.
 */
public void open (int mode) throws SQLException
```

The possible values of the mode parameter are:

```
public static final int MODE_READONLY
public static final int MODE_READWRITE
```

Each call to `open` opens the CLOB. For example,

```
CLOB clob = ...
clob.open (CLOB.MODE_READWRITE);
```

### Checking if the CLOB is Open

To see if a CLOB is opened, your JDBC application can use the `isOpen` method defined in `oracle.sql.CLOB`. The return boolean value indicates whether the CLOB has been previously opened or not. The `isOpen` method is defined as follows:

```
/**
 * Check whether the CLOB is opened.
 * @return true if the LOB is opened.
 */
public boolean isOpen () throws SQLException
```

The usage example is:

```
CLOB clob = ...
```

```
// See if the CLOB is opened
boolean isOpen = clob.isOpen ();
```

### Closing the CLOB

To close a CLOB, the JDBC application can use the close method defined in `oracle.sql.CLOB`. The close API is defined as follows:

```
/**
 * Close a previously opened CLOB.
 */
public void close () throws SQLException
```

The usage example is:

```
CLOB clob = ...
// close the CLOB
clob.close ();
```

## JDBC: Opening and Closing BFILES

`oracle.sql.BFILE` class wraps the database BFILE object. [Table 3–54](#) lists the new Oracle extension APIs in `oracle.sql.BFILE` for opening and closing BFILES.

**Table 3–54** JDBC API Extensions for Opening and Closing BFILES

| Methods   | Description                 |
|---|-----------------------------|
| <code>public void open() throws SQLException</code>         | Opens the BFILE             |
| <code>public void open(int mode) throws SQLException</code> | Opens the BFILE             |
| <code>public boolean isOpen() throws SQLException</code>    | Checks if the BFILE is open |
| <code>public void close() throws SQLException</code>        | Closes the BFILE            |

### Opening BFILES

To open a BFILE, your JDBC application can use the OPEN method defined in `oracle.sql.BFILE` class as follows:

```
/**
 * Open a external LOB in the readonly mode. It is an error
 * to open the same LOB twice.
```

```

*/
public void open () throws SQLException

/**
 * Open a external LOB in the indicated mode. Valid modes include
 * MODE_READONLY only. It is an error to open the same
 * LOB twice.
 */
public void open (int mode) throws SQLException

```

The only possible value of the mode parameter is:

```
public static final int MODE_READONLY
```

Each call to open opens the BFILE. For example,

```

BFILE bfile = ...
bfile.open ();

```

### Checking if the BFILE is Open

To see if a BFILE is opened, your JDBC application can use the `ISOPEN` method defined in `oracle.sql.BFILE`. The return boolean value indicates whether the BFILE has been previously opened or not. The `ISOPEN` method is defined as follows:

```

/**
 * Check whether the BFILE is opened.
 * @return true if the LOB is opened.
 */
public boolean isOpen () throws SQLException

```

The usage example is:

```

BFILE bfile = ...
// See if the BFILE is opened
boolean isOpen = bfile.isOpen ();

```

### Closing the BFILE

To close a BFILE, your JDBC application can use the `CLOSE` method defined in `oracle.sql.BFILE`. The `CLOSE` API is defined as follows:

```

/**
 * Close a previously opened BFILE.

```

```
*/  
public void close () throws SQLException
```

The usage example is --

```
BFILE bfile = ...  
// close the BFILE  
bfile.close ();
```

### Usage Example (OpenCloseLob.java)

```
/*  
 * This sample shows how to open/close BLOB and CLOB.  
 */  
  
// You need to import the java.sql package to use JDBC  
import java.sql.*;  
  
// You need to import the oracle.sql package to use oracle.sql.BLOB  
import oracle.sql.*;  
  
class OpenCloseLob  
{  
    public static void main (String args [])  
        throws SQLException  
    {  
        // Load the Oracle JDBC driver  
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
  
        String url = "jdbc:oracle:oci8:@";  
        try {  
            String url1 = System.getProperty("JDBC_URL");  
            if (url1 != null)  
                url = url1;  
        } catch (Exception e) {  
            // If there is any security exception, ignore it  
            // and use the default  
        }  
  
        // Connect to the database  
        Connection conn =  
            DriverManager.getConnection (url, "scott", "tiger");  
        // It's faster when auto commit is off  
        conn.setAutoCommit (false);
```



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

try
{
    stmt.execute ("drop table basic_lob_table");
}
catch (SQLException e)
{
    // An exception could be raised here if the table did not exist already.
}

// Create a table containing a BLOB and a CLOB
stmt.execute ("create table basic_lob_table (x varchar2 (30), b blob, c clob)");

// Populate the table
stmt.execute (
    "insert into basic_lob_table values"
    + " ('one', '01010101010101010101010101010101', 'onetwothreefour)");

// Select the lob
ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
while (rset.next ())
{
    // Get the lob
    BLOB blob = (BLOB) rset.getObject (2);
    CLOB clob = (CLOB) rset.getObject (3);

    // Open the lob
    System.out.println ("Open the lob");
    blob.open (BLOB.MODE_READWRITE);
    clob.open (CLOB.MODE_READWRITE);

    // Check if the lob is opened
    System.out.println ("blob.isOpen()="+blob.isOpen());
    System.out.println ("clob.isOpen()="+clob.isOpen());

    // Close the lob
    System.out.println ("Close the lob");
    blob.close ();
    clob.close ();

    // Check if the lob is opened
    System.out.println ("blob.isOpen()="+blob.isOpen());
```

```
        System.out.println ("clob.isOpen()="+clob.isOpen());
    }

    // Close the ResultSet
    rset.close ();

    // Close the Statement
    stmt.close ();

    // Close the connection
    conn.close ();
}
}
```

## Trimming LOBs using JDBC

Oracle9i JDBC drivers contain APIs to trim internal LOBs. These APIs replace previous techniques that used `DBMS_LOB.trim()`.

### JDBC: Trimming BLOBs

`oracle.sql.BLOB` class is Oracle JDBC driver's implementation of the standard JDBC `java.sql.Blob` interface. [Table 3-55](#) lists the new Oracle extension API in `oracle.sql.BLOB` that trims BLOBs.

**Table 3-55** *JDBC: Trimming BLOBs*

| Methods  | Description    |
|--|----------------|
| <code>public void trim(long newlen) throws SQLException</code> | Trims the BLOB |

The trim API is defined as follows:

```
/**
 * Trim the value of the BLOB to the length you specify in the newlen parameter.
 * @param newlen the new length of the BLOB.
 */
public void trim (long newlen) throws SQLException
```

The `newlen` parameter specifies the new length of the BLOB.

## JDBC: Trimming CLOBs

`oracle.sql.CLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Clob` interface. [Table 3-56](#) lists the new Oracle extension API in `oracle.sql.CLOB` that trims CLOBs.

**Table 3-56** *JDBC: Trimming CLOBs*

| Methods  | Description    |
|--|----------------|
| <code>public void trim(long newlen) throws SQLException</code> | Trims the CLOB |

The trim API is defined as follows:

```
/**
 * Trim the value of the CLOB to the length you specify in the newlen parameter.
 * @param newlen the new length of the CLOB.
 */
public void trim (long newlen) throws SQLException
```

The `newlen` parameter specifies the new length of the CLOB.

**See:** [Chapter 10, "Internal Persistent LOBs" "Java \(JDBC\): Trimming LOB Data"](#) on page 10-236, for an example.

## JDBC: New LOB Streaming APIs

Oracle9i JDBC drivers contains the new LOB streaming APIs to read from/write to a LOB at the requested position from a Java stream. In prior releases, LOB streaming APIs did not specify the offset.

### New JDBC BLOB Streaming APIs

`oracle.sql.BLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Blob` interface. [Table 3-57](#) lists the new Oracle extension APIs in `oracle.sql.BLOB` that manipulate the BLOB content from the requested position.

**Table 3-57** *JDBC: New BLOB Streaming APIs*

| Methods   | Description                      |
|---|----------------------------------|
| <code>public java.io.OutputStream getBinaryOutputStream (long pos) throws SQLException</code> | Writes to the BLOB from a stream |
| <code>public java.io.InputStream getBinaryStream(long pos) throws SQLException</code>         | Reads from the BLOB as a stream  |

These APIs are defined as follows:

```

/**
 * Write to the BLOB from a stream at the requested position.
 *
 * @param pos is the position data to be put.
 * @return a output stream to write data to the BLOB
 */
public java.io.OutputStream getBinaryOutputStream(long pos) throws SQLException

/**
 * Read from the BLOB as a stream at the requested position.
 *
 * @param pos is the position data to be read.
 * @return a output stream to write data to the BLOB
 */
public java.io.InputStream getBinaryStream(long pos) throws SQLException

```

## New CLOB Streaming APIs

`oracle.sql.CLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Clob` interface. [Table 3-58](#) lists the new Oracle extension APIs in `oracle.sql.CLOB` that manipulate the CLOB content from the requested position.

**Table 3-58 JDBC: New CLOB Streaming APIs**

| Methods  | Description                                |
|--|--|
| <code>public java.io.OutputStream getAsciiOutputStream (long pos) throws SQLException</code> | Writes to the CLOB from an ASCII stream    |
| <code>public java.io.Writer getCharacterOutputStream(long pos) throws SQLException</code>    | Writes to the CLOB from a character stream |
| <code>public java.io.InputStream getAsciiStream(long pos) throws SQLException</code>         | Reads from the CLOB as an ASCII stream     |
| <code>public java.io.Reader getCharacterStream(long pos) throws SQLException</code>          | Reads from the CLOB as a character stream  |

These APIs are defined as follows:

```

/**
 * Write to the CLOB from a stream at the requested position.
 * @param pos is the position data to be put.

```

```

        * @return a output stream to write data to the CLOB
        */
public java.io.OutputStream getAsciiOutputStream(long pos) throws
SQLException

/**
 * Write to the CLOB from a stream at the requested position.
 * @param pos is the position data to be put.
 * @return a output stream to write data to the CLOB
 */
public java.io.Writer getCharacterOutputStream(long pos) throws SQLException

/**
 * Read from the CLOB as a stream at the requested position.
 * @param pos is the position data to be put.
 * @return a output stream to write data to the CLOB
 */
public java.io.InputStream getAsciiStream(long pos) throws SQLException

/**
 * Read from the CLOB as a stream at the requested position.
 * @param pos is the position data to be put.
 * @return a output stream to write data to the CLOB
 */
public java.io.Reader getCharacterStream(long pos) throws SQLException

```

## New BFILE Streaming APIs

`oracle.sql.BFILE` class wraps the database BFILES. [Table 3-59](#) lists the new Oracle extension APIs in `oracle.sql.BFILE` that reads BFILE content from the requested position.

**Table 3-59 JDBC: New BFILE Streaming APIs**

| Methods   | Description                      |
|---|----------------------------------|
| <code>public java.io.InputStream getBinaryStream(long pos) throws SQLException</code> | Reads from the BFILE as a stream |

These APIs are defined as follows:

```

/**
 * Read from the BLOB as a stream at the requested position.
 *
 * @param pos is the position data to be read.

```

```
* @return a output stream to write data to the BLOB
*/
public java.io.InputStream getBinaryStream(long pos) throws SQLException
```

## JDBC BFILE Streaming Example (NewStreamLob.java)

---

---

**Note:** Some of the Java code strings (in quotes) in the example should appear on one line, but instead, they wrap to the next lines. For example, the `stmt.execute` lines. Be aware of this if you are using this code and ensure that the strings appear on one line.

---

---

```
/*
 * This sample shows how to read/write BLOB and CLOB as streams.
 */

import java.io.*;

// You need to import the java.sql package to use JDBC
import java.sql.*;

// You need to import the oracle.sql package to use oracle.sql.BLOB
import oracle.sql.*;

class NewStreamLob
{
    public static void main (String args []) throws Exception
    {
        // Load the Oracle JDBC driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        String url = "jdbc:oracle:oci8:@";
        try {
            String url1 = System.getProperty("JDBC_URL");
            if (url1 != null)
                url = url1;
        } catch (Exception e) {
            // If there is any security exception, ignore it
            // and use the default
        }
    }
}
```

```
// Connect to the database
Connection conn =
    DriverManager.getConnection (url, "scott", "tiger");
// It's faster when auto commit is off
conn.setAutoCommit (false);

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

try
{
    stmt.execute ("drop table basic_lob_table");
}
catch (SQLException e)
{
    // An exception could be raised here if the table did not exist already.
}

// Create a table containing a BLOB and a CLOB
stmt.execute (
    "create table basic_lob_table"
    + "(x varchar2 (30), b blob, c clob)");

// Populate the table
stmt.execute (
    "insert into basic_lob_table values"
    + "('one', '01010101010101010101010101010101', 'onetwothreefour')");

System.out.println ("Dumping lob");

// Select the lob
ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
while (rset.next ())
{
    // Get the lob
    BLOB blob = (BLOB) rset.getObject (2);
    CLOB clob = (CLOB) rset.getObject (3);

    // Print the lob contents
    dumpBlob (conn, blob, 1);
    dumpClob (conn, clob, 1);

    // Change the lob contents
    fillClob (conn, clob, 11, 50);
    fillBlob (conn, blob, 11, 50);
}
```

```
    }
    rset.close ();

    System.out.println ("Dumping lobes again");

    rset = stmt.executeQuery ("select * from basic_lob_table");
    while (rset.next ())
    {
        // Get the lobes
        BLOB blob = (BLOB) rset.getObject (2);
        CLOB clob = (CLOB) rset.getObject (3);

        // Print the lobes contents
        dumpBlob (conn, blob, 11);
        dumpClob (conn, clob, 11);
    }
    // Close all resources
    rset.close();
    stmt.close();
    conn.close();
}

// Utility function to dump Clob contents
static void dumpClob (Connection conn, CLOB clob, long offset)
    throws Exception
{
    // get character stream to retrieve clob data
    Reader instream = clob.getCharacterStream(offset);

    // create temporary buffer for read
    char[] buffer = new char[10];

    // length of characters read
    int length = 0;

    // fetch data
    while ((length = instream.read(buffer)) != -1)
    {
        System.out.print("Read " + length + " chars: ");

        for (int i=0; i<length; i++)
            System.out.print(buffer[i]);
        System.out.println();
    }
}
```



```
// Close input stream
instream.close();
}

// Utility function to dump Blob contents
static void dumpBlob (Connection conn, BLOB blob, long offset)
    throws Exception
{
    // Get binary output stream to retrieve blob data
    InputStream instream = blob.getBinaryStream(offset);
    // Create temporary buffer for read
    byte[] buffer = new byte[10];
    // length of bytes read
    int length = 0;
    // Fetch data
    while ((length = instream.read(buffer)) != -1)
    {
        System.out.print("Read " + length + " bytes: ");

        for (int i=0; i<length; i++)
            System.out.print(buffer[i]+" ");
        System.out.println();
    }

    // Close input stream
    instream.close();
}

// Utility function to put data in a Clob
static void fillClob (Connection conn, CLOB clob, long offset, long length)
    throws Exception
{
    Writer outstream = clob.getCharacterOutputStream(offset);

    int i = 0;
    int chunk = 10;

    while (i < length)
    {
        outstream.write("aaaaaaaaaa", 0, chunk);

        i += chunk;
        if (length - i < chunk)
            chunk = (int) length - i;
    }
}
```

```
        outstream.close();
    }

    // Utility function to put data in a Blob
    static void fillBlob (Connection conn, BLOB blob, long offset, long length)
        throws Exception
    {
        OutputStream outstream = blob.getBinaryOutputStream(offset);

        int i = 0;
        int chunk = 10;

        byte [] data = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

        while (i < length)
        {
            outstream.write(data, 0, chunk);

            i += chunk;
            if (length - i < chunk)
                chunk = (int) length - i;
        }
        outstream.close();
    }
}
```

## JDBC and Empty LOBs

An empty BLOB can be created from the following API from `oracle.sql.BLOB`:

```
public static BLOB empty_lob () throws SQLException
```

Similarly, the following API from `oracle.sql.CLOB` creates a empty CLOB:

```
public static CLOB empty_lob () throws SQLException
```

Empty LOB instances are created by JDBC drivers without making database round trips. Empty LOBs can be used in the following cases:

- “set” APIs of `PreparedStatement`
- “update” APIs of updatable result set
- attribute value of `STRUCTs`
- element value of `ARRAYs`

---

---

**Note:** Empty LOBs are special marker LOBs but not real LOB values.

---

---

JDBC applications cannot read or write to empty LOBs created from the preceding APIs. An ORA-17098 “Invalid empty lob operation” results if your application attempts to read/write to an empty LOB.

## OLEDB (Oracle Provider for OLEDB — OraOLEDB)

Oracle Provider for OLE DB (OraOLEDB) offers high performance and efficient access to Oracle data for OLE DB and ADO developers. Developers programming with Visual Basic, C++, or any COM client can use OraOLEDB to access Oracle databases.

OraOLEDB is an OLE DB provider for Oracle. It offers high performance and efficient access to Oracle data including LOBs, and also allows updates to certain LOB types.

The following LOB types are supported by OraOLEDB:

- *For Persistent LOBs.* READ/WRITE through the rowset.
- *For BFILES.* READ-ONLY through the rowset.
- *Temporary LOBs* are not supported through the rowset.

**See Also:**

- [Chapter 13, "Using OraOLEDB to Manipulate LOBs"](#)
- The *Oracle Provider for OLE DB User's Guide* at: <http://otn.oracle.com/>.

---

# Managing LOBs

This chapter describes the following topics:

- Rules for using Directory Objects and BFILES
- DBA Actions Required Prior to Working with LOBs
- Managing Temporary LOBs
- Using SQL\*Loader to Load LOBs
- Inline versus Out-of-Line LOBs
- Loading InLine and Out-Of-Line Data into Internal LOBs Using SQL\*Loader
- Loading Inline LOB Data
- Loading Out-Of-Line LOB Data
- SQL\*Loader LOB Loading Tips
- LOB Restrictions

## Rules for using Directory Objects and BFILES

When creating a directory object or BFILES, ensure that the following conditions are met:

- The operating system file must not be a symbolic or hard link.
- The operating system directory path named in the Oracle DIRECTORY object must be an existing OS directory path.
- The operating system directory path named in the Oracle DIRECTORY object should not contain any symbolic links in its components.

## DBA Actions Required Prior to Working with LOBs

This section describes actions that you or your database administrator must take prior to working with LOBs.

### Set Maximum Number of Open BFILES

A limited number of BFILES can be open simultaneously in each 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 in each 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 reaches 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 `DBMS_LOB.FILECLOSEALL` call.

### Using SQL DML for Basic Operations on LOBs

SQL Data Manipulation Language (DML) includes basic operations, such as, `INSERT`, `UPDATE`, `DELETE` — that let you make changes to the entire value of *internal* LOBs within Oracle RDBMS.

- *Internal LOBs*: To work with parts of internal LOBs, you will need to use one of the interfaces described in [Chapter 3, "LOB Support in Different Programmatic Environments"](#), that have been developed to handle more complex requirements. Alternatively, you can perform string operations on CLOBs using SQL `VARCHAR2` functions.

**See Also:** [Chapter 7, "Modeling and Design"](#)

For use case examples refer to the following sections in [Chapter 10, "Internal Persistent LOBs"](#):

- `INSERT`:
  - \* ["Inserting a LOB Value using `EMPTY\_CLOB\(\)` or `EMPTY\_BLOB\(\)`"](#) on page 10-16

- \* ["Inserting a Row by Selecting a LOB From Another Table"](#) on page 10-20
- \* ["Inserting a Row by Initializing a LOB Locator Bind Variable"](#) on page 10-23
- UPDATE:
  - \* ["Updating a LOB with EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)"](#) on page 10-271
  - \* ["Updating a Row by Selecting a LOB From Another Table"](#) on page 10-275
  - \* ["Updating by Initializing a LOB Locator Bind Variable"](#) on page 10-277
- DELETE:
  - \* ["Deleting the Row of a Table Containing a LOB"](#) on page 10-286
- *External LOBs (BFILES)*: Oracle8i supports read-only operations on external LOBs. See [Chapter 12, "External LOBs \(BFILES\)"](#):
  - INSERT:
    - \* ["INSERT a Row Using BFILENAME\(\)"](#) on page 12-22
    - \* ["INSERT a BFILE Row by Selecting a BFILE From Another Table"](#) on page 12-31
    - \* ["Inserting a Row With BFILE by Initializing a BFILE Locator"](#) on page 12-33
  - UPDATE: You can use the following methods to UPDATE or 'write to' a BFILE:
    - \* ["Updating a BFILE Using BFILENAME\(\)"](#) on page 12-193
    - \* ["Updating a BFILE by Selecting a BFILE From Another Table"](#) on page 12-196
    - \* ["Updating a BFILE by Initializing a BFILE Locator"](#) on page 12-198
  - DELETE:
    - \* ["Deleting the Row of a Table Containing a BFILE"](#) on page 12-231

## Changing Tablespace Storage for a LOB

It is possible to change the default storage for a LOB after the table has been created.

### **Oracle8 Release 8.0.4.3**

To move the CLOB column from tablespace A to tablespace B, in Oracle8 release 8.0.4.3, requires the following statement:

```
ALTER TABLE test lob(test) STORE AS (tablespace tools);
```

However, this returns the following error message:

```
ORA-02210: no options specified for ALTER TABLE
```

### **Oracle8i and Oracle9i**

- **Using ALTER TABLE... MODIFY:** You can change LOB tablespace storage as follows:

---

---

**Note:**

- The ALTER TABLE syntax for modifying an existing LOB column uses the MODIFY LOB clause, not the LOB...STORE AS clause. The LOB...STORE AS clause is only for newly added LOB columns.
  - There are two kinds of LOB storage clauses: LOB\_storage\_clause and modify\_LOB\_storage\_clause. In the ALTER TABLE MODIFY LOB statement, you can only specify modify\_LOB\_storage\_clause.
- 
- 

```
ALTER TABLE test MODIFY
LOB (lob1)
  STORAGE (
    NEXT          4M
    MAXEXTENTS   100
    PCTINCREASE   50
  )
```

- **Using ALTER TABLE... MOVE:** You can also use the MOVE clause of the ALTER TABLE statement to change LOB tablespace storage. For example:

```
ALTER TABLE test MOVE
  TABLESPACE tbs1
  LOB (lob1, lob2)
  STORE AS (
    TABLESPACE tbs2
    DISABLE STORAGE IN ROW);
```



## Managing Temporary LOBs

Management and security issues of temporary LOBs are discussed in [Chapter 11, "Temporary LOBs"](#).

- [Managing Temporary LOBs](#) on page 11-10
- [Security Issues with Temporary LOBs](#) on page 11-10

## Using SQL\*Loader to Load LOBs

You can use SQL\*Loader to bulk load LOBs. See “Loading LOBs” in *Oracle9i Database Utilities* for details on using SQL\*Loader control file data definition language (DDL) to load LOB types.

Data loaded into LOBs can be lengthy and it is likely that you will want to have the data out- of-line from the rest of the data. LOBFILES provide a method to separate lengthy data.

## LOBFILES

LOBFILES are simple datafiles that facilitate LOB loading. LOBFILES are distinguished from primary datafiles in that in LOBFILES there is no concept of a *record*. In LOBFILES the data is of any of the following types:

- Predetermined size fields (fixed length fields)
- Delimited fields, for example, TERMINATED BY or ENCLOSED BY

---

---

**Note:** The clause PRESERVE BLANKS is not applicable to fields read from a LOBFILE.

---

---

- Length-Value pair fields (variable length fields) -- VARRAW, VARCHAR, or VARCHARC loader datatypes are used for loading from this type of fields.
- A single LOB field into which the entire contents of a file can be read.

---

---

**Note:** A field read from a LOBFILE cannot be used as an argument to a clause (for example, the NULLIF clause).

---

---

## **Inline versus Out-of-Line LOBs**

Inline LOBs are LOBs whose value comes from the primary data file.

Out-of-Line LOBs are LOBs whose value comes from LOBFILES.

## Loading InLine and Out-Of-Line Data into Internal LOBs Using SQL\*Loader

The following sections describe procedures for loading differently formatted inline and out-of-line data into internal LOBs:

- Loading InLine LOB Data
  - [Loading Inline LOB Data in Predetermined Size Fields](#)
  - [Loading Inline LOB Data in Delimited Fields](#)
  - [Loading Inline LOB Data in Length-Value Pair Fields](#)
- Loading Out-Of-Line LOB Data
  - [Loading One LOB Per File](#)
  - [Loading Out-of-Line LOB Data in Predetermined Size Fields](#)
  - [Loading Out-of-Line LOB Data in Delimited Fields](#)
  - [Loading Out-of-Line LOB Data in Length-Value Pair Fields](#)

Other topics discussed are

- [SQL\\*Loader LOB Loading Tips](#)

### SQL\*Loader Performance: Loading Into Internal LOBs

See [Table 4–1, "SQL\\*Loader Performance: Loading Data Into Internal LOBs"](#) for the relative performance when using the preceding methods of loading data into internal LOBs.

**Table 4–1 SQL\*Loader Performance: Loading Data Into Internal LOBs**

| Loading Method For In-Line or Out-Of-Line Data | Relative Performance |
|--|----------------------|
| In Predetermined Size Fields                   | Highest              |
| In Delimited Fields                            | Slower               |
| In Length Value-Pair Fields                    | High                 |
| One LOB Per File                               | High                 |

**See Also:** [Chapter 9, "LOBs: Best Practices"](#)

## Loading Inline LOB Data

- [Loading Inline LOB Data in Predetermined Size Fields](#)
- [Loading Inline LOB Data in Delimited Fields](#)
- [Loading Inline LOB Data in Length-Value Pair Fields](#)

### Loading Inline LOB Data in Predetermined Size Fields

This is a very fast and simple way to load LOBs. Unfortunately, the LOBs to be loaded are not usually the same size.

---

---

**Note:** A possible work-around is to pad LOB data with white space to make all LOBs the same length within the particular datafield; for information on trimming of trailing white spaces see “Trimming of Blanks and Tabs” in *Oracle9i Database 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
APPEND
  (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 15 bytes 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 only if the NULLIF clause is used instead of the DEFAULTIF clause. You can use loader datatypes other than CHAR to load LOBs. Use the RAW datatype when loading BLOBs.

---



---

**Note:** You can specify both NULLIF and DEFAULTIF for the same field, although NULLIF has a higher 'priority' than DEFAULTIF.

---



---

## Loading Inline LOB Data in Delimited Fields

Loading different size LOBs in 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 '<endrec>\n'"
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>' '<endrec>'
008,    <startlob>    Once upon another time ....The end. '<endlob>' '<endrec>'
```

## Loading Inline LOB Data in Length-Value Pair Fields

You could use VARCHAR (see *Oracle9i Database 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. For example:

### Control File

```
LOAD DATA
INFILE 'sample2.dat' "str '<endrec>\r\n'"
INTO TABLE Multimedia_tab
APPEND
FIELDS TERMINATED BY ','
```

```
(  
Clip_ID  INTEGER EXTERNAL (3),  
  Story  VARCHAR (3, 500)  
)
```

### Data File (sample2.dat)

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

---

---

#### Note:

- `Story` is a field corresponding to a CLOB column. In the control file, it is described as a VARCHAR (3, 500) whose length field is 3 bytes long and maximum size is 500 bytes. This tells the Loader that it can find the length of the LOB data in the first 3 bytes.
  - The length subfield of the VARCHAR 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 preceding data file is a line feed.
  - When loading a data stream where a record is terminated by a newline (as on Unix platforms), or a carriage-return and newline (as on Windows platforms), you must specify `\n` (for Unix) or `\r\n` (for Windows) to ensure that these characters are not interpreted as the start of the next record.
- 
- 

## Loading Out-Of-Line LOB Data

This section describes the following topics:

- [Loading One LOB Per File](#)
- [Loading Out-of-Line LOB Data in Predetermined Size Fields](#)
- [Loading Out-of-Line LOB Data in Delimited Fields](#)
- [Loading Out-of-Line LOB Data in Length-Value Pair Fields](#)

As mentioned earlier, LOB data can be so large that it is reasonable to want to load it from secondary datafile(s).

In `LOBFILES`, `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.

## Loading One LOB Per File

Each `LOBFILE` contains a single `LOB`. For example:

### 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:**

- `STORY` tells the Loader that it can find the LOB data in the file whose name is stored in the `ext_FileName` field.
  - `TERMINATED BY EOF` tells the Loader that the LOB will span the entire file.
  - `SQL*Plus` users: Viewing CLOB data using the `SQL*Plus long` parameter displays only the first 80 characters of the CLOB because the default value of the `long` parameter is 80.
  - See also *Oracle9i Database Utilities*
- 

## Loading Out-of-Line LOB Data in Predetermined Size Fields

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 column is assumed to be 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 the LOBs are the same size. For example:

### 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:** SQL \*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.

---



---

## Loading Out-of-Line LOB Data in Delimited Fields

LOB data instances in 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>\n")
```

Windows users: Terminate the control file using:

```
TERMINATED BY "<endlob>\r\n")
```

### 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:**

- The `TERMINATED BY` clause specifies the string that terminates the LOBs.
  - When loading a data stream where a record is terminated by a newline (as on Unix platforms), or a carriage-return and newline (as on Windows platforms), you must specify `\n` (for Unix) or `\r\n` (for Windows) to ensure that these characters are not interpreted as the start of the next record.
- 

## Loading Out-of-Line LOB Data in Length-Value Pair Fields

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 performs better than 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. For example:

### 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:** `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 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 bytes belong to the specified LOB.
  - `0000` results in an empty LOB (not a NULL LOB).
- 
- 

## SQL\*Loader LOB Loading Tips

- For SQL\*Loader conventional path loads, failure to load a particular LOB does not result in the rejection of the record containing that LOB; instead, the record ends up containing an empty LOB.

For SQL\*Loader direct-path loads, the LOB could be *empty* or *truncated*. LOBs are sent in pieces to the server for loading. If there is an error, the LOB piece with the error is discarded and the rest of that LOB is not loaded. In other words, if the entire LOB with the error is contained in the first piece, then that LOB column will either be empty or truncated.

- When loading from LOBFILES specify the maximum length of the field corresponding to a LOB-type column. If the maximum length is specified, it is taken as a hint to help optimize memory usage. It is important that the maximum length specification does not underestimate the true maximum length.
- When using SQL\*Loader direct-path load, loading LOBs can take up substantial memory. If the message "SQL\*Loader 700 (out of memory)" appears when loading LOBs, internal code is probably batching up more rows in each load call that which can be supported by your operating system and process memory. A workaround is to use the `ROWS` option to read a smaller number of rows in each data save.
- You can also use the Direct Path API to load LOBs.

### See Also:

- *Oracle9i Database Utilities* Chapters 7 and 9.
- [Chapter 9, "LOBs: Best Practices", Using SQL\\*Loader.](#)

## LOB Restrictions

This section provides details on LOB restrictions.

**See Also:**

- ["What's New with Large Objects \(LOBs\)?"](#) on page xlv for restrictions removed in a given release.
- [Chapter 5, "Large Objects: Advanced Topics", "Restrictions for LOBs in Partitioned Index-Organized Tables"](#) on page 5-31.
- [Chapter 8, "Migrating From LONGs to LOBs" under "LONG-to-LOB Migration Limitations"](#) on page 8-10, describes LONG to LOB migration limitations for clustered tables, replication, triggers, domain indexes, and functional indexes.

LOB columns are subject to the following restrictions:

- Distributed LOBs are not supported. Therefore, you cannot use a remote locator in `SELECT` or `WHERE` clauses of queries or in functions of the `DBMS_LOB` package.

The following syntax is not supported for LOBs:

```
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;
```

(This statement produces error: ORA-22992 cannot use LOB locators selected from remote tables.)

However, you can use a remote locator in others parts of queries that reference LOBs. The following syntax is supported on remote LOB columns:

```
CREATE TABLE t 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_site ...
UPDATE table1@remote_site ...
DELETE FROM table1@remote_site ...
```

For the first three types of statement, which contain subqueries, only standalone LOB columns are allowed in the select list. SQL functions or `DBMS_LOB` APIs on LOBs are not supported. For example, the following statement is supported:

```
CREATE TABLE AS SELECT clob_col FROM tab@dbs2;
```

However, the following statement is not supported:

```
CREATE TABLE AS SELECT dbms_lob.substr(clob_col) from tab@dbs2;
```

- Clusters cannot contain LOBs, either as key or non-key columns. This produces error, ORA-02335: invalid datatype for cluster column.
- You cannot create a VARRAY of LOBs. This produces error, ORA-02348: cannot create VARRAY column with embedded LOB.
- You cannot specify LOB columns in the ORDER BY clause of a query, or in the GROUP BY clause of a query or in an aggregate function. This produces error, ORA-00932: inconsistent datatypes.
- You cannot specify a LOB column in a SELECT... DISTINCT or SELECT... UNIQUE statement or in a join. However, you can specify a LOB attribute of an object type column in a SELECT... DISTINCT statement or in a query that uses the UNION or MINUS set operator if the column's object type has a MAP or ORDER function defined on it.
- You cannot specify an NCLOB as an attribute of an object type when creating a table. However, you can specify NCLOB parameters in methods.
- You cannot specify LOB columns in ANALYZE... COMPUTE or ANALYZE... ESTIMATE statements.
- You cannot define an UPDATE DML trigger on a LOB column.
- You cannot specify a LOB as a primary key column.
- You cannot specify a LOB column as part of an index key. However, you can specify a LOB column in the function of a function-based index or in the indextype specification of a domain index. In addition, Oracle Text lets you define an index on a CLOB column.

**See Also:** See *Oracle9i Data Cartridge Developer's Guide* for more information about defining triggers on domain indexes

- In an INSERT or UPDATE operation, you can bind data of any size to a LOB column, but you cannot bind data to a LOB attribute of an object type. In an INSERT... AS SELECT operation, you can bind up to 4000 bytes of data to LOB columns.

**See Also:**

- [Chapter 7, "Modeling and Design", "SQL Semantics Support for LOBs"](#) on page 7-33.
  - The “Keywords and Parameters” section of individual SQL statements in *Oracle9i SQL Reference* for additional semantics for the use of LOBs
- If a table has both `LONG` and `LOB` columns, you cannot bind more than 4000 bytes of data to both the `LONG` and `LOB` columns in the same SQL statement. However, you can bind more than 4000 bytes of data to either the `LONG` or the `LOB` column.
  - *First Extent of Any LOB Segment Must be At Least 3 Blocks.* The first extent (or initial extent) of a `LOB` segment requires at least 3 blocks. This restriction has existed since `LOBs` were implemented in Oracle Release 8.0.

The first extent of any segment requires *at least 2 blocks* (if `FREELIST GROUPS` was 0). That is, the initial extent size of the segment should be at least 2 blocks. `LOBs` segments are different because they need *at least 3 blocks* in the first extent. If you try to create a `LOB` segment in a permanent dictionary managed tablespace with `initial = 2` blocks, it will still work because it is possible for segments in permanent dictionary managed tablespaces to override tablespaces' default storage setting.

But if uniform locally managed tablespaces or dictionary managed tablespaces of the temporary type, or locally managed temporary tablespaces have an extent size of 2 blocks, `LOB` segments cannot be created in these tablespaces. This is because in these tablespace types, extent sizes are fixed and tablespaces' default storage setting is not ignored.

You will get a message on trying to create the `LOB` segment: `ORA-3237 "initial extent of specified size cannot be allocated"`. You could be confused by this error, especially if your tablespace has lots of free space.

- *BFILES in Shared Server (Multi-Threaded Server — MTS) Mode.* Oracle9i does not support session migration for `BFILES` in Shared Server (Multi-Threaded Server — MTS) mode. This implies that operations on open `BFILES` can persist beyond the end of a call to a shared server. In shared server sessions, `BFILE` operations will be bound to one shared server, they cannot migrate from one server to another. This restriction will be removed in a forthcoming release.

---

---

**Notes:**

- Oracle8i Release 2 (8.1.6) and higher support the `CACHE READS` setting for LOBs. If you have such LOBs and you downgrade to an earlier release, Oracle generates a warning and converts the LOBs from `CACHE READS` to `CACHE LOGGING`. You can subsequently alter the LOBs to either `NOCACHE LOGGING` or `NOCACHE NOLOGGING`. For more information see [Chapter 7, "Modeling and Design"](#), [Chapter , "CACHE / NOCACHE / CACHE READS"](#) on page 7-8. For a table on which you have defined a DML trigger, if you use OCI functions or `DBMS_LOB` routines to change the value of a LOB column or the LOB attribute of an object type column, Oracle does not fire the DML trigger.
- 
- 

Other general LOB restrictions include the following:

- LOBs are only supported in range partitioned index-organized tables. See ["Partitioned Index-Organized Tables and LOBs"](#) on page 1-6 for more information.
- Client PL/SQL procedures cannot call `DBMS_LOB` routines. See ["PL/SQL - LOB Guidelines"](#) on page 3-7 for more information.
- When using `DBMS_LOB.LOADFROMFILE`, you cannot specify an `amount` parameter value larger than the size of the BFILE. See ["PL/SQL - LOB Guidelines"](#) on page 3-7 for more information.
- When using `DBMS_LOB.READ`, the `amount` parameter can be larger than the size of the data. See ["PL/SQL - LOB Guidelines"](#) on page 3-7 for more information.
- A field read from a LOBFILE cannot be used as an argument to a clause. See ["LOBFILES"](#) on page 4-5 for more information.

**See Also:**

- ["SQL\\*Loader LOB Loading Tips"](#) on page 4-15.
- [Chapter 5, "Large Objects: Advanced Topics"](#), ["Restrictions for LOBs in Partitioned Index-Organized Tables"](#) on page 5-31.





---

---

# Large Objects: Advanced Topics

This chapter contains the following sections:

- [Introducing Large Objects: Advanced Topics](#)
- [Read Consistent Locators](#)
- [LOB Locators and Transaction Boundaries](#)
- [LOBs in the Object Cache](#)
- [LOB Buffering Subsystem](#)
- [Creating a Varray Containing References to LOBs](#)
- [LOBs in Partitioned Index-Organized Tables](#)
- [Restrictions for LOBs in Partitioned Index-Organized Tables](#)

*Note: Examples in this chapter are based on the multimedia schema and table `Multimedia_tab` described in [Appendix B, "The Multimedia Schema"](#).*

## Introducing Large Objects: 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

Oracle provides the same read consistency mechanisms for LOBs as for all other database reads and updates of scalar quantities. Refer to *Oracle9i Database Concepts* for general information about read consistency. Read consistency has some special

applications to LOB locators that you must understand. These applications are described in the following sections.

## A Selected Locator Becomes a Read Consistent Locator

A `SELECT`d 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, note the following:

- The LOB is read as of the point in time of the `SELECT` statement even if the `SELECT` statement includes a `FOR UPDATE`.
- If the LOB value is updated through a different locator (L2) in the same transaction, L1 does not see L2's updates.
- L1 will not see committed updates made to the LOB through *another* transaction.
- 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.

## Updating LOBs and Read-Consistency

Read consistent locators provide the same LOB value regardless of when the `SELECT` occurs.

The following example demonstrates the relationship between read-consistency and updating in a simple example. Using `Multimedia_tab`, as defined in [Appendix B, "The Multimedia Schema"](#), and PL/SQL, three CLOBs are created as potential locators:

- `clob_selected`
- `clob_update`
- `clob_copied`

Observe these progressions in the code, from times t1 through t6:

- 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

```
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;
/

```

## Updating LOBs Via 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 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 using 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 any of the available methods, such as OCI LOB APIs or PL/SQL DBMS\_LOB package, updates the LOB value *and then reselects* 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 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*.

**See Also:** *PL/SQL User's Guide and Reference*

## Example of Updating a LOB Using SQL DML and DBMS\_LOB

Using table `Multimedia_tab` as defined previously, a CLOB locator is created:

- `clob_selected`.

Note the following progressions in the following example PL/SQL (DBMS\_LOB) code, from times t1 through t3:

- 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 statement, 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 using the SQL UPDATE statement. 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 statement. Therefore, in the next DBMS\_LOB.READ(), an error is returned because the LOB value is empty, that is, it does not contain any data.

### Example

```
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;
/
```

## Example of Using One Locator to Update the Same LOB Value

---

---

**Note:** *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 table *Multimedia\_tab* as defined previously, two CLOBs are created as potential locators:

- `clob_updated`
- `clob_copied`

Note these progressions in the following example PL/SQL (DBMS\_LOB) code at times t1 through t5:

- 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

```
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;
/
```

## Example of Updating a LOB with a PL/SQL (DBMS\_LOB) Bind Variable

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
- clob\_copied

Note these progressions in the following example code at the various times t1 through t5:

- 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* (for example, without the new changes made by *clob\_updated*). This is demonstrated by the subsequent DBMS\_LOB.READ() of the value just inserted.

### Example

```
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 using `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.

**See Also:** ["LOB Locators and Transaction Boundaries"](#) on page 5-14, for more information about the relationship between LOBs and transaction boundaries.

Using 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;
/
```

## LOB Locators and Transaction Boundaries

A basic description of LOB locators and their operations is given in [Chapter 2, "Basic LOB Components"](#).

This section discusses the use of LOB locators in transactions, and transaction IDs.

- *Locators Contain Transaction IDs When...*

*You Begin the Transaction, Then Select Locator.* 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.

- *Locators Do Not Contain Transaction IDs When...*

- *You are Outside the Transaction, Then Select Locator.* By contrast, if you select a locator outside of a transaction, the locator does not contain a transaction ID.
- *Locators Do Not Contain Transaction IDs When Selected Prior to DML Statement Execution.* A transaction ID will not be assigned until the first DML statement executes. Therefore, locators that are selected prior to such a DML statement will not contain a transaction ID.

## Transaction IDs: Reading and Writing to a LOB Using Locators

You can always read the LOB data using the locator irrespective of whether the locator contains a transaction ID.

- *Cannot Write Using Locator.* If the locator contains a transaction ID, you cannot write to the LOB outside of that particular transaction.
- *Can Write Using Locator.* If the locator *does not* contain a transaction ID, you can write to the LOB after beginning a transaction either explicitly or implicitly.
- *Cannot Read or Write Using Locator With Serializable Transactions.* If the locator contains a transaction ID of an older transaction, and the current transaction is serializable, you cannot read or write using that locator.
- *Can Read, Not Write Using Locator With Non-Serializable Transactions.* 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

## Non-Serializable Example: Selecting 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 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.

## **Non-Serializable Example: Selecting 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.



**See Also:** ["Read Consistent Locators"](#) on page 5-1 for more information about using the locator to read LOB data.

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 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.

**See Also:** ["Read Consistent Locators"](#) on page 5-1 for more information on the using the locator to read LOB data.

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.

## LOBs in the Object Cache

Consider these object cache issues for internal and external LOB attributes:

- *Internal LOB attributes: Creating an object in object cache, sets the LOB attribute to empty.* 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.
- *External LOB attributes: Creating an object in object cache, sets the BFILE attribute to NULL.* When creating an object with an external LOB (`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:** ["Updating LOBs and Read-Consistency"](#) on page 5-2 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*i* and Oracle9*i* provide a `LOB` buffering subsystem (LBS) for advanced OCI based applications such as Data Cartridges, 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 512KBytes. 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 round-trips 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.

## Guidelines for Using LOB Buffering

The following caveats apply to buffered LOB operations:

- *Explicitly flush LOB buffer contents.* Oracle8i and Oracle9i provide a simple buffering subsystem, and *not* a cache. To be specific, Oracle does not guarantee that the contents of a LOB's buffer are always in sync 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.
- *LOB Buffering is Single User, Single Threaded.* Transactions involving buffered LOB operations cannot migrate across user sessions — the LBS is a single user, single threaded system.
- *Maintain logical savepoints to rollback to.* Oracle 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 "[OCI Example of LOB Buffering](#)" on page 5-25).
- *Ensure LOB is not updated by another bypassing transaction.* 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. Oracle cannot distinguish, and hence prevent, such an operation. This will seriously affect the correctness and integrity of your application.

- *Updating buffer-enabled LOB locators.* 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 5-1.

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*. Oracle will return an error if you attempt to write to the LOB through other locators enabled for buffering. See also, "[Updating LOBs Via Updated Locators](#)" on page 5-5.

- *Passing a buffer-enabled LOB locator an IN OUT or OUT parameter.* 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.

- *When two or more locators point to the same LOB do not enable both for buffering.* If two or more different locators point to the same LOB, it is your responsibility to make sure that you do not enable both the locators for buffering. Otherwise Oracle does not guarantee the contents of the LOB.
- *Buffer-enabled LOBs do not support appends that create zero-byte fillers or spaces.* 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, Oracle requires the client side character set form for the locator bind variable 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 Usage Notes

### LOB Buffer Physical Structure

Each user *session* has the following structure:

- Fixed page pool of 16 pages, shared by all LOBs accessed in buffering mode from that session.
- Each *page* has a fixed size of up to 32K bytes (not characters) where  $\text{page size} = n \times \text{CHUNKSIZE} \approx 32\text{K}$ .

A LOB's buffer consists of one or more of these pages, up to a maximum of 16 in each 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.

### Example of Using the LOB Buffering System (LBS)

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, Oracle8i and Oracle9i allocate 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 32KBytes:

- For an input offset of 1000 and a specified read/write amount of 30000, Oracle 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 Oracle 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, Oracle 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, Oracle 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 *modified*, and either the currently accessed LOB, or one of the other LOBs, need to be flushed. Oracle notifies this condition to the user as an error. Oracle *never* flushes and reallocates a modified page implicitly. You can either flush them explicitly, or discard them by disabling buffering on the LOB.

To illustrate the preceding 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 containing unmodified data read in from the server. Assume similar conditions in L2's buffer. Now, for the next buffered operation on L1, Oracle 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, Oracle 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, Oracle 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

- 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 round-trip 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 in the following situations:

- 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, 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*
- *Return to the calling function/procedure/method in PL/SQL*

*Remember that Oracle never implicitly flushes the LOB.*

## Using Buffer-Enabled Locators

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

- *When it is OK to Use Buffer-Enabled Locators:*
  - *OCI* — A locator that is enabled for buffering can only be used with the following OCI APIs:  
`OCILobRead()`, `OCILobWrite()`, `OCILobAssign()`, `OCILobIsEqual()`,  
`OCILobLocatorIsInit()`, `OCILobCharSetId()`,  
`OCILobCharSetForm()`.

- *When it is Not OK to Use Buffer-Enabled Locators:* The following OCI APIs will return errors if used with a locator enabled for buffering:

- *OCI* — `OCILobCopy()`, `OCILobAppend()`, `OCILobErase()`,  
`OCILobGetLength()`, `OCILobTrim()`, `OCILobWriteAppend()`.

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.

- *PL/SQL (DBMS\_LOB)* — An error is returned from `DBMS_LOB` APIs if the input lob locator has buffering enabled.
- As in the case of all other locators, buffer-enabled locators cannot span transactions.



## Saving Locator State 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 round-trip 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 round-trip to the server, and so would refresh the snapshot in the locator.

Therefore to save the state of a LOB that has been written through the LOB buffer, follow these steps:

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 round-trip to the server to reselect the locator into L2.

## OCI Example of LOB Buffering

The following pseudocode for an OCI program based on the `Multimedia_tab` schema illustrates the issues described in the preceding discussion.

```
OCI_BLOB_buffering_program()
{
    int            amount;
    int            offset;
    OCILobLocator  lbs_loc1, lbs_loc2, lbs_loc3;
    void           *buffer;
    int            buf1;

    -- 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 through 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,
```

```
        buf1, ..);

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; buf1 = 20480;
    buffer = populate_buffer(20480);
    OCILobWrite(.., lbs_loc2, offset, amount, buffer,
        buf1, ..);

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; buf1 = 20000;
    buffer = populate_buffer(20000);
    OCILobWrite(.., lbs_loc3, offset, amount, buffer,
        buf1, ..);

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");
}
```

## Creating a Varray Containing References to LOBs

LOBs, or rather references to LOBs, can also be created using VARRAYs. To create a VARRAY containing references to LOBs read the following:

Column, MAP\_OBJ of type MAP\_TYP, already exists in table Multimedia\_tab. See [Appendix B, "The Multimedia Schema"](#) for a description of table Multimedia\_tab. Column MAP\_OBJ contains a BLOB column named DRAWING.

The syntax for creating the associated types and table Multimedia\_tab is described in [Chapter 10, "Internal Persistent LOBs"](#), [SQL: Create a Table Containing One or More LOB Columns](#), on page 10-7.

### Creating a Varray Containing LOB References: Example

Suppose you need to store multiple map objects in each multimedia clip. To do that follow these steps:

1. Define a VARRAY of type REF MAP\_TYP.

For example:

```
CREATE TYPE MAP_TYP_ARR AS
    VARRAY(10) OF REF MAP_TYP;
```

2. Define a column of the array type in `Multimedia_tab`.

For example:

```
CREATE TABLE MULTIMEDIA_TAB ( .....etc. [list all columns here]
... MAP_OBJ_ARR MAP_TYP_ARR)
VARRAY MAP_OBJ_ARR STORE AS LOB MAP_OBJ_ARR_STORE;
```

## LOBs in Partitioned Index-Organized Tables

Oracle9i introduces support for LOB, VARRAY columns stored as LOBs, and BFILES in partitioned index-organized tables. The behavior of LOB columns in these tables is similar to that of LOB columns in conventional (heap-organized) partitioned tables, except for the following few minor differences:

- *Tablespace Mapping.* In the absence of a tablespace specification for a LOB column at the individual partition level or at the table level, the LOB's data and index segments for that partition are created in the tablespace in which the primary key index segment of the corresponding partition of the index-organized table is created. In other words, they are equi-partitioned and collocated with their corresponding primary key index segments.
- *Inline vs Out-of-line.* If the partitioned index-organized table is defined without an overflow segment, then a LOB column cannot be defined or altered to be created *inline*. This requirement is the same as that for a non-partitioned index-organized table.

LOB columns are supported only in range partitioned index-organized tables.

### Example of LOB Columns in Partitioned Index-Organized Tables

In this section, we'll highlight the differences listed in the preceding for LOBs in partitioned index-organized tables with the `Multimedia_Tab` example described in [Appendix B](#).

Assume that `Multimedia-tab` has been created as a range-partitioned index-organized table, as follows:

```
CREATE TABLE Multimedia_tab (
  CLIP_ID    INTEGER PRIMARY KEY,
  CLIP_DATE  DATE,
  STORY     CLOB,
  FLSUB     NCLOB,
  PHOTO     BFILE,
```

```

FRAME      BLOB,
SOUND      BLOB,
...
)
ORGANIZATION INDEX
  TABLESPACE TBS_IDX
OVERFLOW
  TABLESPACE TBS_OVF
LOB (FRAME, SOUND) STORE AS (TABLESPACE TBS_LOB)
PARTITION BY RANGE (CLIP_DATE)
(PARTITION Jan_Multimedia_tab VALUES LESS THAN (01-FEB-2000)
  LOB (STORY) STORE AS (TABLESPACE TBS_LOB),
PARTITION Feb_Multimedia_tab VALUES LESS THAN (01-MAR-2000)
  LOB (FLSUB) STORE AS (TABLESPACE TBS_LOB
                        ENABLE STORAGE IN ROW)
);

```

In the preceding example, the LOB columns FRAME and SOUND will be stored in the tablespace TBS\_LOB across all the partitions.

- In the partition, Jan\_Multimedia\_tab, the column STORY is stored in TBS\_LOB because of the partition-specific override, but the column FLSUB will be stored in TBS\_IDX, that is, the tablespace of its primary key index segment.
- In the partition, Feb\_Multimedia\_tab, the column STORY is stored in TBS\_IDX, and the column FLSUB will be stored in TBS\_LOB by virtue of the partition-specific override.

---



---

**Note:** Since Multimedia\_tab was created with an overflow segment, the LOB columns are allowed to be stored inline. If that was not the case, the "ENABLE STORAGE IN ROW" clause for FLSUB in Feb\_Multimedia\_tab, would have generated an error.

---



---

The inheritance semantics for the rest of the LOB physical attributes are in line with LOBs in conventional tables.

**See Also:** *Oracle9i SQL Reference* for a description of the lob\_storage\_ clause in CREATE TABLE.

## Restrictions for LOBs in Partitioned Index-Organized Tables

### Range Partitioned Index-Organized Table LOB Restrictions

#### Non-Supported Column Types

The following column types in Range partitioned index-organized table are not supported:

- VARRAY columns STORED AS both inline/out-of-line LOBs.
- Abstract Data Types (ADTs) with LOB attributes
- VARRAY (stored as LOB) of ADT (with LOB attributes).
- NESTED TABLEs with LOB columns.

#### Non-Supported Column Types in Object Range Partitioned Index-Organized Tables

The following column types in Object Range partitioned index-organized tables are not supported:

- ADTs with LOB attributes
- VARRAY columns STORED AS both inline/out-of-line LOBs.
- VARRAY (stored as LOB) of ADT (with LOB attributes).
- NESTED TABLEs with LOB columns.

### Hash Partitioned Index-Organized Table LOB Restrictions

LOB columns are not supported in Hash partitioned index- organized tables.





---

## Frequently Asked Questions about LOBs

This chapter includes the following Frequently Asked Questions (FAQs):

- [Converting Data Types to LOB Data Types](#)
- [General](#)
- [Index-Organized Tables \(IOTs\) and LOBs](#)
- [Initializing LOB Locators](#)
- [JDBC, JPublisher and LOBs](#)
- [LOB Indexing](#)
- [LOB Storage and Space Issues](#)
- [LONG to LOB Migration](#)
- [Converting Between Different LOB Types](#)
- [Performance](#)
- [PL/SQL](#)

## Converting Data Types to LOB Data Types

### Can I Insert or Update Any Length Data Into a LOB Column?

#### Question

Can I insert or update any length of data for a LOB column? Am I still restricted to 4K. How about LOB attributes

#### Answer

When inserting or updating a LOB column you are now not restricted to 4K.

For LOB attributes, you must use the following two steps:

1. INSERT empty LOB with the RETURNING clause
2. Call OCILobWrite to write all the data

### Does COPY LONG to LOB Work if Data is > 64K?

#### Question

Example: Copy Long to LOB Using SQL :

```
INSERT INTO Multimedia_tab (clip_id,sound) SELECT id, TO_LOB(SoundEffects)
```

Does this work if the data in LONG or LONGRAW is > 64K?

#### Answer

Yes. All data in the LONG is copied to the LOB regardless of size.

## General

### How Do I Determine if the LOB Column with a Trigger is Being Updated?

#### Question

The project that I'm working on requires a trigger on a LOB column. The requirement is that when this column is updated, we want to check some conditions. How do I check whether there is any value in the NEW for this LOB column? Null does not work, since you can't compare BLOB with NULL.

#### Answer

You can use the UPDATING clause inside of the trigger to find out if the LOB column is being updated or not.

```
CREATE OR REPLACE TRIGGER.....
...
    IF UPDATING('lobcol')
    THEN .....
...

```

Note: The preceding works only for top-level LOB columns.

### Reading and Loading LOB Data: What Should Amount Parameter Size Be?

#### Question

I read in one of the prior release Application Developer's Guides the following:

"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 4Gb 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. "

And again, under the DBMS\_LOB.LOADFROMFILE() procedure...

"It is not an error to specify an amount that exceeds the length of the data in the source BFILE. Thus, you can specify a large amount to copy from the BFILE which will copy data from the src\_offset to the end of the BFILE. "

However, the following code...

```
declare
  cursor c is
    select id, text from bfiles;
  v_clob      clob;
begin
  for j in c
  loop
    Dbms_Lob.FileOpen ( j.text, Dbms_Lob.File_ReadOnly );
    insert into clobs ( id, text )
      values ( j.id, empty_clob() )
      returning text into v_clob;
    Dbms_Lob.LoadFromFile
      (
        dest_lob   => v_clob,
        src_lob    => j.text,
        amount     => 4294967296, /* = 4Gb */
        dest_offset => 1,
        src_offset  => 1
      );
    Dbms_Lob.FileClose ( j.text );
  end loop;
  commit;
end;
/
```

causes the following error message:

ORA-21560: argument 3 is null, invalid, or out of range

Reducing the amount by 1 to 4294967295 causes the following error message:

ORA-22993: specified input amount is greater than actual source amount

Please help me understand why I am getting errors.

### Answer

- **PL/SQL:**
  - For `DBMS_LOB.LOADFROMFILE`, you cannot specify the `amount` more than the size of the `BFILE`. So the code example you gave returns an error.
  - For `DBMS_LOB.READ`, the `amount` can be larger than the size of the data. But then, since PL/SQL limits the size of the buffer to 32K, and given the fact that the `amount` should be no larger than the size of the buffer, the `amount` is restricted to 32K.

Please note that in PL/SQL, if the `amount` is larger than the buffer size, it returns an error. In any case, the `amount` cannot exceed 4Gig-1 because that is the limit of a `ub4` variable.

- **OCI:** Again, you cannot specify `amount` larger than the length of the BFILE in `OCILobLoadFromFile`. However, in `OCILobRead`, you can specify `amount=4Gig-1`, and it will read to the end of the LOB.

## Is LOB Data Lost After a Crash?

### Question

We have a table with BLOB columns. We use `NOLOGGING` to be fast. It means that the BLOB chunks are not saved in redo logs. What happens if the server crashes? At recovery, is the table data lost or is the table corrupted?

### Answer

Any LOB data not written to the datafiles cannot be recovered from redo logs; it must be reloaded. Because the LOB index updates are written to redo, the index will indicate that the LOB exists, but the LOB will not exist (it was not recovered from redo), so my guess is that you will get a data corruption error.

## Index-Organized Tables (IOTs) and LOBs

### Is Inline Storage Allowed for LOBs in Index-Organized Tables?

#### Question

Is inline storage allowed for LOBs in index-organized tables?

#### Answer

For LOBs in index organized tables, inline LOB storage is allowed only if the table is created with an overflow segment.

## Initializing LOB Locators

### When Do I Use `EMPTY_BLOB()` and `EMPTY_CLOB()`?

#### Question

When must I use `EMPTY_BLOB()` and `EMPTY_CLOB()`? I always thought it was mandatory for each insert of a CLOB or BLOB to initialize the LOB locator first with either `EMPTY_CLOB()` or `EMPTY_BLOB()`.

#### Answer

In Oracle8i release 8.1.5, you can initialize a LOB with data using the `INSERT` statement as long as the data is <4K. This is why your insert statement worked. Note that you can also update a LOB with data that is <4K using the `UPDATE` statement. If the LOB is larger than 4K perform the following steps:

1. Insert into the table initializing the LOB using `EMPTY_BLOB()` or `EMPTY_CLOB()` and use the returning clause to get back the locator
2. For LOB attributes, call `ocilobwrite()` to write the entire data to the LOB. For other than LOB attributes, you can insert all the data using the `INSERT` statement.

Note the following:

- Oracle has now removed the <4K restriction and you can insert >4K worth of data into the LOB using the insert or even the update statement for LOB columns. Note however, that you cannot initialize a LOB attribute which is part of an object type with data and you must use `EMPTY_BLOB()/EMPTY_CLOB()`.
- Also you cannot use >4K as the default value for a LOB even though you can use >4k when inserting or updating the LOB data.
- Initializing the LOB value with data or using `EMPTY_BLOB()/EMPTY_CLOB()` is orthogonal to how the data is stored. If the LOB value is less than approximately 4K, then the value is stored inline (as long as the user doesn't specify `DISABLE STORAGE IN ROW`) and once it grows larger than 4K, it is moved out of line.

## How Do I Initialize a BLOB Attribute Using EMPTY\_BLOB() in Java?

### Question

From java we want to insert a complete object with a BLOB attribute into an Oracle8.1.5 object table. The problem is - in order to do that - we have somehow to initialize the blob attribute with EMPTY\_BLOB(). Is there any way to initialize the BLOB attribute with EMPTY\_BLOB() in java? What I am doing at the moment is:

First I insert the object with null in the BLOB attribute. Afterwards I update the object with an EMPTY\_BLOB(), then select it again, get the BLOB locator and finally write my BLOB.

Is this the only way it works? Is there a way to initialize the BLOB directly in my toDatum method of the Custom Datum interface implementation?

### Answer

Here is the SQLJ equivalent...

```
BLOB myblob = null;
#sql { select empty_blob() into :myblob from dual } ;
```

and use myblob in your code wherever the BLOB needed to be initialized to null.

See also the question and answer under the section, ["JDBC, JPublisher and LOBs"](#), ["How Do I setData to EMPTY\\_BLOB\(\) Using JPublisher?"](#)

## JDBC, JPublisher and LOBs

### How Do I Insert a Row With Empty LOB Locator into Table Using JDBC?

#### Question

Is it possible to insert a row with an empty LOB locator into a table using JDBC?

#### Answer

You can use the EMPTY\_BLOB() in JDBC also.

```
Statement stmt = conn.createStatement() ;
try {
    stmt.execute ("insert into lobtable values (empty_blob())");
}
}
```

```
catch{ ...}
```

Another example is:

```
stmt.execute ("drop table lobtran_table");
stmt.execute ("create table lobtran_table (b1 blob, b2 blob, c1 clob,
      c2 clob, f1 bfile, f2 bfile)");
stmt.execute ("insert into lobtran_table values
      ('01010101010101010101010101010101', empty_blob(),
      'onetwothreefour', empty_clob(),
      bfilename('TEST_DIR','tkpjobLOB11.dat'),
      bfilename ('TEST_DIR','tkpjobLOB12.dat'))");
```

## How Do I setData to EMPTY\_BLOB() Using JPublisher?

### Question

How do I setData to EMPTY\_BLOB() Using JPublisher? Is there something like EMPTY\_BLOB() and EMPTY\_CLOB() in a Java statement, not a SQL statement processed by JDBC? How do we setData to an EMPTY\_BLOB() using JPublisher?

### Answer

One way to build an empty LOB in JPublisher would be as follows:

```
BLOB b1 = new BLOB(conn, null) ;
```

You can use b1 in set method for data column.

## JDBC: Do OracleBlob and OracleClob Work in 8.1.x?

### Question

Do OracleBlob and OracleClob work in 8.1.x?

### Answer

OracleBlob and OracleClob were Oracle specific functions used in JDBC 8.0.x drivers to access LOB data. In Oracle8i version 8.1.x (and subsequent releases such as Oracle 9i and higher), OracleBlob and OracleClob are deprecated.

If you use OracleBlob or OracleClob to access LOB data, you will receive the following typical error message, for example, when attempting to manipulate LOBs with JDBC Thin Driver :



"Dumping lobs java.sql.SQLException: ORA-03115: unsupported network datatype or representation etc."

See *Oracle9i JDBC Developer's Guide and Reference* for a description of these non-supported functions and alternative and improved JDBC methods.

## How Do I Manipulate LOBs With the 8.1.5 JDBC Thin Driver?

### Question

Has anyone come across the following error when attempting to manipulate LOBs with the 8.1.5 JDBC Thin Driver:

```
Dumping lobs
java.sql.SQLException: ORA-03115: unsupported network datatype or representation
at oracle.jdbc.ttc7.TTIOer.processError(TTIOer.java:181)
at oracle.jdbc.ttc7.Odscrarr.receive(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.describe(Compiled Code)
at oracle.jdbc.ttc7.TTC7Protocol.parseExecuteDescribe(TTC7Protocol.java: 516)
at oracle.jdbc.driver.OracleStatement.doExecuteQuery(OracleStatement.java:1002)
at oracle.jdbc.driver.OracleStatement.doExecute(OracleStatement.java:1163)
at oracle.jdbc.driver.OracleStatement.doExecuteWithTimeout(OracleStateme
nt.java:1211)
at oracle.jdbc.driver.OracleStatement.executeQuery(OracleStatement.java: 201)
at LobExample.main(Compiled Code)
-----
```

The code I'm using is the LobExample.java shipped with 8.0.5. This sample was initially and OCI8 sample. (I am using the Thin Driver and instance from a higher release.)

### Answer

You are using a wrong sample. OracleBlob and OracleClob have been deprecated and they no longer work.

## Is the FOR UPDATE Clause Needed on SELECT When Writing to a LOB?

### Question

I am running a Java stored procedure that writes a CLOB and am getting an exception as follows:

ORA-22920: row containing the LOB value is not locked

ORA-06512: at "SYS.DBMS\_LOB", line 708

ORA-06512: at line 1

Once I added a 'FOR UPDATE' clause to my SELECT statement, this exception did not occur.

I feel that the JDBC Developer's Guide and Reference(8.1.5) should be updated to reflect the need for the 'FOR UPDATE' clause on the SELECT.

### **Answer**

This is not a JDBC issue in specific. This is how LOBs work! This got manifested in the JSP because by default autoCommit is false. You would also see the same exception when autoCommit is set to false on the client side. You didn't see the exception when used with 'For Update' because locks are acquired explicitly.

## **What Does DBMS\_LOB.ERASE Do?**

### **Question**

What is DBMS\_LOB.ERASE doing?

### **Answer**

It's just "clearing" a segment of the CLOB. It does *\*not\** shorten the CLOB. So the length of the CLOB is the same before and after the ERASE. You can use DBMS\_LOB.TRIM to make a CLOB shorter.

## **Can I Use putChars()?**

### **Question**

Can I use `oracle.sql.CLOB.putChars()`?

### **Answer**

Yes, you can, but you have to make sure that the position and length arguments are correct. You can also use the recommended OutputStream interface which in turn will call putChars for you.

## Manipulating CLOB CharSetId in JDBC

### Question

OCI provides function to manipulate a CLOB CharSetId. What is the JDBC equivalent?

### Answer

In JDBC CLOBs are *\*always\** in USC2, which is the Oracle character set corresponding to the Java "char" type. So there is no equivalent for the OCI CLOB CharSetId.

## Why is Inserting into BLOBs Slower than into LONG Rows?

### Question

Why is writing into BLOBs slower than inserting into LONG RAWs?

### Answer

It is true that inserting data in BLOBs with JDBC Thin is slower as it still uses the DBMS\_LOB package. With JDBC OCI the inserts are faster as native LOB APIs are used.

## Why Do I Get an ORA-03127 Error with LobLength on a LONG Column?

### Question

Why am I getting an ORA-03127 error when getting the LobLength on a LONG column?

### Answer

This is the correct behavior. LONG columns are not 'fetched' in-place (aka in-row). They are fetched out of place and exists in the pipe until you read them explicitly. In this case, we got the LobLocator (getBlob()) and then we are trying to get the length of this LOB before we read the LONG column. Since the pipe is not clear we are getting the preceding exception. The solution would be to complete reading the LONG column before you do any operation on the BLOB.

## How Do I Create a CLOB Object in a Java Program?

### Question

Here is what I'm trying to do with CLOBs through JDBC:

1. Create a CLOB object in a Java program
2. Populate the CLOB with the characters in a String passed into my program
3. Prepare a call to a stored procedure that receives a CLOB as a parameter.
4. Set the parameter with the Java CLOB
5. Execute

I was looking at the method `SQLUtil.makeOracleDatum()`, but that doesn't work. I get an invalid type error message. The only Oracle examples I've seen have the CLOB object created by reading it in from Oracle through a SQL object. I need to create the CLOB in the Java program.

### Answer

This cannot be done as you describe here. The `oracle.sql.CLOB` class encapsulates a CLOB locator, not the actual data for the CLOB, and the CLOB locator must come from the database. There is no way currently to construct a CLOB locator in the client. You need to insert an `empty_clob()` into the table, retrieve the locator, and then write the data to the CLOB.

PLSQL procedures can be poor vehicles for this particular functionality.

If you make the PLSQL parameter of the CLOB type, it represents the CLOB locator and you still have to use some other interface to write the data to the CLOB. And, passing all the data to PLSQL as a `VARCHAR2` or `LONG` parameter is also a problem because PLSQL parameters are limited to 32K, which is rarely enough to be practically useful in this context.

I would recommend just using the standard JDBC API's for dealing with the CLOB.

You need to encapsulate the entire functionality required to insert a CLOB, in a single stored procedure invoked from a client applicatiLoading LOBs and Data Into LOBs.

## How do I Load a 1MB File into a CLOB Column?

### Question

How do I insert a file of 1MB which is stored on disk, into a CLOB column of my table. I thought `DBMS_LOB.LOADFROMFILE` should do the trick, but, the document says it is valid for BFILE only. How do I do this?

### Answer

You can use SQL\*Loader. See *Oracle9i Database Utilities* or in this manual, [Chapter 4, "Managing LOBs"](#), [Using SQL\\*Loader to Load LOBs](#) on page 4-5.

You can use `loadfromfile()` to load data into a CLOB, but the data is transferred from the BFILE as raw data—no character set conversions are performed. It is up to you to do the character set conversions yourself before calling `loadfromfile()`.

Use `OCILobWrite()` with a callback. The callback can read from the operating system (OS) file and convert the data to the database character set (if it's different than the OS file's character set) and then write the data to the CLOB.

## How Do We Improve BLOB and CLOB Performance When Using JDBC Driver To Load?

### Question

We are facing a performance problem concerning BLOBs and CLOBs. Much time is consumed when loading data into the BLOB or CLOB using JDBC Driver.

### Answer

It's true that inserting data into LOBs using JDBC Thin driver is slower as it still uses the `DBMS_LOB` package and this adds the overhead of a full JDBC `CallableStatement` execution for each LOB operation.

With the JDBC OCI and JDBC server-side internal drivers, the inserts are faster because native LOB APIs are used. There is no extra overhead from JDBC driver implementation.

*It's recommended that you use `InputStream` and `OutputStream` for accessing and manipulating LOB data. By using streaming access of LOBs, JDBC driver will handle the buffering of the LOB data properly to reduce the number of network round-trips and ensure that each database operation uses a data size as a multiple of the LOB's natural chunk size.*

Here is an example that uses `OutputStream` to write data to a BLOB:

```
/*
 * This sample writes the GIF file john.gif to a BLOB.
 */

import java.sql.*;
import java.io.*;
import java.util.*;

// Importing the Oracle Jdbc driver package makes the code more readable
import oracle.jdbc.driver.*;

//needed for new CLOB and BLOB classes
import oracle.sql.*;

public class LobExample
{
    public static void main (String args [])
        throws Exception
    {
        // Register the Oracle JDBC driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the database
        // You can put a database name after the @ sign in the connection URL.
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "scott", "tiger");

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

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

        try
        {
            stmt.execute ("drop table persons");
        }
        catch (SQLException e)
        {
            // An exception could be raised here if the table did not exist already.
        }
    }
}
```

```
// Create a table containing a BLOB and a CLOB
stmt.execute ("create table persons (name varchar2 (30), picture blob)");

// Populate the table
stmt.execute ("insert into persons values ('John', EMPTY_BLOB())");

// Select the BLOB
ResultSet rset = stmt.executeQuery ("select picture from persons where name
= 'John'");
if (rset.next ())
{
    // Get the BLOB locator from the table
    BLOB blob = ((OracleResultSet)rset).getBLOB (1);

    // Declare a file handler for the john.gif file
    File binaryFile = new File ("john.gif");

    // Create a FileInputStream object to read the contents of the GIF file
    FileInputStream istream = new FileInputStream (binaryFile);

    // Create an OutputStream object to write the BLOB as a stream
    OutputStream ostream = blob.getBinaryOutputStream ();

    // Create a temporary buffer
    byte[] buffer = new byte[1024];
    int length = 0;

    // Use the read() method to read the GIF file to the byte
    // array buffer, then use the write() method to write it to
    // the BLOB.
    while ((length = istream.read(buffer)) != -1)
        ostream.write(buffer, 0, length);

    // Close the inputstream and outputstream
    istream.close();
    ostream.close();

    // Check the BLOB size
    System.out.println ("Number of bytes written = "+blob.length());
}

// Close all resources
rset.close();
stmt.close();
conn.close();
```

```
}  
}
```

Note that you'll get even better performance if you use `DBMS_LOB.LOADFROMFILE()` instead of using `DBMS_LOB.WRITE()`.

In order to be able to use `DBMS_LOB.LOADFROMFILE()`, the data to be written into the LOB must be in a server-side file.

## LOB Indexing

### Is LOB Index Created in Same Tablespace as LOB Data?

#### Question

Is the LOB index created for the LOB in the same tablespace as the LOB data?

#### Answer

The LOB index is created on the LOB *column* and it indexes the LOB data. The LOB index resides in the same tablespace as the *locator*.

### Indexing: Why is a BLOB Column Removed on DELETing but not a BFILE Column?

#### Question

The `promotion` column could be defined and indexed as a BFILE, but if for example, a row is DELETED, the Word document *is removed* with it when the `promotion` column is defined as *BLOB*, but it is *not removed* when the column is defined as a *BFILE*. Why?

#### Answer

We don't create an index for BFILE data. Also note that internal persistent LOBs are automatically backed up with the database whereas external BFILES are not and modifications to the internal persistent LOB can be placed in the redo log for future recovery.



## Which Views Can I Query to Find Out About a LOB Index?

### Question

Which views can I query to find out about a LOB index?

### Answer

- *Internal Persistent LOBs:*
  - **ALL\_INDEXES View:** Contains all the indexes the current user has the ability to modify in any way. You will not see the LOB index in this view because LOB indexes cannot be renamed, rebuilt, or modified.
  - **DBA\_INDEXES View:** Contains all the indexes that exist. Query this view to find information about the LOB index.
  - **USER\_INDEXES View:** Contains all the indexes that the user owns. The LOB index will be in this view if the user querying it is the same user that created it.

- *Temporary LOBs:*

For temporary LOBs, the LOB index information can be retrieved from the view, V\$SORT\_USAGE.

For example:

```
SELECT USER#, USERNAME, SEGTYPE, EXTENTS, BLOCKS
       FROM v$sort_usage, v$session
       WHERE SERIAL#=SESSION_NUM;
```

## LOB Storage and Space Issues

### What Happens If I Specify LOB Tablespace and ENABLE STORAGE IN ROW?

#### Question

What happens if I specify a LOB TABLESPACE, but also say ENABLE STORAGE IN ROW?

### **Answer**

If the length of the LOB value is less than approximately 4K, then the data is stored inline in the table. When it grows to beyond approximately 4K, then the LOB value is moved to the specified tablespace.

## **What Are the Pros and Cons of Storing Images in a BFILE Versus a BLOB?**

### **Question**

I am looking for information on the pros and cons of storing images in a BFILE versus a BLOB.

### **Answer**

Here's some basic information.

- Security:
  - BFILES are inherently insecure, as insecure as your operating system (OS).
- Features:
  - BFILES are not writable from typical database APIs whereas BLOBs are.
  - One of the most important features is that BLOBs can participate in transactions and are recoverable. Not so for BFILES.
- Performance:
  - Roughly the same.
  - Upping the size of your buffer cache can make a BIG improvement in BLOB performance.
  - BLOBs can be configured to exist in Oracle's cache which should make repeated/multiple reads faster.
  - Piece wise/non-sequential access of a BLOB is known to be faster than a that of a BFILE.
- Manageability:
  - Only the BFILE locator is stored in an Oracle BACKUP. One needs to do a separate backup to save the OS file that the BFILE locator points to. The BLOB data is backed up along with the rest of the database data.
- Storage:

- The amount of table space required to store file data in a BLOB will be larger than that of the file itself due to LOB index which is the reason for better BLOB performance for piece wise random access of the BLOB value.

## When Should I Specify DISABLE STORAGE IN ROW?

### Question

Should `DISABLE STORAGE IN ROW` always be specified if many `UPDATES`, or `SELECTS` including full table scans are anticipated?

### Answer

Use `DISABLE STORAGE IN ROW` if the *other table data* will be updated or selected frequently, not if the LOB data is updated or selected frequently.

## Do <4K BLOBs Go Into the Same Segment as Table Data, >4K BLOBs Go Into a Specified Segment?

### Question

If I specify a segment and tablespace for the BLOB, and specify `ENABLE STORAGE IN ROW` then look in `USER_LOBS`, I see that the BLOB is defined as `IN_ROW` and it shows that it has a segment specified. What does this mean? That all BLOBs 4K and under will go into the same segment as the table data, but the ones larger than that go into the segment I specified?

### Answer

Yes.

## Is 4K LOB Stored Inline?

### Question

*Oracle9i SQL Reference*, states the following:

"`ENABLE STORAGE IN ROW`--specifies that the LOB value is stored in the row (inline) if its length is less than approximately 4K bytes minus system control information. This is the default. "

If an inline LOB is > 4K, which of the following possibilities is true?

1. The first 4K gets stored in the structured data, and the remainder gets stored elsewhere
2. The whole LOB is stored elsewhere

It sounds to me like #2, but I need to check.

### Answer

You are correct -- it's number 2. Some meta information is stored inline in the row so that accessing the LOB value is faster. However, the entire LOB value is stored elsewhere once it grows beyond approximately 4K bytes.

1. If you have a NULL value for the BLOB *locator*, for example, if you have done the following:

```
INSERT INTO blob_table (key, blob_column) VALUES (1, null);
```

In this case I expect that you do not use any space, like any other NULL value, as we do not have any pointer to a BLOB value at all.

2. If you have a NULL in the BLOB, for example, if you have done the following:

```
INSERT INTO blob_table (key, blob_column) VALUES (1, empty_blob());
```

In this case you would be right, that we need at least a chunk size of space.

We distinguish between when we use BLOBs between NULL values and empty strings.

## How is a LOB Locator Stored If the LOB Column is EMPTY\_CLOB() or EMPTY\_BLOB() Instead of NULL? Are Extra Data Blocks Used For This?

### Question

If a LOB column is EMPTY\_CLOB() or EMPTY\_BLOB() instead of NULL, how is the LOB locator stored in the row and are extra data blocks used for this?

### Answer

See also [Chapter 7, "Modeling and Design"](#), in this manual, under "LOB Storage".

You can run a simple test that creates a table with a LOB column with attribute DISABLE STORAGE IN ROW. Insert thousands of rows with NULL LOBs.

Note that Oracle does not consume thousands of chunks to store NULLs!

## Storing CLOBs Inline: DISABLING STORAGE and Space Used

### Question

I have questions about storing CLOBs inline outside the row. We know when you create a table with LOB column, you can specify `DISABLE STORAGE IN ROW` or `ENABLE STORAGE IN ROW`. My questions are:

1. When you specify `ENABLE STORAGE IN ROW`, does it mean it stores the LOB information in the same block as that row?
2. I found the size of the table itself (not including the CLOB segment) with `ENABLE STORAGE IN ROW` is much bigger than the size of the table with `DISABLE STORAGE IN ROW`, and I know I have separate segment for the CLOB column in both tables. Why?

### Answer

1. If the LOB value is less than approximately 4k then the value is stored inline in the table. Whether or not the entire row is stored in one block depends on the size of the row. Big rows will span multiple blocks. If the LOB is more than 4k, then the LOB value is stored in a different segment.
2. This is because LOBs less than 4k will be stored inline in the table's segment.

## Should I Include a LOB Storage Clause When Creating Tables With Varray Columns?

### Question

What are the effects of providing or not providing a LOB storage clause when creating a table containing a Varray column? The documentation suggests that Varrays will be stored inline or in a LOB depending on their size, so I assume this would be the case even if no LOB storage clause were provided? Does providing one imply that a LOB will always be used?

I assume LOB are named for a reason. It is not clear to me what use the names might be. I understand that it is convenient to name the nested table storage table because you may want to index it, alter it, and so on. But what can I do with the LOB? The only example I found was one that modifies the LOB to cache it?

### Answer

The documentation says: "Varrays are stored in columns either as raw values or BLOBs. Oracle decides how to store the varray when the varray is defined, based on

the maximum possible size of the varray computed using the LIMIT of the declared varray. If the size exceeds approximately 4000 bytes, then the varray is stored in BLOBs. Otherwise, the varray is stored in the column itself as a raw value. In addition, Oracle supports inline LOBs; therefore, elements that fit in the first 4000 bytes of a large varray (with some bytes reserved for the LOB locator) are stored in the column of the row itself."

So, your data will be inline as raw data if you have less than about 4000 bytes and you do NOT specify a LOB storage clause for your varray.

The documentation (*Oracle9i SQL Reference*) also says:

"varray\_storage\_clause: lets you specify separate storage characteristics for the LOB in which a varray will be stored. In addition, if you specify this clause, Oracle will always store the varray in a LOB, even if it is small enough to be stored inline."

So, if you do specify this varray\_storage\_clause, then you will always be storing your varrays in LOBs. However, according to the first paragraph, varrays also support inline LOBs, so by default your first 4000 bytes or so will still be stored inline in the table's row with the other data as an inline LOB. It will also have some extra LOB overhead.

To clarify, if you specify varray store as LOB, and the column you've defined has a max size less than 4000 bytes, then it will be stored as an inline LOB. Here's the whole synopsis:

Calculate MAX possible size of your column, remember that there is some overhead involved so if you have 10 elements of size 1000 the MAX size is still a little bit greater than 10\*1000.

- If the max is less than (4000 minus some small number of bytes) and varray store as LOB is not specified, then it is stored as a raw inline.
- If the max is less than (4000 minus a small # of bytes) and varray store as LOB is specified and disable storage in row is not specified, then it is stored as an inline LOB
- If the max is less than (4000 - small # of bytes) and varray store as LOB is specified and disable storage in row is specified, then it is stored out of line in a LOB.
- If the max is greater than 4000 or so bytes it will always be stored in a LOB even if you don't have a varray store as LOB clause. It can be either an inline one or an out of line LOB depending on it's size and whether or not you've specified disable storage in row.

- If you do not specify a LOB storage clause or if you have a storage clause that doesn't specify disable storage in row then the system will automatically try to put small LOBs inline and only put them out of line when they are greater than approximately 4000 bytes. Otherwise, if the user specifies disable storage in row they will always be out of line.

## LONG to LOB Migration

### How Can We Migrate LONGs to LOBs, If Our Application Cannot Go Down?

#### Question

Our current table consists of records with 3 fields - a sequence, a redundancy check number, and a long raw field. The size of the long raw field is around 100KB but it can be as big as 300KB. The entire file is 160GB and the server's maximum size is 200GB. We converted this database from 7.3.4 to Oracle9i and now our application programs do not work well with the LONG raw fields. We want to convert them to BLOBs. We cannot have the application down while we migrate to BLOBs. What suggestions do you have?

#### Answer

Oracle9i allows you to use ALTER TABLE in order to copy the data from a LONG to a LOB. See [Chapter 8, "Migrating From LONGs to LOBs"](#). But the ALTER TABLE command would make the table unusable for the duration of the ALTER.

Another way to do this is to use the TO\_LOB operator introduced in Oracle 8i to copy data from the LONG to the LOB. You can take a look at *Oracle9i Database Migration*, Chapter 8 -- Copy LONGs to LOBs. In this case, the table will be unusable for a much shorter duration of time.

**See Also:** [Chapter 8, "Migrating From LONGs to LOBs"](#)

## Converting Between Different LOB Types

### Is Implicit LOB Conversion Between Different LOB Types Allowed?

#### Question

There are no implicit LOB conversions between different LOB types? For example, in PL/SQL, I cannot use:

```
INSERT INTO t VALUES ('abc');  
WHERE t CONTAINS a CLOB column....
```

Do you know if this restriction still exists in Oracle8i? I know that this restriction existed in *PL/SQL* for Oracle8 but users could issue the *INSERT* statement in *SQL* as long as data to insert was <4K. My understanding is that this <4K restriction has now been removed in *SQL*.

#### Answer

The PL/SQL restriction is removed. In Oracle8i and higher, you can insert more than 4K worth of data.

## Performance

### What Can We Do To Improve the Poor LOB Loading Performance When Using Veritas File System on Disk Arrays, UNIX, and Oracle?

#### Question 1

We were experiencing a load time of 70+ seconds when attempting to populate a BLOB column in the database with 250MB of video content. Compared to the 15 seconds transfer time using the UNIX copy, this seemed unacceptable. *What can we do to improve this situation?*

The BLOB was being stored in partitioned tablespace and NOLOGGING, NOCACHE options were specified to maximize performance.

The INITIAL and NEXT extents for the partition tablespace and partition storage were defined as 300M, with MINEXTENTS set to 1 in order to incur minimal overhead when loading the data.

CHUNK size was set to 32768 bytes - maximum for Oracle.



INIT.ORA parameters for db\_block\_buffers were increased as well as decreased.

All the preceding did very little to affect the load time - this stayed consistently around the 70-75 seconds range suggesting that there was minimal effect with these settings.

### Answer 1

First examine the I/O storage devices and paths.

### Question 2

**I/O Devices/Paths** 4 SUN AS5200 disk arrays were being used for data storage—the devices where the BLOB was to be written to. Disks on this array were RAID (0+1) with 4 stripes of (9+9). Veritas VxFS 3.2.1 was the file system on all disks.

In order to measure the effect of using a different device, the tablespace for the BLOB was defined on /tmp. /tmp is the swap space.

Needless to say, loading the BLOB now only took 14 seconds, implying a data transfer rate of 1.07GIG per minute - a performance rating as close, if not higher than the UNIX copy!

This prompted a closer examination of what was happening when the BLOB was being loaded to a tablespace on the disk arrays. SAR output indicated significant waits for I/O, gobbling up of memory, high CPU cycles and yes, the ever-consistent load time of 70 seconds. Any suggestions on how to resolve this?

### Answer 2

**Install the Veritas QuickIO Option!** Obviously, there seems to be an issue with Veritas, UNIX, and Oracle operating together. I have come up with supporting documentation on this. For acceptable performance with Veritas file-system on your disk arrays with Oracle, we recommend that you **install the Veritas QuickIO option**.

**A Final Note:** Typically when customers complain that writing LOBs is slow, the problem is usually not how Oracle writes LOBs. In the preceding case, you were using Veritas File System, which uses UNIX file caching, so performance was very poor.

After disabling UNIX caching, performance should improve over that with the native file copy.

## Is There a Difference in Performance When Using DBMS\_LOB.SUBSTR Versus DBMS\_LOB.READ?

### Question

Is there a difference in performance when using DBMS\_LOB.SUBSTR compared with DBMS\_LOB.READ?

### Answer

DBMS\_LOB.SUBSTR is there because it's a function and you can use it in a SQL statement. There is no performance difference.

## Are There Any White Papers or Guidelines on Tuning LOB Performance?

### Question

I was wondering if anyone had any white papers or guidelines on tuning LOB performance.

### Answer

[Chapter 9, "LOBs: Best Practices"](#) in this manual, discusses LOB performance issues. Also see "Selecting a Table Architecture" in [Chapter 7, "Modeling and Design"](#).

## When Should I Use Chunks Over Reading the Whole Thing?

### Question

When should I use chunks over reading the whole thing?

### Answer

If you intend to read more than one chunk of the LOB, then use OCILobRead with the streaming mechanism using either polling or a callback. If you only need to read a small part of the LOB that will fit in one chunk, then only read that chunk. Reading more will incur extra network overhead.

---

## Is Inlining the LOB a Good Idea and If So When?

### Question

Is inlining the LOB a good idea. If so, then when?

### Answer

Inlining the LOB is the default and is recommended most of the time. Oracle9i and higher stores the LOB inline if the value is less than approximately 4K thus providing better performance than storing the value out of line. Once the LOB grows larger than 4K, the LOB value is moved into a different storage segment but meta information that allows quick lookup of the LOB value is still stored inline. So, inlining provides the best performance most of the time.

However, you probably don't want to inline the LOB if you'll be doing a lot of base table processing such as full table scans, multi-row accesses (range scans) or many updates/selects of columns other than the LOB columns.

## How Can I Store LOBs >4Gb in the Database?

### Question

How can I store LOBs that are >4Gb in the database?

### Answer

Your alternatives for storing >4Gb LOBs are:

- Compressing the LOB so that it fits in 4Gb
- Breaking up the LOB into 4Gb chunks as separate LOB columns or as separate rows.

## Why is Performance Affected When Temporary LOBs are Created in a Called Routine?

### Question

We have a nasty performance problem that I have isolated to the creation of temporary LOBs in a called routine. The following procedures demonstrate the problem.

- When RUNLOB() is called with createlob=FALSE (its temporary LOB is created, but the inner routine's is not), it executes in less than a second.
- When run with createlob=TRUE (both outer and inner routine LOBs are created), it takes about 30 seconds and is linear with respect to the size of the loop. Here's a log:

```
SQL> set serveroutput on size 20000;
SQL> execute runlob(FALSE);
Start time (seconds): 52089
End time (seconds): 52089
PL/SQL procedure successfully completed.
```

```
SQL> execute runlob(TRUE);
Start time (seconds): 52102
End time (seconds): 52131
PL/SQL procedure successfully completed.
```

**This is really killing performance of DDL creation in our API. Any ideas what's happening here?**

```
CREATE OR REPLACE PROCEDURE lob(createlob BOOLEAN)
IS
  doc      CLOB;
BEGIN
  IF createlob THEN
    DBMS_LOB.CREATETEMPORARY(doc, TRUE);
    DBMS_LOB.FREETEMPORARY(doc);
  END IF;
END;
/
CREATE OR REPLACE PROCEDURE RUNLOB(createlob BOOLEAN DEFAULT FALSE) AS
doc      CLOB;
BEGIN
  dbms_output.put_line('Start time (seconds):
  '|to_char(sysdate, 'SSSSS'));
  FOR i IN 1..400 LOOP
    DBMS_LOB.CREATETEMPORARY(doc, TRUE);
    lob(createlob);
    DBMS_LOB.FREETEMPORARY(doc);
  END LOOP;
  dbms_output.put_line('End time (seconds):
  '|to_char(sysdate, 'SSSSS'));
END;
/
```

## Answer

In your test case, the difference between creating temporary LOBs in RUNLOB() and in LOB() is that:

- In RUNLOB(), there's a duration for the function frame and all the new temporary LOBs are linked from a single duration state object (DSO) created with the first temporary LOB in the function.
- In LOB(), however, every time the temporary LOB is created Oracle allocates a new DSO since the function frame duration is new.

kdl\_t\_add\_dso\_link() is an expensive operation compared to the rest of the temporary LOB creation cycles in kdl\_t? The overhead is from (de)allocating a DSO for LOB(). kdl\_t\_add\_dso\_link() needs to allocate a new DSO, for its associated memory allocation and control structures initialization. The extra code path accounts for the cost.

To avoid new DSO creation, can you use the workaround of a *package variable tmplob locator* in LOB() instead of a local one? Please try the following modified script. The performance hit is no longer there with this script.

```

create or replace package pk is
    tmplob clob;
end pk;
/

CREATE OR REPLACE PROCEDURE lob(createlob BOOLEAN)
IS
    doc    CLOB;
BEGIN
    IF createlob THEN
        DBMS_LOB.CREATETEMPORARY(pk.tmplob, TRUE);
        DBMS_LOB.FREETEMPORARY(pk.tmplob);
        null;
    END IF;
END;
/

CREATE OR REPLACE PROCEDURE RUNLOB(createlob BOOLEAN DEFAULT FALSE) AS
    doc    CLOB;
BEGIN
    dbms_output.put_line('Start time (seconds):
'|to_char(sysdate, 'SSSSS'));
    FOR i IN 1..400 LOOP
        DBMS_LOB.CREATETEMPORARY(doc, TRUE);

```

```
        lob(createlob);
        DBMS_LOB.FREETEMPORARY(doc);
    END LOOP;
    dbms_output.put_line('End time (seconds):
||to_char(sysdate, 'SSSS')');
END;
/
```

### **Response**

Thank you. We should be able to use package-scoped temporary LOBs almost everywhere we currently have function-local LOBs.

## **PL/SQL**

### **UPLOAD\_AS\_BLOB**

#### **Question**

What is "UPLOAD\_AS\_BLOB"?

#### **Answer**

UPLOAD\_AS\_BLOB is an attribute of Database Access Descriptor (DAD) which is used for uploading documents into BLOB type table column using PL/SQL web gateway interface.

---

---

## Modeling and Design

This chapter discusses the following topics:

- Selecting a Datatype
- Selecting a Table Architecture
- LOB Storage
- How to Create Gigabyte LOBs
- LOB Locators and Transaction Boundaries
- Binds Greater Than 4,000 Bytes in INSERTs and UPDATEs
- OPEN, CLOSE, and ISOPEN Interfaces for Internal LOBs
- LOBs in Index Organized Tables (IOT)
- Manipulating LOBs in Partitioned Tables
- Indexing a LOB Column
- SQL Semantics Support for LOBs
- How SQL VARCHAR2/RAW Semantics Apply to CLOBs/BLOBs
- SQL RAW Type and BLOBs
- SQL DML Changes For LOBs
- SQL Functions/Operators for VARCHAR2s/RAWs and CLOBs/BLOBs
- PL/SQL Statements and Variables: New Semantics Changes
- PL/SQL CLOB Comparison Rules
- Interacting with SQL and PL/SQL in OCI and Java Interfaces
- Performance Attributes When Using SQL Semantics with LOBs

- [User-Defined Aggregates and LOBs](#)

## Selecting a Datatype

When selecting a datatype, consider the following topics:

- [LOBs Compared to LONG and LONG RAW Types](#)
- [Character Set Conversions: Varying-Width and Multi-byte Fixed-Width Character Data](#)

## LOBs Compared to LONG and LONG RAW Types

[Table 7-1](#) lists the similarities and differences between LOBs, LONGs, and LONG RAW types.

**Table 7-1** *LOBs Vs. LONG RAW*

| LOB Data Type   | LONG and LONG RAW Data Type  |
|---|--|
| You can store multiple LOBs in a single row   | You can store only one LONG or LONG RAW in each row.   |
| LOBs can be attributes of a user-defined datatype   | 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.<br><br>For inline LOBs, Oracle will store LOBs that are less than approximately 4,000 bytes of data in the table column. | In the case of a LONG or LONG RAW the entire value is stored in the table column.  |
| When you access a LOB column, you can choose to fetch the locator or the data.  | 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. LOBs can be accessed at random offsets.   | Less flexibility in manipulating data in a random, piece-wise manner with LONG or LONG RAW data. LONGs must be accessed from the beginning to the desired location . |



**Table 7–1 LOBs Vs. LONG RAW**

| LOB Data Type  | LONG and LONG RAW Data Type   |
|--|---|
| You can replicate LOBs in both local and distributed environments. | Replication in both local and distributed environments is not possible with a LONG or LONG RAW (see <i>Oracle9i Replication</i> ) |

## Replication

Oracle does not support the replication of columns that use the LONG and LONG RAW data types. Oracle simply omits columns containing these data types from replicated tables. In Oracle9i, you must convert LONG data types to LOBs and then replicate.

## Converting LONG Columns to LOBs

Existing LONG columns can be converted to LOBs using either of the following methods:

- LONG-to-LOB API described in [Chapter 8, "Migrating From LONGs to LOBs"](#)
- TO\_LOB() function (see ["LONG to LOB Migration Using the LONG-to-LOB API"](#) on page 10-62 in [Chapter 10, "Internal Persistent LOBs"](#)).

---



---

**Note:** Oracle9i does not support conversion of LOBs back to LONGs.

---



---

## Character Set Conversions: Varying-Width and Multi-byte Fixed-Width Character Data

In OCI (Oracle Call Interface), 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`.

**See:** *Oracle9i Database Globalization Support Guide* for more detail on character set conversions.

---

---

**Note:** The ALTER DATABASE command will not work when there are CLOB or NCLOB columns in the tables.

---

---

## Selecting a Table Architecture

When designing your table, consider the following design criteria:

- LOB storage
  - [Where are NULL Values in a LOB Column Stored?](#)
  - [Defining Tablespace and Storage Characteristics for Internal LOBs](#)
  - [LOB Storage Characteristics for LOB Column or Attribute](#)
  - [TABLESPACE and LOB Index](#)
    - \* [PCTVERSION](#)
    - \* [CACHE / NOCACHE / CACHE READS](#)
    - \* [LOGGING / NOLOGGING](#)
    - \* [CHUNK](#)
    - \* [ENABLE | DISABLE STORAGE IN ROW](#)
  - [How to Create Gigabyte LOBs](#)
- [LOBs in Index Organized Tables \(IOT\)](#)
- [Manipulating LOBs in Partitioned Tables](#)
- [Indexing a LOB Column](#)

## LOB Storage

### Where are NULL Values in a LOB Column Stored?

#### **NULL LOB Column Storage: NULL Value is Stored**

If a LOB column is NULL, no data blocks are used to store the information. The NULL *value* is stored in the row just like any other NULL value. This is true even when you specify `DISABLE STORAGE IN ROW` for the LOB.

### **EMPTY\_CLOB() or EMPTY\_BLOB() Column Storage: LOB Locator is Stored**

If a LOB column is initialized with `EMPTY_CLOB()` or `EMPTY_BLOB()`, instead of `NULL`, a *LOB locator* is stored in the row. No additional storage is used.

- **DISABLE STORAGE IN ROW:** If you have a LOB with one byte of data, there will be a LOB locator in the row. This is true whether or not the LOB was created as `ENABLE` or `DISABLE STORAGE IN ROW`. In addition, an entire chunksize of data blocks is used to store the one byte of data if the LOB column was created as `DISABLE STORAGE IN ROW`.
- **ENABLE STORAGE IN ROW:** If the LOB column was created as `ENABLE STORAGE IN ROW`, Oracle<sup>8i</sup> and higher only consumes one extra byte of storage in the row to store the one byte of data. If you have a LOB column created with `ENABLE STORAGE IN ROW` and the amount of data to store is larger than will fit in the row (approximately 4,000 bytes) Oracle uses a multiple of chunksizes to store it.

## **Defining 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.

### **Defining Tablespace and Storage Example1**

```
CREATE TABLE ContainsLOB_tab (n NUMBER, c CLOB)
  lob (c) STORE AS SEGNAME (TABLESPACE lobtbs1 CHUNK 4096
    PCTVERSION 5
    NOCACHE LOGGING
    STORAGE (MAXEXTENTS 5)
  );
```

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` statement.

---

---

**Note:** Only some storage parameters may be modified! For example, you can use the ALTER TABLE ... MODIFY LOB statement to change PCTVERSION, CACHE/NO CACHE LOGGING/NO LOGGING, and the STORAGE clause.

You can also change the TABLESPACE using the ALTER TABLE ...MOVE statement.

However, once the table has been created, you cannot change the CHUNK size, or the ENABLE/DISABLE STORAGE IN ROW settings.

---

---

### Assigning a LOB Data Segment Name

As shown in the "[Defining Tablespace and Storage Example1](#)" on page 7-5, 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 *Oracle9i Database Reference*), you see the LOB data segment that you chose instead of system-generated names.

## LOB Storage Characteristics for LOB Column or Attribute

LOB storage characteristics that can be specified for a LOB column or a LOB attribute include the following:

- TABLESPACE
- PCTVERSION
- CACHE/NOCACHE/CACHE READS
- LOGGING/NOLOGGING
- CHUNK
- ENABLE/DISABLE STORAGE IN ROW
- STORAGE . See the "STORAGE clause" in *Oracle9i SQL Reference* for more information.

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 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 or attribute in order to reduce device contention.

The LOB index is an internal structure that is strongly associated with LOB storage. This implies that a user may not drop the LOB index and rebuild it.

---

---

**Note:** The LOB index cannot be altered.

---

---

The system determines which tablespace to use for LOB data and LOB index depending on your 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.

### Tablespace for LOB Index in Non-Partitioned Table

If in creating tables in Oracle8i Release 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 produced 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 tries 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 old LOB data.

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

To decide what value `PCTVERSION` should be set to, consider the following:

- How often LOBs are updated?
- How often the updated LOBs are read?

[Table 7-2, "Recommended PCTVERSION Settings"](#) provides some guidelines for determining a suitable `PCTVERSION` value.

**Table 7-2 Recommended PCTVERSION Settings**

| 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 updated 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 afterward. 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 / CACHE READS

When creating tables that contain LOBs, use the cache options according to the guidelines in [Table 7-3, "When to Use CACHE, NOCACHE, and CACHE READS"](#):

**Table 7–3 When to Use CACHE, NOCACHE, and CACHE READS**

| Cache Mode        | Read ...             | Written To ...       |
|-------------------|----------------------|----------------------|
| CACHE             | Frequently           | Frequently           |
| NOCACHE (default) | Once or occasionally | Never                |
| CACHE READS       | Frequently           | Once or occasionally |

**CACHE / NOCACHE / CACHE READS: LOB Values and Buffer Cache**

- **CACHE:** Oracle places LOB pages in the buffer cache for faster access.
- **NOCACHE:** As a parameter in the `LOB_storage_clause`, `NOCACHE` specifies that LOB values are either not brought into the buffer cache or are brought into the buffer cache and placed at the least recently used end of the LRU list.
- **CACHE READS:** LOB values are brought into the buffer cache only during read and not during write operations.

**Downgrading to 8.1.5 or 8.0.x**

If you have `CACHE READS` set for LOBs in 8.1.6 and you downgrade to 8.1.5 or 8.0.x, your `CACHE READS` LOBs generate a warning and become `CACHE LOGGING` LOBs.

You can explicitly alter the LOBs' storage characteristics later if you do not want your LOBs to be `CACHE LOGGING`. For example, if you want the LOBs to be `NOCACHE`, use `ALTER TABLE` to clearly modify them to `NOCACHE`.

**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.

- **CACHE is specified** and `[NO]LOGGING` clause is omitted, `LOGGING` is automatically implemented (because you cannot have `CACHE NOLOGGING`).
- **CACHE is not specified** and `[NO]LOGGING` clause is omitted, 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.

**NOLOGGING is Useful for Bulk Loads or Inserts.** For instance, when loading data into the LOB, if you do not 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 provides good performance for the initial load of data.

Once you have completed loading data, if necessary, use ALTER TABLE to modify the LOB storage characteristics for the LOB data segment for normal LOB operations, for example, to CACHE or NOCACHE LOGGING.

---

---

**Note:** CACHE implies that you also get LOGGING.

---

---

## **CHUNK**

Set CHUNK to the total bytes of LOB data in multiples of database block size, that is, the number of blocks that will be read or written using OCILobRead( ), OCILobWrite( ), DBMS\_LOB.READ( ), or DBMS\_LOB.WRITE( ) during one access of the LOB value.

---

---

**Note:** The default value for CHUNK is one Oracle block and does not vary across platforms.

---

---

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.



### Set INITIAL and NEXT to Larger than CHUNK

If you explicitly specify 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 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 they are set in the following manner:

- `CHUNK <= NEXT`
- `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 (in the row) or out of line.

---

---

**Note:** 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`.

## Guidelines for ENABLE or DISABLE STORAGE IN ROW

The maximum amount of LOB data stored in the row is the maximum `VARCHAR2` size (4000). This includes the control information as well as the LOB value. If you indicate 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 guidelines:

The default, `ENABLE STORAGE IN ROW`, is usually the best choice for the following reasons:

- *Small LOBs:* If the LOB is small (< 4000 bytes), then the whole LOB can be read while reading the row without extra disk I/O.
- *Large LOBs:* If the LOB is big (> 4000 bytes), then the control information is still stored in the row if `ENABLE STORAGE IN ROW` is set, even after moving the

LOB data out of the row. This control information could enable us to read the out-of-line LOB data faster.

However, in some cases `DISABLE STORAGE IN ROW` is a better choice. This is because storing the LOB in the row increases the size of the row. This impacts performance if you are 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.

## How to Create Gigabyte LOBs

LOBs in Oracle8i and higher can be up to 4 gigabytes. To create gigabyte LOBs, use the following guidelines to make use of all available space in the tablespace for LOB storage:

- *Single Datafile Size Restrictions:* There are restrictions on the size of a single datafile for each operating system (OS). For example, Solaris 2.5 only allows OS files of up to 2 gigabytes. Hence, add more datafiles to the tablespace when the LOB grows larger than the maximum allowed file size of the OS on which your Oracle database runs.
- *Set PCT INCREASE Parameter to Zero:* `PCTINCREASE` parameter in the LOB storage clause specifies the percent growth of the new extent size. When a LOB is being filled up piece by piece in a tablespace, numerous new extents get created in the process. If the extent sizes keep increasing by the default value of 50 percent every time, extents will become unmanageably big and eventually will waste unnecessary space in the tablespace. Therefore, the `PCTINCREASE` parameter should be set to zero or a small value.
- *Set MAXEXTENTS to a Suitable Value or UNLIMITED:* The `MAXEXTENTS` parameter limits the number of extents allowed for the LOB column. A large number of extents are created incrementally as the LOB size grows. Therefore, the parameter should be set to a value that is large enough to hold all the LOBs for the column. Alternatively, you could set it to `UNLIMITED`.
- *Use a Large Extent Size:* For every new extent created, Oracle generates undo information for the header and other meta data for the extent. If the number of extents is large, the rollback segment can be saturated. To get around this, choose a large extent size, say 100 megabytes, to reduce the frequency of extent creation, or commit the transaction more often to reuse the space in the rollback segment.

## Example 1: Creating a Tablespace and Table to Store Gigabyte LOBs

A working example of creating a tablespace and a table that can store gigabyte LOBs follows. The case applies to the multimedia application example in [Chapter 10, "Internal Persistent LOBs"](#), if the video Frame in the multimedia table is expected to be huge in size, for example, several gigabytes.

```
CREATE TABLESPACE lobtbs1 datafile '/your/own/data/directory/lobtbs_1.dat' size
2000M reuse online nologging default storage (maxextents unlimited);
ALTER TABLESPACE lobtbs1 add datafile '/your/own/data/directory/lobtbs_2.dat'
size 2000M reuse;

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,
  Comments        LONG
)
NESTED TABLE    InSeg_ntab STORE AS InSeg_nestedtab
LOB(Frame) store as (tablespace lobtbs1 chunk 32768 pctversion 0 NOCACHE
NOLOGGING
storage(initial 100M next 100M maxextents unlimited pctincrease 0));
```

## Example 2: Creating a Tablespace and Table to Store Gigabyte LOBs

The difference between Example 1 and this example is that one specifies the storage clause during CREATE TABLE and one does it in CREATE TABLESPACE.

- *For temporary LOBs*, the STORAGE clause has to be specified when creating the temp tablespace
- *For persistent LOBs*, the STORAGE clause can be specified either when creating tablespace or when creating table

### How this Affects the Temporary LOB COPY or APPEND?

The critical factor is setting the PCTINCREASE parameter to 0. Otherwise, the default value is 50%. When a 4gigabyte LOB is being filled up, the extents size expands gradually until it blows up the tablespace, as follows:

1st extent: 100M, 2nd 100M, 3rd, 150M, 4th 225M...

## LOB Locators and Transaction Boundaries

See [Chapter 2, "Basic LOB Components"](#) for a basic description of LOB locators and their operations.

See [Chapter 5, "Large Objects: Advanced Topics"](#) for a description of LOB locator transaction boundaries and using read consistent locators.

## Binds Greater Than 4,000 Bytes in INSERTs and UPDATEs

### Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs

This release supports binds of more than 4,000 bytes of data for LOB INSERTs and UPDATEs. In previous releases this feature was allowed for LONG columns only. You can now bind the following for INSERT or UPDATE into a LOB column:

- Up to 4GB data using OCIBindByPos(), OCIBindByName()
- Up to 32,767 bytes data using PL/SQL binds

Since you can have multiple LOBs in a row, you can bind up to 4GB data for each one of those LOBs in the same INSERT or UPDATE statement. In other words, multiple binds of more than 4,000 bytes in size are allowed in a single statement.

---

---

**Note:** The length of the default values you specify for LOBs still has the 4,000 byte restriction.

---

---

**Ensure Your Temporary Tablespace is Large Enough!** The bind of more than 4,000 bytes of data to a LOB column uses space from temporary tablespace. Hence ensure that your temporary tablespace is large enough to hold at least the sum of all the bind lengths for LOBs.

If your temporary tablespace is extendable, it will be extended automatically after the existing space is fully consumed. Use the following statement:

```
CREATE TABLESPACE .. AUTOEXTEND ON ... TEMPORARY ..;
```

to create an extendable temporary tablespace.

## Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion

Table `Multimedia_tab` is described in [Appendix B, "The Multimedia Schema"](#). The following examples use an additional column called `Comments`. You will need to add the `Comments` column to table `Multimedia_tab`'s `CREATE TABLE` syntax with the following line:

```
Comments LONG -- stores the comments of viewers on this clip
```

Oracle does not do any implicit conversion, such as HEX to RAW or RAW to HEX e.t.c., for data of more than 4000 bytes.

```
declare
  charbuf varchar2(32767);
  rawbuf raw(32767);
begin
  charbuf := lpad ('a', 12000, 'a');
  rawbuf := utl_raw.cast_to_raw(charbuf);
```

[Table 7–4, "Binds of More Than 4,000 Bytes: Allowed INSERT and UPDATE Operations"](#), outlines which INSERT operations are allowed in the preceding example and which are not. The same cases also apply to UPDATE operations.

**Table 7–4 Binds of More Than 4,000 Bytes: Allowed INSERT and UPDATE Operations**

| Allowed INSERTs/UPDATES ...  | Non-Allowed INSERTs/UPDATES ...  |
|--|--|
| <pre>INSERT INTO   Multimedia_tab (story, sound) VALUES (charbuf, rawbuf);</pre> | <pre>INSERT INTO   Multimedia_tab(sound) VALUES(charbuf);</pre> <p>This does not work because Oracle will not do implicit hex to raw conversion.</p> |
| -  | <pre>INSERT INTO   Multimedia_tab(story) VALUES (rawbuf);</pre> <p>This does not work because Oracle will not do implicit hex to raw conversion.</p> |

**Table 7-4 Binds of More Than 4,000 Bytes: Allowed INSERT and UPDATE Operations**

| Allowed INSERTs/UPDATES ... | Non-Allowed INSERTs/UPDATES ...   |
|-----------------------------|---|
| -                           | <pre>INSERT INTO   Multimedia_tab(sound) VALUES(   utl_raw.cast_to_raw(charbuf));</pre> <p>This does not work because Oracle cannot combine utl_raw.cast_to_raw() operator with binds of more than 4,000 bytes.</p> |

### 4,000 Byte Limit On Results of SQL Operator

If you bind more than 4,000 bytes of data to a BLOB or a CLOB, and the data consists of an SQL operator, then Oracle limits the size of the result to at most 4,000 bytes.

The following statement inserts only 4,000 bytes because the result of LPAD is limited to 4,000 bytes:

```
INSERT INTO Multimedia_tab (story) VALUES (lpad('a', 5000, 'a'));
```

The following statement inserts only 2,000 bytes because the result of LPAD is limited to 4,000 bytes, and the implicit hex to raw conversion converts it to 2,000 bytes of RAW data:

```
INSERT INTO Multimedia_tab (sound) VALUES (lpad('a', 5000, 'a'));
```

### Binds of More Than 4,000 Bytes: Restrictions

The following lists the restrictions for binds of more than 4,000 bytes:

- If a table has both LONG and LOB columns then you can bind more than 4,000 bytes of data to either the LONG or LOB columns, but not both in the same statement.
- You cannot bind data of any size to LOB attributes in Abstract Data Types (ADTs). This restriction in prior releases still exists. For LOB attributes, first insert an empty LOB locator and then modify the contents of the LOB using OCILob\* functions.
- In an INSERT AS SELECT operation, binding of any length data to LOB columns is not allowed. This restriction in prior releases still exists.

### Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE

```
CREATE TABLE foo (a INTEGER);
DECLARE
  bigtext    VARCHAR2(32767);
```

```

        smalltext  VARCHAR2(2000);
        bigraw     RAW (32767);
BEGIN
    bigtext       := LPAD('a', 32767, 'a');
    smalltext     := LPAD('a', 2000, 'a');
    bigraw        := utlraw.cast_to_raw (bigtext);

    /* The following is allowed: */
        INSERT INTO Multimedia_tab(clip_id, story, frame, comments)
            VALUES (1,bigtext, bigraw,smalltext);
    /* The following is allowed: */
        INSERT INTO Multimedia_tab (clip_id, story, comments)
            VALUES (2,smalltext, bigtext);

        bigtext     := LPAD('b', 32767, 'b');
        smalltext   := LPAD('b', 20, 'a');
        bigraw      := utlraw.cast_to_raw (bigtext);

    /* The following is allowed: */
        UPDATE Multimedia_tab SET story = bigtext, frame = bigraw,
            comments = smalltext;

    /* The following is allowed */
        UPDATE Multimedia_tab set story = smalltext, comments = bigtext;

    /* The following is NOT allowed because we are trying to insert more than
        4000 bytes of data in a LONG and a LOB column: */
        INSERT INTO Multimedia_tab (clip_id, story, comments)
            VALUES (5, bigtext, bigtext);

    /* The following is NOT allowed because we are trying to insert
        data into LOB attribute */
        INSERT into Multimedia_tab (clip_id,map_obj)
            VALUES (10,map_typ(NULL, NULL, NULL, NULL, NULL,bigtext, NULL));

    /* The following is not allowed because we try to perform INSERT AS
        SELECT data INTO LOB */
        INSERT INTO Multimedia_tab (story) AS SELECT bigtext FROM foo;
END;
```

## Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported

For binds of more than 4,000 bytes, inserts are not supported because hex-to-raw and raw-to-hex conversions are not supported.

```

/* Oracle does not do any implicit conversion (e.g., HEX to RAW or RAW to HEX
   etc.) for data of more than 4000 bytes. Hence, the following cases will not
   work : */

declare
  charbuf  varchar2(32767);
  rawbuf   raw(32767);
begin
  charbuf := lpad ('a', 12000, 'a');
  rawbuf  := utl_raw.cast_to_raw(charbuf);

/* The following is allowed ... */
  INSERT INTO Multimedia_tab (story, sound) VALUES (charbuf, rawbuf);

/* The following is not allowed because Oracle won't do implicit
   hex to raw conversion. */
  INSERT INTO Multimedia_tab (sound) VALUES (charbuf);

/* The following is not allowed because Oracle won't do implicit
   raw to hex conversion. */
  INSERT INTO Multimedia_tab (story) VALUES (rawbuf);

/* The following is not allowed because we can't combine the
   utl_raw.cast_to_raw() operator with the bind of more than 4,000 bytes. */
  INSERT INTO Multimedia_tab (sound) VALUES (utl_raw.cast_to_raw(charbuf));

end;
/

```

## Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes

If you bind more than 4,000 bytes of data to a BLOB or a CLOB, and the data actually consists of an SQL operator, then Oracle limits the size of the result to 4,000 bytes.

For example,

```

/* The following command inserts only 4,000 bytes because the result of
   LPAD is limited to 4,000 bytes */
  INSERT INTO Multimedia_tab (story) VALUES (lpad('a', 5000, 'a'));

/* The following command inserts only 2,000 bytes because the result of

```



*LPAD is limited to 4,000 bytes, and the implicit hex to raw conversion converts it to 2,000 bytes of RAW data. \*/*

```
INSERT INTO Multimedia_tab (sound) VALUES (lpad('a', 5000, 'a'));
```

## Example: C (OCI) - Binds of More than 4,000 Bytes For INSERT and UPDATE

```
CREATE TABLE foo( a INTEGER );
void insert()          /* A function in an OCI program */
{
  /* The following is allowed */
  ub1 buffer[8000];
  text *insert_sql = "INSERT INTO Print_media(ad_sourcetext, ad_composite,
comments)
VALUES (:1, :2, :3)";
  OCISmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
(ub4) OCI_NIV_SYNTAX, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
SQLT_LBI, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 2000,
SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
  OCISmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
  /* The following is allowed */
  ub1 buffer[8000];
  text *insert_sql = "INSERT INTO Print_media (ad_sourcetext,comments)
VALUES (:1, :2)";
  OCISmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
(ub4) OCI_NIV_SYNTAX, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
  OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
  OCISmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
  /* The following is allowed, no matter how many rows it updates */
  ub1 buffer[8000];
  text *insert_sql = (text *)"UPDATE Print_media SET
```

```

        ad_sourcetext = :1, ad_photo=:2, comments=:3";
OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
    SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
    SQLT_LBI, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 2000,
    SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
    /* The following is allowed, no matter how many rows it updates */
    ub1 buffer[8000];
    text *insert_sql = (text *)"UPDATE Print_media SET
        ad_sourcetext = :1, ad_photo=:2, comments=:3";
    OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 2000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)buffer, 8000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
    OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
    /* Piecewise, callback and array insert/update operations similar to
    the allowed regular insert/update operations are also allowed */
}

void insert()
{
    /* The following is NOT allowed because we try to insert >4000 bytes
    to both LOB and LONG columns */
    ub1 buffer[8000];
    text *insert_sql = (text *)"INSERT INTO Print_media (ad_composite, comments)
        VALUES (:1, :2)";
    OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,

```

```

        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)buffer, 8000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
    /* The following is NOT allowed because we try to insert data into
       LOB attributes */
    ub1 buffer[8000];
    text *insert_sql = (text *)"INSERT INTO Print_media (adheader_typ)
        VALUES (adheader_typ(NULL, NULL, NULL, NULL, :1, NULL))";
    OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 2000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
    OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
    /* The following is NOT allowed because we try to do insert as
       select character data into LOB column */
    ub1 buffer[8000];
    text *insert_sql = (text *)"INSERT INTO Print_media (ad_sourcetext)
        SELECT :1 from FOO";
    OCIStmtPrepare(stmthp, errhp, insert_sql, strlen((char*)insert_sql),
        (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)buffer, 8000,
        SQLT_LNG, 0, 0, 0, 0, 0, (ub4) OCI_DEFAULT);
    OCIStmtExecute(svchp, stmthp, errhp, 1, 0, OCI_DEFAULT);
}

void insert()
{
    /* Other update operations similar to the disallowed insert operations are also
       not allowed. Piecewise and callback insert/update operations similar to the
       disallowed regular insert/update operations are also not allowed */
}

```

## OPEN, CLOSE, and ISOPEN Interfaces for Internal LOBs

The OPEN, CLOSE, and ISOPEN 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 does not impact already-existing applications that write to LOBs without first opening them, since these calls did not exist in 8.0.

---

---

**Note:** 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.

---

---

### Wrap LOB Operations Inside an OPEN / CLOSE Call

- *If you do not wrap LOB operations inside an OPEN/CLOSE call operation:* Each modification to the LOB will implicitly open and close the LOB thereby firing any triggers on a domain index. Note that in this case, any domain indexes on the LOB will become updated as soon as LOB modifications are made. Therefore, domain LOB indexes are always valid and may be used at any time.
- *If you wrap your LOB operations inside the OPEN/CLOSE operation:* Triggers will not be fired for each LOB modification. Instead, the trigger on domain indexes will be fired at the CLOSE call. For example, you might design your application so that domain indexes are not be updated until you call CLOSE. However, this means that any domain indexes on the LOB will not be valid in-between the OPEN/CLOSE calls.

### Close All Opened LOBs Before Committing the Transaction

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, but the transaction is successfully committed. Hence, all the changes made to the LOB and non-LOB data in the transaction are committed but but the domain and functional indexes are not updated. If this happens, please rebuild your functional and domain indexes on the LOB column.

---

---

**Note:** Changes to the LOB are not discarded if the COMMIT returns an error.

---

---

At transaction rollback, the openness of all open LOBs still open for that transaction are discarded. Discarding the openness means the following for LOBs:

- LOBs will not be closed
- Triggers on domain indexes will not be fired

### What is a 'Transaction' Where an Open LOB Value is Closed?

A 'transaction' where an open LOB value must be closed meets 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 domain indexes will be fired.

### Do Not Open or Close Same LOB Twice!

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

## Example 1: Correct Use of OPEN/CLOSE Calls to LOBs in a Transaction

This example shows the correct use of open and close calls to LOBs inside and outside a transaction.

```

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: Incorrect Use of OPEN/CLOSE Calls to a LOB in a Transaction

This example the incorrect use of OPEN and CLOSE calls to a LOB and illustrates how committing a transaction which has open LOBs returns an error.

```

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;

```

## LOBs in Index Organized Tables (IOT)

Index Organized Tables (IOT) now support internal and external LOB columns. The SQL DDL, DML and piece wise operations on LOBs in IOT exhibit the same behavior as for 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 ["Defining Tablespace and Storage Characteristics for Internal LOBs"](#) on page 7-5).

**See Also:** [Chapter 5, "Large Objects: Advanced Topics"](#), ["LOBs in Partitioned Index-Organized Tables"](#).

## Example of Index Organized Table (IOT) with LOB Columns

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

---

---

**Note:** 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 for conventional tables.

---

---

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

---

---

## Manipulating LOBs in Partitioned Tables

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.

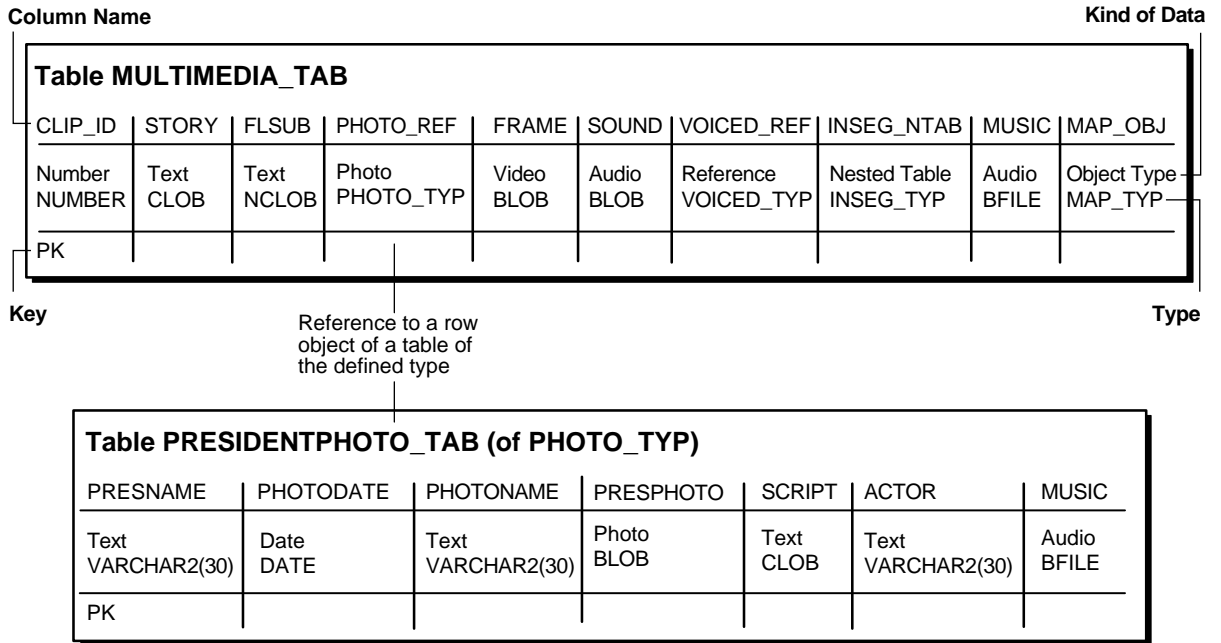
This section describes some of the ways you can manipulate LOBs in partitioned tables.

As an extension to the example multimedia application described in [Appendix B, "The Multimedia Schema"](#), 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 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 structure illustrated in [Figure 7-1](#).

The columns in `Multimedia_tab` are described in [Table 7-5, "Multimedia\\_tab Columns"](#).



**Figure 7-1 Table Multimedia\_tab structure Showing Inclusion of PHOTO\_REF Reference**



**Table 7–5 Multimedia\_tab Columns**

| Column Name | Description  |
|-------------|--|
| PRESNAME    | President's name. This 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   | 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   | 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.<br><br>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   | 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      | 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       | Contains the name of the actor reading the script.   |
| MUSIC       | 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:** In the following examples, 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 the following data structures for certain examples to work:

```

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 or higher, 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-opportunities, 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 the preceding 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);
```

## 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. Oracle8i and higher's extensibility interfaces allow for domain indexing, a framework for implementing such domain specific indexes.

**See Also:** *Oracle9i Data Cartridge Developer's Guide* for information on building domain specific indexes.

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.

**See Also:** *Oracle interMedia User's Guide and Reference* and *Oracle Text Reference*, for more information regarding Oracle's *interMedia* options.

## Functional Indexes on LOB Columns

Oracle9i now supports functional indexing on LOB columns. Analogous to extensible/domain indexes on LOB columns, functional indexes are also automatically updated when a DML operation is performed on the LOB column.

---

---

**Note:** When extensible indexes are being updated, if any functional indexes are present on the LOB column, they are also updated.

---

---

**See Also:** *Oracle9i Application Developer's Guide - Fundamentals*

## SQL Semantics Support for LOBs

This section describes the following topics:

- [How SQL VARCHAR2/RAW Semantics Apply to CLOBs/BLOBs](#)
- [SQL RAW Type and BLOBs](#)
- [SQL DML Changes For LOBs](#)
- [SQL DML Changes For LOBs](#)
- [SQL Functions/Operators for VARCHAR2s/RAWs and CLOBs/BLOBs](#)
- [PL/SQL Statements and Variables: New Semantics Changes](#)
- [PL/SQL CLOB Comparison Rules](#)

In prior releases, you could only access LOBs stored in the database using LOB locators through a set of APIs in various language interfaces (C, C++, OO4O, Java, COBOL, PL/SQL). LOBs could not be used in SQL character functions.

### Improved LOB Usability: Accessing LOBs Using SQL “Character” Functions

In Oracle9i, for the first time, you can access LOBs using SQL VARCHAR2 semantics, such as SQL string operators and functions.

By providing you with an SQL interface, which you are familiar with, accessing LOB data can be greatly facilitated. You can benefit from this added functionality in the following two cases:

- When using small-sized LOBs (~ 10-100K) to store data through the APIs and you need SQL support on LOBs.
- When you have just migrated your LONG columns to LOBs. In this release, you can take advantage of an easier migration process using the LONG-to-LOB migration API described in [Chapter 8, "Migrating From LONGs to LOBs"](#).

Advanced LOB users who need to take advantage of features such as random access and piecewise fetch, should continue using existing LOB API interfaces.

For users of medium-to-large sized (> 1M) LOBs, this SQL interface is not advised due to possible performance issues.

This description is limited to internal persistent LOBs only. This release, does not offer SQL support on BFILEs.

---

---

**Note:** SQL Semantics Support has no impact on current usage of LOBs. Existing LOB applications, using LOB APIs, do not need to be changed.

---

---

## SQL and PL/SQL VARCHAR2 Functions/Operators Now Allowed for CLOBs

The following SQL VARCHAR2 functions and operators are now allowed for CLOBs, as indicated in [Table 7-6](#):

- INSTR related operators/functions
  - INSTR() and variants (See [Table 7-7](#))
  - LIKE
  - REPLACE()
- CONCAT and ||
- LENGTH() and variants (See [Table 7-7](#))
- SUBSTR() and variants (See [Table 7-7](#))
- TRIM(), LTRIM() and RTRIM()
- LOWER(), UPPER(), NLS\_LOWER(), NLS\_UPPER()
- LPAD() and RPAD()

## PL/SQL Relational Operators Now Allowed for LOBs

For LONG to LOB migration, the following relational operators in PL/SQL now work on LONGs and LOBs:

- Operators: >, <, =, !=,
- IN, BETWEEN
- GREATEST and LEAST
- NLSSORT

These operators are also listed in [Table 7-6](#).

## SQL and PL/SQL CHAR to CLOB Conversion Functions

The following CHAR to CLOB conversion functions are now allowed for LOBs:



- TO\_CHAR() and TO\_NCHAR() converts CLOB or NCLOB to CHAR or NCHAR
- TO\_CLOB() and TO\_NCLOB() converts CHAR or NCHAR to CLOB or NCLOB

## Non-Supported SQL Functionality for LOBs

The following SQL functionality is not supported for LOBs because the functions are either infrequently used or have easy workarounds.

- *INDEX on LOB column*: Workaround, use Oracle9i Text (*interMedia Text*)
- *SQL Conversion functions*: TO\_DATE, TO\_NUMBER, TO\_TIMESTAMP, CHARTOROWID, TO\_MULTI\_BYTE, TO\_SINGLE\_BYTE - not frequently used
- *Other SQL functions*: GROUPING - not frequently used.
- *Comparison functions/operators, including the following*:

---



---

**Note:** These operators are allowed in PL/SQL, but do not work in SQL queries issued in PL/SQL blocks.

---



---

- Operators: >, <, =, !=
- IN, SOME, ANY, ALL, BETWEEN
- MAX, MIN, GREATEST and LEAST
- SELECT DISTINCT, GROUP BY, ORDER BY (SORT), UNION, INTERSECT, MINUS
- JOIN
- Miscellaneous SQL functions: INITCAP, NLS\_INITCAP, DUMP, TRANSLATE, VSIZE, DECODE

## Using SQL Functions and Operators for VARCHAR2s on CLOBs

Table 7-6, lists all SQL operators and functions that take a VARCHAR2 as operands/arguments, or return a VARCHAR2 value. With the only exception of the “IS [NOT] NULL” operator, none of the operators/functions in prior releases work on CLOBs.

In [Table 7-6](#), the SQL operators/functions supported on CLOBs in Oracle9i, are indicated in the 4th “SQL” column.

Most functions listed in [Table 7-6](#) also apply to PL/SQL built-in functions (supplied packages). The 5th “PL/SQL” column indicates the availability of the operator/function on CLOBs in PL/SQL.

Implicit conversions between CLOBs and CHAR types are enabled in Oracle9i. Therefore, functions not yet enabled for CLOBs can still accept CLOBs through implicit conversion. In this case, CLOBs are converted to a CHAR or a VARCHAR2 before the function is invoked. If the CLOB is greater than 4K bytes in size, only 4000 bytes will be converted into CHARs or VARCHAR2s.

In [Table 7-6](#), the functions which take CLOB parameters through implicit conversions, are denoted as “CNV”.

**Table 7-6 SQL VARCHAR2 Functions and Operators**

| Category                   | Operator / Function   | SQL Example for CLOB Columns                        | SQL | PL/SQL |
|----------------------------|---|---|-----|--------|
| <b>Concat</b>              | , CONCAT()  | Select clobCol    clobCol2 from tab;                | Yes | Yes    |
| <b>Comparison</b>          | =, !=, >, >=, <, <=, <>, ^=                                     | if clobCol=clobCol2 then...                         | No  | Yes    |
| <b>Comparison</b>          | IN, NOT IN  | if clobCol NOT IN (clob1, clob2, clob3) then...     | No  | Yes    |
| <b>Comparison</b>          | SOME, ANY, ALL  | if clobCol < SOME (select clobCol2 from...) then... | No  | N/A    |
| <b>Comparison</b>          | BETWEEN   | if clobCol BETWEEN clobCol2 and clobCol3 then...    | No  | Yes    |
| <b>Comparison</b>          | LIKE [ESCAPE] and its variants, see <a href="#">Table 7-7</a> . | if clobCol LIKE '%pattern%' then...                 | Yes | Yes    |
| <b>Comparison</b>          | IS [NOT] NULL   | where clobCol IS NOT NULL                           | Yes | Yes    |
| <b>Character Functions</b> | INITCAP, NLS_INITCAP  | select INITCAP(clobCol) from...                     | CNV | CNV    |
| <b>Character Functions</b> | LOWER, NLS_LOWER, UPPER, NLS_UPPER                              | ...where LOWER(clobCol1) = LOWER(clobCol2)          | Yes | Yes    |
| <b>Character Functions</b> | LPAD, RPAD  | select RPAD(clobCol, 20, 'La') from...              | Yes | Yes    |
| <b>Character Functions</b> | TRIM, LTRIM, RTRIM  | ...where RTRIM(LTRIM(clobCol,'ab'), 'xy') = 'cd'    | Yes | Yes    |

**Table 7-6 SQL VARCHAR2 Functions and Operators (Cont.)**

| Category                   | Operator / Function                                      | SQL Example for CLOB Columns   | SQL | PL/SQL |
|----------------------------|--|--|-----|--------|
| <b>Character Functions</b> | REPLACE  | select REPLACE(clobCol, 'orig','new') from...                                      | Yes | Yes    |
| <b>Character Functions</b> | SOUNDEX  | ...where SOUNDEX(clobCol) = SOUNDEX('SMYTHE')                                      | CNV | CNV    |
| <b>Character Functions</b> | SUBSTR and its variants, see <a href="#">Table 7-7</a> . | ...where substr(clobCol, 1,4) = 'THIS'   | Yes | Yes    |
| <b>Character Functions</b> | TRANSLATE  | select TRANSLATE(clobCol, '123abc','NC') from...                                   | CNV | CNV    |
| <b>Character Functions</b> | ASCII  | select ASCII(clobCol) from...  | CNV | CNV    |
| <b>Character Functions</b> | INSTR and its variants, see <a href="#">Table 7-7</a> .  | ...where instr(clobCol, 'book') = 11   | Yes | Yes    |
| <b>Character Functions</b> | LENGTH and its variants, see <a href="#">Table 7-7</a> . | ...where length(clobCol) != 7;   | Yes | Yes    |
| <b>Character Functions</b> | NLSSORT  | ...where NLSSORT (clobCol,'NLS_SORT = German') > NLSSORT ('S','NLS_SORT = German') | CNV | CNV    |
| <b>Conversion</b>          | CHARTOROWID  | CHARTOROWID(clobCol)   | CNV | CNV    |
| <b>Conversion</b>          | HEXTORAW   | HEXTORAW(CLOB)   | No  | CNV    |
| <b>Conversion</b>          | CONVERT  | select CONVERT(clobCol,'WE8DEC','WE8HP') from...                                   | Yes | CNV    |
| <b>Conversion</b>          | TO_DATE  | TO_DATE(clobCol)   | CNV | CNV    |
| <b>Conversion</b>          | TO_NUMBER  | TO_NUMBER(clobCol)   | CNV | CNV    |
| <b>Conversion</b>          | TO_TIMESTAMP   | TO_TIMESTAMP(clobCol)  | No  | CNV    |
| <b>Conversion</b>          | TO_MULTI_BYTE<br>TO_SINGLE_BYTE                          | TO_MULTI_BYTE(clobCol)<br>TO_SINGLE_BYTE(clobCol)                                  | CNV | CNV    |
| <b>Conversion</b>          | TO_CHAR  | TO_CHAR(clobCol)   | Yes | Yes    |
| <b>Conversion</b>          | TO_NCHAR   | TO_NCHAR(clobCol)  | Yes | Yes    |

**Table 7–6 SQL VARCHAR2 Functions and Operators (Cont.)**

| Category                   | Operator / Function | SQL Example for CLOB Columns  | SQL | PL/SQL |
|----------------------------|---------------------|---|-----|--------|
| <b>Conversion</b>          | TO_LOB              | INSERT INTO... SELECT TO_LOB(longCol)...  | N/A | N/A    |
|                            |                     | <b>Note:</b> TO_LOB can only be used to create or insert into a table with LOB columns as SELECT FROM a table with a LONG column. |     |        |
| <b>Conversion</b>          | TO_CLOB             | TO_CLOB(varchar2Col)  | Yes | Yes    |
| <b>Conversion</b>          | TO_NCLOB            | TO_NCLOB(varchar2Clob)  | Yes | Yes    |
| <b>Aggregate Functions</b> | COUNT               | select count(clobCol) from...   | No  | N/A    |
| <b>Aggregate Functions</b> | MAX, MIN            | select MAX(clobCol) from...   | No  | N/A    |
| <b>Aggregate Functions</b> | GROUPING            | select grouping(clobCol) from... group by cube (clobCol);   | No  | N/A    |
| <b>Other Functions</b>     | GREATEST, LEAST     | select GREATEST (clobCol1, clobCol2) from...  | No  | CNV    |
| <b>Other Functions</b>     | DECODE              | select DECODE(clobCol, condition1, value1, defaultValue) from...  | CNV | CNV    |
| <b>Other Functions</b>     | NVL                 | select NVL(clobCol,'NULL') from...  | Yes | Yes    |
| <b>Other Functions</b>     | DUMP                | select DUMP(clobCol) from...  | No  | N/A    |
| <b>Other Functions</b>     | VSIZE               | select VSIZE(clobCol) from...   | No  | N/A    |

**See Also:** *Oracle9i SQL Reference*, Chapter 6, "Functions".

## UNICODE Support for VARCHAR2 and CLOB

In this release, database UNICODE support for VARCHAR2s [unicode] provides a few UNICODE variants on functions INSTR, SUBSTR, LENGTH, and LIKE. These UNICODE functions are not yet supported on CLOBs or NCLOBs as summarized in [Table 7–7](#).

**See Also:**

- *Oracle9i Supplied PL/SQL Packages and Types Reference*
- *Oracle9i Application Developer's Guide - Fundamentals*
- *Oracle9i SQL Reference*
- *Oracle9i Database Globalization Support Guide*

for a detailed description on the usage of UNICODE functions.

**Table 7-7 Unicode Related SQL Functions (CLOB=CLOB Support)**

| SQL Functions                   | Comments  | CLOB |
|---------------------------------|---|------|
| INSTRB, SUBSTRB, LENGTHB        | Byte-based functions, existed in prior in Oracle9i releases | No   |
| INSTR2, SUBSTR2, LENGTH2, LIKE2 | UCS2 character set based, provided in this release          | No   |
| INSTR4, SUBSTR4, LENGTH4, LIKE4 | UCS4 character set based, provided in this release          | No   |
| INSTRC, SUBSTRC, LENGTHC, LIKEC | Character based, provided in this release                   | No   |

## SQL Features Where LOBs Cannot be Used

[Table 7-8](#) lists other SQL features where LOBs cannot be used. Refer to the "[LOB Restrictions](#)" section in [Chapter 4, "Managing LOBs"](#), for further details.

**Table 7-8 SQL Features Where LOBs Cannot be Used**

| SQL Feature               | Example for CLOB Columns                    |
|---------------------------|---|
| SELECT DISTINCT           | SELECT DISTINCT clobCol from...             |
| SELECT clause<br>ORDER BY | SELECT... ORDER BY clobCol                  |
| SELECT clause<br>GROUP BY | SELECT avg(num) FROM...<br>GROUP BY clobCol |

**Table 7–8 SQL Features Where LOBs Cannot be Used (Cont.)**

| SQL Feature                           | Example for CLOB Columns                                   |
|---------------------------------------|--|
| UNION, INTERSECT, MINUS               | SELECT clobCol1 from tab1 UNION SELECT clobCol2 from tab2; |
| <b>Note:</b> UNION ALL works for LOBs |  |
| JOIN                                  | SELECT... FROM... WHERE tab1.clobCol = tab2.clobCol        |
| INDEX                                 | CREATE INDEX clobIndx ON tab(clobCol)...                   |

## How SQL VARCHAR2/RAW Semantics Apply to CLOBs/BLOBs

### Defining CHAR Buffer on CLOB

With Oracle9i, you can retrieve data from LOBs directly using SQL without using any special LOB API.

In PL/SQL, you can define a VARCHAR2 for a CLOB and RAW for a BLOB column. You can also define CLOBs/BLOBs for VARCHAR2/RAW columns.

#### Selecting a CLOB Column into a CHAR Buffer or CLOB

In PL/SQL, if a CLOB column is selected into a local VARCHAR2 variable, data stored in the CLOB column is retrieved and put into the CHAR buffer. If the buffer is not large enough to contain all the CLOB data a truncation error is raised and no data is written to the buffer. After the SELECT, the VARCHAR2 variable behaves the same as a regular character buffer.

In contrast, when a CLOB column is selected into a local CLOB variable, the CLOB locator is fetched. PL/SQL built-in functions that previously took only VARCHAR2s are now enabled to also take CLOB locators as arguments. The return type of the functions is CLOB if the primary argument is a CLOB. At the same time, the CLOB local variable can behave as a LOB locator when passed to DBMS\_LOB APIs.

The preceding statement also applies to RAWs and BLOBs.

### Accepting CLOBs in VARCHAR2 Operators/Functions

SQL operators/functions that currently take VARCHAR2 columns as operands or arguments are now enabled to accept CLOB columns. Previously, in Oracle8i, comparison of LOBs was not allowed, except for comparing LOB functions and the 'IS [NOT] NULL' operator on LOBs. In this release, comparison of LOBs themselves

in PL/SQL is allowed, while comparison in SQL queries is not yet available for performance concerns.

## Returning CLOB Values from SQL Functions/Operators

SQL operators/functions that previously returned VARCHAR2s, now either return a CLOB or a VARCHAR2, depending on the input parameter type.

### Returning VARCHAR2s

Operators/functions continue to return VARCHAR2s when only VARCHAR2s are passed in as arguments. A function with only VARCHAR2 parameters never returns a CLOB.

### Returning CLOBs

Operators/functions return CLOBs when the primary argument, usually the first parameter is passed in as CLOBs. For example, the following SQL statements select out results as CLOB types:

```
SELECT SUBSTR(clobCol, 1,4) FROM .... WHERE LENGTH(clobCol)>4;
SELECT clobCol || charCol FROM ...;
```

---

---

**Note:** For functions like CONCAT(), ||, which do not have a well-defined primary argument, if any parameter is a LOB, the function returns a LOB.

---

---

## Returned LOB is a Temporary LOB Locator

When a LOB is returned, the result from the select list is in the form of a temporary LOB locator. Your application should view the temporary LOB as local storage for the CHAR string returned from the SELECT. In PL/SQL, the temporary LOB has the same lifetime (duration) as other local PL/SQL program variables. It can be passed to subsequent SQL or PL/SQL VARCHAR2 functions or queries:

- As a PL/SQL local variable, the temporary LOB will go out of scope at the end of the residing program block and then the LOB data will automatically be freed. This is the same behavior as other PL/SQL VARCHAR2 variables. At any time, nonetheless, you can issue a `DBMS_LOB.FREETEMPORARY()` call to release the resources taken by the local temporary LOBs.

- In OCI, the temporary LOBs returned from SQL queries are always in 'session' duration, unless a user-defined duration is present, in which case, the temporary LOBs will be in the user-defined duration.

---

---

**ALERT: Ensure that your temporary tablespace is large enough to store all temporary LOB results returned from queries in your program(s).**

---

---

Alternatively, if any of the following transpire:

- You select out a CLOB column into a VARCHAR2
- A function that returns CLOB is put into a VARCHAR2 buffer

the returned result is a regular CHAR buffer with the declared size. If the VARCHAR2 buffer is not large enough to fit the data from the LOB, a truncation error is raised.

### SQL Query Example 1: Using SQL to SELECT out a CLOB into a VARCHAR2

The following example illustrates selecting out a CLOB column into a VARCHAR2 and returning the result as a CHAR buffer of declared size:

```
DECLARE
  vc1 VARCHAR2(32000);
  lb1 CLOB;
  lb2 CLOB;
BEGIN
  SELECT clobCol1 INTO vc1 FROM tab WHERE colID=1;
  -- lb1 is a temporary LOB
  SELECT clobCol2 || clobCol3 INTO lb1 FROM tab WHERE colID=2;

  lb2 := vc1 || lb1;
  -- lb2 is a still temporary LOB, so the persistent data in the database
  -- is not modified. An update is necessary to modify the table data.
  UPDATE tab SET clobCol1 = lb2 WHERE colID = 1;

  DBMS_LOB.FREETEMPORARY(lb2); -- Free up the space taken by lb2

  <... some more queries ...>

  END; -- at the end of the block, lb1 is automatically freed
```



## IS [NOT] NULL in VARCHAR2s and CLOBs

For LOB columns, operator “IS [NOT] NULL” has been allowed since Oracle8. It checks if there is a LOB locator stored in the table row.

For VARCHAR2 columns, operator “IS NULL” indicates an empty string, or a null string.

---



---

### Note: IS NULL Semantic Discrepancy

In the SQL 92 standard, a character string of length zero is distinct from a null string.

For an initialized LOB of length 0, you should expect ‘IS NULL’ to return zero (FALSE), since it is the correct and standard compliant behavior. In contrast, a VARCHAR2 of length 0 returns TRUE on ‘IS NULL’.

In addition, for the LENGTH() function:

- If the input is a character string of zero length, LENGTH() returns *NULL*.
- For a CLOB of zero length, an EMPTY\_CLOB(), *zero* is returned by LENGTH and DBMS\_LOB.GETLENGTH() in SQL and PL/SQL.

This can be misleading. Note this semantic discrepancy.

---



---

## SQL RAW Type and BLOBs

SQL RAW types and BLOBs are handled as follows:

- *In PL/SQL:* BLOBs to be selected into a RAW.
- *In OCI and other interfaces:* Defining any local memory buffer, such as a char array in C, on a BLOB is enabled. You can define a buffer of any type in a C program for a BLOB column as SQLT\_RAW type.

## SQL DML Changes For LOBs

In Oracle9i, there has been no significant change to SQL DML with regards to LOBs. The only related change is in the WHERE clause of UPDATE and DELETE. In prior releases, Oracle did not allow LOBs in the WHERE clause of UPDATE, DELETE, and SELECT. Now, SQL functions of LOBs that do not involve comparing LOB

values, are allowed in predicates of the WHERE. clause. For example, `length()` and `insert()`.

## SQL Functions/Operators for VARCHAR2s/RAWs and CLOBs/BLOBs

As listed in [Table 7-6](#) through [Table 7-8](#), the SQL functions/operators for VARCHAR2s/RAWs have been extended to work on CLOB or BLOB columns.

The return type of the SQL functions depend on the input type. Refer to "[Returning CLOB Values from SQL Functions/Operators](#)" on page 7-41 for a detailed discussion.

The following examples show queries that benefit from the VARCHAR2 semantics on CLOBs. In prior releases, the effects of these queries used to be achieved, in PL/SQL code, using DBMS\_LOB calls. It will be convenient for you to be able to use the same interface as VARCHAR2s to access data.

---

---

**Note:** These examples are based on the following revised version of the Multimedia application schema described in [Appendix B, "The Multimedia Schema"](#) and [Chapter 10, "Internal Persistent LOBs"](#) under "[Creating a Table Containing One or More LOB Columns](#)":

```
CREATE TABLE Multimedia_tab (  
    Clip_ID NUMBER NOT NULL,  
    Story          CLOB default EMPTY_CLOB(),  
    Gist           VARCHAR2(100),  
    .....  
}
```

---

---

### SQL Query Example 2: A few SQL queries on CLOBs

```
SELECT Gist||Story FROM Multimedia_tab WHERE Story LIKE Gist;
```

```
SELECT SUBSTR(Story, 20, 1), LENGTH(Story) FROM Multimedia_tab WHERE Gist NOT IN  
Story;  
-- A temp LOB is created and returned for 'Gist||Story' and 'SUBSTR(Story,20,1)' because story is a  
CLOB.
```

## PL/SQL Statements and Variables: New Semantics Changes

In PL/SQL, a number of semantic changes have been made as described in the previous paragraphs.

---

---

**Note:** The following discussions, concerning CLOBs and VARCHAR2s, also apply to BLOBs and RAWs, unless otherwise noted. In the text, BLOB and RAW are not explicitly mentioned.

---

---

The new PL/SQL semantics support is described in the following sections as follows:

- [Implicit Conversions Between CLOB and VARCHAR2](#)
  - [PL/SQL Example 1: Prior Release SQL Interface for a CLOB/VARCHAR2 Application](#)
  - [PL/SQL Example 2: Accessing CLOB Data When Treated as VARCHAR2s](#)
  - [PL/SQL Example 3: Defining a CLOB Variable on a VARCHAR2](#)
- [Explicit Conversion Functions](#)
- [VARCHAR2 and CLOB in PL/SQL Built-in Functions](#)
  - [PL/SQL Example 4: CLOB Variables in PL/SQL](#)
  - [PL/SQL Example 5: Change in Locator-Data Linkage](#)
  - [PL/SQL Example 6: Freeing Temporary LOBs Automatically and Manually](#)
- [PL/SQL CLOB Comparison Rules](#)
- [Interacting with SQL and PL/SQL in OCI and Java Interfaces](#)

### Implicit Conversions Between CLOB and VARCHAR2

The implicit conversion in both directions, from CLOB to VARCHAR2, and from VARCHAR2 to CLOB, have made the following operations between CLOBs and VARCHAR2s possible:

- CLOB columns can be selected into VARCHAR2 PL/SQL variables
- VARCHAR2 columns can be selected into CLOB variables
- Assignment and parameter passing between CLOBs and VARCHAR2s

## PL/SQL Example 1: Prior Release SQL Interface for a CLOB/VARCHAR2 Application

The following example illustrates the way CLOB data was accessed prior to this release. This application tries to simply display both the Gist and Story from the table `Multimedia_tab`.

```
declare
  myStoryLOB CLOB;
  myStoryBuf VARCHAR2(4001);
  amt NUMBER:=4001;
  offset NUMBER := 1;
begin
  SELECT Story INTO myStoryLOB FROM Multimedia_tab WHERE Clip_ID = 10;
  DBMS_LOB.READ(myStoryLOB, amt, offset, myStoryBuf);
  -- Display Gist and Story by printing 'myStoryBuf'.
end;
```

## PL/SQL Example 2: Accessing CLOB Data When Treated as VARCHAR2s

The following example illustrates the way CLOB data is accessed with this release when the CLOBs are treated as VARCHAR2s:

```
declare
  myStoryBuf VARCHAR2(4001);
begin
  SELECT Story INTO myStoryBuf FROM Multimedia_tab WHERE Clip_ID = 10;
  -- Display Story by printing myStoryBuf directly
end;
```

## PL/SQL Example 3: Defining a CLOB Variable on a VARCHAR2

```
declare
  myGistLOB CLOB;
begin
  SELECT Gist INTO myGistLOB FROM Multimedia_tab WHERE Clip_ID = 10;
  -- myGistLOB is a temporary LOB.
  -- Use myGistLOB as a lob locator
end;
```

---

---

**Note:** In prior releases, in PL/SQL, you had to first issue the `DBMS_LOB.CREATE_TEMPORARY( )` call before using the temporary LOB. From this release, the temporary LOB is created implicitly in 'assignments' and 'defines.'

---

---

## Explicit Conversion Functions

In SQL and PL/SQL, the following new explicit conversion functions have been added to convert other data types to CLOB, NCLOB, and BLOB as part of the LONG-to-LOB migration:

- `TO_CLOB()`: Converting from `VARCHAR2`, `NVARCHAR2`, or `NCLOB` to a `CLOB`
- `TO_NCLOB`: Converting from `VARCHAR2`, `NVARCHAR2`, or `CLOB` to an `NCLOB`
- `TO_BLOB()`: Converting from `RAW` to a `BLOB`
- `TO_CHAR()` is enabled to convert a `CLOB` to a `CHAR` type.
- `TO_NCHAR()` is enabled to convert an `NCLOB` to a `NCHAR` type.

Other explicit conversion functions are not supported in this release, such as, `TO_NUMBER()`, see [Table 7-6](#). Conversion function details are explained in [Chapter 8](#), "Migrating From LONGs to LOBs".

## VARCHAR2 and CLOB in PL/SQL Built-in Functions

`CLOB` and `VARCHAR2` are still two distinct types. But depending on the usage, a `CLOB` can be passed to SQL and PL/SQL `VARCHAR2` built-in functions, behaving exactly like a `VARCHAR2`. Or the variable can be passed into `DBMS_LOB` APIs, acting like a LOB locator. Please see the following combined example, "[PL/SQL Example 4: CLOB Variables in PL/SQL](#)".

PL/SQL `VARCHAR2` functions/operators need to take `CLOBs` as argument or operands.

When the size of a `VARCHAR2` variable is not large enough to contain the result from a function that returns a `CLOB`, or a `SELECT` on a `CLOB` column, an error should be raised and no operation will be performed. This is consistent with current `VARCHAR2` behavior.

## PL/SQL Example 4: CLOB Variables in PL/SQL

```
1 declare
2   myStory CLOB;
3   revisedStory CLOB;
4   myGist VARCHAR2(100);
5   revisedGist VARCHAR2(100);
6 begin
7   -- select a CLOB column into a CLOB variable
8   SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;
9   -- perform VARCHAR2 operations on a CLOB variable
10  revisedStory := UPPER(SUBSTR(myStory, 100, 1));
11  -- revisedStory is a temporary LOB
12  -- Concat a VARCHAR2 at the end of a CLOB
13  revisedStory := revisedStory || myGist;

14  -- The following statement will raise an error since myStory is
15  -- longer than 100 bytes
16  myGist := myStory;
17 end;
```

Please note that in line 10 of ["PL/SQL Example 4: CLOB Variables in PL/SQL"](#), a temporary CLOB is implicitly created and is pointed to by the `revisedStory` CLOB locator. In the current interface the line can be expanded as:

```
buffer VARCHAR2(32000)
DBMS_LOB.CREATETEMPORARY(revisedStory);
buffer := UPPER(DBMS_LOB.SUBSTR(myStory,100,1));
DBMS_LOB.WRITE(revisedStory,length(buffer),1, buffer);
```

In line 13, `myGist` is appended to the end of the temporary LOB, which has the same effect of:

```
DBMS_LOB.WRITEAPPEND(revisedStory, myGist, length(myGist));
```

In some occasions, implicitly created temporary LOBs in PL/SQL statements can change the representation of LOB locators previously defined. Consider the next example.

## PL/SQL Example 5: Change in Locator-Data Linkage

```
1 declare
2   myStory CLOB;
3   amt number:=100;
4   buffer VARCHAR2(100):='some data';
```

```

5 begin
6 -- select a CLOB column into a CLOB variable
7 SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;
8 DBMS_LOB.WRITE(myStory, amt, 1, buf);
9 -- write to the persistent LOB in the table
10
11 myStory:= UPPER(SUBSTR(myStory, 100, 1));
12 -- perform VARCHAR2 operations on a CLOB variable, temporary LOB created.
Changes
13 -- will not be reflected in the database table from this point on.
14
15 update Multimedia_tab set Story = myStory WHERE clip_id = 10;
16 -- an update is necessary to synchronize the data in the table.
17 end;

```

After line 7, `myStory` represents a persistent LOB in `Multimedia_tab`.

The `DBMS_LOB.WRITE()` call in line 8 directly writes the data to the table.

No `UPDATE` statement is necessary. Subsequently in line 11, a temporary LOB is created and assigned to `myStory` because `myStory` should now behave like a local `VARCHAR2` variable. The LOB locator `myStory` now points to the newly-created temporary LOB.

Therefore, modifications to `myStory` will no longer be reflected in the database. To propagate the changes to the database table, an `UPDATE` statement becomes necessary now. Note again that for the previous persistent LOB, the `UPDATE` is not required.

Temporary LOBs created in a program block as a result of a `SELECT` or an assignment are freed automatically at the end of the PL/SQL block/function/procedure. You can choose to free the temporary LOBs to reclaim system resources and temporary tablespace by calling `DBMS_LOB.FREETEMPORARY()` on the CLOB variable.

## PL/SQL Example 6: Freeing Temporary LOBs Automatically and Manually

```

declare
  Story1 CLOB;
  Story2 CLOB;
  StoryCombined CLOB;
  StoryLower CLOB;
begin
  SELECT Story INTO Story1 FROM Multimedia_tab WHERE Clip_ID = 1;
  SELECT Story INTO Story2 FROM Multimedia_tab WHERE Clip_ID = 2;

```

```
StoryCombined := Story1 || Story2; -- StoryCombined is a temporary LOB
-- Free the StoryCombined manually to free up space taken
DBMS_LOB.FREETEMPORARY(StoryCombined);
StoryLower := LOWER(Story1) || LOWER(Story2);
end; -- At the end of block, StoryLower is freed.
```

## PL/SQL CLOB Comparison Rules

Like VARCHAR2s, when a CLOB is compared with another CLOB or compared with a VARCHAR2, a set of rules determines the comparison. The rules are usually called a "collating sequence". In Oracle, CHARs and VARCHAR2s have slightly different sequences due to the blank padding of CHARs.

### CLOBs Follow the VARCHAR2 Collating Sequence

As a rule, CLOBs follow the same collating sequence as VARCHAR2s. That is, when a CLOB is compared, the result is consistent with if the CLOB data content is retrieved into a VARCHAR2 buffer and the VARCHAR2 is compared. The rule applies to all cases including comparisons between CLOB and CLOB, CLOB and VARCHAR2, and CLOB and CHAR.

---

---

**Note:** When a CLOB is compared with a CHAR string, it is always the *character* data of the CLOB being compared with the string. Likewise, when two CLOBs are compared, the data content of the two CLOBs are compared, *not their LOB locators*.

---

---

It makes no sense to compare CLOBs with non-character data, or with BLOBs. An error is returned in these cases.

## Interacting with SQL and PL/SQL in OCI and Java Interfaces

The OCI and Java interfaces now provide the ability to bind and define VARCHAR2 variables to SQL and PL/SQL statements with LOBs.

### See Also:

- *Oracle Call Interface Programmer's Guide*
- *Oracle9i JDBC Developer's Guide and Reference*



This release does not provide the ability to define variables in the other direction, that is, defining a client-side CLOB locator on a VARCHAR2 column.

---

---

**Note:** In OCI, from a SQL query, temporary LOBs are generally returned in 'session' duration.

---

---

## Performance Attributes When Using SQL Semantics with LOBs

Be aware of the following performance issues, when using SQL semantics with LOBs.

### Inserting More than 4K Bytes Data Into LOB Columns

In Oracle9i, the maximum length restriction for all column data and buffer size when processing SQL queries, can be more than 4K bytes. You can process LOB data, which can be as long as 4G bytes, in SQL!

Temporary LOBs are used internally if the data is greater than 4K bytes to store intermediate results.

---

---

**Note:** This could degrade performance. The extra load in query processing comes from both the cost of dealing with the larger amount of intermediate results and the lower efficiency of accessing temporary LOBs.

---

---

For large VARCHARs, SQL queries now perform in a similar fashion to when accessing CLOBs through the previous set of LOB APIs.

### Temporary LOB Creation/Deallocation

In PL/SQL, C (OCI), and Java, SQL query results return temporary LOBs for operation/function calls on LOB columns. For example:

```
SELECT substr(CLOB_Column, 4001, 32000) FROM ...
```

Returned temporary LOBs automatically get freed at the end of a PL/SQL program block.

You can choose to free any unneeded temporary LOBs at any time to free up system resources and temporary tablespace. Without proper deallocation of the temporary

LOBs returned from SQL queries, temporary tablespace gets filled up steadily and you could observe performance degradation. See "[PL/SQL Example 6: Freeing Temporary LOBs Automatically and Manually](#)", for an example of freeing temporary LOBs explicitly.

## Performance Measurement

The performance of an SQL query execution on CLOB columns should be compared to that of a query on VARCHAR2s or LONGs of the same size. Expect the performance on LOBs to be within 80% of VARCHAR2s/LONGs or better.

---

---

**Note: System/Database Management:** After this newly provided enhanced SQL semantics functionality is used in your applications, there will be many more temporary LOBs created silently in SQL and PL/SQL than before. *Ensure that temporary tablespace for storing these temporary LOBs is large enough for your applications!*

---

---

## User-Defined Aggregates and LOBs

User-defined aggregates (UDAGs) provide a mechanism for application/cartridge developers, and end-users to implement and deploy new aggregate functions over scalar data types (including LOBs) as well as object and opaque types.

The following are two examples of applications for user-defined aggregates (UDAGs):

- **Spatial Cartridge.** In Oracle Spatial cartridge, several functions return geometries as the result of a spatial query. These functions can only be aggregated using user-defined aggregate functions. In situations, such as web delivery, it could be efficient to aggregate the results of a query and send one aggregate geometry instead of sending several individual geometries.

For example, a query to find the state boundary by unionizing the county boundaries in each state can be executed as follows:

```
SELECT SDO_AGGR_UNION(county.geometry)
FROM COUNTIES
GROUP BY county.state;
```

- **Trusted Oracle.** In Trusted Oracle, the row label is modelled as an opaque type LBAC\_LABEL. To define aggregate functions over rowlabels such as greatest lower bound (GLB) and least upper bound (LUB), queries such as the following, can then be executed efficiently.

```
SELECT group_column, GLB(rowlabel)
FROM x
GROUP BY group_column.
```

User Defined Aggregate functions (UDAG) refer to aggregate functions with user specified aggregation semantics. You can create a new aggregate function and provide the aggregation logic using a set of routines. Once created, user defined aggregate functions can be used in SQL DML statements like built-in aggregates.

Complex data is typically stored in the database using object types, opaque types or LOBs. User-defined aggregates are useful in specifying aggregation over these domains of data.

UDAGs can also be used to create new aggregate functions over traditional scalar data types for financial or scientific applications. Since, it is not possible to provide native support for all forms of aggregates, this functionality provides you with a flexible way to add new aggregate functions.

An aggregate function takes a set of values as input and returns a single value. The sets of values for aggregation are typically identified using a GROUP BY clause. For example:

```
SELECT AVG(T.Sales)
   FROM AnnualSales T
   GROUP BY T.State
```

UDAGs allow you to register new aggregate functions by providing specific (new) implementations for the preceding primitive operations.

## UDAGs: DDL Support

User-defined aggregate functions have the following DDL support:

- Creating and dropping aggregate functions over scalar data types (including LOBs) and user-defined data types (object types and opaque types).
- Specifying the aggregation logic in form of user supplied routines.
- Specification of implementation routines in any combination of the following languages: PL/SQL, C/C++ or Java.

## UDAGs: DML and Query Support

User-defined aggregate functions have the following DML and query support:

- SQL statements which invoke the UDAG as part of an expression in the SELECT list or as part of the predicate in the HAVING clause
- Both serial and parallel evaluations of the UDAG
- Allow specification of DISTINCT and ALL (default) keywords on the input parameter to the UDAG
- UDAGs with group by extensions - CUBE, ROLLUP and grouping sets
- Materialized views with UDAGs with memory-less refresh option
- Windowing functions with UDAGs

**See Also:** *Oracle9i Data Cartridge Developer's Guide*

---

---

**Note:** In PL/SQL, UDAGs cannot be used in procedural statements, but can be used in embedded SQL.

---

---

---

# Migrating From LONGs to LOBs

This chapter contains the following topics:

- [Introducing LONG-to-LOB Migration](#)
- [Guidelines for Using LONG-to-LOB API](#)
- [Migrating Existing Tables from LONG to LOB](#)
- [LONG-to-LOB Migration Limitations](#)
- [Using LONG-to-LOB API with OCI](#)
- [Using SQL and PL/SQL to Access LONGs and LOBs](#)
- [Applications Requiring Changes When Converting From LONGs to LOBs](#)
- [Using utldtree.sql to Determine Where Your Application Needs Change](#)
- [Examples of Converting from LONG to LOB Using Table Multimedia\\_tab](#)
- [Summary of New Functionality Associated with the LONG-to-LOB API](#)
- [Compatibility and Migration](#)
- [Performance](#)
- [Frequently Asked Questions \(FAQs\): LONG to LOB Migration](#)

## Introducing LONG-to-LOB Migration

To assist you in migrating to LOBs, Oracle supports the LONG API for LOBs. This API ensures that when you change your LONG columns to LOBs, your existing applications will require few changes, if any.

The term, “LONG API”, refers to DML and querying of LONG datatypes. Examples of the LONG API are:

- *For OCI:* OCIBindByName(), OCIBindDynamic(), OCIDefineByPos(), OCIDefineDynamic(),....
- *For PL/SQL:* substr, instr,... and parameter passing

---

---

**Note:** The "LONG API" applies to other datatypes besides LONGs. In this chapter, however, we are specifically interested in this API for LOBs. We refer to it here as the "LONG-to-LOB API".

---

---

Oracle9i supports LONG as well as LOB datatypes. When possible, change your existing applications to use LOBs instead of LONGs because of the added benefits that LOBs provide. See [Chapter 7, "Modeling and Design", "LOBs Compared to LONG and LONG RAW Types"](#) on page 7-2.

This chapter describes how the "LONG API" referred to here as "LONG-to-LOB API", is used for LOBs.

---

---

**Note:** The performance of some LOB operations improves with the LONG-to-LOB API. See "[Performance](#)" on page 8-42 for details.

---

---

## Using the LONG-to-LOB API Results in an Easy Migration

LONG-to-LOB migration allows you to easily migrate your existing applications that access LONG columns, to use LOB columns. The migration has two parts:

- *Data migration.* This consists of the procedure to move existing tables containing LONG columns to use LOBs. This is described in "[Migrating Existing Tables from LONG to LOB](#)" on page 8-6.
- *Application migration.* This specifies how existing LONG applications will change for using LOBs. Your application will only need to change in very rare cases. The LONG-to-LOB API that is implemented for LOBs is described in

["Using LONG-to-LOB API with OCI"](#) on page 8-12 and ["Using SQL and PL/SQL to Access LONGs and LOBs"](#) on page 8-16.

---

---

**Note:** LONGs have various restrictions, such as, there can be at most, only one LONG column in a table,....

On the other hand, LOBs do not have such restrictions. After you migrate your LONG tables to use LOBs, you will no longer have these LONG restrictions. You can have multiple LOB columns in a table, and do multiple >4k binds in a single INSERT/UPDATE,....

---

---

## Guidelines for Using LONG-to-LOB API

The following are guidelines for using LONG-to-LOB API.

### Using ALTER TABLE

Use ALTER TABLE to convert LONG columns in existing tables to LOBs. See ["Migrating LONGs to LOBs: Using ALTER TABLE to Change LONG Column to LOB Types"](#) on page 8-6.

### LONG-to-LOB API and OCI

#### Binds in OCI

Previously, a VARCHAR2 buffer of more than 4000 bytes of data could only be bound to a LONG column. The LONG-to-LOB API now allows this functionality for LOBs. It works for the following:

- Regular, piecewise, and callback binds for INSERTs and UPDATES
- Array binds for INSERTs and UPDATES
- Parameter passing or across PL/SQL and OCI boundaries

The following OCI functions are part of the LONG-to-LOB API:

- OCIBindByPos()
- OCIBindByName()
- OCIBindDynamic()
- OCISstmtSetPieceInfo()

- OCISstmtGetPieceInfo()

They accept the following datatypes for inserting or updating LOB columns:

- SQLT\_CHR and SQLT\_LNG for CLOB columns
- SQLT\_BIN and SQLT\_LBI for BLOB columns

**See Also:**

- ["Using OCI Functions to Perform INSERT or UPDATE on LOBs"](#) on page 8-13
- ["PL/SQL and C Bindings from OCI"](#) on page 8-20

**Defines in OCI**

The LONG-to-LOB API allows the following OCI functions to accept VARCHAR2 buffer and SQLT\_CHR, SQLT\_LNG, SQLT\_LBI, and SQLT\_BIN datatypes as LOB column outputs:

- OCIDefineByPos()
- OCIDefineDynamic()
- OCISstmtSetPieceInfo()
- OCISstmtGetPieceInfo()

When you do this, the LOB data (and not the locator) is selected into your buffer.

---

---

**Note:** In the OCI LONG-to-LOB API, you cannot specify the amount you want to read. You can only specify the buffer length of your buffer. So Oracle just reads whatever amount fits into your buffer.

---

---

**OCI Functions Allow Piecewise and Array INSERT, UPDATE, or Fetch on LOBs**

The OCI functions mentioned earlier, allow piecewise INSERT, UPDATE, or fetch, and array INSERT, UPDATE, or fetch on LOBs. They allow you to provide data dynamically at run-time for INSERTs and UPDATEs into LOBs.

The bind (INSERT and UPDATE) functions worked for LOBs in prior releases in the same way as they do for LONGs.

**See Also:** See ["Using OCI Functions to Perform INSERT or UPDATE on LOBs"](#) on page 8-13.



Defines (SELECT) now work for LOBs in regular, piecewise, callback, and array mode.

**See Also:** ["Using OCI Functions to Perform FETCH on LOBs"](#) on page 8-14.

### **Multibyte Charactersets (OCI)**

When the Client's character set is multibyte, these functions behave the same as for LONGs.

- For a *piecewise* fetch in a multibyte character set, a multibyte character could be cut in middle, with some bytes in one buffer and remaining bytes in the next buffer.
- For *regular* fetch, if the buffer cannot hold all bytes of the last character, then Oracle returns as many bytes as fit into the buffer, hence returning partial characters.

## **LONG-to-LOB API and PL/SQL**

### **INSERT and UPDATE of LOB Columns (PL/SQL)**

In prior releases, in PL/SQL, you could INSERT or UPDATE the following:

- Character data, such as VARCHAR2, CHAR, or LONG, into a CLOB column
- Binary data, such as RAW or LONG RAW, into a BLOB column

See ["PL/SQL Interface"](#) on page 8-40.

### **SELECT on a LOB Column (PL/SQL)**

PL/SQL accepts SELECT statements on a CLOB column, where, a character variable, such as VARCHAR2, CHAR, or LONG, is provided in the INTO clause. See ["Using SQL and PL/SQL to Access LOBs"](#) on page 8-16. The same holds for selecting a BLOB column into a binary variable, such as RAW or LONG RAW.

---

---

**Note:** In the PL/SQL LONG-to-LOB API, you cannot specify the amount you want to read. You can only specify the buffer length of your buffer. If your buffer length is smaller than the LOB data length, Oracle throws an exception.

---

---

### Assignment and Parameters Passing (PL/SQL)

PL/SQL allows *implicit type conversion* and assignment of the following:

- CLOB variables to VARCHAR2, CHAR, or LONG variables and vice-versa.
- BLOB variables to RAW and LONG RAW variables and vice versa.

The same holds for *parameter passing*. Hence PL/SQL allows the passing of the following:

- A CLOB as an actual parameter to a function whose formal parameter is a character type, such as VARCHAR2, CHAR, or LONG, or vice versa
- A BLOB as an actual parameter to a procedure or function whose formal parameter is a binary type, such as RAW or LONG RAW, or vice versa.

#### See Also:

- ["Implicit Conversion of NUMBER, DATE, ROW\\_ID, BINARY\\_INTEGER, and PLS\\_INTEGER to LOB is Not Supported"](#) on page 8-23
- ["No Implicit Conversions of BLOB to VARCHAR2, CHAR, or CLOB to RAW or LONG RAW"](#) on page 8-23.

PL/SQL **built-in functions and operators** which accept VARCHAR2 arguments also accept CLOB arguments now. For example, INSTR, SUBSTR, comparison operators,...

**See Also:** ["VARCHAR2 and CLOB in PL/SQL Built-In Functions"](#) on page 8-19, for a complete list.

## Migrating Existing Tables from LONG to LOB

This section describes techniques for migrating existing tables from LONG to LOB datatypes.

### Migrating LONGs to LOBs: Using ALTER TABLE to Change LONG Column to LOB Types

ALTER TABLE now allows a LONG column to be modified to CLOB or NCLOB and a LONG\_RAW column to be modified to BLOB. The syntax is as follows:

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

```
[DEFAULT <default_value>]) [LOB_storage_clause];
```

For example, if you had a table with the following definition:

```
CREATE TABLE Long_tab (id NUMBER, long_col LONG);
```

you can change the column `long_col` in table `Long_tab` to datatype `CLOB` as follows:

```
ALTER TABLE Long_tab MODIFY ( long_col CLOB );
```

---

---

**Note:** The new ALTER TABLE statement only modifies either of the following:

- A LONG column to a CLOB or an NCLOB column
- A LONG RAW column to a BLOB column

It will not modify a VARCHAR or a RAW column.

---

---

---

---

**Note:** In the new ALTER TABLE statement to change a LONG column to a LOB, the only other operations allowed are:

- Specifying the default value for the LOB column
- Specifying the LOB storage clause for the column being changed from LONG to LOB

Any other ALTER TABLE operation is not allowed with this operation.

---

---

---

---

**Note: Migrating LONGs to LOBs: Method Used in Oracle8i**

This method of migrating LONGs to LOBs replaces the following method used in Oracle8i. Oracle8i added a new operator on LONGs called TO\_LOB(). TO\_LOB() copies the LONG to a LOB. You can use CREATE TABLE AS SELECT or INSERT AS SELECT statements with the TO\_LOB operator to copy data from the LONG to the LOB. For example, if you have a table with the following definition:

```
CREATE TABLE Long_tab (id NUMBER, long_col LONG);
```

Do the following:

```
CREATE TABLE Lob_tab (id NUMBER, clob_col CLOB);
```

```
INSERT INTO Lob_tab SELECT id, TO_LOB(long_col) FROM long_tab;
```

```
DROP TABLE Long_tab;
```

```
CREATE VIEW Long_tab (id, long_col) AS SELECT * from Lob_tab;
```

This series of operations is equivalent to changing the datatype of the column Long\_col of table Long\_tab from LONG to CLOB. With this method (the method of choice prior to this release) you have to create all the constraints, triggers, and indexes on the new table again.

---

---

**All Constraints of LONG Column are Maintained**

All constraints of your previous LONG columns are maintained for the new LOB columns. The only constraint allowed on LONG columns are NULL and NOT-NULL. To alter the constraints for these columns, or alter any other columns or properties of this table, you have to do so in a subsequent ALTER TABLE statement.

**Default Values for LONG are Copied to LOB**

If you do not specify a default value, the default value for the LONG column is copied to the new LOB column.

### Most Triggers Remain Valid

Most of the existing triggers on your table are still usable, however two types of triggers can cause issues.

**See:** ["LONG-to-LOB Migration Limitations"](#) on page 8-10 for more details.

### Indexes Must be Rebuilt — Use ALTER INDEX...REBUILD

Domain indexes on the LONG column must be dropped before ALTERing the LONG column to LOB.

All other indexes, including domain and functional indexes on all columns of the table, will be unusable and must be rebuilt using the `ALTER INDEX <index name> REBUILD` statement.

### Rebuilding Indexes After a LONG to LOB Migration

To rebuild your indexes on a given table, after a LONG to LOB migration, use the following steps:

1. Drop the domain indexes on the LONG column, if any
2. `ALTER TABLE Long_tab MODIFY ( long_col CLOB...)...`;
3. `SELECT index_name FROM user_indexes WHERE table_name='LONG_TAB'`;

---

---

**Note:** The table name has to be capitalized in this query.

---

---

4. For all indexes <index> listed in step 3, issue the command:  
`ALTER INDEX <index> REBUILD`
5. Create the domain index on the LOB column, if desired.

### Space Requirements are Temporarily Doubled

The `ALTER TABLE MODIFY LONG->LOB` statement copies the contents of the table into a new space, and frees the old space at the end of the operation. This temporarily doubles the space requirements. But the advantage is that after the transformation, the table will not have any embedded NULLs, so the performance of subsequent DMLs or queries is good.

## LOGGING

During migration, the redo changes for the table are logged only if the table has LOGGING on. Redo changes for the column being converted from LONG to LOB are logged only if the storage characteristics of the LOB indicate LOGGING. The default value for LOGGING | NOLOGGING for the LOB is inherited from the tablespace in which the LOB is being created.

To prevent generation of redo space during migration, do the following to migrate smoothly:

1. ALTER TABLE Long\_tab NOLOGGING;
2. ALTER TABLE Long\_tab MODIFY ( long\_col CLOB [default <default\_val>]) LOB (long\_col) STORE AS (... NOLOGGING...);
3. ALTER TABLE Long\_tab MODIFY LOB long\_col STORE AS (...LOGGING...);
4. ALTER TABLE Long\_tab LOGGING;
5. Take a backup of the tablespaces containing the table and the LOB.

## LONG-to-LOB Migration Limitations

Before migrating from LONGs to LOBs, note the following issues:

### Clustered Tables

LOBs are not allowed in clustered tables, whereas LONGs are allowed. So if a table is a part of a cluster, its LONG or LONG RAW column cannot be changed to LOB.

### Replication

Oracle does not support the replication of columns that use the LONG and LONG RAW datatypes. Oracle simply omits columns containing these datatypes from replicated tables. You must convert LONG datatypes to LOBs in Oracle8i and then replicate.

This is not a restriction imposed by LONG-to-LOB, but instead, the LONG-to-LOB migration enables the replication of these columns.

If a table is replicated or has materialized views, and its LONG column is changed to LOB, you may have to manually fix the replicas.

### Triggers

Triggers are a problem in the following case:

You cannot have LOB columns in the UPDATE OF list in the UPDATE OF trigger. LONG columns are allowed. For example, you cannot say:

```
create table t(lobcol CLOB);
create trigger trig before/after update of lobcol on t ...;
```

Hence, in the following case the trigger becomes invalidated and cannot be recompiled:

```
create table t(lobcol LONG);
create or replace trigger trig before (or after) update of lobcol on t
for each row
begin
  dbms_output.put_line('lmn');
end;
/
insert into t values('abc');
UPDATE t SET lobcol = 'xyz';

ALTER TABLE t MODIFY (lobcol CLOB); -- invalidates the trigger
UPDATE t SET lobcol = 'xyz'; -- doesn't execute because trigger
                               -- can't be revalidated
```

This restriction may be removed in a future release. All other triggers work without a problem.

## Indexes

Indexes on any column of the table being migrated must be manually rebuilt. This includes functional and domain indexes, must be manually rebuilt.

- **Domain indexes** for the LONG column must be dropped before ALTERing a LONG column to LOB.
- **Functional indexes** on LONG columns converted to LOBs should work without any changes to your code.

## LONGs, LOBs, and NULLs

There is a difference in how NULL and zero-length LONGs and LOBs behave. Applications migrating from LONG-to-LOB are not affected by this behavior, as follows:

Consider these two tables, `long_tab` and `lob_tab`:

```
CREATE TABLE long_tab(id NUMBER, long_col LONG);
```

```
CREATE TABLE lob_tab(id NUMBER, lob_col LOB);
```

### NULL LONGs Versus Zero Length LONGs

Zero length LONGs and NULL LONGs are the same. So the following two statements each produce the same result, each one inserting a NULL in the LONG column:

```
INSERT INTO long_tab values(1, NULL);
INSERT INTO long_tab values(1, ''); -- Zero length string inserts NULL into the
LONG column
```

### NULL LOBs Versus Zero Length LOBs

For LOBs, the following two statements also insert a NULL in the LOB column:

```
INSERT INTO lob_tab values(1, NULL);
INSERT INTO lob_tab values(1, ''); -- A zero length string inserts NULL into
LOB column
```

However, if we truly insert a zero-length LOB using the `empty_clob()` constructor, the LOB column will be non-NULL.

```
INSERT INTO lob_tab values(1, empty_clob()); -- A zero length LOB is not the
same as NULL
```

## Using LONG-to-LOB API with OCI

Prior to this release, OCI provided interface calls for performing piecewise INSERTS, UPDATES, and fetches of LONG data. These APIs also allow you to provide data dynamically in case of array INSERTs or UPDATES, instead of providing a static array of bind values. These piecewise operations can be performed by polling or by providing a callback.

The following functions are now supported for LOBs for you to directly INSERT, UPDATE, and fetch LOB data without your having to deal with the LOB locator:

- `OCIBindByName()` or `OCIBindByPos()`. These functions create an association between a program variable and a placeholder in the SQL statement or a PL/SQL block for INSERT/UPDATE.
- `OCIBindDynamic()`. This call is used to register user callbacks for dynamic data allocation for INSERT/UPDATE.
- `OCIDefineByPos()`. This call associates an item in a SELECT-list with the type and output data buffer.



- `OCIDefineDynamic()`. This call registers user callbacks for SELECTs if the `OCI_DYNAMIC_FETCH` mode was selected in `OCIDefineByPos()`.
- `OCIStmtGetPieceInfo()` and `OCIStmtSetPieceInfo()`. These calls are used to get or set piece information for piecewise operations.

**See Also:** “Runtime data allocation and piecewise operations” in the *Oracle Call Interface Programmer’s Guide*, for details on the API.

## Guidelines for Using LONG-to-LOB API for LOBs with OCI

The aforementioned OCI functions work in this release for LOBs in exactly the same way as they do for LONGs. Using these, you can perform INSERTs, UPDATEs, and fetches of data as described here.

---



---

**Note:** When you use the aforementioned functions for CLOBs, BLOBs, LONGs, and LONG RAWs, specify the datatype (dty) as:

- `SQLT_LNG` and `SQLT_CHR` for CLOBs and LONGs
  - `SQLT_LBI` and `SQLT_BIN` for BLOBs and LONG RAWs
- 
- 

## Using OCI Functions to Perform INSERT or UPDATE on LOBs

There are various ways to perform INSERT or UPDATE of LOB data.

---



---

**Note:** These are in addition to the ways to insert LOB locators, which are documented in [Chapter 10, "Internal Persistent LOBs"](#).

---



---

In all the ways described in the following, it is assumed that you have initialized the OCI environment and allocated all necessary handles.

### Simple INSERTs or UPDATEs in One Piece

To perform simple INSERTs and UPDATEs in one piece, the steps are:

1. `OCIStmtPrepare()` to prepare the statement in `OCI_DEFAULT` mode.
2. `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in `OCI_DEFAULT` mode to bind a LOB as `CHAR` or `BIN`.
3. `OCIStmtExecute()` to do the actual INSERT/UPDATE.

### Using Piecewise INSERTs and UPDATEs with Polling

To perform piecewise INSERTs and UPDATEs with polling, the steps are:

1. `OCIStmtPrepare()` to prepare the statement in `OCI_DEFAULT` mode.
2. `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in `OCI_DATA_AT_EXEC` mode to bind a LOB as CHAR or BIN.
3. `OCIStmtExecute()` in default mode. This should return `OCI_NEED_DATA`.
4. While (returned value is `OCI_NEED_DATA`), do the following:
  - \* `OCIStmtGetPieceInfo()` to retrieve information about piece to be inserted
  - \* `OCIStmtSetPieceInfo()` to set information about piece to be inserted
  - \* `OCIStmtExecute`. You are done when the return value is `OCI_SUCCESS`.

### Piecewise INSERTs and UPDATEs with Callback

To perform piecewise INSERTs and UPDATEs with callback, the steps are:

1. `OCIStmtPrepare()` to prepare the statement in `OCI_DEFAULT` mode.
2. `OCIBindByName()` or `OCIBindbyPos()` to bind a placeholder in `OCI_DATA_AT_EXEC` mode to bind a LOB as CHAR or BIN.
3. `OCIBindDynamic()` to specify the callback.
4. `OCIStmtExecute()` in default mode.

### Array INSERTs and UPDATEs

Use any of the preceding modes in conjunction with `OCIBindArrayOfStruct()`, or by specifying the number of iterations (*iter*) value > 1 in the `OCIStmtExecute()` call.

## Using OCI Functions to Perform FETCH on LOBs

There are three ways to fetch the LOB data.

---

---

**Note:** These are in addition to the ways to fetch the LOB locator, which are documented in [Chapter 10, "Internal Persistent LOBs"](#).

---

---

- Simple fetch in one piece
- Piecewise fetch
- Array fetch

### Simple Fetch in One Piece

To perform a simple fetch on LOBs in one piece, the steps involved are:

1. `OCIStmtPrepare()` to prepare the SELECT statement in `OCI_DEFAULT` mode.
2. `OCIDefineByPos()` to define a select list position in `OCI_DEFAULT` mode to define a LOB as `CHAR` or `BIN`.
3. `OCIStmtExecute()` to execute the SELECT statement.
4. `OCIStmtFetch()` to do the actual fetch.

### Piecewise Fetch with Polling

To perform a piecewise fetch on LOBs with polling, the steps are:

1. `OCIStmtPrepare()` to prepare the SELECT statement in `OCI_DEFAULT` mode.
2. `OCIDefinebyPos()` to define a select list position in `OCI_DYNAMIC_FETCH` mode to define a LOB as `CHAR` or `BIN`.
3. `OCIStmtExecute()` to execute the SELECT statement.
4. `OCIStmtFetch()` in default mode. This should return `OCI_NEED_DATA`.
5. While (returned value is `OCI_NEED_DATA`), do the following:
  - \* `OCIStmtGetPieceInfo()` to retrieve information about piece to be fetched.
  - \* `OCIStmtSetPieceInfo()` to set information about piece to be fetched.
  - \* `OCIStmtFetch`. You are done when the return value is `OCI_SUCCESS`.

### Piecewise with Callback

To perform a piecewise fetch on LOBs with callback, the steps are:

1. `OCIStmtPrepare()` to prepare the statement in `OCI_DEFAULT` mode.

2. `OCIDefinebyPos()` to define a select list position in `OCI_DYNAMIC_FETCH` mode to define a LOB as `CHAR` or `BIN`.
3. `OCIStmtExecute()` to execute the `SELECT` statement.
4. `OCIDefineDynamic()` to specify the callback.
5. `OCIStmtFetch()` in default mode.

### Array Fetch

Use any of the preceding modes in conjunction with `OCIDefineArrayOfStruct()`, or by specifying the number of iterations (*iter*) value  $>1$  in the `OCIStmtExecute()` call.

## Using SQL and PL/SQL to Access LONGs and LOBs

This section describes the following topics:

- [Using SQL and PL/SQL to Access LOBs](#) on page 8-16
- [Implicit Assignment and Parameter Passing](#) on page 8-17
- [PL/SQL and C Binds from OCI](#) on page 8-20
- [Calling PL/SQL and C Procedures from SQL or PL/SQL](#) on page 8-21

## Using SQL and PL/SQL to Access LOBs

Data from `CLOB` and `BLOB` columns can be referenced by regular SQL statements, such as: `INSERT`, `UPDATE` and `SELECT`.

There is no piecewise `INSERT/UPDATE/fetch` routine in `PL/SQL`. Therefore the amount of data that can be accessed from the `LOB` column is limited by the maximum character buffer size. In `Oracle9i`, `PL/SQL` supports character buffer sizes up to 32767 bytes. Hence only `LOBs` of sizes up to 32767 bytes can be accessed by `PL/SQL` applications.

If you need to access more than 32k, `OCI` callouts have to be made from the `PL/SQL` code to utilize the APIs for piecewise insert and fetch.

The following are guidelines for accessing `LOB` columns:

## **INSERTs**

Data can be inserted into tables containing LOB columns by regular INSERTs in the VALUES clause. The field of the LOB column can be PL/SQL character or binary buffer variables ( CHAR, VARCHAR2, RAW,...), a string literal, or a LOB locator.

## **UPDATEs**

LOB columns can be updated as a whole by UPDATE... SET statements. There is no random access of data stored in LOB columns. In the SET clause, the new values can also be literals or any PL/SQL character or binary variables, or a LOB locator.

**Restriction for LONG RAW and RAW Buffers More Than 4000 Bytes.** There is a restriction for binds which exists for both LONGs and LOBs. You cannot bind a VARCHAR2 buffer to a LONG RAW or a BLOB column if the buffer is of size more than 4000 bytes, because SQL will not do implicit HEXTORAW conversions for buffers larger than 4000 bytes. Similarly, you cannot bind a RAW buffer to a LONG or a CLOB column if the buffer is of size more than 4000 bytes because SQL will not do implicit RAWTOHEX conversions for buffers larger than 4000 bytes.

## **SELECTs**

For fetch, in prior releases, you could not use SELECT INTO to bind a character variable to a LOB column. SELECT INTO used to bind LOB locators to the column. This constraint has been removed.

LOB columns can be selected into character or binary buffers in PL/SQL. If the LOB column is longer than the buffer size, an exception is raised without filling the buffer with any data. LOB columns can also be selected into LOB locators.

## **Implicit Assignment and Parameter Passing**

The LONG-to-LOB migration API supports assigning a CLOB (BLOB) variable to a LONG(LONG RAW) or a VARCHAR2(RAW) variable and vice-versa. This is because of the existence of %type and %rowtype datatypes in PL/SQL. The assignments include parameter passing. These features are explained in detail in the following section.

### **Variable Assignment Between CLOB/CHAR and BLOB/RAW**

The following variable assignment between CLOB and CHAR, and BLOB and RAWs are allowed:

```
CLOB_VAR := CHAR_VAR;
```

```
CHAR_VAR := CLOB_VAR;
BLOB_VAR := RAW_VAR;
RAW_VAR := BLOB_VAR;
```

This is done because of the presence of %type and %rowtype in existing code. For example:

```
CREATE TABLE t (long_col LONG); -- Alter this table to change LONG column to LOB
DECLARE
  a VARCHAR2(100);
  b t.long_col%type; -- This variable changes from LONG to CLOB
BEGIN
  SELECT * INTO b FROM t;
  a := b; -- This changes from "VARCHAR2 := LONG to VARCHAR2 := CLOB
  b := a; -- This changes from "LONG := VARCHAR2 to CLOB := VARCHAR2
END;
```

### Function/Procedure Parameter Passing

This allows all the user-defined procedures and functions to use CLOBs and BLOBs as actual parameters where VARCHAR2, LONG, RAW, and LONG RAW are formal parameters and vice-versa. It also allows PL/SQL built-ins like INSTR to accept CLOB data in addition to strings. For example:

```
CREATE PROCEDURE FOO ( a IN OUT t.long_col%type) IS.....
CREATE PROCEDURE BAR (b IN OUT VARCHAR2) IS ...
DECLARE
  a VARCHAR2(100);
  b t.long_col%type -- This changes to CLOB
BEGIN
  a := 'abc';
  SELECT long_col into b from t;
  FOO(a); -- Actual parameter is VARCHAR2, formal parameter is CLOB
  BAR(b); -- Actual parameter is CLOB, formal parameter is VARCHAR2
END;
```

### Explicit Conversion Functions

In PL/SQL, the following two new explicit conversion functions have been added to convert other data types to CLOB and BLOB as part of LONG-to-LOB migration:

- TO\_CLOB() converts LONG, VARCHAR2, and CHAR to CLOB
- TO\_BLOB() converts LONG RAW and RAW to BLOB

TO\_CHAR() is enabled to convert a CLOB to a CHAR type.

## VARCHAR2 and CLOB in PL/SQL Built-In Functions

PL/SQL VARCHAR2 functions and operators take CLOBs as arguments or operands. A CLOB can be passed to SQL and PL/SQL VARCHAR2 built-in functions, behaving exactly like a VARCHAR2. Or the VARCHAR2 variable can be passed into DBMS\_LOB APIs acting like a LOB locator.

The PL/SQL built-in functions which accept CLOB parameters or give CLOB output are:

- LENGTH, LENGTHB
- INSTR, INSTRB
- SUBSTR, SUBSTRB
- CONCAT, ||
- LPAD, RPAD
- LTRIM, RTRIM, TRIM
- LIKE
- REPLACE
- LOWER, UPPER
- NLS\_LOWER, NLS\_UPPER
- NVL
- Comparison operators (>, =, <, !=)

If a function returns a CLOB and the result is assigned to a VARCHAR2 variable, but the size of the VARCHAR2 variable is not large enough to contain the result, an error is raised and no operation is performed. The same holds if you try to SELECT a CLOB into a VARCHAR2 variable. This is consistent with the current VARCHAR2 behavior.

These functions implicitly create temporary LOBs. Hence, some LOB locators can change from *persistent* to *temporary*. As a result, any changes to the data pointed to by the (temporary) LOB locator are not reflected in the persistent LOB which it initially pointed to.

These temporary LOBs are freed automatically at the end of the PL/SQL block. You can choose to free them explicitly to reclaim system resources and temporary tablespace by calling `DBMS_LOB.FREE_TEMPORARY( )` on the CLOB variable.

**See Also:** [Chapter 7, "Modeling and Design", "SQL Semantics Support for LOBs"](#) on page 7-33.

## PL/SQL and C Binds from OCI

When you call a PL/SQL procedure from OCI, and have an in or out or in/out bind, you should be able to:

- Bind a variable as `SQLT_CHR` or `SQLT_LNG` where the formal parameter of the PL/SQL procedure is `SQLT_CLOB`, or
- Bind a variable as `SQLT_BIN` or `SQLT_LBI` where the formal parameter is `SQLT_BLOB`

The following two cases work:

### Calling PL/SQL Outbinds in the "begin foo(:1); end;" Manner.

Here is an example of calling PL/SQL outbinds in the "begin foo(:1);end;" manner:

```
text *sqlstmt = (text *)"BEGIN get_lob(:c); END; " ;
```

### Calling PL/SQL Outbinds in the "call foo(:1);" Manner.

Here is an example of calling PL/SQL outbinds in the "call foo(:1);" manner:

```
text *sqlstmt = (text *)"CALL get_lob( :c );" ;
```

In both these cases, the rest of the program is as follows:

```
OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
              (ub4) OCI_NTIV_SYNTAX, (ub4) OCI_DEFAULT);
curlen = 0;
OCIBindByName(stmthp, &bndhp[3], errhp,
              (text *) ":c", (sb4) strlen((char *) ":c"),
              (dvoid *) buf5, (sb4) LONGLEN, SQLT_CHR,
              (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
              (ub4) 1, (ub4 *) &curlen, (ub4) OCI_DATA_AT_EXEC);
```

The PL/SQL procedure, `get_lob()`, is as follows:

```
procedure get_lob(c INOUT CLOB) is -- This might have been column%type
begin
... /* The procedure body could be in PL/SQL or C*/
end;
```



## Calling PL/SQL and C Procedures from SQL or PL/SQL

### From SQL

When a PL/SQL procedure is called from SQL, LONG parameters are not allowed. So this case is not a part of the LONG-to-LOB conversion process.

### From PL/SQL

You can call a PL/SQL or C procedure from PL/SQL. It is possible to pass a CLOB as an actual parameter where CHR is the formal parameter, or vice versa. The same holds for BLOBs and RAWs.

These cases arise when either the formal or the actual parameter is an anchored type, that is, table%type.

## Applications Requiring Changes When Converting From LONGs to LOBs

Even with implicit conversions to LOBs, some changes will have to be made to your application. Cases where you will have to make changes to your application, are listed in the following paragraphs.

### Overloading with Anchored Types

For applications using anchored types, some overloads would silently resolve to different targets during the conversion to LOBs. For example:

```
procedure p(l long) is ...;      -- (1)
procedure p(c clob) is ...;     -- (2)
```

Consider the caller:

```
declare
    var longtab.longcol%type;
begin
    ...
    p(var);
    ...
end;
```

Prior to LOB migration this call would have resolved to overload (1). Once longtab is migrated to LOBs this call will silently resolve to overload (2). A similar issue arises if the parameter to (1) was CHAR, VARCHAR2, RAW, LONG RAW.

When migrating LONG columns to LOB you have to manually examine and fix dependent applications.

Because of the new conversions, some existing applications with procedure overloads, that include LOB arguments, may still break. This includes applications that DO NOT use LONG anchored types. For example,

```
procedure p(n number) is ...;   -- (1)
procedure p(c clob) is ...;     -- (2)

p('abc');
```

Previously, the only conversion allowed was CHAR to NUMBER, so (1) would be chosen. Now, both conversions are allowed, so the call is ambiguous and raises an overloading error.

## Implicit Conversion of NUMBER, DATE, ROW\_ID, BINARY\_INTEGER, and PLS\_INTEGER to LOB is Not Supported

PL/SQL currently permits conversion of NUMBER, DATE, ROW\_ID, BINARY\_INTEGER, and PLS\_INTEGER to LONG. There are no plans to support implicit conversions from these types to LOB (explicit or implicit). Users relying on these conversions will have to explicitly convert these types TO\_CHAR. Hence, if you had an assignment of the form:

```
number_var := long_var; -- The RHS becomes a LOB variable after conversion
```

Then you have to explicitly modify your code to say:

```
number_var := TO_CHAR(long_var); -- Note that long_var is of type CLOB after conversion
```

## No Implicit Conversions of BLOB to VARCHAR2, CHAR, or CLOB to RAW or LONG RAW

Also, there is no implicit conversion from the following:

- BLOB to VARCHAR2, CHAR, or LONG
- CLOB to RAW or LONG RAW

Hence if you had the following code:

```
SELECT <long raw column> INTO <varchar2> VARIABLE FROM <table>
```

and you changed the LONG RAW column into BLOB, this SELECT statement will not work. You have to add the TO\_RAW or a TO\_CHAR conversion operator on the selected variable such as:

```
SELECT TO_RAW(<long raw column>) INTO <varchar2> VARIABLE FROM <table>
-- note that the long raw column is now a BLOB column
```

The same holds for selecting a CLOB into a RAW variable, or for assignments of CLOB to RAW and BLOB to VARCHAR2.

## Using utldtree.sql to Determine Where Your Application Needs Change

Use the utility, rdbms/admin/utldtree.sql, to determine which parts of your application potentially need rewriting when you ALTER your LONG tables to LOBs.

**utldtree.sql** allows you to see all objects that are (recursively) dependent on a given object. In addition, you will only see objects for which you have permission.

Instructions on how to use **utldtree.sql** is documented in the file itself. Hence you can see all objects which depend on the table with the LONG column, and compare that with the cases documented in the section titled "[Applications Requiring Changes When Converting From LONGs to LOBs](#)" on page 8-22, to see if your application needs changing.

**utldtree.sql** is only needed for PL/SQL. For SQL and OCI you do not need to change your applications.

## Examples of Converting from LONG to LOB Using Table Multimedia\_tab

See [Appendix B, "The Multimedia Schema"](#), for a detailed description of the Multimedia\_tab schema. The fields used in the following examples are:

```
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;
```

Suppose the column, `STORY`, of table `MULTIMEDIA_TAB` was of type `LONG` before, that is, you originally created the table `MULTIMEDIA_TAB` as follows:

```
CREATE TABLE MULTIMEDIA_TAB (CLIP_ID NUMBER,  
  STORY LONG,  
  .... );
```

### To Convert LONG to CLOB, Use ALTER TABLE

To convert the LONG column to CLOB just use `ALTER TABLE` as follows:

```
ALTER TABLE multimedia_tab MODIFY ( story CLOB );
```

and you are done!

Any existing application using table `MULTIMEDIA_TAB` can continue to work with minor modification even after the column `STORY` has been modified to type `CLOB`.

Here are examples of all operations (binds and defines) used by LONGs and that will continue to work for LOBs with minor modifications as described in ["Applications Requiring Changes When Converting From LONGs to LOBs"](#) on page 8-22.

## Converting LONG to LOB Example 1: More than 4K Binds and Simple INSERTS

The following example illustrates converting from LONG to LOBs when using a >4K bind and simple INSERT:

```
word buflen, buf1 = 0;
text buf2[5000];
text *insstmt = (text *)
"INSERT INTO MULTIMEDIA_TAB(CLIP_ID, STORY) VALUES
(:CLIP_ID, :STORY)";

if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
(text *) ":CLIP_ID", (sb4) strlen((char *) ":CLIP_ID"),
(dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
|| OCIBindByName(stmthp, &bndhp[1], errhp,
(text *) ":STORY", (sb4) strlen((char *) ":STORY"),
(dvoid *) buf2, (sb4) sizeof(buf2), SQLT_CHR,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
    return;
}
```

```

buf1 = 101;
memset((void *)buf2, (int)'A', (size_t)5000);

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                  (const OCISnapshot*) 0, (OCISnapshot*) 0,
                  (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtExecute()\n");
    report_error(errhp);
    return;
}

```

## Converting LONG to LOB Example 2: Piecewise INSERT with Polling

Continuing the preceding example...

```

text *sqlstmt = (text *)"INSERT INTO MULTIMEDIA_TAB VALUES (:1, :2)";
ub2 rcode;
ub1 piece, i;

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

OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
              (dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
              (dvoid *) 0, (ub2 *)0, (ub2 *)0,
              (ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT);

OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,
              (dvoid *) 0, (sb4) 15000, SQLT_LNG,
              (dvoid *) 0, (ub2 *)0, (ub2 *)0,
              (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC);

buf1 = 101;
i = 0;
while (1)
{
    i++;
    retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                           (CONST OCISnapshot*) 0, (OCISnapshot*) 0,
                           (ub4) OCI_DEFAULT);

    switch(retval)
    {

```

```

        case OCI_NEED_DATA:

memset((void *)buf2, (int)'A'+i, (size_t)5000);
buflen = 5000;
if (i == 1) piece = OCI_ONE_PIECE
    else if (i == 3) piece = OCI_LAST_PIECE
    else piece = OCI_NEXT_PIECE;

if (OCIStmtSetPieceInfo((dvoid *)bndhp[1],
    (ub4)OCI_HTYPE_BIND, errhp, (dvoid *)buf2,
    &buflen, piece, (dvoid *) 0, &rcode))
{
    DISCARD printf("ERROR: OCIStmtSetPieceInfo: %d \n", retval);
    break;
}

    retval = OCI_NEED_DATA;
    break;
case OCI_SUCCESS:
    break;
default:
    DISCARD printf( "oci exec returned %d \n", retval);
    report_error(errhp);
    retval = 0;
} /* end switch */
if (!retval) break;
} /* end while(1) */

```

## Converting LONG to LOB Example 3: Piecewise INSERT with Callback

The following example illustrates converting from LONG to LOBs when using a piecewise INSERT with callback:

```

void insert_data()
{
text *sqlstmt = (text *) "INSERT INTO MULTIMEDIA_TAB VALUES (:1, :2)";
OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT)
/* bind input */
if (OCIBindByPos(stmthp, &bndhp[0], errhp, (ub4) 1,
    (dvoid *) 0, (sb4) sizeof(buf1), SOLT_INT,
    (dvoid *) 0, (ub2 *)0, (ub2 *)0,
    (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC)
|| OCIBindByPos(stmthp, &bndhp[1], errhp, (ub4) 2,

```

```

        (dvoid *) 0, (sb4) 3 * sizeof(buf2), SQUIT_CHR,
        (dvoid *) 0, (ub2 *)0, (ub2 *)0,
        (ub4) 0, (ub4 *) 0, (ub4) OCI_DATA_AT_EXEC))
    {
        DISCARD printf("FAILED: OCIBindByPos()\n");
        report_error(errhp);
        return OCI_ERROR;
    }
for (i = 0; i < MAXCOLS; i++)
    pos[i] = i+1;
if (OCIBindDynamic(bndhp[0], errhp, (dvoid *) (dvoid *) &pos[0],
    cbf_in_data, (dvoid *) 0, (OCICallbackOutBind) 0)
    || OCIBindDynamic(bndhp[1], errhp, (dvoid *) (dvoid *) &pos[1],
    cbf_in_data, (dvoid *) 0, (OCICallbackOutBind) 0))
    {
        DISCARD printf("FAILED: OCIBindDynamic()\n");
        report_error(errhp);
        return OCI_ERROR;
    }
OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
    (const OCISnapshot*) 0, (OCISnapshot*) 0,
    (ub4) OCI_DEFAULT)
} /* end insert_data() */

/* Inbind callback to specify input data. */
STATICF sb4 cbf_in_data(ctxp, bindp, iter, index, bufpp, alenpp,
    piecep, indpp)

dvoid *ctxp;
OCIBind *bindp;
ub4 iter;
ub4 index;
dvoid **bufpp;
ub4 *alenpp;
ub1 *piecep;
dvoid **indpp;
{
    static int a = 0;
    word j;
    ub4 inpos = *((ub4 *)ctxp);
    switch(inpos)
    {
        case 1:
            buf1 = 175;
            *bufpp = (dvoid *) &buf1;
            *alenpp = sizeof(buf1);

```



```

break;
case 2:
memset((void *)buf2, (int) 'A'+a, (size_t) 5000);
*bufpp = (dvoid *) buf2;
*alenpp = 5000 ;
a++;
break;
default: printf("ERROR: invalid position number: %d\n", pos);
}
*indpp = (dvoid *) 0;
*piecep = OCI_ONE_PIECE;
if (inpos == 2)
{
if (a<=1)
{
*piecep = OCI_FIRST_PIECE;
printf("Insert callback: 1st piece\n");
}
else if (a<3)
{
*piecep = OCI_NEXT_PIECE;
printf("Insert callback: %d'th piece\n", a);
}
else {
*piecep = OCI_LAST_PIECE;
printf("Insert callback: %d'th piece\n", a);
a = 0;
}
}
return OCI_CONTINUE;
}

```

## Converting LONG to LOB Example 4: Array insert

The following example illustrates converting from LONG to LOBs when using an array INSERT:

```

word buflen;
word arrbuf1[5];
text arrbuf2[5][5000];
text *insstmt = (text *)
"INSERT INTO MULTIMEDIA_TAB(CLIP_ID, STORY) VALUES
(:CLIP_ID, :STORY)";

```

```
if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
(text *) ":CLIP_ID", (sb4) strlen((char *) ":CLIP_ID"),
(dvoid *) &arrbuf1[0], (sb4) sizeof(arrbuf1[0]), SQLT_INT,(dvoid *) 0, (ub2 *)
0, (ub2 *) 0,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
|| OCIBindByName(stmthp, &bndhp[1], errhp,
(text *) ":STORY", (sb4) strlen((char *) ":STORY"),
(dvoid *) arrbuf2[0], (sb4) sizeof(arrbuf2[0]), SQLT_CHR,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
    return;
}
OCIBindArrayOfStruct(bndhp[0], ERRH, sizeof(arrbuf1[0]),
    indsk, rlsk, rcsk);
OCIBindArrayOfStruct(bndhp[1], ERRH, sizeof(arrbuf2[0]),
    indsk, rlsk, rcsk);
for (i=0; i<5; i++)
{
    arrbuf1[i] = 101+i;
    memset((void *)arrbuf2[i], (int)'A'+i, (size_t)5000);
}

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 5, (ub4) 0,
(const OCISnapshot*) 0, (OCISnapshot*) 0,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtExecute()\n");
    report_error(errhp);
    return;
}
```

## Converting LONG to LOB Example 5: Simple Fetch

The following example illustrates converting from LONG to LOBs when using a simple fetch:

```
word   i, buf1 = 0;
text   buf2[5000];

text *selstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB
                        ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char *)selstmt),
              (ub4) OCI_NIV_SYNTAX, (ub4) OCI_DEFAULT);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                      (const OCISnapshot*) 0, (OCISnapshot*) 0,
                      (ub4) OCI_DEFAULT);

while (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
{
    OCIDefineByPos(stmthp, &defhp1, errhp, (ub4) 1, (dvoid *) &buf1,
                  (sb4) sizeof(buf1), (ub2) SQLT_INT, (dvoid *) 0,
                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defhp2, errhp, (ub4) 2, (dvoid *) buf2,
                  (sb4) sizeof(buf2), (ub2) SQLT_CHR, (dvoid *) 0,
                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    retval = OCIStmtFetch(stmthp, errhp, (ub4) 1,
                        (ub4) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    if (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
        DISCARD printf("buf1 = %d, buf2 = %.*s\n", buf1, 30, buf2);
}
}
```

## Converting LONG to LOB Example 6: Piecewise Fetch with Polling

The following example illustrates converting from LONG to LOBs when using a piecewise fetch with polling:

```
text *selstmt = (text *) "SELECT STORY FROM MULTIMEDIA_TAB
                        ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, sqlstmt,
              (ub4) strlen((char *)sqlstmt),
              (ub4) OCI_NIV_SYNTAX, (ub4) OCI_DEFAULT);
OCIDefineByPos(stmthp, &dfnhp[1], errhp, (ub4) 1,
              (dvoid *) NULL, (sb4) 100000, SQLT_LNG,
              (dvoid *) 0, (ub2 *) 0,
              (ub2 *) 0, (ub4) OCI_DYNAMIC_FETCH);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
```

```
                (CONST OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                (ub4) OCI_DEFAULT);
retval = OCISstmtFetch(stmthp, errhp, (ub4) 1 ,
                (ub2) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);

while (retval != OCI_NO_DATA && retval != OCI_SUCCESS)
{
    ub1    piece;
    ub4    iter, buflen;
    ub4    idx;
    genclr((void *)buf2, 5000);

    switch(retval)
    {
        case OCI_NEED_DATA:
            OCISstmtGetPieceInfo(stmthp, errhp, &hdlptr, &hdltype,
                &in_out, &iter, &idx, &piece);
            OCISstmtSetPieceInfo(hdlptr, hdltype, errhp,
                (dvoid *) buf2, &buflen, piece,
                (CONST dvoid *) &indp1, (ub2 *) 0));

            retval = OCI_NEED_DATA;
            break;
        default:
            DISCARD printf("ERROR: piece-wise fetching\n");
            return;
    } /* end switch */

    retval = OCISstmtFetch(stmthp, errhp, (ub4) 1 ,
                (ub2) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    printf("Data : %s\n"; buf2);
} /* end while */
```

## Converting LONG to LOB Example 7: Piecewise Fetch with Callback

The following example illustrates converting from LONG to LOBs when using a piecewise fetch with callback:

```
select()
{
text *sqlstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB";

OCISstmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
                (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
```

```

OCIDefineByPos(stmthp, &dfnhp[0], errhp, (ub4) 1,
               (dvoid *) 0, (sb4) sizeof(buf1), SOLT_INT,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) OCI_DYNAMIC_FETCH);
OCIDefineByPos(stmthp, &dfnhp[1], errhp, (ub4) 2,
               (dvoid *) 0, (sb4)3 * sizeof(buf2), SOLT_CHR,
               (dvoid *) 0, (ub2 *)0, (ub2 *)0,
               (ub4) OCI_DYNAMIC_FETCH);
OCIDefineDynamic(dfnhp[0], errhp, (dvoid *) &outpos,
                 (OCICallbackDefine) cbf_get_data);
OCIDefineDynamic(dfnhp[1], errhp, (dvoid *) &outpos2,
                 (OCICallbackDefine) cbf_get_data);
OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
               (const OCISnapshot*) 0, (OCISnapshot*) 0,
               (ub4) OCI_DEFAULT);
buf2[ 4999 ] = '\0';
printf("Select callback: Last piece: %s\n", buf2);
}

/* ----- */
/* Fetch callback to specify buffers. */
/* ----- */
STATICF sb4 cbf_get_data(ctxp, dfnhp, iter, bufpp, alenpp, piecep,
                        indpp, rcpp)

dvoid *ctxp;
OCIDefine *dfnhp;
ub4 iter;
dvoid **bufpp;
ub4 **alenpp;
ub1 *piecep;
dvoid **indpp;
ub2 **rcpp;
{
    static int a = 0;
    ub4 outpos = *((ub4 *)ctxp);
    len = sizeof(buf1);
    len2 = 5000;

    switch(outpos)
    {
        case 1:
            *bufpp = (dvoid *) &buf1;
            *alenpp = &len;
            break;
    }
}

```

```
        case 2:
        a ++;
        *bufpp = (dvoid *) buf2;
        *alenpp = &len2;
        break;
        default:
        *bufpp = (dvoid *) 0;
        *alenpp = (ub4 *) 0;
        DISCARD printf("ERROR: invalid position number: %d\n", pos);
    }

    *indpp = (dvoid *) 0;
    *rcpp = (ub2 *) 0;

    if (outpos == 1)
        *piecep = (ub1)OCI_ONE_PIECE;
    if (outpos == 2)
    {
        out2[len2] = '\0';
        if (a<=1)
        {
            *piecep = OCI_FIRST_PIECE;
            printf("Select callback: 0th piece\n");
        }
        else if (a<3)
        {
            *piecep = OCI_NEXT_PIECE;
            printf("Select callback: %d'th piece: %s\n", a-1, out2);
        }
        else {
            *piecep = OCI_LAST_PIECE;
            printf("Select callback: %d'th piece: %s\n", a-1, out2);
            a = 0;
        }
    }

    return OCI_CONTINUE;
}
```

## Converting LONG to LOB Example 8: Array Fetch

The following example illustrates converting from LONG to LOBs when using an array fetch:

```

word   i;
word arrbuf1[5] = 0;
text   arrbuf2[5][5000];

text *selstmt = (text *) "SELECT CLIP_ID, STORY FROM MULTIMEDIA_TAB
                        ORDER BY CLIP_ID";
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char *)selstmt),
              (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
retval = OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0,
                      (const OCISnapshot*) 0, (OCISnapshot*) 0,
                      (ub4) OCI_DEFAULT);

while (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
{
    OCIDefineByPos(stmthp, &defhp1, errhp, (ub4) 1,
                  (dvoid *) &arrbuf1[0], (sb4) sizeof(arrbuf1[0]),
                  (ub2) SQLT_INT, (dvoid *) 0,
                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defhp2, errhp, (ub4) 2,
                  (dvoid *) arrbuf2[0], (sb4) sizeof(arrbuf2[0]),
                  (ub2) SQLT_CHR, (dvoid *) 0,
                  (ub2 *) 0, (ub2 *) 0, (ub4) OCI_DEFAULT);
    OCIDefineArrayOfStruct(dfnhp[0], ERRH, sizeof(arrbuf1[0]), indsk,
                          rlsk, rcsk);
    OCIDefineArrayOfStruct(dfnhp[1], ERRH, sizeof(arrbuf2[0]), indsk,
                          rlsk, rcsk);

    retval = OCIStmtFetch(stmthp, errhp, (ub4) 5,
                        (ub4) OCI_FETCH_NEXT, (ub4) OCI_DEFAULT);
    if (retval == OCI_SUCCESS || retval == OCI_SUCCESS_WITH_INFO)
    {
        DISCARD printf("%d, %s\n", arrbuf1[0], arrbuf2[0]);
        DISCARD printf("%d, %s\n", arrbuf1[1], arrbuf2[1]);
        DISCARD printf("%d, %s\n", arrbuf1[2], arrbuf2[2]);
        DISCARD printf("%d, %s\n", arrbuf1[3], arrbuf2[3]);
        DISCARD printf("%d, %s\n", arrbuf1[4], arrbuf2[4]);
    }
}

```

## Converting LONG to LOB Example 9: Using PL/SQL in INSERT, UPDATE and SELECT

INSERT/UPDATE statements on LOBs are used in the same way as on LONGs. For example:

```
BEGIN
  INSERT INTO Multimedia_tab VALUES (1, 'A wonderful story', NULL, EMPTY_BLOB,
    EMPTY_BLOB(), NULL, NULL, NULL, NULL, NULL);
  UPDATE Multimedia_tab SET Story = 'A wonderful story';
END;
```

LONG-to-LOB API enables SELECT statements to bind character variables to LOB columns.

```
BEGIN
  story_buffer VARCHAR2(100);
  /* This will get the LOB column if it is upto 100 bytes, otherwise it will
  raise an exception */
  SELECT Story INTO story_buffer FROM Multimedia_tab WHERE Clip_ID = 1;
END;
```

## Converting LONG to LOB Example 10: Assignments and Parameter Passing in PL/SQL

The LONG-to-LOB API enables implicit assignments of LOBs to VARCHAR2s, RAWs,...., including parameter passing. For example:

```
CREATE TABLE t (clob_col CLOB, blob_col BLOB);
INSERT INTO t VALUES('abcdefg', 'aaaaaa');
DECLARE
  var_buf VARCHAR2(100);
  clob_buf CLOB;
  raw_buf RAW(100);
  blob_buf BLOB;
BEGIN
  SELECT * INTO clob_buf, blob_buf FROM t;
  var_buf := clob_buf;
  clob_buf := var_buf;
  raw_buf := blob_buf;
  blob_buf := raw_buf;
END;

CREATE PROCEDURE FOO ( a IN OUT CLOB) IS.....
```



```

CREATE PROCEDURE BAR (b IN OUT VARCHAR2) IS .....
DECLARE
    a VARCHAR2(100) := '1234567';
    b CLOB;
BEGIN
    FOO(a);
    SELECT clob_col INTO b FROM t;
    BAR(b);
END;

```

## Converting LONG to LOB Example 11: CLOBs in PL/SQL Built-In Functions

This example illustrates the use of CLOBs in PL/SQL built-in functions, when converting LONGs to LOBs:

```

DECLARE
    myStory CLOB;
    revisedStory CLOB;
    myGist VARCHAR2(100) := 'This is my gist.';
    revisedGist VARCHAR2(100);
BEGIN
    -- select a CLOB column into a CLOB variable
    SELECT Story INTO myStory FROM Multimedia_tab WHERE clip_id=10;

    -- perform VARCHAR2 operations on a CLOB variable
    revisedStory := UPPER(SUBSTR(myStory, 100, 1));

    -- revisedStory is a temporary LOB
    -- Concat a VARCHAR2 at the end of a CLOB
    revisedStory := revisedStory || myGist;
    -- The following statement will raise an error since myStory is
    -- longer than 100 bytes
    myGist := myStory;
END;

```

## Converting LONG to LOB Example 12: Using PL/SQL Binds from OCI on LOBs

The LONG-to-LOB API allows LOB PL/SQL binds from OCI to work as follows:

When you call a PL/SQL procedure from OCI, and have an in or out or inout bind, you should be able to bind a variable as `SQLT_CHR`, where the formal parameter of the PL/SQL procedure is `SQLT_CLOB`.

---

---

**Note:** C procedures are wrapped inside a PL/SQL stub, so the OCI application always invokes the PL/SQL stub.

---

---

For the OCI calling program, the following are likely cases:

### Calling PL/SQL Outbinds in the "begin foo(:1); end;" Manner

For example:

```
text *sqlstmt = (text *)"BEGIN PKG1.P5 (:c); END; " ;
```

### Calling PL/SQL Outbinds in the "call foo(:1);" Manner

For example:

```
text *sqlstmt = (text *)"CALL PKG1.P5( :c );" ;
```

In both these cases, the rest of the program is as follows:

```
OCIStmtPrepare(stmthp, errhp, sqlstmt, (ub4)strlen((char *)sqlstmt),
               (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT);
curlen = 0;

OCIBindByName(stmthp, &bndhp[3], errhp,
               (text *) ":c4", (sb4) strlen((char *) ":c"),
               (dvoid *) buf5, (sb4) LONGLEN, SQLT_CHR,
               (dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
               (ub4) 1, (ub4 *) &curlen, (ub4) OCI_DATA_AT_EXEC);

OCIStmtExecute(svchp, stmthp, errhp, (ub4) 0, (ub4) 0, (const OCISnapshot*) 0,
               (OCISnapshot*) 0, (ub4) OCI_DEFAULT);
```

The PL/SQL procedure PKG1.P5 is as follows:

```
CREATE OR REPLACE PACKAGE BODY pkg1 AS
...
procedure p5 (c OUT CLOB) is
-- This might have been table%rowtype (so it is CLOB now)
BEGIN
...
END p5;

END pkg1;
```

## Converting LONG to LOB Example 13: Calling PL/SQL and C Procedures from PL/SQL

PL/SQL procedures or functions can accept a CLOB or a VARCHAR2 as a formal parameter. For example the PL/SQL procedure could be one of the following:

- *When the formal parameter is CLOB:*

```
CREATE OR REPLACE PROCEDURE get_lob(table_name IN VARCHAR2, lob INOUT
CLOB) AS
...
BEGIN
...
END;
/
```

- *When the formal parameter is VARCHAR2:*

```
CREATE OR REPLACE PROCEDURE get_lob(table_name IN VARCHAR2, lob INOUT
VARCHAR2) AS
...
BEGIN
...
END;
/
```

The calling function could be of any of the following types:

- *When the actual parameter is a CHR:*

```
create procedure ...
declare
c VARCHAR2[200];
begin
  get_lob('table_name', c);
end;
```

- *When the actual parameter is a CLOB:*

```
create procedure ...
declare
c table_name.column_name%type  -- This is a CLOB now
begin
  get_lob('table_name', c);
end;
```

Both the PL/SQL case stubs works with both cases of the actual parameters.

## Summary of New Functionality Associated with the LONG-to-LOB API

### OCI Functions

`OCIDefineByPos()` function now accepts the following types:

- `SQLT_CHR` and `SQLT_LNG` for CLOB columns
- `SQLT_BIN` and `SQLT_LBI` for BLOB column

So, for a LOB column, you can define a `VARCHAR2` buffer and on the subsequent `OCIStmtFetch()` command, you will get the buffer filled with the CLOB/BLOB data.

`OCIBindByPos()` and `OCIBindByName()` functions now accept buffers of up to 4 gigabytes in size.

### SQL Statements

The following new syntax has been added:

```
ALTER TABLE [<schema>.<table_name>
MODIFY ( <long_column_name> { CLOB | BLOB | NCLOB } [DEFAULT <default_
value> ] [LOB_storage_clause];
```

See "[Migrating LONGs to LOBs: Using ALTER TABLE to Change LONG Column to LOB Types](#)" on page 8-6, for details. Changes made to the ALTER TABLE syntax are as follows:

- The "datatype" can now be:
  - (`CLOB|NCLOB`) if the original datatype of the "long\_column\_name" was `LONG`
  - (`BLOB`) if the original datatype was `LONG RAW`
- Only during this (LONG->LOB) conversion can the `LOB_storage_clause` clause be specified in the MODIFY clause.
- During this (LONG->LOB) conversion, you can only specify the `[DEFAULT "expr"]`. No other ALTER TABLE operation is allowed with this operation.

### PL/SQL Interface

You can now use the following PL/SQL SELECT statements:

- SELECT on CLOB columns INTO a character buffer variable, such as CHAR, LONG, or VARCHAR2
- SELECT on BLOB columns INTO a binary buffer variable, such as RAW and LONG RAW

You can also make the following assignments:

- Assign a CLOB (BLOB) to a VARCHAR2 (RAW) variable
- Assign a VARCHAR2(RAW) variable to a CLOB (BLOB)

In addition, a CLOB (BLOB) can be passed as an actual parameter to a function with a formal parameter of VARCHAR2 (RAW) and vice-versa, and can call PL/SQL built-in functions on LOBs.

**See:** [Chapter 7, "Modeling and Design", "SQL Semantics Support for LOBs"](#) on page 7-33.

## Compatibility and Migration

When you ALTER TABLE to change the LONG column to LOB, the table looks as if you never had the LONG column and always had the LOB column. Once you move all LONG data to LOBs, you cannot ALTER the table back to LONG.

[Table 8-1](#) outlines the behavior of various client-server combinations in this release and prior to this release.

**Table 8-1 Client - Server Combinations for Oracle9i and Prior to Oracle9i**

| Client                        | Oracle9i Release 9.0.0 Server | Servers Prior to Oracle9i |
|-------------------------------|-------------------------------|---------------------------|
| Rel.9.0.0 with CHARs          | Server sends data             | Client raises error.      |
| Rel.9.0.0 with LOBs           | Server sends locator          | Server sends locator.     |
| Prior to Rel.9.0.0 with CHARs | Client raises an error        | Client raises error.      |
| Prior to Rel.9.0.0 with LOBs  | Server sends locator          | Server sends locator.     |

## Performance

### **INSERTS and Fetches have Comparable Performance**

A piecewise INSERT or fetch of LOBs using the LONG-to-LOB API has comparable performance to the piecewise INSERT or fetch of LOBs using existing functions like `OCILOBRead()` and `OCILOBWrite()`.

Since Oracle allows >4k data to be inserted into LOBs in a single OCI call, a round-trip to the server is saved.

Also, you can now read LOB data in one `OCIStmtFetch()` call, instead of fetching the LOB locator first and then doing `OCILOBRead()`. This improves performance when you want to read LOB data starting at the beginning (since `OCIStmtFetch()` returns the data from offset 1). Hence the LONG-to-LOB API improves performance of LOB INSERTs and fetches.

### **PL/SQL**

The performance of assigning a `VARCHAR2` buffer to a LOB variable is worse than the performance of the corresponding assignment of `VARCHAR2` to the LONG variable because the former involves creating temporary LOBs. Hence PL/SQL users will see a silent deterioration in the performance of their applications.

## Frequently Asked Questions (FAQs): LONG to LOB Migration

### Moving From LOBs Back to LONGs

#### **Question**

Once we ALTER a table to change LONG columns to LOB and consequently move all LONG data to LOBs, we cannot ALTER the column back to LONG. Is there a work around?

#### **Answer**

There is a workaround for this. You can add a LONG column and use an OCI application to read the data from the LOB column and insert it into the LONG column. Then you can drop the LOB column.

## Is CREATE VIEW Needed?

### Question

Is CREATE VIEW still needed when migrating from LONGs to LOBs?

### Answer

No, you no longer need to use CREATE VIEW. Use the ALTER TABLE statement.

## Are OCI LOB Routines Obsolete?

### Question

How does `OCIStmtFetch()` work for LOB columns? Does it return `OCI_NEED_DATA` as it previously did for LONG column and must data be completely fetched before the data for other columns is available? Are all OCI routines for LOBs obsolete, such as, `OCIlobRead()`, `OCIlobWrite()`, ...?

### Answer

`OCIStmtFetch()` for LOBs works the same way as it did for LONGs previously *if the datatype is specified as `SQLT_LNG/SQLT_CHR`,... in the define*. If the datatype is specified as `SQLT_CLOB` or `SQLT_BLOB`, then the `OCIStmtFetch()` call fetches the LOB locator and you can call `OCIlobRead()` to read LOB data. OCI LOB calls will not be obsolete.

If the datatype is `SQLT_LNG/SQLT_CHR`,... for a LOB column, then the LOB data needs to be completely fetched before the data for other columns are available. The way `SQL*PLUS` can get around this problem is to continue using the existing OCI LOB interface.

## PL/SQL Issues

### Question

Does a fetch of a LOB column (with size > 32K) into a PL/SQL CHAR/RAW/LONG/LONG RAW buffer raise an exception?

### Answer

In `OCIDefineByPos()` and PL/SQL “SELECT INTO” commands, there is no way of specifying the “amount” wanted. You only specify the buffer length. The correct

amount is fetched without overflowing the buffer, no matter what the LOB size is. If the whole column is not fetched, then in OCI a truncation error is returned, and in PL/SQL, an exception is raised.

This behavior is consistent with the existing behavior for LONGs and VARCHAR2s.

## Retrieving an Entire Image if Less Than 32K

### Question

I can now SELECT LOB data without first retrieving the locator. Can I now retrieve an entire image with a single SELECT in PL/SQL if the image is less than 32K?

### Answer

Yes.

## Triggers in LONGs and LOBs

### Question

In Triggers, some functionality is supported for LONGs that is not supported for LOBs. Will this cause a problem?

### Answer

There are a couple of limitations on how LOBs work with triggers. See "[LONG-to-LOB Migration Limitations](#)" on page 8-10.



---

# LOBS: Best Practices

This chapter discusses the following topics:

- [Using SQL\\*Loader](#)
- [LOB Performance Guidelines](#)
- [Moving Data to LOBs in a Threaded Environment](#)
- [Migrating from LONGs to LOBs](#)

## Using SQL\*Loader

You can use SQL\*Loader to bulk load LOBs.

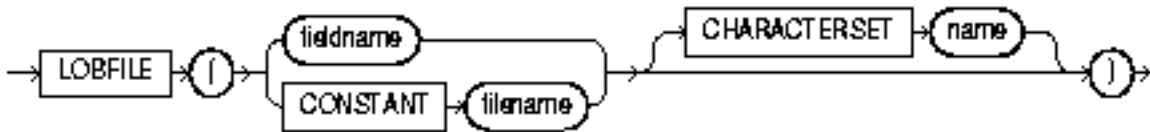
**See:**

- [Chapter 4, "Managing LOBs", "Using SQL\\*Loader to Load LOBs"](#), for a brief description and examples of using SQL\*Loader.
- *Oracle9i Database Utilities* for a detailed descriptions of using SQL\*Loader for loading objects, LOBs, and collections.

## Loading XML Documents Into LOBs With SQL\*Loader

Because LOBs can be quite large, SQL\*Loader can load LOB data from either the main datafile (inline with the rest of the data) or from LOBFILES. [Figure 9-1](#) shows the LOBFILE syntax.

*Figure 9-1 The LOBFILE Syntax*



LOB data can be lengthy enough that it makes sense to load it from a LOBFILE. In LOBFILES, LOB data instances are still considered 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). Therefore, the processing overhead of dealing with records is avoided. This type of organization of data is ideal for LOB loading.

There is no requirement that a LOB from a LOBFILE fit in memory. SQL\*Loader reads LOBFILES in 64K chunks. To load physical records larger than 64K, you can use the READSIZE parameter to specify a larger size.

It is best to load XMLType columns or columns containing XML data in CLOBs, using LOBFILES.

- *When the XML is valid.* If the XML data in the LOBFILE is large and you know that the data is valid XML, then use *direct-path load* since it bypasses all the XML validation processing.
- *When the XML needs validating.* If it is imperative that the validity of the XML data be checked, then use *conventional path load*, keeping in mind that it is not as efficient as a direct-path load.

A *conventional path load* executes SQL INSERT statements to populate tables in an Oracle database. A direct path load eliminates much of the Oracle database overhead by formatting Oracle data blocks and writing the data blocks directly to the database files.

A *direct-path load* does not compete with other users for database resources, so it can usually load data at near disk speed. Considerations inherent to direct path loads, such as restrictions, security, and backup implications, are discussed in Chapter 9 of *Oracle9i Database Utilities*.

Figure 9–2 illustrates SQL\*Loader’s direct-path load and conventional path loads.

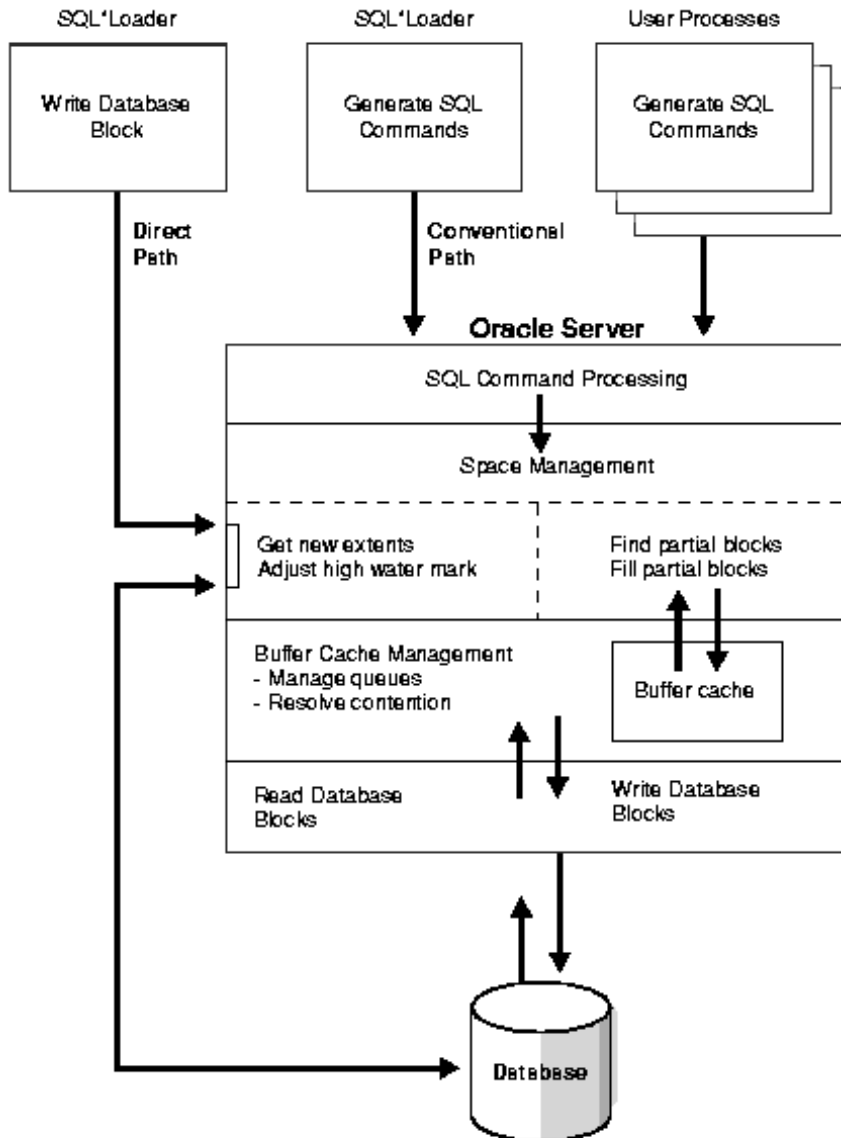
Tables to be loaded must already exist in the database. SQL\*Loader never creates tables. It loads existing tables that either already contain data or are empty.

The following privileges are required for a load:

- You must have INSERT privileges on the table to be loaded.
- You must have DELETE privilege on the table to be loaded, when using the REPLACE or TRUNCATE option to empty out the table's old data before loading the new data in its place.

**See Also:** *Oracle9i Database Utilities* for more information on: loading objects, LOBs, and Collections; and SQL\*Loader case studies.

**Figure 9–2 SQL\*Loader: Direct-Path and Conventional Path Loads**



## LOB Performance Guidelines

- [General Performance Guidelines](#)
- [Temporary LOB Performance Guidelines](#)

### General Performance Guidelines

Use the following guidelines to achieve maximum performance with LOBs:

- *When Possible, Read/Write Large Data Chunks at a Time:* 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.
- *Use LOB Buffering to Read/Write Small Chunks of Data:* 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:** [Chapter 5, "Large Objects: Advanced Topics", "LOB Buffering Subsystem"](#) on page 5-18 for more information on LOB buffering.

- *Use `OCILOBRead()` and `OCILOBWrite()` with Callback:* So that data is streamed to and from the LOB. Ensure 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 operations:
  - `SQL UPDATE` which replaces the entire LOB value

- 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.
- Try to commit changes frequently.

**See Also:** [Chapter 7, "Modeling and Design", "Performance Attributes When Using SQL Semantics with LOBs"](#) on page 7-51, for information about performance issues when using SQL semantics with LOBs

### Some Performance Numbers

[Table 9-1](#) lists the results of a performance test that enqueued 500 messages using a chunk size of 8KB for the LOB part of the payload. This performance test used Oracle8i Release 3 (8.1.7), and a `DB_BLOCKSIZE = 8192 (8K)`, identical to the operating system block size.

**Table 9-1 Response Time When Enqueueing 500 Messages With and Without CACHE and LOGGING**

| CHUNK SIZE | CACHE (Y/N) | LOGGING (Y/N) | MESSAGE_SIZE | RESPONSE TIME |
|------------|-------------|---------------|--------------|---------------|
| 8k         | NOCACHE     | NOLOGGING     | 3900 bytes   | 01:33 sec     |
| 8k         | NOCACHE     | NOLOGGING     | 4000 bytes   | 06:19 sec     |
| 8k         | NOCACHE     | NOLOGGING     | 4000 bytes   | 04:36 sec     |
| 8k         | CACHE       |               | 3900 bytes   | 01:22 sec     |
| 8k         | CACHE       |               | 4000 bytes   | 01:22 sec     |
| 8k         | CACHE       |               | 4000 bytes   | 01:83 sec     |
| 8k         | CACHE       |               | 20000 bytes  | 02:33 sec     |

Previous response times using a 16k chunksize, NOCACHE, and NOLOGGING for a message of 4000 bytes was 12:28 sec.

These results indicate that the CACHE parameter is the parameter giving the best performance improvement.

## Temporary LOB Performance Guidelines

In addition to the guidelines described earlier under "[LOB Performance Guidelines](#)" on LOB performance in general, here are some guidelines for using temporary LOBs:

- *Use a separate temporary tablespace for temporary LOB storage instead of the default system tablespace.* This avoids device contention when copying data from persistent LOBs to temporary LOBs.

If you use the newly provided enhanced SQL semantics functionality in your applications, there will be many more temporary LOBs created silently in SQL and PL/SQL than before. Ensure that *temporary tablespace* for storing these temporary LOBs is *large enough* for your applications. In particular, these temporary LOBs are silently created when you use the following:

- SQL functions on LOBs
- PL/SQL built-in character functions on LOBs
- Variable assignments from VARCHAR2/RAW to CLOBs/BLOBs, respectively.
- Perform a LONG-to-LOB migration
- *In PLSQL, use NOCOPY to pass temporary LOB parameters by reference whenever possible.* Refer to the *PL/SQL User's Guide and Reference*, for more information on passing parameters by reference and parameter aliasing.
- *Using Temporary LOBs in PL/SQL Procedure Loops.* When repeatedly creating temporary LOBs in PL/SQL procedures in a loop, performance is improved when a PL/SQL package LOB locator variable is used inside the procedure instead of using a local variable.

This is because, by using a package variable which persists in a session, allocating extra memory to manage temporary LOBs in every procedure call is avoided.

---

**Note:** Temporary LOBs created using a session locator are not cleaned up automatically at the end of function or procedure calls. The temporary LOB should be explicitly freed by calling DBMS\_LOB.FREETEMPORARY().

---

```
CREATE OR REPLACE PACKAGE pk IS
    tmplob clob;
```

```
END pk;
/
CREATE OR REPLACE PROCEDURE temp_lob_proc
IS
BEGIN
    -- instead of using a local LOB variabe, use a package variable here
    DBMS_LOB.CREATETEMPORARY(pk.tmplob, TRUE);
    -- Do some LOB data manipulation here
    DBMS_LOB.FREETEMPORARY(pk.tmplob);
END;
/
DECLARE
    doc CLOB;
BEGIN
    FOR i IN 1..400 LOOP
        temp_lob_proc();
    END LOOP;
END;
/
```

- *Take advantage of buffer cache on temporary LOBs.* Temporary LOBs created with the CACHE parameter set to true move through the buffer cache. Otherwise temporary LOBs are read directly from, and written directly to, disk.
- *Be aware of the cost incurred in assigning temporary LOB variables.* Temporary LOBs create entirely new copies of themselves on assignments. For example:

```
LOCATOR1 BLOB;
LOCATOR2 BLOB;
DBMS_LOB.CREATETEMPORARY (LOCATOR1,TRUE,DBMS_LOB.SESSION);
LOCATOR2 := LOCATOR1;
```

This code causes a copy of the temporary LOB pointed to by LOCATOR1 to be created. When passing temporary LOB parameters to procedures or functions, you might also want to consider using pass by reference semantics in PL/SQL.

In OCI, to ensure copy semantics of LOB locators and data, OCILobLocatorAssign is used to copy temporary LOB locators as well as the LOB data. To avoid the deep copy, pointer assignment can be done, if copy semantics of locator copy is not intended. For example:

```
OCILOBLocator *LOC1;
OCILOBLocator *LOC2;
OCILOBCREATETEMPORARY (LOC1,TRUE,OCIDURATIONSESSION);
LOC2 = LOC1;
```



- *Use OCI\_OBJECT mode for temporary LOBs*

To improve the performance of temporary LOBs on LOB assignment, use OCI\_OBJECT mode for OCILobLocatorAssign. In the OCI\_OBJECT mode, Oracle tries to minimize the number of deep copies to be done. Hence, after OCILobLocatorAssign is done on a source temporary LOB in OCI\_OBJECT mode, the source and the destination locators will point to the same LOB until any modification is made through either LOB locator.

- *Free up temporary LOBs returned from SQL queries and PLSQL programs.*

In PL/SQL, C (OCI), Java and other programmatic interfaces, SQL query results or PLSQL program executions return temporary LOBs for operation/function calls on LOBs. For example:

```
SELECT substr(CLOB_Column, 4001, 32000) FROM ...
```

If the query is executed in PLSQL, the returned temporary LOBs automatically get freed at the end of a PL/SQL program block. You can also explicitly free the temporary LOBs any time. In OCI and Java, the returned temporary LOB needs to be freed by the user explicitly.

Without proper deallocation of the temporary LOBs returned from SQL queries, temporary tablespace gets filled up steadily and you could observe performance degradation.

## Moving Data to LOBs in a Threaded Environment

### Incorrect procedure

The following sequence, requires a new connection when using a threaded environment, adversely affects performance, and is inaccurate:

1. Create an empty (non-NULL) LOB
2. INSERT using the empty LOB
3. SELECT-FOR-UPDATE of the row just entered
4. Move data into the LOB
5. COMMIT. This releases the SELECT-FOR-UPDATE locks and makes the LOB data persistent.

## The Correct Procedure

---

---

**Note:**

- There is no need to 'create' an empty LOB.
  - You can use the RETURNING clause as part of the INSERT/UPDATE statement to return a locked LOB locator. This eliminates the need for doing a SELECT-FOR-UPDATE, as mentioned in step 3.
- 
- 

Hence the preferred procedure is as follows:

1. INSERT an empty LOB, RETURNING the LOB locator.
2. Move data into the LOB using this locator.
3. COMMIT. This releases the SELECT-FOR-UPDATE locks, and makes the LOB data persistent.

Alternatively, you can insert >4,000 byte of data directly for the LOB columns but not the LOB attributes.

**See Also:** [Chapter 14, "LOBs Case Studies"](#)

## Migrating from LONGs to LOBs

During migration from LONGs to LOBs, the redo changes for the table are logged only if the table has LOGGING on. Redo changes for the column being converted from LONG to LOB are logged only if the storage characteristics of the LOB indicate LOGGING. The default value for LOGGING | NOLOGGING for the LOB is inherited from the tablespace in which the LOB is being created.

### Preventing Generation of Redo Space During Migration

To prevent generation of redo space during migration and migrate smoothly, use the following statements:

1. ALTER TABLE Long\_tab NOLOGGING;
2. ALTER TABLE Long\_tab MODIFY ( long\_col CLOB [default <default\_val>]) LOB (long\_col) STORE AS (... NOLOGGING ...);
3. ALTER TABLE Long\_tab MODIFY LOB long\_col STORE AS (...LOGGING...);

4. ALTER TABLE Long\_tab LOGGING;
5. Take a backup of the tablespaces containing the table and the LOB.

**See Also:** [Chapter 8, "Migrating From LONGs to LOBs"](#)



---

## Internal Persistent LOBs

### Use Case Model

This chapter describes each operation on LOBs (such as "Write Data to a LOB") in terms of a use case. [Table 10–1, "Internal Persistent LOB Basic Operations"](#), alphabetically lists all these use cases.

### Individual Use Cases

Each detailed internal persistent LOB use case operation description is laid out as follows:

- *Use case figure.* This UML drawing depicts the use case. [Appendix A, "Unified Modeling Language Diagrams"](#) explains how to interpret these diagrams.
- *Purpose.* The purpose of this LOB use case.
- *Usage Notes.* Guidelines to assist your implementation of the LOB operation.
- *Syntax.* The main syntax used to perform the LOBs related activity.
- *Scenario.* Describes the implementation of the use case in terms of the sample schema used in the examples. For further details on sample schemas, refer to *Oracle9i Sample Schemas*.
- *Examples.* How to apply each use case based on the sample schema used.

## Use Case Model: Internal Persistent LOBs Operations

**Table 10-1**, indicates with a + where examples are provided for specific use cases in each programmatic environment. An "S" indicates that SQL is used for that use case and applicable programmatic environment(s).

We refer to programmatic environments by means of the following abbreviations:

- **P** — PL/SQL, using the DBMS\_LOB Package
- **O** — C, using OCI (Oracle Call Interface)
- **CP** — C++, using OCCI (Oracle C++ Call Interface)
- **B** — COBOL, using Pro\*COBOL precompiler
- **C** — C/C++, using Pro\*C/C++ precompiler
- **V** — Visual Basic, using OO4O (Oracle Objects for OLE)
- **J** — Java, using JDBC (Java Database Connectivity)
- **S** — SQL

**Table 10-1 Internal Persistent LOB Basic Operations**

| LOB Use Case   | P        | O        | CP | B        | C        | V        | J        |
|--|----------|----------|----|----------|----------|----------|----------|
| <a href="#">Appending One LOB to Another</a> on page 10-193  | +        | +        | -  | +        | +        | +        | +        |
| <a href="#">Java (JDBC): Appending One LOB to Another</a> on page 10-201                           | +        | +        | -  | +        | +        | -        | +        |
| <a href="#">Character Set Form: Determining Character Set Form</a> on page 10-189                  | -        | +        | -  | -        | -        | -        | -        |
| <a href="#">Character Set ID: Determining Character Set ID</a> on page 10-185                      | -        | +        | -  | -        | -        | -        | -        |
| <a href="#">Checking In a LOB</a> on page 10-83  | +        | +        | -  | +        | +        | +        | +        |
| <a href="#">Checking Out a LOB</a> on page 10-71   | +        | +        | -  | +        | +        | +        | +        |
| <a href="#">Closing LOBs - see Chapter 3, "LOB Support in Different Programmatic Environments"</a> | -        | -        | -  | -        | -        | -        | -        |
| <a href="#">Comparing All or Part of Two LOBs</a> on page 10-128                                   | +        | -        | -  | +        | +        | +        | +        |
| <a href="#">Copying a LOB Locator</a> on page 10-164   | +        | +        | -  | +        | +        | +        | +        |
| <a href="#">Copying All or Part of One LOB to Another LOB</a> on page 10-153                       | +        | +        | -  | +        | +        | +        | +        |
| <a href="#">Creating a Nested Table Containing a LOB</a> on page 10-13                             | <b>S</b> | <b>S</b> | -  | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |

**Table 10–1 Internal Persistent LOB Basic Operations (Cont.)**

| <b>LOB Use Case</b>  | <b>P</b> | <b>O</b> | <b>CP</b> | <b>B</b> | <b>C</b> | <b>V</b> | <b>J</b> |
|--|----------|----------|-----------|----------|----------|----------|----------|
| <a href="#">Creating a Table Containing an Object Type with a LOB Attribute</a> on page 10-10                                | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">Creating a Table Containing One or More LOB Columns</a> on page 10-5   | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">Creating a Varray Containing References to LOBs</a> See Chapter 5, "Large Objects: Advanced Topics" on page 5-28 | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">Deleting the Row of a Table Containing a LOB</a> on page 10-286  | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">Disabling LOB Buffering</a> on page 10-263   | -        | +        | -         | +        | +        | +        | -        |
| <a href="#">Displaying LOB Data</a> on page 10-98  | +        | +        | -         | +        | +        | +        | +        |
| <a href="#">Enabling LOB Buffering</a> on page 10-251  | -        | -        | -         | +        | +        | +        | -        |
| <a href="#">Equality: Checking If One LOB Locator Is Equal to Another</a> on page 10-173                                     | -        | +        | -         | -        | +        | -        | +        |
| <a href="#">Erasing Part of a LOB</a> on page 10-241   | +        | +        | -         | +        | +        | +        | +        |
| <a href="#">Flushing the Buffer</a> on page 10-257   | -        | +        | -         | +        | +        | -        | -        |
| <a href="#">Initialized Locator: Checking If a LOB Locator Is Initialized</a> on page 10-180                                 | -        | +        | -         | -        | +        | -        | -        |
| <a href="#">Inserting a LOB Value using EMPTY_CLOB() or EMPTY_BLOB()</a> on page 10-16                                       | S        | S        | S         | S        | S        | S        | +        |
| <a href="#">Inserting a Row by Initializing a LOB Locator Bind Variable</a> on page 10-23                                    | S        | +        | -         | +        | +        | +        | +        |
| <a href="#">Inserting a Row by Selecting a LOB From Another Table</a> on page 10-20  | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">Length: Determining the Length of a LOB</a> on page 10-144   | +        | +        | -         | +        | +        | +        | +        |
| <a href="#">Loading a LOB with Data From a BFILE</a> on page 10-32   | +        | +        | -         | +        | +        | +        | +        |
| <a href="#">Loading an Internal Persistent BLOB with Binary Data from a BFILE</a> on page 10-43                              | +        | -        | -         | -        | -        | -        | -        |
| <a href="#">Loading an Internal Persistent CLOB with BFILE Data</a> on page 10-47  | +        | -        | -         | -        | -        | -        | -        |
| <a href="#">LONG to LOB Copying, Using the TO_LOB Operator</a> on page 10-66   | S        | S        | S         | S        | S        | S        | S        |
| <a href="#">LONG to LOB Migration Using the LONG-to-LOB API</a> on page 10-62  | +        | +        | -         | -        | -        | -        | -        |
| <a href="#">Java (JDBC): Appending One LOB to Another</a> on page 10-201   | +        | +        | -         | +        | +        | -        | +        |

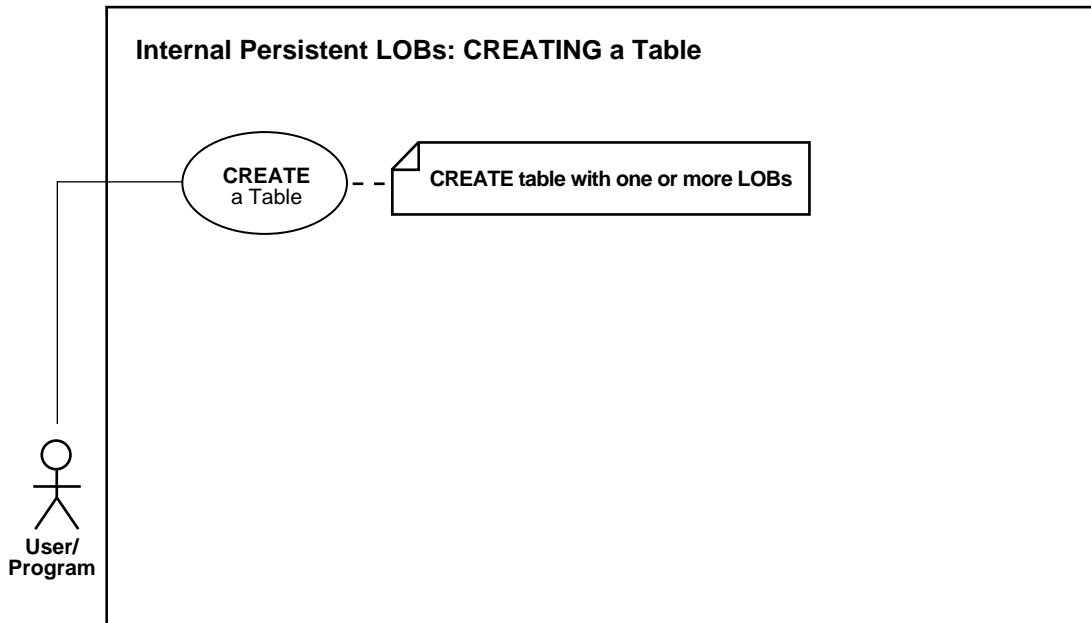
**Table 10–1 Internal Persistent LOB Basic Operations (Cont.)**

| <b>LOB Use Case</b>   | <b>P</b> | <b>O</b> | <b>CP</b> | <b>B</b> | <b>C</b> | <b>V</b> | <b>J</b> |
|---|----------|----------|-----------|----------|----------|----------|----------|
| Opening LOBs - see <a href="#">Chapter 3, "LOB Support in Different Programmatic Environments"</a>  | -        | -        | -         | -        | -        | -        | -        |
| <a href="#">Patterns: Checking for Patterns in the LOB (instr)</a> on page 10-137   | +        | -        | -         | +        | +        | -        | +        |
| <a href="#">Reading a Portion of the LOB (substr)</a> on page 10-120  | +        | -        | -         | +        | +        | +        | +        |
| <a href="#">Reading Data from a LOB</a> on page 10-109  | +        | +        | -         | +        | +        | +        | +        |
| Streaming LOB Data. See " <a href="#">JDBC: New LOB Streaming APIs</a> " on page 3-61. Note: This API has not yet been incorporated in this chapter as a use case. See a forthcoming release. | -        | -        | -         | -        | -        | -        | +        |
| <a href="#">Trimming LOB Data</a> on page 10-229  | +        | +        | -         | +        | +        | +        | +        |
| <a href="#">Updating a LOB with EMPTY_CLOB() or EMPTY_BLOB()</a> on page 10-271   | <b>S</b> | <b>S</b> | <b>S</b>  | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |
| <a href="#">Updating a Row by Selecting a LOB From Another Table</a> on page 10-275   | <b>S</b> | <b>S</b> | <b>S</b>  | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |
| <a href="#">Updating by Initializing a LOB Locator Bind Variable</a> on page 10-277   | <b>S</b> | +        | -         | +        | +        | +        | +        |
| Write-Append, see <a href="#">Append-Writing to the End of a LOB</a> on page 10-203.  | -        | -        | -         | -        | -        | -        | -        |
| <a href="#">Writing Data to a LOB</a> on page 10-212  | +        | +        | +         | +        | +        | +        | +        |



## Creating a Table Containing One or More LOB Columns

Figure 10–1 Use Case Diagram: Creating a Table Containing one or More LOB Columns



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2.
- [Creating a Table Containing an Object Type with a LOB Attribute](#) on page 10-10.
- [Creating a Nested Table Containing a LOB](#) on page 10-13.
- [Creating a Varray Containing References to LOBs](#) in [Chapter 5, "Large Objects: Advanced Topics"](#) on page 5-28.

### Purpose

This procedure describes how to create a table containing one or more LOB columns.

## Usage Notes

When you use functions, `EMPTY_BLOB()` and `EMPTY_CLOB()`, the resulting LOB is initialized, but not populated with data. LOBs that are empty are not null, and vice versa. This is discussed in more detail in ["Inserting a LOB Value using EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)"](#) on page 10-16.

- For information about creating *nested* tables that have one or more columns of LOB datatype, see ["Creating a Nested Table Containing a LOB"](#) on page 10-13.
- Creating an object column containing one or more LOBs is discussed under the heading, ["Creating a Table Containing an Object Type with a LOB Attribute"](#) on page 10-10.

When creating tables that contain LOBs use the guidelines and examples described in the following:

- [Chapter 2, "Basic LOB Components"](#), ["Initializing Internal LOBs to NULL or Empty"](#)
- [Chapter 4, "Managing LOBs"](#)
- [Chapter 7, "Modeling and Design"](#)

### See Also:

*Oracle9i SQL Reference* for a complete specification of syntax for using LOBs in `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

## Syntax

Use the following syntax reference:

- SQL: *Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — `CREATE TABLE`

## Scenario

These examples use the following Oracle9i Sample Schemas:

- Human Resources (HR)
- Order Entry (OE)

- Product Media (PM)

You must create the HR and OE schemas before you create the PM schema. For details on these schemas, you must refer to *Oracle9i Sample Schemas*.

### Examples

How to create a table containing a LOB column is illustrated with the following example, in SQL:

- [SQL: Create a Table Containing One or More LOB Columns](#)

## SQL: Create a Table Containing One or More LOB Columns

You may need to set up the following data structures for certain examples in this manual to work.

---



---

**Note:** Since 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.

---



---

```

/* Setup script for creating Print_media,
   Online_media and associated structures
*/

DROP USER pm CASCADE;
DROP DIRECTORY ADPHOTO_DIR;
DROP DIRECTORY ADCOMPOSITE_DIR;
DROP DIRECTORY ADGRAPHIC_DIR;
DROP INDEX onlinemedia CASCADE CONSTRAINTS;
DROP INDEX printmedia CASCADE CONSTRAINTS;
DROP TABLE online_media CASCADE CONSTRAINTS;
DROP TABLE print_media CASCADE CONSTRAINTS;
DROP TYPE textdoc_typ;
DROP TYPE textdoc_tab;
DROP TYPE adheader_typ;
DROP TABLE adheader_typ;
CREATE USER pm;
GRANT CONNECT, RESOURCE to pm;

CREATE DIRECTORY ADPHOTO_DIR AS '/tmp/';
CREATE DIRECTORY ADCOMPOSITE_DIR AS '/tmp/';
CREATE DIRECTORY ADGRAPHIC_DIR AS '/tmp/';

```

```
CREATE DIRECTORY media_dir AS '/tmp/';
GRANT READ ON DIRECTORY ADPHOTO_DIR to pm;
GRANT READ ON DIRECTORY ADCOMPOSITE_DIR to pm;
GRANT READ ON DIRECTORY ADGRAPHIC_DIR to pm;
GRANT READ ON DIRECTORY media_dir to pm;

CONNECT pm/pm (or &pass);
COMMIT;

CREATE TABLE a_table (blob_col BLOB);

CREATE TYPE adheader_typ AS OBJECT (
    header_name    VARCHAR2(256),
    creation_date  DATE,
    header_text    VARCHAR(1024),
    logo           BLOB );

CREATE TYPE textdoc_typ AS OBJECT (
    document_typ   VARCHAR2(32),
    formatted_doc  BLOB);

CREATE TYPE Textdoc_ntab AS TABLE OF textdoc_typ;

CREATE TABLE adheader_tab OF adheader_typ (
    Ad_finaltext  DEFAULT EMPTY_CLOB(), CONSTRAINT
    Take CHECK (Take IS NOT NULL),  DEFAULT NULL);

CREATE TABLE online_media
( product_id NUMBER(6),
  product_photo ORDSYS.ORDImage,
  product_photo_signature ORDSYS.ORDImageSignature,
  product_thumbnail ORDSYS.ORDImage,
  product_video ORDSYS.ORDVideo,
  product_audio ORDSYS.ORDAudio,
  product_text CLOB,
  product_testimonials ORDSYS.ORDDoc);

CREATE UNIQUE INDEX onlinemedia_pk
  ON online_media (product_id);

ALTER TABLE online_media
ADD (CONSTRAINT onlinemedia_pk
PRIMARY KEY (product_id), CONSTRAINT loc_c_id_fk
FOREIGN KEY (product_id) REFERENCES oe.product_information(product_id)
);
```

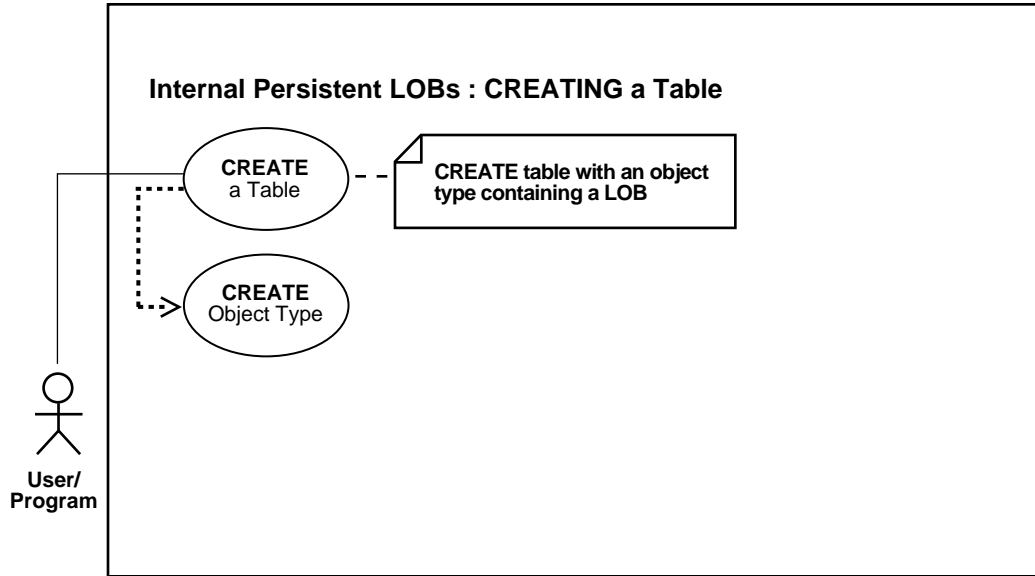
```
CREATE TABLE print_media
(product_id NUMBER(6),
ad_id NUMBER(6),
ad_composite BLOB,
ad_sourcetext CLOB,
ad_finalextext CLOB,
ad_fktextn NCLOB,
ad_testdocs_ntab textdoc_tab,
ad_photo BLOB,
ad_graphic BFILE,
ad_header adheader_typ,
press_release LONG) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;

CREATE UNIQUE INDEX printmedia_pk
ON print_media (product_id, ad_id);

ALTER TABLE print_media
ADD (CONSTRAINT printmedia_pk
PRIMARY KEY (product_id, ad_id),
CONSTRAINT printmedia_fk FOREIGN KEY (product_id)
REFERENCES oe.product_information(product_id)
);
```

## Creating a Table Containing an Object Type with a LOB Attribute

Figure 10–2 Use Case Diagram: Creating a Table Containing an Object Type with a LOB Attribute



**See Also:**

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2.
- [Creating a Table Containing One or More LOB Columns](#) on page 10-5.
- [Creating a Nested Table Containing a LOB](#) on page 10-13.
- [Creating a Varray Containing References to LOBs](#) in Chapter 5, "Large Objects: Advanced Topics" on page 5-28.

**Purpose**

This procedure describes how to create a table containing an object type with a LOB attribute.

## Usage Notes

When creating tables that contain LOBs use the guidelines and examples described in the following:

- [Chapter 2, "Basic LOB Components", "Initializing Internal LOBs to NULL or Empty"](#)
- [Chapter 4, "Managing LOBs"](#)
- [Chapter 7, "Modeling and Design"](#)

## Syntax

See the following specific reference for a detailed syntax description:

- SQL: *Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — CREATE TABLE.

## Scenario

You must create the object type that contains LOB attributes before you can proceed to create a table that makes use of that object type.

This example uses the Product Media schema included with Oracle9i sample schemas. For details on this schema, you must refer to the *Oracle9i Sample Schemas* manual.

## Examples

This example is provided in SQL and applies to all programmatic environments:

- [SQL: Creating a Table Containing an Object Type with a LOB Attribute](#)

## SQL: Creating a Table Containing an Object Type with a LOB Attribute

Create `adheader_typ` as a basis for tables containing ad headings or titles and logos used in these examples:

```
CREATE TYPE adheader_typ AS OBJECT (
  header_name  VARCHAR2(256),
  creation_date DATE,
  header_text  VARCHAR(1024),
  logo         BLOB );

/* Create table adheader_tab Using SQL DDL: */
CREATE TABLE adheader_tab of adheader_typ (
  logo DEFAULT EMPTY_BLOB() CONSTRAINT Take CHECK (Take IS NOT NULL),
  creation_date DATE );
```

Create Type `adheader_typ` using SQL DDL as a basis for a table that will contain the column object:

```
DROP TYPE adheader;
DROP TABLE adheader_tab;
CREATE TYPE adheader_typ AS OBJECT (
    header_name    VARCHAR2(256),
    creation_date  DATE,
    header_text    VARCHAR(1024),
    logo           BLOB );

/* Create support table adheader_tab as an archive of
   ad headers using SQL DDL: */
CREATE TABLE adheader_tab OF adheader_typ;
```

**See Also:** *Oracle9i SQL Reference* for a complete specification of the syntax for using LOBs in DDL commands, CREATE TYPE and ALTER TYPE with BLOB, CLOB, and BFILE attributes.

---

---

**Note:** NCLOBs cannot be attributes of an object type.

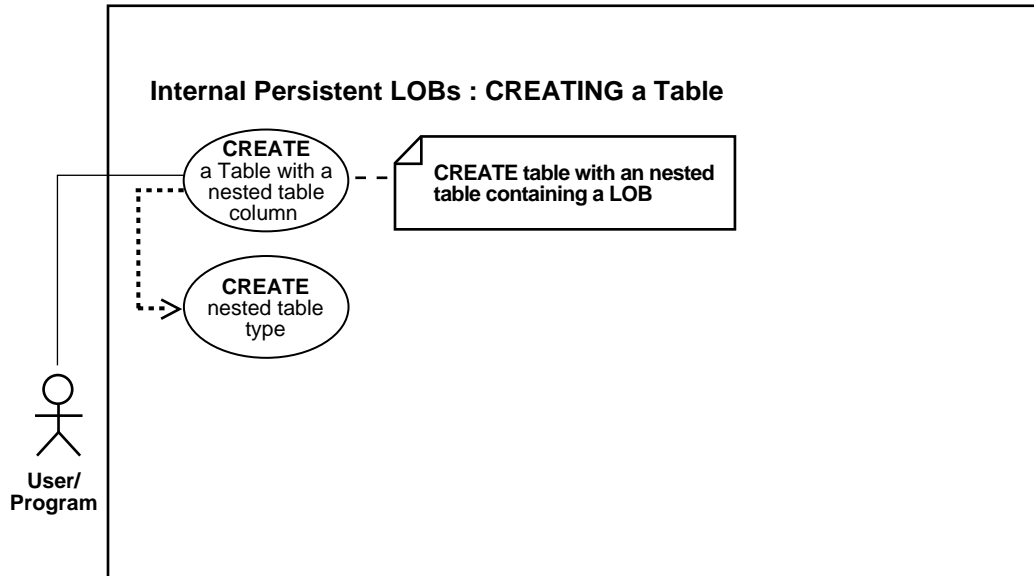
---

---



## Creating a Nested Table Containing a LOB

Figure 10–3 Use Case Diagram: Creating a Nested Table Containing a LOB



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2.
- [Creating a Table Containing One or More LOB Columns](#) on page 10-5.
- [Creating a Table Containing an Object Type with a LOB Attribute](#) on page 10-10.
- [Creating a Varray Containing References to LOBs](#) in Chapter 5, "Large Objects: Advanced Topics" on page 5-28.

### Purpose

This procedure creates a nested table containing a LOB.

## Usage Notes

When creating tables that contain LOBs use the guidelines and examples described in the following sections and these chapters:

- [Chapter 2, "Basic LOB Components"](#), ["Initializing Internal LOBs to NULL or Empty"](#)
- [Chapter 4, "Managing LOBs"](#)
- [Chapter 7, "Modeling and Design"](#)

## Syntax

Use the following syntax reference:

- SQL: *Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — CREATE TABLE.

## Scenario

Create the object type that contains the LOB attributes before you create a nested table based on that object type. In our example, table `Print_media` contains nested table `ad_textdoc_ntab` that has type `textdoc_tab`. This type uses two LOB datatypes:

- `BFILE` - an advertisement graphic
- `CLOB` - an advertisement transcript

We have already described how to create a table with LOB columns in the previous section (see ["Creating a Table Containing One or More LOB Columns"](#) on page 10-5), so here we only describe the syntax for creating the underlying object type.

## Examples

The example ["SQL: Creating a Nested Table Containing a LOB"](#) is provided in the SQL programmatic environment.

## SQL: Creating a Nested Table Containing a LOB

```
/* Create type textdoc_typ as the base type
   for the nested table textdoc_ntab,
   where textdoc_ntab contains a LOB:
*/
DROP TYPE textdoc_typ force;
DROP TYPE textdoc_ntab;
```

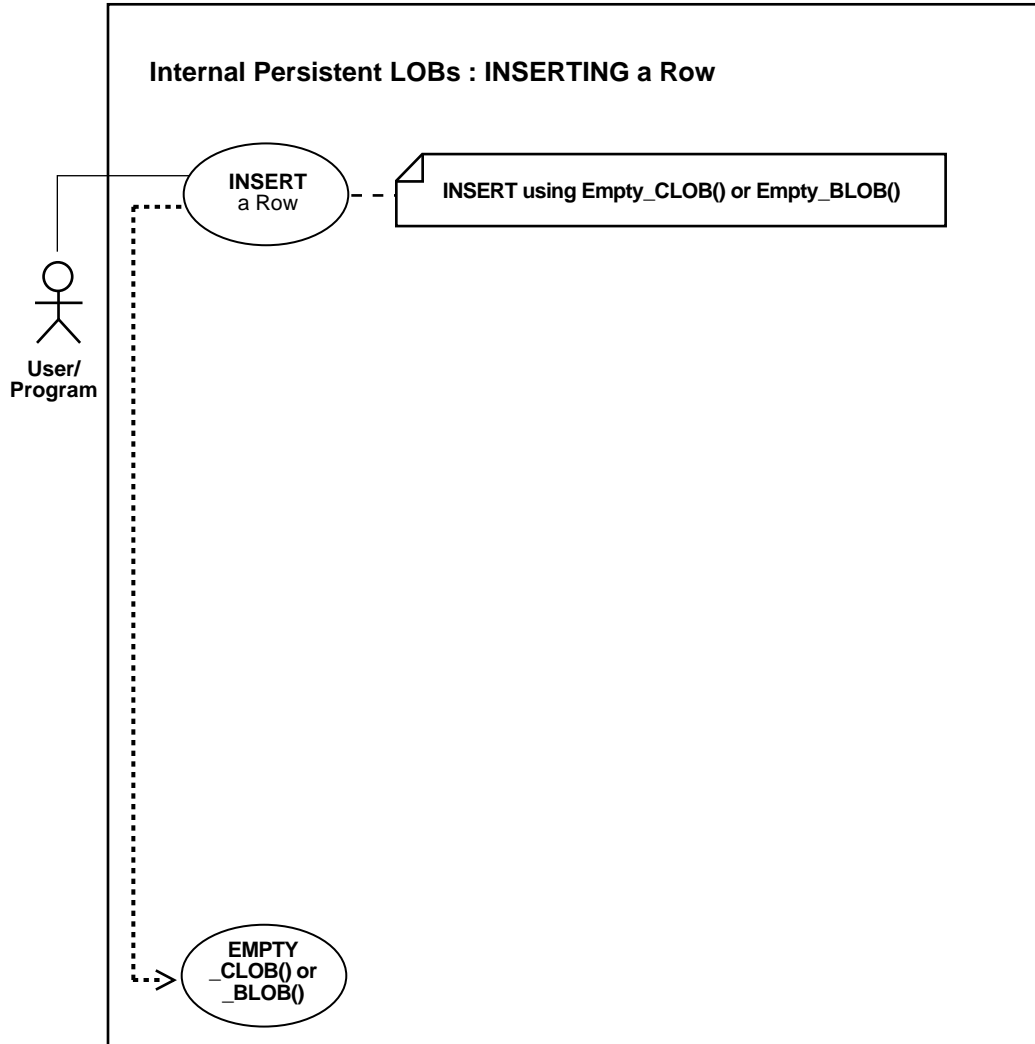
```
DROP TABLE textdoc_nstable;
CREATE TYPE textdoc_typ AS OBJECT
(
    document_typ    VARCHAR2(32),
    formatted_doc   BLOB
);

/* The type has been created. Now you need a */
/* nested table of that type to embed in */
/* table Print_media, so: */
CREATE TYPE textdoc_ntab AS TABLE OF textdoc_typ;
CREATE TABLE textdoc_nstable (
    id number,
    textdoc_ntab textdoc_typ)
NESTED TABLE textdoc_ntab STORE AS textdoc_nestedtab;
```

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 the `NESTED TABLE` statement when the `Print_media` table is created.

## Inserting a LOB Value using EMPTY\_CLOB() or EMPTY\_BLOB()

Figure 10-4 Use Case Diagram: Inserting a Row Using EMPTY\_CLOB() or EMPTY\_BLOB()



**See Also:**

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Inserting a Row by Selecting a LOB From Another Table](#) on page 10-20
- [Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-23

**Purpose**

This procedure describes how to insert a LOB value using EMPTY\_CLOB() or EMPTY\_BLOB().

**Usage Notes**

Here are guidelines for inserting LOBs:

**Before inserting, Make the LOB Column Non-Null**

Before you write data to an internal LOB, make the LOB column non-null; that is, the LOB column 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 also initialize a LOB column with a character or raw string less than 4,000 bytes in size. For example:

```
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
VALUES (1, 1, 'This is a One Line Advertisement');
```

You can perform this initialization during CREATE TABLE (see ["Creating a Table Containing One or More LOB Columns"](#)) or, as in this case, by means of an INSERT.

**Inserting LOBs For Binds of More Than 4,000 Bytes**

For guidelines on how to INSERT into a LOB when binds of more than 4,000 bytes are involved, see the following sections in [Chapter 7, "Modeling and Design"](#):

- [Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs](#) on page 7-14

- [Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion](#) on page 7-15
- [Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE](#) on page 7-16
- [Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported](#) on page 7-18
- [Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes](#) on page 7-18

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- SQL: *Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — INSERT.
- C (OCI): There is no applicable syntax reference for this use case.
- C++ (OCCI): There is no applicable syntax reference for this use case.
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): There is no applicable syntax reference for this use case.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

**See Also:** *Oracle9i Sample Schemas* for a description of the PM Schema and the `Print_media` table used in this example.

### Examples

Examples are provided in the following programmatic environments:

- [SQL: Inserting a Value Using EMPTY\\_CLOB\(\) / EMPTY\\_BLOB\(\)](#) on page 10-19
- C (OCI): No example is provided with this release.
- C++ (OCCI): No example is provided with this release.

- C/C++ (ProC/C++): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

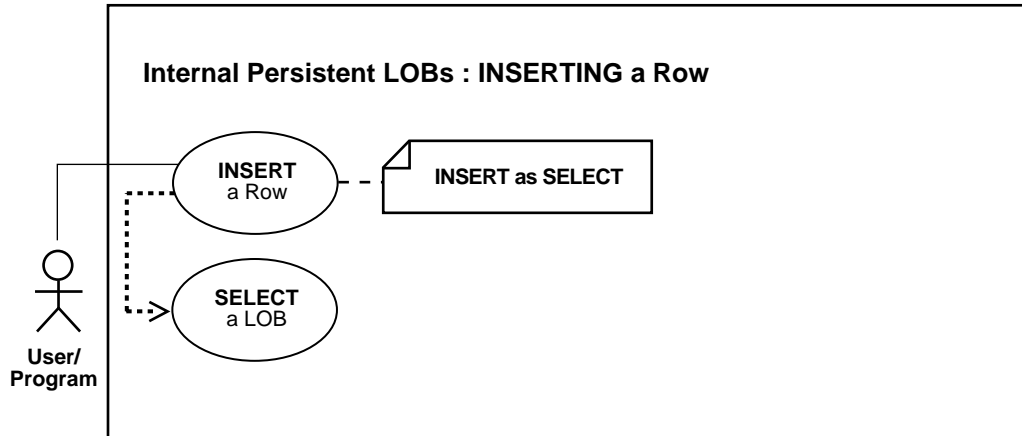
## SQL: Inserting a Value Using EMPTY\_CLOB() / EMPTY\_BLOB()

These functions are special functions in Oracle SQL, and are not part of the DBMS\_LOB package.

```
/* In the new row of table Print_media,
   the columns ad_sourcetext and ad_fltextn are initialized using EMPTY_CLOB(),
   the columns ad_composite and ad_photo are initialized using EMPTY_BLOB(),
   the column formatted-doc in the nested table is initialized using EMPTY_
BLOB(),
   the column logo in the column object is initialized using EMPTY_BLOB(): */
INSERT INTO Print_media
VALUES (3060,11001, EMPTY_BLOB(), EMPTY_CLOB(),EMPTY_CLOB(),EMPTY_CLOB(),
textdoc_tab(textdoc_typ ('HTML', EMPTY_BLOB())), EMPTY_BLOB(), NULL,
adheader_typ('any header name', <any date>, 'ad header text goes here',
EMPTY_BLOB()),
'Press release goes here');
```

## Inserting a Row by Selecting a LOB From Another Table

*Figure 10–5 Use Case Diagram: Inserting a Row by Selecting a LOB From Another Table*



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Inserting a LOB Value using EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)](#) on page 10-16
- [Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-23

### Purpose

This procedure describes how to insert a row containing a LOB as SELECT.



## Usage Notes

---

---

**Note:** Internal LOB types BLOB, CLOB, and NCLOB, use *copy semantics*, as opposed to *reference semantics* that apply to BFILES. When a BLOB, CLOB, or NCLOB is copied from one row to another in the same table or a different table, the *actual* LOB value is copied, not just the LOB locator.

---

---

For example, assuming `Print_media` and `Online_media` have identical schemas. The statement creates a new LOB locator in table `Print_media`. It also copies the LOB data from `Online_media` to the location pointed to by the new LOB locator inserted in table `Print_media`.

### Inserting LOBs For Binds of More Than 4,000 Bytes

For guidelines on how to INSERT into a LOB when binds of more than 4,000 bytes are involved, see the following sections in [Chapter 7, "Modeling and Design"](#):

- [Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs](#) on page 7-14
- [Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion](#) on page 7-15
- [Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE](#) on page 7-16
- [Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported](#) on page 7-18
- [Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes](#) on page 7-18

### Syntax

Use the following syntax reference:

- SQL: *Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — INSERT.

### Scenario

For LOBs, one of the advantages of using 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 working tables that use those libraries, share a common structure.

The following code fragment is based on the fact that the table `Online_media` is of the same type as `Print_media` referenced by the `ad_textdocs_ntab` column of table `Print_media`. It inserts values into the library table, and then inserts this same data into `Print_media` by means of a `SELECT`.

**See Also:** *Oracle9i Sample Schemas* for a description of the PM Schema and the `Print_media` table used in this example.

### Examples

The following example is provided in SQL and applies to all the programmatic environments:

- [SQL: Inserting a Row by Selecting a LOB from Another Table](#) on page 10-22

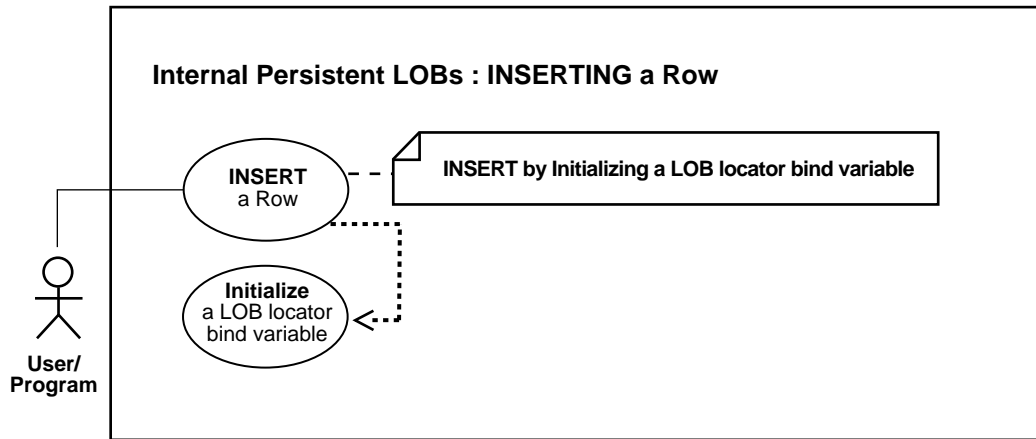
### SQL: Inserting a Row by Selecting a LOB from Another Table

```
/* Store records in the archive table Online_media: */
INSERT INTO Online_media
VALUES (3060, NULL, NULL, NULL, NULL,
       'some text about this CRT Monitor', NULL);

/* Insert values into Print_media by selecting from Online_media: */
INSERT INTO Print_media (product_id, ad_id, ad_sourcetext)
(SELECT product_id, 11001, product_text
 FROM Online_media where product_id = 3060);
```

## Inserting a Row by Initializing a LOB Locator Bind Variable

Figure 10–6 Use Case Diagram: Inserting a Row by Initializing a LOB Locator Bind Variable



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Inserting a LOB Value using EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)](#) on page 10-16
- [Inserting a Row by Selecting a LOB From Another Table](#) on page 10-20

### Purpose

This procedure inserts a row by initializing a LOB locator bind variable.

### Usage Notes

For guidelines on how to INSERT and UPDATE a LOB when binds of more than 4,000 bytes are involved, see the following sections in [Chapter 7, "Modeling and Design"](#):

- [Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs](#) on page 7-14

- [Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion](#) on page 7-15
- [Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE](#) on page 7-16
- [Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported](#) on page 7-18
- [Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes](#) on page 7-18

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- *SQL: Oracle9i SQL Reference*, "Chapter 7, SQL Statements" — INSERT
- *C (OCI): Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions.
- *C++ (OCCI): Oracle C++ Call Interface Programmer's Guide*
- *COBOL (Pro\*COBOL) Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — INSERT.
- *C/C++ (ProC/C++) Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — INSERT
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > Objects > Oradynaset
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

In the following examples you use a LOB locator bind variable to take `ad_photo` data in one row of `Print_media` and insert it into another row.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-25
- [C \(OCI\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-26
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-28
- [C/C++ \(ProC/C++\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-29
- [Visual Basic \(OO4O\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-30
- [Java \(JDBC\): Inserting a Row by Initializing a LOB Locator Bind Variable](#) on page 10-30

## PL/SQL (DBMS\_LOB Package): Inserting a Row by Initializing a LOB Locator Bind Variable

```

/* Note that the example procedure insertUseBindVariable_proc is not part of the
DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE insertUseBindVariable_proc
(productnum IN NUMBER, adnum IN NUMBER, Blob_loc IN BLOB) IS
BEGIN
    INSERT INTO Print_media (product_id, ad_id, ad_photo)
    VALUES (productnum, adnum, Blob_loc);
END;

DECLARE
    Blob_loc BLOB;
BEGIN
    /* Select the LOB from the row where product_id = 3106 and ad_id=13001.
    Initialize the LOB locator bind variable: */
    SELECT ad_photo INTO Blob_loc
    FROM Print_media
    WHERE product_id = 3106 AND ad_id=13001;
    /* Insert into the row where product_id = 2056 AND ad_id=12001 */
    insertUseBindVariable_proc (2056, 12001, Blob_loc);
    COMMIT;
END;

```

## C (OCI): Inserting a Row by Initializing a LOB Locator Bind Variable

```

/* Select the locator into a locator variable */
sb4 select_Printmedia_Locator (Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCIStmt       *stmthp;
OCISvcCtx     *svchp;
{
    OCIDefine *defnp1, *defnp2;

    text *sqlstmt =
        (text *)"SELECT ad_photo FROM Print_media WHERE product_id=2268 AND ad_
id=21001";

    /* 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));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                    (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 Print_media table 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 Print_media (product_id, ad_photo) VALUES (:2268,
:3060)";

    /* 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 Print_media table */
    select_Printmedia_Locator(Lob_loc, errhp, stmthp, svchp);

    /* Insert the locator into the Print_media table with product_id=3060 */
    product_id = 3060;

    /* 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);
```

## COBOL (Pro\*COBOL): Inserting a Row by Initializing a LOB Locator Bind Variable

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

01 BLOB1 SQL-BLOB.
01 USERID PIC X(11) VALUES "PM/PM".
   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 AD_PHOTO INTO :BLOB1
      FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268 AND AD_ID = 21001
END-EXEC.

* Insert the value with PRODUCT_ID of 3060
   EXEC SQL
      INSERT INTO PRINT_MEDIA (PRODUCT_ID, AD_PHOTO)
      VALUES (3060, 11001, :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 ROLLBACK 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.

```

## C/C++ (ProC/C++): Inserting a Row by Initializing a LOB Locator Bind Variable

```

#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, Rownum2;
    OCIBlobLocator *Lob_loc;
{
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL INSERT INTO Print_media (product_id, ad_id, ad_photo)
        VALUES (:Rownum, :Rownum2, :Lob_loc);
}

void insertBLOB_proc()
{
    OCIBlobLocator *Lob_loc;

    /* Initialize the BLOB Locator: */
    EXEC SQL ALLOCATE :Lob_loc;

    /* Select the LOB from the row where product_id = 2268 and ad_id=21001: */
    EXEC SQL SELECT ad_photo INTO :Lob_loc
        FROM Print_media WHERE product_id = 2268 AND ad_id = 21001;

    /* Insert into the row where product_id = 3106 and ad_id = 13001: */
    insertUseBindVariable_proc(3106, 13001, Lob_loc);

    /* Release resources held by the locator: */
}

```

```
EXEC SQL FREE :Lob_loc;
}

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

### Visual Basic (OO40): Inserting a Row by Initializing a LOB Locator Bind Variable

```
Dim OraDyn as OraDynaset, OraPhoto1 as OraBLOB, OraPhotoClone as OraBLOB
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id", ORADYN_DEFAULT)
Set OraPhoto1 = OraDyn.Fields("ad_photo").Value
'Clone it for future reference
Set OraPhotoClone = OraPhoto1

'Go to Next row
OraDyn.MoveNext

'Lets update the current row and set the LOB to OraPhotoClone
OraDyn.Edit
Set OraPhoto1 = OraPhotoClone
OraDyn.Update
```

### Java (JDBC): Inserting a Row by Initializing a LOB Locator Bind Variable

```
// 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
```

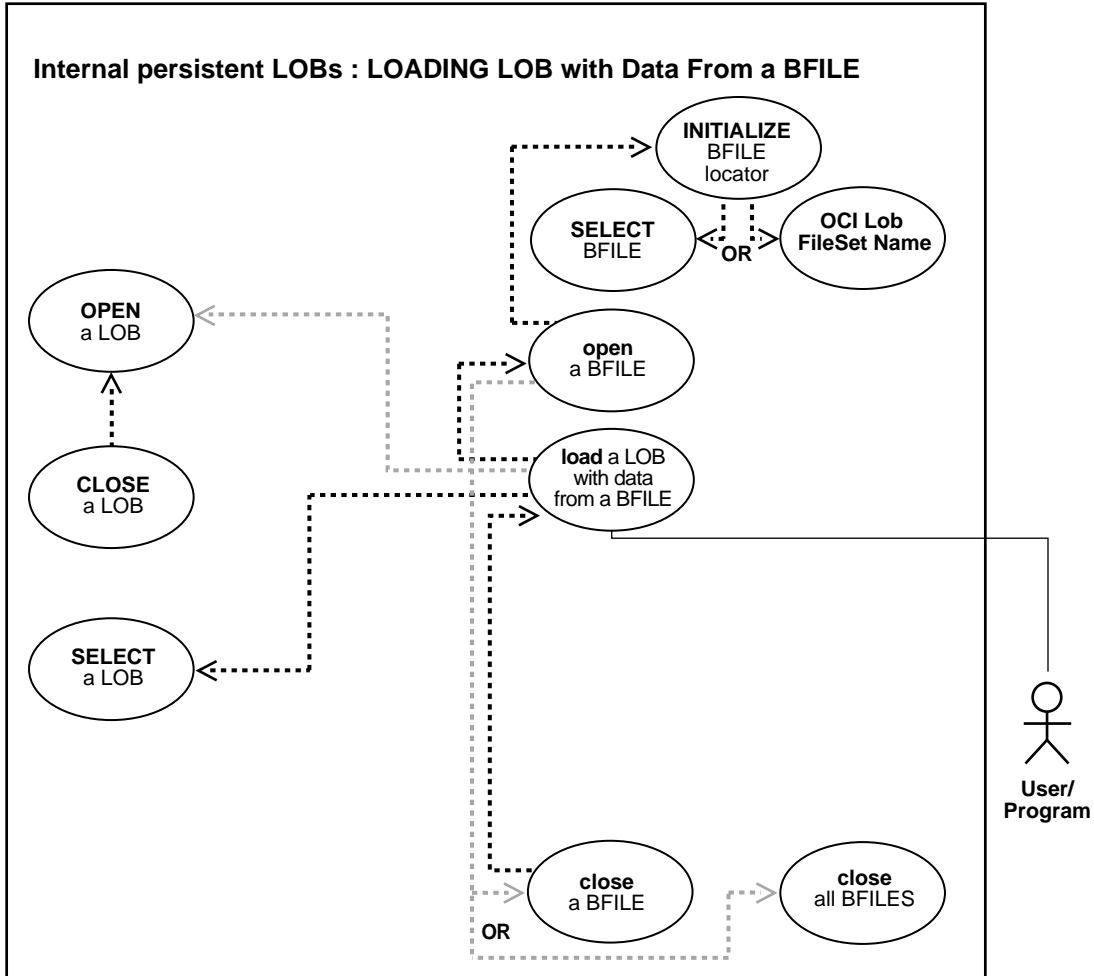
```
{
public static void main (String args [])
    throws Exception
{
    // Load the Oracle JDBC driver
    DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
    // Connect to the database:
    Connection conn =
        DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");

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

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
        ResultSet rset = stmt.executeQuery (
            "SELECT ad_photo FROM Print_media WHERE product_id = 3106 AND ad_id =
13001");
        if (rset.next())
        {
            // retrieve the LOB locator from the ResultSet
            BLOB adphoto_blob = ((OracleResultSet)rset).getBLOB (1);
            OraclePreparedStatement ops =
                (OraclePreparedStatement) conn.prepareStatement(
                    "INSERT INTO Print_media (product_id, ad_id, ad_photo) VALUES
(2268, 21001, ?)");
            ops.setBlob(1, adphoto_blob);
            ops.execute();
            conn.commit();
            conn.close();
        }
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

## Loading a LOB with Data From a BFILE

Figure 10-7 Use Case Diagram: Loading a LOB with Data From a BFILE



**See Also:**

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- ["Loading an Internal Persistent BLOB with Binary Data from a BFILE"](#) on page 10-43
- ["Loading an Internal Persistent CLOB with BFILE Data"](#) on page 10-47

**Purpose**

This procedure describes how to load a LOB with data from a BFILE.

**Usage Notes**

---

---

**Note:** The `LOADBLOBFROMFILE` and `LOADCLOBFROMFILE` procedures implement the functionality of this procedure and provide improved features for loading binary data and character data. The improved procedures are available in the PL/SQL environment only. When possible, using one of the improved procedures is recommended. See ["Loading an Internal Persistent BLOB with Binary Data from a BFILE"](#) on page 10-43 and ["Loading an Internal Persistent CLOB with BFILE Data"](#) on page 10-47 for more information.

---

---

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. LOB data instances can be in predetermined size fields, delimited fields, or length-value pair fields.

For detailed information and tips on using SQL Loader for loading data into an internal LOB see ["Loading Inline LOB Data"](#) and ["Loading Out-Of-Line LOB Data"](#) in [Chapter 4](#) of this guide.

**Binary Data to Character Set Conversion is Needed on BFILE Data**

In using 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. When you use the `DBMS_LOB.LOADFROMFILE` procedure to populate a CLOB or NCLOB, you are populating the LOB with binary data from the BFILE. No implicit translation is performed from binary data to a character set. For this reason, you should use the `LOADCLOBFROMFILE` procedure when loading text (see [Loading an Internal Persistent CLOB with BFILE Data](#) on page 10-47).

### Specify Amount to be Less than the Size of BFILE

Unless you specify `maxlobsize` to load the entire BFILE, the amount (the number of bytes) you specify to load from a BFILE must be less than or equal to the size of BFILE as follows:

- **DBMS\_LOB.LOADFROMFILE:** You cannot specify an amount larger than the size of the BFILE.
- **OciLobLoadFromFile:** You cannot specify an amount larger than the length of the BFILE.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB Package):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADFROMFILE`.
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — `LOB Functions`, `OciLobLoadFromFile`.
- **C++ (OCCI):** *Oracle C++ Call Interface Programmer's Guide*
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — `LOB LOAD`, `LOB OPEN`, `LOB CLOSE`.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — `LOB LOAD`
- **Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help):** From help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBfile > METHODS > CopyFromBFILE and select OO4O Automation Server > OBJECTS > OraDynaset, OraDatabase, OraConnection
- **Java (JDBC):** *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i*

*SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

The examples assume that there is an operating system source file `keyboard_3106_13001` that contains LOB data to be loaded into the target LOB `ad_composite`. The examples also assume that directory object `ADVERT_DIR` already exists and is mapped to the location of the source file.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Loading a LOB with Data from a BFILE on page 10-35](#)
- [C \(OCI\): Loading a LOB with Data from a BFILE on page 10-36](#)
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Loading a LOB with Data from a BFILE on page 10-38](#)
- C/C++ (ProC/C++): No example is provided with this release.
- [Visual Basic \(OO4O\): Loading a LOB with Data from a BFILE on page 10-39](#)
- [Java \(JDBC\): Loading a LOB with Data from a BFILE on page 10-40](#)

## PL/SQL (DBMS\_LOB Package): Loading a LOB with Data from a BFILE

```

/* Loading a LOB with Data from a BFILE. 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('ADPHOTO_DIR', 'keyboard_3106_13001');
  Amount        INTEGER := 4000;
BEGIN
  SELECT ad_photo INTO Dest_loc FROM print_media
     WHERE product_id = 3106 and ad_id=13001 FOR UPDATE;

  /* Opening the source 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(Dest_loc);
DBMS_LOB.CLOSE(Src_loc);
COMMIT;
END;

```

## C (OCI): Loading a LOB with Data from a BFILE

```

/* Selecting a BLOB from Print_media and loading it with data from a BFILE */
sb4 select_lock_adphoto_locator_3(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT ad_photo FROM Print_media WHERE product_id=2056
                AND ad_id = 12001 FOR UPDATE";
    OCIDefine *defnp1, *defnp2;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NIV_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)
              OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                    (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 descriptors*/
  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 ad_photo locator for update */
  printf (" select the ad_photo locator...\n");
  select_lock_adphoto_locator_2056(blob, errhp, svchp, stmthp);

  /* Set the Directory Alias and File Name of the ad_photo file */
  printf (" set the file name in bfile\n");
  checkerr (errhp, OCILobFileSetName(envhp, errhp, &bfile, (text*)"ADPHOTO_DIR",
                                    (ub2)strlen("ADPHOTO_DIR"),
                                    (text*)"mousepad_2056_12001",
                                    (ub2)strlen("mousepad_2056_12001")));

  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 graphic 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;
}

```

**COBOL (Pro\*COBOL): Loading a LOB with Data from a BFILE**

```

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 "PM/PM".
                   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 "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
                   MOVE 9 TO DIR-ALIAS-LEN.
                   MOVE "keyboard_3106_13001" 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 AD_GRAPHIC INTO :DEST
                   FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 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 ROLLBACK 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.

```

## Visual Basic (OO40): Loading a LOB with Data from a BFILE

```

Dim OraDyn as OraDynaset, OraPhoto1 as OraBLOB, OraMyBfile as OraBFile

OraConnection.BeginTrans
Set OraDyn = OraDb.CreateDynaset(
  "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraPhoto1 = OraDyn.Fields("ad_photo").Value

OraDb.Parameters.Add "id", 3060,ORAPARAM_INPUT
OraDb.Parameters.Add "mybfile", Null,ORAPARAM_OUTPUT
OraDb.Parameters("mybfile").serverType = ORATYPE_BFILE

OraDb.ExecutesSQL ("begin GetBFile(:id, :mybfile); end;")

Set OraMyBFile = OraDb.Parameters("mybfile").Value
'Go to Next row
OraDyn.MoveNext

```

```
OraDyn.Edit
'Lets update OraPhoto1 data with that from the BFILE
OraPhoto1.CopyFromBFile OraMyBFile
OraDyn.Update

OraConnection.CommitTrans
```

## Java (JDBC): Loading a LOB with Data from a BFILE

```
// 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:
        DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver ());
        // Connect to the database:
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");
        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('ADPHOTO_DIR', 'keyboard_3106_13001') FROM DUAL");
if (rset.next())
{
    src_lob = ((OracleResultSet)rset).getBFILE (1);
    src_lob.openFile();
    in = src_lob.getBinaryStream();
}

    rset = stmt.executeQuery (
        "SELECT ad_photo FROM Print_media WHERE product_id = 3106
        AND AD_ID = 13001 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);
}

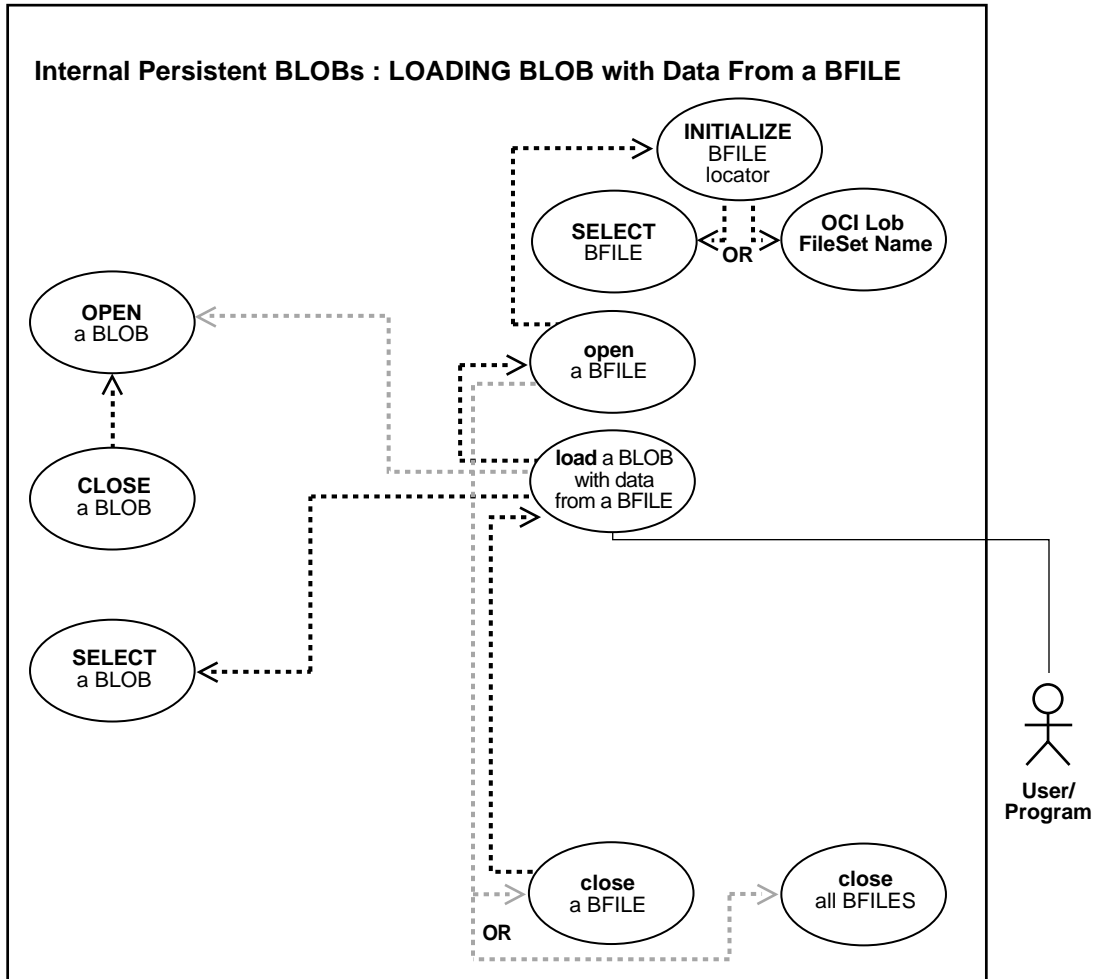
// Close all streams and file handles:
in.close();
out.flush();
out.close();
src_lob.closeFile();

// Commit the transaction:
conn.commit();
conn.close();
```

```
    }  
    catch (SQLException e)  
    {  
        e.printStackTrace();  
    }  
}
```

# Loading an Internal Persistent BLOB with Binary Data from a BFILE

Figure 10–8 Use Case Diagram: Loading a BLOB with Data from a BFILE



**See Also:**

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- ["Loading a LOB with Data From a BFILE"](#) on page 10-32
- ["Loading an Internal Persistent CLOB with BFILE Data"](#) on page 10-47

## Purpose

This procedure describes how to load an internal persistent BLOB with binary data from a BFILE.

## Usage Notes

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for text loading. This achieves the same outcome as `LOADFROMFILE` as well as returning the new offsets to the user. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

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. LOB data instances can be in predetermined size fields, delimited fields, or length-value pair fields.

For detailed information and tips on using SQL Loader for loading data into an internal LOB see ["Loading Inline LOB Data"](#) and ["Loading Out-Of-Line LOB Data"](#) in [Chapter 4](#) of this guide.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADBLOBFROMFILE`



## Scenario

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the binary LOB data to be loaded into the target BLOB.

## Example

The example, "[PL/SQL \(DBMS\\_LOB Package\): Loading an Internal Persistent BLOB with BFILE Data](#)", illustrates `LOADBLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

## PL/SQL (DBMS\_LOB Package): Loading an Internal Persistent BLOB with BFILE Data

The following example illustrates:

- How to use `LOADBLOBFROMFILE` to load the entire file without getting its length first.
- How to use the return value of the offsets to calculate the actual amount loaded.

```

DECLARE
    src_loc      BFILE := bfilename('ADVERT_DIR','display_ad_frame') ;
    dst_loc      BLOB;
    src_offset   NUMBER := 1;
    dst_offset   NUMBER := 1;
    src_osin     NUMBER;
    dst_osin     NUMBER;
    bytes_rd     NUMBER;
    bytes_wt     NUMBER;
BEGIN
    SELECT ad_composite INTO dst_loc FROM Print_media
           WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

    /* Opening the source BFILE is mandatory */
    dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

    /* Opening the LOB is optional */
    dbms_lob.OPEN(dst_loc, dbms_lob.lob_readwrite);
    /* Save the input source/destination offsets */
    src_osin := src_offset;
    dst_osin := dst_offset;
    /* Use LOEMAXSIZE to indicate loading the entire BFILE */
    dbms_lob.LOABLOBFROMFILE(dst_loc,src_loc,dbms_lob.lobmaxsize,src_offset,dst_

```

```
offset) ;

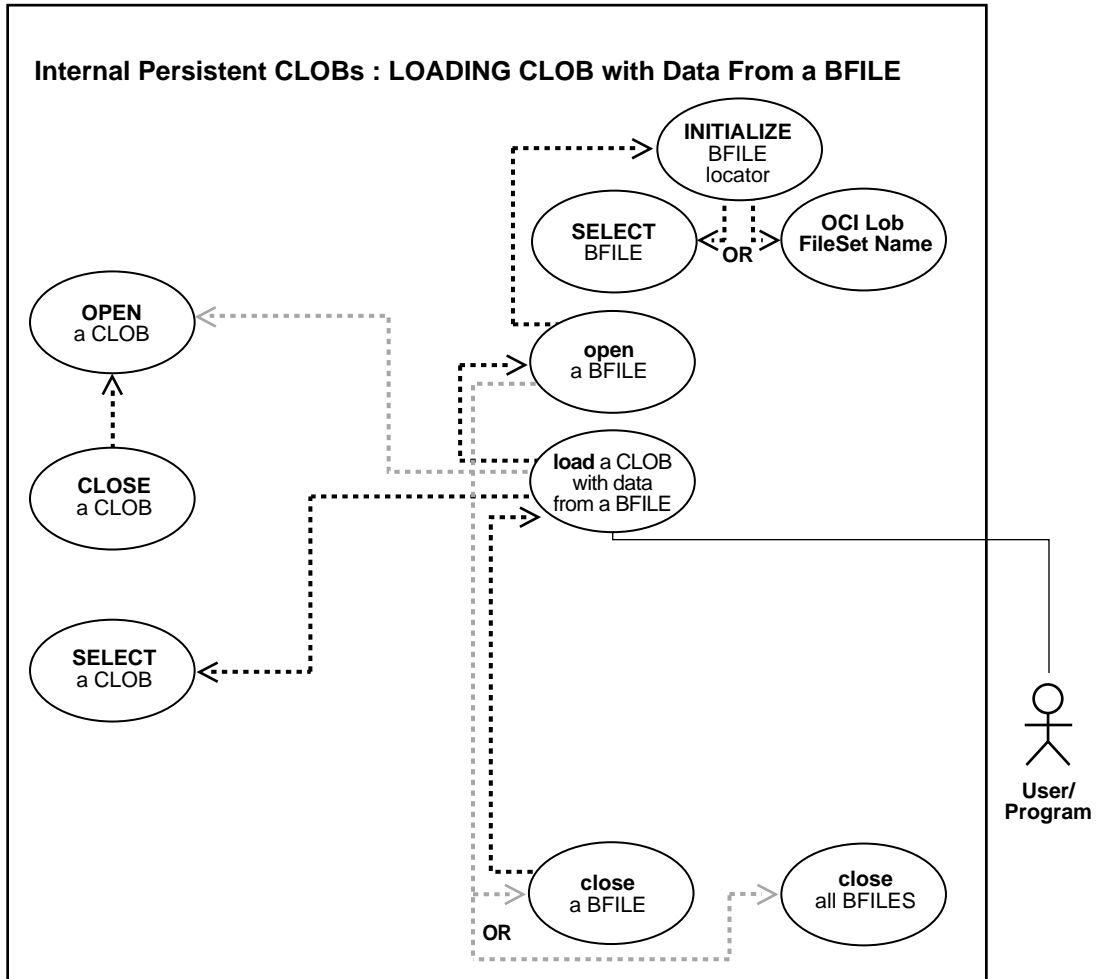
/* Closing the LOB is mandatory if you have opened it */
dbms_lob.close(dst_loc);
dbms_lob.filecloseall();
COMMIT;

/* Use the src_offset returned to calculate the actual amount read from the
BFILE */
bytes_rd := src_offset - src_osin;
dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd ) ;
/* Use the dst_offset returned to calculate the actual amount written to the
BLOB */
bytes_wt := dst_offset - dst_osin;
dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt ) ;
/* If there is no exception the number of bytes read should equal to the
number of bytes written */

END ;
```

## Loading an Internal Persistent CLOB with BFILE Data

Figure 10–9 Use Case Diagram: Loading a CLOB with BFILE Data



**See Also:**

- [Table 10-1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- ["Loading a LOB with Data From a BFILE"](#) on page 10-32
- ["Loading an Internal Persistent BLOB with Binary Data from a BFILE"](#) on page 10-43

**Purpose**

This procedure describes how to load an internal persistent CLOB or NCLOB with character data from a BFILE.

**Usage Notes**

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for loading of text as the latter method lets you specify the character set id of the BFILE. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

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. LOB data instances can be in predetermined size fields, delimited fields, or length-value pair fields.

For detailed information and tips on using SQL Loader for loading data into an internal LOB see ["Loading Inline LOB Data"](#) and ["Loading Out-Of-Line LOB Data"](#) in [Chapter 4](#) of this guide.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADCLOBFROMFILE`

## Scenario

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the character LOB data to be loaded into the target CLOB or NCLOB.

## Examples

The examples given in, [PL/SQL \(DBMS\\_LOB Package\): Loading an Internal Persistent CLOB with BFILE Data](#), illustrate `LOADCLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

## PL/SQL (DBMS\_LOB Package): Loading an Internal Persistent CLOB with BFILE Data

### Example

The following example illustrates:

- How to use default `csid` (0).
- How to load the entire file without calling `getlength` for the BFILE.
- How to find out the actual amount loaded using return offsets.

This example assumes that `ad_source` is a BFILE in UTF8 character set format and the database character set is UTF8.

```
DECLARE
  src_loc      bfile := bfilename('ADVERT_DIR','ad_source_1000') ;
  dst_loc      clob ;
  amt          number := dbms_lob.lobmaxsize;
  src_offset   number := 1 ;
  dst_offset   number := 1 ;
  lang_ctx     number := dbms_lob.default_lang_ctx;
  warning      number;
BEGIN
  select ad_sourcetext into dst_loc from Print_media
     where product_id = 3000 and ad_id = 1000 for update ;
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* The default_csid can be used when the BFILE encoding is in the same charset
   * as the destination CLOB/NCLOB charset
   */
  dbms_lob.LOADCLOBFROMFILE(dst_loc,src_loc,amt,dst_offset,src_offset,
```

```
        dbms_lob.default_csid, lang_ctx,warning) ;

commit;

dbms_output.put_line(' Amount specified ' || amt );
dbms_output.put_line(' Number of bytes read from source: ' ||
    (src_offset-1));
dbms_output.put_line(' Number of characters written to destination: ' ||
    (dst_offset-1) );
if (warning = dbms_lob.warn_inconvertible_char)
then
    dbms_output.put_line('Warning: Inconvertible character');
end if;

dbms_lob.filecloseall() ;
END ;
```

## Example

The following example illustrates:

- How to get the character set ID from the character set name using the NLS\_CHARSET\_ID function.
- How to load a stream of data from a single BFILE into different LOBs using the returned offset value and the language context lang\_ctx.
- How to read a warning message.

This example assumes that *ad\_file\_ext\_01* is a BFILE in JA16TSTSET format and the database national character set is AL16UTF16.

```
DECLARE
    src_loc      bfile := bfilename('ADVERT_DIR','ad_file_ext_01') ;
    dst_loc1     nclob;
    dst_loc2     nclob;
    amt          number := 1000;
    src_offset   number := 1;
    dst_offset   number := 1;
    src_osin     number;
    cs_id        number := NLS_CHARSET_ID('JA16TSTSET'); /* 998 */
    lang_ctx     number := dbms_lob.default_lang_ctx;
    warning      number;
BEGIN
    dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);
    dbms_output.put_line(' BFILE csid is ' || cs_id) ;
```

```

/* Load the first 1KB of the BFILE into dst_loc1 */
dbms_output.put_line(' -----' );
dbms_output.put_line('   First load   ' );
dbms_output.put_line(' -----' );

SELECT ad_flgtextn INTO dst_loc1 FROM Print_media
       WHERE product_id=3106 and ad_id=13000 FOR UPDATE;

dbms_lob.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
                          cs_id, lang_ctx, warning);
commit;

/* the number bytes read may or may not be 1k */
dbms_output.put_line(' Amount specified ' || amt );
dbms_output.put_line(' Number of bytes read from source: ' ||
                      (src_offset-1));
dbms_output.put_line(' Number of characters written to destination: ' ||
                      (dst_offset-1) );
if (warning = dbms_lob.warn_inconvertible_char)
then
  dbms_output.put_line('Warning: Inconvertible character');
end if;

/* load the next 1KB of the BFILE into the dst_loc2 */
dbms_output.put_line(' -----' );
dbms_output.put_line('   Second load   ' );
dbms_output.put_line(' -----' );

SELECT ad_flgtextn INTO dst_loc2 FROM Print_media
       WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

/* Notice we are using the src_offset and lang_ctx returned from the previous
 * load. We do not use value 1001 as the src_offset here because sometimes the
 * actual amount read may not be the same as the amount specified.
 */

src_osin := src_offset;
dst_offset := 1;
dbms_lob.LOADCLOBFROMFILE(dst_loc2, src_loc, amt, dst_offset, src_offset,
                          cs_id, lang_ctx, warning);
commit ;
dbms_output.put_line(' Number of bytes read from source: ' ||
                      (src_offset-src_osin) );
dbms_output.put_line(' Number of characters written to destination: ' ||

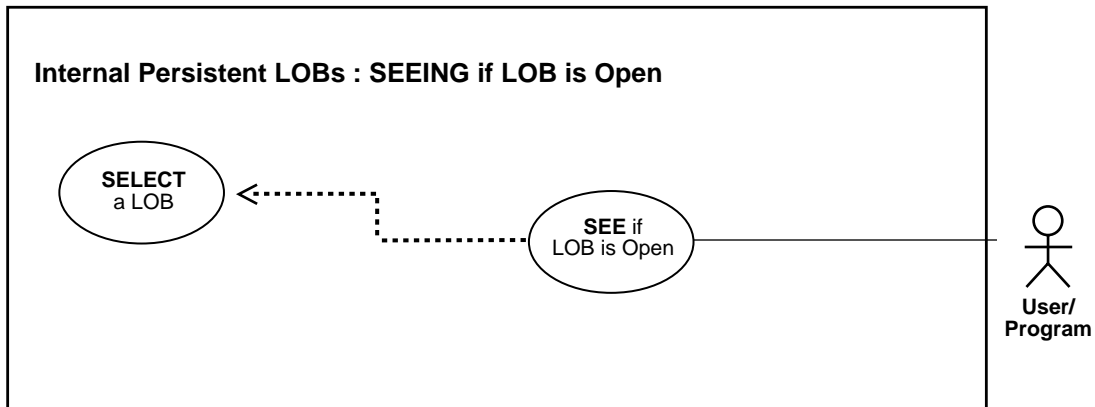
```

```
        (dst_offset-1) );  
if (warning = dbms_lob.warn_inconvertible_char)  
then  
    dbms_output.put_line('Warning: Inconvertible character');  
end if;  
  
dbms_lob.filecloseall() ;  
  
END ;
```



## Open: Checking If a LOB Is Open

Figure 10–10 Use Case Diagram: Checking If a LOB Is Open



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to check if LOB is open.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, ISOPEN.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions — OciSeeIfLob IsOpen.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*

- COBOL (Pro\*COBOL): *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE.
- C/C++ (ProC/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE ... ISOPEN ...
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

The following "Checking if a LOB is Open" examples open a graphic image (`ad_composite`), and then evaluate it to see if the LOB is open.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Checking if a LOB is Open](#) on page 10-54
- [C \(OCI\): Checking if a LOB is Open](#) on page 10-55
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Checking if a LOB is Open](#) on page 10-56
- [C/C++ \(ProC/C++\): Checking if a LOB is Open](#) on page 10-58
- Visual Basic (OO4O): No example is provided with this release.
- [Java \(JDBC\): Checking if a LOB is Open](#) on page 10-59

### PL/SQL (DBMS\_LOB Package): Checking if a LOB is Open

```
/* Checking if a LOB is Open. 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 ad_composite INTO Lob_loc FROM Print_media
  WHERE product_id = 3106 AND ad_id = 13001;
```

```

/* 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;

```

## C (OCI): Checking if a LOB is Open

```

/* Checking if LOB is Open. */
/* Select the locator into a locator variable */
sb4 select_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text      *sqlstmt =
        (text *) "SELECT ad_composite FROM Print_media
                WHERE product_id=2268 AND ad_id = 21001";
    OCIDefine *defnp1 *defnp2;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NIV_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)
            OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                    (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_adcomp_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;
}
```

## COBOL (Pro\*COBOL): Checking if a LOB is Open

```
* Checking if LOB is Open
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 "PM/PM".
                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 AD_COMPOSITE INTO :BLOB1
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
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 ROLLBACK 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.

```

## C/C++ (ProC/C++): Checking if a LOB is Open

```
/* Checking if LOB is open */
#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 ad_composite INTO :Lob_loc
        FROM Print_media WHERE product_id = 3106 and ad_id = 13001;
    /* 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 = "pm/pm";
    EXEC SQL CONNECT :pm;
    seeIfLOBIsOpen();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Java (JDBC): Checking if a LOB is Open

### Checking if a CLOB is Open

To see if a CLOB is open, your JDBC application can now use the `isOpen` method defined in `oracle.sql.CLOB`. The return boolean value indicates whether the CLOB has been previously opened or not. The `isOpen` method is defined as follows:

```
/**
 * Check whether the CLOB is opened.
 * @return true if the LOB is opened.
 */
public boolean isOpen () throws SQLException
```

The usage example is:

```
CLOB clob = ...
// See if the CLOB is opened
boolean isOpen = clob.isOpen ();
```

### Checking If a BLOB is Open

To see if a BLOB is open, your JDBC application can now use the `isOpen` method defined in `oracle.sql.BLOB`. The return boolean value indicates whether the BLOB has been previously opened or not. The `isOpen` method is defined as follows:

```
/**
 * Check whether the BLOB is opened.
 * @return true if the LOB is opened.
 */
public boolean isOpen () throws SQLException
```

The usage example is:

```
BLOB blob = ...
// See if the BLOB is opened
boolean isOpen = blob.isOpen ();
```

### Example

```
// Checking if LOB is open
// Core JDBC classes:
```

```
import java.io.OutputStream;
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:
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        // Connect to the database:
        Connection conn =
            DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");
        // 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 ad_composite FROM Print_media product_id = 3060 AND ad_id =
11001");
            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.prepareStatement (
        "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
cstmt2.setBLOB(1, blob);
cstmt2.execute();

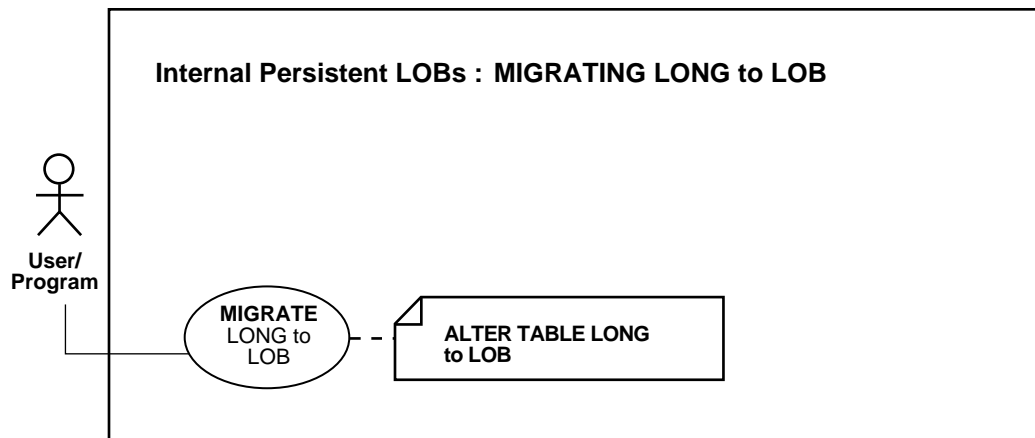
System.out.println("The LOB has been opened with a call to DBMS_LOB.OPEN()");

// 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();
}
}
```

## LONG to LOB Migration Using the LONG-to-LOB API

Figure 10–11 Use Case Diagram: Migrating LONGs to LOBs Using the (new) LONG-to-LOB API



**See Also:**

- [Table 10-1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- ["LONG to LOB Copying, Using the TO\\_LOB Operator"](#) on page 10-66

**Purpose**

This procedure describes how to migrate LONGs to LOBs using the (new) LONG-to-LOB API.

**Usage Notes**

**See Also:** [Chapter 8, "Migrating From LONGs to LOBs"](#) for further details on using the LONG-to-LOB API.

**Syntax**

Use the following syntax reference:

- **SQL:** *Oracle9i SQL Reference*, Chapter 4, "Functions"

## Scenario

The fields used in the following example are:

```
CREATE TABLE Print_media (
    product_id    NUMBER NOT NULL,
    ad_id         NUMBER NOT NULL,
    ad_sourcetext CLOB default EMPTY_CLOB(),
    ad_flgtextn   NCLOB default EMPTY_CLOB(),
    ad_graphic    BFILE default NULL,
    ad_composite  BLOB default EMPTY_BLOB(),
    ad_photo      BLOB default EMPTY_BLOB(),
    ad_graphic    BFILE default NULL,
);
```

The following example assumes that the column, `ad_sourcetext`, of table `PRINT_MEDIA` has been created as follows:

```
CREATE TABLE Print_media (
    ...
    ad_sourcetext LONG,
    ...
);
```

## To Convert LONG to CLOB, Use ALTER TABLE

To convert the LONG column to CLOB just use `ALTER TABLE` as follows:

```
ALTER TABLE Print_media MODIFY ( ad_sourcetext CLOB );
```

Any existing application using table `Print_media` can continue to work with minor modification even after the column `ad_sourcetext` has been converted to type `CLOB`. [Chapter 8, "Migrating From LONGs to LOBs"](#) provides examples of operations (binds and defines) used by LONGs and that continue to work for LOBs with minor modifications.

## Example

The following example illustrates how to use the LONG-to-LOB API with OCI:

- [C \(OCI\): LONG to LOB Migration](#)

## C (OCI): LONG to LOB Migration

```
/* Migrating from LONG to LOB Using LONG-to-LOB API */

word buflen, pid, aid;
```

```
text buf2[5000];
text *insstmt = (text *)
"INSERT INTO Print_media(product_id, ad_id, ad_sourcetext) VALUES
(:PID, :AID, :ADSOURCE)";

if (OCIStmtPrepare(stmthp, errhp, insstmt,
(ub4)strlen((char *)insstmt), (ub4) OCI_NTV_SYNTAX,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtPrepare()\n");
    report_error(errhp);
    return;
}

if (OCIBindByName(stmthp, &bndhp[0], errhp,
(text *) ":PID", (sb4) strlen((char *) ":PID"),
(dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 0,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
|| OCIBindByName(stmthp, &bndhp[1], errhp,
(text *) ":AID", (sb4) strlen((char *) ":AID"),
(dvoid *) &buf1, (sb4) sizeof(buf1), SQLT_INT,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 1,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT)
|| OCIBindByName(stmthp, &bndhp[2], errhp,
(text *) ":ADSOURCE", (sb4) strlen((char *) ":ADSOURCE"),
(dvoid *) buf2, (sb4) sizeof(buf2), SQLT_CHR,
(dvoid *) 0, (ub2 *) 0, (ub2 *) 3,
(ub4) 0, (ub4 *) 0, (ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIBindByName()\n");
    report_error(errhp);
    return;
}

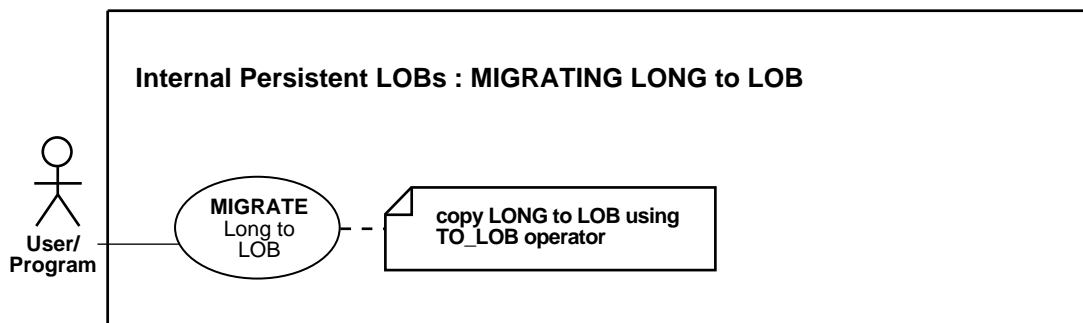
buf1 = 101;
memset((void *)buf2, (int)'A', (size_t)5000);

if (OCIStmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
(const OCISnapshot*) 0, (OCISnapshot*) 0,
(ub4) OCI_DEFAULT))
{
    DISCARD printf("FAILED: OCIStmtExecute()\n");
    report_error(errhp);
    return;
}
```

}

## LONG to LOB Copying, Using the TO\_LOB Operator

Figure 10–12 Use Case Diagram: Copying LONGs to LOBs Using TO\_LOB Operator



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- ["LONG to LOB Migration Using the LONG-to-LOB API"](#) on page 10-62

### Purpose

This procedure describes how to copy a LONG to a LOB using the TO\_LOB operator.

### Usage Notes

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;
  
```

- The TO\_LOB operator cannot be used in the CREATE TABLE AS SELECT statement to convert a LONG or LONG RAW column to a LOB column when creating an Index Organized Table.

The workaround is to create the Index Organized Table, and then do an INSERT AS SELECT of the LONG or LONG RAW column using the TO\_LOB operator.

- 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.
- The TO\_LOB function can be used to copy data to a CLOB but not a NCLOB. This is because LONG datatypes have the database CHAR character set and can only be converted to a CLOB which also uses the database CHAR character set. NCLOB on the other hand, use the database NCHAR character set.

## Syntax

Use the following syntax reference:

- [SQL: Oracle9i SQL Reference](#), Chapter 4, "Functions" — TO\_LOB.

## Scenario

Assume that the following archival source table adlibrary\_tab was defined and contains data:

```
Rem This script is not needed if you use the (newer)
Rem ALTER TABLE when migrating LONGs to LOBs
CREATE TABLE adlibrary_tab
(
  product_id  NUMBER(6),
  ad_id       NUMBER(6),
  photos      LONG RAW
);
```

This example assumes that you want to copy the data from the LONG RAW column (photos) into the BLOB column (ad\_photo) of the Print\_media table, and uses the SQL function TO\_LOB to accomplish this.

## Examples

The example is provided in SQL and applies to all programmatic environments:

- ["SQL: Copying LONGs to LOBs Using TO\\_LOB Operator"](#)

## SQL: Copying LONGs to LOBs Using TO\_LOB Operator

```
INSERT INTO Print_media (product_id, ad_id, ad_photo)
```

```
SELECT product_id, TO_LOB(photos)
FROM adlibrary_tab WHERE product_id =3106;
```

---

---

**Note:** in order for the previous example to succeed, execute:

```
CREATE TABLE adlibrary_tab (
    product_id    NUMBER,
    ad_audience  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;
```

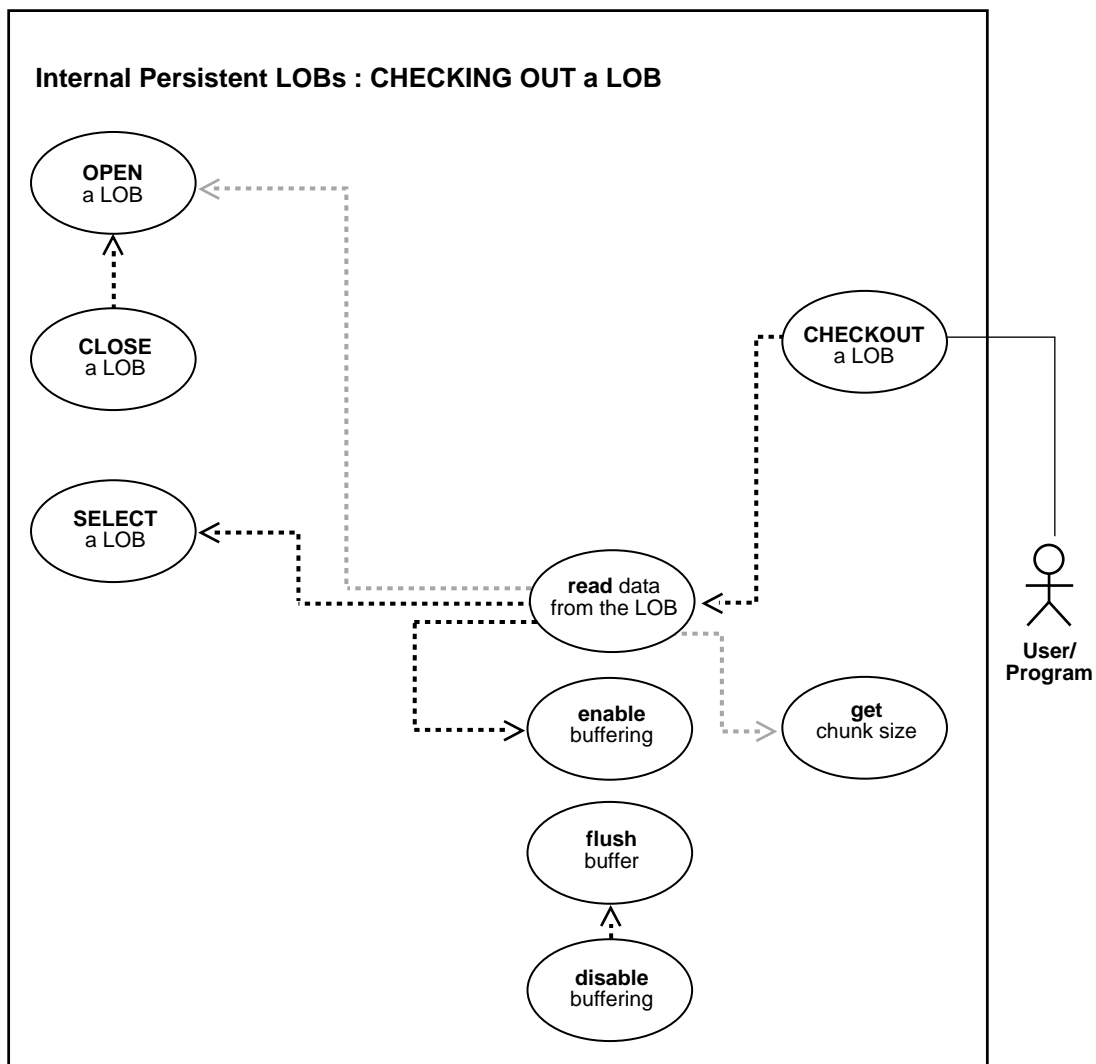
Once the copy is complete, any applications that use the table must be modified to use the LOB data.

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.

# Checking Out a LOB

Figure 10-13 Use Case Diagram: Checking Out a LOB



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to checkout a LOB.

## Usage Notes

**Streaming Mechanism** The most efficient way to read large amounts of LOB data is to use `OCILobRead()` with the streaming mechanism enabled using polling or callback. Use OCI, OCCI, or PRO\*C interfaces with streaming for the underlying read operation. Using `DBMS_LOB.READ` will result in non-optimal performance.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, READ, CLOSE.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions — OCILobOpen, OCILobRead, OCILobClose.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- C/C++ (ProC/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB OPEN, LOB READ
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orblob > METHODS > read, copytofile
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## 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 checkin all the modifications that were made to the document on the client side. The checkin portion of the scenario is described in "[Checking In a LOB](#)" on page 10-83.

## Examples

The following examples are similar to examples provided in "[Displaying LOB Data](#)". Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Checking Out a LOB](#) on page 10-73
- [C \(OCI\): Checking Out a LOB](#) on page 10-74
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Checking Out a LOB](#) on page 10-76
- [C/C++ \(ProC/C++\): Checking Out a LOB](#) on page 10-78
- [Visual Basic \(OO4O\): Checking Out a LOB](#) on page 10-80
- [Java \(JDBC\): Checking Out a LOB](#) on page 10-80

## PL/SQL (DBMS\_LOB Package): Checking Out a LOB

```

/* Checking out a LOB. The procedure checkOutLOB_proc used here is not part of
the
DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkOutLOB_proc IS
    Lob_loc      BLOB;
    Buffer        VARCHAR2(32767);
    Amount       BINARY_INTEGER := 32767;
    Position     INTEGER := 2147483647;
BEGIN
    /* Select the LOB: */
    SELECT Nesttab.formatted_doc INTO Lob_loc
        FROM TABLE(SELECT PMtab.Textdoc_ntab FROM Print_media PMtab
            WHERE PMtab.product_id = 3106 AND PMtab.ad_id = 13001) Nesttab
        WHERE Nesttab.document_typ = 'PDF';
    /* 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;
```

## C (OCI): Checking Out a LOB

/\* Checking out a LOB -- This example reads 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_formatteddoc_locator(Lob_loc, errhp, stmthp, svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT Intab.Transcript \
                FROM TABLE(SELECT pm.ad_textdoc_ntab FROM Print_media pm \
                WHERE pm.product_id = 3060 AND pm.ad_id = 11001) ntab \
                WHERE ntab.document_typ = 'PDF'";
    OCIDefine *defnp1, *defnp2;
    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)
        || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                    (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[MAXBUFLLEN];
    ub4 buflen;

    /* Allocate locators descriptors: */
    (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* Select the BLOB: */
    printf(" select the formatted_doc locator...\n");
    select_formatteddoc_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', MAXBUFLLEN);
    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;
}

```

## COBOL (Pro\*COBOL): Checking Out a LOB

```

* Checking out a LOB
IDENTIFICATION DIVISION.
PROGRAM-ID. CHECKOUT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID   PIC X(11) VALUES "PM/PM".
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 AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
      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
    ROLLBACK 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.

```

## C/C++ (ProC/C++): Checking Out a LOB

```

/* Checking out a LOB -
   This example reads 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. \
            FROM TABLE(SELECT Pmtab.textdoc_ntab FROM Print_media Pmtab \
                WHERE Pmtab.product_id = :pid AND Pmtab.ad_id = :aid) ntab \
                WHERE ntab.document_ttyp = :seg';
    EXEC SQL DECLARE C CURSOR FOR S;
    Clip_ID = Segment = 1;
    EXEC SQL OPEN C USING :PID, :AID, :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 *pm = "pm/pm";
EXEC SQL CONNECT :pm;
checkOutLOB_proc();
EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Visual Basic (OO4O): Checking Out a LOB

```
'Checking out a LOB
'There are two ways of reading a lob using orablob.read or orablob.copytofile

'Using OraBlob.Read mechanism
Dim OraDyn as OraDynaset, OraPhoto as OraBlob, amount_read%, chunksize%, chunk

chunksize = 32767
set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
set OraPhoto = OraDyn.Fields("ad_photo").Value
OraPhoto.PollingAmount = OraPhoto.Size 'Read entire BLOB contents
Do
    amount_read = OraPhoto.Read(chunk,chunksize) 'chunk returned is a variant of
type byte array
Loop Until OraPhoto.Status <> ORALOB_NEED_DATA

'Using OraBlob.CopyToFile mechanism
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value

'Read entire BLOB contents
OraPhoto.CopyToFile "c:\myphoto.jpg"
```

## Java (JDBC): Checking Out a LOB

```
// Checking out a LOB
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:
        DriverManager.registerDriver(new 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];

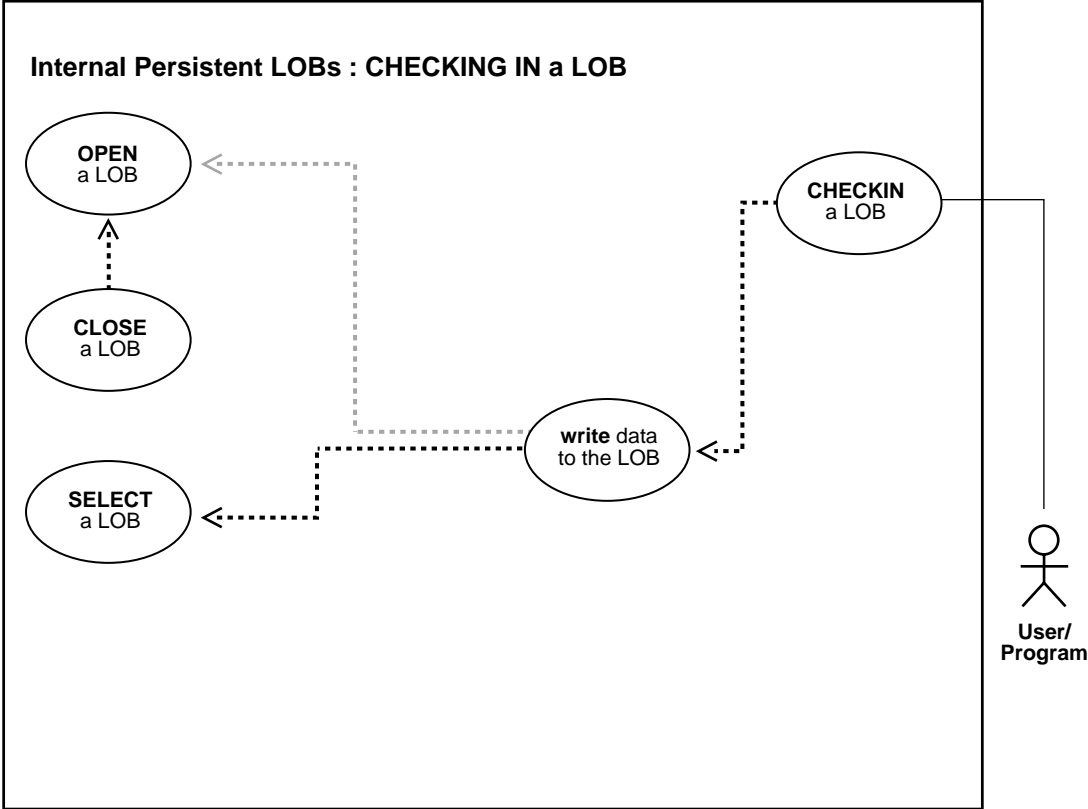
            ResultSet rset = stmt.executeQuery (
                "SELECT ntab.formatted_doc FROM TABLE( "
                +" SELECT pm.ad_textdoc_ntab FROM Print_media pm "
                +" WHERE pm.product_id=3060 AND ad_id = 11001) ntab "
                +" WHERE ntab.document_typ='html'");
            if (rset.next())
            {
                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();  
    rset.close();  
    stmt.close();  
    conn.commit();  
    conn.close();  
    }  
    catch (SQLException e)  
    {  
        e.printStackTrace();  
    }  
    }  
}
```

# Checking In a LOB

Figure 10-14 Use Case Diagram: Checking In a LOB



**See Also:** [Table 10-1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

**Purpose**

This procedure describes how to check in a LOB.

## Usage Notes

**Streaming Mechanism** The most efficient way to write large amounts of LOB data is to use `OCILobWrite()` with the streaming mechanism enabled using polling or callback

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, WRITE, CLOSE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OciLobOpen, OciLobWrite, OciLobClose
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB WRITE.
- C/C++ (ProC/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB WRITE
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > write, and > OBJECTS > Oradynaset > update
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The `checkin` operation demonstrated here follows from "[Checking Out a LOB](#)" on page 10-71. In this case, the procedure writes the data back into the `CLOB` `formatted_doc` column within the nested table `ad_textdoc_ntab` that contains interview segments. As noted earlier, you should use the OCI or PRO\*C interface with streaming for the underlying write operation; using `DBMS_LOB.WRITE` will result in non-optimal performance.



The following examples illustrate how to check in a LOB using various programmatic environments:

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Checking in a LOB](#) on page 10-85
- [C \(OCI\): Checking in a LOB](#) on page 10-86
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Checking in a LOB](#) on page 10-89
- [C/C++ \(ProC/C++\): Checking in a LOB](#) on page 10-92
- [Visual Basic \(OO4O\): Checking in a LOB](#) on page 10-94
- [Java \(JDBC\): Checking in a LOB](#) on page 10-95

## PL/SQL (DBMS\_LOB Package): Checking in a LOB

```

/* Checking in a LOB. The example procedure checkInLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE checkInLOB_proc IS
    Lob_loc      BLOB;
    Buffer        VARCHAR2(32767);
    Amount       BINARY_INTEGER := 32767;
    Position     INTEGER := 2147483647;
    i            INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT ntab.formatted_doc INTO Lob_loc
    FROM TABLE(SELECT PMtab.Textdoc_ntab FROM Print_media PMtab
                WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001) ntab
    WHERE ntab.document_typ = 'pdf' 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;
```

## C (OCI): Checking in a LOB

/\* Checking in a LOB. This example demonstrates how using OCI you can 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 holds the data being written to the LOB. \*/

```

#define MAXBUFLen 32767
/* Select the locator into a locator variable */
sb4 select_lock_formateddoc_locator(Lob_loc, errhp, stmthp,svchp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1, *defnp2;

  text *sqlstmt =
    (text *) "SELECT ntab.formatted_doc \
      FROM TABLE(SELECT pm.ad_textdoc_ntab FROM Print_media pm \
        WHERE pm.product_id = 2268 AND pm.ad_id = 21001) ntab \
        WHERE ntab.document_typ = 'PDF' 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)
    || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
    (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 descriptors*/
    (void) OCIDescriptorAlloc((dvoid *)envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB,(size_t) 0,(dvoid **) 0);
    /* Select the CLOB */
    printf(" select the formatted_doc locator...\n");
    select_lock_formatteddoc_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, &amp;ntp,
                          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 opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

```

## COBOL (Pro\*COBOL): Checking in a LOB

```

* Checking in a LOB
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 "P/PM".
01  CLOB1     SQL-CLOB.M
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 AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3060 AND AD_ID = 11001 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 ROLLBACK WORK RELEASE END-EXEC.
  STOP RUN.

SQL-ERROR.
  EXEC SQL
    WHENEVER SQLERROR CONTINUE END-EXEC.
  DISPLAY " ".
  DISPLAY "ORACLE ERROR DETECTED:".
  DISPLAY " ".
```

```

DISPLAY SQLERRM.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

```

## C/C++ (ProC/C++): Checking in a LOB

```

/* Checking in a LOB.
   This example shows how you can use Pro*C/C++ 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 holds 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 ad_sourcetext INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 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;
}

```

## Visual Basic (OO40): Checking in a LOB

```

'Checking in a LOB
'There are two ways of writing a lob - using orablob.write or
orablob.copyfromfile

'Using the OraBlob.Write mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim fnum As Integer
Dim OraDyn As OraDynaset, OraPhoto As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "pm/pm", 0&)
chunksize = 500
ReDim curchunk(chunksize)
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value

fnum = FreeFile

```

```

Open "c:\tmp\keyboard_3016_13001.jpg" For Binary As #fnum
OraPhoto.offset = 1
OraPhoto.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

Dim piece As Byte
Get #fnum, , curchunk
OraDyn.Edit
piece = ORALOB_FIRST_PIECE
OraPhoto.Write curchunk, chunksize, ORALOB_FIRST_PIECE

While OraPhoto.Status = ORALOB_NEED_DATA
    remainder = remainder - chunksize
    If remainder <= chunksize Then
        chunksize = remainder
        piece = ORALOB_LAST_PIECE
    Else
        piece = ORALOB_NEXT_PIECE
    End If

    Get #fnum, , curchunk
    OraPhoto.Write curchunk, chunksize, piece
Wend

OraDyn.Update

'Using the OraBlob.CopyFromFile mechanism
Set OraDyn = OraDb.CreateDynaset("select * from Print_media order by product_
id, ad_id", ORADYN_DEFAULT)
Set OraPhoto = OraDyn.Fields("ad_photo").Value
OraDyn.Edit
OraPhoto.CopyFromFile "c:\tmp\keyboard3016_13001.jpg"
OraDyn.Update

```

## Java (JDBC): Checking in a LOB

```

// Checking in a LOB
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:
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

        // 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 for the Ad");
            ResultSet rset = stmt.executeQuery (
                "SELECT ad_sourcetext FROM Print_media WHERE product_id = 2268
                AND ad_id = 21001 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++)
            {

```

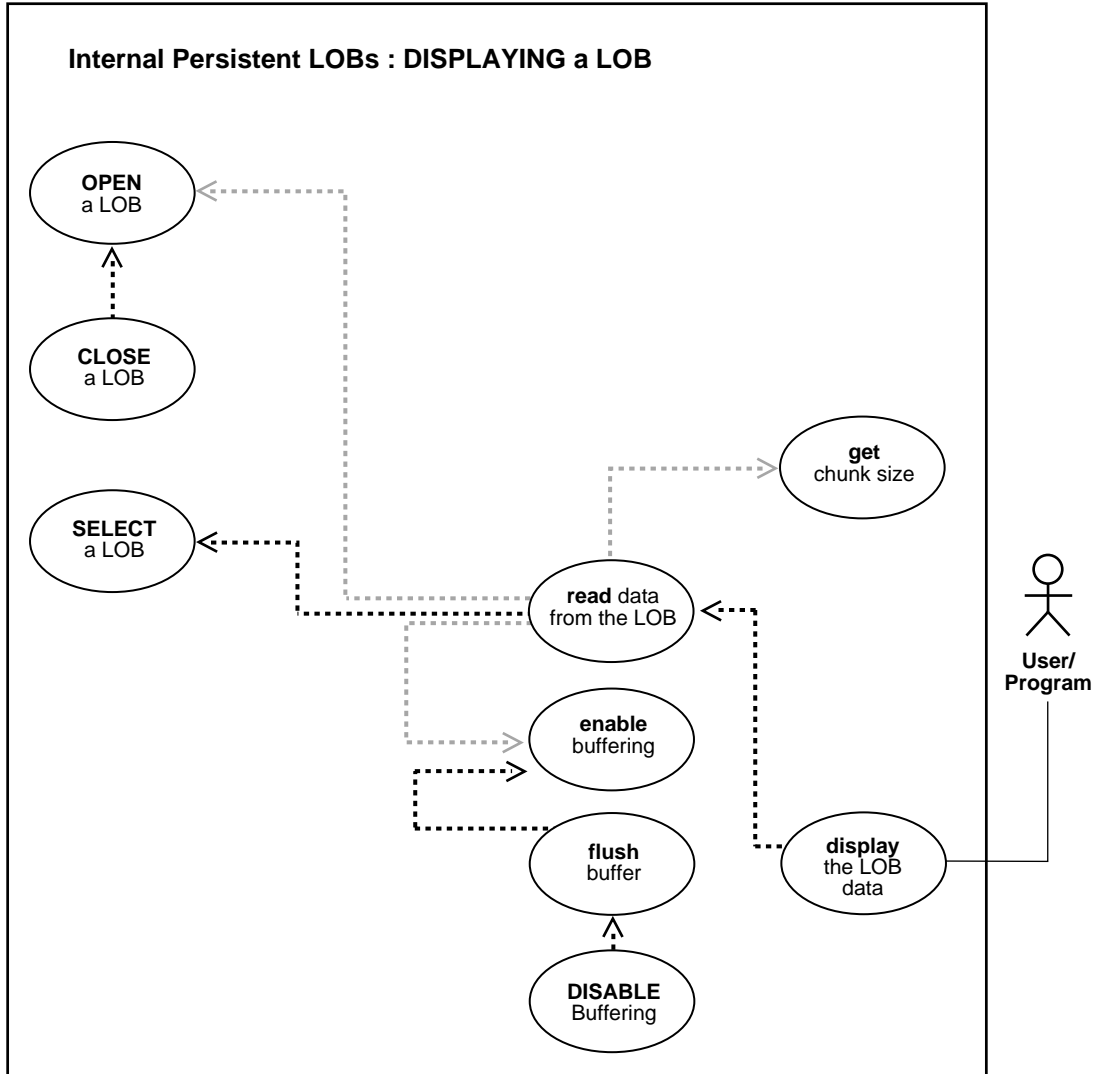
```
pos = lob_loc.length();

// an alternative is: lob_loc.putString(pos, buf);
lob_loc.putString(pos, buf);

// Some debug information:
System.out.println(" putString(" + Long.toString(pos) + " buf);");
}
stmt.close();
conn.commit();
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Displaying LOB Data

Figure 10–15 Use Case Diagram: Displaying LOB Data



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to display LOB data.

## Usage Notes

**Streaming Mechanism** The most efficient way to read large amounts of LOB data is to use `OCILobRead()` with the streaming mechanism enabled.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, READ, CLOSE.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, `OciLobOpen`, `OciLobRead`, `OciLobClose`.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- C/C++ (ProC/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB READ
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oraclob > METHODS > read, and > OBJECTS > Oraclob > PROPERTIES > offset
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

As an example of displaying a LOB, our scenario stream-reads the image `logo` from the column object `ad_header` onto the client-side in order to view the data.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Displaying LOB Data on page 10-100](#)
- [C \(OCI\): Displaying LOB Data on page 10-101](#)
- C++ (OCCI): No example is provided in this release.
- [COBOL \(Pro\\*COBOL\): Displaying LOB Data on page 10-103](#)
- [C/C++ \(ProC/C++\): Displaying LOB Data on page 10-105](#)
- [Visual Basic \(OO4O\): Displaying LOB Data on page 10-106](#)
- [Java \(JDBC\): Displaying LOB Data on page 10-107](#)

## PL/SQL (DBMS\_LOB Package): Displaying LOB Data

```

/* Displaying LOB data.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 pm.ad_header.logo INTO Lob_loc
        FROM print_media pm WHERE pm.product_id = 3060 AND pm.ad_id = 11001;
    /* 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;
```



## C (OCI): Displaying LOB Data

```

/* Displaying LOB data. This example reads the entire contents of a BLOB
   piecewise into a buffer using the standard polling method, processing
   each buffer piece after every READ operation until the entire BLOB
   has been read. */

#define MAXBUFLLEN 32767
/* Select the locator into a locator variable */
sb4 select_mapobjectdrawing_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *defnp1, *defnp2;
    text *sqlstmt =
        (text *) "SELECT pm.Adheader_type.logo \
                FROM Print_media pm WHERE pm.product_id = 3060 AND ad_id =
11001";
    checkerr (errhp, OCISstmtPrepare(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)
        || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                            (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, OCISstmtExecute(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[MAXBUFLLEN];
    ub4 buflen;
    OCILobLocator *Lob_Loc;

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

    /* Select the BLOB */
    printf(" select the adheaderlogo locator...\n");
    select_adheaderlogo_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', MAXBUFLLEN);
    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 opened 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);
}

```

## COBOL (Pro\*COBOL): Displaying LOB Data

```

* DISPLAYING LOB DATA
IDENTIFICATION DIVISION.
PROGRAM-ID. DISPLAY-BLOB.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 PM.AD_PHOTO INTO :BLOB1
FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268 AND AD_ID = 21001
END-EXEC.

DISPLAY "Found column AD_PHOTO".
* 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 ROLLBACK 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.

```

## C/C++ (ProC/C++): Displaying LOB Data

```

/* Displaying LOB data. This example reads 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.ad_header.header_text INTO Lob_loc
        FROM Print_media m WHERE m.product_id = 3060 AND ad_id = 11001;
    /* 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;
}
```

## Visual Basic (OO40): Displaying LOB Data

```
'Displaying LOB data
'Using the 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, OraAdSourceText as OraClob, amount_read%, chunksize%,
chunk
chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value
OraAdSourceText.PollingAmount = OraAdSourceText.Size 'Read entire CLOB contents
Do
```

```

'chunk returned is a variant of type byte array:
  amount_read = OraAdSourceText.Read(chunk, chunksize)
'Msgbox chunk
Loop Until OraAdSourceText.Status <> ORALOB_NEED_DATA

```

## Java (JDBC): Displaying LOB Data

```

// Core JDBC classes:
import java.io.OutputStream;
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:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

    // 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 pm.ad_header.logo FROM Print_media pm
            WHERE pm.product_id = 2056 AND ad_id = 12001");
        if (rset.next())
        {
            lob_loc = ((OracleResultSet)rset).getBLOB (1);
        }

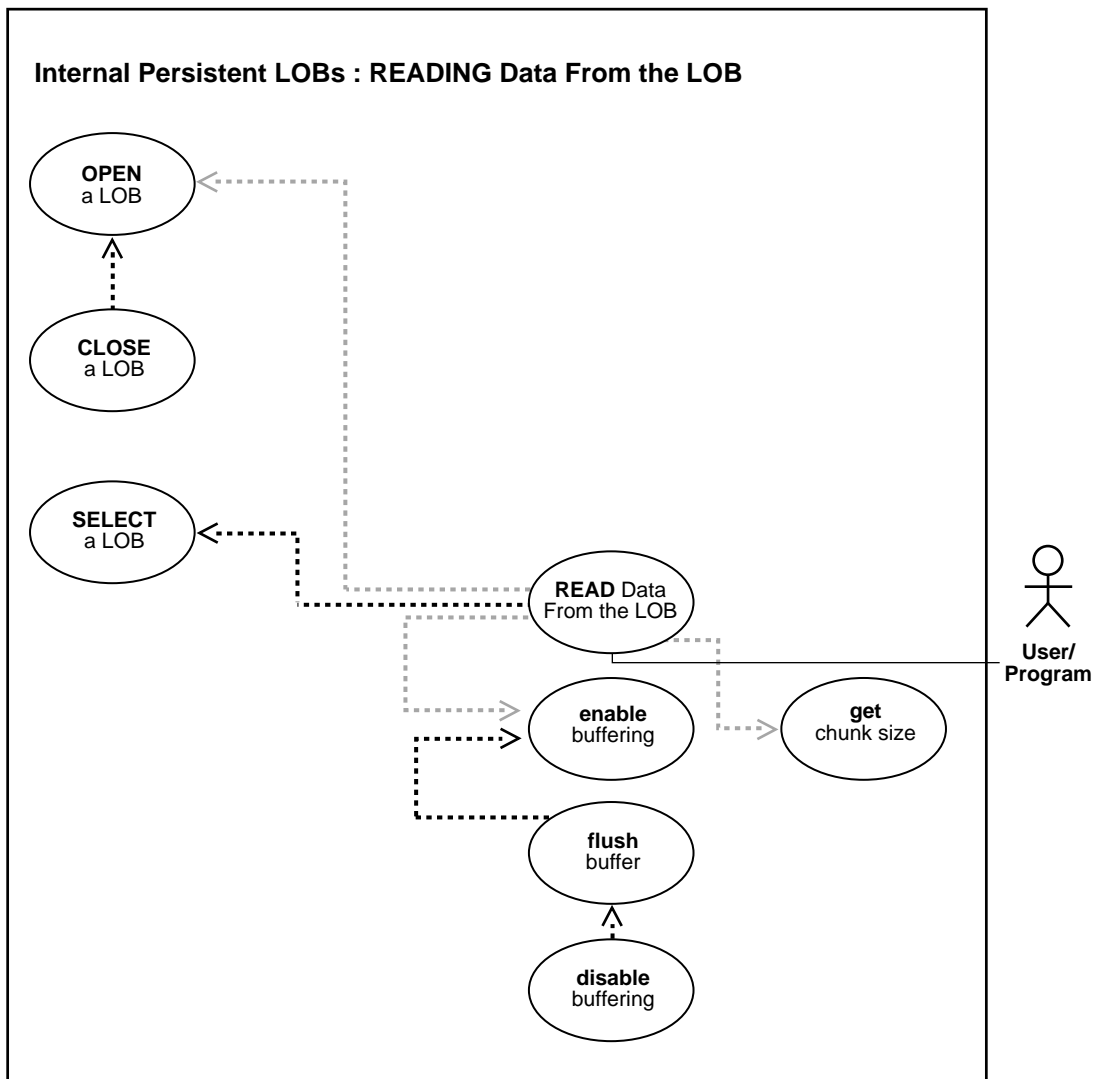
        // read this LOB through an InputStream:
        in = lob_loc.getBinaryStream();
        while ((length = in.read(buf)) != -1)
        {
            pos += length;
            System.out.println(Integer.toString(pos));
            // Process the contents of the buffer here.
        }
        in.close();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
}

```



## Reading Data from a LOB

Figure 10-16 Use Case Diagram: Reading Data from a LOB



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#)  
on page 10-2

## Procedure

This procedure describes how to read data from LOBs.

## Usage Notes

**Stream Read** The most efficient way to read large amounts of LOB data is to use `OCILobRead()` with the streaming mechanism enabled using polling or 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 - 1 regardless of the starting offset and the amount of data in the LOB. Hence, you do not need to incur a round-trip to the server to call `OCILobGetLength()` to find out the length of the LOB value to determine the amount to read.

**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, buf1, 0, 0, 0, 0)
```

---

---

### Note:

- In `DBMS_LOB.READ`, the amount can be larger than the size of the data. In PL/SQL, the amount should be less than or equal to the size of the *buffer*, and the buffer size is limited to 32K.
  - In `OCILobRead`, you can specify `amount = 4 gigabytes-1`, and it will read to the end of the LOB.
- 
- 
- 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 you execute `read` requests using a multiple of this chunk size. The reason for this is that you are 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.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, GETCHUNKSIZE, READ, CLOSE.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 7, "Chapter 15, "Relational Functions" — LOB Functions, OciLobOpen, OciLobRead, OciLobClose.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- C/C++ (ProC/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB READ
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oraclob > METHODS > read
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i*

*SQLJ Developer's Guide and Reference Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.*

### Scenario

The examples read data from a single image.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Reading Data from a LOB](#) on page 10-112
- [C \(OCI\): Reading Data from a LOB](#) on page 10-113
- [COBOL \(Pro\\*COBOL\): Reading Data from a LOB](#) on page 10-115
- [COBOL \(Pro\\*COBOL\): Reading Data from a LOB](#) on page 10-115
- [C \(OCI\): Reading Data from a LOB](#) on page 10-113
- [Visual Basic \(OO4O\): Reading Data from a LOB](#) on page 10-118
- [Java \(JDBC\): Reading Data from a LOB](#) on page 10-118

## PL/SQL (DBMS\_LOB Package): Reading Data from a LOB

```

/* Reading LOB data. 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 ad_composite INTO Lob_loc
        FROM print_media WHERE product_id = 3060 AND ad_id = 11001;
    /* 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;

```

## C (OCI): Reading Data from a LOB

```

/* Reading LOB data. This example reads 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 MAXBUFLLEN 1000

/* Select the locator into a locator variable */
sb4 select_frame_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *defnpl;

    text *sqlstmt =
        (text *) "SELECT ad_composite \
                FROM Print_media pm \
                WHERE pm.product_id = 2268";

    printf(" prepare statement in select_adcomposite_locator\n");
    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    printf(" OCIDefineByPos in select_adcomposite_locator\n");
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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(" OCISstmtExecute in select_adcomposite_locator\n");
    checkerr(errhp, OCISstmtExecute(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[MAXBUFLLEN];
    ub4 buflen;
    boolean done;

    OCILobLocator *Lob_loc;
    OCILobLocator *blob;

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

    /* Select the BLOB */
    printf(" call select_ad4read_locator\n");
    select_adcomposite_locator(Lob_loc, errhp, svchp, stmthp);

    /* Open the BLOB */
    printf(" call OCILobOpen\n");
    checkerr (errhp, OCILobOpen(svchp, errhp, Lob_loc, OCI_LOB_READONLY));

    /* Setting the amt to the buffer length. Note here that amt is in bytes
       since we are using a BLOB */
    amt = 0;
    buflen = sizeof(bufp);

    /* Process the data in pieces */
    printf(" process the data in pieces\n");
    offset = 1;
    memset(bufp, '\0', MAXBUFLLEN);
    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 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 */
    printf("[%.*s]\n", buflen, bufp);
    done = TRUE;
    break;
case OCI_ERROR:
    /* report_error();          this function is not shown here */
    done = TRUE;
    break;
case OCI_NEED_DATA:
    printf("[%.*s]\n", buflen, bufp);
    break;
default:
    (void) printf("Unexpected ERROR: OCILobRead() LOB.\n");
    done = TRUE;
    break;
}
}
/* 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);
}

```

## COBOL (Pro\*COBOL): Reading Data from a LOB

```

* READING LOB DATA
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 "PM/PM".
      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 AD_COMPOSITE INTO :BLOB1  
    FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268  
    AND AD_ID = 21001 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 ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Reading Data from a LOB

```
/* Reading LOB data  
#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 ad_composite INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
    /* 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 = "pm/pm";
    EXEC SQL CONNECT :pm;
    readLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Visual Basic (OO4O): Reading Data from a LOB

```
'Reading LOB data using the 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, OraAdSourceText as OraClob, amount_read%, chunksize%,
chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value
OraAdSourceText.pollingAmount = OraAdSourceText.Size
'Read entire CLOB contents
Do
amount_read = OraAdSourceText.Read(chunk, chunksize)
'chunk returned is a variant of type byte array
Loop Until OraAdSourceText.Status <> ORALOB_NEED_DATA
```

## Java (JDBC): Reading Data from a LOB

```
// Reading LOB data
// 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:
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

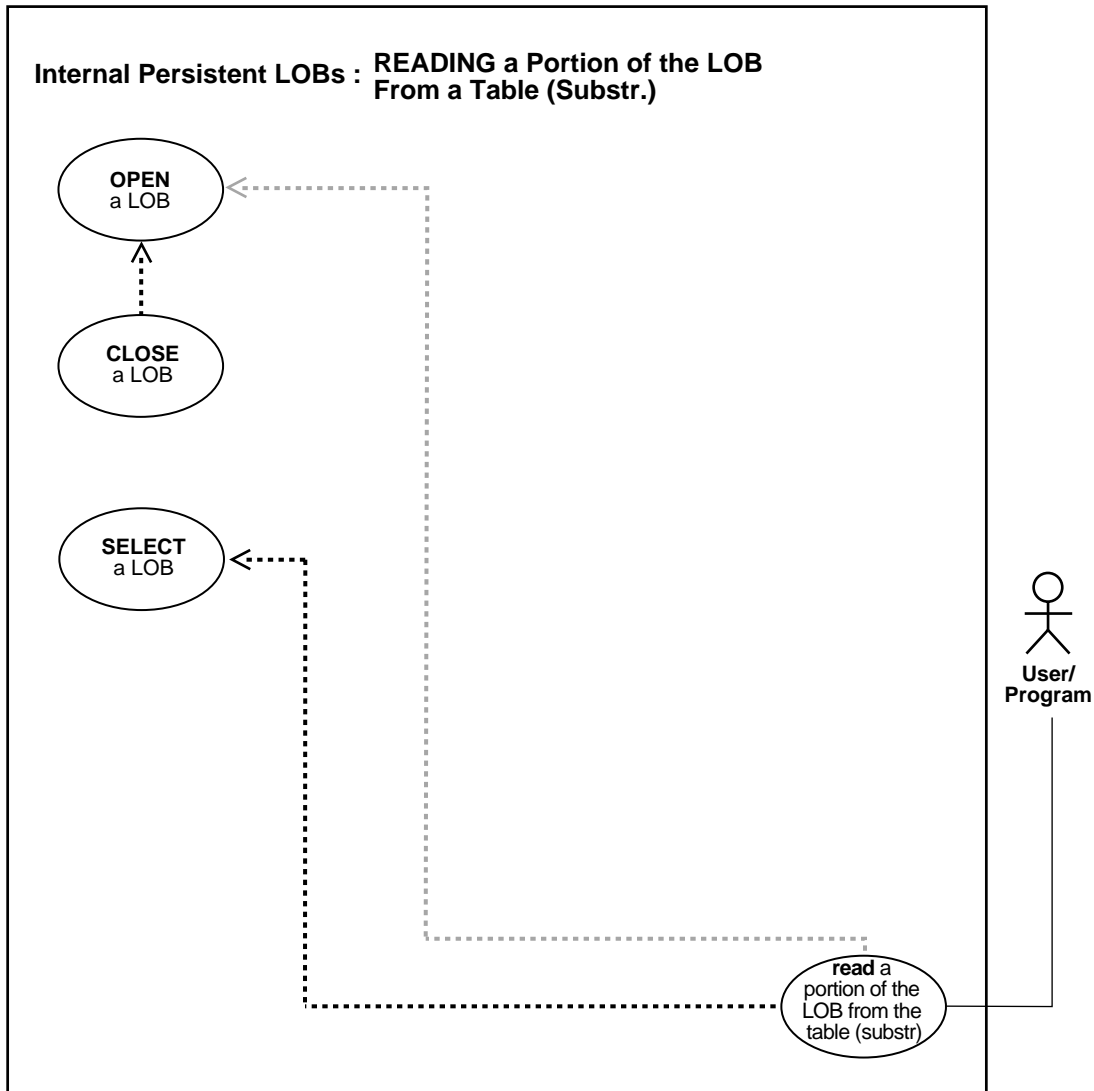
    // It's faster when auto commit is off:
    conn.setAutoCommit (false);

    // Create a Statement:
    Statement stmt = conn.createStatement ();
    try
    {
        BLOB lob_loc = null;
        byte buf[] = new byte[MAXBUFSIZE];
        ResultSet rset = stmt.executeQuery (
            "SELECT ad_composite FROM Print_media WHERE product_id = 2056 AND ad_
id = 12001");
        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();
    }
}
```

## Reading a Portion of the LOB (substr)

Figure 10-17 Use Case Diagram: Reading a Portion of the LOB (substr)



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#)  
on page 10-2

### Purpose

This procedure describes how to read portion of the LOB (substring).

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — SUBSTR, OPEN, CLOSE
- C (OCI): There is no applicable syntax reference for this use case.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — ALLOCATE, LOB OPEN, LOB READ, LOB CLOSE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB READ. See PL/SQL DBMS\_LOB.SUBSTR.
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oralob > PROPERTIES > offset, chunksize, and >OBJECTS > Oralob > METHODS > read
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

This example demonstrates reading a portion from image `ad_photo`.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Reading a Portion of the LOB \(substr\) on page 10-122](#)
- C (OCI): No example is provided with this release.
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Reading a Portion of the LOB \(substr\) on page 10-123](#)
- [C/C++ \(Pro\\*C/C++\): Reading a Portion of the LOB \(substr\) on page 10-124](#)
- [Visual Basic \(OO4O\): Reading a Portion of the LOB \(substr\) on page 10-125](#)
- [Java \(JDBC\): Reading a Portion of the LOB \(substr\) on page 10-126](#)

## PL/SQL (DBMS\_LOB Package): Reading a Portion of the LOB (substr)

```

/* Reading portion of the LOB data using substr.
   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 ad_photo INTO Lob_loc FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;
    /* 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(ad_photo, 255, 1) FROM Print_media WHERE product_id =
3060 AND ad_id = 11001;

```

**COBOL (Pro\*COBOL): Reading a Portion of the LOB (substr)**

```

* READING PORTION OF THE LOB DATA USING SUBSTR
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 "SAMP/SAMP".
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL VAR BUFFER2 IS VARRAW(32767) END-EXEC.

PROCEDURE DIVISION.
BLOB-SUBSTR.
EXEC SQL WHENEVER SQLEERROR 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 AD_COMPOSITE INTO :BLOB1
FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268
AND AD_ID = 21001 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 ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Reading a Portion of the LOB (substr)

/\* Reading portion of the LOB using (substr). 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 ad_photo INTO Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
/* 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);
}

```

## Visual Basic (OO4O): Reading a Portion of the LOB (substr)

```

'Reading portion of a LOB (or BFILE). In OO4O this is accomplished by
'setting the OraBlob.Offset and OraBlob.chunksize properties.
'Using the OraClob.Read mechanism
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn as OraDynaset, OraAdSourceText 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 Print_media", ORADYN_DEFAULT)
Set OraAdSourceText = OraDyn.Fields("ad_sourcetext").Value

'Let's read 100 bytes from the 500th byte onwards:
OraAdSourceText.Offset = 500
OraAdSourceText.PollingAmount = OraAdSourceText.Size 'Read entire CLOB contents
amount_read = OraAdSourceText.Read(chunk, 100)
'chunk returned is a variant of type byte array
```

## Java (JDBC): Reading a Portion of the LOB (substr)

```
// Reading portion of a LOB using substr
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:
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

        // 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 ad_composite FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    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();

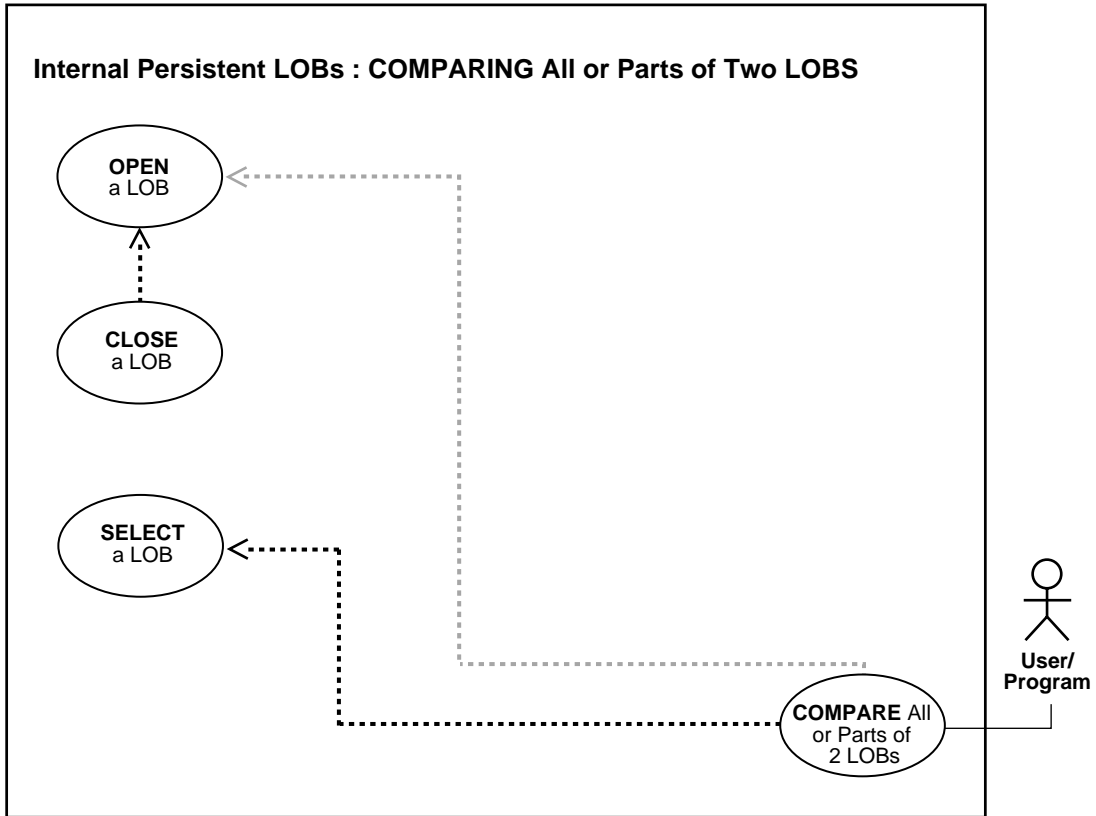
    // MAXBUFSIZE is the number of bytes to read and 1000 is the offset from
    // which to start reading:
    buf = lob_loc.getBytes(1000, MAXBUFSIZE);
    // Display the contents of the buffer here.

    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();
}
}
```

## Comparing All or Part of Two LOBs

Figure 10–18 Use Case Diagram: Comparing All or Part of Two LOBs



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to compare all or part of two LOBs.

### Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — COMPARE.
- C (OCI): There is no applicable syntax reference for this use case.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* or information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — EXECUTE. Also reference PL/SQL DBMS\_LOB.COMPARE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — EXECUTE. Also reference PL/SQL DBMS\_LOB.COMPARE.
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oradynaset > METHODS > movenext
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples compare two images from the table `Print_media` to see whether they are different.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Comparing All or Part of Two LOBs on page 10-130](#)
- C (OCI): No example is provided with this release.
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): [Comparing All or Part of Two LOBs on page 10-130](#)

- [C/C++ \(Pro\\*C/C++\): Comparing All or Part of Two LOBs](#) on page 10-132
- [Visual Basic \(OO4O\): Comparing All or Part of Two LOBs](#) on page 10-133
- [Java \(JDBC\): Comparing All or Part of Two LOBs](#) on page 10-134

## PL/SQL (DBMS\_LOB Package): Comparing All or Part of Two LOBs

```
/* Comparing all or part of two LOBs. 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 ad_composite INTO Lob_loc1 FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;
    SELECT ad_composite INTO Lob_loc2 FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;
```

## COBOL (Pro\*COBOL): Comparing All or Part of Two LOBs

```
* COMPARING ALL OR PART OF TWO LOBS
IDENTIFICATION DIVISION.
PROGRAM-ID. COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
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

```

21001

```

SELECT AD_COMPOSITE INTO :BLOB1
FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 2268 AND AD_ID =

```

```

END-EXEC.

```

```

EXEC SQL

```

```

SELECT AD_COMPOSITE INTO :BLOB2
FROM PRINT_MEDIA PM WHERE PM.PRODUCT_ID = 3060 AND AD_ID = 11001
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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLEERROR CONTINUE END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

```

## C/C++ (Pro\*C/C++): Comparing All or Part of Two LOBs

```

/* Comparing all or part of two LOBs
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 ad_composite INTO :Lob_loc1
      FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
EXEC SQL SELECT ad_composite INTO :Lob_loc2
      FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
/* 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;
}

```

## Visual Basic (OO4O): Comparing All or Part of Two LOBs

```

'Comparing all or part of two LOBs
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, OraAdPhotol as OraBLOB, OraAdPhotoClone as OraBLOB

Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value
'Clone it for future reference
Set OraAdPhotoClone = OraAdPhotol.Clone

'Lets go to the next row and compare LOBs
OraDyn.MoveNext

MsgBox CBool(OraAdPhotol.Compare(OraAdPhotoClone, OraAdPhotoClone.size, 1, 1))
```

## Java (JDBC): Comparing All or Part of Two LOBs

```
// Comparing all or part of two LOBs
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:
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        // Connect to the database:
```

```
Connection conn =
    DriverManager.getConnection ("jdbc:oracle:oci8:@", "pm", "pm");

// 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 ad_composite FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001");
    if (rset.next())
    {
        lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

    rset = stmt.executeQuery (
        "SELECT ad_composite FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    if (rset.next())
    {
        lob_loc2 = ((OracleResultSet)rset).getBLOB (1);
    }

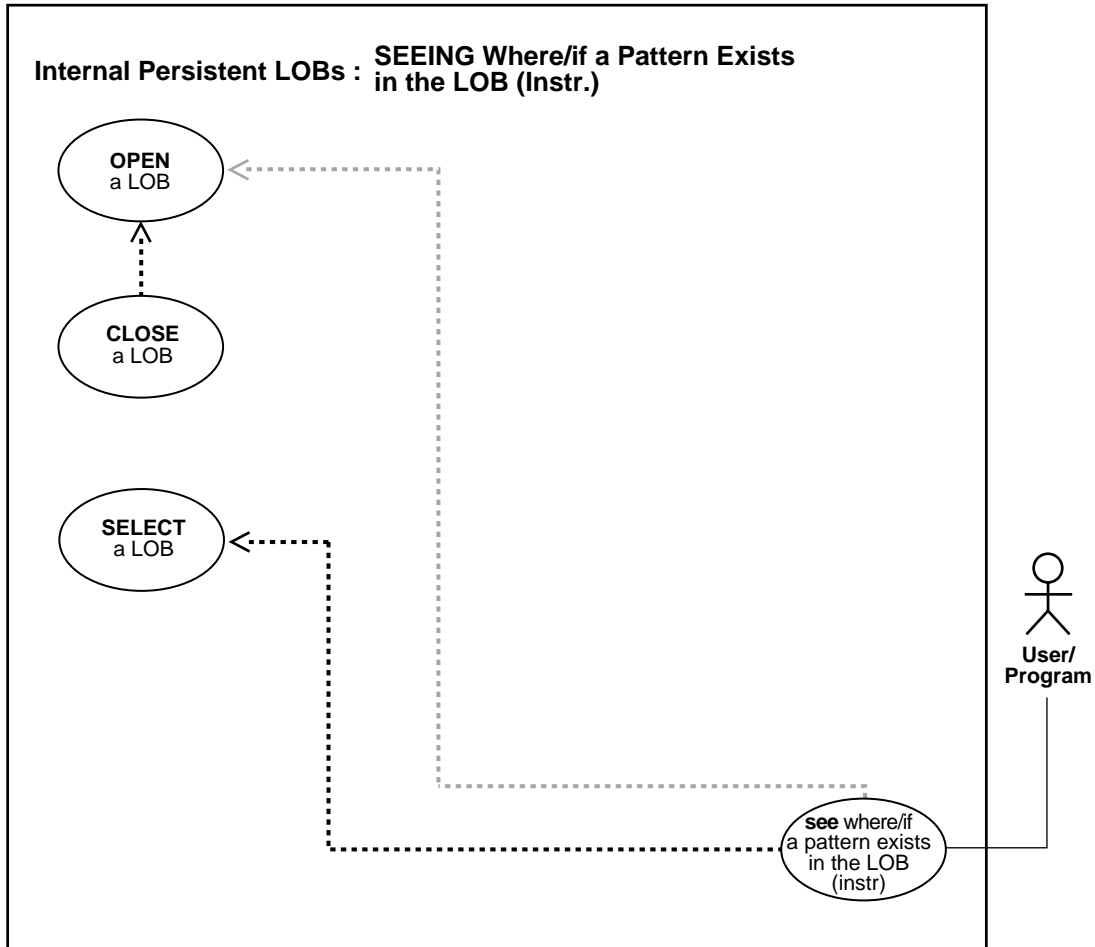
    if (lob_loc1.length() > lob_loc2.length())
        System.out.println ("Looking for LOB2 inside LOB1. result = "
            + Long.toString(lob_loc1.position(lob_loc2, 1)));
    else
        System.out.println("Looking for LOB1 inside LOB2. result = "
            + Long.toString(lob_loc2.position(lob_loc1, 1)));

    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```



## Patterns: Checking for Patterns in the LOB (instr)

Figure 10–19 Use Case Diagram: Checking for Pattern in the LOB (instr)



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to see if a pattern exists in the LOB (instr).

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — INSTR.
- C (OCI): There is no applicable syntax reference for this use case.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — EXECUTE. Also reference PL/SQL DBMS\_LOB.INSTR.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — EXECUTE. Also reference PL/SQL DBMS\_LOB.INSTR.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The examples examine the advertisement text in the `ad_sourcetext` column to see if the string "children" is present.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Checking for Pattern in the LOB \(instr\)](#) on page 10-139
- C (OCI): No example is provided with this release.
- C++ (OCCI): No example is provided with this release.

- [COBOL \(Pro\\*COBOL\): Checking for Patterns in the LOB \(instr\)](#) on page 10-139
- [C/C++ \(Pro\\*C/C++\): Checking for Patterns in the LOB \(instr\)](#) on page 10-141
- Visual Basic (OO4O): No example is provided with this release.
- [Java \(JDBC\): Checking for Patterns in the LOB \(instr\)](#) on page 10-142

## PL/SQL (DBMS\_LOB Package): Checking for Pattern in the LOB (instr)

```

/* Seeing if a pattern exists in the LOB using instr.
   The example procedure instrstringLOB_proc is not part of the
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instrstringLOB_proc IS
  Lob_loc          CLOB;
  Pattern          VARCHAR2(30) := 'children';
  Position         INTEGER := 0;
  Offset           INTEGER := 1;
  Occurrence       INTEGER := 1;
BEGIN
  /* Select the LOB: */
  SELECT ad_sourcetext INTO Lob_loc
    FROM Print_media WHERE product_id = 2268 AND ad_id = 21001;
  /* 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;

```

## COBOL (Pro\*COBOL): Checking for Patterns in the LOB (instr)

```

* SEEING IF A PATTERN EXISTS IN THE LOB USING INSTR
  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 "SAMP/SAMP".
                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 AD_SOURCETEXT INTO :CLOB1
        FROM PRINT_MEDIA
        WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 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 ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Checking for Patterns in the LOB (instr)

```

/* Seeing if a pattern exists in the LOB using instr
#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 ad_sourcetext INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
    /* 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 = "pm/pm";
    EXEC SQL CONNECT :pm;
    instrstringLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Java (JDBC): Checking for Patterns in the LOB (instr)

```
// Seeing if a pattern exists in the LMOB using instr
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:
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    }
}
```

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

// 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 ad_sourcetext FROM Print_media
        WHERE product_id = 2268 AND ad_id = 21001");
    if (rset.next())
    {
        lob_loc = ((OracleResultSet)rset).getCLOB (1);
    }

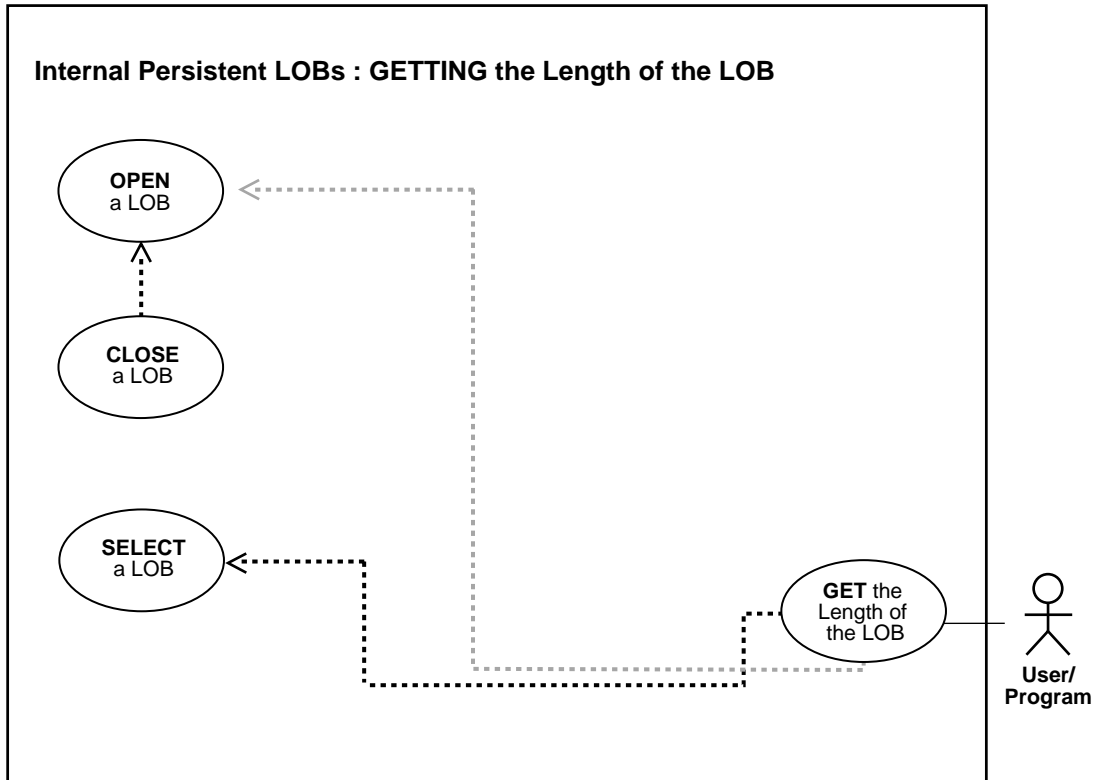
    // Search for location of pattern string in the CLOB, starting at offset 1:
    long result = lob_loc.position(pattern, offset);
    System.out.println("Results of Pattern Comparison : " +
        Long.toString(result));

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Length: Determining the Length of a LOB

Figure 10–20 Use Case Diagram: Determining the Length of a LOB



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to determine the length of a LOB.

### Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — GETLENGTH
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OciLobGetLength.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE ...GET LENGTH...
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oradynaset > PROPERTIES > fields
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples demonstrate how to determine the length of the LOB in the foreign language text (ad\_fltextn) column.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Determining the Length of a LOB on page 10-146](#)
- [C \(OCI\): Determining the Length of a LOB on page 10-146](#)
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Determining the Length of a LOB on page 10-148](#)

- [C/C++ \(Pro\\*C/C++\): Determining the Length of a LOB](#) on page 10-149
- [Visual Basic \(OO4O\): Determining the Length of a LOB](#) on page 10-150
- [Java \(JDBC\): Determining the Length of a LOB](#) on page 10-150

## PL/SQL (DBMS\_LOB Package): Determining the Length of a LOB

```
/* Getting the length of a LOB.
   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 ad_flgtextn INTO Lob_loc FROM Print_media
     WHERE product_id = 3106 AND ad_id = 13001;
  /* 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;
```

## C (OCI): Determining the Length of a LOB

```
/* Getting the length of a LOB
/* Select the locator into a locator variable */
sb4 select_adflgtextn_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
  OCIDefine *defnp1;

  text *sqlstmt =
```

```

(text *)"SELECT ad_flgtextn FROM Print_media
        WHERE product_id = 2268";

checkerr (errhp, OCISstmtPrepare(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, OCISstmtExecute(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;
OCISstmt    *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 ad_flgtextn locator\n");
    select_ad_flgtextn_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 ad_flgtextn.\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;
}
```

## COBOL (Pro\*COBOL): Determining the Length of a LOB

```
* GETTING THE LENGTH OF A LOB
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 "SAMP/SAMP".
   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 AD_SOURCETEXT INTO :CLOB1
       FROM PRINT_MEDIA
       WHERE PRODUCT_ID = 3060 AND AD_ID = 11001 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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

```

```

SQL-ERROR.
EXEC SQL WHENEVER SQLEERROR CONTINUE END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

```

## C/C++ (Pro\*C/C++): Determining the Length of a LOB

```

/* Getting the length of a LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 SQLEERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
    /* 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 uninitialized, 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;
}
```

## Visual Basic (OO4O): Determining the Length of a LOB

```
'Getting the length of a LOB
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value

'Display out size of the lob:
MsgBox "Length of the lob is " & OraAdPhotol.Size
```

## Java (JDBC): Determining the Length of a LOB

```
//Getting the length of a LOB
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:
        DriverManager.registerDriver(new 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 ad_sourcetext FROM Print_media WHERE product_id = 3106");
            if (rset.next())
            {
                lob_loc = ((OracleResultSet)rset).getCLOB (1);
            }

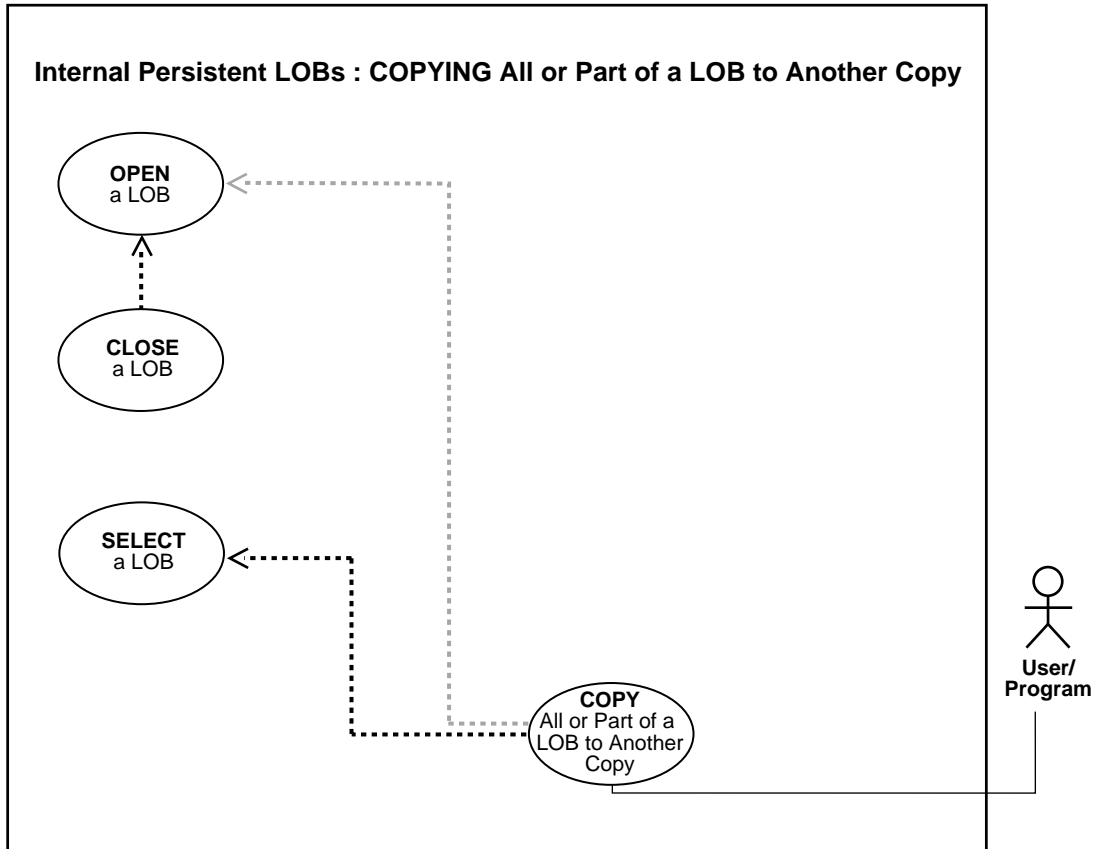
            System.out.println(
                "Length of this column is : " + Long.toString(lob_loc.length()));

            stmt.close();
            conn.commit();
            conn.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

```
}  
}  
}
```

## Copying All or Part of One LOB to Another LOB

Figure 10–21 Use Case Diagram: Copying All or Part of One LOB to Another LOB



See Also: [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to copy all or part of a LOB to another LOB.

## Usage Notes

**Locking the Row Prior to Updating** Prior to updating a LOB value using the PL/SQL DBMS\_LOB package or 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 ["Updating LOBs Via Updated Locators"](#) on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. See the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — COPY
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OciLobCopy
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB COPY. Also reference PL/SQL DBMS\_LOB.COPY.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* for information on embedded SQL statements and directives — LOB COPY
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > copy, and > OBJECTS > Oradynaset > METHODS > movenext, edit
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE

## Scenario

The code in these examples shows you how to copy a portion of an image in the `ad_photo` column to the `ad_composite` column.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Copying All or Part of One LOB to Another LOB on page 10-155](#)
- [C \(OCI\): Copying All or Part of One LOB to Another LOB on page 10-156](#)
- [COBOL \(Pro\\*COBOL\): Copying All or Part of One LOB to Another LOB on page 10-158](#)
- [C/C++ \(Pro\\*C/C++\): Copy All or Part of a LOB to Another LOB on page 10-160](#)
- [Visual Basic \(OO4O\): Copying All or Part of One LOB to Another LOB on page 10-161](#)
- [Java \(JDBC\): Copying All or Part of One LOB to Another LOB on page 10-162](#)

## PL/SQL (DBMS\_LOB Package): Copying All or Part of One LOB to Another LOB

```

/* Copying all or part of a LOB to another LOB.
   Example procedure copyLOB_proc is not part of 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 ad_composite INTO Dest_loc FROM Print_media
     WHERE product_id = 3106 AND ad_id = 13001 FOR UPDATE;
  /* Select the LOB: */
  SELECT ad_photo INTO Src_loc FROM Print_media
     WHERE product_id = 2056 AND ad_id = 12001;
  /* 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;
```

## C (OCI): Copying All or Part of One LOB to Another LOB

```

/* Copying all or part of a LOB to another LOB */
/* Select the locator */
sb4 select_photo_locator_2(Lob_loc, dest_type, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
ub1          dest_type;                /* whether destination locator */
OCIError     *errhp;
OCISvcCtx    *svchp;
OCISntmt     *stmthp;
{
    char      sqlstmt[150];
    OCIDefine *defnp1;
    if (dest_type == TRUE)
    {
        strcpy (sqlstmt,
                (char *)"SELECT ad_photo FROM Print_media
                WHERE product_id=2268 FOR UPDATE");
        printf (" select destination ad_photo locator\n");
    }
    else
    {
        strcpy(sqlstmt, (char *)"SELECT ad_photo FROM Print_media WHERE product_
id=3106");
        printf (" select source ad_photo locator\n");
    }
    checkerr (errhp, OCISntmtPrepare(stmthp, errhp, (text *)sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4) OCI_NIV_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, OCISntmtExecute(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_photo_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
                                                                    /* destination locator */
    select_photo_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;
}
```

## COBOL (Pro\*COBOL): Copying All or Part of One LOB to Another LOB

```
* COPYING ALL OR PART OF A LOB TO ANOTHER LOB
IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-COPY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_PHOTO INTO :SRC
    FROM PRINT_MEDIA PM
    WHERE PM.PRODUCT_ID = 3106 AND AD_ID = 13001 END-EXEC.
DISPLAY "Source LOB populated.".
EXEC SQL
    SELECT AD_PHOTO INTO :DEST
    FROM PRINT_MEDIA PM
    WHERE PM.PRODUCT_ID = 3060 AND AD_ID = 11001 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 ROLLBACK WORK RELEASE END-EXEC.

```

```

        STOP RUN.

SQL-ERROR.
    EXEC SQL WHENEVER SQLEERROR CONTINUE END-EXEC.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED:".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
    STOP RUN.

```

## C/C++ (Pro\*C/C++): Copy All or Part of a LOB to Another LOB

```

/* Copying all or part of a LOB to another LOB */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 SQLEERROR DO Sample_Error();
    /* Allocate the LOB locators: */
    EXEC SQL ALLOCATE :Dest_loc;
    EXEC SQL ALLOCATE :Src_loc;
    /* Select the LOBs: */
    EXEC SQL SELECT ad_photo INTO :Dest_loc
        FROM Print_media WHERE product_id = 2268 AND AD_ID = 21001 FOR
UPDATE;
    EXEC SQL SELECT ad_photo INTO :Src_loc
        FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;
}

```

## Visual Basic (OO4O): Copying All or Part of One LOB to Another LOB

```

'Copying all or part of a LOB to another LOB
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value

Set OraAdPhotoClone = OraAdPhoto1.Clone

'Go to next row and copy LOB

OraDyn.MoveNext

OraDyn.Edit
OraAdPhoto1.Copy OraAdPhotoClone, OraAdPhotoClone.Size, 1, 1
OraDyn.Update

```

## Java (JDBC): Copying All or Part of One LOB to Another LOB

```
// Copying all or part of a LOB to another LOB
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:
        DriverManager.registerDriver(new 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 ad_photo FROM Print_media
        WHERE product_id = 3060 AND ad_ad = 11001");
    if (rset.next())
    {
        src_loc = ((OracleResultSet)rset).getBLOB (1);
    }
    in = src_loc.getBinaryStream();

    rset = stmt.executeQuery (
        "SELECT ad_photo FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001 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);

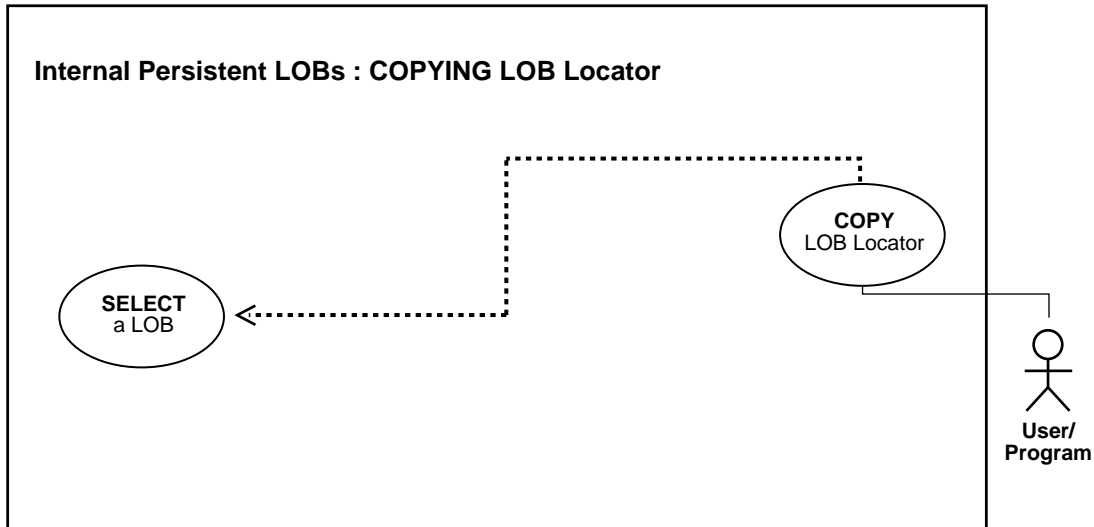
    // write AMOUNT_TO_COPY bytes from buf into output stream, starting at offset
0:
    out.write(buf, 0, AMOUNT_TO_COPY);

    // Close all streams and handles
    in.close();
    out.flush();
    out.close();
    stmt.close();
    conn.commit();
    conn.close();

    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
}
```

## Copying a LOB Locator

Figure 10–22 Use Case Diagram: Copying a LOB Locator



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to copy a LOB locator.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): Refer to the ["Read Consistent Locators"](#) in [Chapter 5, "Large Objects: Advanced Topics"](#) for information on assigning one lob locator to another.



- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OciLobAssign, OciLobIsEqual.
- **C++ (OCCI):** *Oracle C++ Call Interface Programmer's Guide*
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — ALLOCATE, LOB ASSIGN.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — SELECT, LOB ASSIGN
- **Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help):** From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > copy
- **Java (JDBC):** *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples show how to copy one locator to another involving the image (ad\_composite). Note how different locators may point to the same or different, current or outdated data.

## Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Copying a LOB Locator](#) on page 10-166
- **C (OCI):** [Copying a LOB Locator](#) on page 10-166
- **C++ (OCCI):** No example is provided with this release.
- **COBOL (Pro\*COBOL):** [Copying a LOB Locator](#) on page 10-168
- **C/C++ (Pro\*C/C++):** [Copying a LOB Locator](#) on page 10-169
- **Visual Basic (OO4O):** [Copying a LOB Locator](#) on page 10-170
- **Java (JDBC):** [Copying a LOB Locator](#) on page 10-170

## PL/SQL (DBMS\_LOB Package): Copying a LOB Locator

---



---

**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 5-1.

---



---

```

/* Copying a LOB locator.
   Example procedure lobAssign_proc is not part of DBMS_LOB package. */
CREATE OR REPLACE PROCEDURE lobAssign_proc IS
  Lob_loc1   blob;
  Lob_loc2   blob;
BEGIN
  SELECT ad_composite INTO Lob_loc1 FROM Print_media
     WHERE product_id = 3106 AND ad_id = 13001 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;
```

## C (OCI): Copying a LOB Locator

```

/* Copying a LOB locator */
/* Select the locator */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
  text *sqlstmt =
    (text *)"SELECT ad_composite FROM Print_media
           WHERE product_id=3106 AND ad_id = 13001 FOR UPDATE";
  OCIDefine *defnp1;
  checkerr (errhp, OCISstmtPrepare(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, OCISstmtExecute(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;
OCISstmt    *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_adcomp_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;
}

```

## COBOL (Pro\*COBOL): Copying a LOB Locator

```

* COPYING A LOB LOCATOR
IDENTIFICATION DIVISION.
PROGRAM-ID. COPY-LOCATOR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_COMPOSITE INTO :SRC
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 2268 AND AD_ID = 21001 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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

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

## C/C++ (Pro\*C/C++): Copying a LOB Locator

```
/* Copying a LOB locator */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 SQLEERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc1;
    EXEC SQL ALLOCATE :Lob_loc2;
    EXEC SQL SELECT ad_composite INTO :Lob_loc1
        FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001 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;
}
```

## Visual Basic (OO4O): Copying a LOB Locator

```
'Copying a LOB locator
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id ", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value
Set OraAdPhotoClone = OraAdPhotol.Clone

OraDyn.MoveNext

'Copy 1000 bytes of LOB values OraAdPhotoClone to OraAdPhotol at OraAdPhotol
'offset 100:
OraDyn.Edit
OraAdPhotol.Copy OraAdPhotoClone, 1000, 100
OraDyn.Update
```

## Java (JDBC): Copying a LOB Locator

```
// Copying a LOB locator
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:
    DriverManager.registerDriver(new 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 ad_composite FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001");
    if (rset.next())
    {
    lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
    }

    // When you write data to LOB through lob_loc1,lob_loc2 will not see changes
    lob_loc2 = lob_loc1;
    stmt.close();
    conn.commit();
    conn.close();

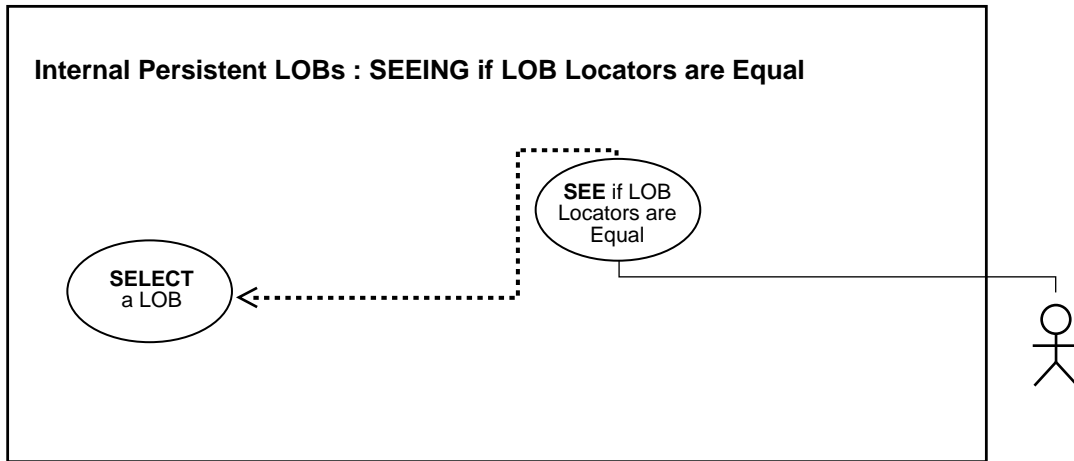
    }
}
```

```
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```



## Equality: Checking If One LOB Locator Is Equal to Another

Figure 10–23 Use Case Diagram: Checking If One LOB Locator Is Equal to Another



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to see if one LOB locator is equal to another.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, `OciLobAssign`, `OciLobIsEqual`.

- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB ASSIGN
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### 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 5-1). 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.

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB): No example is provided with this release.
- [C \(OCI\): Checking If One LOB Locator Is Equal to Another](#) on page 10-174
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- [C/C++ \(Pro\\*C/C++\): Checking If One LOB Locator Is Equal to Another](#) on page 10-176
- Visual Basic (OO4O): No example is provided with this release.
- [Java \(JDBC\): Checking If One LOB Locator Is Equal to Another](#) on page 10-177

## C (OCI): Checking If One LOB Locator Is Equal to Another

```
/* Seeing if One LOB locator is Equal to Another */
/* Select the locator: */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
```

```

OCIStmt      *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT ad_composite FROM Print_media
                WHERE product_id=2268 AND ad_id = 21001 FOR UPDATE";
    OCIDefine *defnpl;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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 locatorIsEqual(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 an ad_composite locator\n");
    select_lock_adcomp_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;
}
```

## C/C++ (Pro\*C/C++): Checking If One LOB Locator Is Equal to Another

```
/* Seeing if One LOB locator is equal to another */
/* Pro*C/C++ does not provide a mechanism to test the equality of two
   locators. But you can use 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 ad_composite INTO Lob_loc1
        FROM Print_media
            where product_id = 3060 AND ad_id = 11001 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;
}

```

## Java (JDBC): Checking If One LOB Locator Is Equal to Another

```
// Seeing if one LOB locator is equal to another
```

```
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:
        DriverManager.registerDriver(new 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 ad_photo FROM Print_media
                WHERE product_id = 3106 AND ad_id = 13001");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBLOB (1);
            }

            // When you write data to 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.
        }
    }
}
```

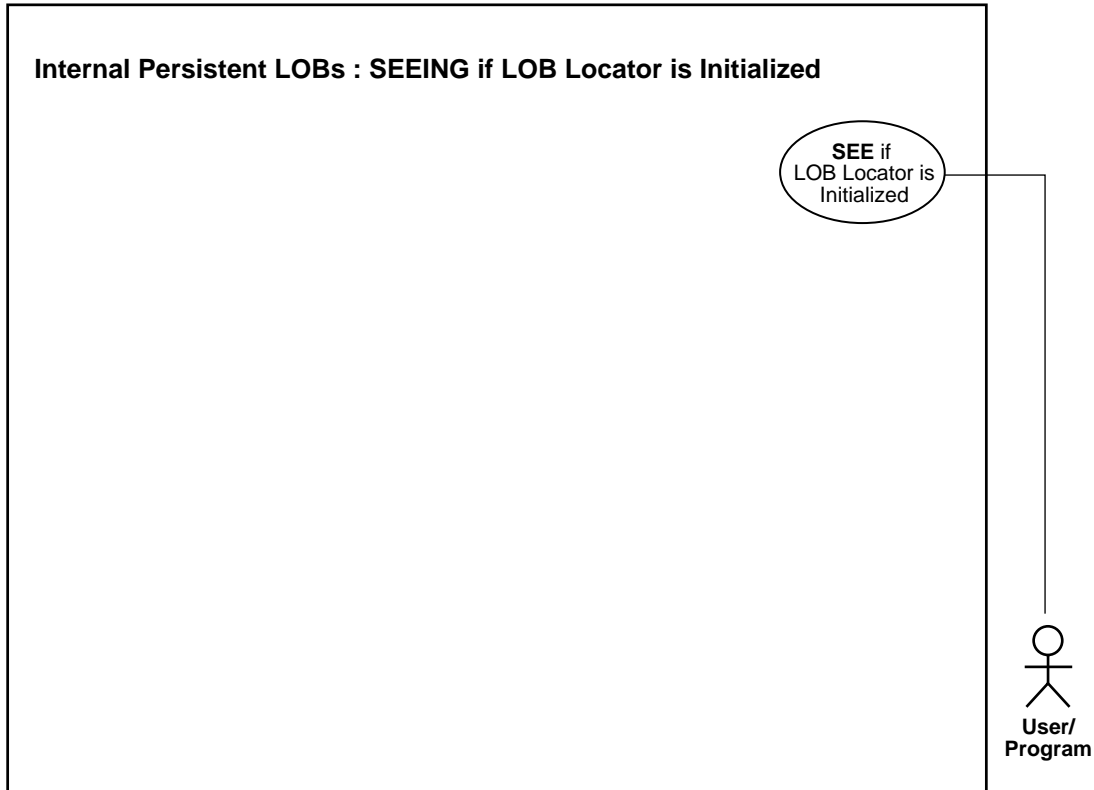
```
    if (lob_loc1.equals(lob_loc2))
    {
        // The Locators are equal:
        System.out.println("Locators are equal");
    }
    else
    {
        // The Locators are different:
        System.out.println("Locators are NOT equal");
    }

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Initialized Locator: Checking If a LOB Locator Is Initialized

*Figure 10–24 Use Case Diagram: Checking If a LOB Locator Is Initialized*



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### **Purpose**

This procedure describes how to see if a LOB locator is initialized.

### **Usage Notes**

Not applicable.



## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — `OciLobLocatorIsInit`.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++) *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives". See C(OCI), `OciLobLocatorIsInit`.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): There is no applicable syntax reference for this use case.

## 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.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- C (OCI): [Checking If a LOB Locator Is Initialized](#) on page 10-181
- C (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): [Checking If a LOB Locator Is Initialized](#) on page 10-183
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Checking If a LOB Locator Is Initialized

```
/* Seeing if a LOB locator is initialized */
/* Select the locator: */
```

```

sb4 select_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text      *sqlstmt =
        (text *) "SELECT ad_composite FROM Print_media
                WHERE product_id=2268 AND ad_id = 21001";
    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 an ad_composite locator into locator 1\n");
    select_adcomp_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 FALSE 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;
}

```

## C/C++ (Pro\*C/C++): Checking If a LOB Locator Is Initialized

```

/* Seeing if a LOB locator is initialized */
/* 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. 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();
}

```

```

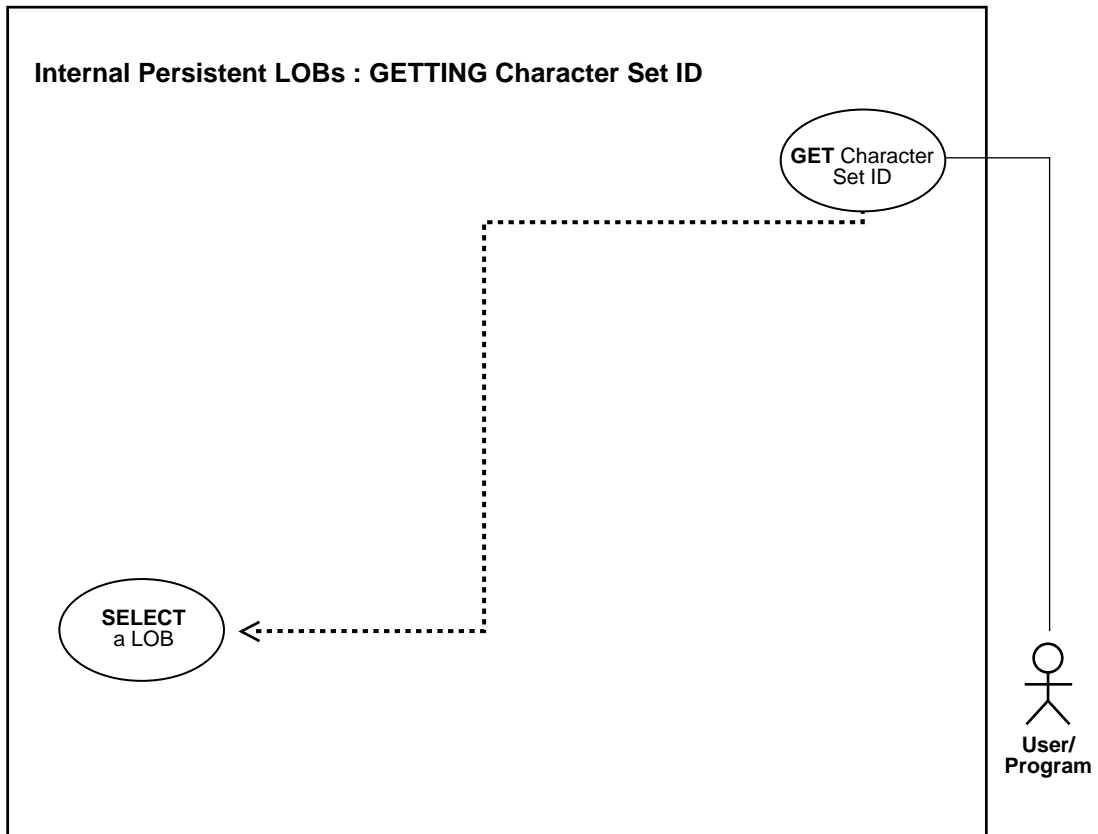
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL SELECT ad_composite INTO Lob_loc
      FROM Print_media where product_id = 2056 AND ad_id = 12001;
/* Get the OCI Environment Handle using a SQLLIB Routine: */
(void) SQLEnvGet(SQL_SINGLE_RCTX, &oeh);
/* Allocate the OCI Error Handle: */
(void) OCIHandleAlloc((dvoid *)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;
}

```

## Character Set ID: Determining Character Set ID

Figure 10–25 Use Case Diagram: Determining Character Set ID



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to determine the character set ID.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCharSetId
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): There is no applicable syntax reference for this use case.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): There is no applicable syntax reference for this use case.

## Scenario

The use case demonstrates how to determine the character set ID of the foreign text (`ad_fltextrn`).

## Example

This functionality is currently available only in OCI:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- [C \(OCI\): Determining Character Set ID](#) on page 10-187
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Determining Character Set ID

```

/* Getting character set id */
/* This function takes a valid LOB locator and prints the character set id of
the LOB. */
/* Select the locator */
sb4 select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *defnpl;
    text *sqlstmt =
        (text *)"SELECT ad_fltextn FROM Print_media
                WHERE product_id = 2268 AND ad_id = 21001";
    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, &defnpl, 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 ad_fltextn locator\n");
    select_adfltextn_locator(Lob_loc, errhp, svchp, stmthp);
    printf (" get the character set id of adfltextn_locator\n");

    /* Get the charactersid ID of the LOB*/

```

```
checkerr (errhp, OCILobCharSetId(envhp, errhp, Lob_loc, &charsetid));
printf(" character Set ID of ad_flgtextn is : %d\n", charsetid);

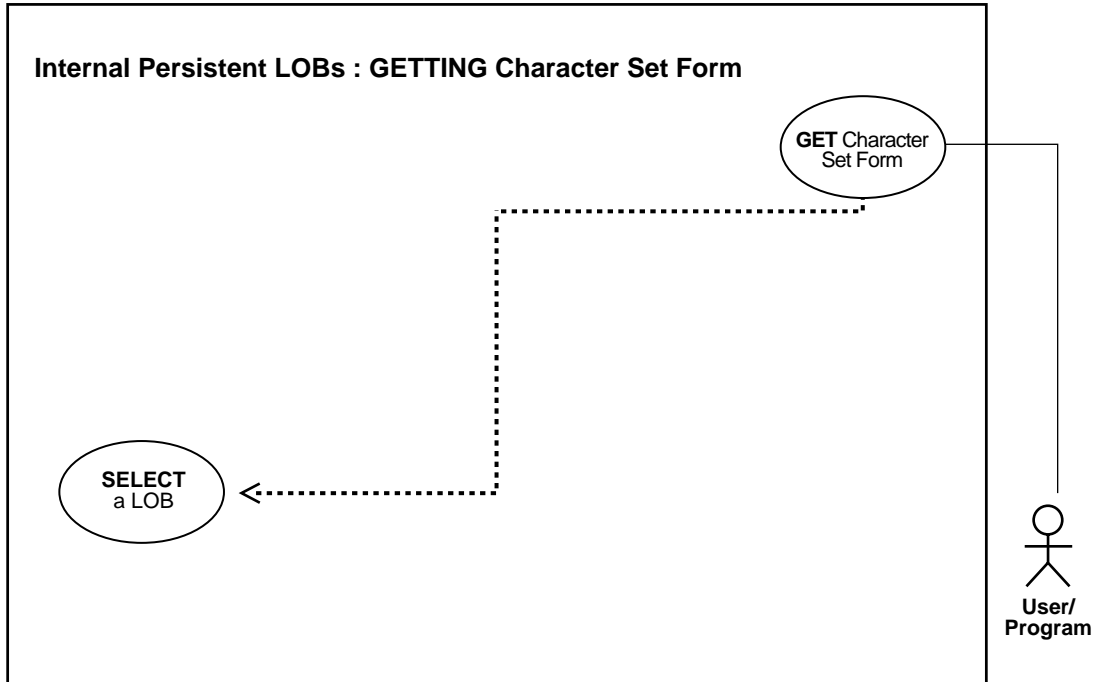
/* Free resources held by the locators*/
(void) OCIDescriptorFree((dvoid *) Lob_loc, (ub4) OCI_DTYPE_LOB);

return;
}
```



## Character Set Form: Determining Character Set Form

Figure 10–26 Use Case Diagram: Determining Character Set Form



**See Also:** [Table 10-1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to get the character set form.

### Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCharSetForm
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): There is no applicable syntax reference for this use case.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): There is no applicable syntax reference for this use case.

## Scenario

The use case demonstrates how to determine the character set form of the foreign language text (`ad_flgtextn`).

## Example

This functionality is currently available only in OCI:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- [C \(OCI\): Determining Character Set Form](#) on page 10-190
- C (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Determining Character Set Form

```
/* Getting character set form of the foreign language ad text, ad_flgtextn */
/* Select the locator */
sb4 select_adflgtextn_locator(lob_loc, errhp, svchp, stmthp)
```

```

OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *defnpl;
    text *sqlstmt =
        (text *) "SELECT ad_flgtextn FROM Print_media
                WHERE product_id = 2268 AND ad_id = 21001";
    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, sqlstmt,
                                     (ub4)strlen((char *)sqlstmt),
                                     (ub4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

    /* Define the column being selected */
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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, OCISstmtExecute(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;
OCISstmt    *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 an ad_flgtextn locator\n");
    select_adflgtextn_locator(Lob_loc, errhp, svchp, stmthp);
    printf (" get the character set form of ad_flgtextn\n");
}

```

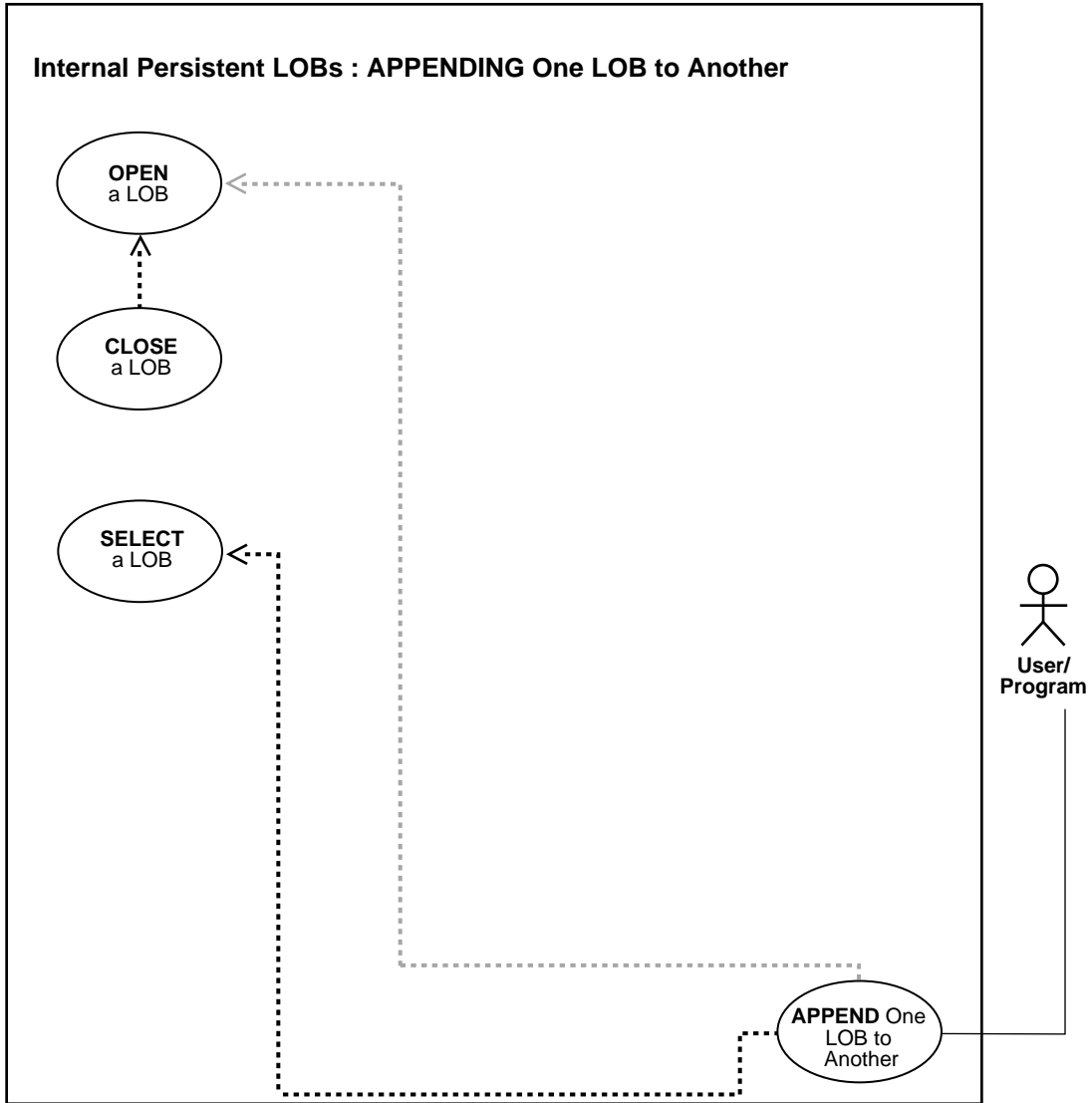
```
/* Get the charactersid form of the LOB*/
checkerr (errhp, OCILobCharSetForm(envhp, errhp, Lob_loc, &charset_form));
printf(" character Set Form of ad_flgtextn is : %d\n", charset_form);

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

return;
}
```

# Appending One LOB to Another

Figure 10-27 Use Case Diagram: Appending One LOB to Another



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to append one LOB to another.

## Usage Notes

**Locking the Row Prior to Updating** Prior to updating a LOB value using 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 ["Updating LOBs Via Updated Locators"](#) on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. See the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — APPEND
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — OCILobAppend
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB APPEND.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* for information on embedded SQL statements and directives — LOB APPEND
- Visual Basic (OO4O): (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oralob > METHODS > append
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i*

*SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples append one image to another.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Appending One LOB to Another](#) on page 10-195
- [C \(OCI\): Appending One LOB to Another](#) on page 10-196
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Appending One LOB to Another](#) on page 10-198
- [C/C++ \(Pro\\*C/C++\): Appending One LOB to Another](#) on page 10-199
- [Visual Basic \(OO4O\): Appending One LOB to Another](#) on page 10-200
- [Java \(JDBC\): Appending One LOB to Another](#) on page 10-201

## PL/SQL (DBMS\_LOB Package): Appending One LOB to Another

```

/* Appending one LOB to another */
/* 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 ad_photo INTO Dest_loc FROM Print_media
        WHERE product_id = 2268 AND ad_id = 21001 FOR UPDATE;
    /* Select the LOB, get the destination LOB locator: */
    SELECT ad_photo INTO Src_loc FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;

```

## C (OCI): Appending One LOB to Another

```

/* Appending one LOB to another. */
/* This function appends the Source LOB to the end of the Destination LOB */
/* Select the locator */
sb4 select_lock_adphoto_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 ad_photo FROM Print_media
                WHERE product_id=2268 AND ad_id=21001 FOR UPDATE");
        printf (" select destination ad_photo locator\n");
    }
    else
    {
        strcpy(sqlstmt, (char *)"SELECT ad_photo FROM Print_media
            WHERE product_id=3106 and ad_id=13001");
        printf (" select source ad_photo locator\n");
    }
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, (text *)sqlstmt,
        (ub4)strlen((char *)sqlstmt),
        (ub4) OCI_NIV_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_adphoto_locator_2(Dest_loc, TRUE, errhp, svchp, stmthp);
    /* destination locator */
    select_lock_adphoto_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;
}

```

## COBOL (Pro\*COBOL): Appending One LOB to Another

```
* APPENDING ONE LOB TO ANOTHER
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-APPEND.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(11) VALUES "SAMP/SAMP".
01  DEST            SQL-BLOB.
01  SRC             SQL-BLOB.
      EXEC SQL INCLUDE SQLCA END-EXEC.

PROCEDURE DIVISION.
APPEND-BLOB.
      EXEC SQL WHENEVER SQLEERROR 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 AD_PHOTO INTO :DEST
      FROM PRINT_MEDIA WHERE PRODUCT_ID = 2268
      AND AD_ID = 21001 FOR UPDATE END-EXEC.
      EXEC SQL SELECT AD_PHOTO INTO :SRC
      FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060
      AND AD_ID = 11001 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 ROLLBACK 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.
    
```

## C/C++ (Pro\*C/C++): Appending One LOB to Another

```

/* Appending one LOB to another */
#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 Print_media WHERE product_id = 2268 AND
        ad_id = 21001 FOR UPDATE;

    /* Select the source locator: */
    EXEC SQL SELECT Sound INTO :Src_loc
        FROM Print_media WHERE product_id = 3060 AND
        ad_id = 11001;
    
```

```
/* 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;
}
```

## Visual Basic (OO4O): Appending One LOB to Another

```
'Appending one LOB to another
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value
Set OraAdPhotoClone = OraAdPhotol

OraDyn.MoveNext
OraDyn.Edit
OraAdPhotol.Append OraAdPhotoClone
OraDyn.Update
```

## Java (JDBC): Appending One LOB to Another

```
// Appending one LOB to another
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:
        DriverManager.registerDriver(new 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 ad_photo FROM Print_media
        WHERE product_id = 2268 AND ad_id = 21001");
    if (rset.next())
    {
        src_loc = ((OracleResultSet)rset).getBLOB (1);
    }
    in = src_loc.getBinaryStream();

        rset = stmt.executeQuery (
            "SELECT ad_photo FROM Print_media
            WHERE product_id = 3060 AND ad_id = 11001 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();
    while ((length = in.read(buf)) != -1)
    {
        // Write the contents of the buffer into position pos of the output LOB:
        dest_loc.putBytes(pos, buf);
        pos += length;
    }

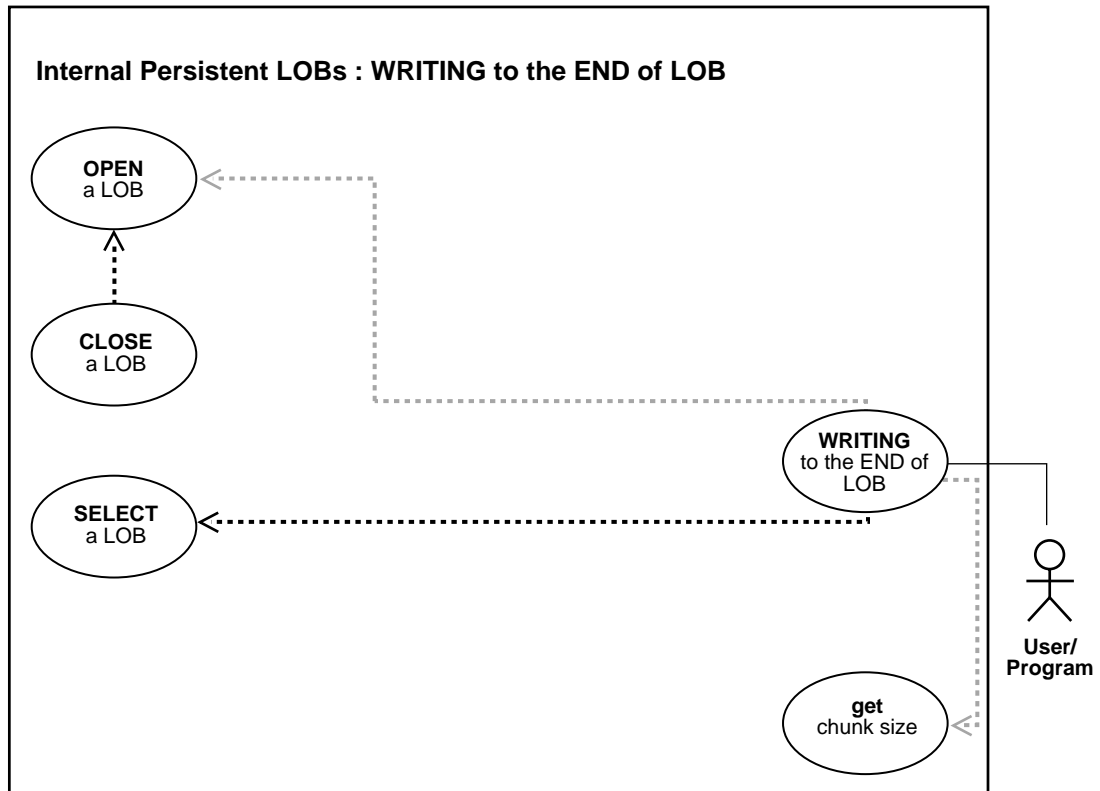
    // Close all streams and handles:
    in.close();
    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
}

```

## Append-Writing to the End of a LOB

Figure 10–28 Use Case Diagram: Append-Writing to the End of a LOB



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### Purpose

This procedure describes how to write to the end of (append-write to) a LOB.

## Usage Notes

**Writing Singly or Piecewise** The `writeappend` operation writes a buffer to the end of a LOB.

For 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.

**Writing Piecewise: When to Use Callbacks or Polling?** 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 using 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 an 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 ["Updating LOBs Via Updated Locators"](#) on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `WRITEAPPEND`
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — `LOB Functions, OCILobWriteAppend`
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*



- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB WRITE APPEND.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB WRITE APPEND
- Visual Basic (OO4O): No syntax reference is provided with this release.
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples demonstrate writing to the end of an image.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Writing to the End of \(Appending to\) a LOB](#) on page 10-205
- [C \(OCI\): Writing to the End of \(Appending to\) a LOB](#) on page 10-206
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Writing to the End of \(Appending to\) a LOB](#) on page 10-208
- [C/C++ \(Pro\\*C/C++\): Writing to the End of \(Appending to\) a LOB](#) on page 10-209
- Visual Basic (OO4O): No example is provided with this release.
- [Java \(JDBC\): Writing to the End of \(Append-Write to\) a LOB](#) on page 10-210

## PL/SQL (DBMS\_LOB Package): Writing to the End of (Appending to) a LOB

```

/* Write-appending to a LOB
/* Example procedure lobWriteAppend_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE lobWriteAppend_proc IS
  Lob_loc      BLOB;
  Buffer        RAW(32767);
  Amount       Binary_integer := 32767;
BEGIN

```

```
SELECT ad_composite INTO Lob_loc FROM Print_media
WHERE product_id = 2056 AND ad_id = 12001 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;
```

## C (OCI): Writing to the End of (Appending to) a LOB

```
/* Write-appending to a LOB */
/* Select the locator into a locator variable: */
sb4 select_lock_adcomp_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT ad_composite FROM Print_media
                WHERE product_id=2268 AND ad_id = 21001 FOR UPDATE";
    OCIDefine *defnpl;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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;

    /* Allocate the locator: */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                              (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* Select the BLOB: */
    printf(" select and lock an ad-composite locator\n");
    select_lock_adcomp_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;
}

```

## COBOL (Pro\*COBOL): Writing to the End of (Appending to) a LOB

```
* WRITE-APPENDING TO A LOB
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 "SAMP/SAMP".
      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 AD_COMPOSITE INTO :BLOB1
      FROM PRINT_MEDIA
      WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 FOR UPDATE END-EXEC.

* Open the target LOB:
      EXEC SQL LOB OPEN :BLOB1 READ WRITE END-EXEC.

* 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 ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Writing to the End of (Appending to) a LOB

```

/* Write-appending to a LOB */
#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 ad_composite INTO :Lob_loc
        FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001 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;
}
```

## Java (JDBC): Writing to the End of (Append-Write to) a LOB

```
// Write-appending to a LOB
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:
        DriverManager.registerDriver(new 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 ad_composite FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001 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();

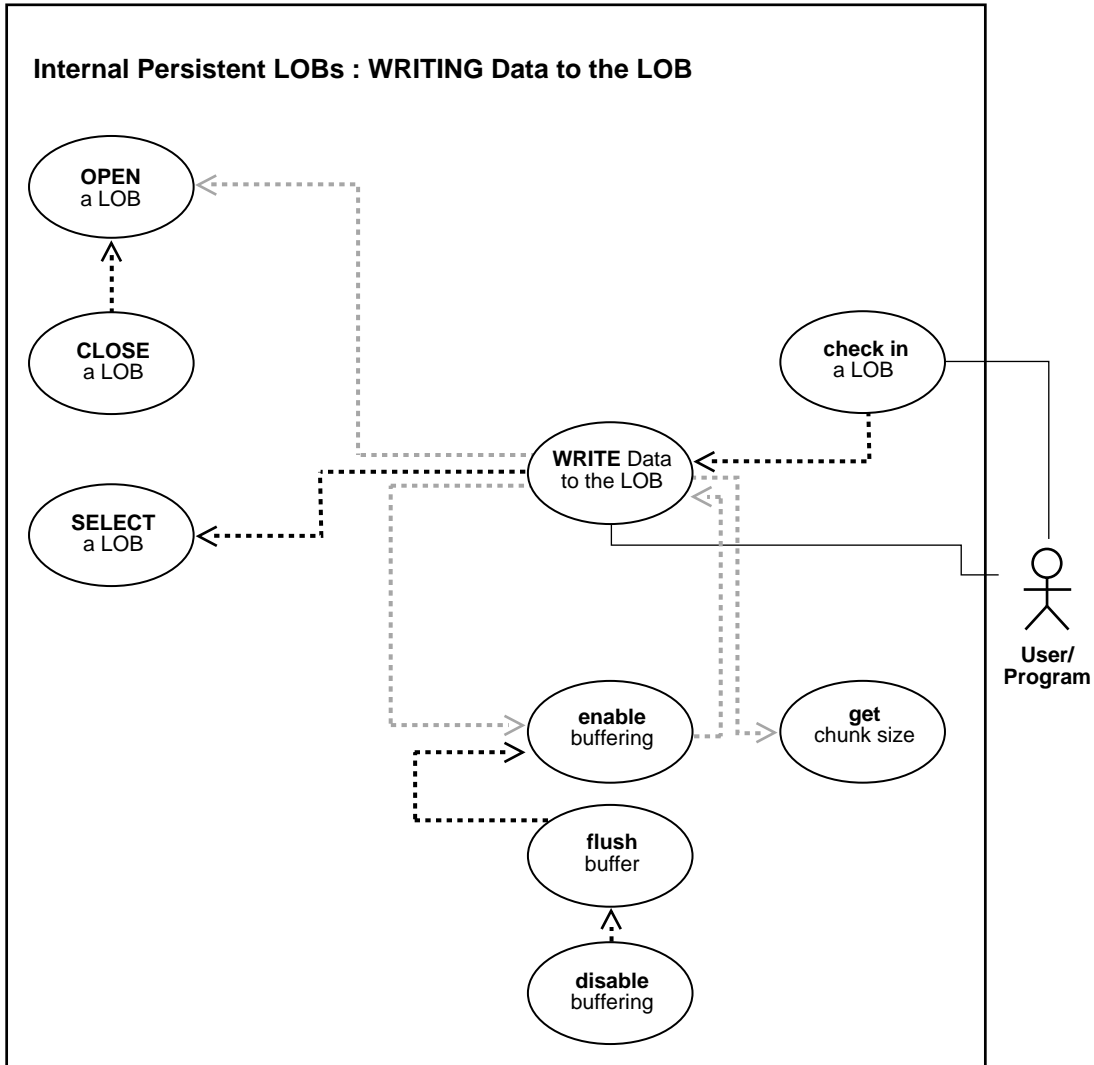
    // Write the contents of the buffer into position pos of the output LOB:
    dest_loc.putBytes(pos, buf);

    // Close all streams and handles:
    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Writing Data to a LOB

Figure 10–29 Use Case Diagram: Writing Data to a LOB





**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to write data to a LOB.

## Usage Notes

**Stream Write** The most efficient way to write large amounts of LOB data is to use `OCILOBWrite()` with the streaming mechanism enabled using 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.

**Use a Multiple of Chunksize to Improve Write Performance.** You will improve performance if 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 using the PL/SQL `DBMS_LOB` package or 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 "[Updating LOBs Via Updated Locators](#)" on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

**Using DBMS\_LOB.WRITE() to Write Data to a BLOB** When you are passing a hexadecimal string to DBMS\_LOB.WRITE() to write data to a BLOB, use the following guidelines:

- The amount parameter should be  $\leq$  the buffer length parameter
- The length of the buffer should be  $((\text{amount} * 2) - 1)$ . This guideline exists because the two characters of the string are seen as one hexadecimal character (and an implicit hexadecimal-to-raw conversion takes place), that is, every two bytes of the string are converted to one raw byte.

The following example is *correct*:

```
declare
  blob_loc BLOB;
  rawbuf RAW(10);
  an_offset INTEGER := 1;
  an_amount BINARY_INTEGER := 10;
begin
  select blob_col into blob_loc from a_table
  where id = 1;
  rawbuf := '1234567890123456789';
  dbms_lob.write(blob_loc, an_amount, an_offset,
  rawbuf);
  commit;
end;
```

Replacing the value for 'an\_amount' in the previous example with the following values, yields error message, ora\_21560:

```
an_amount BINARY_INTEGER := 11;
```

or

```
an_amount BINARY_INTEGER := 19;
```

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — WRITE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobWrite.

- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB WRITE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB WRITE
- Visual Basic (OO4O): (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > write, copyfromfile
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples allow the `ad_sourcetext` data (the advertisement text) to be updated by writing data to the LOB.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Writing Data to a LOB](#) on page 10-215
- [C \(OCI\): Writing Data to a LOB](#) on page 10-217
- [COBOL \(Pro\\*COBOL\): Writing Data to a LOB](#) on page 10-220
- [C/C++ \(Pro\\*C/C++\): Writing Data to a LOB](#) on page 10-222
- [Visual Basic \(OO4O\): Writing Data to a LOB](#) on page 10-225
- [Java \(JDBC\): Writing Data to a LOB](#) on page 10-226

## PL/SQL (DBMS\_LOB Package): Writing Data to a LOB

```

/* Writing data to a LOB */
/* Example procedure writeDataToLOB_proc is not part of DBMS_LOB package. */
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 ad_sourcetext INTO Lob_loc
           FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001 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 ad_sourcetext INTO Lob_loc
           FROM Print_media
           WHERE product_id = 2056 AND ad_id = 12001 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;
```

## C (OCI): Writing Data to a LOB

```

/* Writing data to a LOB.
   Using OCI you can write arbitrary amounts of data
   to an Internal LOB in either a single piece or in multiple pieces using
   streaming with standard polling. A dynamically allocated Buffer
   holds the data being written to the LOB. */

/* Select the locator into a locator variable */
sb4 select_lock_adsourcetext_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt      *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT ad_sourcetext FROM Print_media pm \
                WHERE pm.product_id = 2268 AND ad_id = 21001 FOR UPDATE";
    OCIDefine *defnpl;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NIV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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);
}

#define MAXBUFLLEN 32767
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 descriptors*/
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
        (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

    /* Select the CLOB */
    printf (" select an ad_source_text Lob\n");
    select_lock_adsourcetext_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 > MAXBUFLLEN ) 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 > MAXBUFLLEN)
            nbytes = MAXBUFLLEN;           /* Still have more pieces to go */
        else
        {
            nbytes = remainder;           /* Here, (remainder <= MAXBUFLLEN) */
            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;
}

```

## COBOL (Pro\*COBOL): Writing Data to a LOB

```

* WRITING DATA TO A LOB
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 "SAMP/SAMP".

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 AD_SOURCETEXT INTO :CLOB1 FROM PRINT_MEDIA
WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 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
    ROLLBACK WORK RELEASE
END-EXEC.
STOP RUN.

WRITE-TO-CLOB.
IF ( END-OF-FILE = "N" )
    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.
DISPLAY "END-OF-FILE IS " 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.
```

## C/C++ (Pro\*C/C++): Writing Data to a LOB

```
/* Writing data to a LOB */
```

```

/* This example shows how you can use Pro*C/C++ 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 holds 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 ad_sourcetext INTO Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 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;
}

```

## Visual Basic (OO4O):Writing Data to a LOB

```

'Writing data to a LOB
'There are two ways of writing a lob, with orablob.write or
orablob.copyfromfile

'Using the OraBlob.Write mechanism
Dim OraDyn As OraDynaset, OraAdPhoto As OraBlob, amount_written%, chunksize%,
curchunk() As Byte

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media", ORADYN_DEFAULT)
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value

fnum = FreeFile
Open "c:\tmp\keyboard_3106_13001" For Binary As #fnum

OraAdPhoto.offset = 1
OraAdPhoto.pollingAmount = LOF(fnum)
remainder = LOF(fnum)

Dim piece As Byte
Get #fnum, , curchunk

OraDyn.Edit

piece = ORALOB_FIRST_PIECE

```

```
OraAdPhoto.Write curchunk, chunksize, ORALOB_FIRST_PIECE

While OraAdPhoto.Status = ORALOB_NEED_DATA
  remainder = remainder - chunksize
  If remainder <= chunksize Then
    chunksize = remainder
    piece = ORALOB_LAST_PIECE
  Else
    piece = ORALOB_NEXT_PIECE
  End If

  Get #fnum, , curchunk
  OraAdPhoto.Write curchunk, chunksize, piece

Wend

OraDyn.Update

'Using the OraBlob.CopyFromFile mechanism

Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value

OraDyn.Edit
OraAdPhoto.CopyFromFile "c:\keyboardphoto3106.jpg"
OraDyn.Update
```

## Java (JDBC): Writing Data to a LOB

```
//Writing data to a LOB
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:
        DriverManager.registerDriver(new 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 ad_composite FROM Print_media
                WHERE product_id = 2056 AND ad_id = 12001 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);

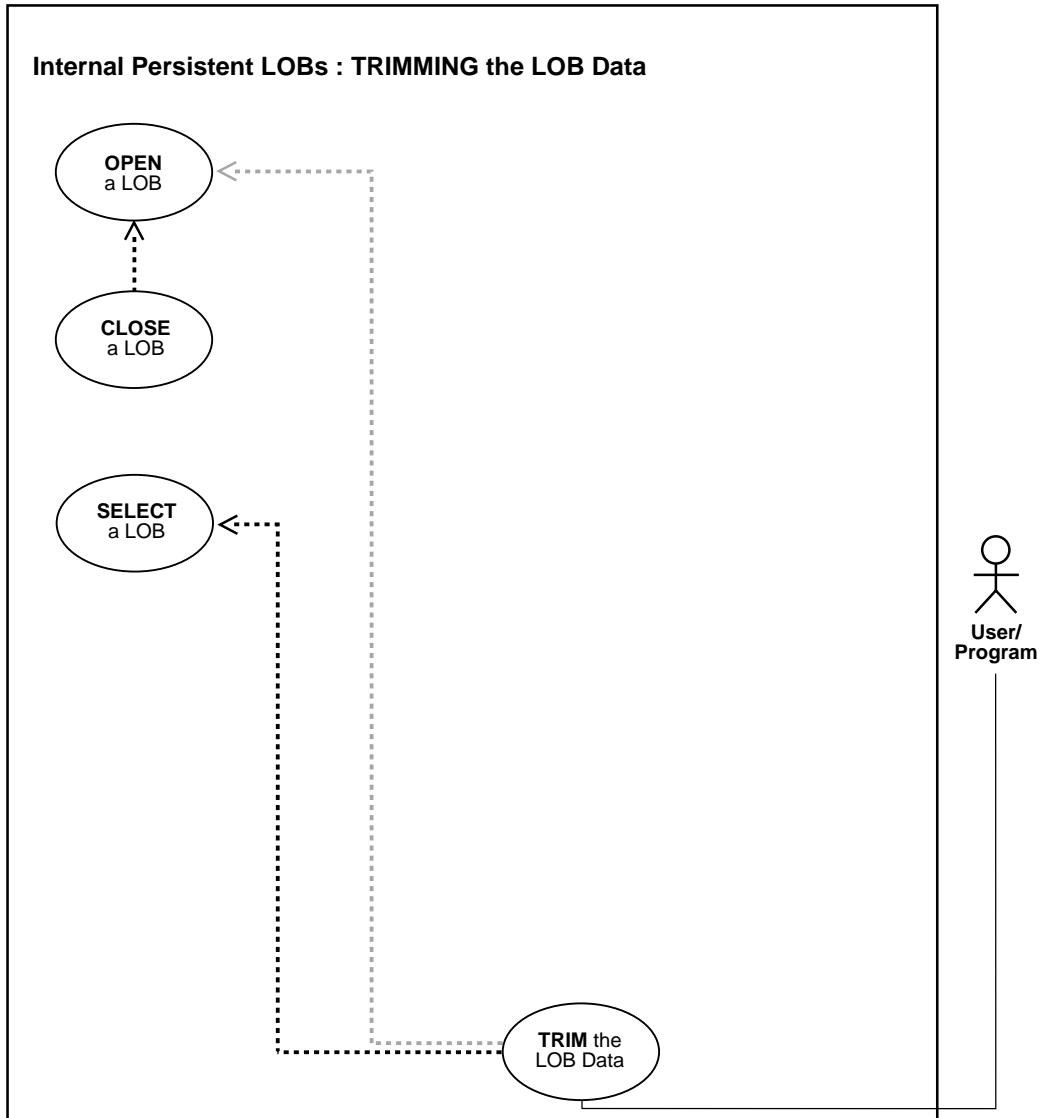
            // Close all streams and handles:
            stmt.close();
            conn.commit();
            conn.close();
        }
    }
}
```

```
    }  
    catch (SQLException e)  
    {  
        e.printStackTrace();  
    }  
}
```



# Trimming LOB Data

Figure 10–30 Use Case Diagram: Trimming LOB Data



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to trim LOB data.

## Usage Notes

**Locking the Row Prior to Updating** Prior to updating a LOB value using the PL/SQL `DBMS_LOB` package, or 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 `SELECT FOR UPDATE` statement in SQL and PL/SQL programs.
- An OCI `pin` or `lock` function in OCI programs.

For more details on the state of the locator after an update, refer to ["Updating LOBs Via Updated Locators"](#) on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — TRIM
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobTrim.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB TRIM.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL and Precompiler Directives" — LOB TRIM
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > trim

- **Java (JDBC):** *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

Unless otherwise noted, these examples access text (CLOB data) referenced in the `ad_finaltext` column of table `Adheader_typ`, and trim it.

## Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Trimming LOB Data](#) on page 10-231
- **C (OCI):** [Trimming LOB Data](#) on page 10-232
- **C++ (OCCI):** No example is provided with this release.
- **COBOL (Pro\*COBOL):** [Trimming LOB Data](#) on page 10-233
- **C/C++ (Pro\*C/C++):** [Trimming LOB Data](#) on page 10-234
- **Visual Basic (OO4O):** [Trimming LOB Data](#) on page 10-236
- **Java (JDBC):** [Trimming LOB Data](#) on page 10-236

## PL/SQL (DBMS\_LOB Package): Trimming LOB Data

```

/* Trimming LOB data */
/* Example procedure trimLOB_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE trimLOB_proc IS
    Lob_loc          CLOB;
BEGIN
    /* Select the LOB, get the LOB locator: */
    SELECT pm.Adheader_typ.ad_finaltext INTO Lob_loc FROM Print_media pm
        WHERE pm.product_id = 2056 AND pm.ad_id = 12001 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
            DEMS_OUTPUT.PUT_LINE('Operation failed');
    END;
```

## C (OCI): Trimming LOB Data

```
/* Trimming LOB data
/* Select the locator into a locator variable */
sb4 select_lock_adfinaltext_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT pm.ad_finaltext \
                FROM Print_media pm WHERE pm.product_id = 2268
                AND ad_id = 21001 FOR UPDATE";
    OCIDefine *defnpl;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NTV_SYNTAX, (ub4) OCI_DEFAULT));
    checkerr (errhp, OCIDefineByPos(stmthp, &defnpl, 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 an ad_finaltext LOB\n");
select_lock_adfinaltext_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 opened it */
checkerr (errhp, OCILobClose(svchp, errhp, Lob_loc));

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

```

## COBOL (Pro\*COBOL): Trimming LOB Data

```

* Trimming LOB data
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 "SAMP/SAMP".
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 PM.AD_SOURCETEXT INTO :CLOB1
    FROM PRINT_MEDIA PM
    WHERE PM.PRODUCT_ID = 3060
    AND AD_ID = 11001 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 ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Trimming LOB Data

```
/* Trimming LOB data */
#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 PMtab.ad_sourctext INTO :vt_ref
    FROM Print_media PMtab
    WHERE PMtab.product_id = 3060 AND ad_id = 11001 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;
}
```

## Visual Basic (OO4O): Trimming LOB Data

```
'Trimming LOB data
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("examplecdb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value

OraDyn.Edit
OraAdPhotol.Trim 10
OraDyn.Update
```

## Java (JDBC): Trimming LOB Data

```
// Trimming BLOBs and CLOBs.
// You need to import the java.sql package to use JDBC
import java.sql.*;

// You need to import the oracle.sql package to use oracle.sql.BLOB
import oracle.sql.*;
```



```
class TrimLob
{
    public static void main (String args [])
        throws SQLException
    {
        // Load the Oracle JDBC driver
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        String url = "jdbc:oracle:oci8:@";
        try {
            String url1 = System.getProperty("JDBC_URL");
            if (url1 != null)
                url = url1;
        } catch (Exception e) {
            // If there is any security exception, ignore it
            // and use the default
        }

        // Connect to the database
        Connection conn =
            DriverManager.getConnection (url, "pm", "pm");
        // It's faster when auto commit is off
        conn.setAutoCommit (false);

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

        try
        {
            stmt.execute ("drop table basic_lob_table");
        }
        catch (SQLException e)
        {
            // An exception could be raised here if the table did not exist already.
        }

        // Create a table containing a BLOB and a CLOB
        stmt.execute ("create table basic_lob_table (x varchar2 (30), b blob, c
        clob)");

        // Populate the table
        stmt.execute ("insert into basic_lob_table values ('one',
        '010101010101010101010101010101', 'onetwothreefour')");
    }
}
```

```
// Select the lob
ResultSet rset = stmt.executeQuery ("select * from basic_lob_table");
while (rset.next ())
{
    // Get the lob
    BLOB blob = (BLOB) rset.getObject (2);
    CLOB clob = (CLOB) rset.getObject (3);

    // Show the original lob length
    System.out.println ("Open the lob");
    System.out.println ("blob.length()="+blob.length());
    System.out.println ("clob.length()="+clob.length());

    // Trim the lob
    System.out.println ("Trim the lob to length = 6");
    blob.trim (6);
    clob.trim (6);

    // Show the lob length after trim()
    System.out.println ("Open the lob");
    System.out.println ("blob.length()="+blob.length());
    System.out.println ("clob.length()="+clob.length());
}

// Close the ResultSet
rset.close ();

// Close the Statement
stmt.close ();

// Close the connection
conn.close ();
}
}
```

Here is the old way of trimming LOB data, using `DBMS_LOB.trim`:

```
// Trimming LOB data
// 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:
        DriverManager.registerDriver(new 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 pm.ad_finaltext FROM Print_media pm
                WHERE pm.product_id = 2056 AND ad_id = 12001 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();

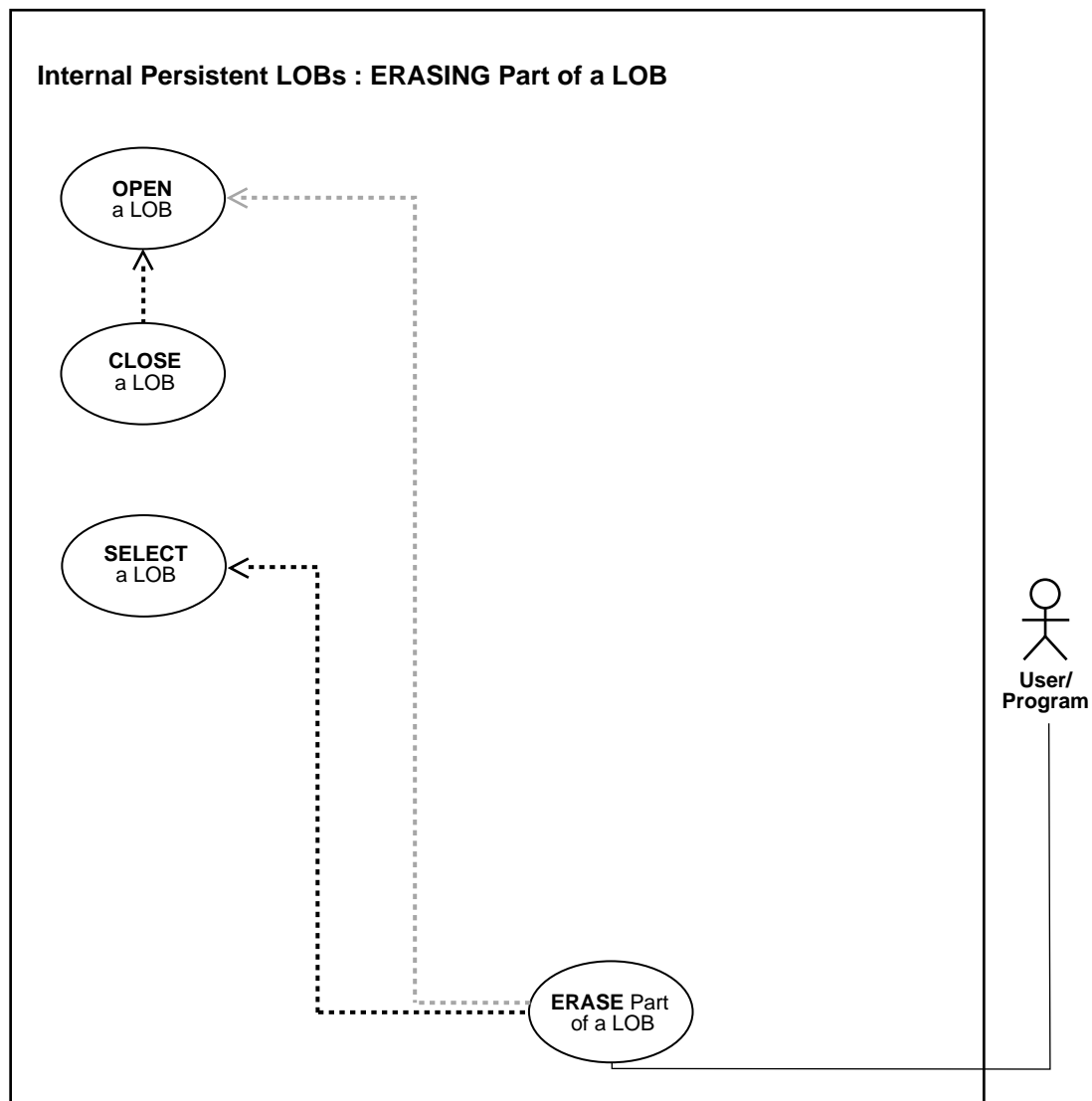
        // Trim the LOB to length of 400:
        cstmt = (OracleCallableStatement)
            conn.prepareCall ("BEGIN DBMS_LOB.TRIM(?, 400); END;");
        cstmt.setCLOB(1, lob_loc);
        cstmt.execute();

        // 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();
    }
}
}
```

## Erasing Part of a LOB

Figure 10-31 Use Case Diagram: Erasing Part of a LOB



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to erase part of a LOB.

## Usage Notes

**Locking the Row Prior to Updating** Prior to updating a LOB value using the PL/SQL `DBMS_LOB` package or OCI, you must lock the row containing the LOB. While `INSERT` and `UPDATE` statements implicitly lock the row, locking is done explicitly by means of a `SELECT FOR UPDATE` statement in SQL and PL/SQL programs, or by using the `OCI pin` or `lock` function in OCI programs.

For more details on the state of the locator after an update, refer to ["Updating LOBs Via Updated Locators"](#) on page 5-5 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBM\_LOB Package): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — ERASE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobErase.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB ERASE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL and Precompiler Directives" — LOB ERASE
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > erase
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i*

*SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The examples demonstrate erasing a portion of the image (ad\_photo).

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Erasing Part of a LOB](#) on page 10-243
- [C \(OCI\): Erasing Part of a LOB](#) on page 10-244
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Erasing Part of a LOB](#) on page 10-245
- [C/C++ \(Pro\\*C/C++\): Erasing Part of a LOB](#) on page 10-246
- [Visual Basic \(OO4O\): Erasing Part of a LOB](#) on page 10-247
- [Java \(JDBC\): Erasing Part of a LOB](#) on page 10-248

## PL/SQL (DBMS\_LOB Package): Erasing Part of a LOB

```

/* Erasing part of a LOB.
   Example procedure eraseLOB_proc is not part of 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 ad_photo INTO lob_loc FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001 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;
```

## C (OCI): Erasing Part of a LOB

```
/* Erasing part of a LOB (persistent LOBs)
   Select the locator into a locator variable: */
sb4 select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT ad_photo FROM Print_media
                WHERE product_id=3060 AND ad_id = 11001 FOR UPDATE";
    OCIDefine *defnp1, *defnp2;

    checkerr (errhp, OCISstmtPrepare(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)
              || OCIDefineByPos(stmthp, &defnp2, errhp, (ub4) 2,
                                     (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, OCISstmtExecute(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;
OCISstmt    *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 an ad_photo LOB\n");
select_lock_adphoto_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 ad_photo 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;
}

```

## COBOL (Pro\*COBOL): Erasing Part of a LOB

```

* ERASING PART OF A LOB
  IDENTIFICATION DIVISION.
  PROGRAM-ID. ERASE-BLOB.
  ENVIRONMENT DIVISION.
  DATA DIVISION.
  WORKING-STORAGE SECTION.

  01  USERID    PIC X(11) VALUES "SAMP/SAMP".
  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 AD_PHOTO INTO :BLOB1
    FROM PRINT_MEDIA PM
    WHERE PM.PRODUCT_ID = 2268 AND AD_ID = 21001 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 ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Erasing Part of a LOB

```
/* Erasing part of a LOB */
#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 ad_composite INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060 AND ad_id = 11001 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;
}

```

## Visual Basic (OO4O): Erasing Part of a LOB

```

'Erasing part of a LOB
Dim MySession As OraSession

```

```
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhoto1 As OraBlob, OraAdPhotoClone As OraBlob

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)

Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media ORDER BY product_
id", ORADYN_DEFAULT)
Set OraAdPhoto1 = OraDyn.Fields("ad_photo").Value
'Erase 10 bytes beginning from the 100th byte:
OraDyn.Edit
OraAdPhoto1.Erase 10, 100
OraDyn.Update
```

## Java (JDBC): Erasing Part of a LOB

```
// Erasing part of a LOB
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:
        DriverManager.registerDriver(new 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 ad_photo FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001 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);

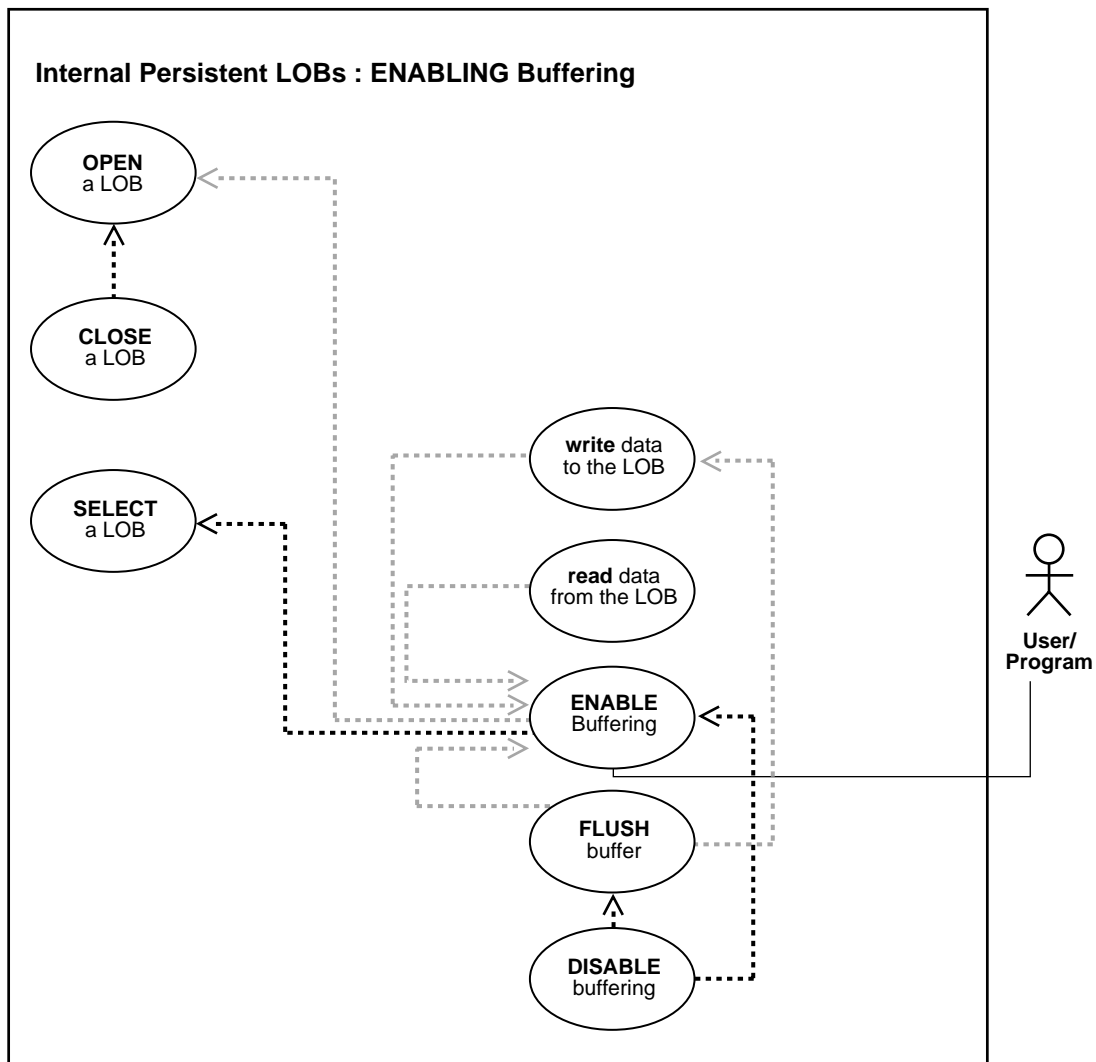
// Close the LOB:
    cstmt = (OracleCallableStatement) conn.prepareCall (
        "BEGIN DBMS_LOB.CLOSE(?); END;");
    cstmt.setBLOB(1, lob_loc);
    cstmt.execute();

    conn.commit();
    stmt.close();
}
```

```
        cstmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

# Enabling LOB Buffering

Figure 10-32 Use Case Diagram: Enabling LOB Buffering



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to enable LOB buffering.

## Usage Notes

Enable buffering when performing a small read or write of data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations.

---

---

### Note:

- You must flush the buffer in order to make your modifications persistent.
  - Do not enable buffering for the stream read and write involved in checkin and checkout.
- 
- 

For more information, refer to ["LOB Buffering Subsystem"](#) on page 5-18 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB ENABLE BUFFERING.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL and Precompiler Directives" — LOB ENABLE BUFFERING



- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > EnableBuffering
- Java (JDBC): There is no applicable syntax reference for this use case.

### Scenario

This scenario is part of the management of a buffering example related to `ad_photo` that is developed in this and related methods.

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB): No example is provided with this release.
- C (OCI): No example is provided with this release.
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Erasing Part of a LOB](#) on page 10-245
- [C/C++ \(Pro\\*C/C++\): Erasing Part of a LOB](#) on page 10-246
- [Visual Basic \(OO4O\): Erasing Part of a LOB](#) on page 10-247

## C (OCI): Enabling LOB Buffering

**See:** This is executed in a similar way to that described in the example, "[Disabling LOB Buffering](#)" on page 10-263.

## COBOL (Pro\*COBOL): Enabling LOB Buffering

```
* ENABLING LOB BUFFERING
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 ad_photo INTO :BLOB1
          FROM PRINT_MEDIA
          WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
          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 ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Enabling LOB Buffering

```

/* Enabling LOB buffering
#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 ad_composite INTO :Lob_loc
        FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001 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;
}

```

## Visual Basic (OO40): Enabling LOB Buffering

```

'Enabling LOB buffering (persistent LOBs)
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

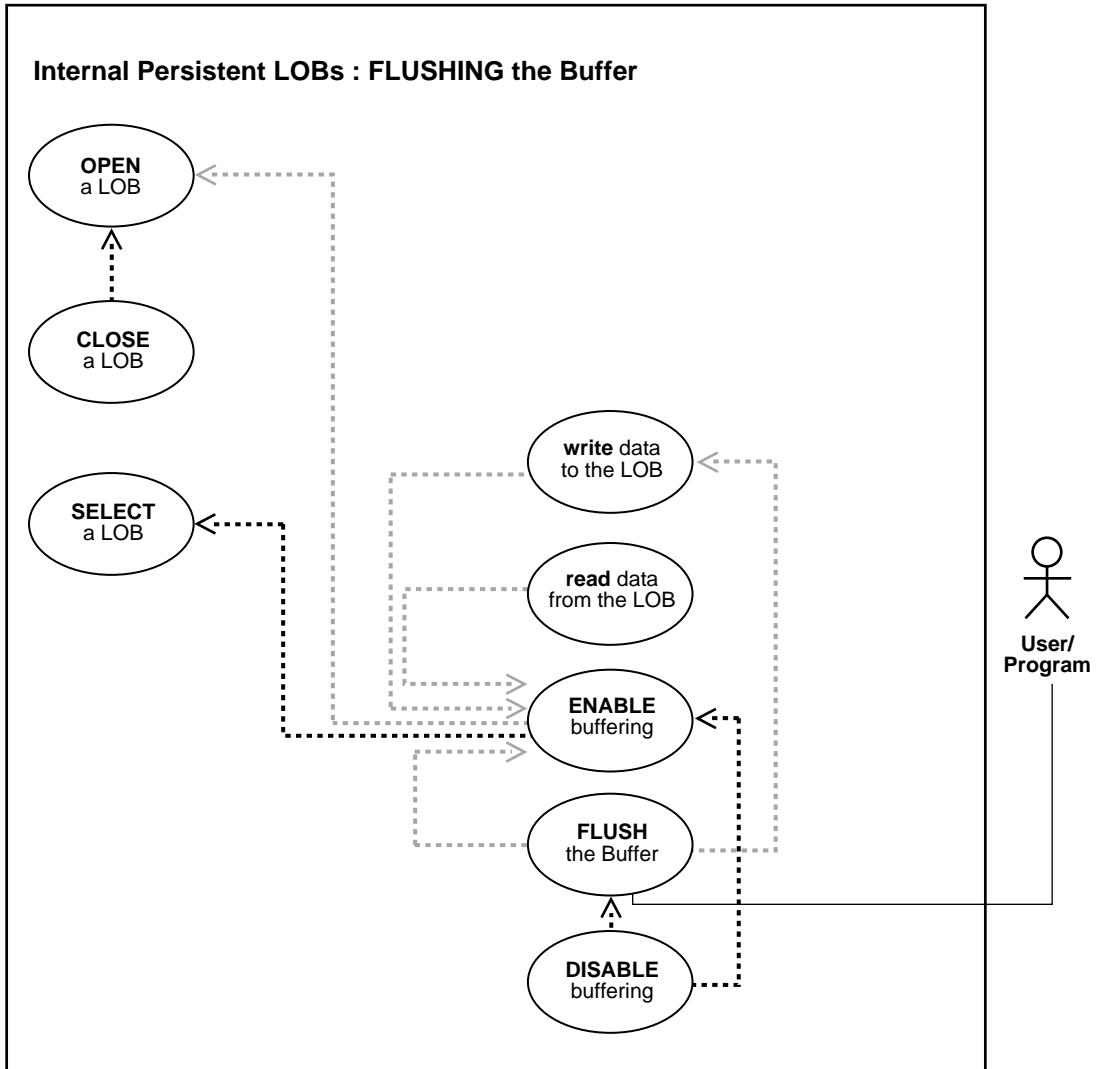
Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id", ORADYN_DEFAULT)
Set OraAdPhotol = OraDyn.Fields("ad_photo").Value

'Enable buffering:
OraAdPhotol.EnableBuffering

```

# Flushing the Buffer

Figure 10–33 Use Case Diagram: Flushing the Buffer



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to flush the LOB buffer.

## Usage Notes

Enable buffering when performing a small read or write of data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations.

---

---

### Notes:

- You must flush the buffer in order to make your modifications persistent.
  - Do not enable buffering for the stream read and write involved in checkin and checkout.
- 
- 

For more information, refer to ["LOB Buffering Subsystem"](#) on page 5-18 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB FLUSH BUFFER.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB FLUSH BUFFER.

- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > FlushBuffer.
- Java (JDBC): There is no applicable syntax reference for this use case.

### Scenario

This scenario is part of the management of a buffering example related to `ad_photo` that is developed in this and related methods.

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- [C \(OCI\): Flushing the Buffer](#) on page 10-259
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Flushing the Buffer](#) on page 10-259
- [C/C++ \(Pro\\*C/C++\): Flushing the Buffer](#) on page 10-261
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Flushing the Buffer

**See:** Flushing the Buffer in OCI is executed in a similar fashion to that described in the example, "[Disabling LOB Buffering](#)" on page 10-263.

## COBOL (Pro\*COBOL): Flushing the Buffer

```
* Flushing the LOB buffer (persistent LOBs)
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_PHOTO INTO :BLOB1
       FROM PRINT_MEDIA
       WHERE PRODUCT_ID = 2056 AND AD_ID = 12001 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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLEERROR CONTINUE END-EXEC.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED:".
DISPLAY " ".
DISPLAY SQLEERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

```

## C/C++ (Pro\*C/C++): Flushing the Buffer

```

/* Flushing the LOB Buffer (persistent LOBs)
#include <oci.h>
#include <stdio.h>
#include <string.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 SQLEERROR 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;
}
```

## Visual Basic (OO4O): Flushing the Buffer

---

---

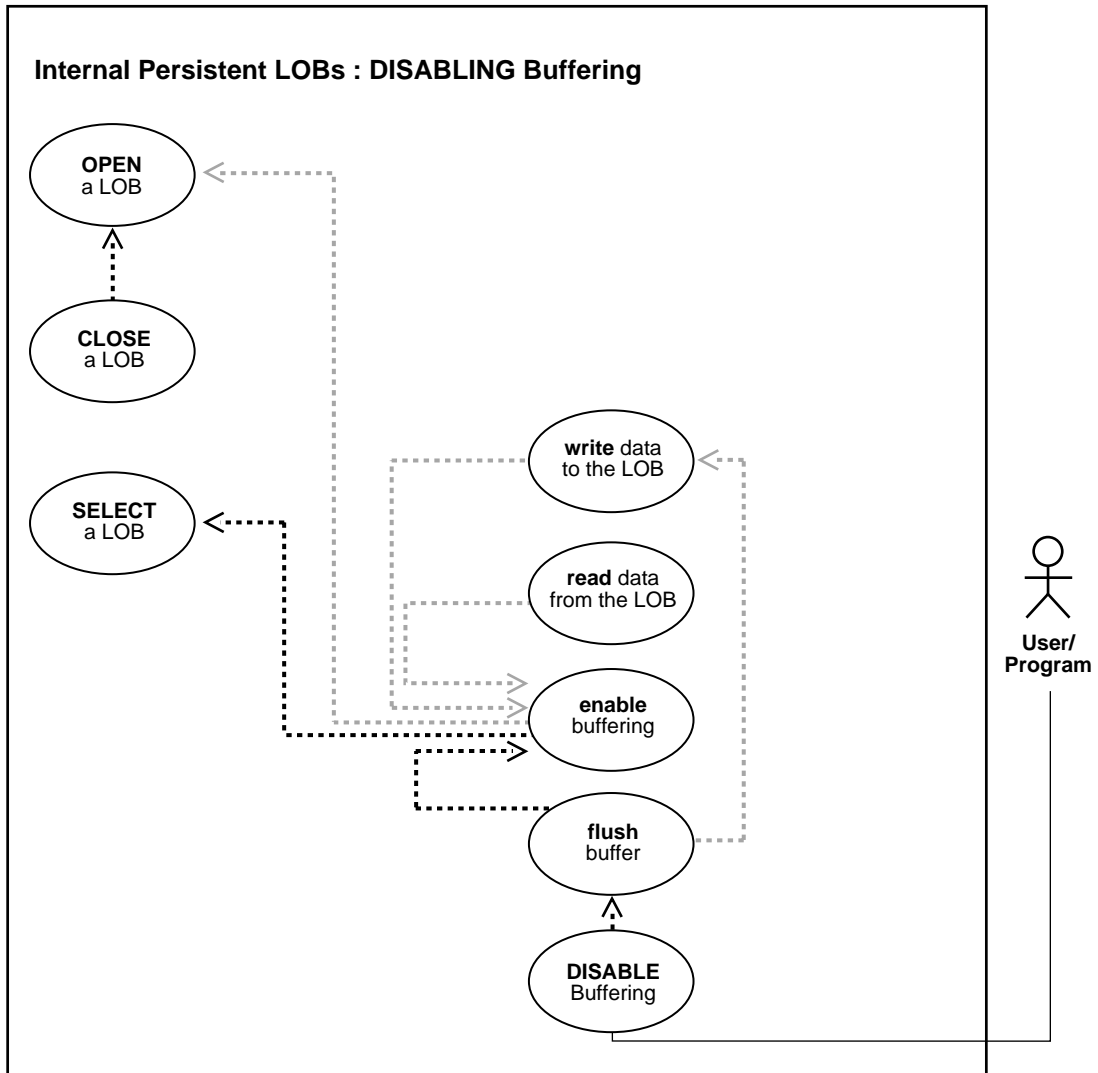
**Note:** A Visual Basic (OO4O) example will be made available in a subsequent release.

---

---

## Disabling LOB Buffering

Figure 10–34 Use Case Diagram: Disabling LOB Buffering



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

## Purpose

This procedure describes how to disable LOB buffering.

## Usage Notes

Enable buffering when performing a small read or write of data. Once you have completed these tasks, you must disable buffering before you can continue with any other LOB operations.

---

---

### Note:

- You must flush the buffer in order to make your modifications persistent.
  - Do not enable buffering for the stream read and write involved in checkin and checkout.
- 
- 

For more information, refer to ["LOB Buffering Subsystem"](#) on page 5-18 in [Chapter 5, "Large Objects: Advanced Topics"](#).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB Package): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCIFlushBuffer
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DISABLE BUFFER.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DISABLE BUFFER

- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Orablob > METHODS > DisableBuffering
- Java (JDBC): There is no applicable syntax reference for this use case.

## Scenario

This scenario is part of the management of a buffering example related to `ad_photo` that is developed in this and related methods.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- [C \(OCI\): Disabling LOB Buffering](#) on page 10-265
- C++ (OCCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Disabling LOB Buffering](#) on page 10-267
- [C/C++ \(Pro\\*C/C++\): Disabling LOB Buffering](#) on page 10-269
- [Visual Basic \(OO4O\): Disabling LOB Buffering](#) on page 10-270
- Java (JDBC): No example is provided with this release.

## C (OCI): Disabling LOB Buffering

```

/* Disabling LOB buffering (persistent LOBs) */
/* Select the locator into a locator variable: */
sb4 select_lock_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    text *sqlstmt =
        (text *)"SELECT ad_photo FROM Print_media
                WHERE product_id=3060 FOR UPDATE";
    OCIDefine *defnp1;
    checkerr (errhp, OCIStmtPrepare(stmthp, errhp, sqlstmt,
                                    (ub4)strlen((char *)sqlstmt),
                                    (ub4) OCI_NIV_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, OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                               (CONST OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                               (ub4) OCI_DEFAULT));

    return 0;
}
#define MAXBUFLLEN 32767
void lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{
    OCILobLocator *Lob_loc;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen;

/* Allocate the locator descriptor: */
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &Lob_loc,
                          (ub4)OCI_DTYPE_LOB, (size_t) 0, (dvoid **) 0);

/* Select the BLOB: */
printf (" select an ad_photo Lob\n");
select_lock_adphoto_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;
}

```

## COBOL (Pro\*COBOL): Disabling LOB Buffering

```

* DISABLING LOB BUFFERING (PERSISTENT LOBS)
IDENTIFICATION DIVISION.
PROGRAM-ID. LOB-BUFFERING.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Disabling LOB Buffering

```

/* Disabling LOB buffering (persistent LOBs) */
#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 ad_photo INTO :Lob_loc
        FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001 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;
}

```

## Visual Basic (OO4O): Disabling LOB Buffering

```

'Disabling LOB buffering (persistent LOBs)
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraDyn As OraDynaset, OraAdPhotol As OraBlob, OraAdPhotoClone As OraBlob

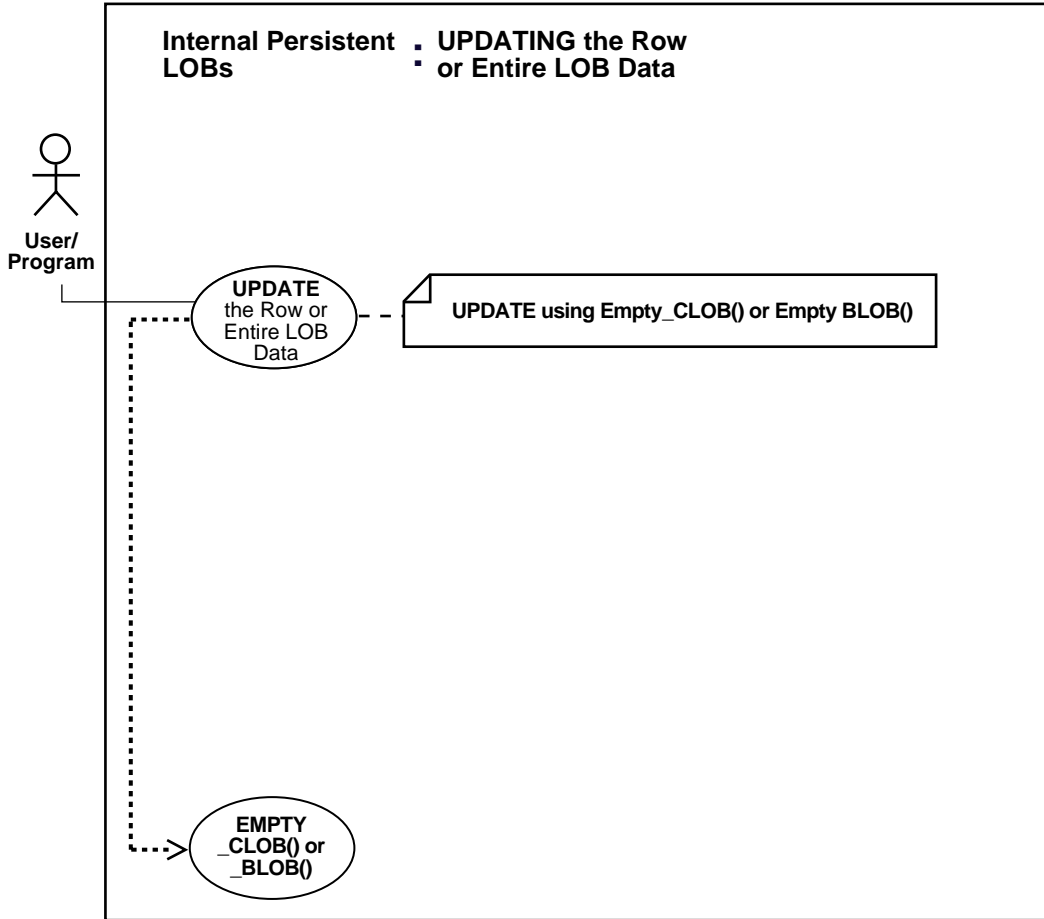
Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("exampledb", "samp/samp", 0&)
Set OraDyn = OraDb.CreateDynaset(
    "SELECT * FROM Print_media ORDER BY product_id, ad_id", ORADYN_DEFAULT)

Set OraAdPhotol = OraDyn.Fields("ad_photo").Value
'Disable buffering:
OraAdPhotol.DisableBuffering

```

## Updating a LOB with EMPTY\_CLOB() or EMPTY\_BLOB()

Figure 10–35 Use Case Diagram: Updating a LOB with EMPTY\_CLOB() or EMPTY\_BLOB()



**See Also:**

- [Table 10-1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Updating a Row by Selecting a LOB From Another Table](#) on page 10-276
- [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-278

**For Binds of More Than 4,000 Bytes**

For information on how to UPDATE a LOB when binds of more than 4,000 bytes are involved, see the following sections in [Chapter 7, "Modeling and Design"](#):

- [Binds Greater than 4,000 Bytes are Now Allowed For LOB INSERTs and UPDATEs](#) on page 7-14
- [Binds of More Than 4,000 Bytes ... No HEX to RAW or RAW to HEX Conversion](#) on page 7-15
- [Example: PL/SQL - Using Binds of More Than 4,000 Bytes in INSERT and UPDATE](#) on page 7-16
- [Example: PL/SQL - Binds of More Than 4,000 Bytes -- Inserts Not Supported](#) on page 7-18
- [Example: PL/SQL - 4,000 Byte Result Limit in Binds of More than 4,000 Bytes](#) on page 7-18
- [Example: C \(OCI\) - Binds of More than 4,000 Bytes For INSERT and UPDATE](#) on page 7-19

---

---

**Note:** Performance improves when you update the LOB with the actual value, instead of using EMPTY\_CLOB() or EMPTY\_BLOB().

---

---

**Purpose**

This procedure describes how to UPDATE a LOB with EMPTY\_CLOB() or EMPTY\_BLOB().

## Usage Notes

**Making a LOB Column Non-Null** Before you write data to an internal LOB, make the LOB column non-null; that is, the LOB column 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 also initialize a LOB column with a character or raw string less than 4,000 bytes in size. For example:

```
UPDATE Print_media
   SET ad_sourcetext = 'This is a One Line Story'
   WHERE product_id = 2268;
```

You can perform this initialization during `CREATE TABLE` (see ["Creating a Table Containing One or More LOB Columns"](#)) or, as in this case, by means of an `INSERT`.

## Syntax

Use the following syntax reference:

- [SQL: Oracle9i SQL Reference Chapter 7, "SQL Statements" — UPDATE](#)

## Scenario

The following example shows a series of updates using the `EMPTY_CLOB` operation to different data types.

## Examples

The example is provided in SQL and applies to all the programmatic environments:

- [SQL: UPDATE a LOB with EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)](#)

## SQL: UPDATE a LOB with EMPTY\_CLOB() or EMPTY\_BLOB()

```
UPDATE Print_media SET ad_sourcetext = EMPTY_CLOB()
   WHERE product_id = 3060 AND ad_id = 11001;
```

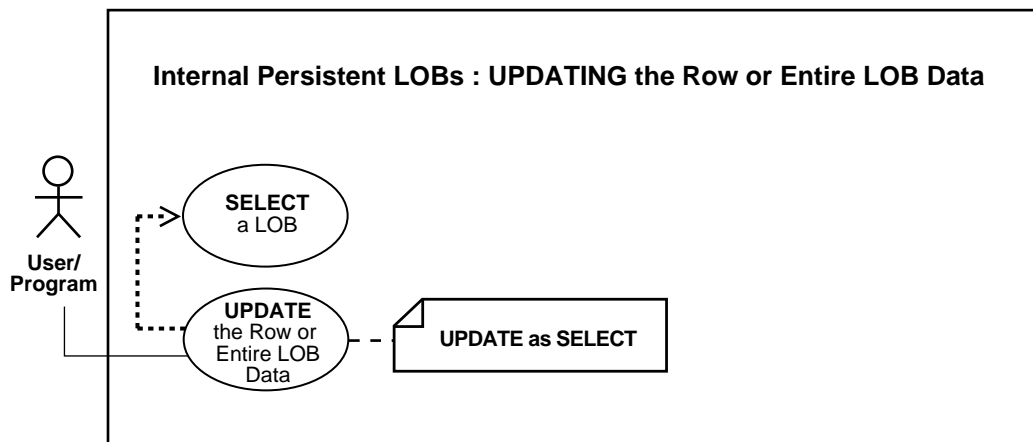
```
UPDATE Print_media SET ad_fltextn = EMPTY_CLOB()
   WHERE product_id = 3060 AND ad_id = 11001;
```

```
UPDATE Print_media SET ad_photo = EMPTY_BLOB()
   WHERE product_id = 3060 AND ad_id = 11001;
```



## Updating a Row by Selecting a LOB From Another Table

Figure 10–36 Use Case Diagram: Updating a Row by Selecting a LOB from Another Table



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Updating a LOB with EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)](#) on page 10-273
- [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-278

### Purpose

This procedure describes how to use UPDATE as SELECT with LOBs.

### Usage Notes

Not applicable.

### Syntax

Use the following syntax reference:

- [SQL: Oracle9i SQL Reference](#) , Chapter 7, "SQL Statements" — UPDATE

### Scenario

This example updates data from `online_media` by means of a reference:

### Examples

The SQL example provided applies to all programmatic environments:

- [SQL: Update a Row by Selecting a LOB From Another Table](#)

## SQL: Update a Row by Selecting a LOB From Another Table

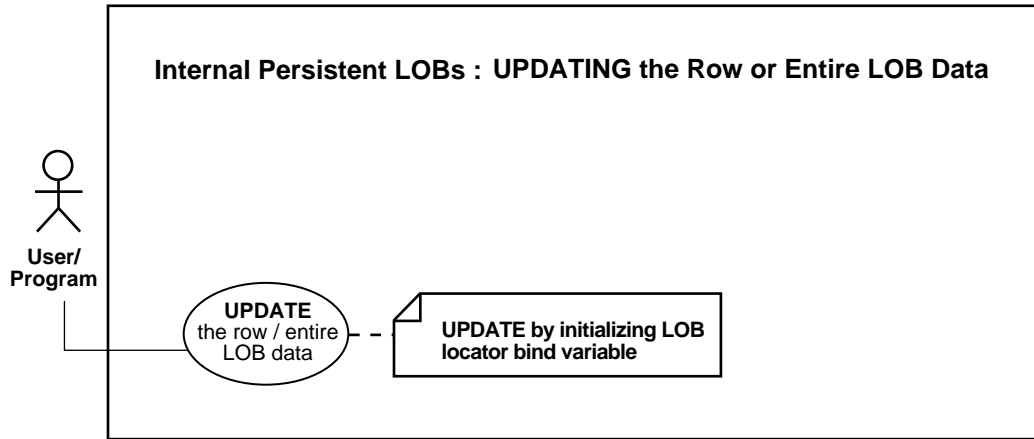
Rem Updating a row by selecting a LOB from another table (persistent LOBs)

```
UPDATE Print_media SET ad_sourcetext =  
  (SELECT * product_text FROM online_media WHERE product_id = 3060);  
  WHERE product_id = 3060 AND ad_id = 11001;
```



## Updating by Initializing a LOB Locator Bind Variable

Figure 10–37 Use Case Diagram: Updating by Initializing a LOB Locator Bind Variable



### See Also:

- [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2
- [Updating a LOB with EMPTY\\_CLOB\(\) or EMPTY\\_BLOB\(\)](#) on page 10-273
- [Updating a Row by Selecting a LOB From Another Table](#) on page 10-276

### Purpose

This procedure describes how to UPDATE by initializing a LOB locator bind variable.

### Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL: *Oracle9i SQL Reference*, Chapter 7, "SQL Statements" — UPDATE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions.
- C++ (OCCI): *Oracle C++ Call Interface Programmer's Guide*
- COBOL (Pro\*COBOL): *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — ALLOCATE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — ALLOCATE.
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > Oradatabase > METHODS > ExecuteSQL
- Java (JDBC): *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples update `ad_photo` data by means of a locator bind variable.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL: [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-278
- C (OCI): [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-279
- C++ (OCCI): No example is provided with this release.
- COBOL (Pro\*COBOL): [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-281
- C/C++ (Pro\*C/C++): [Updating by Initializing a LOB Locator Bind Variable](#) on page 10-282

- [Visual Basic \(OO4O\): Updating by Initializing a LOB Locator Bind Variable](#) on page 10-283
- [Java \(JDBC\): Updating by Initializing a LOB Locator Bind Variable](#) on page 10-284

## PL/SQL: Updating by Initializing a LOB Locator Bind Variable

```

/* Updating a LOB by initializing a LOB locator bind variable */
/* Example procedure updateUseBindVariable_proc is not part of
   DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (Lob_loc BLOB) IS
BEGIN
    UPDATE Print_media SET ad_photo = lob_loc
        WHERE product_id = 3060 AND ad_id = 11001;
END;

DECLARE
    Lob_loc    BLOB;
BEGIN
    /* Select the LOB: */
    SELECT ad_photo INTO Lob_loc
        FROM Print_media
            WHERE product-id = 3060 AND ad_id = 11001;
    updateUseBindVariable_proc (Lob_loc);
    COMMIT;
END;

```

## C (OCI): Updating by Initializing a LOB Locator Bind Variable

```

/* Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)
*/
/* Select the locator into a locator variable: */
sb4 select_adphoto_locator(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    text *sqlstmt =
        (text *) "SELECT ad_photo FROM Print_media
                WHERE product_id=2268";
    OCIDefine *defnpl;

```

```

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 Print_media SET ad_photo = :1 WHERE product_id = 3106";
    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 an ad_photo locator\n");
    (void)select_adphoto_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;
}

```

## COBOL (Pro\*COBOL): Updating by Initializing a LOB Locator Bind Variable

```

* Updating a LOB by initializing a LOB locator bind variable
* [Example script: 3806.pco]
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 "SAMP/SAMP".
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 AD_PHOTO INTO :BLOB1
    FROM PRINT_MEDIA
    WHERE PRODUCT_ID = 2056 AND AD_ID = 12001
END-EXEC.

EXEC SQL
    UPDATE PRINT_MEDIA
    SET AD_PHOTO = :BLOB1
    WHERE PRODUCT_ID = 2268 AND AD_ID = 21001
END-EXEC.

END-OF-BLOB.

EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL FREE :BLOB1 END-EXEC.
EXEC SQL ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Updating by Initializing a LOB Locator Bind Variable

```
/* Updating a LOB by initializing a LOB locator bind variable (persistent
LOBs)*/
#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 Print_media SET ad_photo = :Lob_loc WHERE product_id = 2268;
}

void updateLOB_proc()
{
    OCIBlobLocator *Lob_loc;

    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT ad_photo INTO :Lob_loc
        FROM Print_media WHERE product_id = 3060;
    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;
}

```

## Visual Basic (OO4O): Updating by Initializing a LOB Locator Bind Variable

```

'Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)
Dim OraDyn As OraDynaset, OraAdPhoto as OraBlob

```

```

'Select a column with product_id = 3106:
Set OraDyn = OraDb.CreateDynaset("SELECT * FROM Print_media WHERE
    product_id= 3106", ORADYN_DEFAULT)

```

```

'Get the OraBlob object from the field:
Set OraAdPhoto = OraDyn.Fields("AD_PHOTO").Value

```

```

'Create a parameter for OraBlob object:
OraDb.Parameters.Add "AD_PHOTO", Null, ORAPARM_INPUT, ORATYPE_BLOB

```

```

'Set the value of ad_photo parameter to OraAdPhoto:
OraDb.Parameters("AD_PHOTO").Value = OraAdPhoto

```

```
'Update table Print_media with OraAdPhoto for product_id = 2268:
OraDb.ExecuteSQL("Update Print_media SET ad_photo = :AD_PHOTO
WHERE product_id = 2268")
```

## Java (JDBC): Updating by Initializing a LOB Locator Bind Variable

```
// Updating a LOB by initializing a LOB locator bind variable (persistent LOBs)
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:
        DriverManager.registerDriver(new 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);

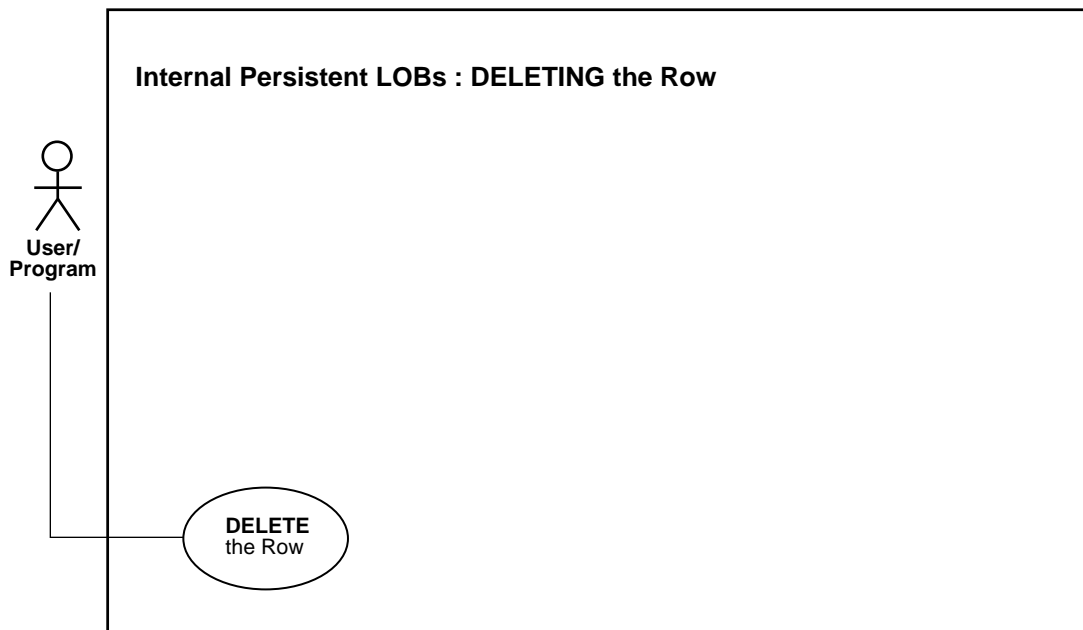
        // Create a Statement:
        Statement stmt = conn.createStatement ();
        try
        {
            ResultSet rset = stmt.executeQuery (
                "SELECT ad_photo FROM Print_media
                WHERE product_id = 3106 AND ad_id = 13001");
            if (rset.next())
            {
                // retrieve the LOB locator from the ResultSet:
                BLOB photo_blob = ((OracleResultSet)rset).getBLOB (1);
            }
        }
    }
}
```



```
OraclePreparedStatement ops =
    (OraclePreparedStatement) conn.prepareStatement(
        "UPDATE Print_media SET ad_photo = ?
         WHERE product_id = 2056 AND ad_id = 12001");
ops.setBlob(1, photo_blob);
ops.execute();
rset.close();
stmt.close();
conn.commit();
conn.close();
}
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Deleting the Row of a Table Containing a LOB

*Figure 10–38 Use Case Diagram: Deleting the Row of a Table Containing a LOB*



**See Also:** [Table 10–1, "Internal Persistent LOB Basic Operations"](#) on page 10-2

### **Purpose**

This procedure describes how to delete the row of a table containing a LOB.

### **Usage Notes**

To delete a row that contains an internal LOB column or attribute use one of the following commands

- SQL DML: DELETE
- SQL DDL that effectively deletes it:
  - DROP TABLE

- TRUNCATE TABLE
- DROP TABLESPACE.

In either case you delete the LOB locator *and the LOB value as well*.

---

---

**Note:** 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. This is an advanced topic. It is discussed in more detail with regard to "[Read Consistent Locators](#)" on page 5-1.

---

---

**Distinct LOB Locators for Distinct Rows** 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.

### Syntax

Use the following syntax reference:

- SQL: *Oracle9i SQL Reference*, Chapter 7, "SQL Statements" — DELETE, DROP TABLE, TRUNCATE TABLE

### Scenario

In the following example, all data associated with the product having `product_id = 3060 AND ad_id 11001` is deleted.

### Examples

The SQL example provide applies to all programmatic environments:

- [SQL: Delete a LOB](#) on page 10-287

## SQL: Delete a LOB

```
DELETE FROM Print_media WHERE product_id = 3060 AND ad_id = 11001;
TRUNCATE TABLE Print_media;
DROP TABLE Print_media;
```



---

## Temporary LOBs

### Use Case Model

In this chapter we discuss each operation on a Temporary LOB (such as "Loading a Temporary LOB with Data from a BFILE") in terms of a use case. [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) lists all the use cases.

### Individual Use Cases

Each Internal Persistent LOB use case is described as follows:

- *Use case figure.* Depicts the use case. For help in understanding the use case diagrams, see [Appendix A, "Unified Modeling Language Diagrams"](#).
- *Purpose.* The purpose of this use case with regards to LOBs.
- *Usage Notes.* Guidelines to assist your implementation of the LOB operation.
- *Syntax.* Pointers to the syntax in different programmatic environments that underlies the LOBs related activity for the use case.
- *Scenario.* Describes the implementation of the use case in terms of the sample schema used in the examples. For further details on sample schemas, refer to *Oracle9i Sample Schemas*.
- *Examples.* How to apply each use case based on the sample schema used.

## Use Case Model: Internal Temporary LOBs

[Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) indicates with + where examples are provided for specific use cases and in which programmatic environment (see [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a complete discussion and references to related manuals).

We refer to programmatic environments by means of the following abbreviations:

- **P** — PL/SQL using the DBMS\_LOB Package
- **O** — C using OCI (Oracle Call Interface)
- **B** — COBOL using Pro\*COBOL precompiler
- **C** — C/C++ using Pro\*C/C++ precompiler
- **V** — Visual Basic using OO4O (Oracle Objects for OLE)
- **J** — Java using JDBC (Java Database Connectivity)
- **S** — SQL

**Table 11-1 Use Case Model Overview: Internal Temporary LOBs**

| Use Case and Page  | P | O | B | C | V* | J |
|--|---|---|---|---|----|---|
| <a href="#">Appending One Temporary LOB to Another</a> on page 11-150                          | + | + | + | + | -  | - |
| <a href="#">Checking If a LOB is Temporary</a> on page 11-23                                   | + | + | + | + | -  | + |
| <a href="#">Comparing All or Part of Two Temporary LOBs</a> on page 11-92                      | + | - | + | + | -  | - |
| <a href="#">Copying a LOB Locator for a Temporary LOB</a> on page 11-125                       | + | + | + | + | -  | - |
| <a href="#">Copying All or Part of One Temporary LOB to Another</a> on page 11-115             | + | + | + | + | -  | - |
| <a href="#">Creating a Temporary LOB</a> on page 11-13   | + | + | + | + | -  | + |
| <a href="#">Determining if a LOB Locator for a Temporary LOB Is Initialized</a> on page 11-140 | - | + | - | + | -  | - |
| <a href="#">Determining If a Pattern Exists in a Temporary LOB (instr)</a> on page 11-99       | + | - | + | + | -  | - |
| <a href="#">Loading a Temporary LOB with Data from a BFILE</a> on page 11-39                   | + | + | + | + | -  | - |
| <a href="#">Loading a Temporary BLOB with Binary Data from a BFILE</a> on page 11-48           | + | - | - | - | -  | - |
| <a href="#">Loading a Temporary CLOB/NCLOB with Character Data from a File</a> on page 11-52   | + | - | - | - | -  | - |
| <a href="#">Disabling LOB Buffering for a Temporary LOB</a> on page 11-211                     | - | + | + | + | -  | - |
| <a href="#">Displaying Temporary LOB Data</a> on page 11-64                                    | + | + | + | + | -  | - |
| <a href="#">Enabling LOB Buffering for a Temporary LOB</a> on page 11-197                      | - | + | + | + | -  | - |
| <a href="#">Erasing Part of a Temporary LOB</a> on page 11-188                                 | + | + | + | + | -  | - |
| <a href="#">Finding Character Set Form of a Temporary LOB</a> on page 11-147                   | - | + | - | - | -  | - |
| <a href="#">Finding Character Set ID of a Temporary LOB</a> on page 11-144                     | - | + | - | - | -  | - |

**Table 11-1 Use Case Model Overview: Internal Temporary LOBs (Cont.)**

| Use Case and Page  | P | O | B | C | V* | J |
|--|---|---|---|---|----|---|
| <a href="#">Finding the Length of a Temporary LOB</a> on page 11-106                         | + | + | + | + | -  | - |
| <a href="#">Flushing Buffer for a Temporary LOB</a> on page 11-204                           | - | + | + | + | -  | - |
| <a href="#">Freeing a Temporary LOB</a> on page 11-31  | + | + | + | + | -  | + |
| <a href="#">Is One Temporary LOB Locator Equal to Another</a> on page 11-135                 | - | + | - | + | -  | - |
| <a href="#">Loading a Temporary LOB with Data from a BFILE</a> on page 11-39                 | + | + | + | + | -  | - |
| <a href="#">Loading a Temporary BLOB with Binary Data from a BFILE</a> on page 11-48         | + | - | - | - | -  | - |
| <a href="#">Loading a Temporary CLOB/NCLOB with Character Data from a File</a> on page 11-52 | + | - | - | - | -  | - |
| <a href="#">Reading Data from a Temporary LOB</a> on page 11-74                              | + | + | + | + | -  | - |
| <a href="#">Reading Portion of Temporary LOB (Substr)</a> on page 11-85                      | + | - | + | + | -  | - |
| <a href="#">Trimming Temporary LOB Data</a> on page 11-179                                   | + | + | + | + | -  | - |
| <a href="#">Write-Appending to a Temporary LOB</a> on page 11-160                            | + | + | + | + | -  | - |
| <a href="#">Writing Data to a Temporary LOB</a> on page 11-168                               | + | + | + | + | -  | - |

\* There are no Visual Basic examples provided for internal temporary LOB use cases.

## Programmatic Environments

Oracle9i supports the definition, creation, deletion, access, and update of temporary LOBs in the following programmatic environments or "interfaces":

- PL/SQL, using the DBMS\_LOB package
- C/C++, using PRO\*C precompiler
- COBOL, using Pro\*COBOL precompiler
- C, using OCI
- Java, using JDBC

## Locators

The 'interfaces' listed earlier, 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. You must manipulate them using the DBMS\_LOB package, OCI, or the other programmatic interfaces.

## Temporary LOB Locators Can be IN Values

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 follows:

- *As a value in a **WHERE** clause* for INSERT, UPDATE, DELETE, or SELECT. For example:

```
SELECT pattern FROM composite_image WHERE temp_lob_pattern_id =  
somepattern_match_function(lobvalue);
```

- *As a variable in a **SELECT INTO...** statement.* For example:

```
SELECT PermanentLob INTO TemporaryLob_loc FROM Demo_tab WHERE Column1 := 1;
```

---

---

**Note:** Selecting a permanent LOB into a LOB locator that points to a temporary LOB will cause the locator to point to a permanent LOB. It does not cause a copy of the permanent LOB to be put in the temporary LOB.

---

---



---

## Can You Use the Same Functions for Temporary and Internal Persistent LOBs?

You can use the following functions for internal persistent LOBs and 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 (OCILobLocatorAssign, OCILobLocatorIsInit, and so on).

In addition, you can use the ISTEMPORARY function to determine if a LOB is temporarily based on its locator.

---

---

**Note:** One thing to keep in mind is that temporary LOBs do not support transactions and consistent reads.

---

---

## DBMS\_LOB.createtemporary() Parameter is a Hint

The DBMS\_LOB.createtemporary() call takes a duration parameter. This parameter is merely a hint and is not strictly enforced.

## Temporary LOB Data is Stored in Temporary Tablespace

Temporary LOBs are not stored permanently in the database like other data. The data is stored in temporary tablespaces and 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 table schema, there is no meaning to the terms "inline" and "out-of-line" for temporary LOBs.

---

---

**Note:** All temporary LOBs reside on the *server*. There is no support for *client-side* temporary LOBs.

---

---

## 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 instance terminates, all temporary LOBs are deleted.

### OCI Can Group Temporary LOBs into Logical Buckets

OCI users can group temporary LOBs together into a logical bucket.

"`OCIDuration`" represents a store for temporary LOBs. There is a default duration for every session into which temporary LOBs are placed if you do not specify a specific duration. The default duration ends when your session ends. Also, you can perform an `OCIDurationEnd` operation that frees all `OCIDuration` contents.

## Memory Handling

### LOB Buffering and `CACHE`, `NOCACHE`, `CACHE READS`

Temporary LOBs are especially useful when you want to perform transformational operations on a LOB — such as morphing an image, or changing a LOB from one format to another — and then return it to the database.

These transformational operations can use LOB Buffering. You can specify `CACHE`, `NOCACHE`, or `CACHE READS` for *each* temporary LOB, and `FREE` an individual temporary LOB when you have no further need for it.

### Temporary Tablespace

Your temporary tablespace is used to store temporary LOB data. Data storage resources are controlled by the DBA through control of a user's access to temporary tablespaces, and by the creation of different temporary tablespaces.

### Explicitly Free Temporary LOB Space to Reuse It

Memory usage increases incrementally as the number of temporary LOBs grows. You can reuse temporary LOB space in your session by freeing temporary LOBs explicitly.

- *When the Session Finishes:* 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.
- *When the Session Dies:* 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.

### Selecting a Permanent LOB INTO a Temporary LOB Locator

We previously noted that if you perform the following:

```
SELECT permanent_lob INTO temporary_lob_locator FROM y_blah WHERE x_blah
```

the `temporary_lob_locator` will get overwritten with the locator for the `permanent_lob`. The `temporary_lob_locator` now points to the LOB stored in the table.

---

---

**Note:** Unless you saved the locator for the `temporary_lob` in another variable, you will lose track of the LOB that `temporary_lob_locator` originally pointed to before the `SELECT INTO` operation.

In this case the temporary LOB *will not get implicitly freed*. If you do not wish to waste space, explicitly free a temporary LOB before overwriting it with a permanent LOB locator.

---

---

Since CR and rollbacks are not supported for temporary LOBs, you will have to free the temporary LOB and start over again if you run into an error.

**See Also:** [Chapter 7, "Modeling and Design"](#), "LOB Storage" on page 7-4 and [Chapter 9, "LOBs: Best Practices"](#).

## Locators and Semantics

Creation of a temporary LOB instance by a user causes the engine to create, and return a locator to LOB data. Temporary LOBs do not support any operations that are not supported for persistent LOB locators, but temporary LOB locators have specific features.

## Features Specific to Temporary LOBs

The following features are specific to temporary LOBs:

- *Temporary LOB Locator is Overwritten by Permanent LOB Locator*

For example, 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 by 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.

- **Assigning Multiple Locators to Same Temporary LOB Impacts Performance**

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. This is because semantically each locator will have its own copy of the temporary LOB. Each time you use `OCILOBLocatorAssign`, or the equivalent assignment in PL/SQL, the database makes a copy of the temporary LOB (although it may be done lazily for performance reasons).

Each locator points 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 `OCILOBLocatorAssign`, the database copies the original temporary LOB and cause the second locator to point to the copy, not the original temporary LOB.

- **Avoid Using More than One Locator For Each 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 for each 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 round trip to the server:

- \* **Assigning one temporary LOB to another**

```
DECLARE
  Va BLOB;
  Vb BLOB;
BEGIN
  DBMS_LOB.CREATETEMPORARY (Vb, TRUE);
```

```

DBMS_LOB.CREATETEMPORARY (Va , TRUE ) ;
Va := Vb;
END;

```

This causes Oracle to create a copy of `Vb` and point the locator `Va` to it. This also frees 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 are 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.

**See Also:**

- *Oracle9i Database Concepts*
- *Oracle9i 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 overhead mentioned earlier, then persistent internal LOBs may be more suitable for such applications. More precisely:

- \* Do not use temporary LOBs inside collections or complex objects when you are doing assignments or copies of those collections or complex objects.
- \* Do not select LOB values into temporary LOB locators.

- *Freeing Temporary LOBs in User-Defined Durations. Reassigning Locators Incurs Overhead*
  - **OCI:** You will incur overhead if you have a temporary LOB in a duration, you call `OCIDurationEnd` on that duration, and then reassign the locator for that temporary LOB to another LOB.

Irrespective of whether there was a previous `OCIDurationEnd` call, Oracle attempts to free the temporary LOB to which the locator pointed. If you try to access the temporary LOB with that locator you will incur an error. Once you issue `OCIDurationEnd`, all temporary LOBs in that duration are freed

regardless of the fact that locators may still exist that used to refer to the now freed LOBs.

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.

- **PL/SQL:** 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`

## 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 not be able to access temporary LOBs in the *new* session from the original session.
  - 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 is 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.

## NOCOPY Restrictions

See *PL/SQL User's Guide and Reference*, Chapter 7: "SUBPROGRAMS" — `NOCOPY COMPILER HINT`, for guidelines, restrictions, and tips on using `NOCOPY`.

## Managing Temporary LOBs

Oracle keeps track of temporary LOBs in each session, and provides a `v$` view called `v$temporary_lobs`. From the session the application can determine that user owns the temporary LOBs. This view can be used by DBAs to monitor and guide any emergency cleanup of temporary space used by temporary LOBs.

## Using JDBC and Temporary LOBs

Oracle9i JDBC drivers contain APIs to create and close temporary LOBs. These APIs should replace the previous workaround of using `dbms_lob.createTemporary()`, `dbms_lob.isTemporary()` and `dbms_lob.freeTemporary()` in `dbms_lob` PL/SQL package.

### Using JDBC and Temporary BLOBs

The `oracle.sql.BLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Blob` interface. Each `oracle.sql.BLOB` instance represents a BLOB object in the database. [Table 11-2](#) lists new Oracle extension APIs in `oracle.sql.BLOB` to access temporary BLOBs.

**Table 11-2 JDBC: Temporary BLOB APIs**

| Methods   | Description   |
|---|---|
| <code>public static BLOB createTemporary(Connection conn, boolean cache, int duration) throws SQLException</code> | Creates a temporary BLOB. See                                   |
| <code>public static boolean isTemporary(BLOB blob) throws SQLException</code>                                     | Checks if the specified BLOB locator refers to a temporary BLOB |
| <code>public boolean isTemporary() throws SQLException</code>   | Checks if the current BLOB locator refers to a temporary BLOB   |
| <code>public static void freeTemporary(BLOB temp_blob) throws SQLException</code>                                 | Frees the specified temporary BLOB                              |
| <code>public void freeTemporary() throws SQLException</code>  | Frees the temporary BLOB  |

### Using JDBC and Temporary CLOBs

The `oracle.sql.CLOB` class is the Oracle JDBC driver's implementation of standard JDBC `java.sql.Clob` interface. [Table 11-3](#) lists the new Oracle extension APIs in `oracle.sql.CLOB` for accessing temporary CLOBs.

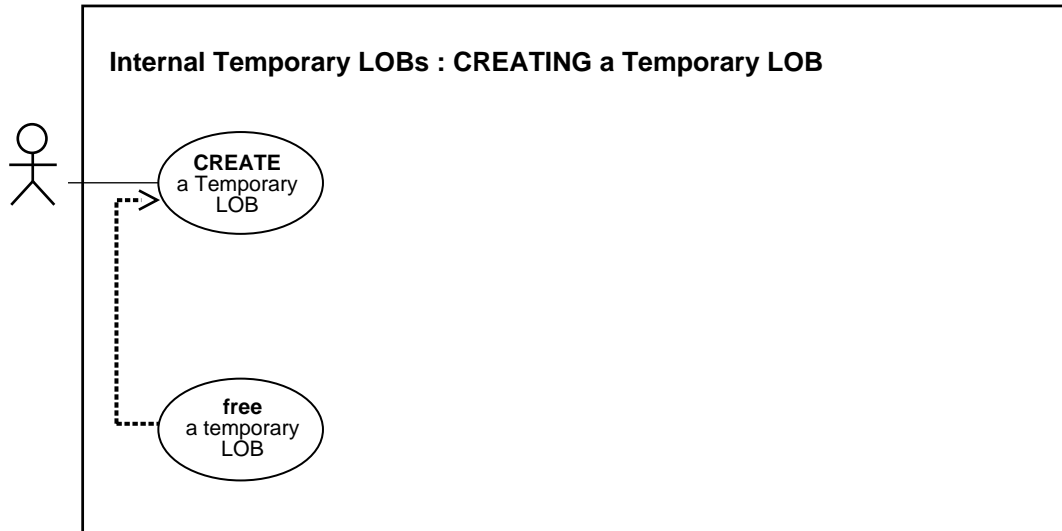
**Table 11–3 JDBC: Temporary CLOB APIs**

| <b>Methods</b>   | <b>Description</b>  |
|--|---|
| <pre>public static CLOB createTemporary(     Connection conn, boolean cache,     int duration)     throws SQLException</pre> | Creates a temporary CLOB  |
| <pre>public static boolean isTemporary(     CLOB clob) throws SQLException</pre>   | Checks if the specified CLOB locator refers to a temporary CLOB |
| <pre>public boolean isTemporary()     throws SQLException</pre>  | Checks if the current CLOB locator refers to a temporary CLOB   |
| <pre>public static void freeTemporary(     CLOB temp_clob)     throws SQLException</pre>                                     | Frees the specified temporary CLOB                              |
| <pre>public void freeTemporary()     throws SQLException</pre>   | Frees the temporary CLOB  |



## Creating a Temporary LOB

Figure 11-1 Use Case Diagram: Creating a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to create a temporary LOB.

### Usage Notes

A temporary LOB is 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.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — CREATETEMPORARY, FREETEMPORARY.
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCreateTemporary
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DESCRIBE, LOB COPY.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE, LOB COPY
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These examples create a temporary LOB and copy the object contents from an ad\_ composite image from the Print\_media table to the temporary LOB. This technique is useful, for example, if you need to convert the image from one graphic format to another. The temporary LOB is read through the CACHE, and is automatically cleaned up at the end of the user's session, if it is not explicitly freed sooner.

### Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Creating a Temporary LOB](#) on page 11-15
- **C (OCI):** [Creating a Temporary LOB](#) on page 11-16
- **COBOL (Pro\*COBOL):** [Creating a Temporary LOB](#) on page 11-18
- **C/C++ (Pro\*C/C++):** [Creating a Temporary LOB](#) on page 11-19
- **Visual Basic (OO4O):** No example is provided with this release.
- **Java (JDBC):** [Creating a Temporary BLOB](#) on page 11-20

## PL/SQL (DBMS\_LOB Package): Creating a Temporary LOB

---



---

**Note:** You must set up the following data structures to use most of the examples given in this chapter:

---



---

Rem Set up script for working with Temporary LOBs

```
DROP TABLE long_raw_tab;
CREATE TABLE long_raw_tab (id number, long_raw_col long raw);
INSERT INTO long_raw_tab VALUES (1, HEXTORAW('7D'));
INSERT INTO Print_media (product_id,ad_composite) SELECT
    id,TO_LOB(long_raw_col) FROM long_raw_tab;
```

---



---

**Note:** The DBMS\_LOB.CREATETEMPORARY procedure takes an optional duration parameter. In PL/SQL, this duration parameter is used only as a hint about the duration of the LOB data. PL/SQL calculates the duration of the LOB data internally, taking into account your hint. You do not have to specify the duration of the LOB data!

---



---

```
DECLARE
    Dest_loc      BLOB;
    Src_loc       BLOB;
    Amount        INTEGER := 4000;
BEGIN
    SELECT ad_composite INTO Src_loc FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;

    /* Create a temporary LOB: */
    DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);

    /* Copy the entire object 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;
```

## C (OCI): Creating a Temporary LOB

```

/* Creating a temporary LOB using C(OCI). [Example script: 3820.c]
   This function reads in one of the composite ads, ad_composite,
   from table Print_media.It creates a temporary LOB so that you can use the
   temporary LOB to convert the image from one format to another, say JPG to
   GIFF.
   The Temporary LOB created is read through the CACHE, and is automatically
   cleaned
   up after the your 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, *defnp2;
    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 product_id =3060: */
    sqlstmt = (text *)
        "SELECT ad_composite FROM Print_media
         WHERE product_id = 3060 AND ad_id = 11001 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)
|| &defnp2, errhp, (ub4) 2, (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 OCI_Snapshot*) 0, (OCI_Snapshot*) 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;

```

}

## COBOL (Pro\*COBOL): Creating a Temporary LOB

```

* Creating a Temporary LOB [example script: 3821.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. CREATE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_COMPOSITE INTO :BLOB1
FROM PRINT_MEDIA
WHERE PRODUCT_ID = 3060 AND AD_ID = 11001
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 SQLEERROR 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.

```

## C/C++ (Pro\*C/C++): Creating a Temporary LOB

```

/* Creating a temporary LOB [example script #: 3822.c] */
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 ad_composite INTO :Lob_loc FROM Print_media
        WHERE product_ID = 3060 AND ad_id = 111001;

    /* 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;
}
```

## Java (JDBC): Creating a Temporary BLOB

To create a temporary BLOB, the JDBC application can use the `createTemporary` static method defined in `oracle.sql.BLOB` class as follows:

```
/**
 * Create a temporary blob.
```



```

*
* @param cache Specifies if LOB should be read into buffer cache or not.
* @param duration The duration of the temporary LOB. The following are
*                 valid values: DURATION_SESSION, DURATION_CALL.
* @return A temporary blob.
* @since 8.2.0
*/
public static BLOB createTemporary (Connection conn, boolean cache, int
duration)          throws SQLException

```

Possible values of the duration parameter are:

```
public static final int DURATION_SESSION
```

```
public static final int DURATION_CALL
```

Each call to createTemporary returns a temporary BLOB. For example,

```

// Make a JDBC connection
Connection conn = ...

// Create a temporary BLOB
BLOB temporaryBlob = BLOB.createTemporary (conn, true,
BLOB.DURATION_SESSION);

```

The new createTemporary API replaces previous workarounds using DBMS\_LOB.createtemporary() in the DBMS\_LOB PL/SQL package.

## Java (JDBC): Creating a Temporary CLOB

To create a temporary CLOB, the JDBC application can use the createTemporary static method defined in oracle.sql.CLOB class as follows:

```

/**
* Create a temporary clob.
*
* @param cache Specifies if LOB should be read into buffer cache or not.
* @param duration The duration of the temporary LOB. The following are
*                 valid values: DURATION_SESSION, DURATION_CALL.
* @return A temporary clob.
*/
public static CLOB createTemporary (Connection conn, boolean cache, int
duration)
throws SQLException

```

The possible values of the duration parameter are --

```
public static final int DURATION_SESSION
public static final int DURATION_CALL
```

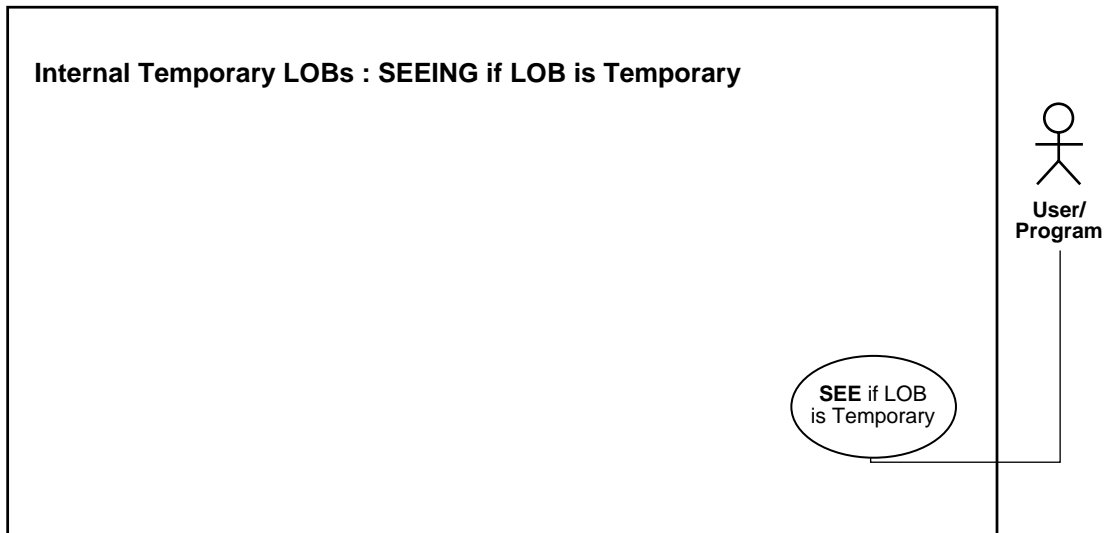
Each call to `createTemporary` returns a temporary CLOB. For example,

```
// Make a JDBC connection
    Connection conn = ...
// Create a temporary CLOB
    CLOB temporaryClob = CLOB.createTemporary (conn, true, CLOB.DURATION_
SESSION);
```

The new `createTemporary` API should replace the previous workaround of using `dbms_lob.createtemporary()` in `dbms_lob` PL/SQL package.

## Checking If a LOB is Temporary

Figure 11–2 Use Case Diagram: Checking If a LOB is Temporary



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to see if a LOB is temporary.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — ISTEMPORARY, FREETEMPORARY

- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobIsTemporary
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DESCRIBE, ISTEMPORARY.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE...ISTEMPORARY
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** *Oracle9i JDBC Developer's Guide and Reference* Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. *Oracle9i SQLJ Developer's Guide and Reference* Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These are generic examples that query whether the locator is associated with a temporary LOB or not.

### Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Checking If a LOB is Temporary](#) on page 11-24
- **C (OCI):** [Checking If a LOB is Temporary](#) on page 11-25
- **COBOL (Pro\*COBOL):** [Checking If a LOB is Temporary](#) on page 11-26
- **C/C++ (Pro\*C/C++):** [Checking If a LOB is Temporary](#) on page 11-27
- **Visual Basic (OO4O):** No example is provided with this release.
- **Java (JDBC):** [Checking if a BLOB is Temporary](#) on page 11-28
- **Java (JDBC):** [Checking if a CLOB is Temporary](#) on page 11-29.

## PL/SQL (DBMS\_LOB Package): Checking If a LOB is Temporary

```
/* Checking if a LOB is temporary. [Example script: 3828.sql]
   This is an example of freeing a temporary LOB. First test to make
   sure that the LOB locator points to a temporary LOB, then free it.
   Otherwise, 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;

```

## C (OCI): Checking If a LOB is Temporary

/\* Checking if a LOB is temporary. [Example script: 3829.c]  
 This function frees a temporary LOB. It takes a locator as an argument,  
 checks to see if it is a temporary LOB. If it is, the function frees  
 the temporary LOB. Otherwise, it prints out a message saying the locator  
 was not a temporary LOB locator. This function returns 0 if it  
 completes successfully, -1 otherwise: \*/

```

sb4 check_and_free_temp(OCILOBLocator *tblob,
                        OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCISmt       *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;
}
```

## COBOL (Pro\*COBOL): Checking If a LOB is Temporary

```
* Checking if a LOB is temporary [Example script: 3830.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-ISTEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 SQLEERROR 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.

```

## C/C++ (Pro\*C/C++): Checking If a LOB is Temporary

```

/* Checking if a LOB is temporary [Example script: 3831.pc]
#include <oci.h>
#include <stdio.h>
#include <sqlca.h>

void Sample_Error()
{
  EXEC SQL WHENEVER SQLEERROR 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;

    /* Note that in this example, isTemporary should be 1 (TRUE) */
    if (isTemporary)
        printf("Locator is a Temporary LOB locator\n");
    /* Free the Temporary LOB: */
    EXEC SQL LOB FREE TEMPORARY :Temp_loc;
    /* Release resources held by the Locator: */
    EXEC SQL FREE :Temp_loc;
else
    printf("Locator is not a Temporary LOB locator \n");
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    lobIsTemp_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Java (JDBC): Checking if a BLOB is Temporary

To see if a BLOB is temporary, the JDBC application can either use the `isTemporary` instance method to determine whether the current BLOB object is temporary, or pass the BLOB object to the static `isTemporary` method to determine whether the specified BLOB object is temporary. These two methods are defined as follows:

```
/**
 * Checking if a BLOB is temporary. [Example script: 3833.java]
 * Returns true if LOB locator points to a temporary BLOB, False if not.
```



```

* @param lob the BLOB to test.
* @returns true if LOB locator points to a temporary BLOB, False if not.
*/
    public static boolean isTemporary (BLOB lob) throws SQLException

/**
* Returns true if LOB locator points to a temporary BLOB, False if not.
* @returns true if LOB locator points to a temporary BLOB, False if not.
*/
    public boolean isTemporary () throws SQLException

//The usage example is--

BLOB blob = ...

// See if the BLOB is temporary
boolean isTemporary = blob.isTemporary ();

// See if the specified BLOB is temporary
boolean isTemporary2 = BLOB.isTemporary(blob);

```

This JDBC API replaces previous workarounds that use `DBMS_LOB.isTemporary()`.

## Java (JDBC): Checking if a CLOB is Temporary

To see if a CLOB is temporary, the JDBC application can either use the `isTemporary` instance method to determine whether the current CLOB object is temporary, or pass the CLOB object to the static `isTemporary` method to determine whether the specified CLOB object is temporary. These two methods are defined as follows:

```

/**
* Checking if LOB is temporary [Example script: 3834.java]
* Return true if the LOB locator points to a temporary CLOB, False if it
* does not.
*
* @param lob the BLOB to test.
* @return true if the LOB locator points to a temporary CLOB, False if it
*         does not.
*/
    public static boolean isTemporary (CLOB lob) throws SQLException

/**

```

```
* Return true if the LOB locator points to a temporary CLOB, False if it
* does not.
*
* @return true if the LOB locator points to a temporary CLOB, False if it
*         does not.
*/
public boolean isTemporary () throws SQLException

//The usage example is--

CLOB clob = ...

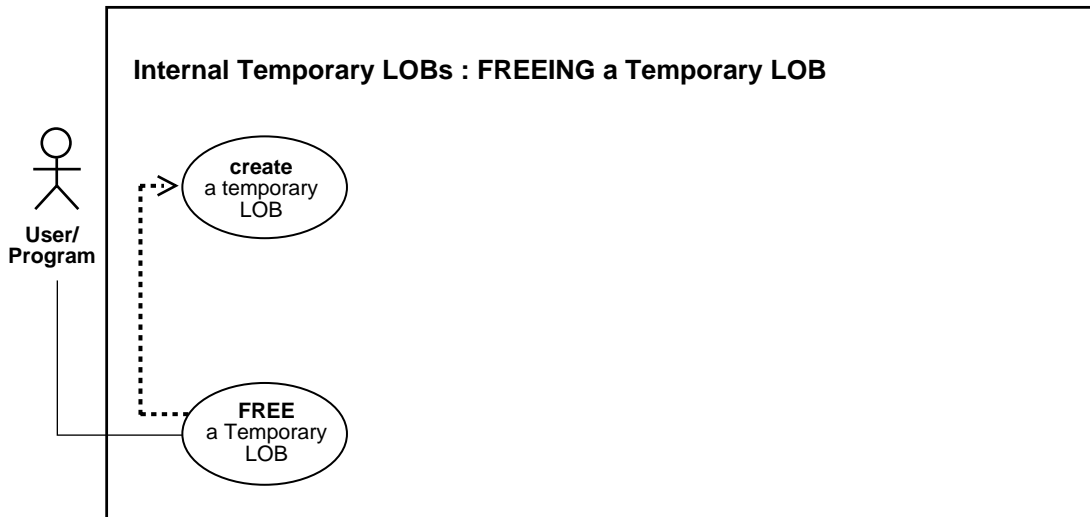
// See if the CLOB is temporary
boolean isTemporary = clob.isTemporary ();

// See if the specified CLOB is temporary
boolean isTemporary2 = CLOB.isTemporary(clob);
```

This API replaces previous workarounds that used `DBMS_LOB.istemporary()`.

## Freeing a Temporary LOB

Figure 11-3 Use Case Diagram: Freeing a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to free a temporary LOB.

### Usage Notes

A temporary LOB instance can only be destroyed for example, in 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.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment.

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — FREETEMPORARY
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobFreeTemporary
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB FREE TEMPORARY.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB FREE TEMPORARY
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 3, "LOB Support in Different Programmatic Environments"](#), [Table 3-50, "JDBC: Temporary BLOB APIs"](#). Note: `temporaryClob.java` has been deprecated as there is no longer a need for this class in the JDBC standard.

### Scenario

Not applicable.

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Freeing a Temporary LOB](#) on page 11-32
- C (OCI): [Freeing a Temporary LOB](#) on page 11-33
- COBOL (Pro\*COBOL): [Freeing a Temporary LOB](#) on page 11-34
- C/C++ (Pro\*C/C++): [Freeing a Temporary LOB](#) on page 11-35
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): [Freeing a Temporary BLOB](#) and [Java \(JDBC\): Freeing a Temporary CLOB](#).
- Java (JDBC): [Creating and Freeing a Temporary CLOB Using TemporaryClob.java](#) has been deprecated.

## PL/SQL (DBMS\_LOB Package): Freeing a Temporary LOB

```
/* Freeing a temporary LOB [Example script: 3836.sql]
DECLARE
    Dest_loc          BLOB;
```

```

Src_loc      BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_3060_11001');
Amount      INTEGER := 4000;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
  /* 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;

```

## C (OCI): Freeing a Temporary LOB

```

/* Freeing a temporary LOB. [Example script: 3837.c]
   This function creates a temporary LOB and then frees it:
   This function returns 0 if it completes successfully, -1 otherwise: */

sb4 freeTempLob(OCIError      *errhp,
                OCISvcCtx    *svchp,
                OCISstmt     *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():freeTempLob\n");
    return -1;
  }

  if(OCILobFreeTemporary(svchp, errhp, tblob))
  {
    printf ("FAILED: OCILobFreeTemporary call in freeTempLob\n");
  }
}

```

```
        return -1;

    }else
    {
        printf("Temporary LOB freed in freeTempLob\n");
    }
    return 0;
}
```

## COBOL (Pro\*COBOL): Freeing a Temporary LOB

```
* Freeing a temporary LOB [Example script: 3838.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".

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.

```

## C/C++ (Pro\*C/C++): Freeing a Temporary LOB

```

/* Freeing a temporary LOB. [Example script: 3839.pc]
#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;
}
```

## Java (JDBC): Freeing a Temporary BLOB

To free a temporary BLOB, the JDBC application can either use the `freeTemporary` instance method to free the current BLOB object, or pass the temporary BLOB to be freed to the static `freeTemporary` method to free the specified temporary BLOB. These two methods are defined as follows:

```
/**
 * Freeing a temporary BLOB. [Example script: 3840.java]
 * This example frees the contents and locator of a temporary BLOB.
 * @param temp_lob A temporary BLOB to be freed.
 * @exception SQLException if temp_lob is a permanent LOB or temp_lob has already
 * been freed.
 */

public static void freeTemporary (BLOB temp_lob) throws SQLException

/**
 * Free the contents and the locator of the temporary BLOB.
 * @exception SQLException if self is a permanent LOB or self has already been
 * freed.
 */

public void freeTemporary() throws SQLException

/**
 * The usage example example is --
 * BLOB tempBlob1 = ...
 * BLOB tempBlob2 = ...
 * // free the temporary BLOB
 * tempBlob1.freeTemporary ();
 * // free the specified temporary BLOB
 * BLOB.freeTemporary(tempBlob2);
 *
 * The newer freeTemporary APIs should replace previous workaround of
 * using dbms_lob.freeTemporary() in dbms_lob PL/SQL package.
 */
```



The `freeTemporary` APIs should replace the previous workaround of using `dbms_lob.freeTemporary()` in `dbms_lob` PL/SQL package.

## Java (JDBC): Freeing a Temporary CLOB

To free a temporary CLOB, the JDBC application can either use the `freeTemporary` instance method to free the current CLOB object, or pass the temporary CLOB to be freed to the static `freeTemporary` method to free the specified temporary CLOB. These two methods are defined as follows:

```
/**
 * Freeing a temporary CLOB. [Example script: 3841.java]
 * Free the contents and the locator of the temporary blob.
 * @param temp_lob A temporary CLOB to be freed.
 * @since 8.2.0
 * @exception SQLException if temp_lob is a permanent LOB or temp_lob has
 *         already been freed.
 */
public static void freeTemporary (CLOB temp_lob) throws SQLException

/**
 * Free the contents and the locator of the temporary CLOB.
 *
 * @since 8.2.0
 * @exception SQLException if self is a permanent lob or self has
 *         already been freed.
 */
public void freeTemporary() throws SQLException

/**
 * Use the free temporary CLOB API as follows:
 * CLOB tempClob1 = ...
 * CLOB tempClob2 = ...
 * // free the temporary CLOB
 * tempClob1.freeTemporary ();
 * // free the specified temporary CLOB
 * CLOB.freeTemporary(tempClob2);
 *
 *The freeTemporary API replaces previous workarounds that use DBMS_
LOB.freeTemporary().
 */
```

The `freeTemporary` API replaces previous workarounds that use `DBMS_LOB.freeTemporary()`.

## Java (JDBC): Creating and Freeing a Temporary CLOB Using `TemporaryClob.java`

---

---

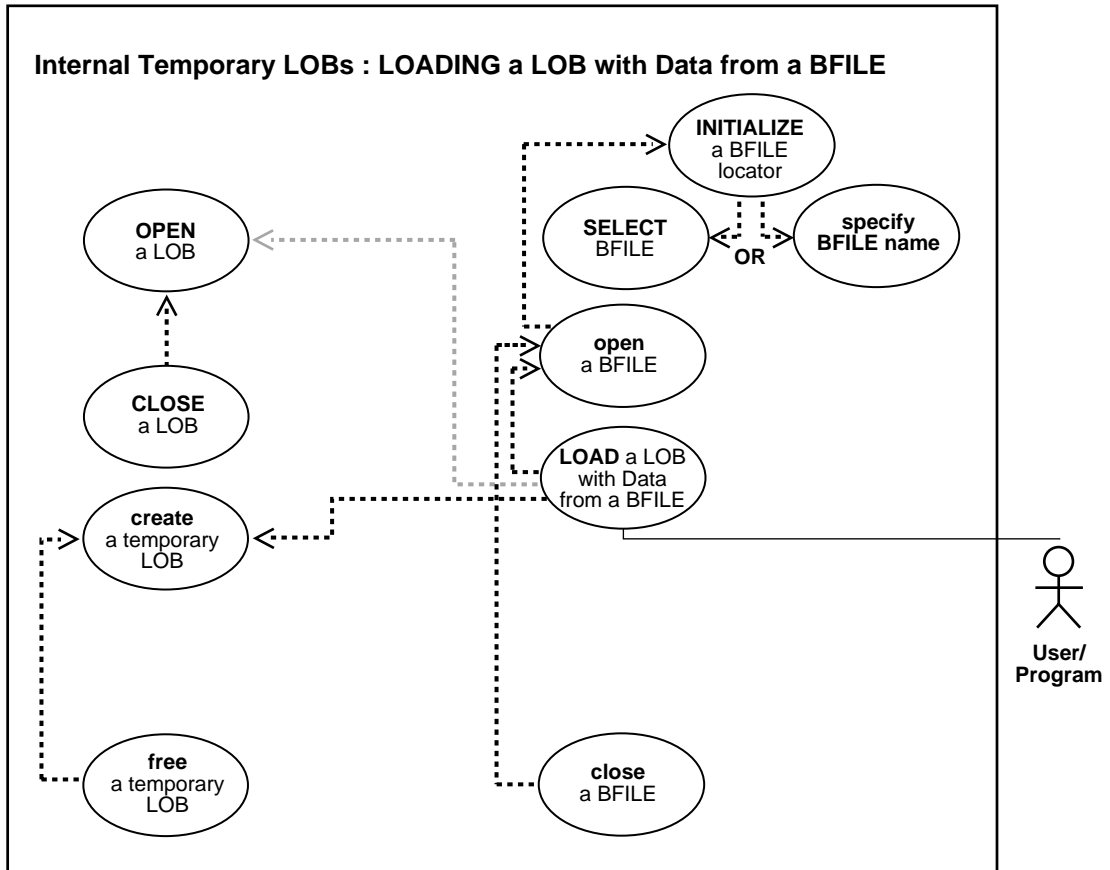
**Note:** `temporaryClob.java` and `temporaryBlob.java` classes have been deprecated. These classes are no longer needed or used in the JDBC standard.

---

---

# Loading a Temporary LOB with Data from a BFILE

Figure 11-4 Use Case Diagram: Loading a LOB with Data from a BFILE



**See Also:**

- [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs" on page 11-2](#)
- ["Loading a Temporary BLOB with Binary Data from a BFILE" on page 11-48](#)
- ["Loading a Temporary CLOB/NCLOB with Character Data from a File" on page 11-52](#)

**Purpose**

This procedure describes how to load a temporary LOB with data from a BFILE.

**Usage Notes**

---

---

**Note:** The `LOADBLOBFROMFILE` and `LOADCLOBFROMFILE` procedures implement the functionality of this procedure and provide improved features for loading binary data and character data. The improved procedures are available in the PL/SQL environment only. When possible, using one of the improved procedures is recommended. See ["Loading a Temporary BLOB with Binary Data from a BFILE" on page 11-48](#) and ["Loading a Temporary CLOB/NCLOB with Character Data from a File" on page 11-52](#) for more information.

---

---

**Binary Data to Character Set Conversion is Needed on BFILE Data** In using 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. When you use the `DBMS_LOB.LOADFROMFILE` procedure to populate a CLOB or NCLOB, you are populating the LOB with binary data from the BFILE. No implicit translation is performed from binary data to a character set. For this reason, you should use the `LOADCLOBFROMFILE` procedure when loading text (see [Loading a Temporary CLOB/NCLOB with Character Data from a File](#) on page 11-52).

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — CREATETEMPORARY, FREETEMPORARY
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobLoadFromFile
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB LOAD.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB LOAD
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Loading a LOB with Data from a BFILE"](#) on page 10-40.

### Scenario

The example procedures assume that there is an operating system source directory (ADPHOTO\_DIR) that contains the LOB data to be loaded into the target LOB.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Loading a Temporary LOB with Data from a BFILE](#) on page 11-41
- [C \(OCI\): Loading a Temporary LOB with Data from a BFILE](#) on page 11-42
- [COBOL \(Pro\\*COBOL\): Loading a Temporary LOB with Data from a BFILE](#) on page 11-44
- [C/C++ \(Pro\\*C/C++\): Loading a Temporary LOB with Data from a BFILE](#) on page 11-46
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Loading a Temporary LOB with Data from a BFILE

```

/* Loading a temporary LOB with data from a BFILE.
   Example procedure freeTempLob_proc is not part of DBMS_LOB package: */

CREATE or REPLACE PROCEDURE freeTempLob_proc(Lob_loc IN OUT BLOB) IS

```

```

BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
    /* 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;
```

## C (OCI): Loading a Temporary LOB with Data from a BFILE

```

/* Loading a temporary LOB with data from a BFILE.
   This section of code shows you how to create a temporary LOB, and load
   the contents of a BFILE into that 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 *)"ADPHOTO_DIR",
                    (ub2)strlen("ADPHOTO_DIR"), (text *)"monitor_3060_11001",
                    (ub2)strlen("monitor_3060_11001")))
{
    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 lob: */
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;
}
}
```

## COBOL (Pro\*COBOL): Loading a Temporary LOB with Data from a BFILE

```
* Loading a temporary LOB with data from a BFILE.
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Loading a Temporary LOB with Data from a BFILE

```
/* Loading a temporary LOB with data from a BFILE. */
#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

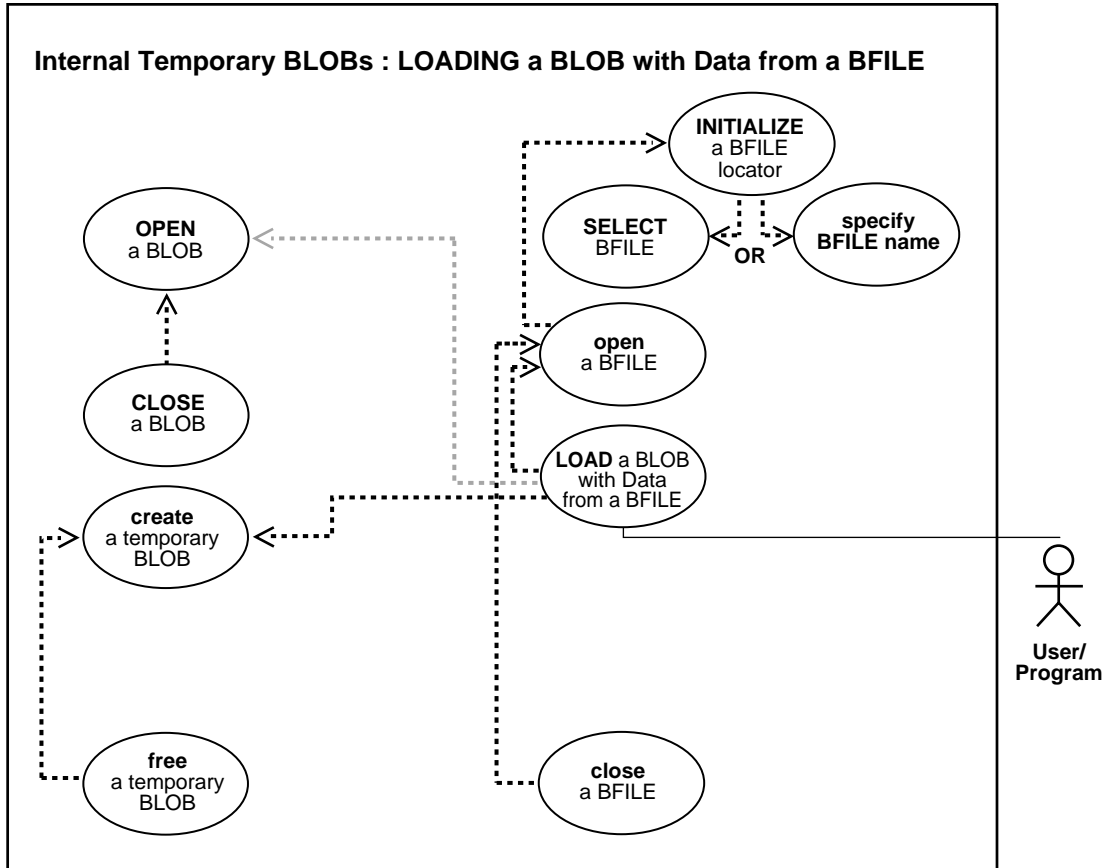
    /* Opening the BFILE is mandatory; */
    /* Opening the LOB is optional: */
    EXEC SQL LOB OPEN :Lob_loc READ ONLY;
    EXEC SQL LOB OPEN :Temp_loc READ WRITE;
    /* Load the data from the BFILE into the Temporary LOB: */
    EXEC SQL LOB LOAD :Amount FROM FILE :Lob_loc INTO :Temp_loc;
    /* Closing 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;
}
```

## Loading a Temporary BLOB with Binary Data from a BFILE

Figure 11-5 Use Case Diagram: Loading a BLOB with Data from a BFILE



**See Also:**

- [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2
- ["Loading a Temporary CLOB/NCLOB with Character Data from a File"](#) on page 11-52
- ["Loading a Temporary LOB with Data from a BFILE"](#) on page 11-39

**Purpose**

This procedure describes how to load a temporary BLOB with binary data from a BFILE.

**Usage Notes**

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for text loading. This achieves the same outcome as `LOADFROMFILE` as well as returning the new offsets to the user. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADBLOBFROMFILE`

**Scenario**

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the binary LOB data to be loaded into the target BLOB.

**Examples**

The example, ["PL/SQL \(DBMS\\_LOB Package\): Loading a Temporary BLOB with BFILE data"](#), illustrates `LOADBLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

## PL/SQL (DBMS\_LOB Package): Loading a Temporary BLOB with BFILE data

The following example illustrates:

- How to load a temporary BLOB with binary data from a BFILE.
- How to load a BLOB with BFILE data without getting the length of the BFILE first.
- How to use the source offset and destination offset values returned by `LOADBLOBFROMFILE` to determine the size of the BLOB.

```

DECLARE
    src_loc      BFILE := bfilename('ADPHOTO_DIR','monitor_3060_11001');
    dst_loc      BLOB;
    src_offset   NUMBER := 1;
    dst_offset   NUMBER := 1;
    src_osin     NUMBER;
    dst_osin     NUMBER;
    bytes_rd     NUMBER;
    bytes_wt     NUMBER;
BEGIN
    dbms_lob.createtemporary(dst_loc, TRUE);

    /* Opening the source BFILE is mandatory. */
    dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

    /* Opening the LOB is optional. */
    dbms_lob.open(dst_loc, dbms_lob.lob_readwrite);
    /* Save the input source/destination offsets. */
    src_osin := src_offset;
    dst_osin := dst_offset;

    /* Use LOBMAXSIZE to indicate loading the entire BFILE. */
    dbms_lob.LOADBLOBFROMFILE(dst_loc, src_loc, dbms_lob.lobmaxsize, src_offset,
        dst_offset) ;

    /* Closing the LOB is mandatory if you have opened it */
    dbms_lob.close(dst_loc);
    dbms_lob.filecloseall();

    /* Use the src_offset returned to calculate the actual
    * amount read from the BFILE.
    */
    bytes_rd := src_offset - src_osin;
    dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd );

```

```
/* Use the dst_offset returned to calculate the actual
 * amount written to the BLOB.
 */
bytes_wt := dst_offset - dst_osin;
dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt );

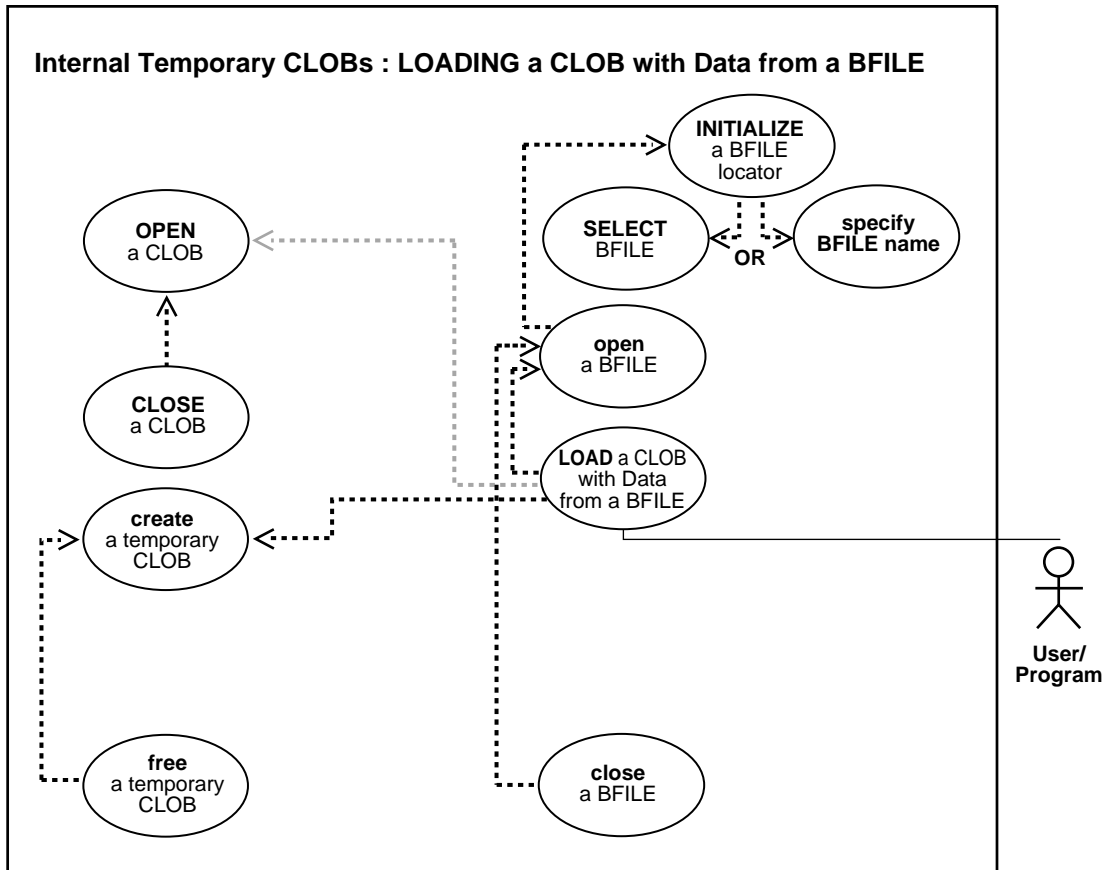
/* If there is no exception the number of bytes read
 * should equal to the number of bytes written.
 */

/* Free the temporary LOB. */
dbms_lob.freetemporary(dst_loc);

END ;
```

## Loading a Temporary CLOB/NCLOB with Character Data from a File

Figure 11-6 Use Case Diagram: Loading a CLOB or NCLOB with Data from a BFILE





**See Also:**

- [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2
- ["Loading a Temporary BLOB with Binary Data from a BFILE"](#) on page 11-48
- ["Loading a Temporary LOB with Data from a BFILE"](#) on page 11-39

**Purpose**

This procedure describes how to load a temporary CLOB or NCLOB with character data from a BFILE.

**Usage Notes**

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for loading of text as the latter method lets you specify the character set id of the BFILE. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADCLOBFROMFILE`

**Scenario**

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the character LOB data to be loaded into the target CLOB or NCLOB.

**Examples**

The example, ["PL/SQL \(DBMS\\_LOB Package\): Loading a Temporary CLOB/NCLOB with BFILE Data"](#), illustrates `LOADCLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

## PL/SQL (DBMS\_LOB Package): Loading a Temporary CLOB/NCLOB with BFILE Data

### Example

The following example illustrates:

- How to load a temporary CLOB or NCLOB with character data from BFILE.
- How load the entire file without calling `getlength` on the BFILE.
- How to find out the actual amount loaded using return offsets.

This example assumes that `ad_source` is a BFILE in UTF8 character set format and the database character set is UTF8.

```
DECLARE
  src_loc      bfile := bfilename('ADVERT_DIR','ad_source_1000');
  dst_loc      clob ;
  amt          number := dbms_lob.lobmaxsize;
  src_offset   number := 1 ;
  dst_offset   number := 1 ;
  lang_ctx     number := dbms_lob.default_lang_ctx;
  warning      number;
BEGIN
  dbms_lob.createtemporary(dst_loc, TRUE);
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

  /* The default_csid can be used when the BFILE encoding is in the same charset
   * as the destination CLOB/NCLOB charset
   */

  dbms_lob.LOADCLOBFROMFILE(dst_loc, src_loc, amt, dst_offset, src_offset,
    dbms_lob.default_csid, lang_ctx,warning) ;
  commit ;

  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
    (src_offset-1));
  dbms_output.put_line(' Number of characters written to destination: ' ||
    (dst_offset-1) );
  if (warning = dbms_lob.warn_inconvertible_char)
  then
    dbms_output.put_line('Warning: Inconvertible character');
  end if;

  dbms_lob.filecloseall() ;
  dbms_lob.freetemporary(dst_loc);
```

```
END ;
```

## Example

The following example illustrates:

- How to get the character set ID from the character set name using the NLS\_CHARSET\_ID function.
- How to load a stream of data from a single BFILE into different LOBs using the returned offset value and the language context lang\_ctx.
- How to read a warning message.

This example assumes that *ad\_file\_ext\_01* is a BFILE in JA16TSTSET format and the database national character set is AL16UTF16.

```
DECLARE
  src_loc      bfile := bfilename('ADVERT_DIR','ad_file_ext_01') ;
  dst_loc1     nclob;
  dst_loc2     nclob;
  amt          number := 1000;
  src_offset   number := 1;
  dst_offset   number := 1;
  src_osin     number;
  cs_id        number := NLS_CHARSET_ID('JA16TSTSET'); /* 998 */
  lang_ctx     number := dbms_lob.default_lang_ctx;
  warning      number;
BEGIN
  dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);
  dbms_output.put_line(' BFILE csid is ' || cs_id) ;

  /* Load the first 1KB of the BFILE into dst_loc1 */
  dbms_output.put_line(' -----' ) ;
  dbms_output.put_line('   First load   ' ) ;
  dbms_output.put_line(' -----' ) ;

  dbms_lob.createtemporary(dst_loc1, TRUE);

  dbms_lob.LOADCLOBFROMFILE(dst_loc1, src_loc, amt, dst_offset, src_offset,
    cs_id, lang_ctx,warning);
  commit;

  /* The number bytes read may or may not be 1k */
  dbms_output.put_line(' Amount specified ' || amt ) ;
  dbms_output.put_line(' Number of bytes read from source: ' ||
```

```

        (src_offset-1) );
dbms_output.put_line(' Number of characters written to destination: ' ||
        (dst_offset-1) );
IF (warning = dbms_lob.warn_inconvertible_char)
then
    dbms_output.put_line('Warning: Inconvertible character');
END IF;

/* load the next 1KB of the BFILE into the dst_loc2 */
dbms_output.put_line(' -----' );
dbms_output.put_line('   Second load   ' );
dbms_output.put_line(' -----' );

dbms_lob.createtemporary(dst_loc2, TRUE);
/* Notice we are using the src_offset and lang_ctx returned from the previous
 * load. We do not use value 1001 as the src_offset here because sometimes the
 * actual amount read may not be the same as the amount specified.
 */
src_osin := src_offset;
dst_offset := 1;
dbms_lob.LOADCLOBFROMFILE(dst_loc2, src_loc, amt, dst_offset, src_offset,
        cs_id, lang_ctx, warning);
commit;
dbms_output.put_line(' Number of bytes read from source: ' ||
        (src_offset-src_osin) );
dbms_output.put_line(' Number of characters written to destination: ' ||
        (dst_offset-1) );
if (warning = dbms_lob.warn_inconvertible_char)
then
    dbms_output.put_line('Warning: Inconvertible character');
END IF;

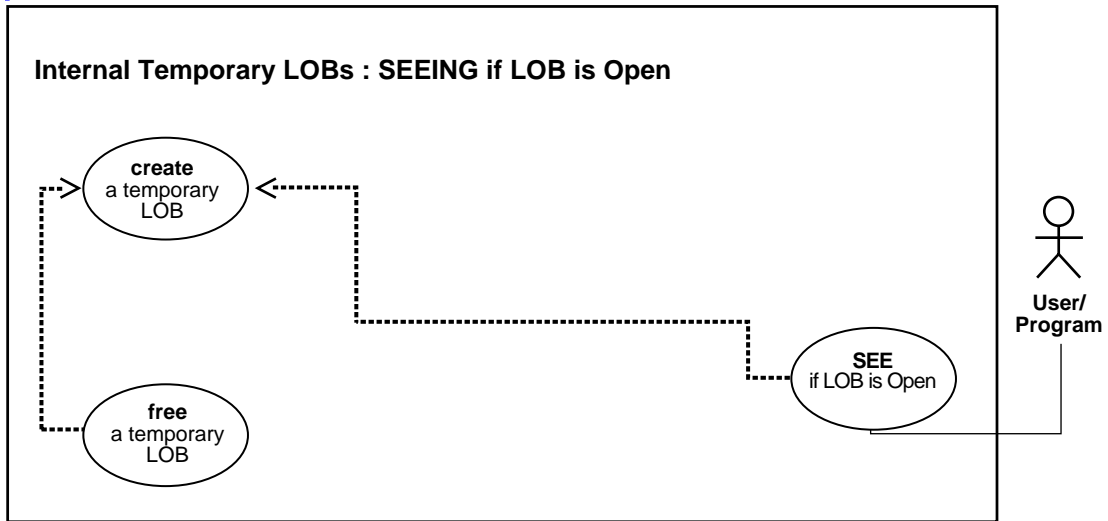
dbms_lob.filecloseall() ;
dbms_lob.freetemporary(dst_loc1);
dbms_lob.freetemporary(src_loc2);

END ;

```

## Determining If a Temporary LOB Is Open

Figure 11-7 Use Case Diagram: Determining If a Temporary LOB Is Open



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to see if a temporary LOB is open.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — ISOPEN

- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobIsOpen
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DESCRIBE...ISOPEN.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE...ISOPEN
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Checking if a LOB is Open"](#) on page 10-59.

### Scenario

These generic examples takes a locator as input, create a temporary LOB, open it and test if the LOB is open.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL: Determining if a Temporary LOB is Open](#) on page 11-58
- [C \(OCI\) Determining if a Temporary LOB is Open](#) on page 11-59
- [COBOL \(Pro\\*COBOL\): Determining if a Temporary LOB is Open](#) on page 11-60
- [C/C++ \(Pro\\*C/C++\): Determining if a Temporary LOB is Open](#) on page 11-62
- **Visual Basic (OO4O):** No example is provided with this release.
- **Java (JDBC):** No example is provided with this release.

## PL/SQL: Determining if a Temporary LOB is Open

```
/* Determining if a temporary LOB is open.
   Example procedure seeTempLOBIsOpen_proc is not part of
   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);
```

```

/* 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;

```

## C (OCI) Determining if a Temporary LOB is Open

```

/* Determining if a temporary LOB is open.
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;

    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, lob_loc))
{
    printf("OCIlobFreeTemporary FAILED \n");
    return -1;
}
return 0;
}
```

## COBOL (Pro\*COBOL): Determining if a Temporary LOB is Open

\* Determining if a temporary LOB is open. [Example script: 3850.pco]

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

01  USERID    PIC X(11) 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  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.
EXEC SQL
    ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Determining if a Temporary LOB is Open

```
/* Determining if a temporary LOB is open. [Example script: 3851.pc] */
#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;

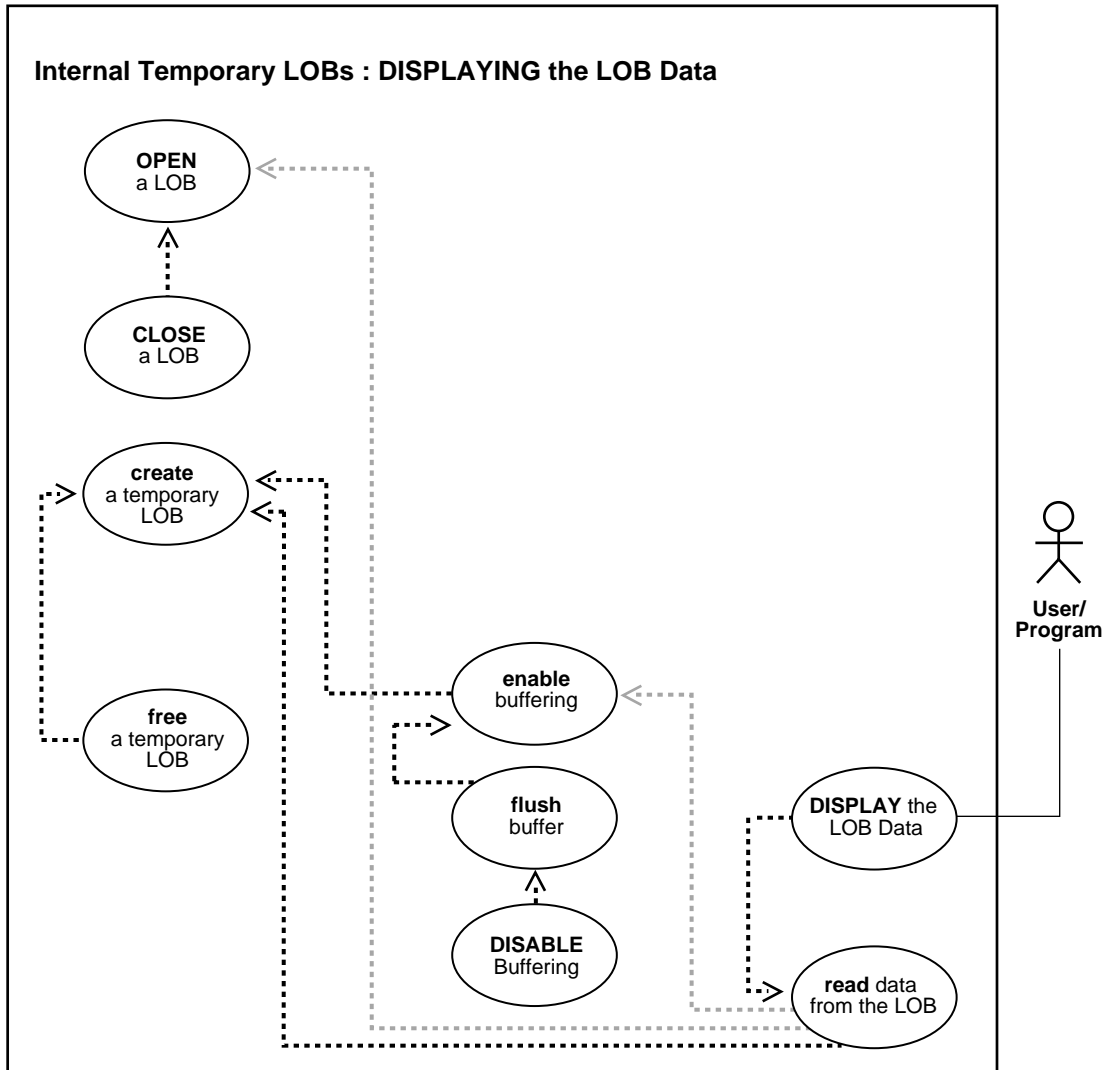
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* Allocate and Create the Temporary LOB */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* Open the Temporary LOB */
    EXEC SQL LOB OPEN :Temp_loc READ ONLY;
    /* Determine if the LOB is Open */
    EXEC SQL LOB 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;  
}
```

## Displaying Temporary LOB Data

Figure 11-8 Use Case Diagram: Displaying Temporary LOB Data



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

## Purpose

This procedure describes how to display temporary LOB data.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — OPEN, LOADFROMFILE, READ. Chapter 29, "DBMS\_OUTPUT" — PUT\_LINE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobRead
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB READ
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Displaying LOB Data"](#) on page 10-107.

## Scenario

As an instance of displaying a LOB, our example stream-reads the image `monitor_photo` from the column object `Adheader_typ` onto the client-side in order to view the data.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Displaying Temporary LOB Data](#) on page 11-66
- [C \(OCI\): Displaying Temporary LOB Data](#) on page 11-67
- [COBOL \(Pro\\*COBOL\): Displaying Temporary LOB Data](#) on page 11-70
- [C/C++ \(Pro\\*C/C++\): Displaying Temporary LOB Data](#) on page 11-72
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Displaying Temporary LOB Data

```
/* Displaying temporary LOB data. [Example script: 3853.sql]
   The following function accesses the monitor_photo 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('ADPHOTO_DIR', 'monitor_photo_3060_11001');
    Amount        INTEGER := 128;
    Bbuf          RAW(128);
    Position      INTEGER :=1;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
    /* 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;

```

## C (OCI): Displaying Temporary LOB Data

```

/* Displaying temporary LOB data. [Example script: 3854.c]
This function accesses the monitor_photo file for product 3060 with ad_id
11001. It creates a temporary LOB, loads some data from the file, 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 "ADPHOTO_DIR". It also assumes that the file is at least
14000 bytes long, which is the amount 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 MAXBUFLLEN 32767
sb4 display_file_to_lob( OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCISstmt      *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[MAXBUFLLEN];
    sb4 return_code = 0;

    (void) printf("\n====> Testing loading files into lobes 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*)"ADPHOTO_DIR",
        (ub2)
        strlen("ADPHOTO_DIR"), (text*)"monitor_photo_3060_11001",
        (ub2)strlen("monitor_photo_3060_11001")))
    {
        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', MAXBUFLLEN);

    retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                      (dvoid *) bufp, (amount < MAXBUFLLEN ? amount : MAXBUFLLEN),
                      (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_ERROR:
        /* 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', MAXBUFLLEN);
            amtp = 0;
            remainder -= MAXBUFLLEN;
            printf("remainder is %d \n",remainder);
            retval = OCILobRead(svchp, errhp, tblob, &amtp, offset,
                              (dvoid *) bufp, (ub4) MAXBUFLLEN, (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;
}

```

## COBOL (Pro\*COBOL): Displaying Temporary LOB Data

```

* Displaying temporary LOB data. [Example script: 3855.pco]
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Displaying Temporary LOB Data

```
/* Displaying temporary LOB data. */
#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 displayTempLOB_proc()
{
    OCIBlobLocator *Temp_loc;
    OCIBFileLocator *Lob_loc;
    char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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);

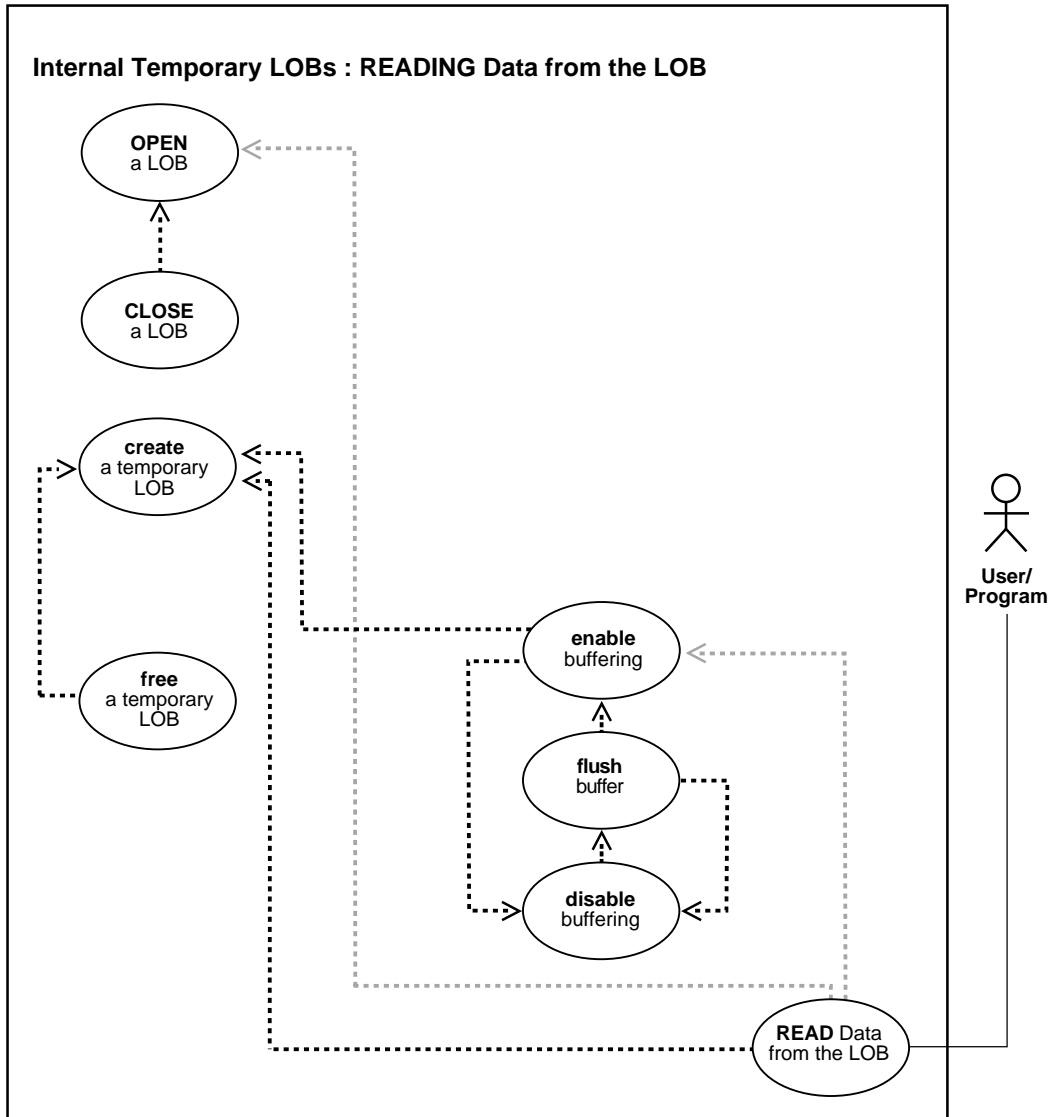
EXEC SQL WHENEVER SQLERROR DO Sample_Error();
/* Allocate and Initialize the LOB Locators */
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
/* Opening the LOBs is Optional */
EXEC SQL 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;
}

```

## Reading Data from a Temporary LOB

Figure 11-9 Use Case Diagram: Reading Data from a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

## Purpose

This procedure describes how to read data from a temporary LOB.

## Usage Notes

**Stream Reading** The most efficient way to read large amounts of LOB data is to use `OCILOBRead()` with the streaming mechanism enabled using 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, buflen, 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, input to the callback, indicates how many bytes are filled in the buffer. 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*).

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — READ

- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobRead
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB READ
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Reading Data from a LOB](#)" on page 10-118.

### Scenario

These examples read LOB data from the AD\_PHOTO LOB column.

### Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Reading Data from a Temporary LOB](#) on page 11-76
- **C (OCI):** [Reading Data from a Temporary LOB](#) on page 11-77
- **COBOL (Pro\*COBOL):** [Reading Data from a Temporary LOB](#) on page 11-80
- **C/C++ (Pro\*C/C++):** [Reading Data from a Temporary LOB](#) on page 11-82
- **Visual Basic (OO4O):** No example is provided with this release.
- **Java (JDBC):** No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Reading Data from a Temporary LOB

```
/* Reading temporary LOB data. [Example script: 3859.sql]
   PL/SQL does not support streaming reads. See the OCI example for an
   illustration of streaming reads: */
DECLARE
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3106_13001');
  Amount        INTEGER := 4000;
  Bbuf          RAW(32767);
  Position      INTEGER :=1;
BEGIN
```



```

DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
/* 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);
END;

```

## C (OCI): Reading Data from a Temporary LOB

```

/* Reading temporary LOB data. [Example script: 3861.c]
This is the same example as for reading and displaying data from a
temporary LOB. This function takes the 'monitor_photo for product 3060,
ad_id 11001' 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, -1 if it fails. */

#define MAXBUFLLEN 32767

sb4 test_file_to_lob (OCILOBLocator *lob_loc,
                    OCIError *errhp,
                    OCISvcCtx *svchp,
                    OCIStmt *stmthp,
                    OCIEnv *envhp)
{
    int rowind;
    OCILOBLocator *bfile;

    ub4 amount = 14000;
    ub4 offset = 0;
    ub4 loblen = 0;
    ub4 amtp = 0;
    sword retval;
    ub4 piece = 1;
    ub4 remainder=0;
    ub1 bufp[MAXBUFLLEN];

    (void) printf(
        "\n====> Testing loading files into lobes and displaying them\n\n");

```

```

    if (OCIDescriptorAlloc((dvoid **)&bfile,
                          (ub4)OCI_DTYPE_LOB, (size_t)0,
                          (dvoid**)0))

/* Create a temporary LOB: */
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(OCILobFileSetName(envhp, errhp, &bfile, (text*)"ADPHOTO_DIR",
                    (ub2)strlen("ADPHOTO_DIR"),
                    (text*)"monitor_photo_3060_11001",
                    (ub2)strlen("monitor_photo_3060_11001")))
{
    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, lob_loc, (OCILobLocator*)bfile, (ub4)amount,
                    (ub4)1, (ub4)1))
{
    printf("OCILobLoadFromFile FAILED\n");
    return -1;
}

offset = 1;
memset(bufp, '\0', MAXBUFLLEN);

retval = OCILobRead(svchp, errhp, lob_loc, &amp;tp, offset, (dvoid *) bufp,
                  (amount < MAXBUFLLEN ? amount : MAXBUFLLEN), (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_ERROR:
        /* 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, lob_loc, &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;
}
if (OCIDescriptorFree ((dvoid*) lob_loc, (ub4) OCI_DTYPE_LOB))
{
    printf ("failed in OCIDescriptor Free\n");
    return -1;
}

```

```

/* Clean up the temp LOB now that we are done with it: */
if(check_and_free_temp(lob_loc, errhp, svchp, stnthp, envhp))
{
    printf("check and free failed in load test\n");
    return -1;
}
return 0;
}

sb4 check_and_free_temp(OCILOBLocator *tblob,
                        OCIError      *errhp,
                        OCISvcCtx     *svchp,
                        OCISstmt     *stnthp,
                        OCIEnv        *envhp)
{
    boolean is_temp;
    is_temp = FALSE;
    if (OCILOBIsTemporary (envhp, errhp, tblob, &is_temp))
    {
        printf ("FAILED: OciLobIsTemporary call \n");
    }
    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;
}

```

## COBOL (Pro\*COBOL): Reading Data from a Temporary LOB

```

* Reading temporary LOB data. [Example script: 3862.pco]
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 SQL 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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Reading Data from a Temporary LOB

```
/* Reading temporary LOB data. [Example script: 3863.pc] */
#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

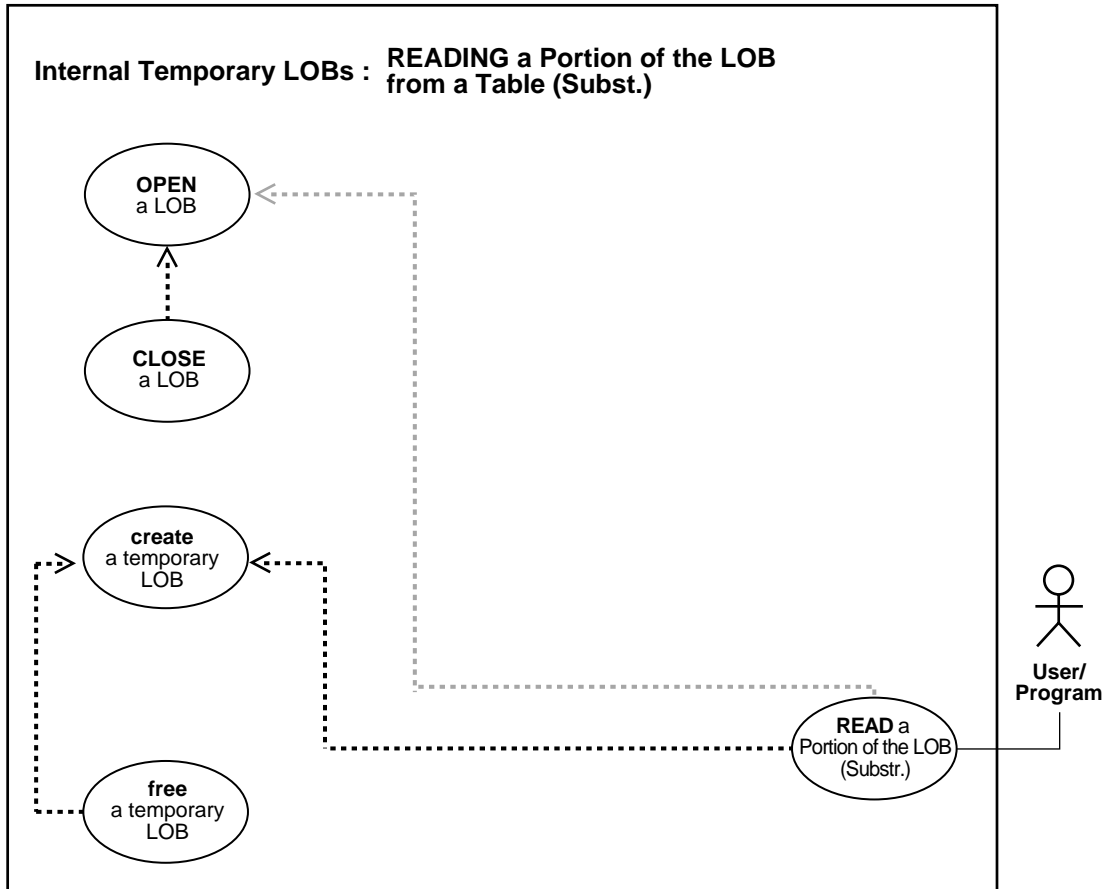
/* Release resources held by the Locators */
EXEC SQL FREE :Temp_loc;
EXEC SQL FREE :Lob_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    readTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```



## Reading Portion of Temporary LOB (Substr)

Figure 11–10 Use Case Diagram: Reading Portion of Temporary LOB from the Table (Substr)



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to read portion of a temporary LOB (substr).

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — SUBSTR
- C (OCI): There is no applicable syntax reference for this use case.
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB READ.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB LOAD. See also PL/SQL DBMS\_LOB.SUBSTR.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Reading a Portion of the LOB \(substr\)](#)" on page 10-126.

## Scenario

These examples show the operation in terms of reading a portion of the image in the AD\_PHOTO column.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Reading a Portion of Temporary LOB \(substr\)](#) on page 11-86
- C (OCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Reading a Portion of Temporary LOB \(substr\)](#) on page 11-87
- [C/C++ \(Pro\\*C/C++\): Reading a Portion of Temporary LOB \(substr\)](#) on page 11-89
- Visual Basic (OO4O): No example is provided with this release.

- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Reading a Portion of Temporary LOB (substr)

```

/* Reading portion of a temporary LOB using substr. [Example script: 3864.sql]
Example procedure substrTempLOB_proc is not part of DBMS_LOB package.
This example assumes you have a 'monitor_photo_3060_11001' file in a
directory which has a ADPHOTO alias */
CREATE or REPLACE PROCEDURE substrTempLOB_proc IS
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
  Amount        INTEGER := 32767;
  Bbuf          RAW(32767);
  Position      INTEGER :=128;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
  /* 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;

```

## COBOL (Pro\*COBOL): Reading a Portion of Temporary LOB (substr)

```

* Reading portion of a temporary LOB using substr
* [Example script: 3865.pco]
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.

```

## C/C++ (Pro\*C/C++): Reading a Portion of Temporary LOB (substr)

```

/* Reading a portion of a temporary LOB. [Example script: 3866.pc]
Pro*C/C++ lacks an equivalent embedded SQL form for the DEMS_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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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);

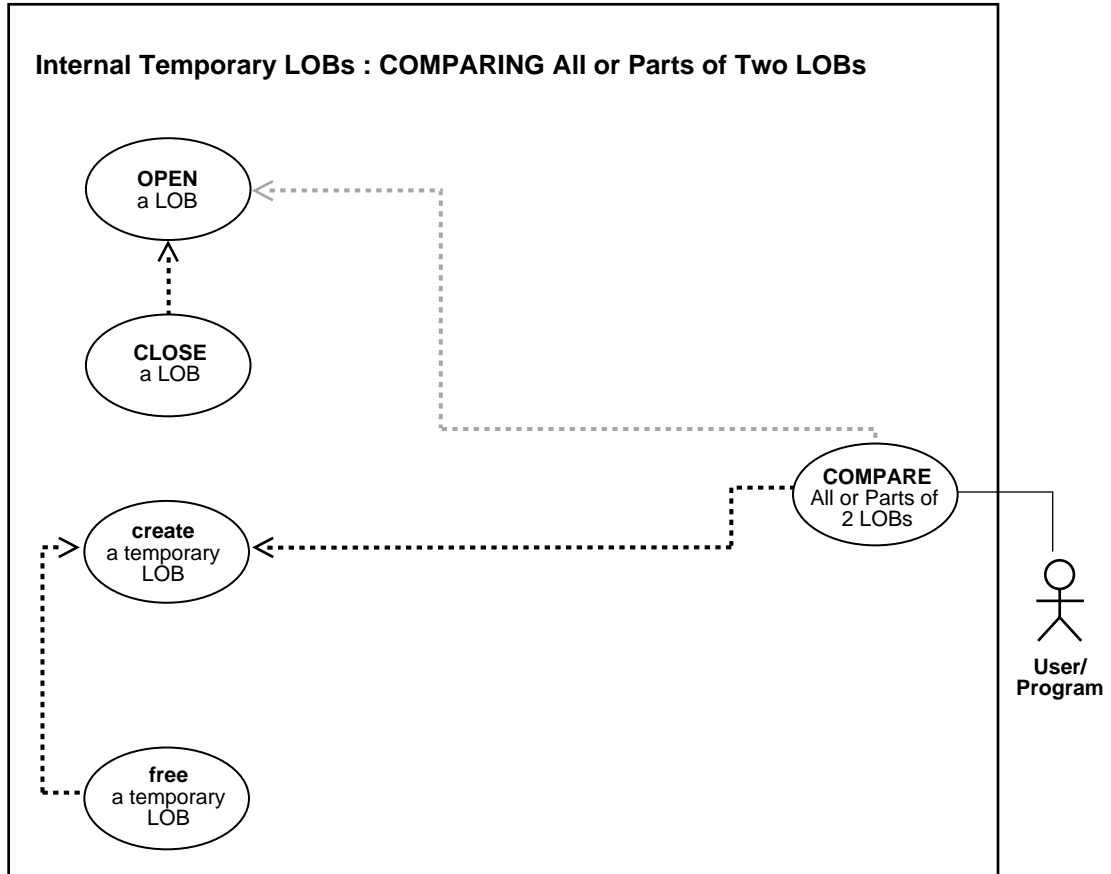
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* Allocate and Create the Temporary LOB: */
    EXEC SQL ALLOCATE :Temp_loc;
    EXEC SQL LOB CREATE TEMPORARY :Temp_loc;
    /* Allocate and Initialize the BFILE Locator: */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :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;
}
```

## Comparing All or Part of Two Temporary LOBs

Figure 11-11 Use Case Diagram: Comparing All or Part of Two Temporary LOBs



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to compare all or part of two temporary LOBs.



## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — COMPARE
- C (OCI): There is no applicable syntax reference for this use case.
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB COPY. See also PL/SQL DBMS\_LOB.COMPARE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB COPY. See also PL/SQL DBMS\_LOB.COMPARE.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "Java (JDBC): [Comparing All or Part of Two LOBs](#)" on page 10-134.

## Scenario

The following examples compare two composite advertisements to see whether they are different. Depending on the result of the comparison, the examples insert the composite into the table.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Comparing All or Part of Two Temporary LOBs](#) on page 11-93
- C (OCI): No example is provided with this release.
- COBOL (Pro\*COBOL): [Comparing All or Part of Two Temporary LOBs](#) on page 11-95
- C/C++ (Pro\*C/C++): [Comparing All or Part of Two Temporary LOBs](#) on page 11-97

- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Comparing All or Part of Two Temporary LOBs

```
/* Comparing all or part of two temporary LOBs [Example script: 3868.sql]
The procedure compareTwoTemporPersistLOBs_proc is not part
of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE compareTwoTmpPerLOBs_proc IS
    Lob_loc1 BLOB;
    Lob_loc2 BLOB;
    Temp_loc BLOB;
    Amount    INTEGER := 32767;
    Retval    INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT ad_composite INTO Lob_loc1 FROM Print_media
        WHERE product_ID = 3060 AND ad_id = 11001;
    SELECT ad_composite INTO Lob_loc2 FROM Print_media
        WHERE product_ID = 2268 AND ad_id = 21001;

    /* Copy one of the composite ad files into a temp LOB and convert
    it to a different format before comparing the ads : */
    DBMS_LOB.CREATETEMPORARY(Temp_loc, TRUE);
    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 you are done using it: */
    DBMS_LOB.FREETEMPORARY(Temp_loc);
```

END;

## COBOL (Pro\*COBOL): Comparing All or Part of Two Temporary LOBs

\* Comparing all or part of two temporary LOBs [Example script: 3869.pco]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BLOB-COMPARE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
```

```
01 USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_COMPOSITE INTO :BLOB1
    FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 3106 AND AD_ID = 13001
END-EXEC.

EXEC SQL
    SELECT AD_COMPOSITE INTO :BLOB2
    FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 2268 AND AD_ID = 21001
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 := DEMS_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.

```

## C/C++ (Pro\*C/C++): Comparing All or Part of Two Temporary LOBs

```

/* Comparing all or part of two temporary LOBs. [Example script: 3870.pc]
#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 ad_composite INTO :Lob_loc1
        FROM Print_media WHERE Product_ID = 3060 AND ad_id = 11001;
    EXEC SQL SELECT ad_composite INTO :Lob_loc2
        FROM Print_media WHERE Product_ID = 2268 AND ad_id = 21001;

```

```
/* 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;

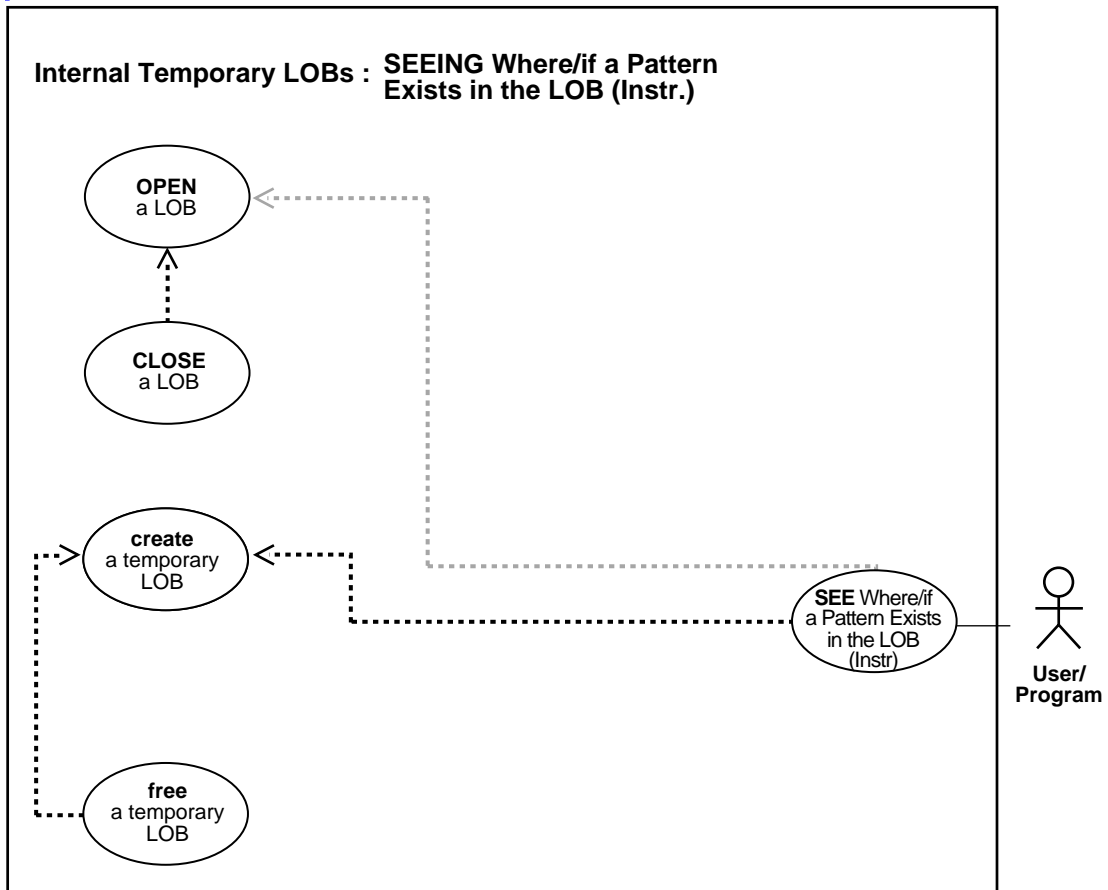
/* Compare the two Frames using DBMS_LOB.COMPARE() from within PL/SQL: */
EXEC SQL EXECUTE
BEGIN
    :RetVal := DBMS_LOB.COMPARE(:Lob_loc1, :Temp_loc, :Amount, 1, 1);
END;
END-EXEC;
if (0 == Retval)
    printf("Frames are equal\n");
else
    printf("Frames are not equal\n");
/* Closing the LOBs is mandatory 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;
}
```

## Determining If a Pattern Exists in a Temporary LOB (instr)

Figure 11-12 Use Case Diagram: Determining If a Pattern Exists in a Temporary LOB (instr)



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to see if a pattern exists in a temporary LOB (instr).

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — INSTR
- C (OCI): There is no applicable syntax reference for this use case.
- COBOL (Pro\*COBOL): *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB COPY. See also PL/SQL DBMS\_LOB.INSTR.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB COPY. See also DBMS\_LOB.INSTR.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Checking for Patterns in the LOB \(instr\)](#)" on page 10-142.

## Scenario

The following examples examine the advertisement text to see if the string "children" is present.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Determining If a Pattern Exists in a Temporary LOB](#) on page 11-100
- C (OCI): No example is provided with this release.
- COBOL (Pro\*COBOL): [Determining If a Pattern Exists in a Temporary LOB \(instr\)](#) on page 11-101
- Table , "C/C++ (Pro\*C/C++): [Determining If a Pattern Exists in a Temporary LOB \(instr\)](#)" on page 11-104
- Visual Basic (OO4O): No example is provided with this release.



- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Determining If a Pattern Exists in a Temporary LOB

```

/* Determining if a pattern exists in a temporary LOB. [Example script:
3871.sql]
    Procedure instrTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE instrTempLOB_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);
    SELECT ad_sourcetext INTO Lob_loc FROM Print_media
        WHERE Product_ID = 3060 AND ad_id = 11001;

    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;

```

## COBOL (Pro\*COBOL): Determining If a Pattern Exists in a Temporary LOB (instr)

```

* Determining if a pattern exists in a temporary LOB. [Example script: 3872.pco]
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CLOB-INSTR.
    ENVIRONMENT DIVISION.

```

## Determining If a Pattern Exists in a Temporary LOB (instr)

---

```
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_SOURCETEXT INTO :CLOB1
    FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 11001
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.
```

## C/C++ (Pro\*C/C++): Determining If a Pattern Exists in a Temporary LOB (instr)

```
/* Determining if a pattern exists in a temporary LOB using instr
   [Example script: 3873.pc] */
#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 instrTempLOB_proc()
{
    OCIClobLocator *Lob_loc, *Temp_loc;
    char *Pattern = "The End";
    unsigned int Length;
    int Position = 0;
    int Offset = 1;
    int Occurrence = 1;

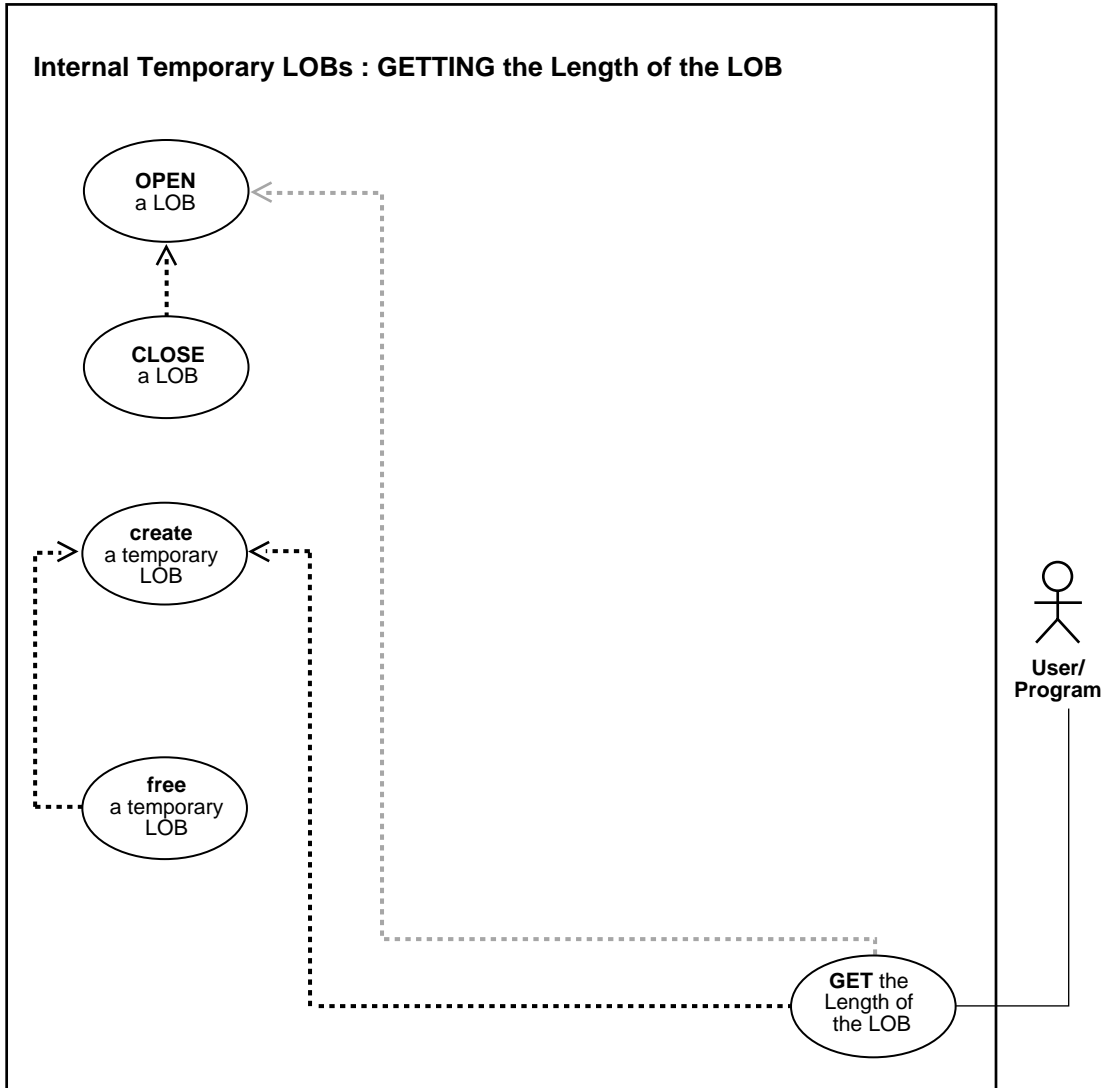
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    /* Allocate and Initialize the Persistent LOB: */
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT ad_sourcetext INTO :Lob_loc
        FROM print_media WHERE product_ID = 3060 AND ad_id = 11001;
    /* Allocate and Create the Temporary LOB: */
    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;
    instrTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```

## Finding the Length of a Temporary LOB

Figure 11-13 Use Case Diagram: Finding the Length of a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### **Purpose**

This procedure describes how to get the length of a temporary LOB.

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — GETLENGTH
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobGetLength
- **COBOL (Pro\*COBOL)** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DESCRIBE... GET LENGTH.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DESCRIBE...GET LENGTH
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Determining the Length of a LOB"](#) on page 10-150.

### **Scenario**

The following examples get the length of the photo LOB to determine whether it will run over the 4 gigabyte limit.

### **Examples**

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Finding the Length of a Temporary LOB](#) on page 11-107
- **C (OCI):** [Finding the Length of a Temporary LOB](#) on page 11-108

- [COBOL \(Pro\\*COBOL\): Finding the Length of a Temporary LOB](#) on page 11-110
- [C/C++ \(Pro\\*C/C++\): Finding the Length of a Temporary LOB](#) on page 11-112
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Finding the Length of a Temporary LOB

```
/* Finding the length of a temporary LOB. [Example script: 3874.sql]
   Procedure getLengthTempCLOB_proc is not part of 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('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
    DBMS_LOB.CREATETEMPORARY(tlob,TRUE);
    /* 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.');
```

## C (OCI): Finding the Length of a Temporary LOB

```
/* Finding the length of a temporary LOB. [Example script: 3875.c]
```



```

    This function takes a temporary LOB locator 'amount' as argument, and
    prints out the length of the corresponding LOB. The function returns
    0 if it completes successfully, -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 *)"ADPHOTO_DIR",
                          (ub2)strlen("ADPHOTO_DIR"),
                          (text *)"monitor_photo_3060_11001",
                          (ub2)strlen("monitor_photo_3060_11001")))
    {
        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;
}

```

## COBOL (Pro\*COBOL): Finding the Length of a Temporary LOB

\* Finding the length of a temporary LOB. [Example script: 3876.pco]

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-LOB-LENGTH.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) 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  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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Finding the Length of a Temporary LOB

```
/* Finding the length of a temporary LOB. [Example script: 3877.pc] */
#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 = "ADPHOTO_DIR", *Name = "modem_photo_2268_21001";
    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;

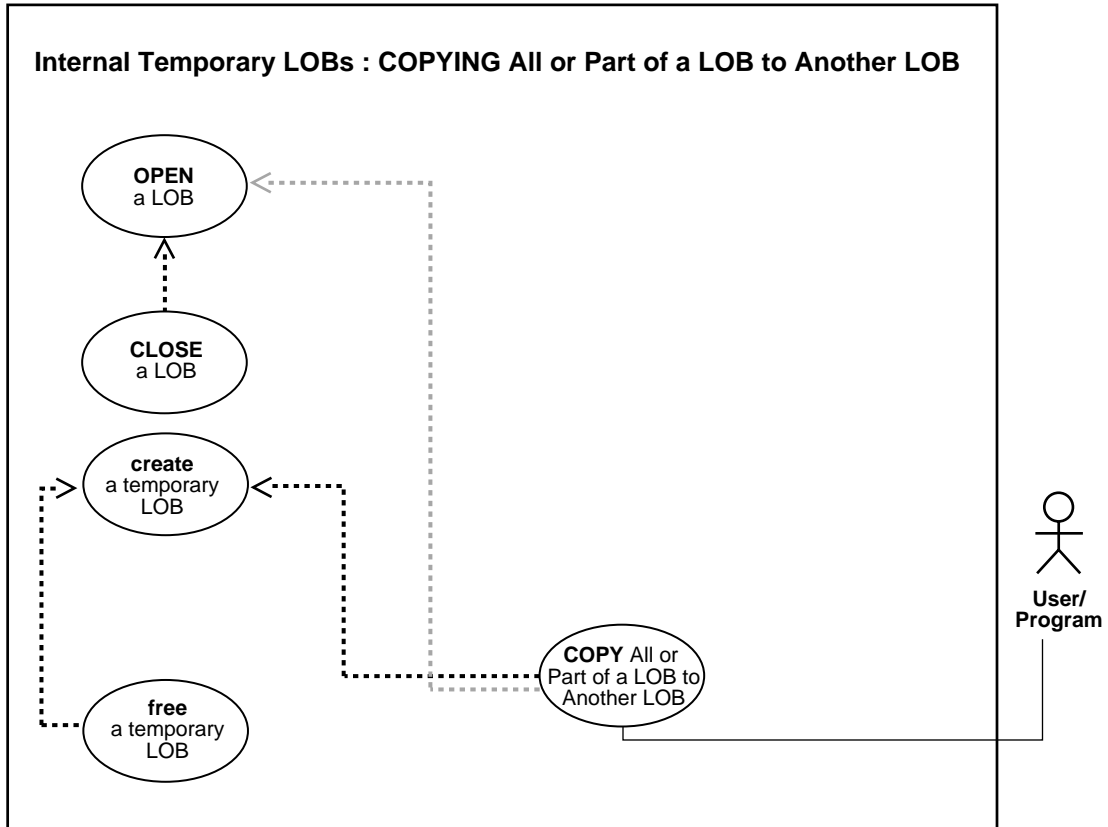
    /* Free the Temporary LOB: */
```

```
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;
}
```

## Copying All or Part of One Temporary LOB to Another

Figure 11-14 Use Case Diagram: Copying All or Part of One Temporary LOB to Another



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to copy all or part of one temporary LOB to another.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — COPY
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCopy.
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB COPY.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB COPY
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Copying All or Part of One LOB to Another LOB](#)" on page 10-162.

## Scenario

Assume the following table:

```
CREATE TABLE ad_library_tab of adheader_typ;  
  
INSERT INTO adheader_tab  
  (SELECT * FROM adheaderlib_tab vtab1  
   WHERE T2.creation_date = '08/16/2001');
```

These examples create a new LOB locator in table `adheader_tab`, and copy LOB data from `vtab1` to the location pointed to by a new LOB locator inserted into table `adheader_tab`.

## Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Copying All or Part of One Temporary LOB](#) on page 11-116



- [C \(OCI\): Copying All or Part of One Temporary LOB to Another](#) on page 11-118
- [COBOL \(Pro\\*COBOL\): Copying All or Part of One Temporary LOB to Another](#) on page 11-121
- [C/C++ \(Pro\\*C/C++\): Copying All or Part of One Temporary LOB to Another](#) on page 11-123
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Copying All or Part of One Temporary LOB

```

/* Copying all or part of one temporary LOB to another. [Example script:
3880.sql]
   Procedure copyTempLOB_proc is not part of 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('ADPHOTO_DIR', 'monitor_photo_3060_11001');
  Amount        INTEGER := 32767;
BEGIN
  DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE);
  DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
  /* 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 tlob_src: */
  DBMS_LOB.COPY(Dest_loc2, Dest_loc, Amount, Dest_pos, Src_pos);

```

```

        COMMIT;
    EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Operation failed');
        DEMS_LOB.CLOSE(Dest_loc);
        DEMS_LOB.CLOSE(Dest_loc2);
        DEMS_LOB.CLOSE(Src_loc);
        DEMS_LOB.FREETEMPORARY(Dest_loc);
        DEMS_LOB.FREETEMPORARY(Dest_loc2);
END;
```

## C (OCI): Copying All or Part of One Temporary LOB to Another

```

/* Copying all or part of one temporary LOB to another. [Example script: 3881.c]
   This function 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, -1 otherwise. */
sb4 copy_temp_lobs (OCIError      *errhp,
                   OCISvcCtx     *svchp,
                   OCISmt       *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 *)"ADPHOTO_DIR",
                    (ub2)strlen("ADPHOTO_DIR"),
                    (text *)"monitor_photo_3060_11001",
                    (ub2)strlen("monitor_photo_3060_11001")))
{
    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 lobes 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;
}

```

## COBOL (Pro\*COBOL): Copying All or Part of One Temporary LOB to Another

```

* Copying all or part of one temporary LOB to another
* [Example script: 3882.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-COPY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 SQLEERROR 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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Copying All or Part of One Temporary LOB to Another

```
/* Copying all or part of one temporary LOB to another. [Example script:
3883.pc] */
#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;
    OCIFileLocator *Lob_loc;
    char *Dir = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;
```

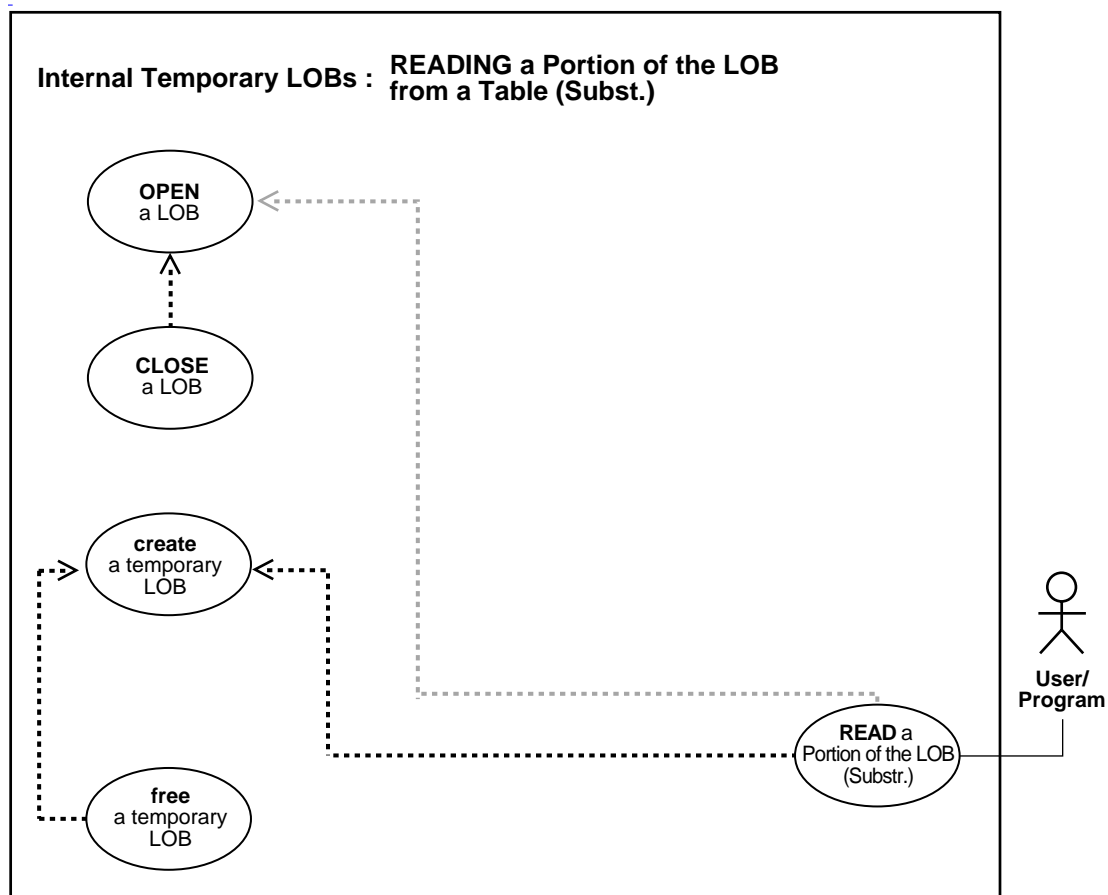
```
EXEC SQL LOB CREATE TEMPORARY :Temp_loc2;
/* Allocate and Initialize the BFILE Locator: */
EXEC SQL ALLOCATE :Lob_loc;
EXEC SQL LOB FILE SET :Lob_loc DIRECTORY = :Dir, FILENAME = :Name;
/* Opening the LOBs is Optional: */
EXEC SQL LOB OPEN :Lob_loc READ ONLY;
EXEC SQL LOB OPEN :Temp_loc1 READ WRITE;
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;
}
```



## Copying a LOB Locator for a Temporary LOB

Figure 11-15 Use Case Diagram: Copying a LOB Locator for a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to copy a LOB locator for a temporary LOB.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* "DBMS\_LOB" — CREATETEMPORARY, LOADFROMFILE, FREETEMPORARY
- PL/SQL (DBMS\_LOB): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobLocatorIsInit
- COBOL (Pro\*COBOL): *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB ASSIGN.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB ASSIGN
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Copying a LOB Locator](#)" on page 10-170.

## Scenario

This generic operation copies one temporary LOB locator to another.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Copying a LOB Locator for a Temporary LOB](#) on page 11-126
- C (OCI): [Copying a LOB Locator for a Temporary LOB](#) on page 11-128
- COBOL (Pro\*COBOL): [Copying a LOB Locator for a Temporary LOB](#) on page 11-130
- C/C++ (Pro\*C/C++): [Copying a LOB Locator for a Temporary LOB](#) on page 11-132
- Visual Basic (OO4O): No example is provided with this release.

- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Copying a LOB Locator for a Temporary LOB

---



---

**Note:** Assigning one LOB to another using PL/SQL entails using the "=" sign. This is discussed in more detail with regard to "[Read Consistent Locators](#)" in [Chapter 5, "Large Objects: Advanced Topics"](#).

---



---

```

/* Copying a LOB locator for a temporary LOB.
   Procedure copyTempLOBLocator_proc is not part of 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('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
  DBMS_LOB.CREATETEMPORARY(Lob_loc1,TRUE);
  DBMS_LOB.CREATETEMPORARY(Lob_loc2,TRUE);
  /* 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;

## C (OCI): Copying a LOB Locator for a Temporary LOB

\* Copying a LOB locator for a temporary LOB. [Example script: 3886.c]  
 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 *)"PHOTO_DIR",
                        (ub2)strlen("ADPHOTO_DIR"),
                        (text *)"monitor_photo_3060_11001",
                        (ub2)strlen("monitor_photo_3060_11001")))
    {
        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 lobbs */
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 lobbs 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;
}
}
```

## COBOL (Pro\*COBOL): Copying a LOB Locator for a Temporary LOB

```
* Copying a LOB locator for a temporary LOB
* [Example script: 3887.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-COPY-LOCATOR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".

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 SQLEERROR 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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Copying a LOB Locator for a Temporary LOB

```
/* copying a LOB locator for a temporary LOB. [Example script: 3888.pc]
#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

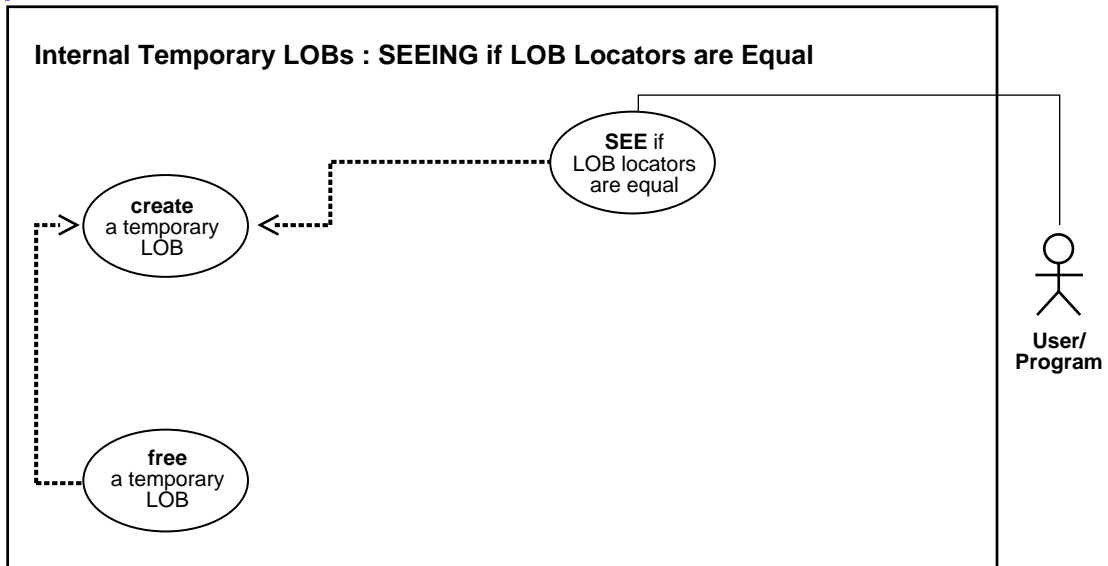
    /* Release resources held by the Locators: */
}
```

```
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;
}
```

## Is One Temporary LOB Locator Equal to Another

Figure 11–16 Use Case Diagram: Is One Temporary LOB Locator Equal to Another



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to see if one LOB locator for a temporary LOB is equal to another.

### Usage Notes

If two locators are equal they refer to the same version of the LOB data (see "[Read Consistent Locators](#)" in Chapter 5, "[Large Objects: Advanced Topics](#)").

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobIsEqual
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB ASSIGN. See also C(OCI) function, OCILobIsEqual
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Checking If One LOB Locator Is Equal to Another"](#) on page 10-177.

### Scenario

Not applicable.

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package). No example is provided with this release.
- C (OCI): [Is One LOB Locator for a Temporary LOB Equal to Another](#) on page 11-136
- COBOL (Pro\*Cobol): No example is provided with this release.
- C/C++ (Pro\*C/C++): [Is One LOB Locator for a Temporary LOB Equal to Another](#) on page 11-138
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Is One LOB Locator for a Temporary LOB Equal to Another

```
/* Equality - Is one temporary LOB locator equal to another? [Example script:  
3889.c] */
```

```
sb4 ck_isequal (OCIError      *errhp,  
               OCISvcCtx     *svchp,  
               OCIStmt       *stmthp,  
               OCIEnv        *envhp)  
{  
    OCILobLocator *loc1;
```

```

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;
}
OCILobDescriptor free????
return 0;
}

```

## C/C++ (Pro\*C/C++): Is One LOB Locator for a Temporary LOB Equal to Another

```
/* Equality - Is one LOB locator for a temporary LOB equal to another? */
/* [Example script: 3890.pc] */

#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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);

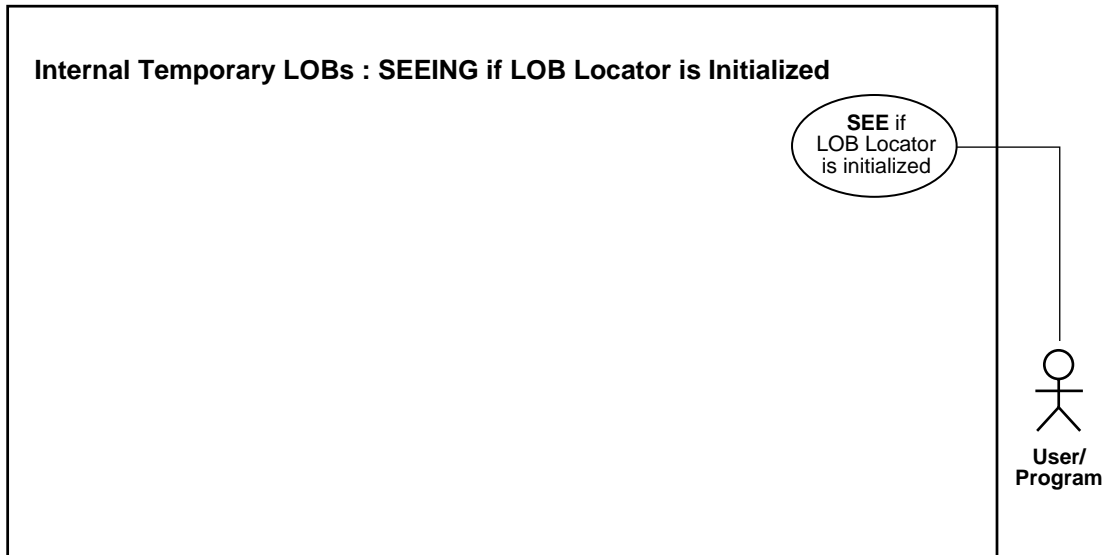
/* The value of isEqual should be 1 (TRUE) in this case: */
printf("Locators %s equal\n", isEqual ? "are" : "are not");

/* 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 implicitly 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;
}
```

## Determining if a LOB Locator for a Temporary LOB Is Initialized

*Figure 11-17 Use Case Diagram: Determining If a LOB Locator for a Temporary LOB Is Initialized*



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### **Purpose**

This procedure describes how to see if a LOB locator for a temporary LOB is initialized.

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): There is no applicable syntax reference for this use case.



- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobLocatorIsInit
- COBOL: A syntax reference is not applicable in this release.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB CREATE TEMPORARY. See also C(OCI) function, OCILobLocatorIsInit
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "C (OCI): Checking If a LOB Locator Is Initialized"

### 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".

### Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package). No example is provided with this release.
- [C \(OCI\): Determining If a LOB Locator for a Temporary LOB Is Initialized](#) on page 11-141
- COBOL (Pro\*Cobol): No example is provided with this release.
- [C/C++ \(Pro\\*C/C++\): Determining If a LOB Locator Is Initialized](#) on page 11-142
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Determining If a LOB Locator for a Temporary LOB Is Initialized

```
/* Is a LOB locator for a temporary LOB is initialized? [Example script:3892.c]
   This function takes a LOB locator and checks if it is initialized. If it is
   initialized, it prints out a message, "LOB is initialized".
   Otherwise, it says "LOB is not initialized". This function returns
   0 if it completes successfully, -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;
}

```

## C/C++ (Pro\*C/C++): Determining If a LOB Locator Is Initialized

```

/* Is a LOB locator for a temporary LOB initialized? [Example script: 3893.pc]
*/

#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;

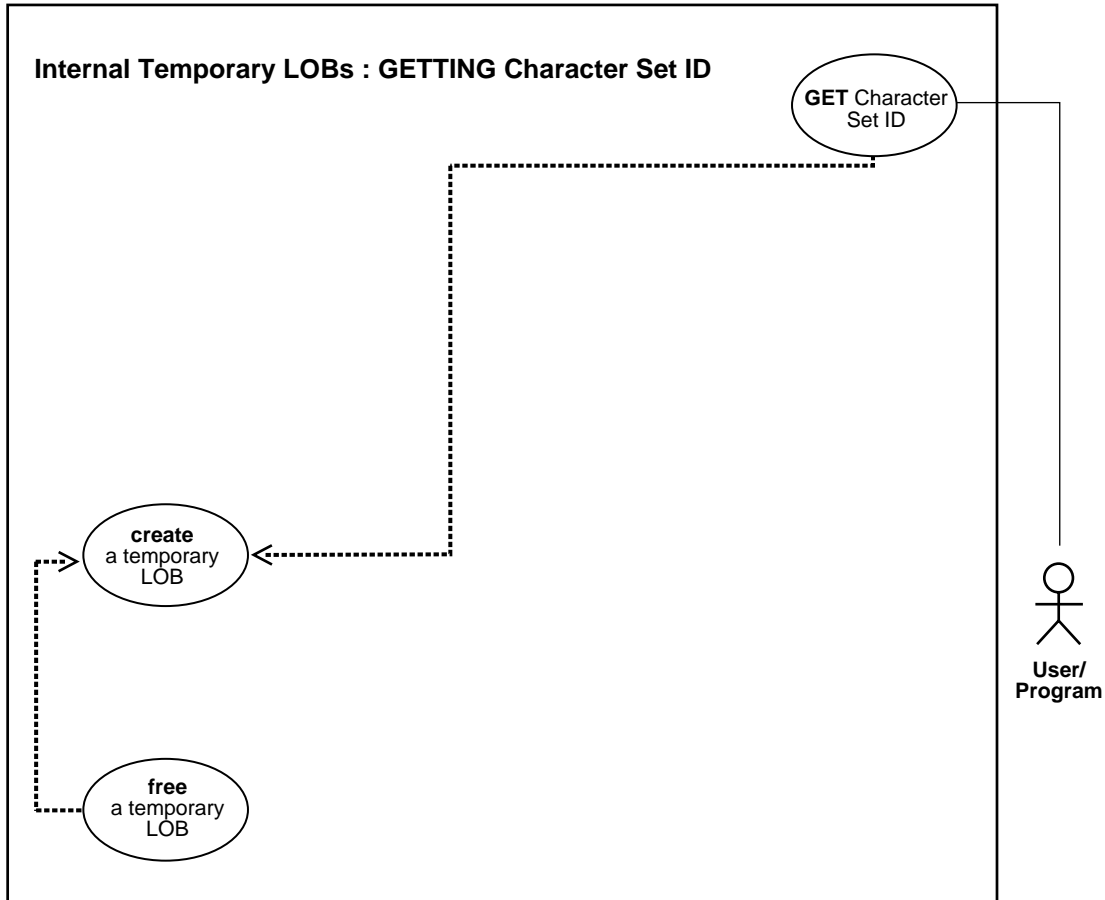
```

```
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;
}
```

## Finding Character Set ID of a Temporary LOB

Figure 11-18 Use Case Diagram: Finding Character Set ID for a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to get the character set ID of a temporary LOB.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): A syntax reference is not applicable with this release.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCharSetId
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): There is no applicable syntax reference for this use case.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "C (OCI): Determining Character Set ID".

## Scenario

This function takes a LOB locator and prints the character set id of the LOB.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package). No example is provided with this release.
- C (OCI): [Finding Character Set ID of a Temporary LOB](#) on page 11-145
- COBOL (Pro\*Cobol): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Finding Character Set ID of a Temporary LOB

```

/* Finding the character set id of a temporary LOB. [Example script: 3894.c]
   This function takes a LOB locator and prints the character set id of the LOB.
   The function returns 0 if it completes successfully, -1 if it doesn't. */

```

```

sb4 get_charsetid (OCIlobLocator *lob_loc,
                  OCIError      *errhp,
                  OCISvcCtx     *svchp,
                  OCISstmt      *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;
    }

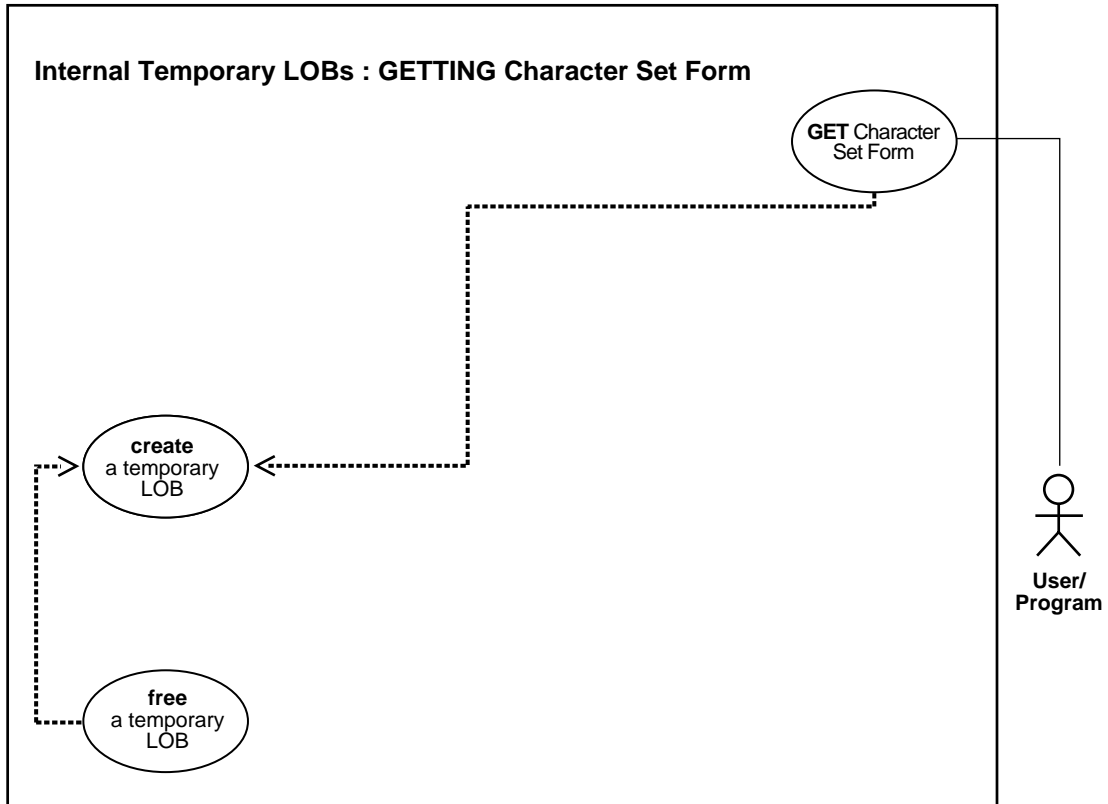
    if (OCIlobCharSetId(envhp, errhp, lob_loc, &charsetid))
    {
        printf ("FAILED: OCIlobCharSetId call\n");
        return -1;
    }
    fprintf (stderr,"LOB charsetid is %d\n",charsetid);
    if(OCIlobFreeTemporary(svchp,errhp,lob_loc))
    {
        printf("FAILED: OCIlobFreeTemporary \n");
        return -1;
    }

    return 0;
}

```

## Finding Character Set Form of a Temporary LOB

Figure 11–19 Use Case Diagram: Finding Character Set Form of a Temporary LOB



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to get the character set form of a temporary LOB.

### Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): A syntax reference is not applicable in this release.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobCharSetForm
- COBOL (Pro\*COBOL): There is no applicable syntax reference for this use case.
- C/C++ (Pro\*C/C++): There is no applicable syntax reference for this use case.
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "C (OCI): Determining Character Set Form".

## Scenario

This function takes a LOB locator and prints the character set form for the LOB.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package). No example is provided with this release.
- C (OCI): [Finding Character Set Form of a Temporary LOB](#) on page 11-148
- COBOL (Pro\*Cobol): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Finding Character Set Form of a Temporary LOB

```
/* Finding the character set form of a temporary LOB [Example script: 3895.c]
   This function takes a LOB locator and prints out the character set form for
   the LOB. It returns 0 if it completes successfully, -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;
    }

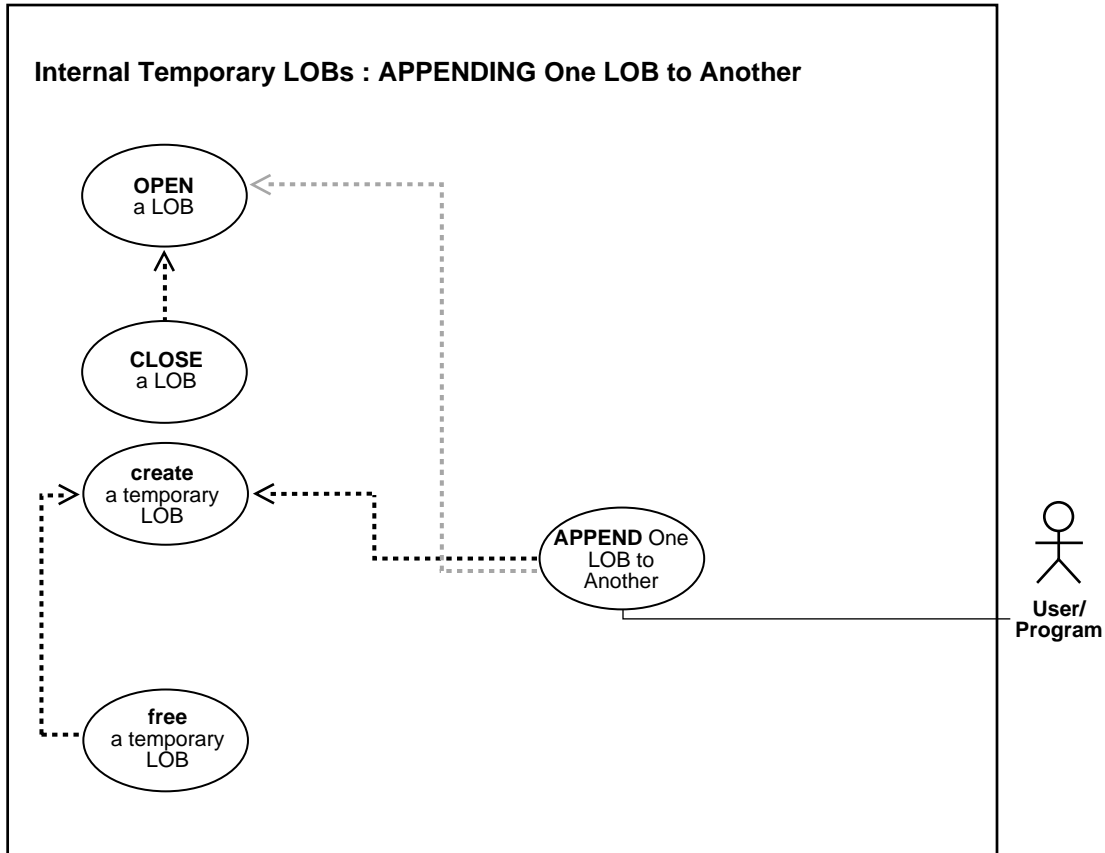
    if (OCILOBCharSetForm(envhp,errhp, lob_loc, &charsetform))
    {
        printf ("FAILED: OCILOBCharSetForm call\n");
        return -1;
    }
    fprintf (stderr,"LOB charsetform is %d\n",charsetform);

    if(OCILOBFreeTemporary(svchp,errhp,lob_loc))
    {
        printf("FAILED: OCILOBFreeTemporary \n");
        return -1;
    }
    return 0;
}

```

## Appending One Temporary LOB to Another

Figure 11–20 Use Case Diagram: Appending One Temporary LOB to Another



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to append one temporary LOB to another.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — APPEND
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobAppend.
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB APPEND.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB APPEND
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "Java (JDBC): [Appending One LOB to Another](#)" on page 10-201.

## Scenario

These examples deal with the task of appending one segment of sound to another.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): [Appending One Temporary LOB to Another](#) on page 11-151
- C (OCI): [Appending One Temporary LOB to Another](#) on page 11-152
- COBOL (Pro\*COBOL): [Appending One Temporary LOB to Another](#)
- C/C++ (Pro\*C/C++): [Appending One Temporary LOB to Another](#)
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Appending One Temporary LOB to Another

```

/* Appending one temporary LOB to another [Example script: 3896.sql]
   Procedure appendTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE appendTempLOB_proc IS
    Dest_loc2 CLOB;
    Dest_loc  CLOB;
    Amount    NUMBER;
    Src_loc   BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
BEGIN
    DBMS_LOB.CREATETEMPORARY(Dest_loc,TRUE);
    DBMS_LOB.CREATETEMPORARY(Dest_loc2,TRUE);
    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 lobbs 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;

```

## C (OCI): Appending One Temporary LOB to Another

```

/* Appending one temporary LOB to another [Example script: 3897.c]
   This function takes two temporary LOB locators and appends the second
   LOB to the first one. It returns 0 if it completes successfully, -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 monitor_photo file */
if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"PHOTO_DIR",
                     (ub2)strlen("ADPHOTO_DIR"),
                     (text *)"monitor_photo_3060_11001",
                     (ub2)strlen("monitor_photo_3060_11001")))
{
    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 lob: */
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;
}
    
```

## COBOL (Pro\*COBOL): Appending One Temporary LOB to Another

```

* Appending one temporary LOB to another [Example script: 3898.pco]
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 "SAMP/SAMP".

* 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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_11001" 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.
    
```

## C/C++ (Pro\*C/C++): Appending One Temporary LOB to Another

```

/* Appending one temporary LOB to another. [Example script: 3899.pc] */

#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

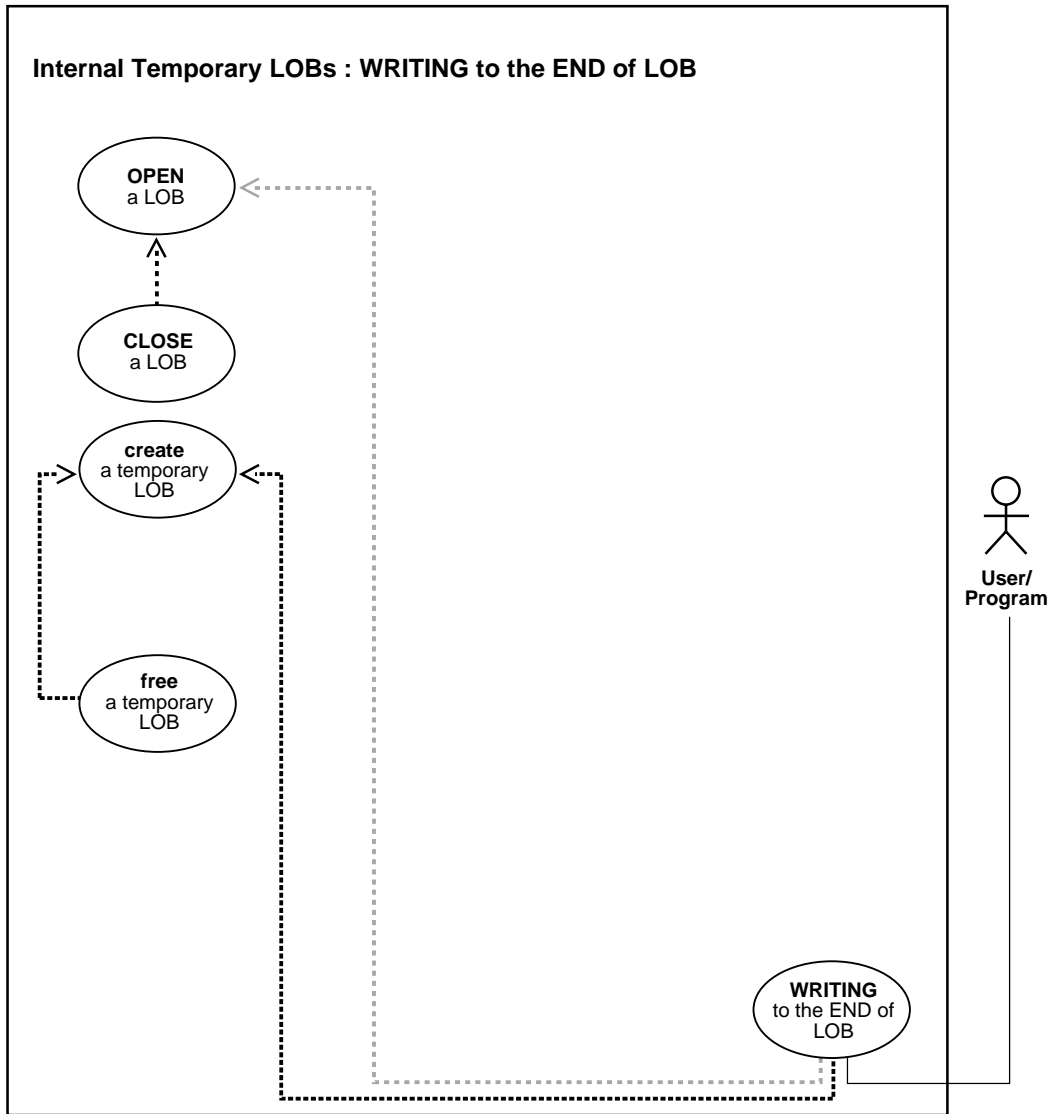
/* Release resources held by the Locators: */
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc1;
EXEC SQL FREE :Temp_loc2;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    appendTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}

```

## Write-Appending to a Temporary LOB

Figure 11-21 Use Case Diagram: Write-Appending to a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### **Purpose**

This procedure describes how to write append to a temporary LOB.

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — WRITEAPPEND
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobWriteAppend
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB WRITE.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB WRITE
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), ["Java \(JDBC\): Writing to the End of \(Append-Write to\) a LOB"](#) on page 10-210.

### **Scenario**

These examples read in 32767 bytes of data from an image file and append it to a temporary LOB.

### **Examples**

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Writing-Appending to a Temporary LOB](#) on page 11-161
- **C (OCI):** [Writing-Appending to a Temporary LOB](#) on page 11-162

- [COBOL \(Pro\\*COBOL\): Write-Appending to a Temporary LOB](#) on page 11-164
- [C/C++ \(Pro\\*C/C++\): Write-Appending to a Temporary LOB](#) on page 11-166
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Writing-Appending to a Temporary LOB

```
/* Write-Appending to a temporary LOB. [Example script: 3900.sql]
   Procedure writeAppendTempLOB_proc is not part of DBMS_LOB package.
   This procedure reads in 32767 bytes of data from the monitor_photo_3060_11001
   file and appends it to a temporary LOB. */

CREATE OR REPLACE PROCEDURE writeAppendTempLOB_proc IS
    Lob_loc      BLOB;
    Buffer        RAW(32767);
    Src_loc      BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
    Amount       Binary_integer := 32767;
    Position     Binary_integer := 128;
BEGIN
    DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
    /* 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;
```

## C (OCI): Writing-Appending to a Temporary LOB

```
/* Write-Appending to a temporary LOB [Example script: 3901.c] */

#define MAXBUFLLEN 32767
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 descriptors: */
    (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);
}
```

## COBOL (Pro\*COBOL): Write-Appending to a Temporary LOB

```
* Write-appending to a temporary LOB. [Example script: 3902.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-APPEND-TEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_11001" 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.
```

## C/C++ (Pro\*C/C++): Write-Appending to a Temporary LOB

```
/* Write-Appending to a temporary LOB. [Example script: 3903.pc] */

#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

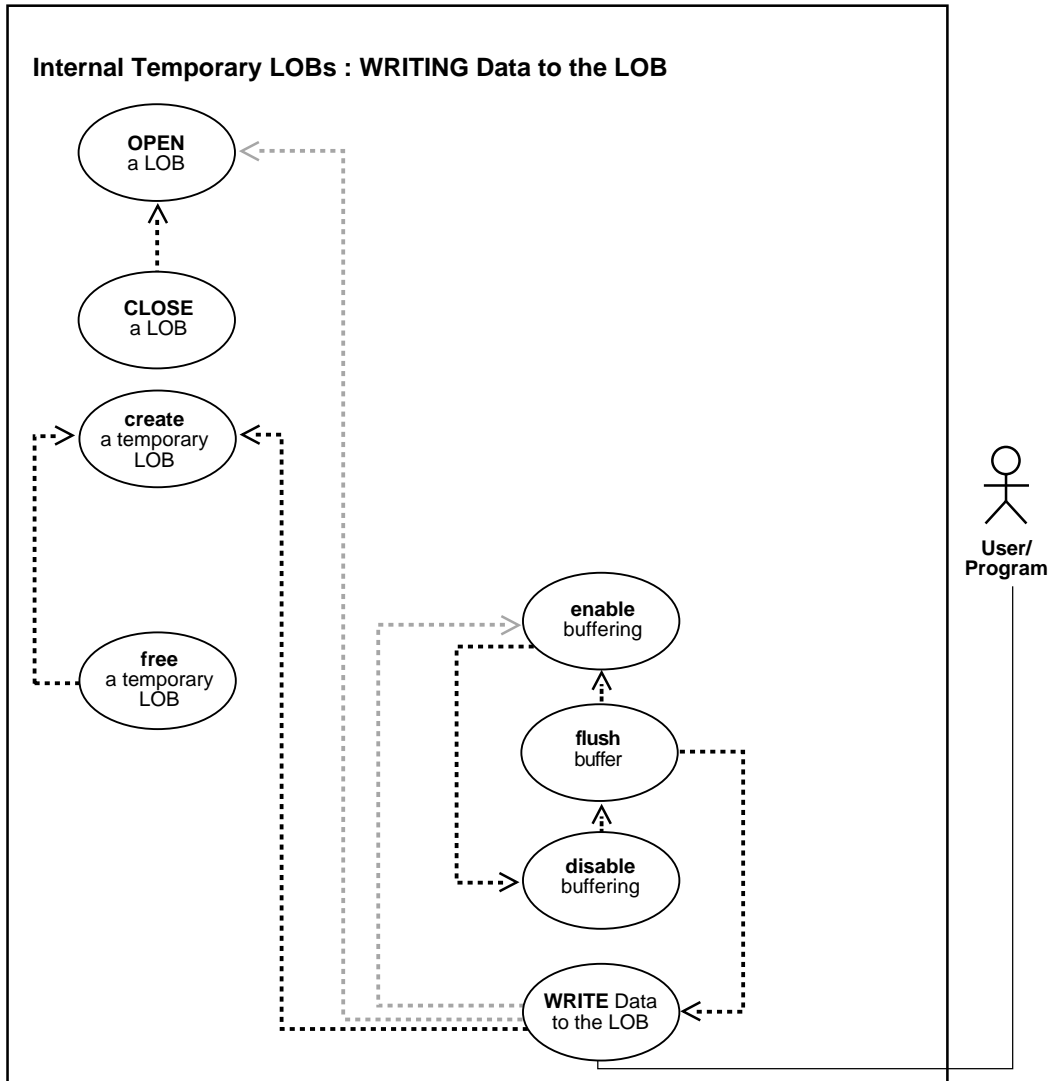
/* Closing the LOBs is Mandatory if they have been Opened: */
EXEC SQL LOB CLOSE :Lob_loc;
EXEC SQL LOB CLOSE :Temp_loc;

/* 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;
}
```

## Writing Data to a Temporary LOB

Figure 11–22 Use Case Diagram: Writing Data to a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

## Purpose

This procedure describes how to write data to a temporary LOB.

## Usage Notes

**Stream Writing** The most efficient way to write large amounts of LOB data is to use `OCILOBWrite()` with the streaming mechanism enabled using 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.

**Using DBMS\_LOB.WRITE() to Write Data to a Temporary BLOB** When you are passing a hexadecimal string to `DBMS_LOB.WRITE()` to write data to a BLOB, use the following guidelines:

- The `amount` parameter should be  $\leq$  the `buffer length` parameter
- The length of the buffer should be  $((\text{amount} * 2) - 1)$ . This guideline exists because the two characters of the string are seen as one hexadecimal character (and an implicit hexadecimal-to-raw conversion takes place), that is, every two bytes of the string are converted to one raw byte.

The following example is *correct*:

```
/* Writing data to a temporary LOB. [Example script: 3905.sql]

declare
  blob_loc BLOB;
  rawbuf RAW(10);
  an_offset INTEGER := 1;
  an_amount BINARY_INTEGER := 10;
begin
  select blob_col into blob_loc from a_table
  where id = 1;
  rawbuf := '1234567890123456789';
  dbms_lob.write(blob_loc, an_amount, an_offset,
  rawbuf);
  commit;
end;
```

Replacing the value for 'an\_amount' in the previous example with the following values, yields error message, ora\_21560:

```
an_amount BINARY_INTEGER := 11;  
or  
an_amount BINARY_INTEGER := 19;
```

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — WRITE
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILOBWRITE
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB WRITE.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB WRITE
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Writing Data to a LOB](#)" on page 10-226.

## Scenario

The example procedures allow the `ad_sourcetext` data (the text for the advertisement) to be updated by writing data to the LOB.

## Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Writing Data to a Temporary LOB](#) on page 11-170
- **C (OCI):** [Writing Data to a Temporary LOB](#) on page 11-171
- **COBOL (Pro\*COBOL):** [Writing Data to a Temporary LOB](#) on page 11-174
- **C/C++ (Pro\*C/C++):** [Writing Data to a Temporary LOB](#) on page 11-176

- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Writing Data to a Temporary LOB

```
/* Writing data to a temporary LOB [Example script: 3909.sql]
   Procedure writeDataToTempLOB_proc is not part of 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);
  /* 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 := 'abcdefghijklmnopqrstuvwxy';

  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;
```

## C (OCI): Writing Data to a Temporary LOB

```
/* Writing data to a temporary LOB. [Example script: 3910] */
/* This example illustrates streaming writes with polling */
```

```
#define MAXBUFLLEN 32767
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[MAXBUFLLEN];
sb4 err;

/* Allocate the locators descriptors: */
(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 > MAXBUFLLEN)
    nbytes = MAXBUFLLEN; /* 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 <= MAXBUFLLEN ) so we can WRITE in ONE piece: */
}

```



```

checkerr (errhp, OCILobWrite (svchp, errhp, tclob, &amp;mp,
                             offset, bufp, nbytes,
                             OCI_ONE_PIECE, (dvoid *)0,
                             (sb4 (*) (dvoid*, dvoid*, ub4*, ub1 *) )0,
                             0, SQLCS_IMPLICIT));
}
else
{
    /* Here (Total > MAXBUFLLEN ) so we use Streaming via Standard Polling: */
    /* WRITE the FIRST piece. Specifying FIRST initiates Polling: */
    err = OCILobWrite (svchp, errhp, tclob, &amp;mp,
                     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 > MAXBUFLLEN)
            nbytes = MAXBUFLLEN;          /* Still have more pieces to go */
        else
        {
            nbytes = remainder;          /* Here, (remainder <= MAXBUFLLEN) */
            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, &amp;mp,
                             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, &amp;tp,
                          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);
}

```

## COBOL (Pro\*COBOL): Writing Data to a Temporary LOB

```

* Writing data to a temporary LOB [Example script: 3911.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-TEMP.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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.
```

## C/C++ (Pro\*C/C++): Writing Data to a Temporary LOB

```
/* Writing data to a temporary LOB. [Example script: 3912] */  
  
#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);
    }

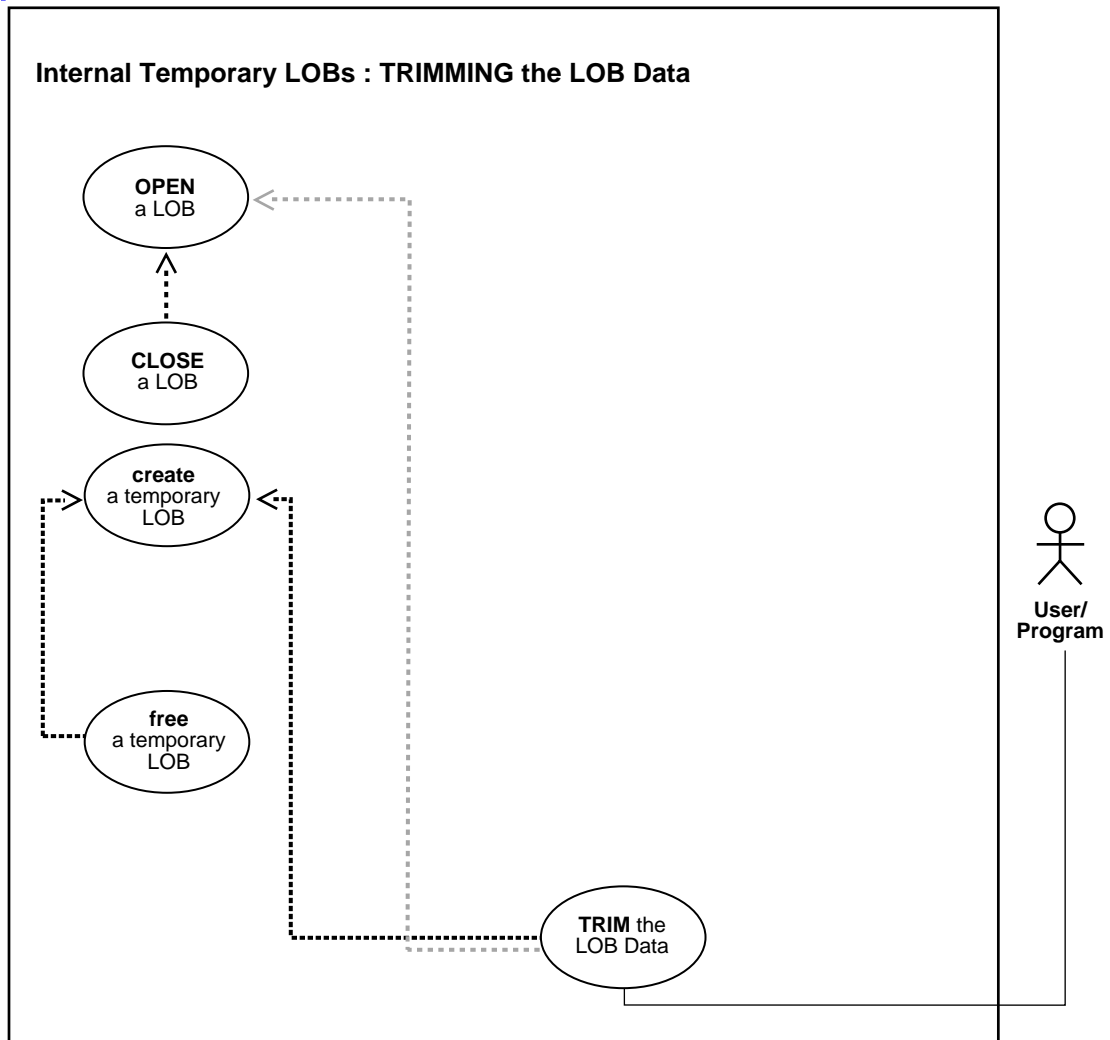
```

```
    }
    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;
}
```

## Trimming Temporary LOB Data

Figure 11–23 Use Case Diagram: Trimming Temporary LOB Data



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### **Purpose**

This procedure describes how to trim temporary LOB data.

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB):** *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — TRIM
- **C (OCI):** *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobTrim
- **COBOL (Pro\*COBOL):** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB TRIM.
- **C/C++ (Pro\*C/C++):** *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB TRIM
- **Visual Basic (OO4O):** There is no applicable syntax reference for this use case.
- **Java (JDBC):** See [Chapter 10, "Internal Persistent LOBs"](#), "[Java \(JDBC\): Trimming LOB Data](#)" on page 10-236.

### **Scenario**

The following examples access text (CLOB data) referenced in the `ad_finaltext` column of table `adheader_tab`, and trim it.

### **Examples**

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Trimming Temporary LOB Data](#) on page 11-180
- **C (OCI):** [Trimming Temporary LOB Data](#) on page 11-181



- [COBOL \(Pro\\*COBOL\): Trimming Temporary LOB Data](#) on page 11-184
- [C/C++ \(Pro\\*C/C++\): Trimming Temporary LOB Data](#) on page 11-185
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Trimming Temporary LOB Data

```

/* Trimming temporary LOB data. [Example script: 3914.sql]
   Procedure trimTempLOB_proc is not part of DBMS_LOB package. */

CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
  Lob_loc      CLOB;
  Amount       number;
  Src_loc      BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
  TrimAmount   number := 100;
BEGIN
  /* Create a temporary LOB: */
  DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);
  /* 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;
```

## C (OCI): Trimming Temporary LOB Data

```

/* Trimming temporary LOB data. [Example script: 3915.c]

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 monitor_photo file: */
    if(OCILobFileSetName(envhp, errhp, &bfile, (text *)"ADPHOTO_DIR",
                          (ub2)strlen("ADPHOTO_DIR"),
                          (text *)"monitor_photo_3060_11001",
                          (ub2)strlen("monitor_photo_3060_11001")))
    {
        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;  
}
```

## COBOL (Pro\*COBOL): Trimming Temporary LOB Data

```
* Trimming temporary LOB data. [Example script: 3916.pco]  
IDENTIFICATION DIVISION.  
PROGRAM-ID. TEMP-LOB-TRIM.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 USERID    PIC X(11) 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 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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.  
    MOVE 9 TO DIR-ALIAS-LEN.
```

```

MOVE "keyboard_photo_3106_13001" 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.

```

## C/C++ (Pro\*C/C++): Trimming Temporary LOB Data

```
/* Trimming temporary LOB data. [Example script: 3917.pc] */
```

```
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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

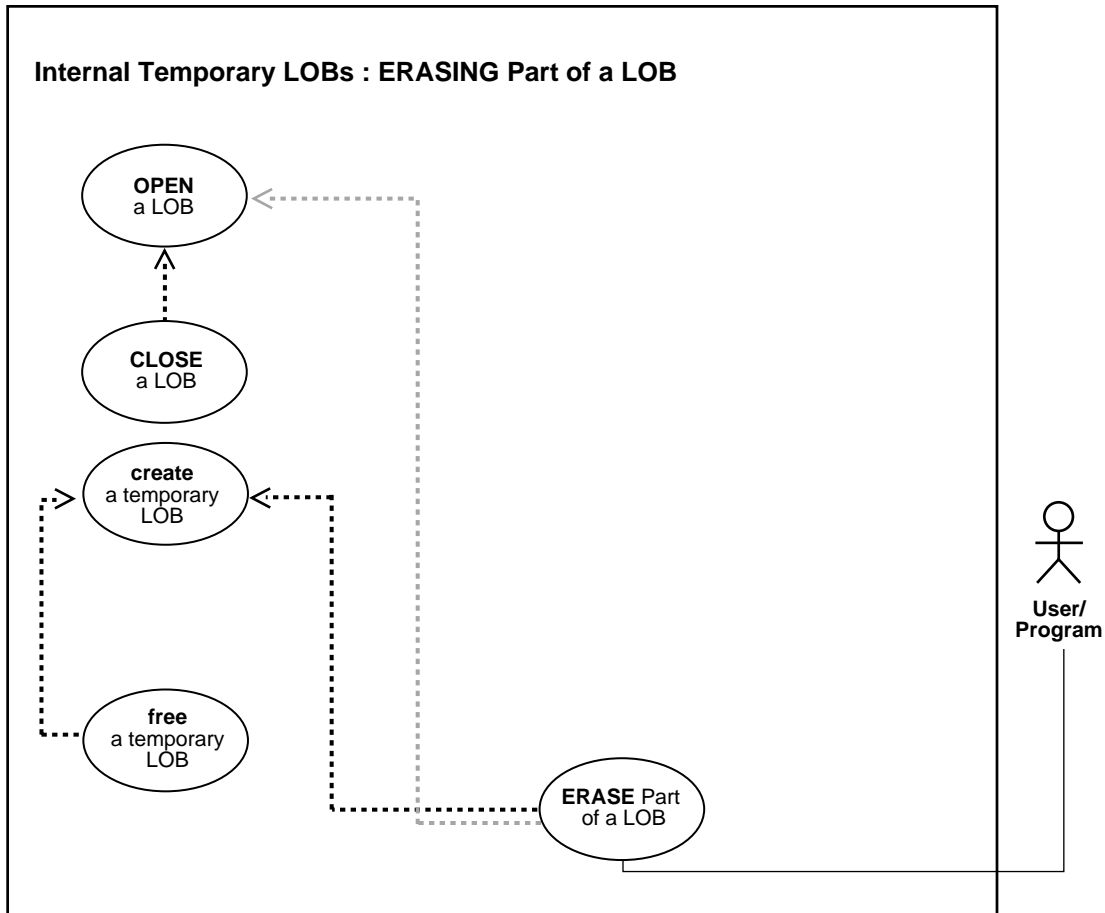
/* Free the Temporary LOB: */
EXEC SQL LOB FREE TEMPORARY :Temp_loc;

/* 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;
}
```

## Erasing Part of a Temporary LOB

Figure 11–24 Use Case Diagram: Erasing Part of a Temporary LOB



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to erase part of a temporary LOB.



## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — ERASE
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobErase
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB ERASE.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB ERASE
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See "[Java \(JDBC\): Erasing Part of a LOB](#)" on page 10-248 in [Chapter 10, "Internal Persistent LOBs"](#).

## Scenario

Not applicable.

## Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Erasing Part of a Temporary LOB](#) on page 11-189
- [\(OCI\): Erasing Part of a Temporary LOB](#) on page 11-190
- [COBOL \(Pro\\*COBOL\): Erasing Part of a Temporary LOB](#) on page 11-193
- [C/C++ \(Pro\\*C/C++\): Erasing Part of a Temporary LOB](#) on page 11-195
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## PL/SQL (DBMS\_LOB Package): Erasing Part of a Temporary LOB

```

/* Erasing part of a temporary LOB. [Example script: 3918.sql]
   Procedure eraseTempLOB_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE trimTempLOB_proc IS
  Lob_loc          CLOB;
  amt              number;
  Src_loc          BFILE := BFILENAME('ADPHOTO_DIR', 'monitor_photo_3060_11001');
  Amount           INTEGER := 32767;
BEGIN

  /* Create a temporary LOB: */
  DBMS_LOB.CREATETEMPORARY(Lob_loc,TRUE);

  /* 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;

```

## (OCI): Erasing Part of a Temporary LOB

```

/* Erasing part of a temporary LOB. [Example script: 3919.c]
   This example erases 2 bytes at offset 100 in a temporary LOB: */

sb4 erase_temp_lob ( 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 monitor_photo_3060_11001 file: */
    if(OCILobFileSetName(envhp, errhp, &bfile,
                          (text *)"ADPHOTO_DIR",
                          (ub2)strlen("ADPHOTO_DIR"),
                          (text *)"monitor_photo_3060_11001",
                          (ub2)strlen("monitor_photo_3060_11001")))
    {
        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;
}

```

## COBOL (Pro\*COBOL): Erasing Part of a Temporary LOB

\* Erasing part of a temporary LOB. [Example script: 3920.pco]

```

IDENTIFICATION DIVISION.
PROGRAM-ID. TEMP-BLOB-ERASE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) 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  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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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.

## C/C++ (Pro\*C/C++): Erasing Part of a Temporary LOB

```

/* Erasing part of a temporary LOB. [Example script: 3921.pc] */

#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 = "ADPHOTO_DIR", *Name = "monitor_photo_3060_11001";
    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;

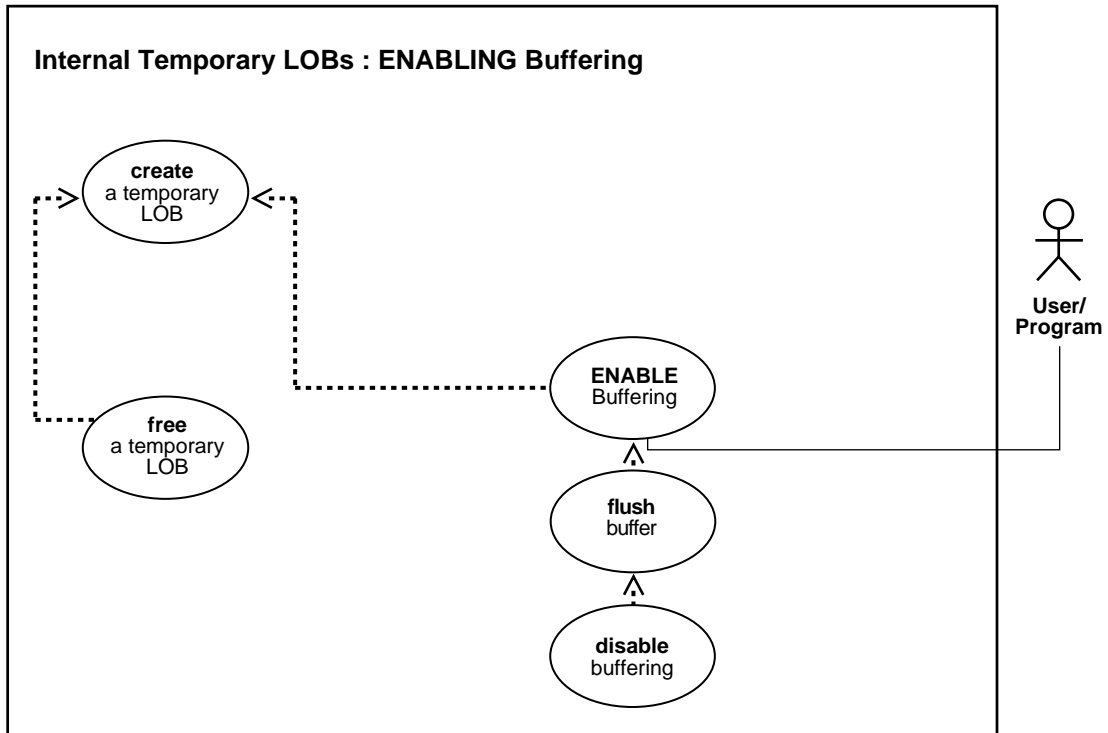
/* Release resources held by the Locators: */
EXEC SQL FREE :Lob_loc;
EXEC SQL FREE :Temp_loc;
}

void main()
{
    char *samp = "samp/samp";
    EXEC SQL CONNECT :samp;
    eraseTempLOB_proc();
    EXEC SQL ROLLBACK WORK RELEASE;
}
```



## Enabling LOB Buffering for a Temporary LOB

Figure 11–25 Use Case Diagram: Enabling LOB Buffering for a Temporary LOB



**See Also:** [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to enable LOB buffering for a temporary LOB.

### Usage Notes

Enable buffering when performing 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.

---

---

**Note:** Do not enable buffering to perform the stream read and write involved in checkin and checkout.

---

---

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCIEnableLobBuffering, OCIDisableLobBuffering, OCILobFlushBuffer
- COBOL (Pro\*COBOL): *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB ENABLE BUFFERING.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB ENABLE BUFFERING
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), ["Enabling LOB Buffering"](#) on page 10-251.

## Scenario

Not applicable.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- C (OCI): [Enabling LOB Buffering for a Temporary LOB](#) on page 11-198
- COBOL (Pro\*COBOL): [Enabling LOB Buffering for a Temporary LOB](#) on page 11-200
- C/C++ (Pro\*C/C++): [Enabling LOB Buffering for a Temporary LOB](#) on page 11-202
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Enabling LOB Buffering for a Temporary LOB

```

/* Enabling LOB buffering for a temporary LOB. [Example script: 3922.c] */

#define MAXBUFLLEN 32767
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *tblob;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    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;
}
```

## COBOL (Pro\*COBOL): Enabling LOB Buffering for a Temporary LOB

```
* Enabling LOB buffering for a temporary LOB. [Example script: 3923.pco]
IDENTIFICATION DIVISION.
```

```
PROGRAM-ID. TEMP-LOB-BUFFERING.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
  
01 USERID PIC X(11) VALUES "SAMP/SAMP".  
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 '2525252626262525' 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 SQLEERROR 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.
```

## C/C++ (Pro\*C/C++): Enabling LOB Buffering for a Temporary LOB

```
/* Enabling LOB buffering for a temporary LOB. [Example script: 3924.pc] */

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

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 SQLEERROR DO Sample_Error();
```

```
/* Allocate and Create the Temporary LOB: */
EXEC SQL ALLOCATE :Temp_loc;
EXEC SQL LOB CREATE TEMPORARY :Temp_loc;

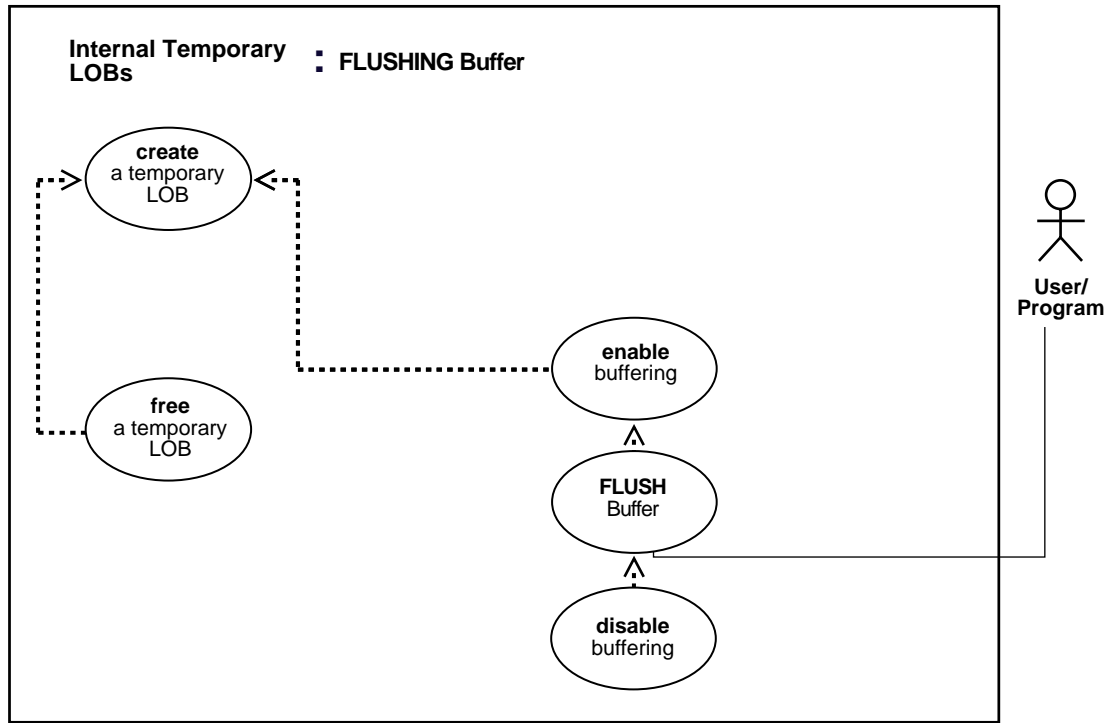
/* Enable use of the LOB Buffering Subsystem: */
EXEC SQL LOB ENABLE BUFFERING :Temp_loc;
memset((void *)Buffer.arr, 42, BufferLength);
Buffer.len = BufferLength;
for (multiple = 0; multiple < 8; multiple++)
{
    /* Write Data to the Temporary LOB: */
    EXEC SQL LOB WRITE ONE :Amount
        FROM :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;
}
```

## Flushing Buffer for a Temporary LOB

Figure 11–26 Use Case Diagram: Flushing Buffer for a Temporary LOB



**See Also:** [Table 11–1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2.

### Purpose

This procedure describes how to flush the buffer for a temporary LOB.

### Usage Notes

Not applicable.



## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): There is no applicable syntax reference for this use case.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobFlushBuffer
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB FLUSH BUFFER.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB FLUSH BUFFER
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See [Chapter 10, "Internal Persistent LOBs"](#), "Flushing the Buffer" on page 10-257.

## Scenario

Not applicable.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example is provided with this release.
- C (OCI): [Flushing Buffer for a Temporary LOB](#) on page 11-100
- COBOL (Pro\*COBOL): [Determining If a Pattern Exists in a Temporary LOB \(instr\)](#) on page 11-101
- C/C++ (Pro\*C/C++): [Determining If a Pattern Exists in a Temporary LOB \(instr\)](#) on page 11-104
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Flushing Buffer for a Temporary LOB

```
/* Flushing the temporary LOB buffer. [Example script: 3926] */
```

```
sb4 lobBuffering (envhp, errhp, svchp, stmthp)
OCIEnv    *envhp;
OCIError  *errhp;
OCISvcCtx *svchp;
OCIStmt   *stmthp;
{
    OCILobLocator *tblob;
    ub4 amt;
    ub4 offset;
    sword retval;
    ub1 bufp[MAXBUFLLEN];
    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;
}

```

## COBOL (Pro\*COBOL): Flushing Buffer for a Temporary LOB

```

* Flushing a temporary LOB buffer. [Example script: 3927.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. FREE-TEMPORARY.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".

01  TEMP-BLOB SQL-BLOB.

```

```
01 IS-TEMP          PIC S9(9) COMP.
01 ORASLNDR        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 ORASLNDR TO ORASLNDR.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNDR, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

## C/C++ (Pro\*C/C++): Flushing Buffer for a Temporary LOB

```

/* Flushing a temporary LOB buffer. [Example script: 3928.pc] */

#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;
}

```

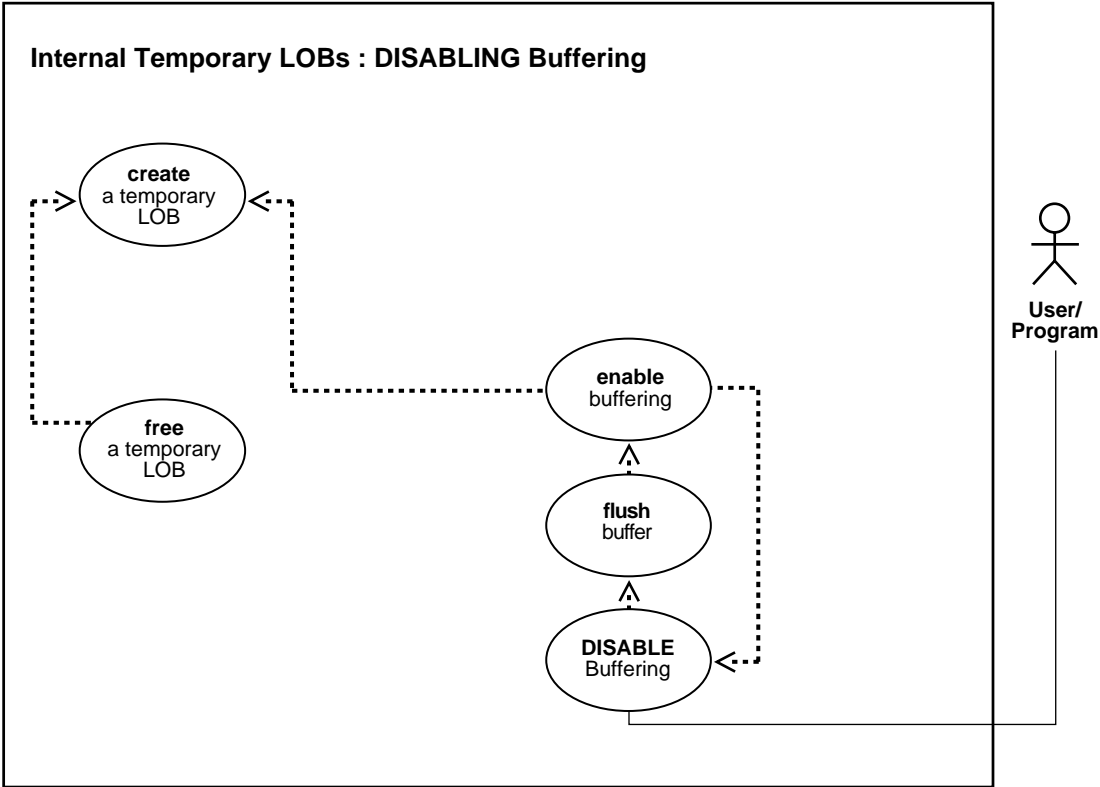
```
EXEC SQL LOB DESCRIBE :Temp_loc GET LENGTH INTO :Length;
printf("Wrote %d characters using the Buffering Subsystem\n", Length);

/* 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;
}
```

# Disabling LOB Buffering for a Temporary LOB

Figure 11-27 Use Case Diagram: Disabling LOB Buffering



See Also: [Table 11-1, "Use Case Model Overview: Internal Temporary LOBs"](#) on page 11-2

**Purpose**

This procedure describes how to disable temporary LOB buffering.

## Usage Notes

You enable buffering when performing 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.

---

---

**Note:** Do not enable buffering to perform the stream read and write involved in checkin and checkout.

---

---

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): A syntax reference is not applicable in this release.
- C (OCI): *Oracle Call Interface Programmer's Guide* Chapter 15, "Relational Functions" — LOB Functions, OCILobDisableBuffering
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB DISABLE BUFFERING.
- C/C++ (Pro\*C/C++): *Pro\*C/C++ Precompiler Programmer's Guide* Appendix F, "Embedded SQL Statements and Directives" — LOB DISABLE BUFFERING
- Visual Basic (OO4O): There is no applicable syntax reference for this use case.
- Java (JDBC): See the section, [Chapter 10, "Internal Persistent LOBs"](#), "C (OCI): Disabling LOB Buffering".

## Scenario

Not applicable.

## Examples

Examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB Package): No example provided with this release.
- [C \(OCI\): Disabling LOB Buffering](#) on page 11-212
- [COBOL \(Pro\\*COBOL\): Disabling LOB Buffering for a Temporary LOB](#) on page 11-215



- [C/C++ \(Pro\\*C/C++\): Disabling LOB Buffering for a Temporary LOB on page 11-216](#)
- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Disabling LOB Buffering

```

/* Disabling LOB buffering for temporary LOBs. [Example script: 3929.c] */

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;
}
```

## COBOL (Pro\*COBOL): Disabling LOB Buffering for a Temporary LOB

\* Disabling LOB buffering for a temporary LOB. [Example script: 3930.pco]

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

01 USERID    PIC X(11) VALUES "SAMP/SAMP".
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 '2525252626262525' 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 SQLEERROR 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.
```

## C/C++ (Pro\*C/C++): Disabling LOB Buffering for a Temporary LOB

```
/* Disabling LOB buffering for a temporary LOB. [Example script: 3931.pc] */

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

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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;
}
```

---

## External LOBs (BFILEs)

### Use Case Model

In this chapter we discuss each operation on External LOBs (such as "Reading Data from a BFILE") in terms of a use case. Table 12-1, "Use Case Model: External LOBs (BFILEs)" lists all the use cases.

### Individual Use Cases

Each External LOB (BFILE) use case is described as follows:

- *Use case figure.* A figure that depicts the use case. See [Appendix A, "Unified Modeling Language Diagrams"](#) for help in understanding these UML based diagrams.
- *Purpose.* The purpose of this use case with regards to LOBs.
- *Usage Notes.* Guidelines to assist your implementation of the LOB operation.
- *Syntax.* Pointers to the syntax in different programmatic environments that underlies the LOBs related activity for the use case.
- *Scenario.* Describes the implementation of the use case in terms of the sample schema used in the examples. For further details on sample schemas, refer to *Oracle9i Sample Schemas*.
- *Examples.* How to apply each use case based on the sample schema used.

## Use Case Model: External LOBs (BFILES)

Table 12–1, "Use Case Model: External LOBs (BFILES)" indicates with + where examples are provided for specific use cases and in which programmatic environment (see Chapter 3, "LOB Support in Different Programmatic Environments" for a complete discussion and references to related manuals).

Programmatic environment abbreviations used in the following table, are as follows:

- **P** — PL/SQL using the DBMS\_LOB Package
- **O** — C using OCI (Oracle Call Interface)
- **B** — COBOL using Pro\*COBOL precompiler
- **C** — C/C++ using Pro\*C/C++ precompiler
- **V** — Visual Basic using OO4O (Oracle Objects for OLE)
- **J** — Java using JDBC (Java Database Connectivity)
- **S** — SQL

**Table 12–1 Use Case Model: External LOBs (BFILES)**

| Use Case and Page  | P | O | B | C | V | J |
|--|---|---|---|---|---|---|
| <a href="#">Creating a Table Containing One or More BFILE Columns</a> on page 12-13      | S | S | S | S | S | S |
| <a href="#">Creating a Table of an Object Type with a BFILE Attribute</a> on page 12-17  | S | S | S | S | S | S |
| <a href="#">Creating a Table with a Nested Table Containing a BFILE</a> on page 12-20    | S | S | S | S | S | S |
| <a href="#">INSERT a Row Using BFILENAME()</a> on page 12-22                             | S | + | + | + | + | + |
| <a href="#">INSERT a BFILE Row by Selecting a BFILE From Another Table</a> on page 12-31 | S | S | S | S | S | S |
| <a href="#">Inserting a Row With BFILE by Initializing a BFILE Locator</a> on page 12-33 | + | + | + | + | + | + |
| <a href="#">Loading Data Into External LOB (BFILE)</a> on page 12-42                     | S | S | S | S | S | S |
| <a href="#">Loading a LOB with BFILE Data</a> on page 12-46                              | + | + | + | + | + | + |
| <a href="#">Loading a BLOB with BFILE Data</a> on page 12-55                             | + | - | - | - | - | - |
| <a href="#">Loading a CLOB with BFILE Data</a> on page 12-59                             | + | - | - | - | - | - |
| <a href="#">Ways to Open a BFILE</a> on page 12-63                                       | - | - | - | - | - | - |
| <a href="#">Opening a BFILE with FILEOPEN</a> on page 12-65                              | + | + | - | - | - | + |



**Table 12–1 Use Case Model: External LOBs (BFILES) (Cont.)**

| <b>Use Case and Page</b>  | <b>P</b> | <b>O</b> | <b>B</b> | <b>C</b> | <b>V</b> | <b>J</b> |
|---|----------|----------|----------|----------|----------|----------|
| Opening a BFILE with OPEN on page 12-70                                       | +        | +        | +        | +        | +        | +        |
| Ways to See If a BFILE is Open on page 12-78                                  | -        | -        | -        | -        | -        | -        |
| Checking If the BFILE is Open with FILEISOPEN on page 12-80                   | +        | +        | -        | -        | -        | +        |
| Checking If a BFILE is Open Using ISOPEN on page 12-86                        | +        | +        | +        | +        | +        | +        |
| Displaying BFILE Data on page 12-96n  | +        | +        | +        | +        | +        | +        |
| Reading Data from a BFILE on page 12-108n                                     | +        | +        | +        | +        | +        | +        |
| Reading a Portion of BFILE Data (substr) on page 12-119                       | +        | -        | +        | +        | +        | +        |
| Comparing All or Parts of Two BFILES on page 12-127                           | +        | -        | +        | +        | +        | +        |
| Checking If a Pattern Exists (instr) in the BFILE on page 12-137              | +        | -        | +        | +        | -        | +        |
| Checking If the BFILE Exists on page 12-145                                   | +        | +        | +        | +        | +        | +        |
| Getting the Length of a BFILE on page 12-154                                  | +        | +        | +        | +        | +        | +        |
| Copying a LOB Locator for a BFILE on page 12-164                              | +        | +        | +        | +        | -        | +        |
| Determining If a LOB Locator for a BFILE Is Initialized on page 12-172        | -        | +        | -        | +        | -        | -        |
| Determining If One LOB Locator for a BFILE Is Equal to Another on page 12-177 | -        | +        | -        | +        | -        | +        |
| Getting DIRECTORY Alias and Filename on page 12-184n                          | +        | +        | +        | +        | +        | +        |
| Updating a BFILE Using BFILENAME() on page 12-193                             | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |
| Updating a BFILE by Selecting a BFILE From Another Table on page 12-196       | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |
| Updating a BFILE by Initializing a BFILE Locator on page 12-198               | +        | +        | +        | +        | +        | +        |
| Closing a BFILE with FILECLOSE on page 12-208n                                | +        | +        | -        | -        | +        | +        |
| Closing a BFILE with CLOSE on page 12-214                                     | +        | +        | +        | +        | +        | +        |
| Closing All Open BFILES with FILECLOSEALL on page 12-222                      | +        | +        | +        | +        | +        | +        |
| Deleting the Row of a Table Containing a BFILE on page 12-231                 | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> | <b>S</b> |

## Accessing External LOBs (BFILES)

To access external LOBs (BFILES) use one of the following interfaces:

- Precompilers, such as Pro\*C/C++ and Pro\*COBOL
- OCI (Oracle Call Interface)
- PL/SQL (DBMS\_LOB package)
- Java (JDBC)
- Oracle Objects for OLE (OO4O)

**See Also:** [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for information about the six interfaces used to access external LOBs (BFILES) and their available functions.

## Directory Object

The DIRECTORY object facilitates administering access and usage of BFILES in an Oracle Server (see CREATE DIRECTORY in *Oracle9i SQL Reference*). A DIRECTORY specifies a *logical alias name* for a physical directory on the server's file system under which the file to be accessed is located. You can access a file in the server's file system only if granted the required access privilege on DIRECTORY object.

## Initializing a BFILE Locator

DIRECTORY object also provides the flexibility to manage the locations of the files, instead of forcing you to hardcode the absolute path names of physical files in your applications. A DIRECTORY alias is used in conjunction with the BFILENAME() function, in SQL and PL/SQL, or the OCILobFileNameSetName(), in OCI for initializing a BFILE locator.

---

---

**Note:** Oracle does not verify that the directory and path name you specify actually exist. You should take care to specify a valid directory in your operating system. If your operating system uses case-sensitive path names, be sure you specify the directory in the correct format. There is no need to specify a terminating slash (for example, `/tmp/` is not necessary, simply use `/tmp`).

---

---

## How to Associate Operating System Files with Database Records

To associate an operating system (OS) file to a BFILE, first create a DIRECTORY object which is an alias for the full path name to the operating system file.

To associate existing operating system files with relevant database records of a particular table use Oracle SQL DML (Data Manipulation Language). For example:

- Use INSERT to initialize a BFILE column to point to an existing file in the server's file system
- Use UPDATE to change the reference target of the BFILE
- Initialize a BFILE to NULL and then update it later to refer to an operating system file using the BFILENAME() function.
- OCI users can also use OCILobFileName() to initialize a BFILE locator variable that is then used in the VALUES clause of an INSERT statement.

### Examples

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.gif and image2.gif 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 following UPDATE statement 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() and Initialization

`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 following `UPDATE` statements 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;
```

Think of `BFILENAME()` in terms of initialization — it can initialize the value for the following:

- `BFILE` column
- `BFILE` (automatic) variable declared inside a PL/SQL module

**Advantages.** This has the following advantages:

- 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.
- 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 using a `SELECT`, you save a round-trip to the server.

For more information, refer to the example given for `DBMS_LOB.LOADFROMFILE` (see "[Loading a LOB with BFILE Data](#)" on page 12-46).

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 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')
```

### On Windows Platforms

On Windows NT, for example, the directory names are case-insensitive. Therefore the following two statements refer to the same directory:

```
CREATE DIRECTORY "big_cap_dir" AS "g:\data\source";
```

```
CREATE DIRECTORY "small_cap_dir" AS "G:\DATA\SOURCE";
```

## BFILE Security

This section introduces the `BFILE` security model and associated SQL statements. The main SQL statements associated with `BFILE` security are:

- SQL DDL: `CREATE` and `REPLACE` or `ALTER` a `DIRECTORY` object
- SQL DML: `GRANT` and `REVOKE` the `READ` system and object privileges on `DIRECTORY` objects

## Ownership and Privileges

The `DIRECTORY` object is a *system owned* object. For more information on system owned objects, see *Oracle9i SQL Reference*. Oracle9i supports two new system privileges, which are granted only to DBA:

- `CREATE ANY DIRECTORY` — for creating or altering the directory object creation
- `DROP ANY DIRECTORY` — for deleting the directory object

## Read Permission on Directory Object

`READ` permission 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` permission with `GRANT` option, you may in turn grant this privilege to other users/roles and add them to your privilege domains.

---

---

**Note:** The `READ` permission is defined only on the `DIRECTORY object`, not on individual files. Hence there is no way to assign different privileges to files in the same directory.

---

---

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 the following:

- That the physical directory exists
- *Read* permission for the Oracle Server process is enabled on the file, the directory, and the path leading to it
- The directory remains available, and *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 `CREATE ANY DIRECTORY` and `DROP ANY DIRECTORY` privileges potentially expose the server file system to all database users, the DBA should be prudent in granting these privileges to normal database users to prevent security breach.

---

---

## SQL DDL for BFILE Security

Refer to the *Oracle9i SQL Reference* for information about the following SQL DDL statements that create, replace, and drop directory objects:

- CREATE DIRECTORY
- DROP DIRECTORY

## SQL DML for BFILE Security

Refer to the *Oracle9i SQL Reference* for information about the following SQL DML statements 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 corresponding paths and privileges. Supported views are:

- ALL\_DIRECTORIES (OWNER, DIRECTORY\_NAME, DIRECTORY\_PATH)  
This view describes all directories accessible to the user.
- DBA\_DIRECTORIES(OWNER, DIRECTORY\_NAME, DIRECTORY\_PATH)  
This view describes all 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 file system. But to realize this goal, it is very important that the DBA follow these guidelines when using DIRECTORY objects:

- *Do Not Map DIRECTORY to Directories of Data Files, And So On.* A DIRECTORY should not be mapped to physical directories that contain Oracle data files, control files, log files, and other system files. Tampering with these files (accidental or otherwise) could corrupt the database or the server operating system.

- *Only the DBA Should Have System Privileges.* 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.
- *Use Caution When Granting DIRECTORY Object Privilege.* 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.
- *Do not Drop or Replace DIRECTORY Objects When Database is in Operation.* DIRECTORY objects should not be arbitrarily dropped or replaced when the database is in operation. If this were to happen, 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 shut down.  
  
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.
- *Caution When Revoking User's Privilege on DIRECTORY Objects.* Revoking a user's privilege on a DIRECTORY object 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 Shared Server (Multi-Threaded Server — MTS) Mode**

Oracle9i does not support session migration for BFILES in Shared Server (Multi-Threaded Server — MTS) mode. This implies that operations on open BFILES can persist beyond the end of a call to a shared server.



In shared server sessions, BFILE operations will be bound to one shared server, they cannot migrate from one server to another. This restriction will be removed in a forthcoming release.

## External LOB (BFILE) Locators

For BFILES, the value is stored in a server-side operating system file; in other words, external to the database. The BFILE locator that refers to that file is stored in the row.

**When Two Rows in a BFILE Table Refer to the Same File** 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.

**BFILE Locator Variable** A BFILE locator variable 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.

### Guidelines

- **Open and Close a File From Same Program Block at Same Nesting Level.** 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.
- **Set the BFILE Value Before Flushing Object to Database.** 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()`.
- **Indicate Directory Alias and Filename Before INSERT or UPDATE of BFILE.** It is an error to `INSERT` or `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.

- Initialize BFILE Before INSERT or UPDATE

---

---

**Note:** `OCISetAttr()` does not allow the user to set a BFILE locator to `NULL`.

---

---

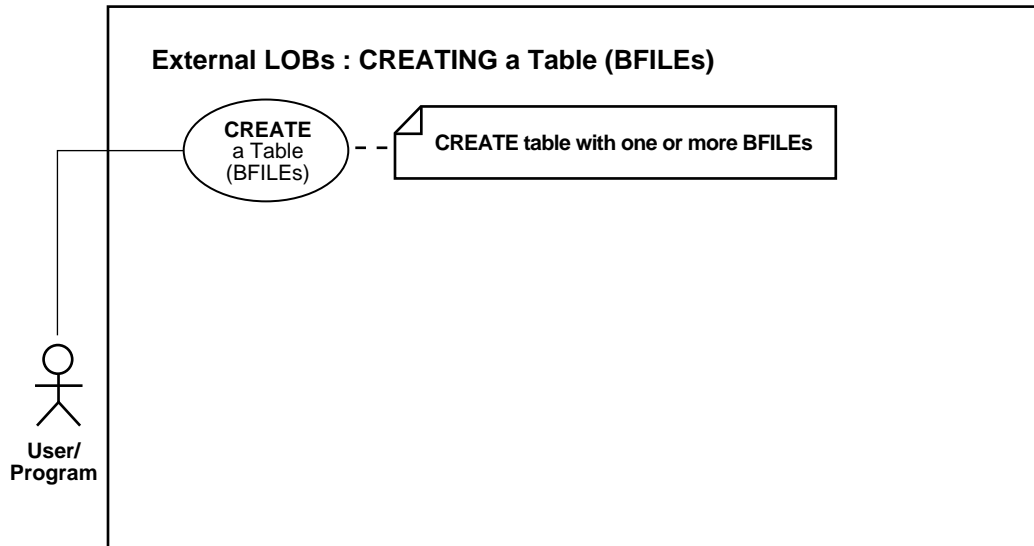
### General Rule

Before using SQL to insert or update a row with a BFILE, the user must initialize the BFILE to one of the following :

- `NULL` (not possible if using an OCI bind variable)
- A directory alias and filename

## Creating a Table Containing One or More BFILE Columns

Figure 12–1 Use Case Diagram: Creating a Table Containing One or More BFILE Columns



### See Also:

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Creating a Table of an Object Type with a BFILE Attribute](#) on page 12-17
- [Creating a Table with a Nested Table Containing a BFILE](#) on page 12-20

### Purpose

This procedure describes how to create a table containing one or more BFILE columns.

## Usage Notes

SQL Data Definition Language (DDL) is used to define `BFILE` columns in a table and `BFILE` attributes in an object type.

## Syntax

Use the following syntax references:

- [SQL \(Oracle9i SQL Reference\), Chapter 7, "SQL Statements" — CREATE TABLE](#)

## Scenario

The heart of our hypothetical application is the table `Print_media`. The varied types that make up the columns of this table make it possible to collect together the many different kinds of elements used in printed media.

## Examples

The following example is provided in SQL and applies to all programmatic environments:

- [SQL: Creating a Table Containing One or More BFILE Columns](#) on page 12-14

## SQL: Creating a Table Containing One or More BFILE Columns

You may need to set up the following data structures for certain examples in this chapter to work:

```
/* Setup script for creating Print_media, Online_media and associated
structures */

Rem The HR and OE Schema need to be created before you create the PM Schema
Rem For a detailed listing of the pm_drop.sql and pm_main.sql scripts see
Rem the manual, Oracle9i Sample Schemas.

DROP USER pm CASCADE;
DROP DIRECTORY ADPHOTO_DIR;
DROP DIRECTORY ADCOMPOSITE_DIR;
DROP DIRECTORY ADGRAPHIC_DIR;
DROP INDEX onlinemedia CASCADE CONSTRAINTS;
DROP INDEX printmedia CASCADE CONSTRAINTS;
DROP TABLE online_media CASCADE CONSTRAINTS;
DROP TABLE print_media CASCADE CONSTRAINTS;
DROP TYPE textdoc_typ;
DROP TYPE textdoc_tab;
DROP TYPE adheader_typ;
```

```

DROP TABLE adheader_typ;
CREATE USER pm;
GRANT CONNECT, RESOURCE to pm;

CREATE DIRECTORY ADPHOTO_DIR AS '/tmp/';
CREATE DIRECTORY ADCOMPOSITE_DIR AS '/tmp/';
CREATE DIRECTORY ADGRAPHIC_DIR AS '/tmp/';
CREATE DIRECTORY media_dir AS '/tmp/';
GRANT READ ON DIRECTORY ADPHOTO_DIR to pm;
GRANT READ ON DIRECTORY ADCOMPOSITE_DIR to pm;
GRANT READ ON DIRECTORY ADGRAPHIC_DIR to pm;
GRANT READ ON DIRECTORY media_dir to pm;

CONNECT pm/pm (or &pass);
COMMIT;

CREATE TABLE a_table (blob_col BLOB);

CREATE TYPE adheader_typ AS OBJECT (
    header_name    VARCHAR2(256),
    creation_date  DATE,
    header_text    VARCHAR(1024),
    logo           BLOB );

CREATE TYPE textdoc_typ AS OBJECT (
    document_typ   VARCHAR2(32),
    formatted_doc  BLOB);

CREATE TYPE Textdoc_ntab AS TABLE of textdoc_typ;

CREATE TABLE Adheader_tab of adheader_typ (
    header_name    VARCHAR2(256) CONSTRAINT hname CHECK (hname IS NOT NULL),
    creation_date  DATE DEFAULT NULL,
    logo           DEFAULT EMPTY_BLOB()
);

CREATE TABLE online_media
( product_id NUMBER(6),
  product_photo ORDSYS.ORDImage,
  product_photo_signature ORDSYS.ORDImageSignature,
  product_thumbnail ORDSYS.ORDImage,
  product_video ORDSYS.ORDVideo,
  product_audio ORDSYS.ORDAudio,
  product_text CLOB,
  product_testimonials ORDSYS.ORDDoc);

```

```
CREATE UNIQUE INDEX onlinemedia_pk
  ON online_media (product_id);

ALTER TABLE online_media
ADD (CONSTRAINT onlinemedia_pk
PRIMARY KEY (product_id), CONSTRAINT loc_c_id_fk
FOREIGN KEY (product_id) REFERENCES oe.product_information(product_id)
);

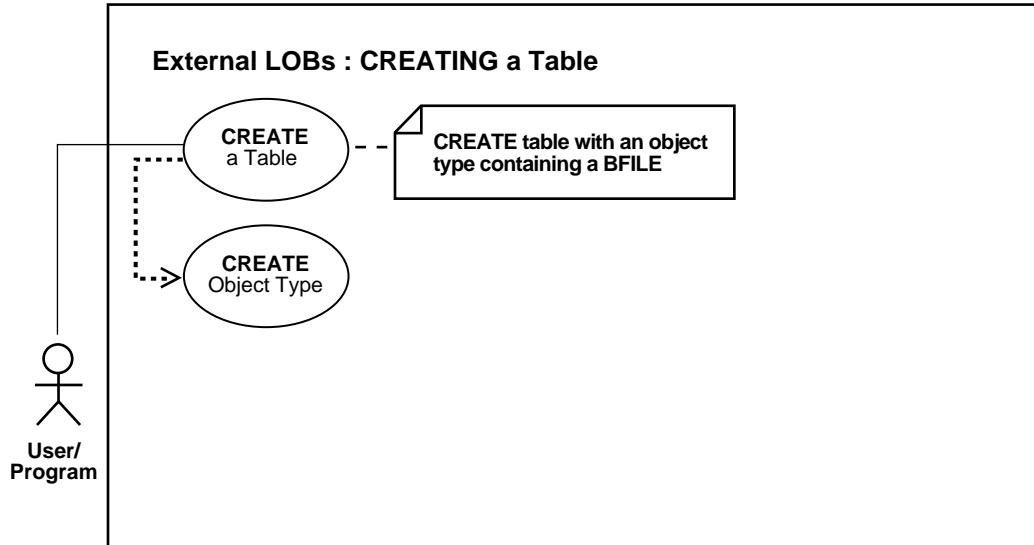
CREATE TABLE print_media
(product_id NUMBER(6),
ad_id NUMBER(6),
ad_composite BLOB,
ad_sourcetext CLOB,
ad_finaltext CLOB,
ad_fktextn NCLOB,
ad_testdocs_ntab textdoc_tab,
ad_photo BLOB,
ad_graphic BFILE,
ad_header adheader_typ,
press_release LONG) NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;

CREATE UNIQUE INDEX printmedia_pk
  ON print_media (product_id, ad_id);

ALTER TABLE print_media
ADD (CONSTRAINT printmedia_pk
PRIMARY KEY (product_id, ad_id),
CONSTRAINT printmedia_fk FOREIGN KEY (product_id)
REFERENCES oe.product_information(product_id)
);
```

## Creating a Table of an Object Type with a BFILE Attribute

Figure 12–2 Use Case Diagram: Creating a Table Containing a BFILE



### See Also:

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Creating a Table Containing One or More BFILE Columns](#) on page 12-13
- [Creating a Table with a Nested Table Containing a BFILE](#) on page 12-20

### Purpose

This procedure describes how to create a table of an object type with a BFILE attribute.

## Usage Notes

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. SQL Data Definition Language (DDL) is used to define `BFILE` columns in a table and `BFILE` attributes in an object type.

## Syntax

Use the following syntax references:

- *SQL (Oracle9i SQL Reference)*, Chapter 7, "SQL Statements" — CREATE TABLE, CREATE TYPE

Note that `NCLOBs` cannot be attributes of an object type.

## Scenario

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

- `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.
- `Multimedia_tab` contains 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.

## Examples

The following example is provided in SQL and applies to all programmatic environments:

- [SQL: Creating a Table of an Object Type with a BFILE Attribute](#) on page 12-18

## SQL: Creating a Table of an Object Type with a BFILE Attribute

```
/* 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
```



```
);

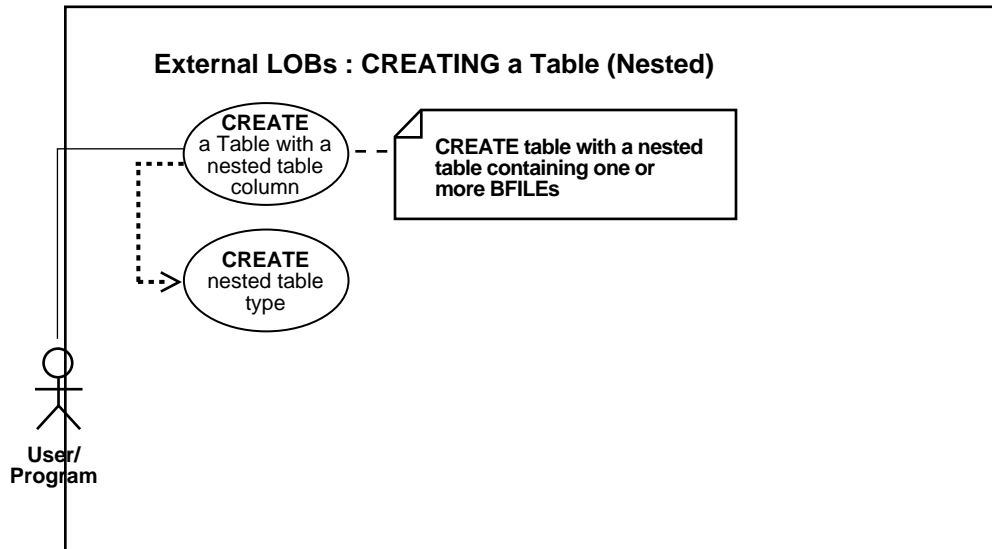
/* Create table Voiceover_tab Using SQL DDL: */
CREATE TABLE Voiceover_tab OF Voiced_typ
(   Script DEFAULT EMPTY_CLOB(),
    CONSTRAINT Take CHECK (Take IS NOT NULL),
    Recording DEFAULT NULL
);

/* 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;
```

## Creating a Table with a Nested Table Containing a BFILE

Figure 12–3 Use Case Diagram: Creating a Table with a Nested Table Containing a BFILE



### See Also:

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Creating a Table Containing One or More BFILE Columns](#) on page 12-13
- [Creating a Table of an Object Type with a BFILE Attribute](#) on page 12-17

### Purpose

This procedure describes how to create a table with nested table containing a BFILE.

### Usage Notes

As shown in the use case diagram, you must create the object type that contains BFILE attributes before you create a nested table that uses that object type. SQL

Data Definition Language (DDL) is used to define BFILE columns in a table and BFILE attributes in an object type.

### Syntax

Use the following syntax references:

- *SQL (Oracle9i SQL Reference)*, Chapter 7, "SQL Statements" — CREATE TABLE, CREATE TYPE

### Scenario

In our example, `Print_media` table contains a nested table `ad_textdoc_ntab` that includes type `textdoc_typ`. This type makes use of two LOB datatypes — a BFILE for graphic images of products, and a BLOB for formatted documents.

We have already described how to create a table with BFILE columns (see "[Creating a Table Containing One or More BFILE Columns](#)" on page 12-13), so here we only describe the SQL syntax for creating the underlying object type.

### Examples

The following example is provided in SQL and applies to all programmatic environments:

- [SQL: Creating a Table with a Nested Table Containing a BFILE](#) on page 12-21

## SQL: Creating a Table with a Nested Table Containing a BFILE

Because you use SQL DDL directly to create a table, the `DBMS_LOB` package is not relevant.

```
CREATE TYPE textdoc_typ AS OBJECT (
    document_typ  VARCHAR2(32),
    formatted_doc BLOB);

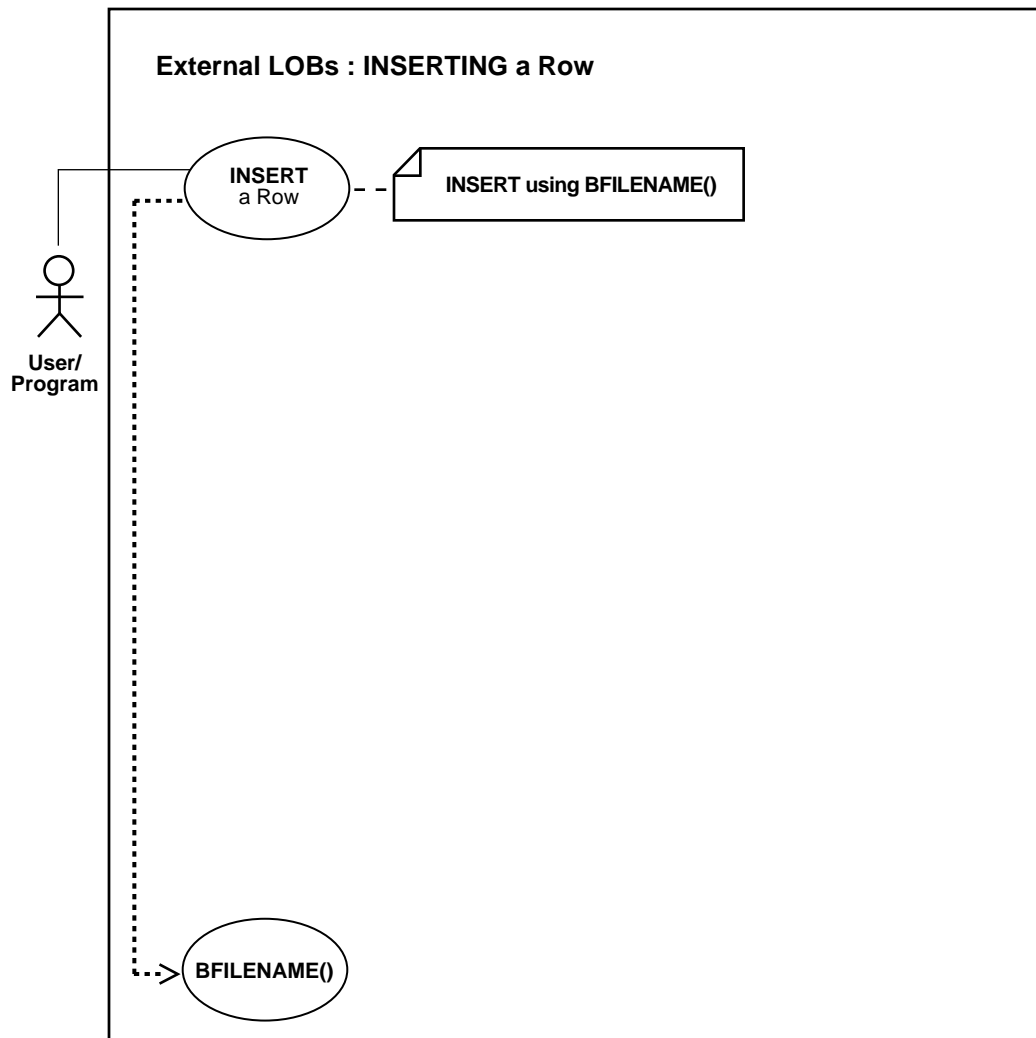
CREATE TYPE Textdoc_ntab AS TABLE OF textdoc_typ;

/* Embedding the nested table is accomplished when the structure
   of the containing table is defined. Using the PM sample schema,
   this is done by adding the following clause to the end of the CREATE
   Print_media statement: */

NESTED TABLE ad_textdocs_ntab STORE AS textdocs_nestedtab;
```

## INSERT a Row Using BFILENAME()

Figure 12-4 Use Case Diagram: INSERT a Row Using BFILENAME()



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [INSERT a BFILE Row by Selecting a BFILE From Another Table](#) on page 12-31
- [Inserting a Row With BFILE by Initializing a BFILE Locator](#) on page 12-33

**Purpose**

This procedure describes how to insert a row using BFILENAME().

**Usage Notes**

Call BFILENAME() function as part of an INSERT to initialize a BFILE column or attribute for a particular row, by associating it with a physical file in the server's file system.

Although DIRECTORY object, represented by the directory\_alias parameter to BFILENAME(), need not already be defined *before* BFILENAME() is called by a SQL or PL/SQL program, the *DIRECTORY object and operating system file must exist* by the time you actually use the BFILE locator. For example, when used as a parameter to one of the following operations:

- OCILobFileOpen()
- DBMS\_LOB.FILEOPEN()
- OCILobOpen()
- DBMS\_LOB.OPEN()

---

---

**Note:** 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 BFILENAME().

---

---

---

---

**Note:** Before you insert, you must initialize the BFILE either to NULL or to a directory alias and filename.

---

---

## Ways BFILENAME() is Used to Initialize BFILE Column or Locator Variable

You can use BFILENAME() in the following ways to initialize a BFILE column:

- As part of an SQL INSERT statement
- As part of an UPDATE statement

You can use BFILENAME( ) to initialize a BFILE locator variable in one of the programmatic interface programs, and use that locator for file operations. However, if the corresponding directory alias or filename does not exist then PL/SQL DBMS\_LOB or other relevant 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 12-7

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. See the following syntax references for each programmatic environment:

- *SQL Oracle9i SQL Reference*, Chapter 7, "SQL Statements" — INSERT
- *C (OCI) Oracle Call Interface Programmer's Guide*: Chapter 7, "LOB and File Operations", for usage notes. Chapter 15, "Relational Functions" — LOB Functions
- *COBOL (Pro\*COBOL) Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, embedded SQL, and precompiler directives. See also *Oracle9i SQL Reference*, for related information on the SQL INSERT statement.
- *C/C++ (Pro\*C/C++) Pro\*C/C++ Precompiler Programmer's Guide*: "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives". See *Oracle9i SQL Reference, Chapter 7, "SQL Statements"* — INSERT
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraDynaset > METHODS > AddNew
- *Java (JDBC) Oracle9i JDBC Developer's Guide and Reference*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in *Oracle9i SQLJ Developer's Guide and Reference*: Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples illustrate how to insert a row using BFILENAME().

## Examples

Examples are provided in the following six programmatic environments:

- [SQL: Inserting a Row by means of BFILENAME\(\)](#) on page 12-25
- [C \(OCI\): Inserting a Row by means of BFILENAME\(\)](#) on page 12-25
- [COBOL \(Pro\\*COBOL\): Inserting a Row by means of BFILENAME\(\)](#) on page 12-26
- [C/C++ \(Pro\\*C/C++\): Inserting a Row by means of BFILENAME\(\)](#) [C/C++ \(Pro\\*C/C++\): Inserting a Row by means of BFILENAME\(\)](#) on page 12-27
- [Visual Basic \(OO4O\): Inserting a Row by means of BFILENAME\(\)](#) [Visual Basic \(OO4O\): Inserting a Row by means of BFILENAME\(\)](#) on page 12-28
- [Java \(JDBC\): Inserting a Row by means of BFILENAME\(\)](#) on page 12-29

## SQL: Inserting a Row by means of BFILENAME()

```
/* Inserting a row using BFILENAME(). [Example script: 3945.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 Print_media VALUES (3106, 13001, EMPTY_BLOB(),
    EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(), NULL,
    EMPTY_BLOB(), BFILENAME('AD_GRAPHIC_DIR', '3106_keyboard'),
    NULL, "Your press release text goes here");
```

## C (OCI): Inserting a Row by means of BFILENAME()

```
/* Inserting a row using BFILENAME. [Example script: 3946.c] */
```

```
void insertUsingBfilename(svchp, stmhpp, errhp)
OCIStmt *svchp;
OCIStmt *stmhpp;
OCIError *errhp;
{
    text *insstmt =
        (text *) "INSERT INTO Print_media VALUES (3060, 11001, EMPTY_BLOB(), \\"
```

## INSERT a Row Using BFILENAME()

---

```
EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(), \
(SELECT REF(ad) FROM Textdoc_ntab ad WHERE document_typ = 'PDF'), \
EMPTY_BLOB(), BFILENAME ('ADGRAPHIC_DIR','monitor_3060_11001'), \
(SELECT REF(adhead) FROM Adheader_typ Adhead \
WHERE creation_date = '1-20-2001'), \
"PRESS RELEASE \
Date of Press Release: January 11, 2001 \
Contact Information: Shelley and Co., Oracle Corporation, 500 Oracle Parkway, \
Redwood City, CA 94065 \
Disclaimer: This product, product name, and information is fictitious and has \
been composed to illustrate the functionality of Oracle products. \
Any similarity to existing products or product names is coincidental. \
TIGER2 3060 Monitor ..... an Exceptional Visual Experience! \
Oracle announces its return to manufacturing hardware and computer peripherals! \
The first model to have completed rigorous usability and stress tests is the \
TIGER2 +3060 17-Inch CRT MONITOR with its cousin the TIGER2 3060a 17-inch \
Flatscreen." \
)";

/* 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));
}
```

## COBOL (Pro\*COBOL): Inserting a Row by means of BFILENAME()

```
* Inserting a row using BFILENAME() [Example script: 3947.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-INSERT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 USERID          PIC X(11) VALUES "SAMP/SAMP".
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 PRINT_MEDIA (PRODUCT_ID, AD_GRAPHIC)
    VALUES (1, BFILENAME('ADGRAPHIC_DIR', 'KEYBOARD_310_13001'))
END-EXEC.
EXEC SQL
    ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Inserting a Row by means of BFILENAME()

```

/* Inserting a row using BFILENAME(). [Example script: 3948.pc] */

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

void Sample_Error()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;

```

## INSERT a Row Using BFILENAME()

---

```
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 Print_media WHERE product_id = 2056 AND ad_id = 12001;

/* Insert a new row using the BFILENAME() function for BFILES: */
EXEC SQL INSERT INTO Print_media
VALUES (2056, 12001, EMPTY_BLOB(), EMPTY_CLOB(), EMPTY_CLOB(), EMPTY_CLOB(),
ad_textdocs(textdoc_typ(PDF, EMPTY_BLOB())),
EMPTY_BLOB(),
BFILENAME('ADGRAPHIC_DIR', 'mousepad_2056_12001'),
NULL,
'You Can't Beat this Mousepad for Ergonomic Value!!')
;
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;
}
```

## Visual Basic (OO4O): Inserting a Row by means of BFILENAME()

'Inserting a row using BFILENAME(). [Example script: 3949.txt]

```
Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile

Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
OraDyn.AddNew
OraDyn.Fields("product_id").value = 3060
OraDyn.Fields("ad_sourcetext").value = Empty 'This is equivalent to EMPTY_CLOB()
```

```
in SQL
OraDyn.Fields("fltextn").value = Empty
'Initialize BFile Data:
OraAdGraphic.DirectoryName = "ADGRAPHIC_DIR"
OraAdGraphic.FileName = "monitor_graphic_3060_11001"
OraDyn.Fields("ad_composite").Value = Empty
OraDyn.Fields("ad_photo").Value = Empty
OraDyn.Update
'Add the row to the table
```

## Java (JDBC): Inserting a Row by means of BFILENAME()

```
// Inserting a row using BFILENAME(). [Example script: 3951.java]

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:
        DriverManager.registerDriver(new 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
        {
```

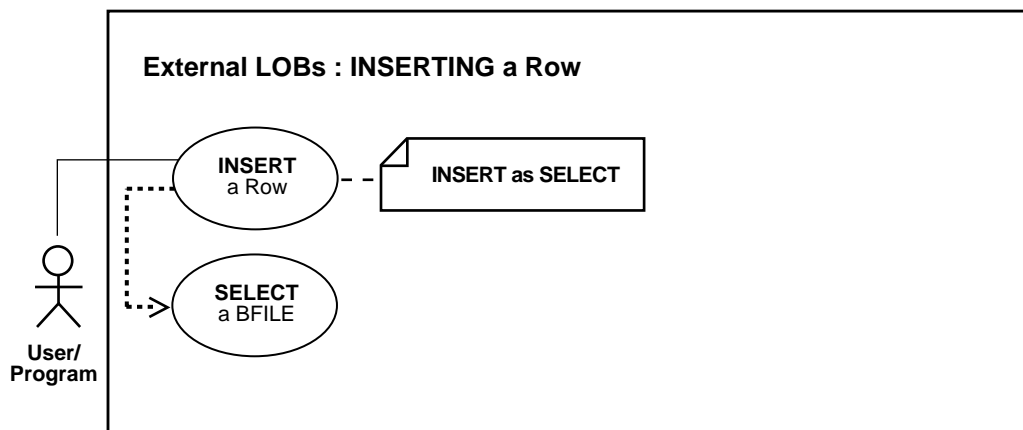
## INSERT a Row Using BFILENAME()

---

```
stmt.execute("INSERT INTO Print_media "
+"VALUES (3060, 11001, EMPTY_BLOB(), EMPTY_CLOB(), "
+" EMPTY_CLOB(), EMPTY_CLOB()),"
+"(SELECT REF(ad) FROM Textdoc_ntab ad"
+" WHERE document_typ = 'PDF'),"
+"EMPTY_BLOB(), BFILENAME ('AD_GRAPHIC','monitor_3060'), "
+"(SELECT REF(adhead) FROM Adheader_typ Adhead"
+" WHERE creation_date = '1-20-2001'), "
+"PRESS RELEASE \")
+"Date of Press Release: January 11, 2001 \")
+"Contact Information: Any name,Oracle Corporation, 500 Oracle Parkway,"
+"Redwood City, CA 94065 \")
+"Disclaimer: This product, product name, and information is fictitious and has"
been" +"composed to illustrate the functionality of Oracle products. \")
+"Any similarity to existing products or product names is coincidental. \")
+"TIGER2 3060 Monitor ..... an Exceptional Visual Experience! \")
+"Oracle announces its return to manufacturing hardware and computer
peripherals! \")
+"The first model to have completed rigorous usability and stress tests is the"
+"TIGER2 +3060 17-Inch CRT MONITOR with its cousin the TIGER2 3060a 17-inch"
+"Flatscreen. \")
+"Its initial offering is for $150 and its suggested retail value is $299. \")
);
// Commit the transaction:
conn.commit();
stmt.close();
conn.close();
}
catch (SQLException e)
{
e.printStackTrace();
}
}
```

## INSERT a BFILE Row by Selecting a BFILE From Another Table

Figure 12–5 Use Case Diagram: INSERT a Row Containing a BFILE by Selecting a BFILE From Another Table (INSERT ... AS ... SELECT)



### See Also:

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [INSERT a Row Using BFILENAME\(\)](#) on page 12-22
- [Inserting a Row With BFILE by Initializing a BFILE Locator](#) on page 12-33

### Purpose

This procedure describes how to INSERT a row containing a BFILE by selecting a BFILE from another table.

### Usage Notes

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. See the following "Scenario".

---

---

**Note:** Before you insert, you must initialize the BFILE either to NULL or to a directory alias and filename.

---

---

### Syntax

See the following syntax reference:

- [SQL \(Oracle9i SQL Reference\): Chapter 7, "SQL Statements" — INSERT](#)

### Scenario

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 column `Voiced_ref` of `Multimedia_tab` table.

It inserts values from the library table into `Multimedia_tab` by means of a `SELECT`.

### Examples

The example is provided in SQL and applies to all programmatic environments:

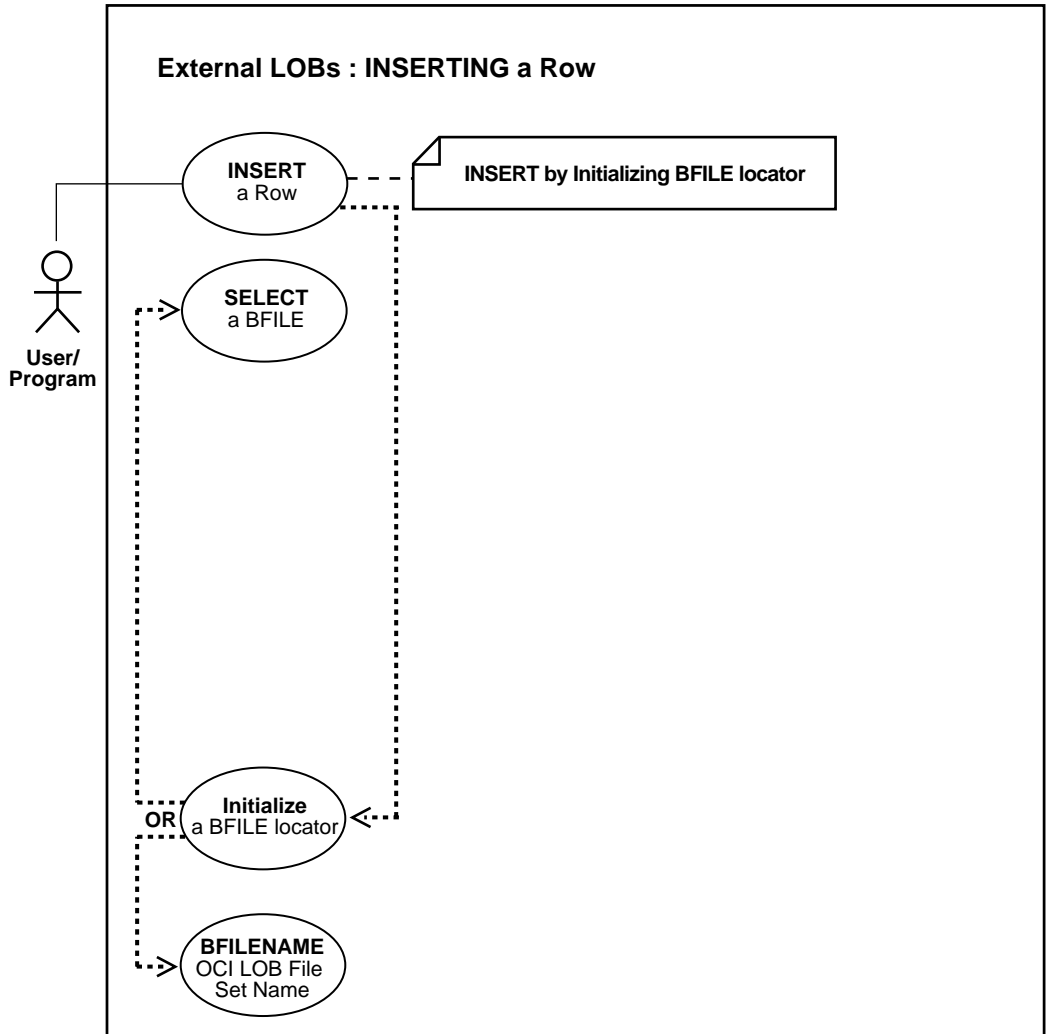
- [SQL: Inserting a Row Containing a BFILE by Selecting a BFILE From Another Table](#) on page 12-32

## SQL: Inserting a Row Containing a BFILE by Selecting a BFILE From Another Table

```
INSERT INTO Voiceover_tab
  (SELECT * from VoiceoverLib_tab
   WHERE Take = 12345);
```

# Inserting a Row With BFILE by Initializing a BFILE Locator

Figure 12-6 Use Case Diagram: Inserting a Row by Initializing a BFILE Locator



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [INSERT a Row Using BFILENAME\(\)](#) on page 12-22
- [INSERT a BFILE Row by Selecting a BFILE From Another Table](#) on page 12-31

**Purpose**

This procedure describes how to INSERT a row containing a BFILE by initializing a BFILE locator.

**Usage Notes**

---

---

**Note:** You must initialize the BFILE locator bind variable to a directory alias and filename before issuing the insert statement.

---

---

---

---

**Note:** Before you insert, you must initialize the BFILE either to NULL or to a directory alias and filename.

---

---

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. See the following syntax references for each programmatic environment:

- SQL (*Oracle9i SQL Reference*, Chapter 7 "SQL Statements" — INSERT
- C (OCI) *Oracle Call Interface Programmer's Guide*: Chapter 7, "LOB and File Operations". Chapter 15, "Relational Functions" — LOB Functions.
- COBOL (Pro\*COBOL) *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, embedded SQL, and precompiler directives. See also *Oracle9i SQL Reference*, for related information on the SQL INSERT statement.
- C/C++ (Pro\*C/C++) *Pro\*C/C++ Precompiler Programmer's Guide*: "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB FILE SET. See also (*Oracle9i SQL Reference*), Chapter 7 "SQL Statements" — INSERT



- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBfile > METHODS > DirectoryName, FileName; and > OBJECTS > OraDynaset > METHODS > Update
- Java (JDBC) *Oracle9i JDBC Developer's Guide and Reference*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in *Oracle9i SQLJ Developer's Guide and Reference*: Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

In these examples we insert an `ad_graphic` from an operating system source file (`ADGRAPHIC_DIR`).

### Examples

Examples in the following programmatic environments are provided:

- [PL/SQL: Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-35
- [C \(OCI\): Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-36
- [COBOL \(Pro\\*COBOL\): Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-37
- [C/C++ \(Pro\\*C/C++\): Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-38
- [Visual Basic \(OO4O\): Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-39
- [Java \(JDBC\): Inserting a Row Containing a BFILE by Initializing a BFILE Locator](#) on page 12-39

## PL/SQL: Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```

/* Inserting row containing a BFILE by initializing a BFILE locator
   [Example script: 3953.sql] */

DECLARE
  /* Initialize the BFILE locator: */
  Lob_loc BFILE := BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001');
BEGIN

```

```

INSERT INTO Print_media
(product_id, ad_id, ad_graphic) VALUES (3106, 13001, Lob_loc);
COMMIT;
END;

```

## C (OCI): Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```

/* Inserting a row by initializing a BFILE Locator. [Example script: 3954.c] */

void insertUsingBfileLocator(envhp, svchp, stmthp, errhp)
OCIEnv *envhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
OCIError *errhp;
{
    text *insstmt =
        (text *) "INSERT INTO Print_media (product_id, ad_graphic) \
        VALUES (2056, :Lob_loc)";
    OCIBind *bndhp;
    OCILobLocator *Lob_loc;
    OraText *Dir = (OraText *)"ADGRAPHIC_DIR", *Name = (OraText *)"mousepad_2056_
12001";

    /* 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);
}

```

## COBOL (Pro\*COBOL): Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```

* Inserting a row containing a BFILE by initializing a BFILE
* [Example script: 3955.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-INSERT-INIT.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) 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  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 "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "keyboard_graphic_3106_13001" 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 PRINT_MEDIA (PRODUCT_ID, AD_GRAPHIC)
    VALUES (3106, :SRC-BFILE)END-EXEC.
EXEC SQL ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```
/* Inserting a row containing a BFILE by initializing a BFILE */
/* [Example script: 3958.pc] */
#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 = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";
    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 Print_media (Product_ID, ad_graphic) VALUES (2056, :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;
}
}

```

## Visual Basic (OO4O): Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```

' Inserting a row containing a BFILE by initializing a BFILE.
' [Example script: 3959.txt]

```

```

Dim OraDyn as OraDynaset, OraPhoto as OraBFile, OraMusic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraMusic = OraDyn.Fields("ad_graphic").Value

```

```

'Edit the first row and initiliaze the "ad_graphic" column:
OraDyn.Edit
OraPhoto.DirectoryName = "ADGRAPHIC_DIR"
OraPhoto.FileName = "mousepad_graphic_2056_12001"
OraDyn.Update

```

## Java (JDBC): Inserting a Row Containing a BFILE by Initializing a BFILE Locator

```

// Inserting a row containing a BFILE by initializing a BFILE.
// [Example script: 3960.java]

// 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:
        DriverManager.registerDriver(new 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('ADGRAPHIC_DIR','monitor_graphic_3060_11001') FROM
DUAL");
            if (rset.next())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);
            }

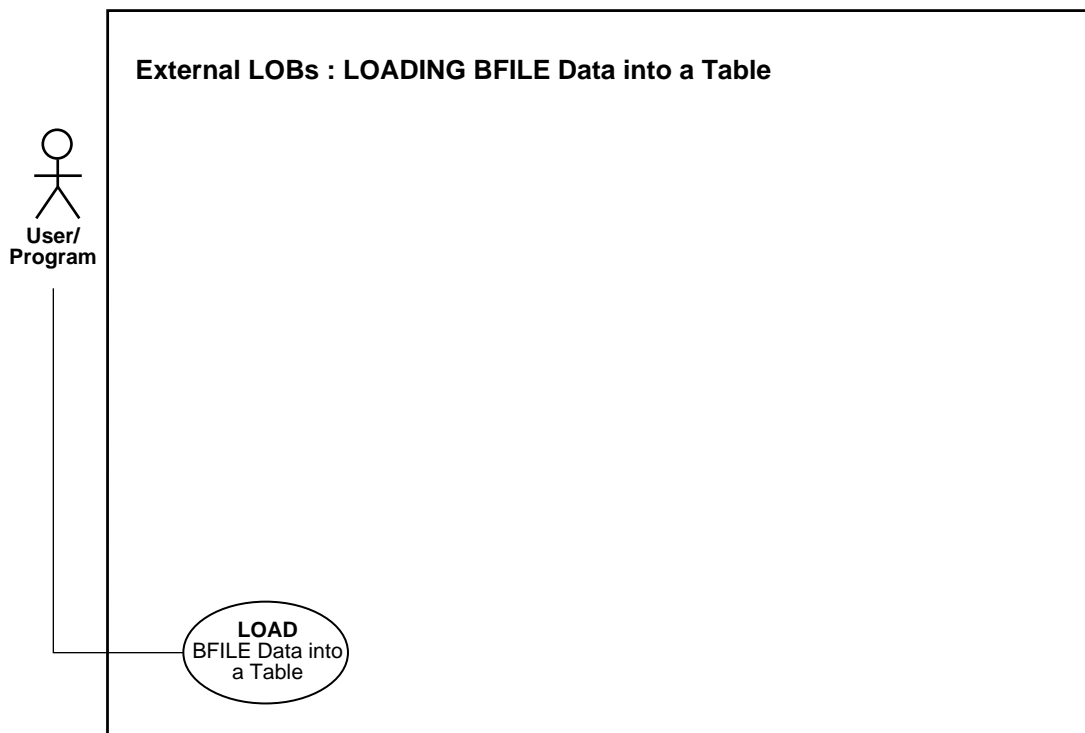
            // Prepare a CallableStatement to OPEN the LOB for READWRITE:
            cstmt = (OracleCallableStatement) conn.prepareCall (
                "INSERT INTO Print_media (product_id, ad_graphic) VALUES (3060, ?)");
            cstmt.setBFILE(1, src_lob);
            cstmt.execute();

            //Close the statements and commit the transaction:
            stmt.close();
            cstmt.close();
            conn.commit();
        }
    }
}
```

```
conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Loading Data Into External LOB (BFILE)

Figure 12–7 Use Case Diagram: Loading Initial Data into External LOB (BFILE)



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#)  
on page 12-2

### Purpose

This procedure describes how to load initial data into a BFILE and the BFILE data into a table.

### Usage Notes

The `BFILE` datatype stores unstructured *binary data* in operating-system files outside 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 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 file system) have already been created.

**See Also:** *Oracle9i Application Developer's Guide - Fundamentals* for more information on BFILES.

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.

## Syntax

Use the following syntax references:

- SQL\*Loader (*Oracle9i Database Utilities*)
- [Chapter 4, "Managing LOBs", Using SQL\\*Loader to Load LOBs](#)

## Scenario

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 adgraphic_photo as '/tmp';
CREATE OR REPLACE DIRECTORY adgraphic_dir as '/tmp';
```

---

---

## Examples

The following examples load data into BFILES:

- [Loading Data Into BFILES: File Name Only is Specified Dynamically](#)
- [Loading Data into BFILES: File Name and DIRECTORY Object Dynamically Specified](#)

## Loading Data Into BFILES: File Name Only is Specified Dynamically

### Control File

```
LOAD DATA
INFILE sample9.dat
INTO TABLE Print_media
FIELDS TERMINATED BY ','
(product_id INTEGER EXTERNAL(6),
FileName FILLER CHAR(30),
ad_graphic BFILE(CONSTANT "modem_graphic_2268_21001", FileName))
```

### Data file (sample9.dat)

```
007, modem_2268.jpg,
008, monitor_3060.jpg,
009, keyboard_2056.jpg,
```

---

---

**Note:** product\_ID defaults to (255) if a size is not specified. It is mapped to the file names in the datafile. ADGRAPHIC\_PHOTO is the directory where all files are stored. ADGRAPHIC\_DIR is a DIRECTORY object created previously.

---

---

## Loading Data into BFILES: File Name and DIRECTORY Object Dynamically Specified

### Control File

```
LOAD DATA
INFILE sample10.dat
INTO TABLE Print_media
FIELDS TERMINATED BY ','
```

```
(  
  product_id INTEGER EXTERNAL(6),  
  ad_graphic BFILE (DirName, FileName),  
  FileName FILLER CHAR(30),  
  DirName FILLER CHAR(30)  
)
```

### Data file (sample10.dat)

```
007,monitor_3060.jpg,ADGRAPHIC_PHOTO,  
008,modem_2268.jpg,ADGRAPHIC_PHOTO,  
009,keyboard_2056.jpg,ADGRAPHIC_DIR,
```

---

---

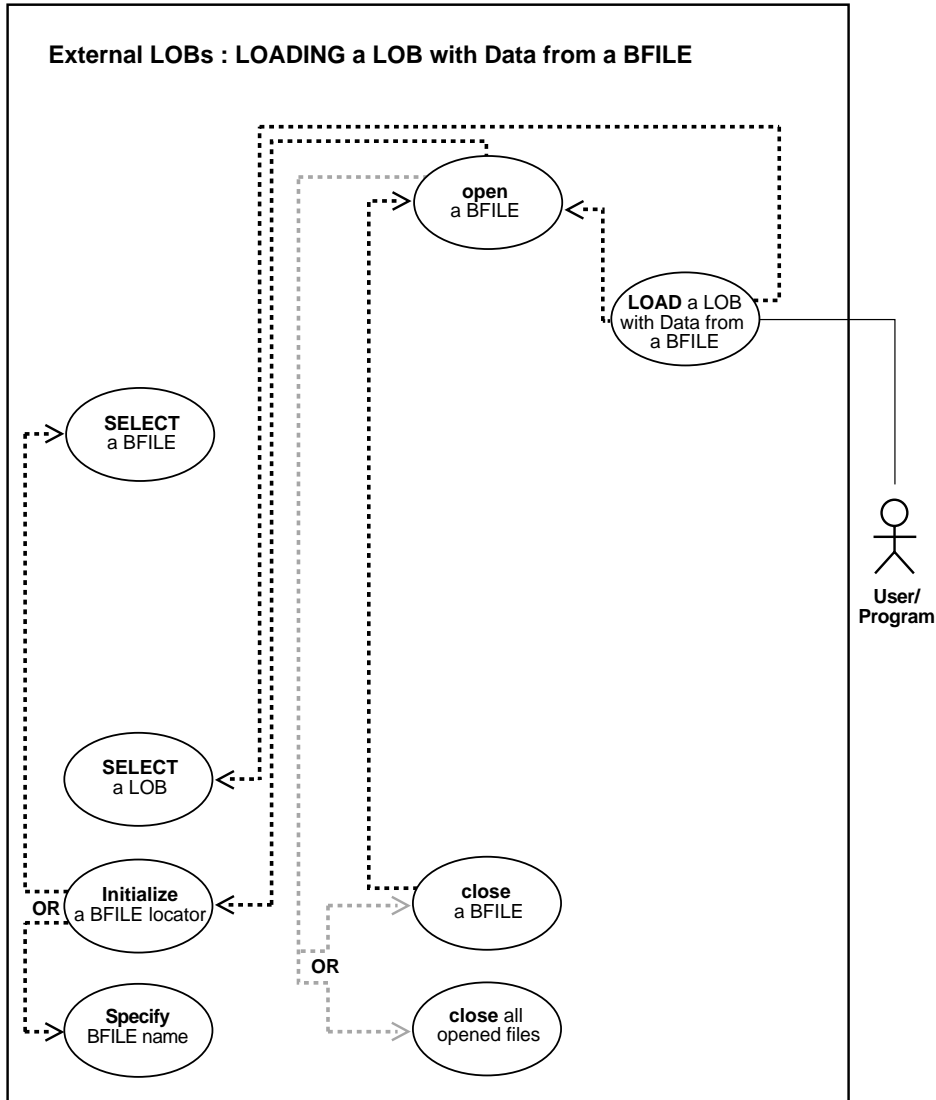
**Note:** DirName FILLER CHAR ( 30 ) is mapped to the datafile field containing the directory name corresponding to the file being loaded.

---

---

## Loading a LOB with BFILE Data

Figure 12–8 Use Case Diagram: Loading a LOB with BFILE Data



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Loading a BLOB with BFILE Data](#) on page 12-55
- [Loading a CLOB with BFILE Data](#) on page 12-59

**Purpose**

This procedure describes how to load a LOB with BFILE data.

**Usage Notes**

---

---

**Note:** The `LOADBLOBFROMFILE` and `LOADCLOBFROMFILE` procedures implement the functionality of this procedure and provide improved features for loading binary data and character data. The improved procedures are available in the PL/SQL environment only. When possible, using one of the improved procedures is recommended. See ["Loading a BLOB with BFILE Data"](#) and ["Loading a CLOB with BFILE Data"](#) for more information.

---

---

**Character Set Conversion** In using 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.

**BFILE to CLOB or NCLOB: Converting From Binary Data to a Character Set** When you use the `DBMS_LOB.LOADFROMFILE` procedure to populate a CLOB or NCLOB, you are populating the LOB with binary data from the BFILE. *No implicit translation* is performed from binary data to a character set. For this reason, you should use the `LOADCLOBFROMFILE` procedure when loading text (see [Loading a CLOB with BFILE Data](#) on page 12-59).

**See Also:** *Oracle9i Database Globalization Support Guide* for character set conversion issues.

**Specify Amount Parameter to be Less than the Size of the BFILE!**

- `DBMS_LOB.LOADFROMFILE`: You cannot specify the amount parameter to be larger than the size of the BFILE.

- **OCILobLoadFromFile:** You cannot specify the `amount` parameter to be larger than the length of the BFILE.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB)** (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — LOADFROMFILE
- **C (OCI)** (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations", for usage notes and examples. Chapter 15, "Relational Functions" — LOB Functions, OCILobLoadFromFile.
- **COBOL (Pro\*COBOL)** (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, embedded SQL, and LOB LOAD precompiler directives.
- **C/C++ (Pro\*C/C++)** (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements" "Embedded SQL Statements and Directives"— LOB LOAD.
- **Visual Basic (OO4O)** (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBLOB.OraCLOB > METHODS > CopyFromBfile
- **Java (JDBC)** (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These example procedures assume there is a directory object (`ADGRAPHIC_DIR`) that contains the LOB data to be loaded into the destination LOB.

### Examples

Examples are provided in the following programmatic environments:

- **PL/SQL (DBMS\_LOB Package):** [Loading a LOB with BFILE Data](#) on page 12-49
- **C (OCI):** [Loading a LOB with BFILE Data](#) on page 12-49

- [COBOL \(Pro\\*COBOL\): Loading a LOB with BFILE Data](#) on page 12-51
- [C/C++ \(Pro\\*C/C++\): Loading a LOB with BFILE Data](#) on page 12-53
- [Visual Basic \(OO4O\): Loading a LOB with BFILE Data](#) on page 12-54

## PL/SQL (DBMS\_LOB Package): Loading a LOB with BFILE Data

```

/* Loading a LOB with BFILE data.
   Procedure loadLOBFromBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE loadLOBFromBFILE_proc IS
  Dest_loc      BLOB;
  Src_loc       BFILE := BFILENAME('ADGRAPHIC_DIR',
                                   'keyboard_graphic_3106_13001');
  Amount        INTEGER := 4000;
BEGIN
  SELECT ad_graphic INTO Dest_loc FROM Print_media
     WHERE product_id = 3060 AND ad_id = 13001 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;

```

## C (OCI): Loading a LOB with BFILE Data

```

/* Loading a LOB with BFILE data.
   Select the lob/bfile from the Print_media table */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
  char selstmt[150];
  OCIDefine *dfnhp, *dfnhp2;

```

```

strcpy(selstmt, (char *) "SELECT ad_photo FROM Print_media \
                          WHERE product_id=3106 AND ad_id = 13001 FOR UPDATE");

/* Prepare the SQL select statement */
checkerr (errhp, OCISstmtPrepare(stmthp, errhp, selstmt,
                                (ub4) strlen((char *) selstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
/* Define the column being selected */
checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                (dvoid *)&Lob_loc, 0, SOLT_BLOB,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT)
        || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                                (dvoid *)&Lob_loc, 0, SOLT_BLOB,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT));
/* Execute the SQL select statement */
checkerr (errhp, OCISstmtExecute(svchp, stmthp, errhp, (ub4) 1, (ub4) 0,
                                (CONST OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

void loadLobFromBfile(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCISstmt *stmthp;
{
    OCILobLocator *dest_loc;
    OCILobLocator *src_loc;

    /* Allocate locators */
    (void) OCIDescriptorAlloc((dvoid *) envhp,
                              (dvoid **) &dest_loc, (ub4)OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp,
                              (dvoid **) &src_loc, (ub4)OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    checkerr(errhp, OCILobFileSetName(envhp, errhp, &src_loc,
                                      (text *) "ADPHOTO_DIR", (ub2) strlen("ADPHOTO_DIR"),
                                      (text *) "keyboard_photo_3106_13001",
                                      (ub2) strlen(keyboard_photo_3106_13001)));
}

```



```

selectLob(dest_loc, errhp, svchp, stmthp);

checkerr(errhp, OCILobFileOpen(svchp, errhp, src_loc,
                               (ub1)OCI_FILE_READONLY));
checkerr(errhp, OCILobOpen(svchp, errhp, dest_loc, (ub1)OCI_LOB_READWRITE));
checkerr (errhp, OCILobLoadFromFile(svchp, errhp, dest_loc, src_loc,
                                   (ub4)4000, (ub4)1, (ub4)1));
checkerr(errhp, OCILobClose(svchp, errhp, dest_loc));
checkerr(errhp, OCILobFileClose(svchp, errhp, src_loc));
}

```

## COBOL (Pro\*COBOL): Loading a LOB with BFILE Data

```

* Loading a LOB with BFILE data.
IDENTIFICATION DIVISION.
PROGRAM-ID. LOAD-BFILE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
      MOVE 9 TO DIR-ALIAS-LEN.

```

```
MOVE "keyboard_photo_3106_13001" 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 AD_GRAPHIC INTO :SRC-BFILE
    FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
    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 ROLLBACK 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.
```

## C/C++ (Pro\*C/C++): Loading a LOB with BFILE Data

```

/* Loading a LOB with BFILE data. */

#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 = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";
    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 ad_photo INTO :Dest_loc FROM Print_media
        WHERE Product_ID = 2056 AND AD_ID = 12001 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;
}
```

## Visual Basic (OO4O): Loading a LOB with BFILE Data

```
'Loading a LOB with BFILE data
```

```
Dim OraDyn as OraDynaset, OraDyn2 as OraDynaset, OraAdGraphic as OraBFile
Dim OraAdPhoto as OraBlob
```

```
chunksize = 32767
```

```
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
```

```
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
```

```
Set OraAdPhoto = OraDyn.Fields("ad_photo").Value
```

```
OraDyn.Edit
```

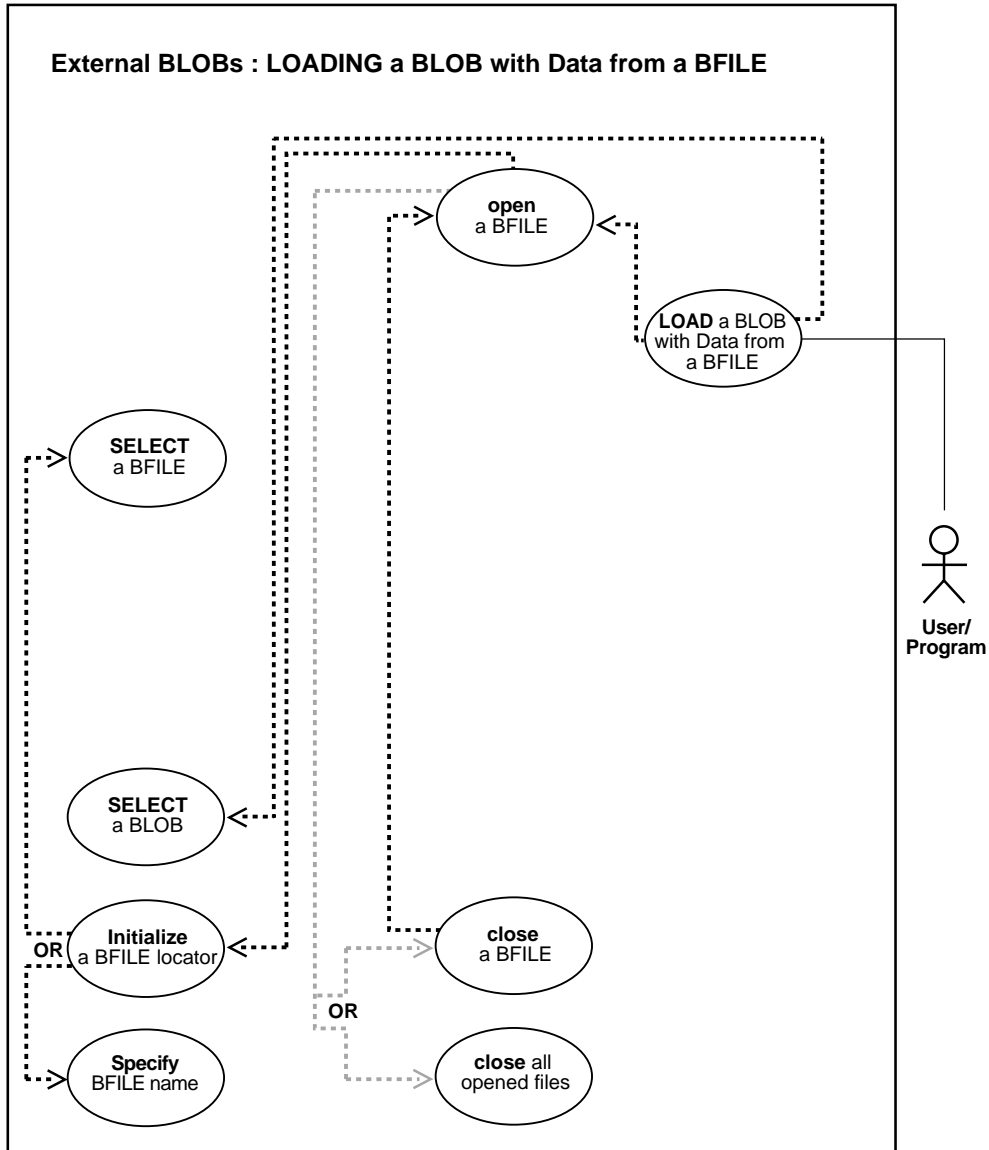
```
'Load LOB with data from BFILE:
```

```
OraAdPhoto.CopyFromBFile (OraAdGraphic)
```

```
OraDyn.Update
```

## Loading a BLOB with BFILE Data

Figure 12–9 Use Case Diagram: Loading a BLOB with Binary Data



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Loading a LOB with BFILE Data](#) on page 12-46
- [Loading a CLOB with BFILE Data](#) on page 12-59

**Purpose**

This procedure describes how to load a BLOB with binary data from a BFILE. This achieves the same outcome as `LOADFROMFILE` as well as returning the new offsets to the user.

**Usage Notes**

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for text loading. Note that since this functionality does not support BFILE on the client side, the loading takes place on the server side only. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADBLOBFROMFILE`

**Scenario**

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the binary LOB data to be loaded into the target BLOB.

**Examples**

The example, ["PL/SQL \(DBMS\\_LOB Package\): Loading an Internal Persistent BLOB with BFILE Data"](#), illustrates `LOADBLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

## PL/SQL (DBMS\_LOB Package): Loading an Internal Persistent BLOB with BFILE Data

The following example illustrates:

- How to load an internal persistent BLOB with binary data from a BFILE.
- How to use LOADBLOBFROMFILE to load the entire file without getting its length first.
- How to use the return value of the offsets to calculate the actual amount loaded.

```

DECLARE
    src_loc      BFILE := bfilename('ADVERT_DIR','display_ad_frame') ;
    dst_loc      BLOB;
    src_offset   NUMBER := 1;
    dst_offset   NUMBER := 1;
    src_osin     NUMBER;
    dst_osin     NUMBER;
    bytes_rd     NUMBER;
    bytes_wt     NUMBER;
BEGIN
    SELECT ad_composite INTO dst_loc FROM Print_media
           WHERE product_id=3106 and ad_id=13001 FOR UPDATE;

    /* Opening the source BFILE is mandatory */
    dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

    /* Opening the LOB is optional */
    dbms_lob.OPEN(dst_loc, dbms_lob.lob_readwrite);
    /* Save the input source/destination offsets */
    src_osin := src_offset;
    dst_osin := dst_offset;
    /* Use LOBMAXSIZE to indicate loading the entire BFILE */
    dbms_lob.LOADBLOBFROMFILE(dst_loc,src_loc,dbms_lob.lobmaxsize,src_offset,dst_
offset) ;

    /* Closing the LOB is mandatory if you have opened it */
    dbms_lob.close(dst_loc);
    dbms_lob.filecloseall();
    COMMIT;

    /* Use the src_offset returned to calculate the actual amount read from the
BFILE */
    bytes_rd := src_offset - src_osin;
    dbms_output.put_line(' Number of bytes read from the BFILE ' || bytes_rd ) ;
    /* Use the dst_offset returned to calculate the actual amount written to the

```

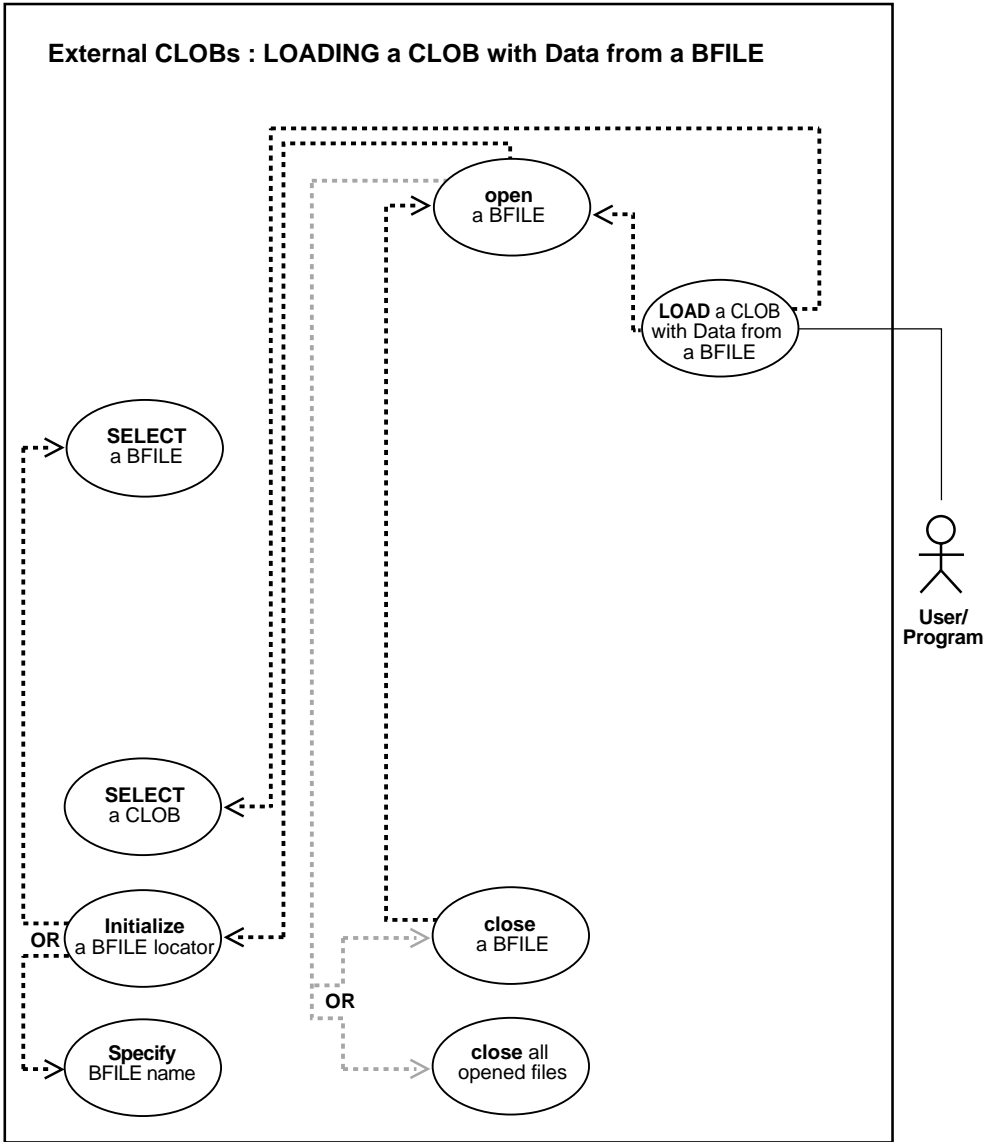
```
BLOB */
  bytes_wt := dst_offset - dst_osin;
  dbms_output.put_line(' Number of bytes written to the BLOB ' || bytes_wt ) ;
  /* If there is no exception the number of bytes read should equal to the
number of bytes written */

END ;
```



# Loading a CLOB with BFILE Data

Figure 12-10 Use Case Diagram: Loading a CLOB or NCLOB with Character Data



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Loading a LOB with BFILE Data](#) on page 12-46
- [Loading a BLOB with BFILE Data](#) on page 12-55

**Purpose**

This procedure describes how to load an internal persistent CLOB or NCLOB with character data from a BFILE.

**Usage Notes**

Use `LOADBLOBFROMFILE` for loading of binary data and `LOADCLOBFROMFILE` for loading of text as the latter method lets you specify the character set id of the BFILE. Note that since this functionality does not support BFILE on the client side, the loading takes place on the server side only. The `LOADCLOBFROMFILE` API allows you to specify the character set id of the BFILE ensuring that the character set is properly converted from the BFILE data character set to the destination CLOB/NCLOB character set.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): *Oracle9i Supplied PL/SQL Packages and Types Reference* Chapter 18, "DBMS\_LOB" — `LOADCLOBFROMFILE`

**Scenario**

The example procedures in this section use the `Print_media` table of the Product Media sample schema and assume that there is an operating system source directory that contains the character LOB data to be loaded into the target CLOB or NCLOB.

**Examples**

The example, ["PL/SQL \(DBMS\\_LOB Package\): Loading a CLOB/NCLOB with BFILE Data"](#), illustrates `LOADCLOBFROMFILE` usage in the PL/SQL programmatic environment. (Other programmatic environments are not supported.)

**See Also:**

- [PL/SQL \(DBMS\\_LOB Package\): Loading an Internal Persistent CLOB with BFILE Data](#) on page 10-49
- [PL/SQL \(DBMS\\_LOB Package\): Loading a Temporary CLOB/NCLOB with BFILE Data](#) on page 11-54

**PL/SQL (DBMS\_LOB Package): Loading a CLOB/NCLOB with BFILE Data**

The following example illustrates:

- How to load an internal persistent CLOB or NCLOB with character data from a BFILE.
- How to use default csid (0).
- How to load the entire file without calling `getlength` for the BFILE.
- How to find out the actual amount loaded using return offsets.

This example assumes that `ad_source` is a BFILE in UTF8 character set format and the database charset is UTF8.

```

DECLARE
    src_loc      bfile := bfilename('ADVERT_DIR','ad_source_1000') ;
    dst_loc      clob ;
    amt          number := dbms_lob.lobmaxsize;
    src_offset   number := 1 ;
    dst_offset   number := 1 ;
    lang_ctx     number := dbms_lob.default_lang_ctx;
    warning      number;
BEGIN
    select ad_sourcetext into dst_loc from Print_media
        where product_id = 3000 and ad_id = 1000 for update ;
    dbms_lob.fileopen(src_loc, dbms_lob.file_readonly);

    /* The default_csid can be used when the BFILE encoding is in the same charset
     * as the destination CLOB/NCLOB charset
     */
    dbms_lob.LOADCLOBFROMFILE(dst_loc,src_loc, amt, dst_offset, src_offset,
        dbms_lob.default_csid, lang_ctx,warning) ;

    commit;

    dbms_output.put_line(' Amount specified ' || amt ) ;
    dbms_output.put_line(' Number of bytes read from source: ' ||

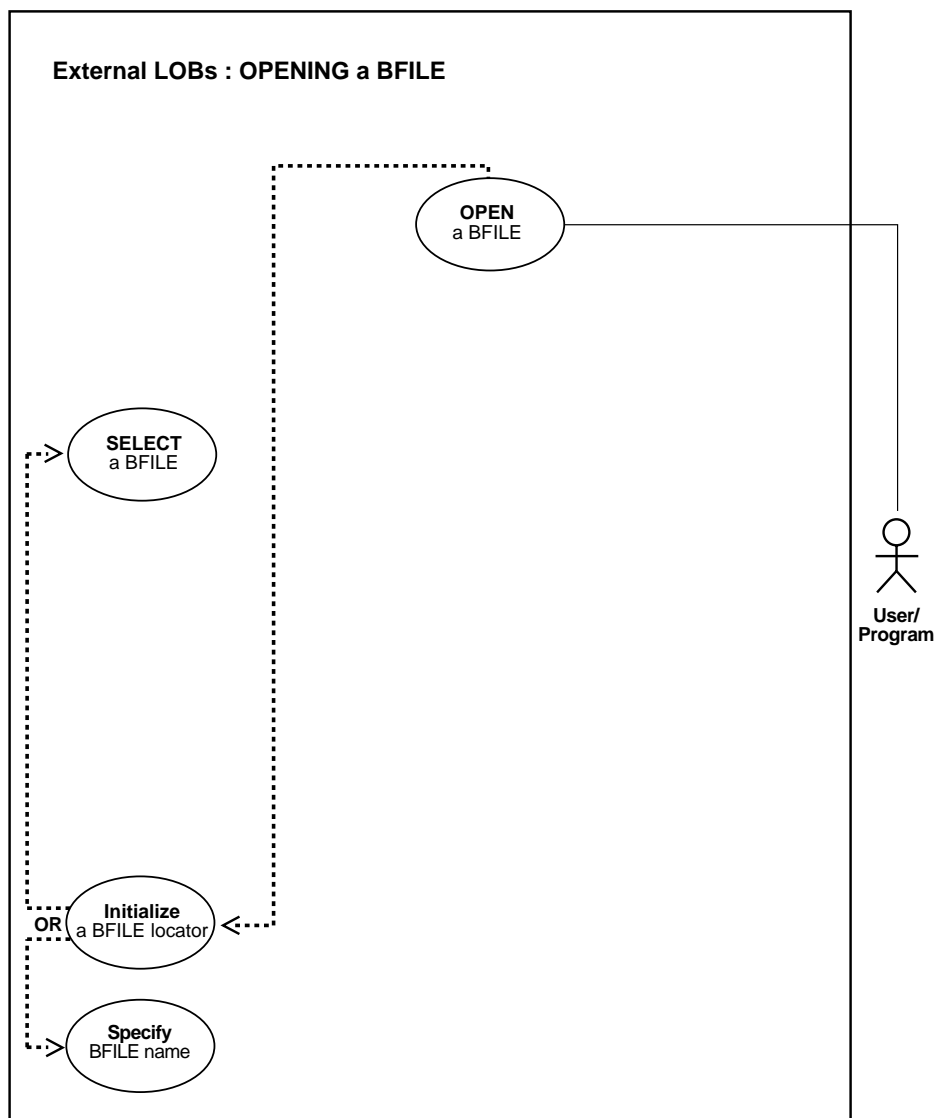
```

```
        (src_offset-1));
dbms_output.put_line(' Number of characters written to destination: ' ||
        (dst_offset-1) );
if (warning = dbms_lob.warn_inconvertible_char)
then
    dbms_output.put_line('Warning: Inconvertible character');
end if;

dbms_lob.filecloseall() ;
END ;
```

# Ways to Open a BFILE

Figure 12-11 Use Case Diagram: Ways to Open a BFILE



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Opening a BFILE with FILEOPEN](#) on page 12-65
- [Opening a BFILE with OPEN](#) on page 12-70

## **Recommendation: Use OPEN to Open BFILE**

Opening a BFILE using `FILEOPEN` is still supported; however, using `OPEN` is recommended for extensibility in future releases.

## **Specify the Maximum Number of Open BFILES: `SESSION_MAX_OPEN_FILES`**

A limited number of BFILES can be open simultaneously in each 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 in each 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.

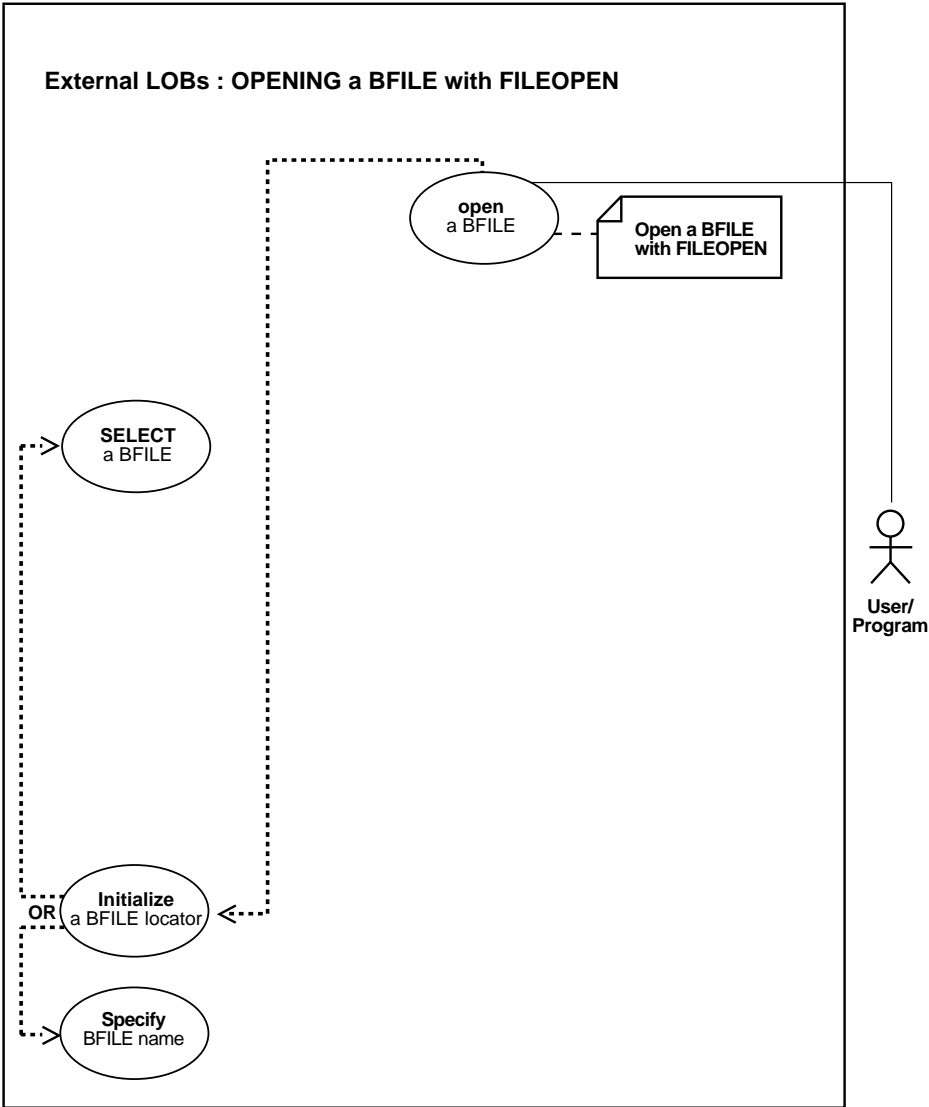
To close all open files, use the `FILECLOSEALL` call.

### **Close Files After Use**

It is good practice to close files after use to keep the `SESSION_MAX_OPEN_FILES` value small. Choosing a larger value would entail a higher memory usage.

# Opening a BFILE with FILEOPEN

Figure 12-12 Use Case Diagram: Opening a BFILE with FILEOPEN



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Ways to Open a BFILE](#) on page 12-63
- [Opening a BFILE with OPEN](#) on page 12-70

**Purpose**

This procedure describes how to open a BFILE using `FILEOPEN`.

**Usage Notes**

While you can continue to use the older `FILEOPEN` form, we *strongly recommend* that you switch to using `OPEN`, because this facilitates future extensibility.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — `FILEOPEN`, `FILECLOSE`
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations, for usage notes. Chapter 15, "Relational Functions" — `LOB Functions`, `OCILobFileOpen`, `OCILobFileClose`, `OCILobFileName`
- COBOL (Pro\*COBOL): A syntax reference is not applicable in this release.
- C/C++ (Pro\*C/C++): A syntax reference is not applicable in this release.
- Visual Basic (OO4O): A syntax reference is not applicable in this release.
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

**Scenario**

These examples open `keyboard_photo3060` in operating system file `ADPHOTO_DIR`.



## Examples

Examples are provided in the following four programmatic environments:

- [PL/SQL: Opening a BFILE with FILEOPEN](#) on page 12-67
- [C \(OCI\): Opening a BFILE with FILEOPEN](#) on page 12-67
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- [Visual Basic \(OO4O\): Opening a BFILE with FILEOPEN](#) on page 12-68
- [Java \(JDBC\): Opening a BFILE with FILEOPEN](#) on page 12-68

### PL/SQL: Opening a BFILE with FILEOPEN

```

/* Opening a BFILE with FILEOPEN [Example script: 3973.sql] */
/* Procedure openBFILE_procOne is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE openBFILE_procOne IS
    File_loc    BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo3060');
BEGIN
    /* Open the BFILE: */
    DBMS_LOB.FILEOPEN (File_loc, DBMS_LOB.FILE_READONLY);
    /* ... Do some processing. */
    DBMS_LOB.FILECLOSE(File_loc);
END;

```

### C (OCI): Opening a BFILE with FILEOPEN

```

/* Opening a BFILE with FILEOPEN */

void BfileOpen(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    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 *)"ADGRAPHIC_DIR",
                                (ub2)strlen("ADGRAPHIC_DIR"),
                                (OraText *)"keyboard_graphic_3106_
13001",
                                (ub2)strlen("keyboard_graphic_3106_
13001"))));
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);
}
```

## Visual Basic (OO4O): Opening a BFILE with FILEOPEN

---

---

**Note:** At the present time, OO4O only offers BFILE opening with OPEN (see ["Visual Basic \(OO4O\) Opening a BFILE with OPEN"](#) on page 12-75).

---

---

## Java (JDBC): Opening a BFILE with FILEOPEN

```
// Opening a BFILE with FILEOPEN
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:
    DriverManager.registerDriver(new 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;

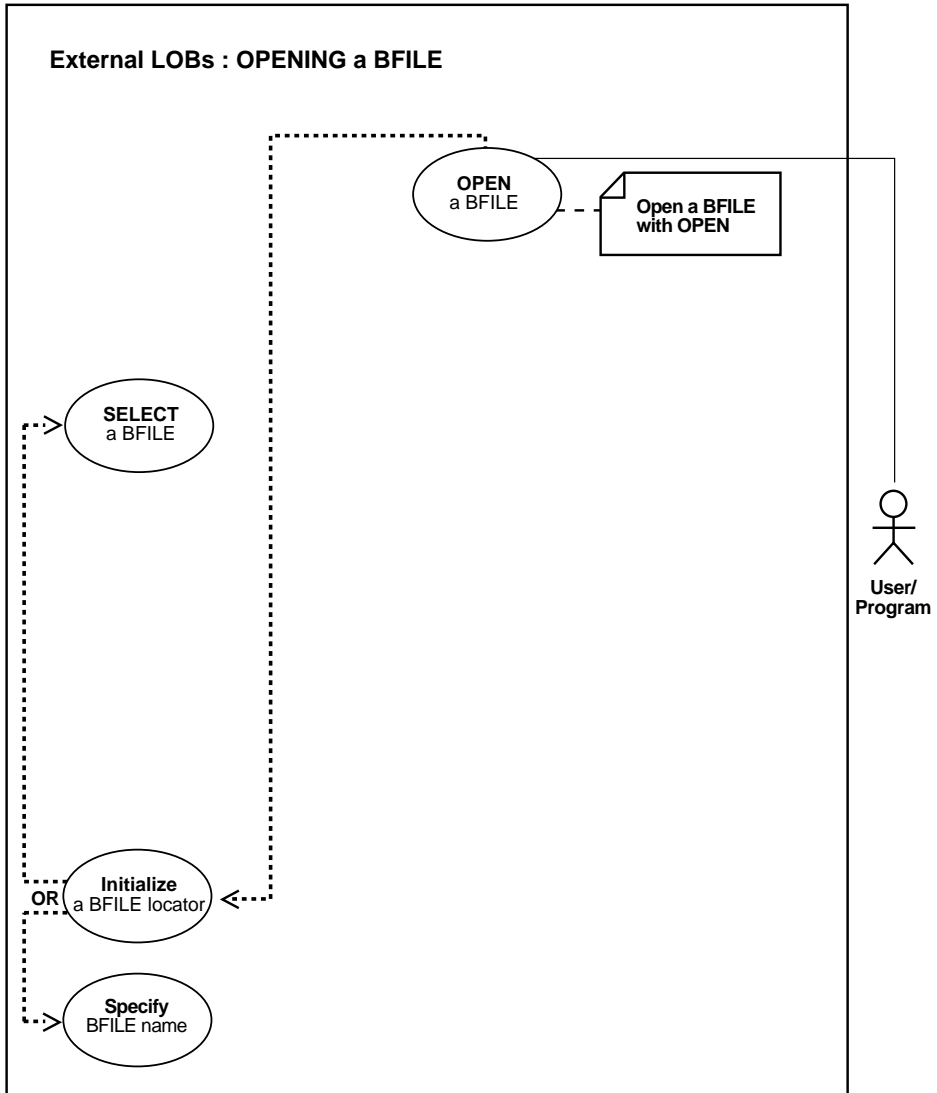
        rset = stmt.executeQuery (
            "SELECT BFILENAME('AD_GRAPHIC', 'monitor_3060') FROM DUAL");
        if (rset.next())
        {
            src_lob = ((OracleResultSet)rset).getBFILE (1);

            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();
    }
}
```

## Opening a BFILE with OPEN

Figure 12–13 Use Case Diagram: Opening a BFILE with OPEN



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Ways to Open a BFILE](#) on page 12-63
- [Opening a BFILE with FILEOPEN](#) on page 12-65

**Purpose**

This procedure describes how to open a BFILE with `OPEN`.

**Usage Notes**

Not applicable.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — OPEN
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations", for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobOpen, OCILobClose
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB statements, and embedded SQL and precompiler directives — LOB OPEN.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB OPEN.
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBfile > METHODS > Open, and > OBJECTS > OraDynaset > METHODS > MoveFirst MoveLast MovePrevious MoveNext
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*):

## Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These examples open an image in operating system file `ADPHOTO_DIR`.

### Examples

Examples are provided in the following six programmatic environments:

- [PL/SQL: Opening a BFILE with OPEN](#) on page 12-72
- [C \(OCI\): Opening a BFILE with OPEN](#) on page 12-72
- [C/C++ \(Pro\\*C/C++\): Opening a BFILE with OPEN](#) on page 12-74
- [COBOL \(Pro\\*COBOL\): Opening a BFILE with OPEN](#) on page 12-73
- [Visual Basic \(OO4O\) Opening a BFILE with OPEN](#) on page 12-75
- [Java \(JDBC\): Opening a BFILE with OPEN](#) on page 12-76

### PL/SQL: Opening a BFILE with OPEN

```
/* Opening a BFILE with OPEN. */
/* Procedure openBFILE_procTwo is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE openBFILE_procTwo IS
  File_loc    BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_11001');
BEGIN
  /* Open the BFILE: */
  DBMS_LOB.OPEN (File_loc, DBMS_LOB.LOB_READONLY);
  /* ... Do some processing: */
  DBMS_LOB.CLOSE(File_loc);
END;
```

### C (OCI): Opening a BFILE with OPEN

```
/* Opening a BFILE with OPEN. */

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

```

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 *) "ADGRAPHIC_DIR", (ub2)strlen("ADGRAPHIC_
DIR"),
                                (OraText *) "keyboard_graphic_3106_13001",
                                (ub2)strlen("keyboard_graphic_3106_13001"))));
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);
}

```

## COBOL (Pro\*COBOL): Opening a BFILE with OPEN

```

* Opening a BFILE with OPEN. [Example script: 3978.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. OPEN-BFILE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3106_13001" 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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNDR.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNDR, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

## C/C++ (Pro\*C/C++): Opening a BFILE with OPEN

```
/* Opening a BFILE using OPEN. [Example script: 3979.pc]
In Pro*C/C++ there is only one form of OPEN 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 = "GRAPHIC_DIR", *Name = "mousepad_2056";

    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;
}

```

## Visual Basic (OO4O) Opening a BFILE with OPEN

```

'Opening a BFILE using OPEN. [Example script: 3981.txt]
Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile
Set OraDyn = OraDb.CreateDynaset("select * from Print_media",ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

'Go to the last row and open the Bfile for reading:
OraDyn.MoveLast

```

```
OraAdGraphic.Open 'Open Bfile for reading  
'Do some processing:  
OraAdGraphic.Close
```

## Java (JDBC): Opening a BFILE with OPEN

```
// Opening a BFILE with OPEN. [Example script: 3982.java]  
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:  
        DriverManager.registerDriver(new 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('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') FROM  
                DUAL");
```

```
if (rset.next())
{
    src_lob = ((OracleResultSet)rset).getBFILE (1);

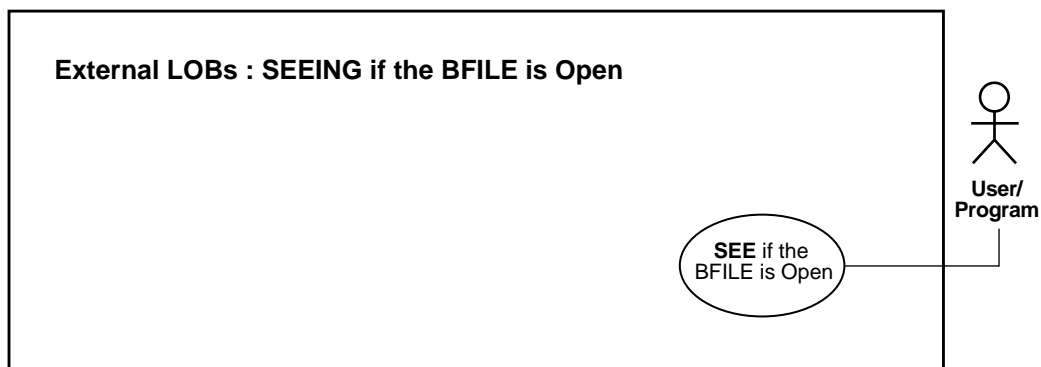
    OracleCallableStatement cstmt = (OracleCallableStatement)
    conn.prepareCall ("begin dbms_lob.open (?,dbms_lob.lob_readonly);
end;");

    cstmt.registerOutParameter(1,OracleTypes.BFILE);
    cstmt.setBFILE (1, src_lob);
    cstmt.execute();
    src_lob = cstmt.getBFILE(1);
    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();
}
}
```

## Ways to See If a BFILE is Open

*Figure 12–14 Use Case Diagram: Two Ways to See If a BFILE is Open*



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Checking If the BFILE is Open with FILEISOPEN](#) on page 12-80
- [Checking If a BFILE is Open Using ISOPEN](#) on page 12-86

### Recommendation: Use OPEN to Open BFILE

As you can see by comparing the code, these alternative methods are very similar. While you can continue to use the older `FILEISOPEN` form, we strongly recommend that you switch to using `ISOPEN`, because this facilitates future extensibility.

### Specify the Maximum Number of Open BFILES: `SESSION_MAX_OPEN_FILES`

A limited number of `BFILES` can be open simultaneously in each 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.

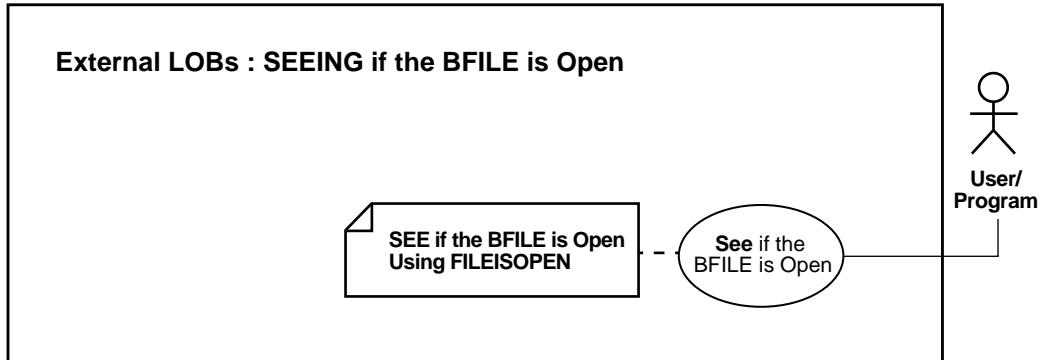
That is, a maximum of 10 files can be opened simultaneously in each session if the default value is utilized. The database administrator can change the value of this parameter in the `init.ora` file. For example:

```
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.

## Checking If the BFILE is Open with FILEISOPEN

Figure 12–15 Use Case Diagram: Checking If BFILE is Open Using FILEISOPEN



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Ways to See If a BFILE is Open](#) on page 12-78
- [Checking If a BFILE is Open Using ISOPEN](#) on page 12-86

**Purpose**

This procedure describes how to see if a BFILE is OPEN with FILEISOPEN.

**Usage Notes**

While you can continue to use the older FILEISOPEN form, we *strongly recommend* that you switch to using ISOPEN, because this facilitates future extensibility.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — FILEISOPEN

- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileIsOpen
- COBOL (Pro\*COBOL): A syntax reference is not applicable in this release.
- C/C++ (Pro\*C/C++): A syntax reference is not applicable in this release.
- Visual Basic (OO4O): A syntax reference is not applicable in this release.
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These examples query whether a BFILE associated with `ad_graphic` is open.

### Examples

Examples are provided in the following programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Checking If the BFILE is Open with FILEISOPEN](#) on page 12-81
- [C \(OCI\): Checking If the BFILE is Open with FILEISOPEN](#) on page 12-82
- C/C++ (Pro\*C/C++): No example is provided with this release.
- COBOL (Pro\*COBOL): No example is provided with this release.
- Visual Basic (OO4O): No example is provided. See note on page 12-84
- [Java \(JDBC\): Checking If the BFILE is Open with FILEISOPEN](#) on page 12-84

## PL/SQL (DBMS\_LOB Package): Checking If the BFILE is Open with FILEISOPEN

```
/* Checking if the BFILE is OPEN with FILEISOPEN. [Example script: 3984.sql]
   Procedure seeIfOpenBFILE_procOne is not part of DBMS_LOB package: */
```

```
CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procOne IS
  File_loc      BFILE;
  RetVal        INTEGER;
BEGIN
  /* Select the LOB, initializing the BFILE locator: */
  SELECT ad_graphic INTO File_loc FROM Print_media
```

```
        WHERE product_ID = 3060 AND ad_id = 11001;
RetVal := DBMS_LOB.FILEISOPEN(File_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;
```

## C (OCI): Checking If the BFILE is Open with FILEISOPEN

```
/* Checking if the BFILE is open with FILEISOPEN. [Example script: 3985.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt =
        (text *) "SELECT ad_graphic FROM Print_media
                WHERE product_id=3106 AND ad_id = 13001";

    /* 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(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT)
    || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                       (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 BfileFileIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    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(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                   (ub1)OCI_FILE_READONLY));

    checkerr(errhp, OCILobFileIsOpen(svchp, errhp, bfile_loc, &flag));

    if (flag == TRUE)
    {
        printf("File is open\n");
    }
    else
    {
        printf("File is not open\n");
    }

    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

## Visual Basic (OO4O): Checking If the BFILE is Open with FILEISOPEN

---

---

**Note:** At the present time, OO4O only offers ISOPEN to test whether or not a BFILE is open (see "[Visual Basic \(OO4O\): Checking If the BFILE is Open with FILEISOPEN](#)" on page 12-84).

---

---

## Java (JDBC): Checking If the BFILE is Open with FILEISOPEN

```
// Checking if the BFEIL is open with FILEISOPEN.[Example script:3986.java]

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:
        DriverManager.registerDriver(new 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 = false;

    rset = stmt.executeQuery (
        "SELECT BFILENAME('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') FROM
DUAL");
    if (rset.next())
    {
        src_lob = ((OracleResultSet)rset).getBFILE (1);
    }

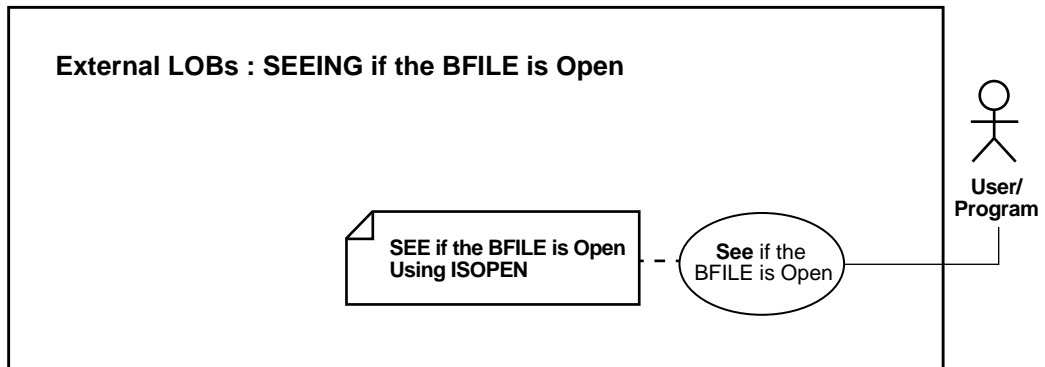
    result = src_lob.isFileOpen();
    System.out.println(
        "result of fileIsOpen() before opening file : " + result);
    if (!result)
        src_lob.openFile();

    result = src_lob.isFileOpen();
    System.out.println(
        "result of fileIsOpen() after opening file : " + result);

    // Close the BFILE, statement and connection:
    src_lob.closeFile();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Checking If a BFILE is Open Using ISOPEN

Figure 12–16 Use Case Diagram: Checking If a BFILE is Open Using ISOPEN



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs
- [Ways to See If a BFILE is Open](#) on page 12-78
- [Checking If the BFILE is Open with FILEISOPEN](#) on page 12-80

**Purpose**

This procedure describes how to see if a BFILE is open using ISOPEN.

**Usage Notes**

Not applicable.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — ISOPEN

- **C (OCI)** (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileIsOpen
- **COBOL (Pro\*COBOL)** (*Pro\*COBOL Precompiler Programmer's Guide*) for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE ... ISOPEN.
- **C/C++ (Pro\*C/C++)** (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB DESCRIBE ... ISOPEN
- **Visual Basic (OO4O)** (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBfile > METHODS > IsOpen and > OBJECTS > OraDynaset
- **Java (JDBC)** (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples query whether the a BFILE is open that is associated with ad\_graphic.

## Examples

Examples are provided in the following six programmatic environments:

- **PL/SQL (DBMS\_LOB Package)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-87
- **C (OCI)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-88
- **COBOL (Pro\*COBOL)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-90
- **C/C++ (Pro\*C/C++)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-91
- **Visual Basic (OO4O)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-92
- **Java (JDBC)**: [Checking If the BFILE is Open with ISOPEN](#) on page 12-93

## PL/SQL (DBMS\_LOB Package): Checking If the BFILE is Open with ISOPEN

```
/* Checking if the BFILE is open with ISOPEN. [Example script: 3987.sql] */
/* Procedure seeIfOpenBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfOpenBFILE_procTwo IS
    File_loc      BFILE;
    RetVal        INTEGER;
BEGIN
    /* Select the LOB, initializing the BFILE locator: */
    SELECT ad_graphic INTO File_loc FROM Print_media
        WHERE product_ID = 3060 AND ad_id = 11001;
    RetVal := DBMS_LOB.ISOPEN(File_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;
```

## C (OCI): Checking If the BFILE is Open with ISOPEN

```
/* Checking if the BFILE is Open with ISOPEN. Example script:3988.c */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt =
        (text *) "SELECT ad_graphic FROM Print_media
                WHERE product_id=3106 AND ad_id = 13001";

    /* Prepare the SQL select statement */
    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, selstmt,
        (ub4) strlen((char *) selstmt),
        (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));
```

```

/* Call define for the bfile column */
checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT)
|| OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                   (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                   (dvoid *)0, (ub2 *)0, (ub2 *)0,
                   OCI_DEFAULT));

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

void BfileIsOpen(envhp, errhp, svchp, stmthp)
OCIEnv *envhp;
OCIError *errhp;
OCISvcCtx *svchp;
OCISstmt *stmthp;
{

    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(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                              (ub1)OCI_FILE_READONLY));

    checkerr(errhp, OCILobIsOpen(svchp, errhp, bfile_loc, &flag));

    if (flag == TRUE)
    {
        printf("File is open\n");
    }
    else
    {
        printf("File is not open\n");
    }
}

```

```
checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

/* Free the locator descriptor */
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

## COBOL (Pro\*COBOL): Checking If the BFILE is Open with ISOPEN

```
* Checking if BFILE is open with ISOPEN. [Example script: 3989.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. OPEN-BFILE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 "ADPHOTO_DIR" TO DIR-ALIAS-ARR.
MOVE 9 TO DIR-ALIAS-LEN.
MOVE "keyboard_photo_3060_11001" 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
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.

SQL-ERROR.
    EXEC SQL
        WHENEVER SQLERROR CONTINUE
    END-EXEC.
    MOVE ORASLNR TO ORASLNDR.
    DISPLAY " ".
    DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNDR, ":".
    DISPLAY " ".
    DISPLAY SQLERRMC.
    EXEC SQL
        ROLLBACK WORK RELEASE
    END-EXEC.
    STOP RUN.

```

## C/C++ (Pro\*C/C++): Checking If the BFILE is Open with ISOPEN

```

/* Checking if the BFILE is open with ISOPEN. [Example script: 3990.pc]
   In Pro*C/C++, there is only one form of ISOPEN to determine whether
   or not a BFILE is OPEN. There is no FILEISOPEN, 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 ad_graphic INTO :Lob_loc FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;
}
```

## Visual Basic (OO4O): Checking If the BFILE is Open with ISOPEN

```
' Checking if the BFILE is open with ISOPEN. [Example script: 3992.txt]
Dim OraDyn as OraDynaset, OraAdGraphic as OraBFile, amount_read%, chunksize%,
chunk

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
```

```

Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

If OraAdGraphic.IsOpen then
    'Process, if the file is already open:
Else
    'Process, if the file is not open, and return an error:
End If

```

## Java (JDBC): Checking If the BFILE is Open with ISOPEN

```

// Checking if the BFILE is open with ISOPEN. [Example script: 3993.java]

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:
        DriverManager.registerDriver(new 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('ADGRAPHIC_DIR', 'monitor_graphic_3060_11001') 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();

        int i = cstmt.getInt(1);
        System.out.println("The result is: " + Integer.toString(i));

        OracleCallableStatement cstmt2 = (OracleCallableStatement)
        conn.prepareCall (
            "BEGIN DBMS_LOB.OPEN(?,DBMS_LOB.LOB_READONLY); END;");
        cstmt2.setBFILE(1, bfile);
        cstmt2.execute();

        System.out.println("The BFILE has been opened with a call to "
+"DBMS_LOB.OPEN()");

        // Use the existing cstmt handle to re-query the status of the locator:
        cstmt.setBFILE(2, bfile);
        cstmt.execute();
        i = cstmt.getInt(1);
        System.out.println("This result is: " + Integer.toString(i));

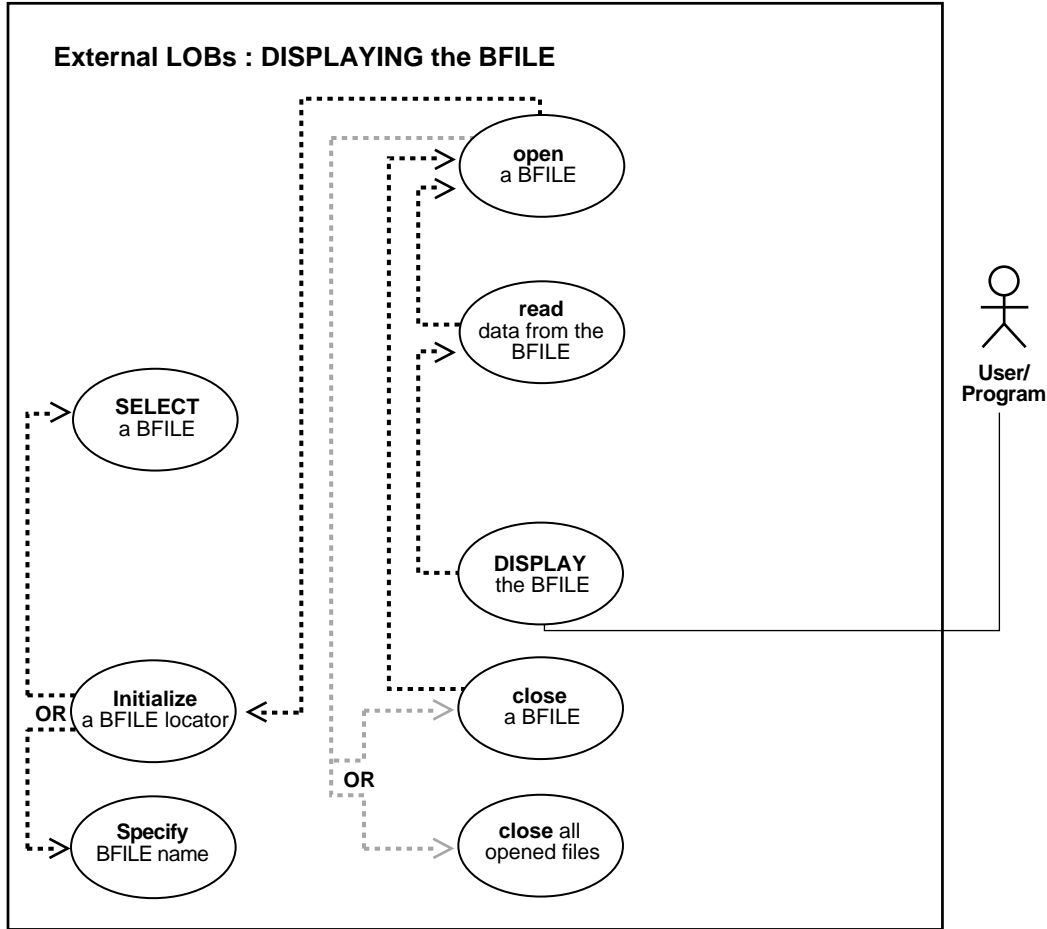
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)

```

```
    {  
      e.printStackTrace();  
    }  
  }  
}
```

## Displaying BFILE Data

Figure 12–17 Use Case Diagram: Displaying BFILE Data



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to display BFILE data.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — READ. Chapter 29, "DBMS\_OUTPUT" - PUT\_LINE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileOpen, OCILobRead
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB READ, DISPLAY.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements" — READ
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > METHODS > Read, and OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > PollingAmount, Offset, Status. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

These examples open and display BFILE data.

## Examples

Examples are provided in six programmatic environments:

- [PL/SQL: Displaying BFILE Data](#) on page 12-98
- [C \(OCI\): Displaying BFILE Data](#) on page 12-98

- [COBOL \(Pro\\*COBOL\): Displaying BFILE Data](#) on page 12-101
- [C/C++ \(Pro\\*C/C++\): Displaying BFILE Data](#) on page 12-103
- [Visual Basic \(OO4O\): Displaying BFILE Data](#) on page 12-104
- [Java \(JDBC\): Displaying BFILE Data](#) on page 12-105

## PL/SQL: Displaying BFILE Data

```
/* Displaying BFILE data. [Example script: 3994.sql] */
/* Procedure displayBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE displayBFILE_proc IS
    File_loc BFILE;
    Buffer RAW(1024);
    Amount BINARY_INTEGER := 1024;
    Position INTEGER := 1;
BEGIN
    /* Select the LOB: */
    SELECT ad_graphic INTO File_loc
        FROM print_media WHERE Product_ID = 3060 AND ad_id = 11001;
    /* Opening the BFILE: */
    DBMS_LOB.OPEN (File_loc, DBMS_LOB.LOB_READONLY);
    LOOP
        DBMS_LOB.READ (File_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 (File_loc);
    EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('End of data');
END;
```

## C (OCI): Displaying BFILE Data

```
/* Displaying BFILE data. [Example script: 3995.c] */
/* Select the lob/bfile from the Print_media table */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError *errhp;
```



```

OCISvcCtx      *svchp;
OCIStmt        *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt =
        (text *) "SELECT ad_graphic FROM Print_media
                WHERE product_id=3106 AND ad_id = 13001";

    /* 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(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0 , SOLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT)
    || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                    (dvoid *)&Lob_loc, 0 , SOLT_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, errhp, svchp, stmthp)
OCIEnv        *envhp;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    /* 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(bfile_loc, errhp, svchp, stmthp);

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 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:
            /* 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);
}

```

## COBOL (Pro\*COBOL): Displaying BFILE Data

```

* Displaying BFILE data. [Example script: 3996.pco]
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 AD_GRAPHIC INTO :SRC-BFILE
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
    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 SQLEERROR 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.

```

## C/C++ (Pro\*C/C++): Displaying BFILE Data

```

/* Displaying BFILE data. [Example script: 3997.pc]
   This example reads 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 ad_graphic INTO :Lob_loc
        FROM Print_media WHERE Product_ID = 2056 AND ad_id = 12001;
/* 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;
}

```

## Visual Basic (OO4O): Displaying BFILE Data

```

' Displaying BFILE data. [Example script: 3999.txt]
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%,
chunk As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

```

```

OraAdGraphic.offset = 1
OraAdGraphic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents

'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.Read(chunk, chunksize)

While OraAdGraphic.Status = ORALOB_NEED_DATA
    amount_read = OraAdGraphic.Read(chunk, chunksize)
Wend
OraAdGraphic.Close

```

## Java (JDBC): Displaying BFILE Data

```

// Displaying BFILE data. [Example script: 4000.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    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();
}

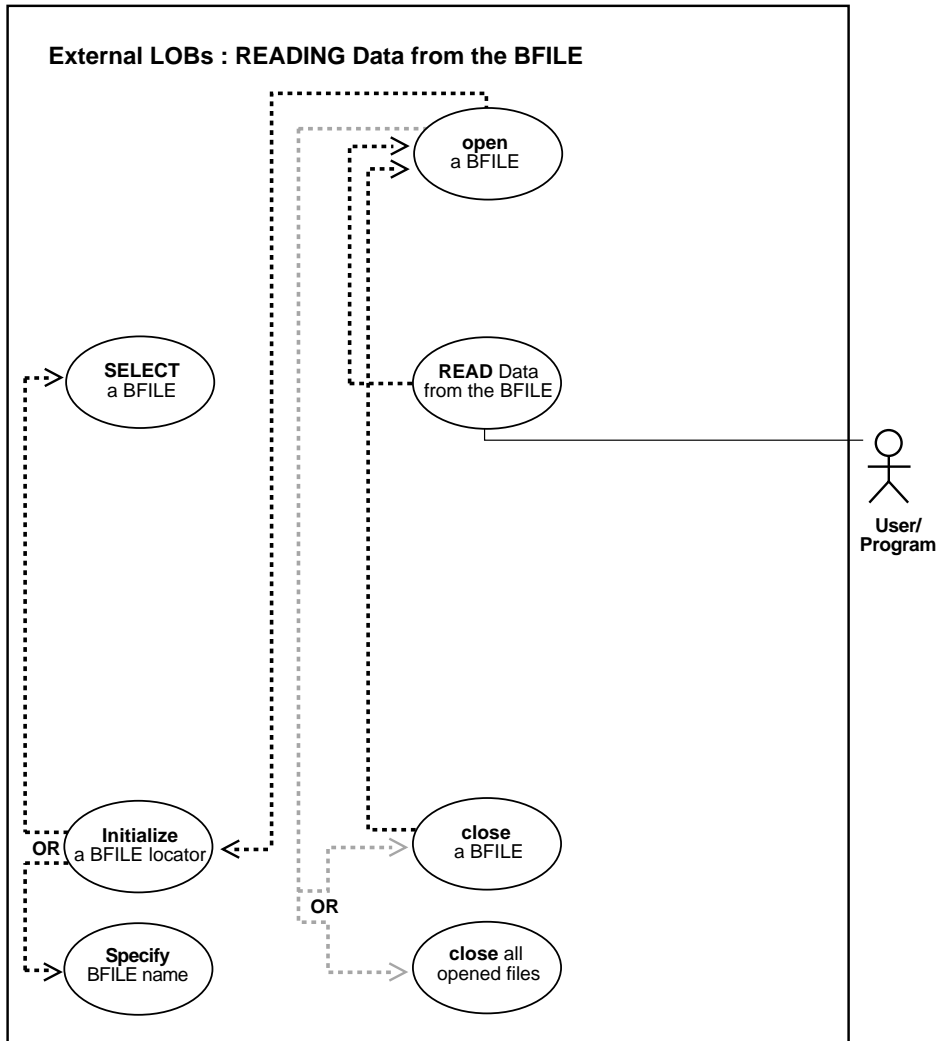
```



```
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

## Reading Data from a BFILE

Figure 12-18 Use Case Diagram: Reading Data from a BFILE



**See Also:** [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

## Purpose

This procedure describes how to read data from a BFILE.

## Usage Notes

**Always Specify 4 GByte - 1 Regardless of LOB Size** 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 specify an input amount of 4 GByte -1 regardless of the starting `offset` and the amount of data in the LOB. Hence, 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.

## Example

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, buf1, 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 using polling or a callback. See Also: [Chapter 10, "Internal Persistent LOBs"](#), ["Loading a LOB with Data From a BFILE"](#), Usage Notes.

---



---

## The Amount Parameter

- In **DBMS\_LOB.READ**, the amount parameter can be larger than the size of the data. In PL/SQL, the amount parameter should be less than or equal to the size of the buffer, and the buffer size is limited to 32K.
- In **OCILobRead**, you can specify `amount = 4 Gb - 1`, and it will read to the end of the LOB.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB)** (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — READ
- **C (OCI)** (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobRead
- **COBOL (Pro\*COBOL)** (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB READ.
- **C/C++ (Pro\*C/C++)** (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB READ
- **Visual Basic (OO4O)** (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > METHODS > Read, and OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > PollingAmount, Offset, Status. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- **Java (JDBC)** (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples read a photograph into `ad_graphic` from a BFILE in `'ADPHOTO_DIR'`.

## Examples

Examples are provided in these six programmatic environments:

- **PL/SQL (DBMS\_LOB Package)**: [Reading Data from a BFILE](#) on page 12-111
- **C (OCI)**: [Reading Data from a BFILE](#) on page 12-111
- **COBOL (Pro\*COBOL)**: [Reading Data from a BFILE](#) on page 12-113

- [C/C++ \(Pro\\*C/C++\): Reading Data from a BFILE](#) on page 12-114
- [Visual Basic \(OO4O\): Reading Data from a BFILE](#) on page 12-115
- [Java \(JDBC\): Reading Data from a BFILE](#) on page 12-116

## PL/SQL (DBMS\_LOB Package): Reading Data from a BFILE

```

/* Reading data from a BFILE. [Example script: 4002.sql] */
/* Procedure readBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE readBFILE_proc IS
    File_loc      BFILE := BFILENAME('ADPHOTO_DIR',
        'keyboard_photo_3060_11001');
    Amount        INTEGER := 32767;
    Position      INTEGER := 1;
    Buffer         RAW(32767);
BEGIN
    /* Select the LOB: */
    SELECT ad_graphic INTO File_loc FROM print_media
        WHERE Product_ID = 3060 AND ad-Id = 11001;
    /* Open the BFILE: */
    DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
    /* Read data: */
    DBMS_LOB.READ(File_loc, Amount, Position, Buffer);
    /* Close the BFILE: */
    DBMS_LOB.CLOSE(File_loc);
END;

```

## C (OCI): Reading Data from a BFILE

```

/* Reading data from a BFILE. [Example script: 4003.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt =
        (text *) "SELECT ad_graphic FROM Print_media
            WHERE product_id=3106 AND ad_id = 13001";

```

```

/* Prepare the SQL select statement */
checkerr (errhp, OCISstmtPrepare(stmthp, errhp, selstmt,
                                (ub4) strlen((char *) selstmt),
                                (ub4) OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT));

/* Call define for the bfile column */
checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT)
|| OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                  (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                  (dvoid *)0, (ub2 *)0, (ub2 *)0,
                  OCI_DEFAULT));

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

#define MAXBUFLLEN 32767

void BfileRead(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{

    OCILobLocator *bfile_loc;
    ub1 bufp[MAXBUFLLEN];
    ub4 buflen, amt, offset;
    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(bfile_loc, errhp, svchp, stmthp);

    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);

retval = 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);
}

```

## COBOL (Pro\*COBOL): Reading Data from a BFILE

```

* Reading data from a BFILE. [Example script: 4004.pco]
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 AD_GRAPHIC INTO :BFILE1
    FROM PRINT_MEDIA M WHERE M.PRODUCT_ID = 3106 AND AD_ID = 13001
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.
```

## C/C++ (Pro\*C/C++): Reading Data from a BFILE

```
/* Reading data from BFILE. [Example script: 4005.pc] */

#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 ad_graphic INTO :Lob_loc
        FROM Print_media WHERE Product_ID = 2056;
    /* 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;
}

```

## Visual Basic (OO4O): Reading Data from a BFILE

```

' Reading data from a BFILE [Example script: 4007.txt]

Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%,
chunk As Variant

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

```

```
chunksize = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

OraAdGraphic.offset = 1
OraAdGraphic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents

'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.Read(chunk, chunksize)
While OraAdGraphic.Status = ORALOB_NEED_DATA
    amount_read = OraAdGraphic.Read(chunk, chunksize)
Wend
OraAdGraphic.Close
```

## Java (JDBC): Reading Data from a BFILE

```
// Reading data from a BFILE. [Example script: 4008.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    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

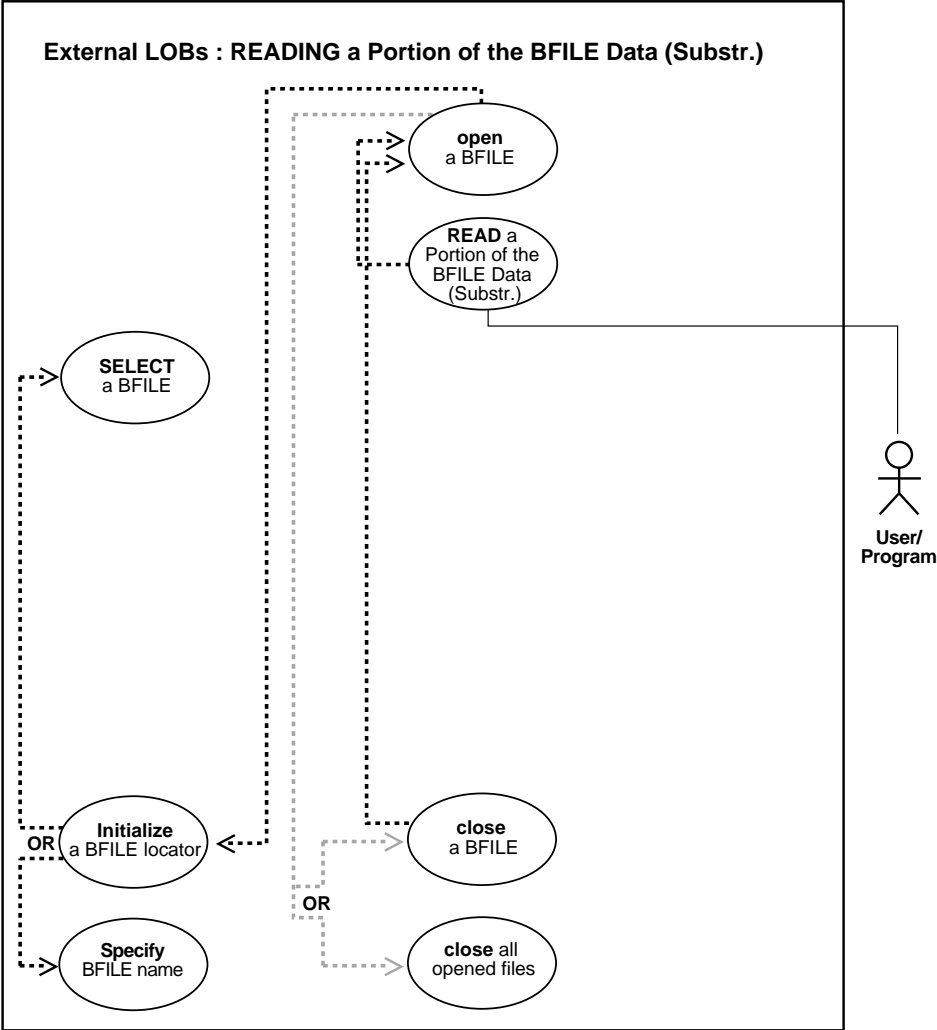
        // We will only display the first CHUNK in this example:
        if (! alreadyDisplayed)
        {
            System.out.println("Bytes read in: " + Integer.toString(length));
            System.out.println(new String(buf));
            alreadyDisplayed = true;
        }
    }
}

// Close the stream, BFILE, statement and connection:
```

```
        in.close();
        src_lob.closeFile();
        stmt.close();
        conn.commit();
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

# Reading a Portion of BFILE Data (substr)

Figure 12-19 Use Case Diagram: Reading a Portion of BFILE Data (substr)



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### **Purpose**

This procedure describes how to read portion of BFILE data (substr).

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — SUBSTR
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB OPEN, LOB CLOSE. See PL/SQL DBMS\_LOB.SUBSTR.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB OPEN. See also PL/SQL DBMS\_LOB.SUBSTR
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > METHODS > Open, and OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > PollingAmount, Offset, Status. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples read a graphic image into `ad_graphic` from BFILE `'ADPHOTO_DIR'`.

## Examples

Examples are provided in these five programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Reading a Portion of BFILE Data \(substr\)](#) on page 12-121
- C (OCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Reading a Portion of BFILE Data \(substr\)](#) on page 12-121
- [C/C++ \(Pro\\*C/C++\): Reading a Portion of BFILE Data \(substr\)](#) on page 12-123
- [Visual Basic \(OO4O\): Reading a Portion of BFILE Data \(substr\)](#) on page 12-124
- [Java \(JDBC\): Reading a Portion of BFILE Data \(substr\)](#) on page 12-124

## PL/SQL (DBMS\_LOB Package): Reading a Portion of BFILE Data (substr)

```

/* Reading portion of a BFILE data using substr. [Example script: 4009.sql] */
/* Procedure substringBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE substringBFILE_proc IS
    File_loc      BFILE;
    Position      INTEGER := 1;
    Buffer         RAW(32767);
BEGIN
    /* Select the LOB: */
    SELECT PMtab.ad_graphic INTO File_loc FROM Print_media PMtab
        WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001;
    /* Open the BFILE: */
    DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
    Buffer := DBMS_LOB.SUBSTR(File_loc, 255, Position);
    /* Close the BFILE: */
    DBMS_LOB.CLOSE(File_loc);
END;
```

## COBOL (Pro\*COBOL): Reading a Portion of BFILE Data (substr)

```
* Reading portion of a BFILE data using substr. [Example script: 4010.pco]
```

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-SUBSTR.
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 INCLUDE SQLCA END-EXEC.
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 PTAB.AD_GRAPHIC INTO :BFILE1
      FROM PRINT_MEDIA PTAB WHERE PTAB.PRODUCT_ID = 3106 AND PTAB.AD_
ID = 13001
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.
EXEC SQL FREE :BFILE1 END-EXEC.
```



## C/C++ (Pro\*C/C++): Reading a Portion of BFILE Data (substr)

```

/* Reading portion of a BFILE data using substr. [Example script: 4011.pc]
   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 PMtab.ad_graphic INTO :Lob_loc
        FROM Print_media PMtab WHERE PMtab.product_id = 2056 AND PMtab.ad_id
= 12001;
    /* 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;
}
```

## Visual Basic (OO40): Reading a Portion of BFILE Data (substr)

```
' Reading portion of a BFILE data using substr. [Example script: 4013.txt]
Dim MySession As OraSession
Dim OraDb As OraDatabase

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%,
chunk

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

chunk_size = 32767
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value
OraMusic.PollingAmount = OraAdGraphic.Size 'Read entire BFILE contents
OraAdGraphic.offset = 255 'Read from the 255th position
'Open the Bfile for reading:
OraAdGraphic.Open
amount_read = OraAdGraphic.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
```

## Java (JDBC): Reading a Portion of BFILE Data (substr)

```
// Reading portion of a BFILE data using substr. [Example script: 4014.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media WHERE product_id = 3106 AND ad_id =
13001");
            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)
```

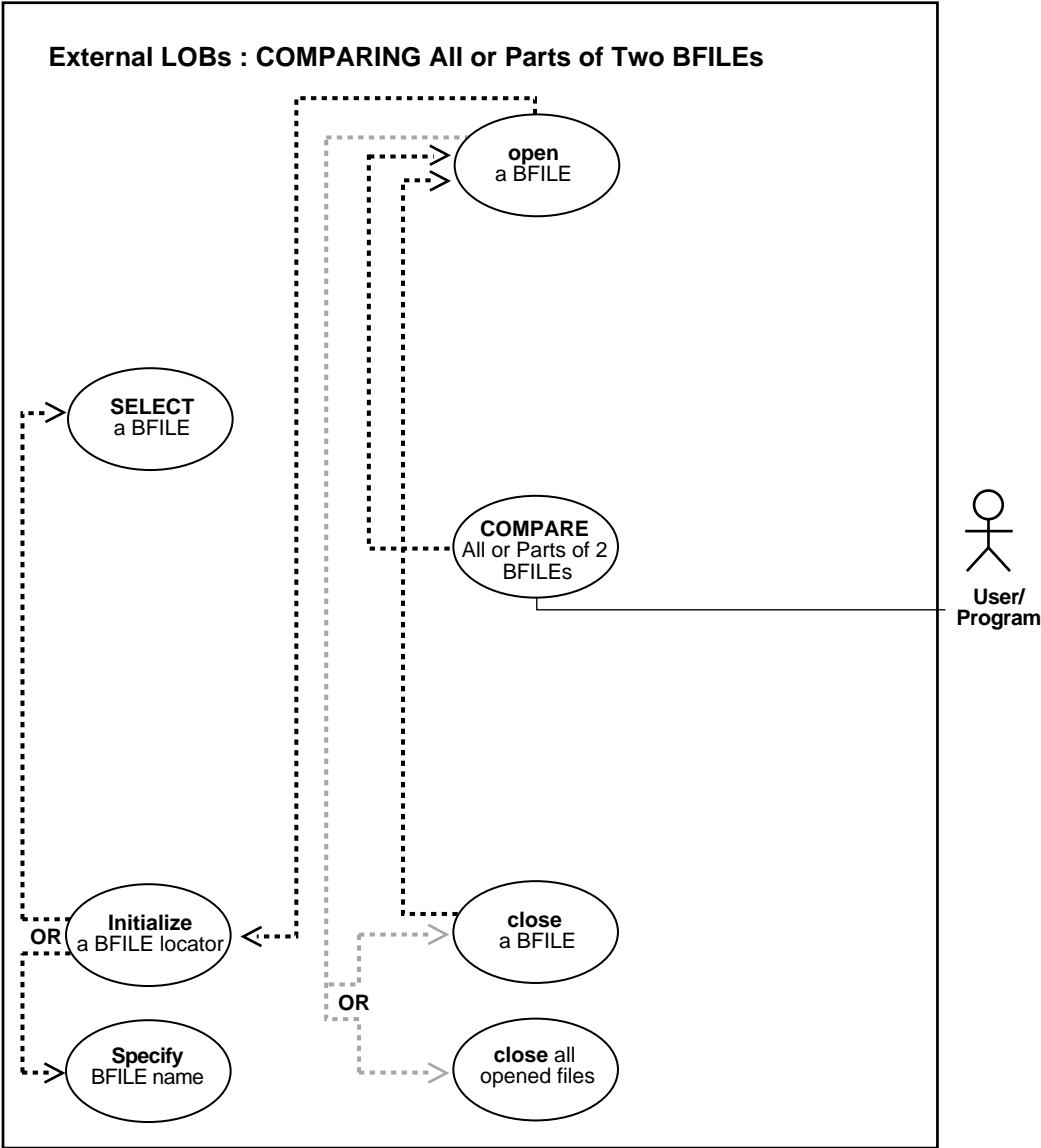
```
{
    // request 255 bytes into buf, starting from offset 1.
    // length = # bytes actually returned from stream:
    length = in.read(buf, 1, 255);
    System.out.println("Bytes read in: " + Integer.toString(length));

    // 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();
}
}
```

# Comparing All or Parts of Two BFILES

Figure 12-20 Use Case Diagram: Comparing All or Parts of Two BFILES



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### **Purpose**

This procedure describes how to compare all or parts of two BFILES.

### **Usage Notes**

Not applicable.

### **Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL(DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — COMPARE
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB OPEN. See PL/SQL DBMS\_LOB.COMPARE.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB OPEN. See PL/SQL DBMS\_LOB.COMPARE.
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > METHODS > Open, Compare, and OO4O Automation Server > OBJECTS > OraDatabase > PROPERTIES > Parameters. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples determine whether a photograph in file, 'ADPHOTO\_DIR', has already been used as a specific ad\_graphic by comparing each data entity bit by bit.

---



---

**Note:** LOBMAXSIZE is set at 4 Gb so that you do not have to find out the length of each BFILE before beginning the comparison.

---



---

## Examples

Examples are provided in these five programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Comparing All or Parts of Two BFILES](#) on page 12-129
- C (OCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Comparing All or Parts of Two BFILES](#) on page 12-130
- [C/C++ \(Pro\\*C/C++\): Comparing All or Parts of Two BFILES](#) on page 12-132
- [Visual Basic \(OO4O\): Comparing All or Parts of Two BFILES](#) on page 12-133
- [Java \(JDBC\): Comparing All or Parts of Two BFILES](#) on page 12-134

## PL/SQL (DBMS\_LOB Package): Comparing All or Parts of Two BFILES

```

/* Comparing all or parts of two BFILES. [Example script: 4015.sql] */
/* Procedure instringBFILE_proc is not part of DBMS_LOB package: */
CREATE OR REPLACE PROCEDURE instringBFILE_proc IS
    File_loc      BFILE;
    Pattern        RAW(32767);
    Position       INTEGER;
BEGIN
    /* Select the LOB: */
    SELECT PMtab.ad_graphic INTO File_loc
    FROM THE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab
    WHERE Product_ID = 3060 AND ad_id = 11001) PMtab
    WHERE Segment = 1;
    /* Open the BFILE: */
    DBMS_LOB.OPEN(File_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(File_loc, Pattern, 1, 2);
    /* Close the BFILE: */

```

```
        DBMS_LOB.CLOSE(File_loc);  
END;
```

## COBOL (Pro\*COBOL): Comparing All or Parts of Two BFILES

\* Comparing all or parts of two BFILES. [Example script: 4016.pco]

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. BFILE-COMPARE.  
ENVIRONMENT DIVISION.  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01  USERID          PIC X(11) VALUES "SAMP/SAMP".  
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 "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.  
MOVE 9 TO DIR-ALIAS-LEN.  
MOVE "keyboard_graphic_3106_13001" 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 AD_GRAPHIC INTO :BFILE2
    FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND ad_id = 13001
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.
    
```

## C/C++ (Pro\*C/C++): Comparing All or Parts of Two BFILES

/\* Comparing all or parts of two BFILES. [Example script: 4017.pc]  
 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 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 = "GRAPHIC_DIR", *Name1 = "mousepad_2056";

    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 Print_media
        WHERE Product_ID = 2056;
    /* 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;
}

```

## Visual Basic (OO4O): Comparing All or Parts of Two BFILES

'Comparing all or parts of two BFILES. [Example script: 4018.txt]  
 '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, OraAdGraphic As OraBfile, OraMyAdGraphic As OraBfile,
OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 3106, ORAPARM_INPUT

```

```
'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
  OraDb.CreateSql(
    "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE product_
id = :id;
      END;", ORASQL_FAILEXEC)

Set OraMyAdGraphic = OraParameters("MyAdGraphic").Value

'Create dynaset:
Set OraDyn =
  OraDb.CreateDynaset(
    "SELECT * FROM Print_media WHERE product_id = 3106", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

'Open the Bfile for reading:
OraAdGraphic.Open
OraMyAdGraphic.Open

If OraAdGraphic.Compare(OraMyAdGraphic) Then
  'Process the data
Else
  'Do error processing
End If
OraDb.Connection.CommitTrans
```

## Java (JDBC): Comparing All or Parts of Two BFILES

```
// Comparing all or parts of two BFILES. [Example script: 4019.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media WHERE product_id = 3106");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
            }

            rset = stmt.executeQuery (
                "SELECT BFILENAME('AD_GRAPHIC', 'keyboard_3106') FROM DUAL");
            if (rset.next())
            {
                lob_loc2 = ((OracleResultSet)rset).getBFILE (1);
            }
        }
    }
}
```

```
    }

    lob_loc1.openFile ();
    lob_loc2.openFile ();

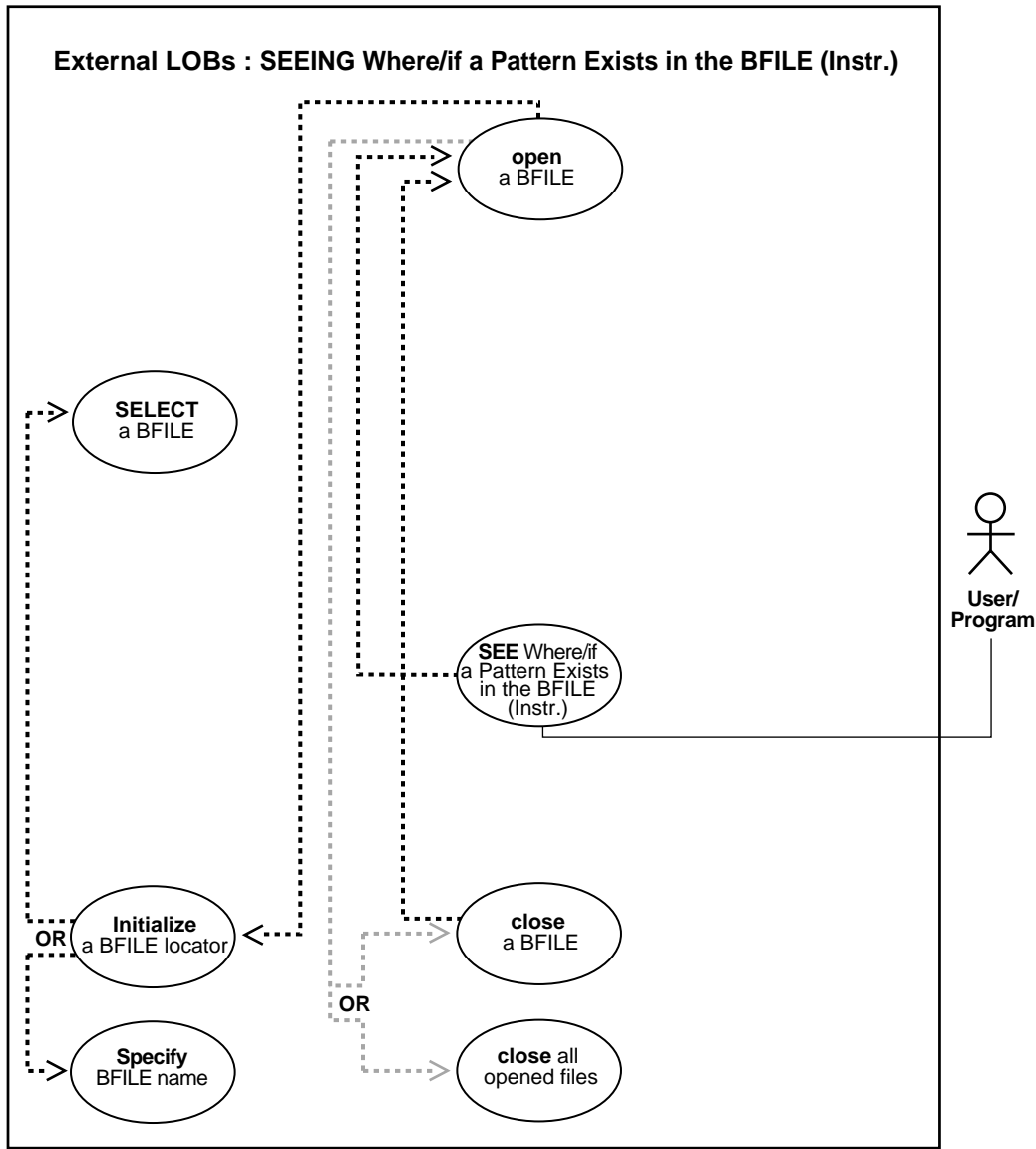
    if (lob_loc1.length() > lob_loc2.length())
        System.out.println("Looking for LOB2 inside LOB1.  result = " +
            lob_loc1.position(lob_loc2, 1));
    else
        System.out.println("Looking for LOB1 inside LOB2.  result = " +
            lob_loc2.position(lob_loc1, 1));

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Checking If a Pattern Exists (instr) in the BFILE

Figure 12–21 Use Case Diagram: Checking If a Pattern Exists in the BFILE



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

## Purpose

This procedure describes how to see if a pattern exists (instr) in the BFILE.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — INSTR
- C (OCI): A syntax reference is not applicable in this release.
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB OPEN. See PL/SQL DBMS\_LOB.INSTR.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB OPEN. See PL/SQL DBMS\_LOB.INSTR.
- Visual Basic (OO4O): A syntax reference is not applicable in this release.
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

The following examples search for the occurrence of a pattern in an `ad_graphic` image.

## Examples

These examples are provided in the following four programmatic environments:



- [PL/SQL \(DBMS\\_LOB Package\): Checking If a Pattern Exists \(instr\) in the BFILE on page 12-139](#)
- C (OCI): No example is provided with this release.
- [COBOL \(Pro\\*COBOL\): Checking If a Pattern Exists \(instr\) in the BFILE on page 12-139](#)
- [C/C++ \(Pro\\*C/C++\): Checking If a Pattern Exists \(instr\) in the BFILE on page 12-141](#)
- Visual Basic (OO4O): No example is provided with this release.
- [Java \(JDBC\): Checking If a Pattern Exists \(instr\) in the BFILE on page 12-143](#)

## PL/SQL (DBMS\_LOB Package): Checking If a Pattern Exists (instr) in the BFILE

```

/* Checking if a pattern exists in a BFILE using instr [Example script:
4030.sql]
/* Procedure compareBFILES_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE compareBFILES_proc IS
  /* Initialize the BFILE locator: */
  File_loc1      BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_
11001');
  File_loc2      BFILE;
  Retval         INTEGER;
BEGIN
  /* Select the LOB: */
  SELECT ad_graphic INTO File_loc2 FROM print_media
    WHERE Product_ID = 3060 AND ad_id = 11001;
  /* Open the BFILES: */
  DBMS_LOB.OPEN(File_loc1, DBMS_LOB.LOB_READONLY);
  DBMS_LOB.OPEN(File_loc2, DBMS_LOB.LOB_READONLY);
  Retval := DBMS_LOB.COMPARE(File_loc2, File_loc1, DBMS_LOB.LOBMAXSIZE, 1, 1);
  /* Close the BFILES: */
  DBMS_LOB.CLOSE(File_loc1);
  DBMS_LOB.CLOSE(File_loc2);
END;

```

## COBOL (Pro\*COBOL): Checking If a Pattern Exists (instr) in the BFILE

```

* Checking if a pattern exists in a BFILE using instr [Example script:
4021.pco]
  IDENTIFICATION DIVISION.

```

## Checking If a Pattern Exists (instr) in the BFILE

---

```
PROGRAM-ID. BFILE-INSTR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 AD_GRAPHIC INTO :BFILE1
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
    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 ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Checking If a Pattern Exists (instr) in the BFILE

/\* Checking if a pattern exists in a BFILE using instr [Example script: 4022.pc]  
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 instrinBFILE_proc()
{
    OCIBFileLocator *Lob_loc;
    unsigned int Position = 0;
    int Product_id = 2056, 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.ad_graphic \
          FROM TABLE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab \
                       WHERE product_id = :cid) PMtab \
          WHERE PMtab.Segment = :seg';
    EXEC SQL DECLARE C CURSOR FOR S;
    EXEC SQL OPEN C USING :Product_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 occurrence 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;
}

```

```

instr(BFILE_proc());
EXEC SQL ROLLBACK WORK RELEASE;
}

```

## Java (JDBC): Checking If a Pattern Exists (instr) in the BFILE

// Checking if a pattern exists in a BFILE using instr [Example script:  
4024.java]

```

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    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));

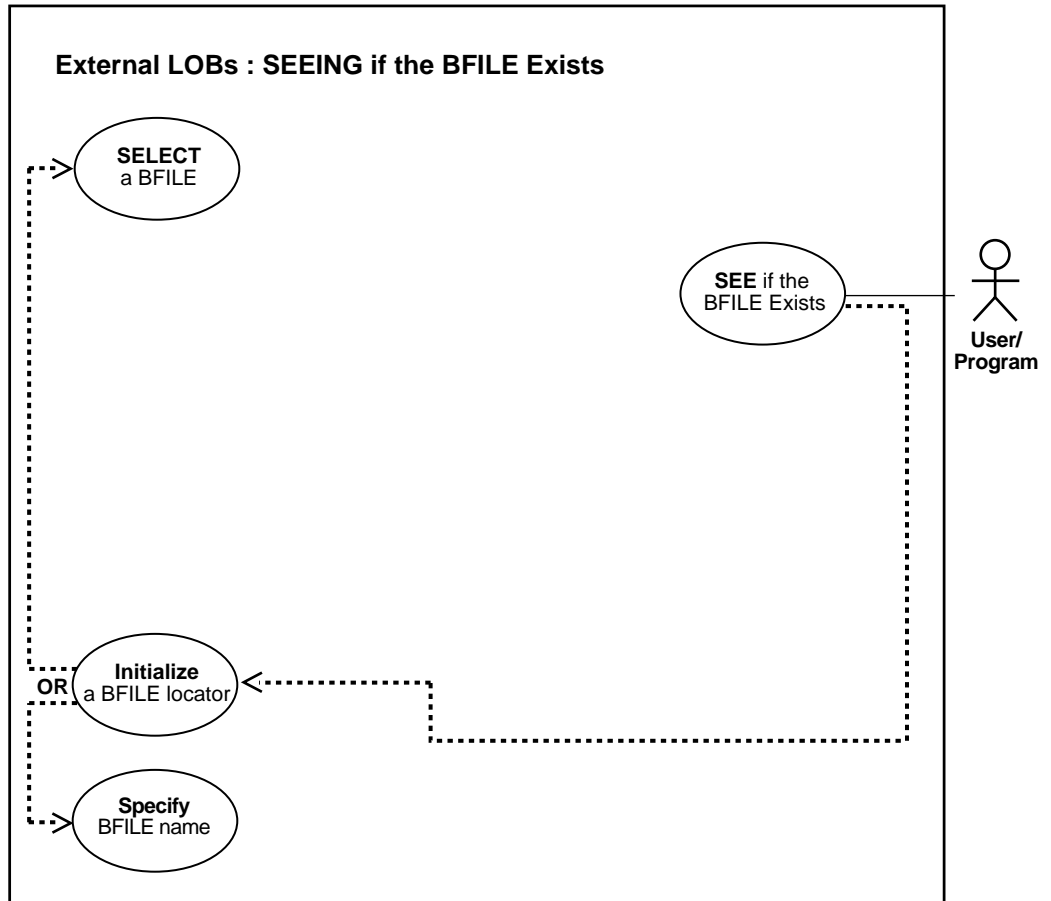
    // Close the LOB:
    lob_loc.closeFile();

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Checking If the BFILE Exists

Figure 12–22 Use Case Diagram: Checking If the BFILE Exists



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to see if a BFILE exists.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **PL/SQL (DBMS\_LOB)** *Oracle9i Supplied PL/SQL Packages and Types Reference*: Chapter 18, "DBMS\_LOB" — FILEEXISTS
- **C (OCI)** *Oracle Call Interface Programmer's Guide*: Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileExists
- **COBOL (Pro\*COBOL)** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE ... FILEEXISTS.
- **C/C++ (Pro\*C/C++)** *Pro\*C/C++ Precompiler Programmer's Guide*: "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB DESCRIBE ...GET FILEEXISTS
- **Visual Basic (OO4O)** (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > Exists, and OO4O Automation Server > OBJECTS > OraDatabase > PROPERTIES > Parameters. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- **Java (JDBC)** *Oracle9i JDBC Developer's Guide and Reference* : Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in *Oracle9i SQLJ Developer's Guide and Reference*: Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

This example queries whether a BFILE that is associated with Recording.

## Examples

The examples are provided in the following six programmatic environments:

- **PL/SQL (DBMS\_LOB Package)**: [Checking If the BFILE Exists](#) on page 12-147
- **C (OCI)**: [Checking If the BFILE Exists](#) on page 12-147



- [COBOL \(Pro\\*COBOL\): Checking If the BFILE Exists](#) on page 12-149
- [C/C++ \(Pro\\*C/C++\): Checking If the BFILE Exists](#) on page 12-150
- [Visual Basic \(OO4O\): Checking If the BFILE Exists](#) on page 12-151
- [Java \(JDBC\): Checking If the BFILE Exists](#) on page 12-152

## PL/SQL (DBMS\_LOB Package): Checking If the BFILE Exists

```

/* Checking if a BFILE exists [Example script: 4025.sql] */
/* Procedure seeIfExistsBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE seeIfExistsBFILE_proc IS
    File_loc      BFILE;
BEGIN
    /* Select the LOB: */
    SELECT Intab.ad_graphic INTO File_loc
        FROM THE(SELECT PMtab.textdoc_ntab FROM Print_media PMtab
            WHERE PMtab.product_id = 3060 AND ad_id = 11001) PMtab
            WHERE PMtab.Segment = 1;
    /* See If the BFILE exists: */
    IF (DBMS_LOB.FILEEXISTS(File_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;

```

## C (OCI): Checking If the BFILE Exists

```

/* Checking if a BFILE exists [Example script: 4026.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{

```

```

OCIDefine *dfnbp, *dfnbp2;
text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                        WHERE product_id = 3106 AND ad_id = 13001";

/* 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(stmthp, &dfnbp, errhp, 1,
                                (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                                (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                OCI_DEFAULT)
|| OCIDefineByPos(stmthp, &dfnbp2, errhp, 2,
                   (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 BfileExists(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{

    OCILobLocator *bfile_loc;
    boolean is_exist;

    /* Allocate the locator descriptor */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* Select the bfile */
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr (errhp, OCILobFileExists(svchp, errhp, bfile_loc, &is_exist));

    if (is_exist == TRUE)

```

```

    {
        printf("File exists\n");
    }
    else
    {
        printf("File does not exist\n");
    }

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

## COBOL (Pro\*COBOL): Checking If the BFILE Exists

```

* Checking if a BFILE exists. [Example script: 4027.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-EXISTS.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 AD_GRAPHIC INTO :BFILE1

```

```
        FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
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 ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.

SQL-ERROR.
EXEC SQL WHENEVER SQLEERROR CONTINUE END-EXEC.
MOVE ORASLNR TO ORASLNDR.
DISPLAY " ".
DISPLAY "ORACLE ERROR DETECTED ON LINE ", ORASLNDR, ":".
DISPLAY " ".
DISPLAY SQLERRMC.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

## C/C++ (Pro\*C/C++): Checking If the BFILE Exists

```
/* Checking if a BFILE exists. [Example script: 4028.pc] */

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

void Sample_Error()
{
    EXEC SQL WHENEVER SQLEERROR 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 PMtab.ad_graphic INTO :Lob_loc
        FROM Print_media PMtab WHERE PMtab.Product_ID = 2056 AND PMtab.ad_id
= 12001;
    /* 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;
}

```

## Visual Basic (OO4O): Checking If the BFILE Exists

```

'Checking if a BFILE exists. [Example script: 4030.txt]
'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 OraAdGraphic As OraBfile, OraSql As OraSqlStmnt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

```

```
OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
  OraDb.CreateSql(
    "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE
      product_id = :id;
      END;", ORASQL_FAILEXEC)

Set OraAdGraphic = OraParameters("MyAdGraphic").Value

If OraAdGraphic.Exists Then
  'Process the data
Else
  'Do error processing
End If
OraDb.Connection.CommitTrans
```

## Java (JDBC): Checking If the BFILE Exists

```
// Checking if a BFILE exists. [Example script: 4031.java]

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:
    DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
            WHERE product_id = 3106 AND ad_id = 13001");
        if (rset.next())
        {
            lob_loc = ((OracleResultSet)rset).getBFILE (1);
        }

        // See if the BFILE exists:
        System.out.println("Result from fileExists(): " + lob_loc.fileExists());

        // Return the length of the BFILE:
        long length = lob_loc.length();
        System.out.println("Length of BFILE: " + length);

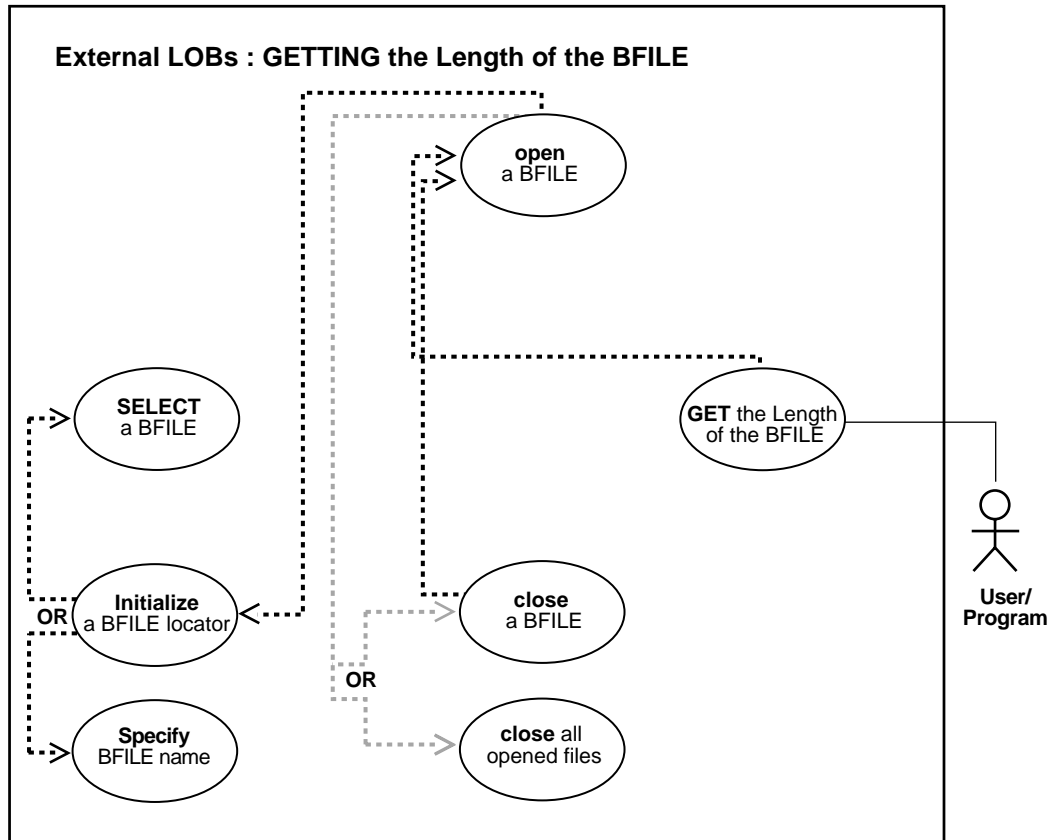
        // Get the directory alias for this BFILE:
        System.out.println("Directory alias: " + lob_loc.getDirAlias());

        // 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();  
}  
}
```

## Getting the Length of a BFILE

Figure 12-23 Use Case Diagram: Getting the Length of a BFILE





**See Also:** [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

## Purpose

This procedure describes how to get the length of a BFILE.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — GETLENGTH
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations". Chapter 15, "Relational Functions" — LOB Functions, OCILobGetLength
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE ... GET LENGTH INTO ...
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB DESCRIBE ... GET LENGTH INTO ...
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > Size. See also OO4O Automation Server > OBJECTS > Orafile > Examples
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

This example gets the length of a BFILE that is associated with `ad_graphic`.

## Examples

The examples are provided in six programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Getting the Length of a BFILE on page 12-154](#)
- [C \(OCI\): Getting the Length of a BFILE on page 12-156](#)
- [COBOL \(Pro\\*COBOL\): Getting the Length of a BFILE on page 12-158](#)
- [C/C++ \(Pro\\*C/C++\): Getting the Length of a BFILE on page 12-159](#)
- [Visual Basic \(OO4O\): Getting the Length of a BFILE on page 12-160](#)
- [Java \(JDBC\): Getting the Length of a BFILE on page 12-161](#)

## PL/SQL (DBMS\_LOB Package): Getting the Length of a BFILE

```
/* Getting the length of a BFILE. [Example script: 4032.sql]
/* Procedure getLengthBFILE_proc is not part of DBMS_LOB package: */
```

```
CREATE OR REPLACE PROCEDURE getLengthBFILE_proc IS
    File_loc      BFILE;
    Length        INTEGER;
BEGIN
    /* Initialize the BFILE locator by selecting the LOB: */
    SELECT PMtab.ad_graphic INTO File_loc FROM Print_media PMtab
        WHERE PMtab.product_id = 3060 AND PMtab.ad_id = 11001;
    /* Open the BFILE: */
    DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
    /* Get the length of the LOB: */
    Length := DBMS_LOB.GETLENGTH(File_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(File_loc);
END;
```

## C (OCI): Getting the Length of a BFILE

```
/* Getting the length of a BFILE. [Example script: 4033.c] */
/* Select the lob/bfile from table Print_media */
```

```

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                               WHERE product_id = 3106 AND ad_id = 13001";

    /* Prepare the SQL select statement */
    checkerr (errhp, OCISstmtPrepare(stmthp, errhp, selstmt,
                                     (ub4) strlen((char *) selstmt),
                                     (ub4) OCI_NTIV_SYNTAX, (ub4)OCI_DEFAULT));

    /* Call define for the bfile column */
    checkerr (errhp, OCIDefineByPos(stmthp, &dfnhp, errhp, 1,
                                     (dvoid *)&Lob_loc, 0 , SOLT_BFILE,
                                     (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                     OCI_DEFAULT)
    || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                       (dvoid *)&Lob_loc, 0 , SOLT_BFILE,
                       (dvoid *)0, (ub2 *)0, (ub2 *)0,
                       OCI_DEFAULT));

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

void BfileLength(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCISstmt    *stmthp;
{
    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(bfile_loc, errhp, svchp, stmthp);

checkerr (errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                               (ub1) OCI_FILE_READONLY));

checkerr (errhp, OCILobGetLength(svchp, errhp, bfile_loc, &len));

printf("Length of bfile = %d\n", len);

checkerr (errhp, OCILobFileClose(svchp, errhp, bfile_loc));

/* Free the locator descriptor */
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}
```

## COBOL (Pro\*COBOL): Getting the Length of a BFILE

```
* Getting the length of a BFILE. [Example script: 4034.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-LENGTH.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 AD_GRAPHIC INTO :BFILE1
    FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106
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.

```

## C/C++ (Pro\*C/C++): Getting the Length of a BFILE

```

/* Getting the length of a BFILE. [Example script: 4035.pc] */

#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 PMtab.ad_graphic INTO :Lob_loc
        FROM Print_media PMtab
        WHERE PMtab.product_id = 3060 AND ad_id = 11001;
    /* 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 uninitialized, 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;
}
```

## Visual Basic (OO4O): Getting the Length of a BFILE

```
'Getting the length of a BFILE. [Example script: 4037.txt]
'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("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans
```

```

Set OraParameters = OraDb.Parameters

OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "AdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
  OraDb.CreateSql(
    "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media WHERE product_
id = :id;
    END;", ORASQL_FAILEXEC)

Set OraAdGraphic = OraParameters("MyAdGraphic").Value

If OraAdGraphic.Size = 0 Then
  MsgBox "BFile size is 0"
Else
  MsgBox "BFile size is " & OraAdGraphic.Size
End If
OraDb.Connection.CommitTrans

```

## Java (JDBC): Getting the Length of a BFILE

```

// Getting the length of a BFILE. [Example script: 4038.java]

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:
    DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
             WHERE product_id = 3106 AND ad_id = 13001");
        if (rset.next())
        {
            lob_loc = ((OracleResultSet)rset).getBFILE (1);
        }

        // See if the BFILE exists:
        System.out.println("Result from fileExists(): " + lob_loc.fileExists());

        // Return the length of the BFILE:
        long length = lob_loc.length();
        System.out.println("Length of BFILE: " + 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());

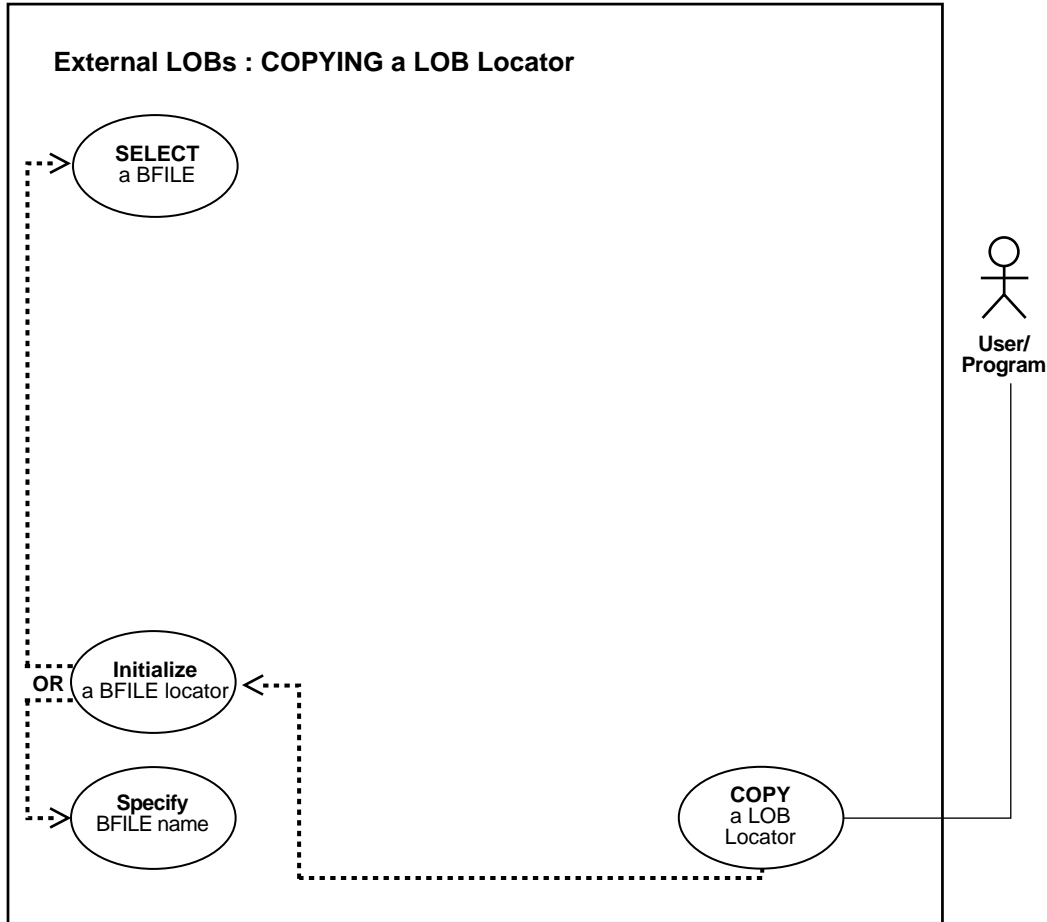
        stmt.close();
        conn.commit();
    }
}
```



```
        conn.close();
    }
    catch (SQLException e)
    {
        e.printStackTrace();
    }
}
```

## Copying a LOB Locator for a BFILE

Figure 12–24 Use Case Diagram: Copying a LOB Locator for a BFILE



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to copy a LOB locator for a BFILE.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- **SQL** (*Oracle9i SQL Reference*): Chapter 7, "SQL Statements" — CREATE PROCEDURE
- **PL/SQL** (DBMS\_LOB package): Refer to [Chapter 5, "Large Objects: Advanced Topics"](#) of this manual for information on assigning one lob locator to another.
- **C (OCI)** (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobLocatorAssign
- **COBOL (Pro\*COBOL)** *Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB ASSIGN
- **C/C++ (Pro\*C/C++)** (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB ASSIGN
- **Visual Basic (OO4O)**: A syntax reference is not applicable in this release.
- **Java (JDBC)** *Oracle9i JDBC Developer's Guide and Reference* : Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

This example assigns one BFILE locator to another related to `ad_graphic`.

## Examples

The examples are provided in the following five programmatic environments:

- **PL/SQL**: [Copying a LOB Locator for a BFILE](#) on page 12-159
- **C (OCI)**: [Copying a LOB Locator for a BFILE](#) on page 12-166
- **COBOL (Pro\*COBOL)**: [Copying a LOB Locator for a BFILE](#) on page 12-168

- [C/C++ \(Pro\\*C/C++\): Copying a LOB Locator for a BFILE](#) on page 12-169
- Visual Basic: An example is not provided with this release.
- [Java \(JDBC\): Copying a LOB Locator for a BFILE](#) on page 12-170

## PL/SQL: Copying a LOB Locator for a BFILE

---

---

**Note:** Assigning one BFILE to another using PL/SQL entails using the "=" sign. This is discussed in more detail with regard to ["Read Consistent Locators"](#) in Chapter 5, ["Large Objects: Advanced Topics"](#).

---

---

```
/* Copying a LOB locator for a BFILE. [Example script: 4039.sql] */
/* Procedure BFILEAssign_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE BFILEAssign_proc IS
    File_loc1    BFILE;
    File_loc2    BFILE;
BEGIN
    SELECT Photo INTO File_loc1 FROM print_media
        WHERE Product_ID = 3060 AND ad_id = 11001 FOR UPDATE;
    /* Assign File_loc1 to File_loc2 so that they both refer to the same
operating
        system file: */
    File_loc2 := File_loc1;
    /* Now you can read the bfile from either File_loc1 or File_loc2. */
END;
```

## C (OCI): Copying a LOB Locator for a BFILE

```
/* Copying a LOB locator for a BFILE. [Example script: 4040.c] */
/* Select the lob/bfile from the Print_media table */

void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
```

```

WHERE product_id = 3106 AND ad_id = 13001";

/* 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(stmthp, &dfnhp, errhp, 1,
                               (dvoid *)&Lob_loc, 0 , SQLT_BFILE,
                               (dvoid *)0, (ub2 *)0, (ub2 *)0,
                               OCI_DEFAULT)
|| OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                  (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 BfileAssign(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    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(src_loc, errhp, svchp, stmthp);
}

```

```
checkerr(errhp, OCILobLocatorAssign(svchp, errhp, src_loc, &dest_loc));

/* Free the locator descriptor */
OCIDescriptorFree((dvoid *)src_loc, (ub4)OCI_DTYPE_FILE);
OCIDescriptorFree((dvoid *)dest_loc, (ub4)OCI_DTYPE_FILE);
}
```

## COBOL (Pro\*COBOL): Copying a LOB Locator for a BFILE

```
* Copying a LOB locator for a BFILE. [Example script: 4041.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-COPY-LOCATOR.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 SQLEERROR 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 AD_GRAPHIC INTO :BFILE1
    FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106
    AND AD_ID = 13001 END-EXEC.
EXEC SQLLOB 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 SQLERRM.
EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
STOP RUN.
```

## C/C++ (Pro\*C/C++): Copying a LOB Locator for a BFILE

```
/* Copying a LOB locator for a BFILE. [Example script: 4042.pc] */

#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 ad_graphic INTO :Lob_loc1
        FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;
}
```

## Java (JDBC): Copying a LOB Locator for a BFILE

```
// Copying a LOB locator for a BFILE. [Example script: 4044.java]
```

```
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:
        DriverManager.registerDriver(new 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;
```



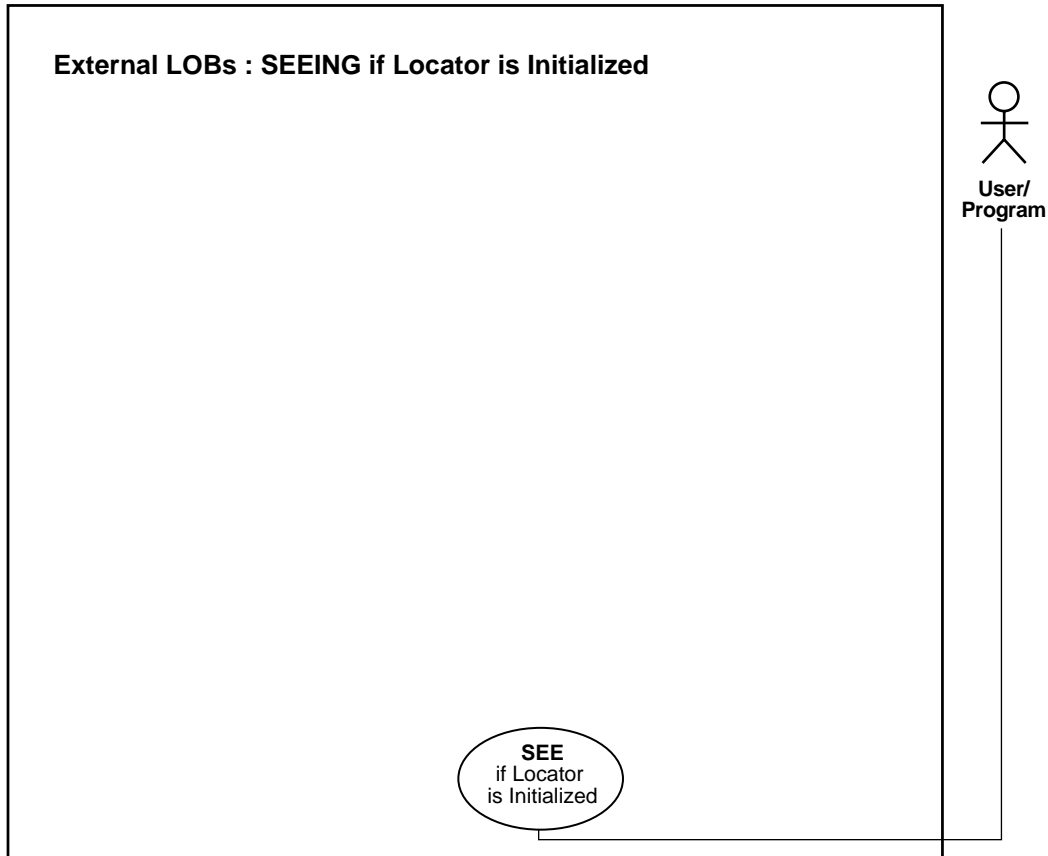
```
        BFILE lob_loc2 = null;

        ResultSet rset = stmt.executeQuery (
            "SELECT ad_graphic FROM Print_media
             WHERE product_id = 3106 AND ad_id = 13001");
        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)
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
```

## Determining If a LOB Locator for a BFILE Is Initialized

*Figure 12–25 Use Case Diagram: Determining If a LOB Locator Is Initialized*



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### **Purpose**

This procedure describes how to determine if a BFILE LOB locator is initialized.

## Usage Notes

On the client side, before you call any OCILob\* interfaces (such as OCILobWrite), or any programmatic environments that use OCILob\* interfaces, first initialize the LOB locator, using a SELECT, for example.

If your application requires a locator to be passed from one function to another, you may want to verify that the locator has already been initialized. If 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.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): A syntax reference is not applicable in this release.
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobLocatorIsInit
- COBOL (Pro\*COBOL): A syntax reference is not applicable in this release.
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives". See also C(OCI) function, OCILobLocatorIsInit
- Visual Basic (OO4O): A syntax reference is not applicable in this release
- Java (JDBC): A syntax reference is not applicable in this release.

## Scenario

Not applicable.

## Examples

The examples are provided in the following programmatic environments:

- PL/SQL (DBMS\_LOB): No example is provided with this release.
- C (OCI): [Determining If a LOB Locator for a BFILE Is Initialized](#) on page 12-179
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): [Determining If a LOB Locator for a BFILE Is Initialized](#) on page 12-175

- Visual Basic (OO4O): No example is provided with this release.
- Java (JDBC): No example is provided with this release.

## C (OCI): Determining If a LOB Locator for a BFILE Is Initialized

```

/* Determining if a LOB locator for a BFILE is initialized.
   [Example script: 4045.c]*/

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILobLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCISstmt      *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
                               WHERE product_id = 3106 AND ad_id = 13001";

    /* 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(stmthp, &dfnhp, errhp, 1,
                                    (dvoid *)&Lob_loc, 0, SOLT_BFILE,
                                    (dvoid *)0, (ub2 *)0, (ub2 *)0,
                                    OCI_DEFAULT)
    || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                       (dvoid *)&Lob_loc, 0, SOLT_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 BfileIsInit(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;

```

```

OCIStmt      *stmthp;
{

    OCILobLocator *bfile_loc;
    boolean is_init;

    /* Allocate the locator descriptors */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* Select the bfile */
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobLocatorIsInit(envhp, errhp, bfile_loc, &is_init));

    if (is_init == TRUE)
    {
        printf("Locator is initialized\n");
    }
    else
    {
        printf("Locator is not initialized\n");
    }
    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

## C/C++ (Pro\*C/C++): Determining If a LOB Locator for a BFILE Is Initialized

```

/* Determining if a LOB locator for a BFILE is initialized.
   [Example script: 4046.pc]
   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 with the EXEC SQL ALLOCATE statement. However, an example
   can be written that uses embedded SQL and the OCI as 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;

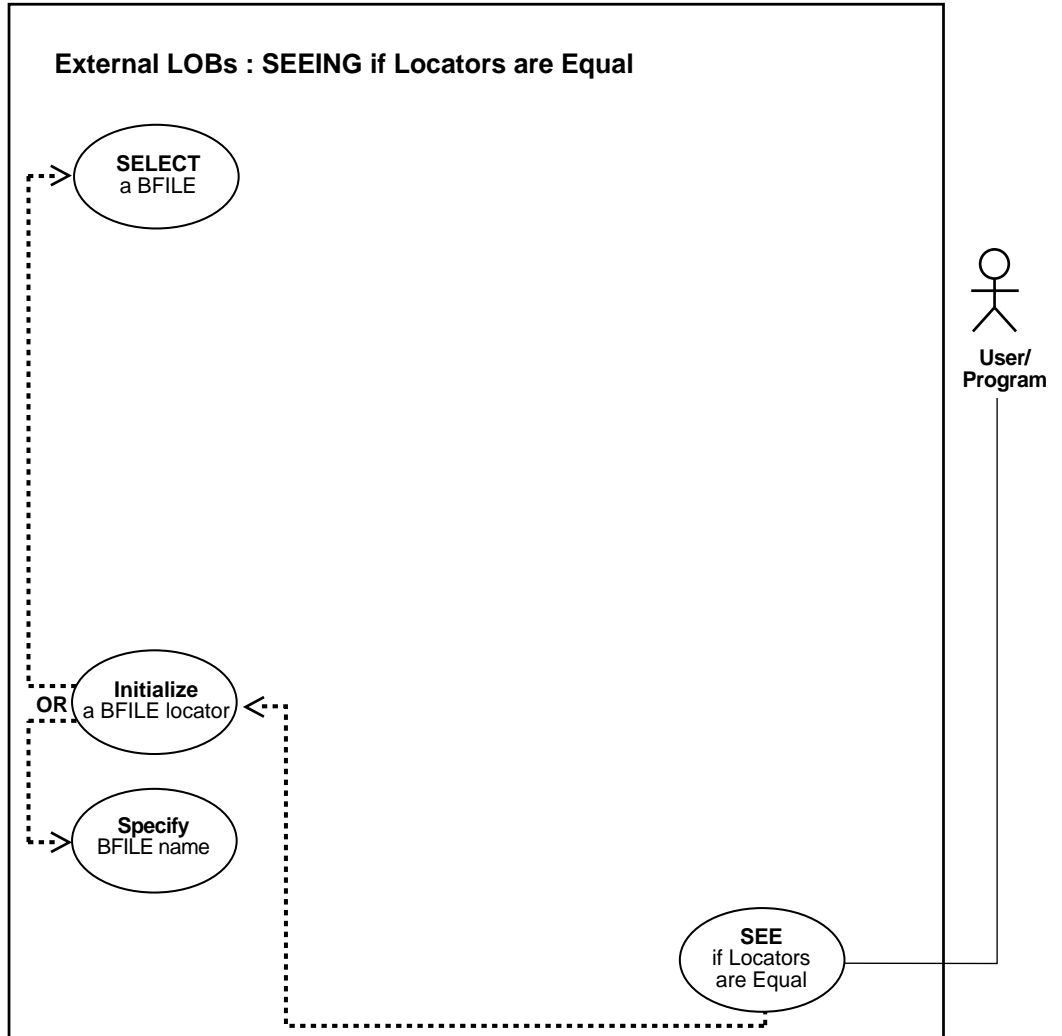
    EXEC SQL WHENEVER SQLERROR DO Sample_Error();
    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT Mtab.ad_graphic INTO :Lob_loc
        FROM Print_media PMtab
        WHERE PMtab.product_id = 2056 AND ad_id = 12001;
    /* Get the OCI Environment Handle using a SQLLIB Routine: */
    (void) SQLEnvGet(SQL_SINGLE_RCTX, &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;
}

```

## Determining If One LOB Locator for a BFILE Is Equal to Another

Figure 12–26 Use Case Diagram: Determining If One LOB Locator for a BFILE Is Equal to Another



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to see if one BFILE LOB locator is equal to another.

### Usage Notes

Not applicable.

### Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): A syntax reference is not applicable in this release.
- (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobIsEqual
- COBOL (Pro\*COBOL): A syntax reference is not applicable in this release
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB ASSIGN. See also C(OCI) function, OCILobIsEqual
- Visual Basic (OO4O): A syntax reference is not applicable in this release
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference* : Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

If two locators are equal, this means that they refer to the same version of the LOB data (see ["Read Consistent Locators"](#) in [Chapter 5, "Large Objects: Advanced Topics"](#)).

### Examples

The examples are provided in the following three programmatic environments:



- PL/SQL: An example is not provided with this release.
- [C \(OCI\): Determining If One LOB Locator for a BFILE Is Equal to Another](#) on page 12-174
- COBOL (Pro\*COBOL): An example is not provided with this release.
- [C/C++ \(Pro\\*C/C++\): Determining If One BFILE LOB Locator Is Equal to Another](#) on page 12-181
- Visual Basic (OO4O): An example is not provided with this release.
- [Java \(JDBC\): Determining If One LOB Locator for a BFILE Is Equal to Another](#) on page 12-182

## C (OCI): Determining If One LOB Locator for a BFILE Is Equal to Another

```

/* Determining if one LOB locator for a BFILE is equal to another */
/* [Example script: 4047.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError *errhp;
OCISvcCtx *svchp;
OCIStmt *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
        WHERE product_id = 3106 AND ad_id = 13001";

    /* 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(stmthp, &dfnhp, errhp, 1,
        (dvoid *)&Lob_loc, 0, SOLT_BFILE,
        (dvoid *)0, (ub2 *)0, (ub2 *)0,
        OCI_DEFAULT)
    || OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
        (dvoid *)&Lob_loc, 0, SOLT_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 BfileIsEqual(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{

    OCILobLocator *bfile_loc1;
    OCILobLocator *bfile_loc2;
    boolean is_equal;

    /* Allocate the locator descriptors */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* Select the bfile */
    selectLob(bfile_loc1, errhp, svchp, stmthp);

    checkerr(errhp,
             OCILobLocatorAssign(svchp, errhp, bfile_loc1, &bfile_loc2));

    checkerr(errhp, OCILobIsEqual(envhp, bfile_loc1, bfile_loc2, &is_equal));

    if (is_equal == TRUE)
    {
        printf("Locators are equal\n");
    }
    else
    {
        printf("Locators are not equal\n");
    }

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
}

```

```

    OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}

```

## C/C++ (Pro\*C/C++): Determining If One BFILE LOB Locator Is Equal to Another

```

/* Determining if one LOB locator for a BFILE is equal to another */
[Example script: 4048.pc]
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 ad_graphic INTO :Lob_loc1
        FROM Print_media WHERE product_id = 2056 AND ad_id = 12001;
    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;
}
```

## Java (JDBC): Determining If One LOB Locator for a BFILE Is Equal to Another

```
// Determining if one LOB locator for a BFILE is equal to another
// [Example script: 4050.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
        WHERE product_id = 3106 AND ad_id = 13001");
    if (rset.next())
    {
        lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
    }

    // Set both LOBS to reference the same BFILE:
    lob_loc2 = lob_loc1;

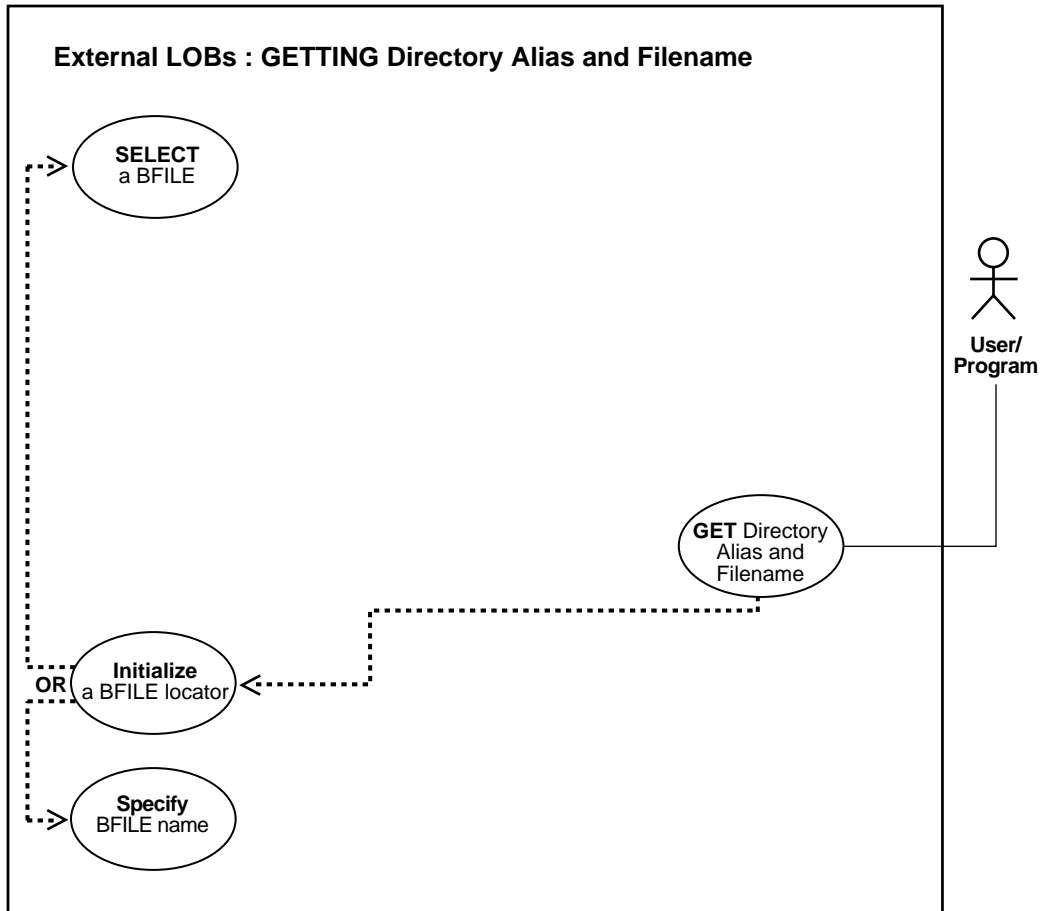
    // Note that in this example, the Locators will be equal:
    if (lob_loc1.equals(lob_loc2))
    {
        // The Locators are equal:
        System.out.println("The BFILES are equal");
    }
    else
    {
        // The Locators are different:
        System.out.println("The BFILES are NOT equal");
    }

    stmt.close();
    conn.commit();
    conn.close();

}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Getting DIRECTORY Alias and Filename

Figure 12-27 Use Case Diagram: Get DIRECTORY Alias and Filename



**See Also:** [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to get DIRECTORY alias and filename.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — FILEGETNAME
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileName
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB DESCRIBE ...GET DIRECTORY ...
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB DESCRIBE ...GET DIRECTORY ...
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

This example retrieves the DIRECTORY alias and filename related to the BFILE, `ad_graphic`.

## Examples

The examples are provided in the following six programmatic environments:

- [PL/SQL \(DBMS\\_LOB Package\): Getting Directory Alias and Filename on page 12-181](#)
- [C \(OCI\): Getting Directory Alias and Filename on page 12-186](#)
- [COBOL \(Pro\\*COBOL\): Getting Directory Alias and Filename on page 12-188](#)
- [C/C++ \(Pro\\*C/C++\): Getting Directory Alias and Filename on page 12-189](#)

- [Visual Basic \(OO4O\): Getting Directory Alias and Filename](#) on page 12-190
- [Java \(JDBC\): Getting Directory Alias and Filename](#) on page 12-191

## PL/SQL (DBMS\_LOB Package): Getting Directory Alias and Filename

```
/* Getting the directory alias and filename [Example script: 4051.sql] */

CREATE OR REPLACE PROCEDURE getNameBFILE_proc IS
    File_loc          BFILE;
    DirAlias_name     VARCHAR2(30);
    File_name         VARCHAR2(40);
BEGIN
    SELECT ad_graphic INTO File_loc FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;
    DBMS_LOB.FILEGETNAME(File_loc, DirAlias_name, File_name);
    /* do some processing based on the directory alias and file names */
END;
```

## C (OCI): Getting Directory Alias and Filename

```
/* Getting the directory alias and filename [Example script: 4052.c] */

/* Select the lob/bfile from the Print_media table */
void selectLob(Lob_loc, errhp, svchp, stmthp)
OCILOBLocator *Lob_loc;
OCIError      *errhp;
OCISvcCtx     *svchp;
OCIStmt       *stmthp;
{
    OCIDefine *dfnhp, *dfnhp2;
    text *selstmt = (text *) "SELECT ad_graphic FROM Print_media \
        WHERE product_id = 3106";

    /* 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(stmthp, &dfnhp, errhp, 1,
        (dvoid *)&Lob_loc, 0, SOLT_BFILE,
        (dvoid *)0, (ub2 *)0, (ub2 *)0,
        OCI_DEFAULT));
}
```



```

|| OCIDefineByPos(stmthp, &dfnhp2, errhp, 2,
                    (dvoid *)&lob_loc, 0, SOLT_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 OCI_Snapshot*) 0, (OCI_Snapshot*) 0,
                                (ub4) OCI_DEFAULT));
}

void BfileGetDirFile(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *bfile_loc;
    OraText dir_alias[32] = NULL;
    OraText filename[256] = NULL;
    ub2 d_length = 32;
    ub2 f_length = 256;

    /* Allocate the locator descriptors */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    /* Select the bfile */
    selectLob(bfile_loc, errhp, svchp, stmthp);

    checkerr(errhp, OCILobFileGetName(envhp, errhp, bfile_loc,
                                     dir_alias, &d_length, filename, &f_length));

    printf("Directory Alias : [%s]\n", dir_alias);
    printf("File name : [%s]\n", filename);

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

## COBOL (Pro\*COBOL): Getting Directory Alias and Filename

```

d      * Getting the directory alias and filename [Example script: 4053.pco]
      IDENTIFICATION DIVISION.
      PROGRAM-ID. BFILE-DIR-ALIAS.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 AD_GRAPHIC INTO :BFILE1
          FROM PRINT_MEDIA WHERE PRODUCT_ID = 3106 AND AD_ID = 13001
      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.

```

## C/C++ (Pro\*C/C++): Getting Directory Alias and Filename

```

/* Getting the directory alias and filename [Example script: 4054.pc] */

#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 ad_graphic INTO :Lob_loc
        FROM print_media WHERE product_id = 2056 AND ad_id = 12001;
    /* 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;
}
```

## Visual Basic (OO4O): Getting Directory Alias and Filename

'Getting the directory alias and filename [Example script: 4056.txt]  
'The PL/SQL packages and tables mentioned here are not part of the  
'standard OO4O installation:

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraAdGraphic1 As OraBfile, OraSql As OraSqlStmt

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)
OraDb.Connection.BeginTrans
Set OraParameters = OraDb.Parameters
OraParameters.Add "id", 2056, ORAPARM_INPUT

'Define out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_OUTPUT
OraParameters("MyAdGraphic").ServerType = ORATYPE_BFILE

Set OraSql =
    OraDb.CreateSql(
        "BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media
        WHERE product_id = :id;
        END;", ORASQL_FAILEXEC)
```

```

Set OraAdGraphic1 = OraParameters("MyAdGraphic").Value
'Get Directory alias and filename:
MsgBox " Directory alias is " & OraAdGraphic1.DirectoryName &
      " Filename is " & OraAdGraphic1.filename

OraDb.Connection.CommitTrans

```

## Java (JDBC): Getting Directory Alias and Filename

```

// Getting the directory alias and filename [Example script: 4057.java]

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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
      WHERE product_id = 3106 AND ad_id = 13001");
if (rset.next())
{
    lob_loc = ((OracleResultSet)rset).getBFILE (1);
}
// See if the BFILE exists:
System.out.println("Result from fileExists(): " + lob_loc.fileExists());

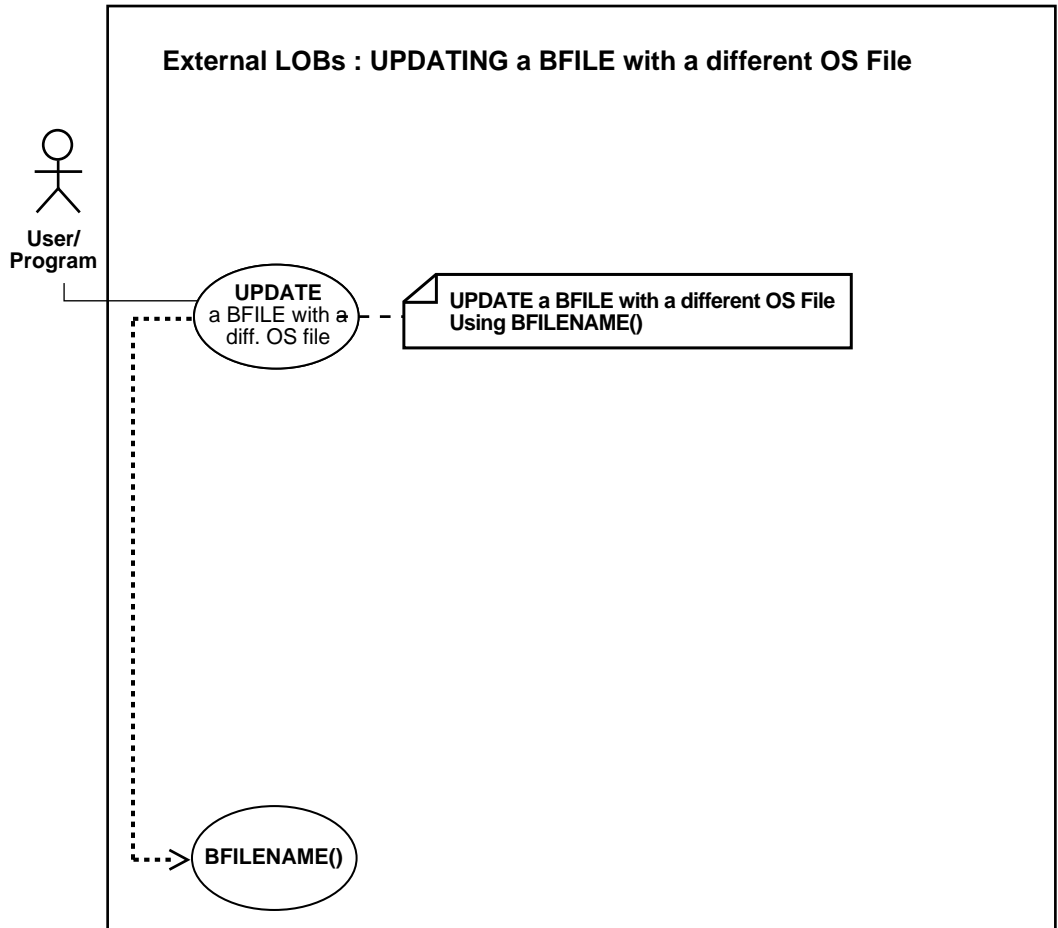
// Return the length of the BFILE:
long length = lob_loc.length();
System.out.println("Length of BFILE: " + length);

// 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();
}
}
```

## Updating a BFILE Using BFILENAME()

Figure 12–28 Use Case Diagram: Updating a BFILE Using BFILENAME()



**See Also:**

- [Table 12-1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Updating a BFILE by Selecting a BFILE From Another Table](#) on page 12-196
- [Updating a BFILE by Initializing a BFILE Locator](#) on page 12-198

## Usage Notes

**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 file system.

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 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.

## Syntax

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



**See Also:** ["DIRECTORY Name Specification"](#) on page 12-7 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.

Use the following syntax references:

- [SQL \(Oracle9i SQL Reference\):Chapter 7, "SQL Statements" — UPDATE. Chapter 4, "Functions" — BFILENAME\(\)](#)

### Scenario

This example updates the `Print_media` table by means of the `BFILENAME` function.

### Examples

The example is provided in SQL syntax and applies to all programmatic environments:

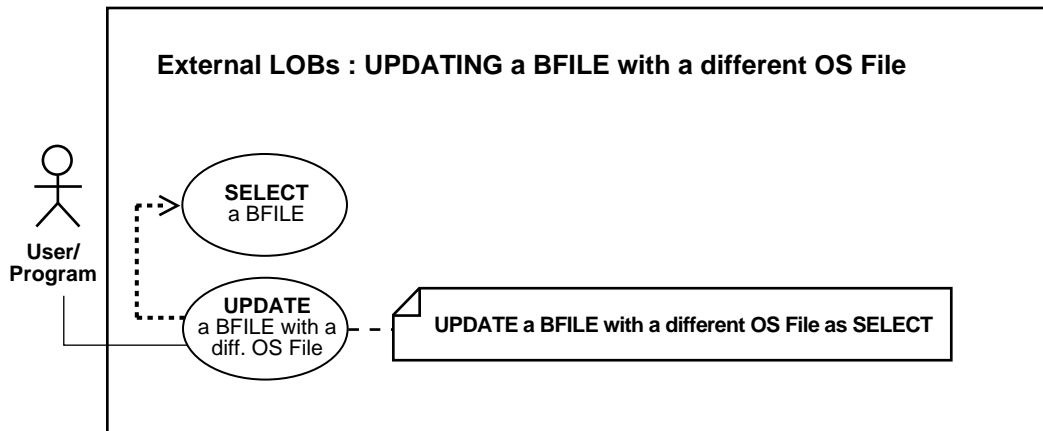
- [SQL: Updating a BFILE by means of BFILENAME\(\)](#) on page 12-195

## SQL: Updating a BFILE by means of BFILENAME()

```
/* Updating a BFILE using BFILENAME() [Example script: 4059.sql] */  
  
UPDATE Print_media  
SET ad_graphic = BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001')  
WHERE product_id = 3106 AND ad_id = 13001;
```

## Updating a BFILE by Selecting a BFILE From Another Table

*Figure 12–29 Use Case Diagram: Updating a BFILE by Selecting a BFILE From Another Table*



### See Also:

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Updating a BFILE Using BFILENAME\(\)](#) on page 12-193
- [Updating a BFILE by Initializing a BFILE Locator](#) on page 12-198

### Purpose

This procedure describes how to UPDATE a BFILE by selecting a BFILE from another table.

### Usage Notes

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.

## Syntax

Use the following syntax references:

- *SQL (Oracle9i SQL Reference)*, Chapter 7, "SQL Statements" — UPDATE

## Scenario

This example updates the table, `Voiceover_tab` by selecting from the archival storage table, `VoiceoverLib_tab`.

## Examples

The example is provided in SQL and applies to all programmatic environments:

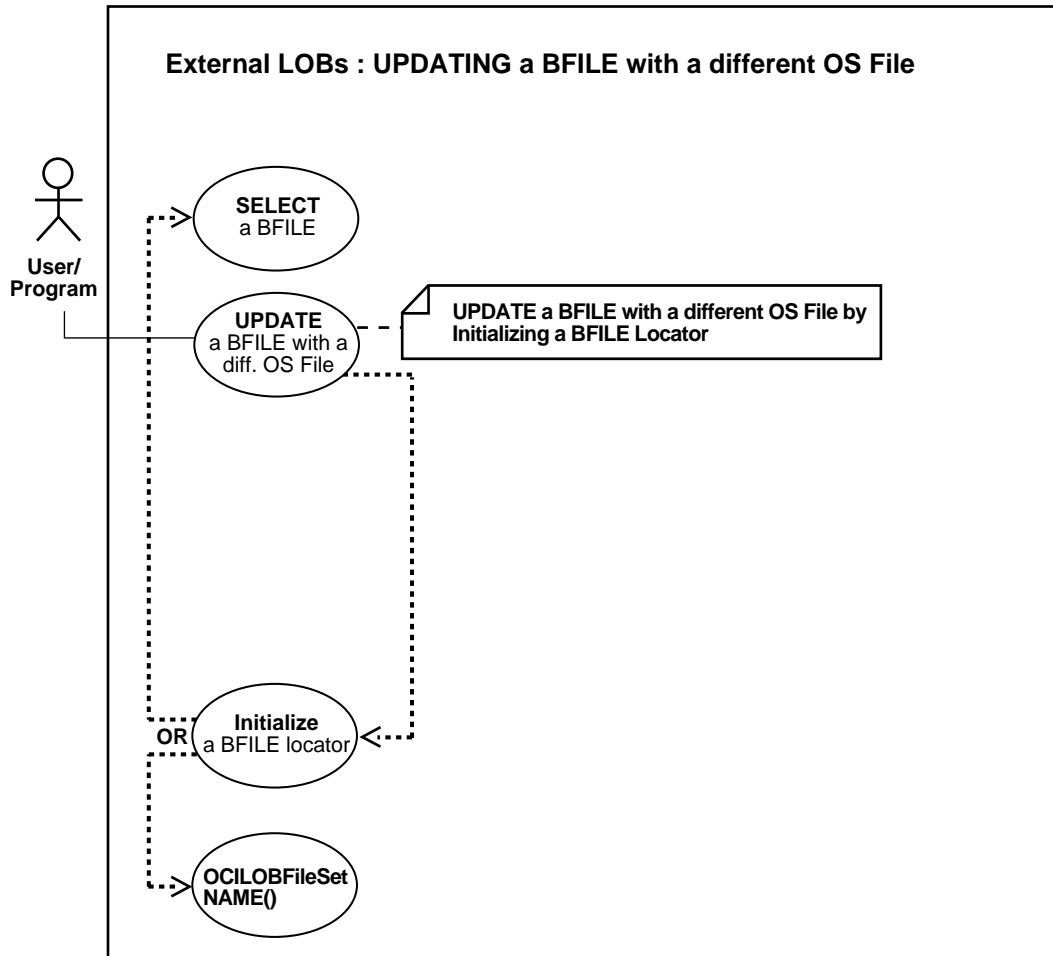
- [SQL: Updating a BFILE by Selecting a BFILE From Another Table](#) on page 12-197

## SQL: Updating a BFILE by Selecting a BFILE From Another Table

```
UPDATE Adheader_tab
SET (header_name, creation_date, header_text, logo) =
(SELECT * FROM AdheaderLib_tab AHTab
WHERE AHTab.creation_date = '08/08/2001');
```

## Updating a BFILE by Initializing a BFILE Locator

Figure 12-30 Use Case Diagram: Updating a BFILE by Initializing a BFILE Locator



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Updating a BFILE Using BFILENAME\(\)](#) on page 12-193
- [Updating a BFILE by Selecting a BFILE From Another Table](#) on page 12-196

**Purpose**

This procedure describes how to UPDATE a BFILE by initializing a BFILE locator.

**Usage Notes**

You must initialize the BFILE locator bind variable to a directory alias and filename *before* issuing the update statement.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB): See the (*Oracle9i SQL Reference*), Chapter 7, "SQL Statements" — UPDATE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileName
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — ALLOCATE. See also (*Oracle9i SQL Reference*), Chapter 7, "SQL Statements" — UPDATE
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives". See also (*Oracle9i SQL Reference*), Chapter 7, "SQL Statements" — UPDATE
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > DirectoryName, FileName, and OO4O Automation Server > OBJECTS > OraDatabase > METHODS > ExecuteSQL. See also OO4O Automation Server > OBJECTS > OraBfile > Examples

- Java (JDBC) *Oracle9i JDBC Developer's Guide and Reference*: Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in ( *Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

Not applicable.

### Examples

The examples are provided in six programmatic environments:

- [PL/SQL: Updating a BFILE by Initializing a BFILE Locator](#) on page 12-198
- [C \(OCI\): Updating a BFILE by Initializing a BFILE Locator](#) on page 12-201
- [COBOL \(Pro\\*COBOL\): Updating a BFILE by Initializing a BFILE Locator](#) on page 12-202
- [C/C++ \(Pro\\*C/C++\): Updating a BFILE by Initializing a BFILE Locator](#) on page 12-203
- [Visual Basic \(OO4O\): Updating a BFILE by Initializing a BFILE Locator](#) on page 12-204
- [Java \(JDBC\): Updating a BFILE by Initializing a BFILE Locator](#) on page 12-206

## PL/SQL: Updating a BFILE by Initializing a BFILE Locator

```
/* Updating a BFILE by initializing a BFILE locator. [Example script:4061.sql]
   Procedure updateUseBindVariable_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE updateUseBindVariable_proc (File_loc BFILE) IS
BEGIN
    UPDATE Print_media SET ad_graphic = File_loc
        WHERE product_id = 3060 AND ad_id = 11001;
END;

DECLARE
    File_loc BFILE;
BEGIN
    SELECT ad_graphic INTO File_loc
        FROM Print_media
        WHERE product_id = 3060 AND ad_id = 11001;
    updateUseBindVariable_proc (File_loc);
```

```

COMMIT;
END;

```

## C (OCI): Updating a BFILE by Initializing a BFILE Locator

```

/* Updating a BFILE by initializing a BFILE locator. [Example script: 4062.c] */
void BfileUpdate(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *Lob_loc;
    OCIBind *bndhp, *bndhp2;

    text *updstmt =
        (text *) "UPDATE Print_media SET ad_graphic = :Lob_loc
                WHERE product_id = 3106 AND ad_id = 13001";

    OraText *Dir = (OraText *) "ADGRAPHIC_DIR",
              *Name = (OraText *) "keyboard_graphic_3106_13001";

    /* 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)
              || OCIBindByPos(stmthp, &bndhp2, errhp, (ub4) 2,
                              (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, OCISntmExecute(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);
}
```

## COBOL (Pro\*COBOL): Updating a BFILE by Initializing a BFILE Locator

\* Updating a BFILE by initializing a BFILE locator. [Example script:  
4063.pco]

```
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-UPDATE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID          PIC X(11) VALUES "SAMP/SAMP".
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 AD_GRAPHIC INTO :BFILE1:BFILE-IND
FROM PRINT_MEDIA WHERE PRODUCT_ID = 3060
AND AD_ID = 13001 END-EXEC.

* Make graphic associated with product_id=3106 same as product_id=3060
* and ad_id = 13001:
EXEC SQL
    UPDATE PRINT_MEDIA SET AD_GRAPHIC = :BFILE1:BFILE-IND
    WHERE PRODUCT_ID = 3106 AND AD_ID = 13001 END-EXEC.

* Free the BFILE:
END-OF-BFILE.
EXEC SQL WHENEVER NOT FOUND CONTINUE END-EXEC.
EXEC SQL FREE :BFILE1 END-EXEC.
EXEC SQL ROLLBACK 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.

```

## C/C++ (Pro\*C/C++): Updating a BFILE by Initializing a BFILE Locator

```

/* Updating a BFILE by initializing a BFILE locator. [Example script: 4064.pc]
*/

#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 Print_media SET ad_graphic = :Lob_loc
        WHERE product_ID = 2056 AND ad_id = 12001;
}

void updateBFILE_proc()
{
    OCIBFileLocator *Lob_loc;

    EXEC SQL ALLOCATE :Lob_loc;
    EXEC SQL SELECT ad_graphic INTO :Lob_loc
        FROM Print_media WHERE product_id = 2056 AND ad_id 12001;
    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;
}
```

## Visual Basic (OO40): Updating a BFILE by Initializing a BFILE Locator

'Updating a BFILE by initializing a BFILE locator. [Example script:4066.txt]

```
Dim MySession As OraSession
Dim OraDb As OraDatabase
Dim OraParameters As OraParameters, OraAdGraphic As OraBfile

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

OraDb.Connection.BeginTrans

Set OraParameters = OraDb.Parameters

'Define in out parameter of BFILE type:
OraParameters.Add "MyAdGraphic", Null, ORAPARM_BOTH, ORATYPE_BFILE
```

```
'Define out parameter of BFILE type:
OraDb.ExecuteSQL (
"BEGIN SELECT ad_graphic INTO :MyAdGraphic FROM Print_media
      WHERE product_id = 2056 AND ad_id = 12001;
      END;")

'Update the ad_graphic BFile for product_id=2056 AND ad_id = 12001
      to product_id=2268 AND ad_id = 21001:
OraDb.ExecuteSQL (
      "UPDATE Print_media SET ad_graphic = :MyAdGraphic
      WHERE product_id = 2268 AND ad_id = 21001")

'Get Directory alias and filename
'MsgBox " Directory alias is " & OraAdGraphic1.DirectoryName & " Filename is " &
OraAdGraphic1.filename

OraDb.Connection.CommitTrans
```

## Java (JDBC): Updating a BFILE by Initializing a BFILE Locator

```
// Updating a BFILE by initializing a BFILE locator. [Example script: 4067.java]

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:
        DriverManager.registerDriver(new 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;
            OraclePreparedStatement pstmt = null;

            rset = stmt.executeQuery (
                "SELECT ad_graphic FROM Print_media
```

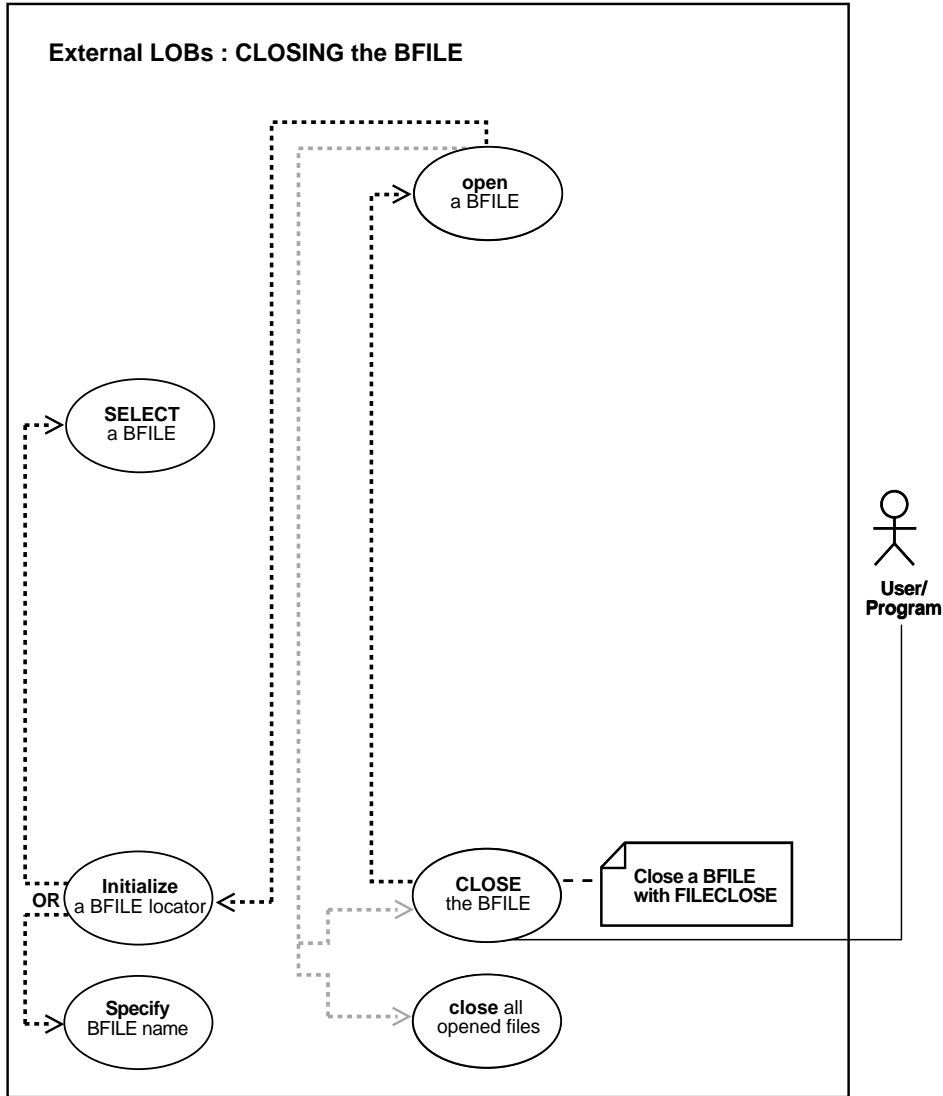
```
        WHERE product_id = 3106 AND ad_id = 13001");
    if (rset.next())
    {
        src_lob = ((OracleResultSet)rset).getBFILE (1);
    }

    // Prepare a CallableStatement to OPEN the LOB for READWRITE:
    pstmt = (OraclePreparedStatement) conn.prepareStatement (
        "UPDATE Print_media SET ad_graphic = ?
        WHERE product_id = 3060 AND ad_id = 11001");
    pstmt.setBFILE(1, src_lob);
    pstmt.execute();

    //Close the statements and commit the transaction:
    stmt.close();
    pstmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Closing a BFILE with FILECLOSE

Figure 12–31 Use Case Diagram: Closing a BFILE with FILECLOSE



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Closing a BFILE with CLOSE](#) on page 12-214

**Purpose**

This procedure describes how to close a BFILE with FILECLOSE.

**Usage Notes**

Although closing a BFILE with FILECLOSE is still supported, we strongly recommend that you use CLOSE instead. Doing so facilitates future extensibility.

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — FILEOPEN, FILECLOSE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileClose
- COBOL (Pro\*COBOL): A syntax reference is not applicable in this release.
- C/C++ (Pro\*C/C++): A syntax reference is not applicable in this release.
- Visual Basic (OO4O): A syntax reference is not applicable in this release.
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference*): Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in *Oracle9i SQLJ Developer's Guide and Reference*: Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

**Scenario**

This example closes a BFILE in ADPHOTO\_DIR.

## Examples

- [PL/SQL \(DBMS\\_LOB Package\): Closing a BFile with FILECLOSE](#) on page 12-208
- [C \(OCI\): Closing a BFile with FILECLOSE](#) on page 12-208
- COBOL (Pro\*COBOL): No example is provided with this release.
- C/C++ (Pro\*C/C++): No example is provided with this release.
- Visual Basic (OO4O): No example is provided. See note on page 12-211.
- [Java \(JDBC\): Closing a BFile with FILECLOSE](#) on page 12-211

## PL/SQL (DBMS\_LOB Package): Closing a BFile with FILECLOSE

```
/* Closing a BFILE with FILECLOSE. [Example script: 4068.sql]
   Procedure closeBFILE_procOne is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procOne IS
    File_loc BFILE := BFILENAME('ADPHOTO_DIR', 'keyboard_photo_3060_11001');
BEGIN
    DBMS_LOB.FILEOPEN(File_loc, DBMS_LOB.FILE_READONLY);
    /* ...Do some processing. */
    DBMS_LOB.FILECLOSE(File_loc);
END;
```

## C (OCI): Closing a BFile with FILECLOSE

```
/* Closing a BFILE with FILECLOSE. [Example script: 4069.c] */

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

    OCILobLocator *bfile_loc;

    /* Allocate the locator descriptors */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);
```



```

    checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                                     (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_
DIR"),
                                     (OraText *) "keyboard_graphic_3106_13001",
                                     (ub2) strlen("keyboard_graphic_3106_13001")));

    checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc,
                                   (ub1) OCI_FILE_READONLY));

    checkerr(errhp, OCILobFileClose(svchp, errhp, bfile_loc));

    /* Free the locator descriptor */
    OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

## Visual Basic (OO4O): Closing a BFile with FILECLOSE

---



---

**Note:** At the present time, OO4O only offers BFILE closing with CLOSE.

---



---

## Java (JDBC): Closing a BFile with FILECLOSE

```

// Closing a BFILE with FILECLOSE. [Example script: 4071.java]

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:
        DriverManager.registerDriver(new 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 = false;

            rset = stmt.executeQuery (
                "SELECT BFILENAME('ADGRAPHIC_DIR','keyboard_graphic_3106_11001')
                 FROM DUAL");
            if (rset.next())
            {
                src_lob = ((OracleResultSet)rset).getBFILE (1);
            }

            result = src_lob.isFileOpen();
            System.out.println(
                "result of fileIsOpen() before opening file : " + result);

            src_lob.openFile();

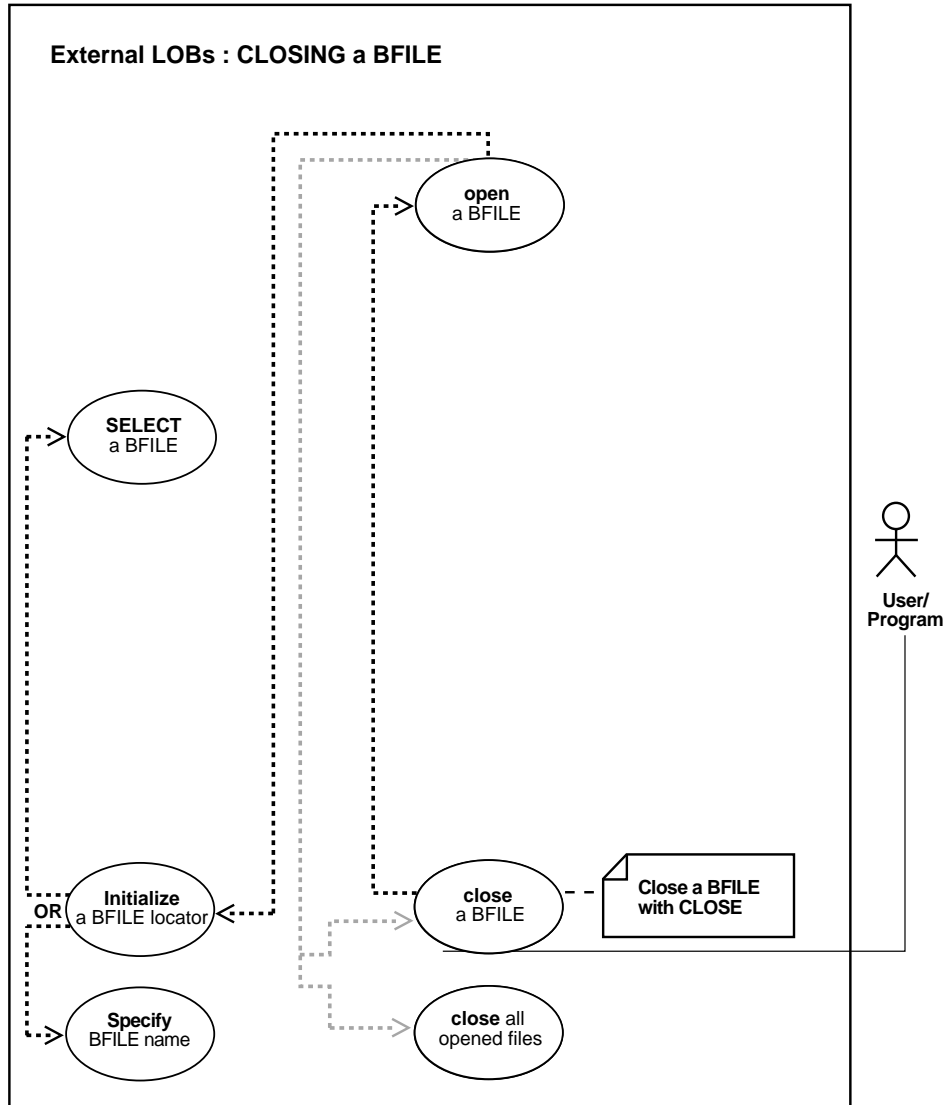
            result = src_lob.isFileOpen();
            System.out.println(
                "result of fileIsOpen() after opening file : " + result);

            // Close the BFILE, statement and connection:
            src_lob.closeFile();
            stmt.close();
            conn.commit();
            conn.close();
        }
    }
}
```

```
    }  
    catch (SQLException e)  
    {  
        e.printStackTrace();  
    }  
}
```

## Closing a BFILE with CLOSE

Figure 12-32 Use Case Diagram: Closing an Open BFILE with CLOSE



**See Also:**

- [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2
- [Closing a BFILE with FILECLOSE](#) on page 12-208

**Purpose**

This procedure describes how to close a BFILE with CLOSE.

**Usage Notes**

Use CLOSE in conjunction with OPEN.

**See:** [Opening a BFILE with OPEN](#) on page 12-70

**Syntax**

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — CLOSE
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobClose
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB CLOSE
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB CLOSE
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > PROPERTIES > IsOpen. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples.
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference* : Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*):

## Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

### Scenario

These examples close a BFILE in `ADGRAPHIC_DIR`.

### Examples

- [PL/SQL \(DBMS\\_LOB Package\): Closing a Bfile with CLOSE](#) on page 12-216
- [C \(OCI\): Closing a Bfile with CLOSE](#) on page 12-216
- [COBOL \(Pro\\*COBOL\): Closing a BFILE with CLOSE](#) on page 12-217
- [C/C++ \(Pro\\*C/C++\): Closing a Bfile with CLOSE](#) on page 12-218
- [Visual Basic \(OO4O\): Closing a Bfile with CLOSE](#) on page 12-219
- [Java \(JDBC\): Closing a Bfile with CLOSE](#) on page 12-220

### PL/SQL (DBMS\_LOB Package): Closing a Bfile with CLOSE

```
/* Closing a BFILE with CLOSE. [Example script: 4072.sql]
   Procedure closeBFILE_procTwo is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeBFILE_procTwo IS
    File_loc BFILE := BFILENAME('ADGRAPHIC_DIR','keyboard_graphic_3060_11001');
BEGIN
    DBMS_LOB.OPEN(File_loc, DBMS_LOB.LOB_READONLY);
    /* ...Do some processing. */
    DBMS_LOB.CLOSE(File_loc);
END;
```

### C (OCI): Closing a Bfile with CLOSE

```
/* Closing a BFILE with CLOSE. [Example script: 4073.c] */

void BfileClose(envhp, errhp, svchp, stmthp)
OCIEnv      *envhp;
OCIError    *errhp;
OCISvcCtx   *svchp;
OCIStmt     *stmthp;
{
    OCILobLocator *bfile_loc;
```

```

/* Allocate the locator descriptors */
(void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc,
                        (ub4) OCI_DTYPE_FILE,
                        (size_t) 0, (dvoid **) 0);

checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc,
                                (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                                (OraText *) "keyboard_3106",
                                (ub2) strlen("keyboard_3106")));

checkerr(errhp, OCILobOpen(svchp, errhp, bfile_loc,
                          (ub1) OCI_LOB_READONLY));

checkerr(errhp, OCILobClose(svchp, errhp, bfile_loc));

/* Free the locator descriptor */
OCIDescriptorFree((dvoid *)bfile_loc, (ub4)OCI_DTYPE_FILE);
}

```

## COBOL (Pro\*COBOL): Closing a BFILE with CLOSE

```

* Closing a BFILE with CLOSE. [Example script: 4074.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-CLOSE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 "ADGRAPHIC_DIR" TO DIR-ALIAS-ARR.
    MOVE 9 TO DIR-ALIAS-LEN.
    MOVE "keyboard_graphic_3106_13001" 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.
```

## C/C++ (Pro\*C/C++): Closing a BFile with CLOSE

```
/* Closing a BFILE with CLOSE. [Example script: 4075.pc]
Pro*C/C++ has only one form of CLOSE for BFILEs. Pro*C/C++ has no
FILECLOSE 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 = "ADGRAPHIC_DIR", *Name = "mousepad_graphic_2056_12001";

    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;
}

```

## Visual Basic (OO4O): Closing a BFile with CLOSE

'Closing a BFILE with CLOSE. [Example script: 4076.txt

```

Dim MySession As OraSession
Dim OraDb As OraDatabase

```

```

Dim OraDyn As OraDynaset, OraAdGraphic As OraBfile, amount_read%, chunksize%,
chunk

```

```

Set MySession = CreateObject("OracleInProcServer.XOraSession")
Set OraDb = MySession.OpenDatabase("pmschema", "pm/pm", 0&)

```

```

chunksize = 32767

```

```
Set OraDyn = OraDb.CreateDynaset("select * from Print_media", ORADYN_DEFAULT)
Set OraAdGraphic = OraDyn.Fields("ad_graphic").Value

If OraAdGraphic.IsOpen Then
    'Process because the file is already open
    OraAdGraphic.Close
End If
```

## Java (JDBC): Closing a BFile with CLOSE

```
// Closing a BFILE with CLOSE. [Example script: 4077.java]

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:
        DriverManager.registerDriver(new 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
```

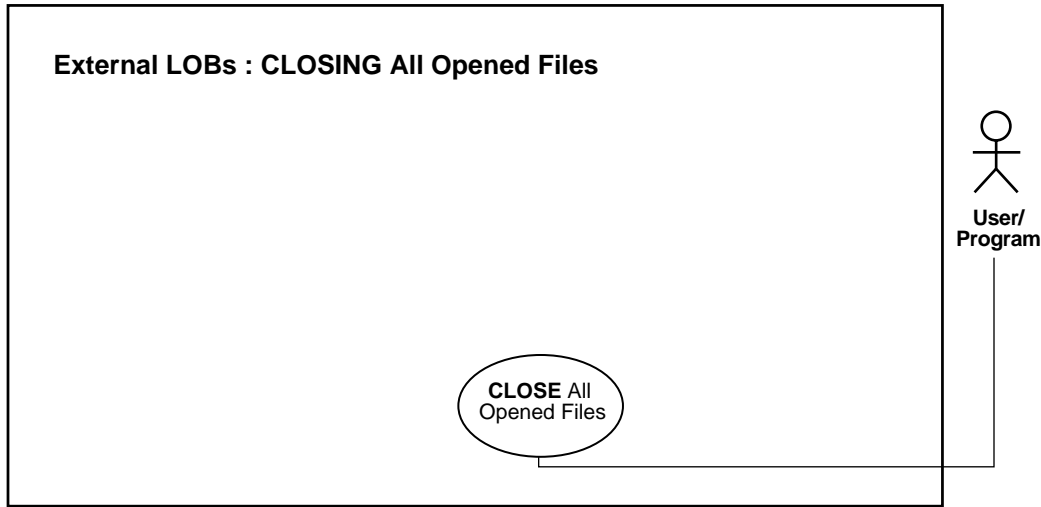
```
{
    BFILE src_lob = null;
    ResultSet rset = null;

    rset = stmt.executeQuery (
        "SELECT BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001') FROM
DUAL");
    OracleCallableStatement cstmt = null;
    if (rset.next())
    {
        src_lob = ((OracleResultSet)rset).getBFILE (1);
        cstmt = (OracleCallableStatement)conn.prepareCall
            ("begin dbms_lob.open (?,dbms_lob.lob_readonly); end;");
        cstmt.registerOutParameter(1,OracleTypes.BFILE);
        cstmt.setBFILE (1, src_lob);
        cstmt.execute();
        src_lob = cstmt.getBFILE(1);
        System.out.println ("the file is now open");
    }

    // Close the BFILE, statement and connection:
    cstmt = (OracleCallableStatement)
        conn.prepareCall ("begin dbms_lob.close(?); end;");
    cstmt.setBFILE(1,src_lob);
    cstmt.execute();
    stmt.close();
    conn.commit();
    conn.close();
}
catch (SQLException e)
{
    e.printStackTrace();
}
}
```

## Closing All Open BFILEs with FILECLOSEALL

**Figure 12–33 Use Case Diagram: Closing All Open BFILEs**



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILEs\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

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 that have raised errors. The exception handler should make provision to close any files that were opened before the occurrence of the exception or abnormal termination.

If this is not done, Oracle considers these files unclosed.

**See Also:** ["Specify the Maximum Number of Open BFILEs: SESSION\\_MAX\\_OPEN\\_FILES"](#) on page 12-64

### Purpose

This procedure describes how to close all BFILEs.

## Usage Notes

Not applicable.

## Syntax

See [Chapter 3, "LOB Support in Different Programmatic Environments"](#) for a list of available functions in each programmatic environment. Use the following syntax references for each programmatic environment:

- PL/SQL (DBMS\_LOB) (*Oracle9i Supplied PL/SQL Packages and Types Reference*): Chapter 18, "DBMS\_LOB" — FILECLOSEALL
- C (OCI) (*Oracle Call Interface Programmer's Guide*): Chapter 7, "LOB and File Operations" for usage notes. Chapter 15, "Relational Functions" — LOB Functions, OCILobFileCloseAll
- COBOL (Pro\*COBOL) (*Pro\*COBOL Precompiler Programmer's Guide* for information on LOBs, usage notes on LOB Statements, and embedded SQL and precompiler directives — LOB FILE CLOSE ALL
- C/C++ (Pro\*C/C++) (*Pro\*C/C++ Precompiler Programmer's Guide*): "Large Objects (LOBs)", "LOB Statements", "Embedded SQL Statements and Directives" — LOB FILE CLOSE ALL
- Visual Basic (OO4O) (Oracle Objects for OLE (OO4O) Online Help): From Help Topics, Contents tab, select OO4O Automation Server > OBJECTS > OraBFILE > METHODS > CloseAll. See also OO4O Automation Server > OBJECTS > OraBFILE > Examples
- Java (JDBC) (*Oracle9i JDBC Developer's Guide and Reference* : Chapter 7, "Working With LOBs" — Creating and Populating a BLOB or CLOB Column. Further extensions are available in (*Oracle9i SQLJ Developer's Guide and Reference*): Chapter 5, "Type Support", Oracle Type Support, Support for BLOB, CLOB, and BFILE.

## Scenario

Not applicable.

## Examples

- [PL/SQL \(DBMS\\_LOB Package\): Closing All Open BFiles](#) on page 12-224
- [C \(OCI\): Closing All Open BFiles](#) on page 12-224
- [COBOL \(Pro\\*COBOL\): Closing All Open BFiles](#) on page 12-225

- [C/C++ \(Pro\\*C/C++\): Closing All Open BFiles on page 12-227](#)
- [Visual Basic \(OO4O\): Closing All Open BFiles on page 12-228](#)
- [Java \(JDBC\): Closing All Open BFiles with FILECLOSEALL on page 12-228](#)

## PL/SQL (DBMS\_LOB Package): Closing All Open BFiles

```
/* Closing all open BFILES. [Example script: 4078.sql]
   Procedure closeAllOpenFilesBFILE_proc is not part of DBMS_LOB package: */

CREATE OR REPLACE PROCEDURE closeAllOpenFilesBFILE_proc IS
BEGIN
    /* Close all open BFILES: */
    DBMS_LOB.FILECLOSEALL;
END;
```

## C (OCI): Closing All Open BFiles

```
/* Closing all open BFILES. [Example script: 4079.c] */

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

    OCILobLocator *bfile_loc1;
    OCILobLocator *bfile_loc2;

    /* Allocate the locator descriptors */
    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc1,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    (void) OCIDescriptorAlloc((dvoid *) envhp, (dvoid **) &bfile_loc2,
                              (ub4) OCI_DTYPE_FILE,
                              (size_t) 0, (dvoid **) 0);

    checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc1,
                                       (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
                                       (OraText *) "keyboard_graphic_3106_13001",
                                       (ub2) strlen("keyboard_graphic_3106_13001")));
}
```

```

checkerr(errhp, OCILobFileSetName(envhp, errhp, &bfile_loc2,
    (OraText *) "ADGRAPHIC_DIR", (ub2) strlen("ADGRAPHIC_DIR"),
    (OraText *) "monitor_graphic_3060_11001",
    (ub2) strlen("monitor_graphic_306_11001")));

checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc1,
    (ub1) OCI_LOB_READONLY));

checkerr(errhp, OCILobFileOpen(svchp, errhp, bfile_loc2,
    (ub1) OCI_LOB_READONLY));

checkerr(errhp, OCILobFileCloseAll(svchp, errhp));

/* Free the locator descriptor */
OCIDescriptorFree((dvoid *)bfile_loc1, (ub4)OCI_DTYPE_FILE);
OCIDescriptorFree((dvoid *)bfile_loc2, (ub4)OCI_DTYPE_FILE);
}

```

## COBOL (Pro\*COBOL): Closing All Open BFiles

```

* Closing all open BFILES. [Example script: 4080.pco]
IDENTIFICATION DIVISION.
PROGRAM-ID. BFILE-CLOSE-ALL.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01  USERID    PIC X(11) VALUES "SAMP/SAMP".
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 "ADGRAPHIC_DIR" TO DIR-ALIAS1-ARR.
MOVE 9 TO DIR-ALIAS1-LEN.
MOVE "keyboard_graphic_3106_13001" 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 "ADGRAPHIC_DIR" TO DIR-ALIAS2-ARR.
MOVE 9 TO DIR-ALIAS2-LEN.
MOVE "mousepad_graphic_2056_12001" 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.
```



## C/C++ (Pro\*C/C++): Closing All Open BFiles

```

/* Closing all open BFILES. [Example script: 4081.pc] */

#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 ad_graphic INTO :Lob_loc1
        FROM Print_media
        WHERE product_id = 2056 AND ad_id = 12001;
    EXEC SQL SELECT Mtab.ad_graphic INTO Lob_loc2
        FROM Print_media PMtab
        WHERE PMtab.product_id = 3060 AND ad_id = 11001;
    /* 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;  
}
```

### Visual Basic (OO40): Closing All Open BFiles

```
'Closing all open BFILES. [Example script: 4083.txt]  
  
Dim OraParameters as OraParameters, OraAdGraphic as OraBFile  
OraConnection.BeginTrans  
  
Set OraParameters = OraDatabase.Parameters  
  
'Define in out parameter of BFILE type:  
OraParameters.Add "MyAdGraphic", Null,ORAPARAM_BOTH,ORATYPE_BFILE  
  
'Select the ad graphic BFile for product_id 2268:  
OraDatabase.ExecutesQL("Begin SELECT ad_graphic INTO :MyAdGraphic FROM  
Print_media WHERE product_id = 2268 AND ad_id = 21001; END; " )  
  
'Get the BFile ad_graphic column:  
set OraAdGraphic = OraParameters("MyAdGraphic").Value  
  
'Open the OraAdGraphic:  
OraAdGraphic.Open  
  
'Do some processing on OraAdGraphic  
  
'Close all the BFILES associated with OraAdGraphic:  
OraAdGraphic.CloseAll
```

### Java (JDBC): Closing All Open BFiles with FILECLOSEALL

```
// Closing all open BFILES. [Example script: 4084.java]  
  
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:
        DriverManager.registerDriver(new 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 ad_graphic FROM Print_media
                 WHERE product_id = 3106 AND ad_id = 13001");
            if (rset.next())
            {
                lob_loc1 = ((OracleResultSet)rset).getBFILE (1);
            }

            rset = stmt.executeQuery (
                "SELECT BFILENAME('ADGRAPHIC_DIR', 'keyboard_graphic_3106_13001')
                 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();

        lob_loc1.openFile ();
        lob_loc2.openFile ();

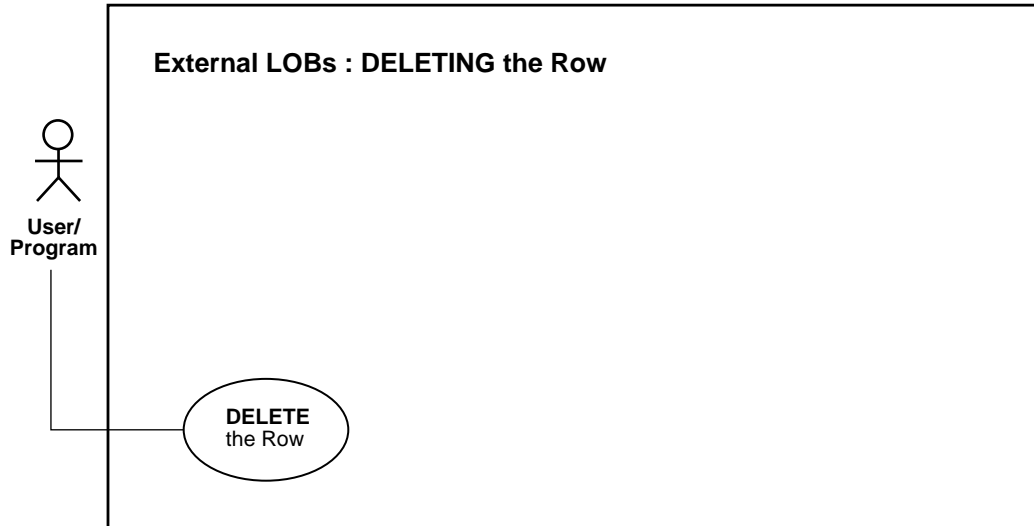
        // Compare MAXBUFSIZE bytes starting at the first byte of
        // both lob_loc1 and lob_loc2:
        cstmt = (OracleCallableStatement) conn.prepareCall (
            "BEGIN ? := DBMS_LOB.COMPARE(?, ?, ?, 1, 1); END;");
        cstmt.registerOutParameter (1, Types.NUMERIC);
        cstmt.setBFILE(2, 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();
    }
}
```

## Deleting the Row of a Table Containing a BFILE

Figure 12–34 Use Case Diagram: Deleting the Row of a Table Containing a BFILE



**See Also:** [Table 12–1, "Use Case Model: External LOBs \(BFILES\)"](#) on page 12-2, for all basic operations of Internal Temporary LOBs

### Purpose

This procedure describes how to DELETE the row of a table containing a BFILE.

### Usage Notes

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.

### Syntax

See the following syntax reference:

- *SQL (Oracle9i SQL Reference)*, Chapter 7, "SQL Statements" — DELETE, DROP, TRUNCATE

### Scenario

The following DELETE, DROP TABLE, or TRUNCATE TABLE statements delete the row, and hence the BFILE locator that refers to the advertisement graphic image for the product with `product_id` 3106 and `ad_id` 13001, but do not delete the operating system file for the graphic image.

### Examples

The following examples are provided in SQL and apply to all programmatic environments:

- ["SQL: Deleting a Row from a Table"](#)

## SQL: Deleting a Row from a Table

```
/* Deleting the row of a table containing a BFILE [Example script: 4085.sql] */  
  
DELETE FROM Print_media  
  WHERE product_id = 3106 AND ad_id = 13001;  
  
DROP TABLE Multimedia_tab;  
  
TRUNCATE TABLE Multimedia_tab;
```

---

## Using OraOLEDB to Manipulate LOBs

This chapter contains the following sections:

- [Introducing OLE DB](#)
- [Manipulating LOBs Using ADO Recordsets and OLE DB Rowsets](#)
- [Manipulating LOBs Using OraOLEDB Commands](#)
- [ADO and LOBs Example 1: Inserting LOB Data From a File](#)

## Introducing OLE DB

OLE DB is an open specification for accessing various types of data from different stores in a uniform way. It uses a set of COM interfaces for accessing and manipulating different types of data. The interfaces are available from various database providers.

OLE DB introduces the concept of a consumer and a provider. A consumer is a client application that uses or 'consumes' an OLE DB interface. A provider is a component that exposes an OLE DB interface.

A typical provider can retrieve data from a particular data store and expose the data to a consumer in tabular form.

### OraOLEDB: OLE DB and Oracle Large Object (LOB) Support

OraOLEDB is an OLE DB provider for Oracle. It offers high performance and efficient access to Oracle data including LOBs. It also allows updates to certain LOB types.

The following LOB types are supported by OraOLEDB:

- *For Persistent LOBs*. READ/WRITE through the rowset.
- *For BFILEs*. READ-ONLY through the rowset.
- *Temporary LOBs* are not supported through the rowset.

### Rowset Object

Rowset is an OLE DB object that provides READ/WRITE capability to data obtained by executing an SQL SELECT statement or a stored procedure that returns a REF Cursor.

BFILEs can be part of the rowset but they are read-only.



## Manipulating LOBs Using ADO Recordsets and OLE DB Rowsets

LOB data is never retrieved and stored in the provider cache. When a server cursor is used, OraOLEDB provides the LOB data to the consumer only when it is requested.

---

---

**Note:** Although most LOB columns in an Oracle database support up to 4 GB of data storage, ADO limits the maximum column size to 2 GB.

---

---

To incur less round trips to the database, reads and writes should be carried out in large chunks for better performance.

### Use Explicit Transactions

When using server cursor in an auto-commit mode, all LOB data modifications are transmitted to the database and committed. This means that even if the recordset is in a deferred update mode, the LOB data modifications and any previous deferred updates, will be permanent. To have flexibility of rolling back LOB data modifications, it is advised that explicit transactions are used when manipulation LOB data.

## ADO Recordsets and LOBs

### GetChunk()

The GetChunk method of ADO recordset object retrieves LOB data. When subsequent GetChunk() calls are made on the same LOB column, data is retrieved from where it left off. However, if the current row changes or if another LOB column is read from or written to, calling GetChunk() again on the original LOB column will retrieve data from the beginning.

### Writing Data to a LOB Column With AppendChunk()

The AppendChunk() method of ADO recordset object writes data to a LOB column. The initial AppendChunk() method will overwrite any existing data. Subsequent AppendChunk() calls will append the data, but the appending will end when the current row changes or when another LOB column data is updated or read from.

## OLE DB Rowsets and LOBs

The following OLE DB rowset methods read and write LOB data:

- `IRowset::GetData()` and `ISequentialStream::Read()` reads LOB data.
- `IRowsetChange::SetData()` and `ISequentialStream::Write()` writes LOB data.

## Manipulating LOBs Using OraOLEDB Commands

In OraOLEDB, the following functionality is supported:

- LOB input bind parameters, through commands
- Tables with any type of LOB, including BFiles, can be created through commands

LOB input or output parameters are supported in stored procedure executions using OraOLEDB 8.1.7 or higher. In addition, the database must be Oracle8i Release 8.1 or higher.

## ADO and LOBs Example 1: Inserting LOB Data From a File

The following is an ADO sample that demonstrates the insertion of a new row with a LOB column. A file called "c:\myfile.txt" will need to be created on your machine for this sample to work. It can be created using your favorite editor to contain any character data such as "This is only a test". This character data will then be used by the program to populate the CLOB column in the MULTIMEDIA\_TAB table.

The program then retrieves the newly inserted data from the database and validates the inserted data. The inserted row is then deleted before the program exits.

The example covers the following ADO methods that can be used for LOBs, namely:

- `GetChunk` method
- `AppendChunk` method
- `ActualSize` property

```
Sub Main()  
    Dim con As New ADODB.Connection  
    Dim cmd As New ADODB.Command  
  
    Dim rst As New ADODB.Recordset
```

```
Dim LogFileName As String
Dim LogFileNum As Integer

Dim sql As String      ' SELECT statement
Dim clob_data As Variant ' data from a text file
Dim vardata As Variant  ' data retrieved from clob data in chunks
Dim vardata_len As Long ' length of the data retrieved from the CLOB
column
Dim done As Boolean     ' done = True if finished retrieving all the data
Dim Data As Variant     ' the entire data retrieved from the CLOB column

On Error GoTo ErrorHandler

' open a text file
LogFileName = "c:\myfile.txt"
LogFileNum = FreeFile
Open LogFileName For Input As LogFileNum

' load text from file to a local variable
clob_data = Input$(LOF(LogFileNum), LogFileNum)
Close #LogFileNum

' connect as adldemo/adldemo
con.CursorLocation = adUseServer
con.Open "Provider=OraOLEDB.Oracle;Data Source=db9i;" & _
        "User Id=adldemo;Password=adldemo;"

' open a recordset
sql = "select clip_id, story from MULTIMEDIA_TAB"
rst.Open sql, con, adOpenStatic, adLockOptimistic, adCmdText

' add a new record
rst.AddNew
rst!clip_id = 1234
rst!story.AppendChunk (clob_data)
rst.Update

' fetch entire CLOB data
Do While (Not (done))
    vardata = rst!story.GetChunk(4096)
    If Not (IsNull(vardata)) Then
        Data = Data & vardata
    Else
        done = True
    End If
```

```
Loop

' validate fetched data
If Data = clob_data And Len(clob_data) = rst!story.ActualSize Then
    MsgBox "The CLOB data (of " & Len(clob_data) & " bytes) " & _
        "was inserted and retrieved properly!"
End If

' cleanup
con.Execute "delete from multimedia_tab where clip_id = 1234"
rst.Close
con.Close

Exit Sub
ErrorHandler:
    MsgBox "Error: " & Err.Description
End Sub
```

---

## LOBs Case Studies

This chapter contains the following sections:

- [Building a Multimedia Repository](#)
- [Building a LOB-Based Web Site: First Steps](#)

## Building a Multimedia Repository

This description has been extracted from an article by Samir S. Shah in Java Developer's Journal. Reprinted by permission of Java Developer's Journal.

### Toolset Used

- Jdeveloper 2.0 with JDK 1.1.7
- Oracle 8.1.5 or higher
- JDBC Thin Driver
- Oracle8i (8.1.5) Enterprise server
- Java Web Server 2.0
- Oracle intermedia 8.1.5.
- Platform: Windows 2000 Server

Today building an information repository is essential for businesses. the information repository helps establish a paperless office and allows data to be shared in or outside an enterprise.

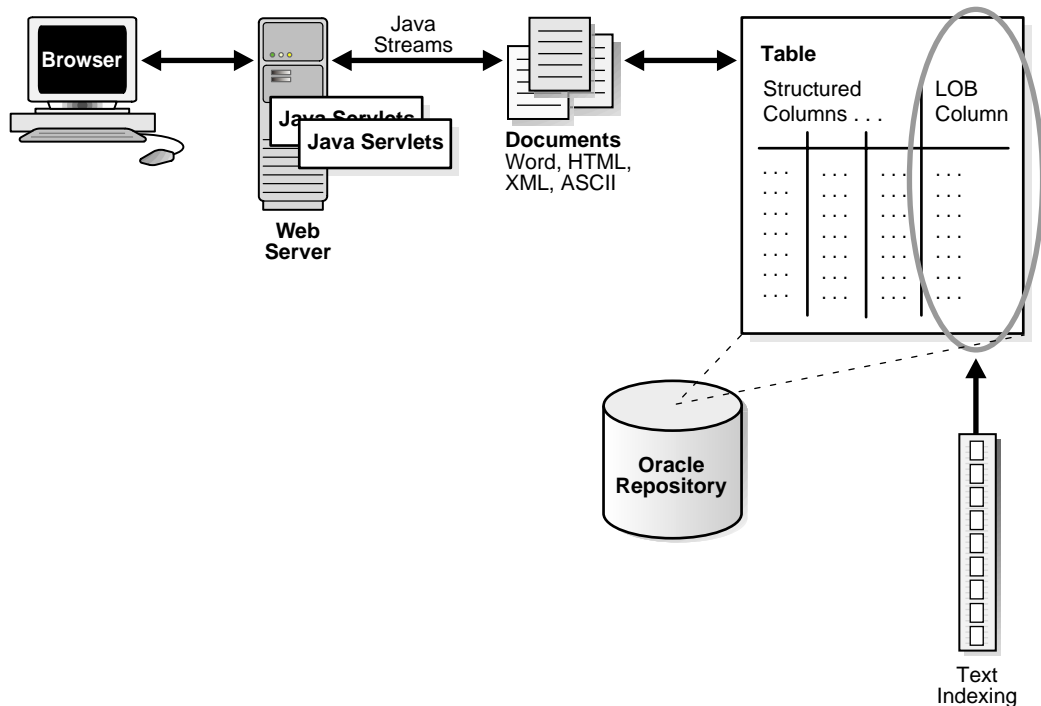
With the toolset mentioned earlier, you can build an enterprise-class, scalable web-enabled multimedia-rich information repository that incorporates various forms of media. This repository includes unstructured data, such as document files, video clips, photographs, ... and sound files. It uses Java and Oracle's Large Objects (LOBs).

This section describes how you can build such an information repository for storing and searching documents such as Microsoft Word, HTML, and XML files which are stored in a LOB column of a database table.

The example used here populates the repository with Microsoft Word resumes, indexes it using Oracle Text (*interMedia Text*), and reads the repository using Java streams from a servlet.

See [Figure 14-1](#).

Figure 14–1 Data Repository Using Oracle and Java



Building repositories using Java and Oracle8i/9i has several benefits. The documents can inherently take advantage of the transaction management and ACID (Atomocity, Concurrency, Integrity, and Durability) properties of the relational database. This means that changes to an internal LOB can be committed or rolled-back. Moreover, because the unstructured data is stored by the database, your applications can seamlessly take advantage of database features such as backup and recovery. This helps Administrators who would no longer have to perform separate database and file system backups for relational information and documents.

All data in the database, including structured (relational) and unstructured (document files), can be written, searched, and accessed using SQL. The SQL statements can be executed from Java using JDBC.

## How this Application Uses LOBs

Oracle8i and Oracle9i support several types of LOB columns. One type, BLOBs, can house binary information such as audio, video, images, and comments internally in the database. Each row can store up to 4 gigabytes of data. The application described here uses a BLOB data type to store Microsoft Word resumes.

The Oracle database stores a locator in-line with the data. The locator is a pointer to the actual location of the data (LOB value). The LOB data can be stored in the same or a separate table. The advantage of using the locator is that the database will not have to scan the LOB data each time it reads multiple rows because only the LOB locator value is read. The actual LOB data is read only when required.

When working with Java and LOBs, first execute the SELECT statement to get the LOB locator, then read or write LOBs using JDBC.

---

---

**Note:** The JDBC driver's Oracle type extension package, `oracle.sql`, is used to read and write from an oracle database.

---

---

The actual LOB data is materialized as a java stream from the database, where the locator represents the data in the table. The following code reads the resume of an employee whose employee number is 7900. Employee number is stored in a LOB column called "resume" in table, `sam_emp`.

```
Statement st = cn.createStatement();
ResultSet rs = st.executeQuery
("Select resume from sam_emp where empno=7900");
rs.next();
oracle.sql.BLOB blob=((OracleResultSet)rs).getBLOB(1);
InputStream is=blob.getBinaryStream();
```

## Populating the Repository

The documents can be written to LOB columns using Java, PL/SQL, or a bulk loading utility called Oracle SQL\*Loader. To insert a new row, perform the following:

1. Execute the SQL insert statement with an empty BLOB.
2. Query the same row to get the locator object. Use this to write your document to the LOB column.



---



---

**Note:** Java streams are employed to write the documents to the LOB column.

---



---

3. Create the Java output stream using the `getBinaryOutputStream()` method of this object to write your document or any binary information to that column. For example, to insert information about a new employee whose employee number is 9001 in table `sam_emp`, first insert all the structured information along with an empty BLOB using JDBC. Next select the LOB column, resume, of the same row to get the `oracle.sql.BLOB` object (the locator).
4. Finally, create the Java output stream from this object. For example, to insert information about a new employee whose employee number is 9001 in the `sam_emp` table, first insert all the structured data along with an empty BLOB, using JDBC. Next, select the LOB column, resume, from the same row to get the `oracle.sql.BLOB` object (the locator). Finally, create the Java output stream from this object:

```
st.execute("INSERT INTO sam_emp(empno, resume)
VALUES(9001,empty_blob())");
ResultSet rs = st.executeQuery(
    "select resume from sam_emp where empno=9001 for update");
rs.next();
oracle.sql.BLOB blob = ((OracleResultSet)rs).getBLOB(1);
OutputStream os = blob.getBinaryOutputStream();
```

Optionally, you can use `java.awt.FileDialog` class and `java.io` package to dynamically select and read a file from your PC. Then, load it to a LOB column using the preceding code.

The way you search and retrieve documents does not depend on how you load the documents. For example, you can store the documents using PL/SQL or SQL\*Loader, then search and retrieve using Java servlets.

## Example 1: Inserting a Word document into a BLOB Column using PL/SQL

The following code (steps 2-5) inserts `MyResume.doc` in the `resume` column of `sam_emp` table.

1. Create a directory object in Oracle. Here is how to create a directory object called `MY_FILES` which represents `C:\MY_DATA` directory.

You must have `CREATE DIRECTORY` privilege in Oracle.

```
create or replace directory MY_FILES as 'C:\MY_DATA';
```

2. Insert a row with empty BLOB in your table and return the locator.
3. Point to the Word file to be loaded from the directory created in Step 1, using the BFILE data type.
4. Open the file and use the locator from step 2 to insert the file.
5. Close the file and commit the transaction.

```
declare
    f_lob    bfile;
    b_lob    blob;

begin

    insert into sam_emp(empno,ename,resume)
    values ( 9001, 'Samir',empty_blob() )
    return documents into b_lob;

    f_lob := bfilename( 'MY_FILES', 'MyResume.doc' );
    dbms_lob.fileopen(f_lob, dbms_lob.file_readonly);
    dbms_lob.loadfromfile
    ( b_lob, f_lob, dbms_lob.getlength(f_lob) );
    dbms_lob.fileclose(f_lob);

    commit;

end;
/
```

## Searching the Repository

Documents stored in the LOB columns can be indexed using Oracle Text (*interMedia* Text). Oracle9i Text provides you with advanced search capabilities such as fuzzy, stemming, proxy, phrases, and more. It can also generate thematic searches and gist. The documents can be indexed using 'create index' database command.

### See Also:

- *Oracle Text Application Developer's Guide*
- *Oracle Text Reference*

## How the Index Was Built on Table `sam_emp`, resume Column

The following code shows you how the index was built on the resume column of the `sam_emp` table. Once the index is created, the Java applications can search the repository by simply submitting SELECT statements.

The following steps index all the Microsoft Word formatted resumes stored in the resume column to the `sam_emp` table. The resumes can then be searched using SQL.

1. Add primary key to your table if it does not exist. To make empno primary key of the `sam_emp` table execute following command:

```
alter table sam_emp add constraint
pk_sam_emp primary key(empno);
```

2. Get the privileges (ctxapp role) to create text indexes from your administrators.
3. Create the index with appropriate filter object. Filters determine how to extract text for document indexing from word processor, formatted documents as well as plain text.

```
create index ctx_doc_idx on sam_emp(resume)
indextype is ctxsys.context parameters
('filter CTXSYS.INSO_FILTER');
```

### See Also:

- *Oracle Text Application Developer's Guide*
- *Oracle Text Reference*

for a complete list of filters.

## MyServletCtx Servlet

The following code lists the servlet 'MyServletCtx'. It searches the term passed to it as a parameter in the resume column of table, `sam_emp`. The servlet returns the rows matching the search criteria in HTML table format. The employee names in the HTML table are hyperlinked to another servlet, 'MyServlet', which reads the entire resumé from the database, in its original format.

### MyServletCtx.java

```
12345678901234567890123456789012345678901234567890123456789012
package package1;
import javax.servlet.*;
```

```
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;

/**
 * This servlet searches documents stored in Oracle8i
 * database repository using SQL and JDBC. The hit
 * list is displayed in html table with hyper links.
 * JDK 1.1.7 and Oracle Thin JDBC 1.22 compliant
 * driver is used.
 *
 * @author Samir Shah
 * @version 1.0
 */
public class MyServletCtx extends HttpServlet{
    Connection cn;

    public void init(ServletConfig paml)
    throws ServletException {
        super.init( paml);
        try{
            DriverManager.registerDriver(
(new oracle.jdbc.driver.OracleDriver()));
            cn =DriverManager.getConnection
("jdbc:oracle:thin:@sshah:1521:o8i",
            "scott", "tiger");
        }
        catch (SQLException se){se.printStackTrace();}
    }

    public void doGet(HttpServletRequest req,
        HttpServletResponse res) throws IOException{

        doPost(req,res);
    }

    public void doPost(HttpServletRequest req,
        HttpServletResponse res) throws IOException{

        PrintWriter out = res.getWriter();
        res.setContentType("text/html");

        //The term to search in resume column
        String term = req.getParameter("term");
        if (term == null)
```

```
        term="security";

        out.print("<html>");
        out.print("<body>");
        out.print("<H1>Search Result</H1>");
        out.print("<table border=1 bgcolor=lightblue>");
        out.print("<tr><th>ID#</th><th>Name</th></tr>");
        out.print("<tr>");
        try{
            Statement st = cn.createStatement();

            //search the term in resume column using SQL
            String query =
                "Select empno,ename from sam_emp" +
                " where contains(resume,'" +term+"' )>0";

            ResultSet rs = st.executeQuery(query);

            while (rs.next()){
                out.print("<td>" + rs.getInt(1)+"</td>");
                out.print("<td>" +
                    "<A HREF=http://sshah:8080/" +
                    "servlet/MyServlet?term=" +
                    rs.getString(1) +
                    " target=Document>" +
                    rs.getString(2) +
                    "</A></td>");
                out.print("</tr>");
            }

            out.print("</table>");
            out.print("</body>");
            out.print("</html>");
        }//try
        catch (SQLException se){se.printStackTrace();}

    }
}
```

## Retrieving Data from the Repository

The document retrieval using Java is similar to writing documents to the repository. The section, "[How this Application Uses LOBs](#)" on page 14-4 describes how to read LOBs from the database.

The following code in 'MyServlet' reads a Microsoft Word resumé from the table, sam\_emp. It sets the content type, then streams it out to the browser using an output stream.

### MyServlet.java

```
12345678901234567890123456789012345678901234567890123456789012
package package1;

import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.io.*;
import oracle.jdbc.driver.*;
import oracle.sql.*; //for oracle.sql.BLOB

/**
 * This class reads the entire document from the
 * resume LOB column. It takes one parameter, term,
 * to search a specific employee from the sam_emp
 * table and returns the document stored in that
 * row.
 *
 *
 * JDK 1.1.7, Oracle Thin JDBC 1.22 compliant driver
 * Use Oracle JDBC Type extends package oracle.sql.
 *
 * @author Samir Shah
 * @version 1.0
 */
public class MyServlet extends HttpServlet{
    Connection cn;

    public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    {
        try{
            doPost(req,res);
        }catch (IOException ie){ie.printStackTrace();}
    }
}
```

```
public void init(ServletConfig parml)
throws ServletException
{

    super.init( parml);
    try{
    DriverManager.registerDriver(
    (new oracle.jdbc.driver.OracleDriver()));
    cn =DriverManager.getConnection(
        "jdbc:oracle:thin:@sshah:1521:o8i",
        "scott", "tiger");
    }
    catch (SQLException se){se.printStackTrace();}
}

public void doPost(HttpServletRequest req,
    HttpServletResponse res) throws IOException
{
    InputStream is=null;
    oracle.sql.BLOB blob=null;

    res.setContentType("application/msword");
    OutputStream os = res.getOutputStream();
    String term = req.getParameter("term");

    if (term==null)
        term="9001";

    try{
    Statement st = cn.createStatement();
    ResultSet rs = st.executeQuery
        ("Select resume from sam_emp"+
        " where empno="+term);

    while (rs.next()){
        blob=((OracleResultSet)rs).getBLOB(1);
        is=blob.getBinaryStream();
    }

    int pos=0;
    int length=0;
    byte[] b = new byte[blob.getChunkSize()];

    while((length=is.read(b))!= -1){
```

```
        pos+=length;
        os.write(b);
    }
} //try
catch (SQLException se)
{
    se.printStackTrace();
}
finally {
    is.close();
}
}
}
```

## Summary

This section showed you how to store, search and retrieve Word documents using LOB data types and Java.

You can also store, index, parse and transform XML documents using the Oracle9i database. By storing XML documents in the database, there is no need to administer and manage multiple repositories for relational and XML data. Oracle9i and Oracl9i Application Server are XML-enabled whereby you can run the Oracle XML Parser for Java and parse and transform XML files in the database before outputting to an application server.



## Building a LOB-Based Web Site: First Steps

### Problem

Design and Build a LOB and interMedia Based Web site. The Web site must include video 'thumbnails' where by users can click a specific thumbnail to see a short 6 - 8 second video clip.

### First Steps Solution

Here are some ideas for setting up your LOB-based web-site:

1. Install Oracle9i (including *interMedia*) on your database server.
2. Install a web server, such as, Oracle9i Application Server, IIS, Netscape Web server, or Apache.
3. Install the *interMedia* Web Agent on your web server
4. Install the *interMedia* ClipBoard on your client (PC)
5. On your server, create a table with at least three columns, such as:

```
create table video_clips (  
  move_id integer,  
  thumbnail ordsys.ordimage,  
  movie ordsys.ordvideo);
```

See Note 2.

6. Collect/Capture your media content (images, movies)
7. If you're using a digital camera or scanner *interMedia* ClipBoard will help you with this
8. Use *interMedia* ClipBoard to upload your media content into the database, into the table you created in step 5.
9. Use a HTML authoring tool, such as DreamWeaver, FrontPage, ... in conjunction with *interMedia* ClipBoard to build your web pages.
10. Add the thumbnails with the help of *interMedia* ClipBoard, with a caption. Make the thumbnails have hyperlinks to the movie clips. It is recommended to not use a separate streaming server at this point. One way to do this is to encode the movies as Apple QuickTime files, for example, if you do this correctly they'll start playing as they download... This is not quite the same as

"streaming". If you have reasonable bandwidth, this should be more than sufficient.

11. DO you need plug-ins? How about Space requirements? Assume you have about 100 movie clips and they all take a total of about 30+ minutes. You should not need any plugins, that is no Real Networks plugins.

Your disk space depends on the frame size, frame rate, and compression settings. One second of video, at 720x480 pixels, 30 frames per second (fps), takes roughly 3.6MB of disk space. 720x480 is pretty big for the web but should be fine if this is on an intranet. 30 fps looks very smooth but might not be necessary. Test a sample to see what 320x240 looks like. Check if there is sufficient detail present. If not, increase the resolution until you are satisfied.

---

---

**Note 1:**

- This isn't likely to be trivial to set up. Just getting everything installed and configured could be challenging. Enroll the help of Oracle DBAs and consultants
  - If you can, specify a `DB_BLOCKS_SIZE` of 8K and as many `DB_BLOCK_BUFFERS` as you can.
- 
- 

---

---

**Note 2:** The foregoing example is a simplistic create table example. You typically need LOB storage clauses for LOBs inside `ORDImage` and `ORDVideo`. You also need a separate tablespace for these LOBs, `CHUNK 32768`, `NOCACHE` on the `VIDEO` LOB, `CACHE` on the `IMAGE` LOB.

---

---

**See Also:**

- *Oracle interMedia User's Guide and Reference*
- *Using Oracle8i with the Web*

---

## Unified Modeling Language Diagrams

The Unified Modeling Language (UML) use case diagrams in this manual present a representation of the technology used in Advanced Queuing. A brief explanation of use case diagrams and UML notation follows.

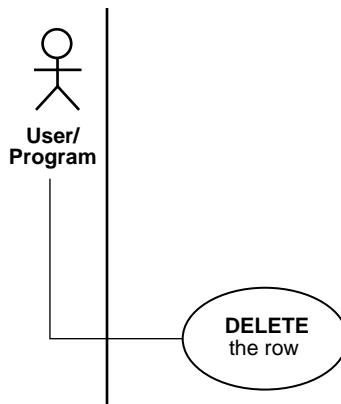
This chapter discusses the following topics:

- [Use Case Diagrams](#)
- [State Diagrams](#)

## Use Case Diagrams

In a use case diagram, the primary use case is instigated by an actor (stickman), which can be a human user, an application, or a subprogram. The actor is connected to the primary use case, which is depicted as an oval (bubble) enclosing the use case action, as shown in [Figure A-1](#).

**Figure A-1** A Primary Use Case



Primary use cases may require other operations to complete them. In [Figure A-2](#),

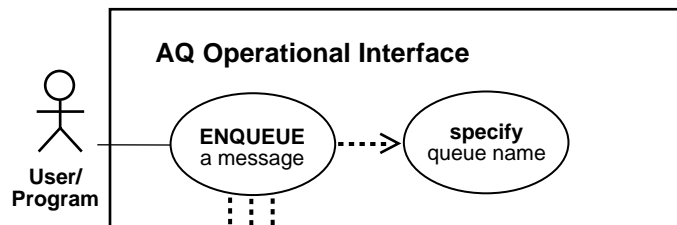
- **specify** queue name

is one of the suboperations, 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).

**Figure A-2** A Primary Use Case with Suboperations



As shown in [Figure A-3](#), a secondary use case with a drop shadow expands into its own use case diagram, thus making it easier to:

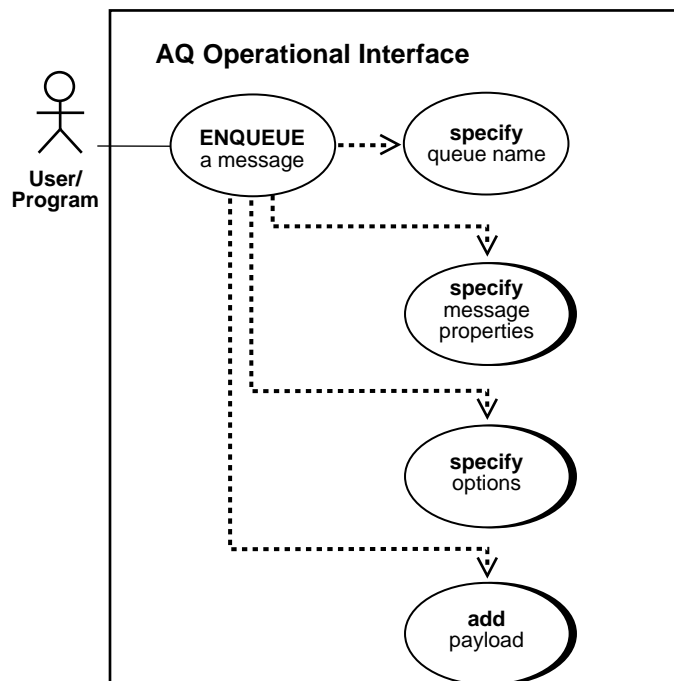
- Understand the logic of the operation
- Continue a complex operation across multiple pages

In this example

- **specify** message properties
- **specify** options
- **add** payload

are all expanded in separate use case diagrams.

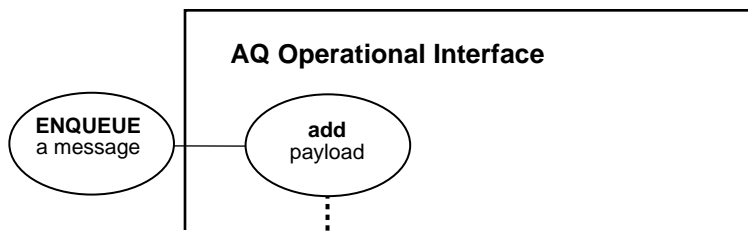
**Figure A-3** A Use Case Diagram with Drop Shadows Representing Secondary Use Cases



The diagram fragment in [Figure A-4](#) shows an expanded use case diagram. While the standard diagram has the actor as the initiator, here the use case itself is the point of departure for the suboperation. In this example, the expanded view of

- **add** payload
- represents a constituent operation of
- **ENQUEUE** a message

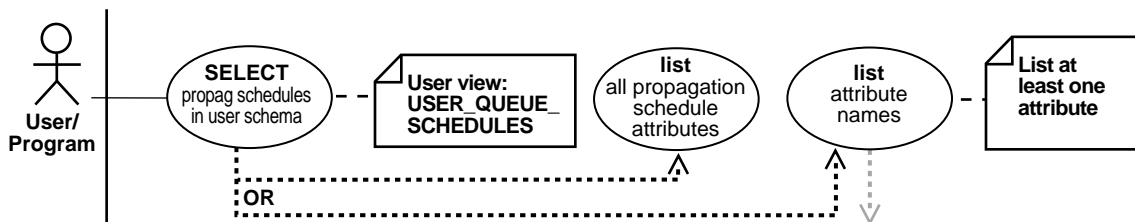
**Figure A-4** *The Expanded Use Case Diagram*



[Figure A-5](#) shows how note boxes are used:

- Note boxes can present an alternative name. In this case, the action `SELECT` propagation schedules in the user schema is represented by the view `USER_QUEUE_SCHEDULES`.
- Note boxes can qualify the use case action. In this case, the list attribute names action is qualified by the note that you must list at least one attribute if you do not list all the attributes of the propagation schedule.

**Figure A-5** *Note Boxes*



The dotted arrow in the use case diagram indicates dependency. In [Figure A-6](#)

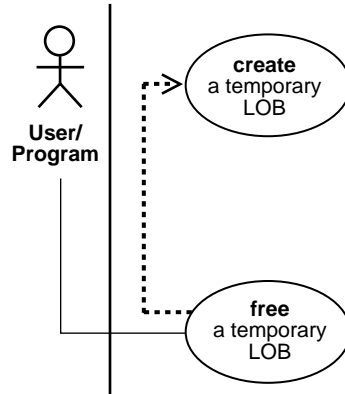
- **free** a temporary LOB

requires that you first

- **create** a temporary LOB

The target of the arrow shows the operation that must be performed first.

**Figure A-6** Dependencies



Use cases and their suboperations can be linked in complex relationships. In the example in [Figure A-7](#), you must first

- **REGISTER** for notification

to later

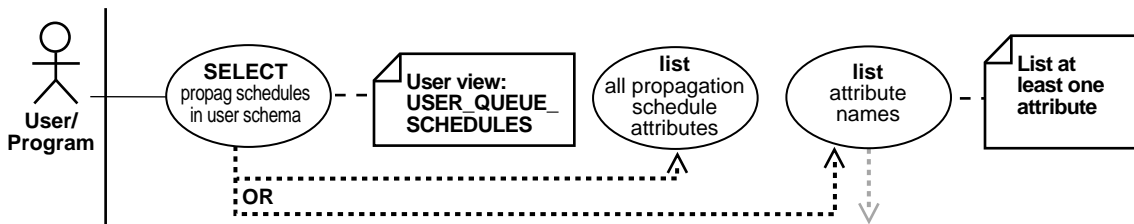
- **receive** a notification

**Figure A-7** Use Case and Suboperation Relationships



In [Figure A-8](#), the branching paths of an OR condition are shown. In invoking the view, you can choose to list all the attributes or view one or more attributes. The grayed arrow indicates that you can stipulate which attributes you want to view.

**Figure A-8 The Branching Paths of an OR Condition**



In [Figure A-9](#), the black dashed line and arrow indicate that the targeted operation is required. The gray dashed line and arrow indicate that the targeted operation is optional. In this example, executing

- **write** append

on a LOB requires that you first

- **SELECT** a LOB

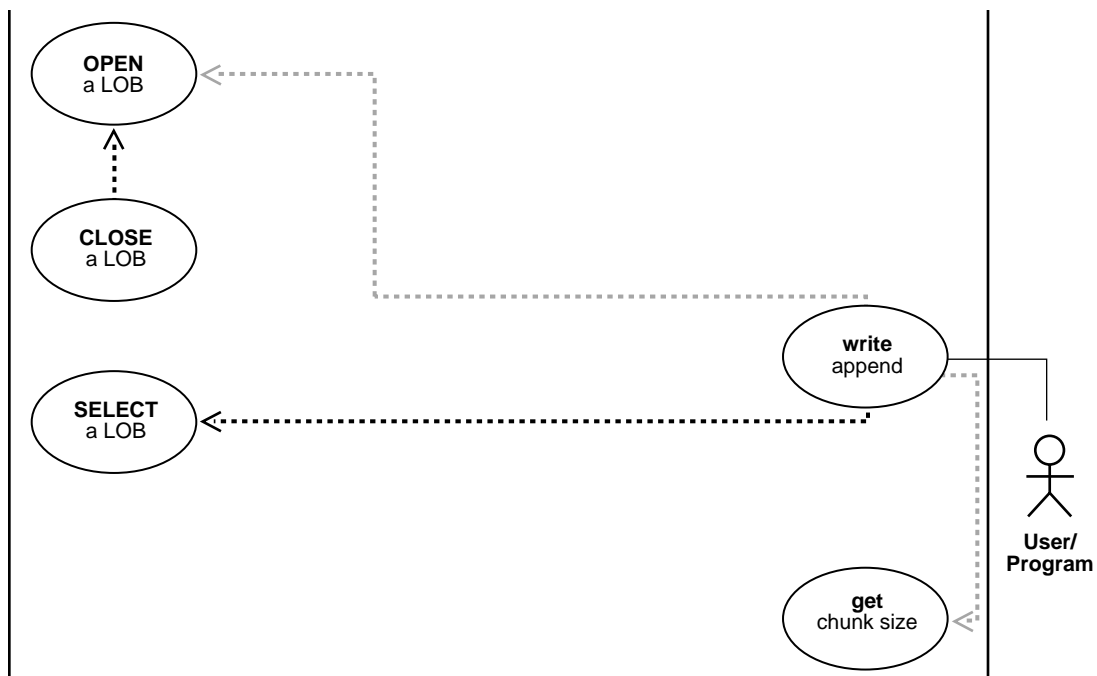
You may optionally choose to

- **OPEN** a LOB or get chunk size

The diagram shows that if you open a LOB, you must also close it.



Figure A-9 Required and Optional Operations

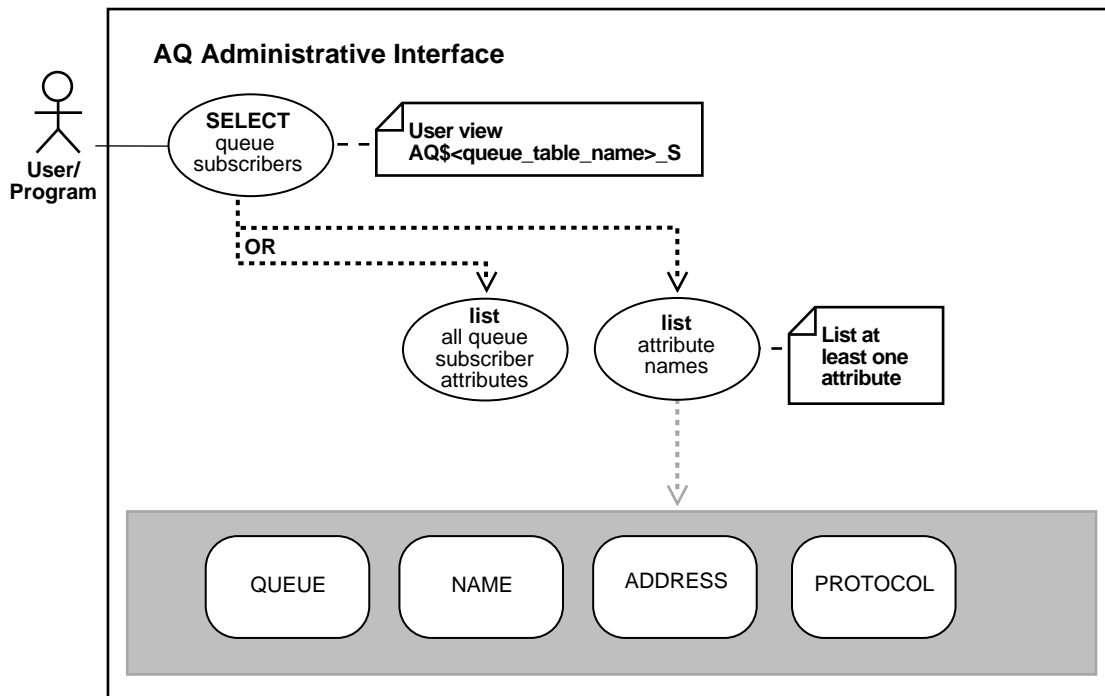


## State Diagrams

A state diagram presents the attributes of a view. Attributes of a view have two states—visible or invisible. In this example, a state diagram (the Queue, Name, Address, and Protocol boxes in the gray area at the bottom of the figure) is added below a use case diagram to show all the attributes of the view.

Figure A-10 shows that the view is for querying queue subscribers. You can stipulate one attribute, some combination of the four attributes, or all four attributes.

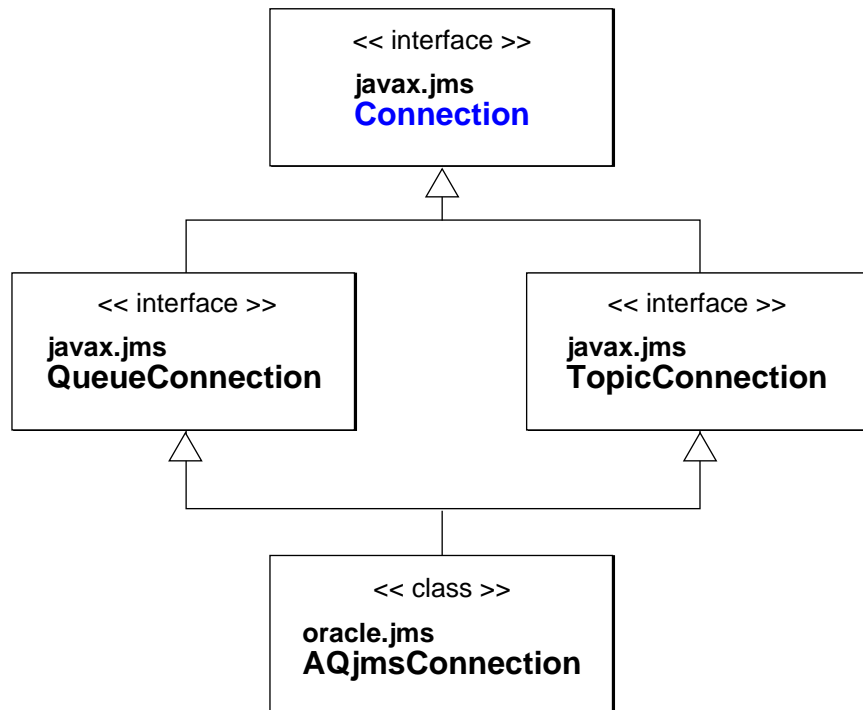
Figure A-10 Use Case and State Diagram to Show Attributes of a View



The class diagram in [Figure A-11](#) shows:

- Whether classes, interfaces, and exceptions are entailed in the interrelationship by means of the <<>>, stereotype, such as <<interface>>
- The name of the package in which the class is found, such as **oracle.jms**
- The name of the class, such as **AQjmsConnection**

Figure A-11 A Class Diagram Representing Classes, Interfaces, and Exceptions





---

---

## The Multimedia Schema

---

---

**Note:** The Multimedia Sample Schema has been deprecated and is not supplied with Oracle9i. This appendix is included to provide context for examples in this guide that have not yet been migrated to the new Product Media (PM) Sample Schema which replaces the Multimedia schema for most LOB examples.

For more information about sample schemas, see "[Examples in This Guide](#)" on page 1-9.

---

---

This appendix describes the following topics:

- [A Typical Multimedia Application](#)
- [The Multimedia Schema](#)
- [Table Multimedia\\_Tab](#)
- [Script for Creating the Multimedia Schema](#)

## A Typical Multimedia Application

Oracle9i supports LOBs, *large objects* which can hold up to 4 gigabytes of binary or character data. What does this mean to you, the application developer?

Consider the following multimedia scenario.

Multimedia data is used in an increasing variety of media channels — film, television, Web pages, 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). Despite these differences, there is often considerable similarity in the multimedia authoring process, especially with regard to assembling content.

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, such as, storyboard, transcript, subtitles
- Images, such as, photographs, video frames
- Line drawings, such as, maps
- Audio, such as, 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.

### **This Scenario is Only An Example**

Our mission in this appendix is not to create this real-life application, but to describe some typical scenarios you may 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 LOBs.

Also 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.*

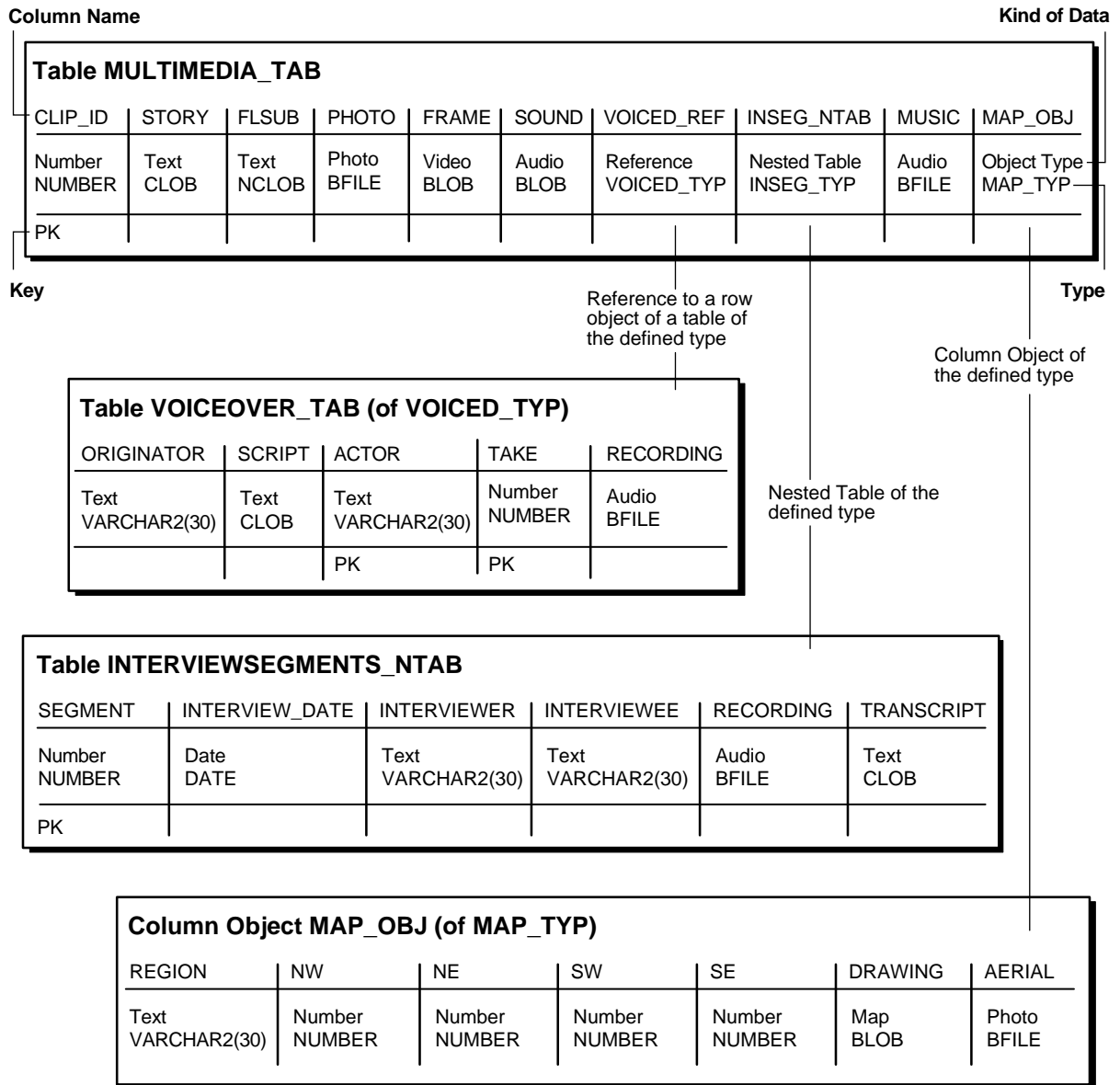
## **The Multimedia Schema**

[Figure B-1](#) illustrates multimedia schema used for the examples in this manual.

The Multimedia schema is comprised of the following components:

- Table Multimedia\_tab
- Table VoiceOver\_tab
- Nested Table INSEG\_NTAB
- Column Object MAP\_OBJ

Figure B-1 The Multimedia Schema





## Table Multimedia\_Tab

Figure B-1, "The Multimedia Schema", shows table Multimedia\_tab's structure. Table Multimedia\_tab columns are described in the following:

- **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:** A `BLOB` column for sound-effects.
- **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 living, 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:

- CLOB to store the script which the actor reads
- BFILE for the audio recordings.
- **INSEG\_NTAB**: While it is not possible to store a Varray of LOBs, application builders can store a variable number of multimedia *elements* in a single row using *nested tables*. In our application, nested table `InSeg_ntab` of 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, nested table, `interviewsegments_ntab`, makes use of the following two LOB datatypes:

- BFILE to store the audio recording of the interview
- 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.

- **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 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.

## Script for Creating the Multimedia Schema

Here is the script used to create the Multimedia schema:

```

CONNECT system/manager;
DROP USER samp CASCADE;
DROP DIRECTORY AUDIO_DIR;
DROP DIRECTORY FRAME_DIR;
DROP DIRECTORY PHOTO_DIR;
DROP TYPE InSeg_typ force;
DROP TYPE InSeg_tab;
DROP TABLE InSeg_table;
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;
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;
```

**See Also:** [Chapter 10, "Internal Persistent LOBs", "Creating a Table Containing One or More LOB Columns"](#) on page 10-5.

This script is also located in \$HOME Oracle9i "demo" directory in the following files:

- lobdemo.sql
- adloci.sql.

**See Also:** For further LOB examples:

- *Oracle interMedia User's Guide and Reference*
- *Oracle interMedia Java Classes User's Guide and Reference*
- *Oracle Text Application Developer's Guide*
- *Oracle Text Reference*



---

---

# Index

## A

---

accessing external LOBs, 12-4  
ALTER TABLE  
    migrating from LONG to LOB, 8-3  
amount, 12-109  
amount parameter  
    reading and loading LOB data, the size of  
        (FAQ), 6-3  
    used with BFILES, 12-47  
ANSI standard for LOBs, 11-8  
AppendChunk(), see OraOLEDB, 13-3  
appending  
    one LOB to another  
        internal persistent LOBs, 10-193  
        one temporary LOB to another, 11-150  
    writing to the end of a LOB  
        internal persistent LOBs, 10-203  
assigning  
    one collection to another collection in temporary  
        LOBs, 11-9  
    one temporary LOB to another, 11-8

## B

---

BFILE class, See JDBC  
BFILE-buffering, See JDBC  
BFILENAME(), 12-22, 12-194  
    advantages of using, 12-6  
BFILES  
    accessing, 12-4  
    converting to CLOB or NCLOB, 12-47  
    creating an object in object cache, 5-17  
    datatype, 2-2, 2-3

DBMS\_LOB read-only procedures, 3-10  
DBMS\_LOB, offset and amount parameters in  
    bytes, 3-7  
equal locators, check for, 12-177  
hard links and symbolic links not allowed, 4-1  
initializing using BFILENAME(), 2-6  
locators, 2-5  
maximum number of open, 4-2, 12-154  
multi-threaded server (MTS), 4-18, 12-10  
OCI functions to read/examine values, 3-15,  
    3-27  
OCI read-only functions, 3-15, 3-27  
opening and closing using JDBC, 3-56  
operating system files, and, 4-1  
Oracle Objects for OLE (OO4O)  
    opening/closing methods, 3-42  
    properties, 3-44  
    read-only methods, 3-43  
Pro\*C/C++ precompiler statements, 3-31  
Pro\*COBOL precompiler embedded SQL  
    statements, 3-35  
reading with DBMS\_LOB, 3-9  
reference semantics, 2-3  
rules for using, 4-1  
security, 12-7  
storage devices, 2-2  
streaming APIs, 3-63  
using JDBC to read/examine, 3-50  
using Pro\*C/C++ precompiler to open and  
    close, 3-32  
binding data to internal LOBs, restriction  
    removal, xlix  
binding LONG columns or LOB columns, but not  
    both, xlix

- binds
    - HEX to RAW or RAW to HEX conversion, 7-15
    - updating more than 4,000 bytes
      - internal persistent LOBs, 10-272
    - See also INSERT statements and UPDATE statements
  - BLOB-buffering, See JDBC
  - BLOBs
    - class, See JDBC
    - datatype, 2-2
    - DBMS\_LOB, offset and amount parameters in
      - bytes, 3-7
    - modify using DBMS\_LOB, 3-8
    - using JDBC to modify, 3-48
    - using JDBC to read/examine BLOB values, 3-48
    - using oracle.sql.BLOB methods to modify, 3-48
  - buffering
    - disable
      - internal persistent LOBs, 10-263
    - enable
      - internal persistent LOBs, 10-251
    - flush
      - internal persistent LOBs, 10-257
    - LOB buffering subsystem, 5-21
- ## C
- 
- C++, See Pro\*C/C++ precompiler
  - C, See OCI
  - CACHE / NOCACHE, 7-8
  - caches
    - object cache, 5-17
  - callback, 10-72, 10-84, 10-110, 10-204, 11-169
  - catalog views
    - v\$temporary\_lob, 11-10
  - CHAR buffer, defining on CLOB, 7-40
  - CHAR to CLOB
    - SQL and PL/SQL conversion, 7-35
  - character data
    - varying width, 2-4
  - character set form
    - getting
      - internal persistent LOBs, 10-189
  - character set ID
    - getting the
      - internal persistent LOBs, 10-185
      - temporary LOB of, getting the, 11-144
    - See CSID parameter
  - charactersets
    - multibyte, LONGs and LOBs, 8-5
  - checking in a LOB
    - internal persistent LOBs, 10-83
  - checking out a LOB
    - internal persistent LOBs, 10-71
  - CHUNK, 7-10
  - chunks
    - when to use, 6-26
  - chunksize, 10-213
    - multiple of, to improve performance, 10-111
  - classes
    - putChars(), 6-10
  - CLOB class, See JDBC
  - CLOB-buffering, See JDBC
  - CLOBs
    - columns
      - varying- width character data, 2-4
    - datatype, 2-2
      - varying-width columns, 2-4
    - DBMS\_LOB, offset and amount parameters in
      - characters, 3-7
    - modify using DBMS\_LOB, 3-8
    - opening and closing using JDBC, 3-54
    - reading/examining with JDBC, 3-49
    - using JDBC to modify, 3-49
  - Clone method, See Oracle Objects for OLE (OO4O)
  - closing
    - all open BFILES, 12-222
    - BFILES with CLOSE, 12-214
    - BFILES with FILECLOSE, 12-208
  - clustered tables, 8-10
  - COBOL, See Pro\*COBOL precompiler
  - comparing
    - all or part of two LOBs
      - internal persistent LOBs, 10-128
      - all or part of two temporary LOBs, 11-92
      - all or parts of two BFILES, 12-127
  - compatibility, 1-8
  - conventional path load, 9-3
  - conversion
    - explicit functions for PL/SQL, 7-47



- conversions
  - character set, 12-47
  - from binary data to character set, 12-47
  - implicit, between CLOB and VARCHAR2, 7-45
  - See also binds HEX to RAW
- conversions, implicit between CLOBs and CHAR, 7-36
- converting
  - between different LOB types, 6-24
  - to CLOB, 7-47
- converting to LOB data types, 6-2
- copy semantics, 2-3
  - internal LOBs, 10-21
- copying
  - all or part of a LOB to another LOB
    - internal persistent LOBs, 10-153
  - all or part of one temporary LOB to another, 11-115
  - for BFILEs there is no copy function, 12-196
  - LOB locator
    - internal persistent LOBs, 10-164
  - LOB locator for BFILE, 12-164
  - LONG to LOB (FAQ), 6-2
  - temporary LOB locator, 11-125
  - TO\_LOB limitations, 10-66
- crashing
  - Is LOB data lost after (FAQ), 6-5
- creating a temporary LOB, 11-13
- creating tables
  - containing an object type with LOB attribute
    - internal Persistent LOBs, 10-10
  - containing one or more LOB columns
    - internal persistent LOBs, 10-5
  - containing one ore more BFILE columns, 12-13
  - nested, containing LOB
    - internal persistent LOBs, 10-13
  - of an object type with BFILE attribute, 12-17
  - with a nested table containing a BFILE, 12-20
- creating VARRAYs
  - containing references to LOBs, 5-28
- CSID parameter
  - setting OCILobRead and OCILobWrite to OCI\_UCS2ID, 3-11

## D

---

- datatypes
  - converting to LOBs FAQ, 6-2
- DBMS\_LOB
  - ERASE, 6-10
  - substr compared with read, 6-26
  - updating LOB with bind variable, 5-10
  - WRITE()
    - passing hexadecimal string to, 10-214
- DBMS\_LOB functions on a NULL LOB
  - restriction, 2-9
- DBMS\_LOB package
  - available LOB procedures/functions, 3-3, 3-4
  - client procedures cannot call DBMS\_LOB, 3-7
  - CREATETEMPORARY(), 11-15
  - .createTemporary() as JDBC workaround, 11-11
  - for temporary LOBs, 3-9
  - functions/procedures to modify BLOB, CLOB, and NCLOB, 3-8
  - functions/procedures to read/examine internal and external LOBs, 3-9
  - isTemporary() as JDBC workaround, 11-11
  - LOADFROMFILE(), 12-47
  - multi-threaded server (MTS), 4-18, 12-10
  - offset and amount parameter rules, 3-7
  - open and close, JDBC replacements for, 3-52
  - opening/closing internal and external LOBs, 3-10
  - provide LOB locator before invoking, 3-6
  - read-only functions/procedures for BFILEs, 3-10
  - to work with LOBs, using, 3-6
  - WRITE()
    - guidelines, 10-214
    - guidelines for temporary LOBs, 11-169
    - passing hexadecimal string to, 11-169
- DBMS\_LOB()
  - READ, 10-110
- DBMS\_LOB.createtemporary(), 11-5
- DBMS\_LOB.isTemporary(), previous workaround for JDBC, 11-29
- DBMS\_LOB.LOADFROMFILE, 2-4
- DBMS\_LOB.READ, 12-109
- DELETE

- BLOB columns versus BFILE columns, and LOB indexing, 6-16
- deleting
  - row containing LOB
    - internal persistent LOBs, 10-286
- directories
  - catalog views, 12-9
  - guidelines for usage, 12-9
  - ownership and privileges, 12-7
- DIRECTORY name specification, 12-7
- DIRECTORY object, 12-4
  - catalog views, 12-9
  - getting the alias and filename, 12-184
  - guidelines for usage, 12-9
  - names on Windows NT, 12-7
  - naming convention, 12-7
  - OS file must exist before locator use, and, 12-23
  - READ permission on object not individual files, 12-8
  - rules for using, 4-1
  - symbolic links, 4-1
  - symbolic links, and, 4-1
- directory objects, 12-4
- directory\_alias parameter, 12-24
- direct-path load, 9-3
- disable buffering, See LOB buffering
- DISABLE STORAGE IN ROW, 6-21
  - when to use, 6-19
- disk striping of LOB files, B-3
- displaying
  - LOB data for internal persistent LOBs, 10-98
  - temporary LOB data, 11-64

## E

---

- embedded SQL statements, See Pro\*C/C++ precompiler and Pro\*COBOL precompiler
- empty LOBs
  - creating using JDBC, 3-68
  - JDBC, 3-68
- EMPTY\_BLOB()
  - setdata using JPublisher (FAQ), 6-8
- EMPTY\_BLOB()/EMPTY\_CLOB()
  - when to use (FAQ), 6-6
- EMPTY\_CLOB()

- LOB locator storage, 6-20
- EMPTY\_CLOB()/BLOB()
  - to initialize a BFILE, 2-6
  - to initialize internal LOB
- equal
  - one LOB locator to another
    - internal persistent LOBs, 10-173
    - one temporary LOB locator, to another, 11-135
- equal locators
  - checking if one BFILE LOB locator equals another, 12-177
- erasing
  - part of LOB
    - internal persistent LOBs, 10-241
    - part of temporary LOBs, 11-188
- errors
  - ORA-03127, 6-11
- examples
  - repercussions of mixing SQL DML with DBMS\_LOB, 5-6
  - updated LOB locators, 5-8
  - updating a LOB with a PL/SQL variable, 5-10
- existence
  - check for BFILE, 12-145
- extensible indexes, 7-32
- external callout, 5-23
- external LOBs (BFILES)
  - See BFILES
- external LOBs (BFILES), See BFILES

## F

---

- FILECLOSEALL(), 12-10, 12-64, 12-79
- flushing
  - LOB buffer, 5-22
- flushing buffer, 5-18
  - temporary LOB, 11-204
- FOR UPDATE clause
  - LOB locator, 5-2
  - LOBs, 2-7
- freeing
  - temporary LOBs, 11-31
- FREETEMPORARY(), 11-31
- freeTemporary() as JDBC workaround, 11-11
- functional indexes, 7-32

function-based indexing, 1-7

## G

---

GetChunk(), see OraOLEDB, 13-3

## H

---

hard links, rules with BFILES, 4-1

hexadecimal string

    passing to DBMS\_LOB.WRITE(), 10-214, 11-169

## I

---

implicit conversions, 7-36

Improved LOB Usability, Accessing LOBs Using

    SQL “Character” Functions, 7-33

index organized tables, 10-66

indexes

    function-based, 1-7

    rebuilding after LONG-to-LOB migration, 8-9

indexes, restrictions, 8-11

index-organized tables

    inline storage for LOBs and (FAQ), 6-5

initialized

    checking if BFILE LOB locator is, 12-172

initializing

    BFILE column or locator variable using

        BFILENAME(), 12-24

    BLOB attribute using EMPTY\_BLOB() FAQ, 6-7

    BLOB attribute with EMPTY\_BLOB() in Java  
    (FAQ), 6-7

    during CREATE TABLE or INSERT, 10-17

    external LOBs, 2-6

    internal LOBs, See LOBs

        internal LOBs

    using EMPTY\_CLOB(), EMPTY\_BLOB()

inline

    when to use, 6-27

INSERT statements

    binds of greater than 4000 bytes, 7-14

inserting

    a row by initializing a LOB locator

        internal persistent LOBs, 10-23

    a row by initializing BFILE locator, 12-33

    a row by selecting a LOB from another table

        internal persistent LOBs, 10-20

    a row containing a BFILE by selecting BFILE  
    from another table, 12-31

    a row using BFILENAME(), 12-22

    any length data (FAQ), 6-2

    binds of more than 4,000 bytes, 10-17, 10-21,  
    10-23

    LOB value using EMPTY\_CLOB()/EMPTY\_  
    BLOB()

        internal persistent LOBs, 10-16

    row with empty LOB using JDBS (FAQ), 6-7

interfaces for LOBs, see programmatic environments

## J

---

Java, See JDBC

JDBC

    available LOB methods/properties, 3-4

    BFILE class

    BFILE streaming APIs, 3-63

    BFILE-buffering, 3-51

    BLOB and CLOB classes

    calling DBMS\_LOB package, 3-46

    changing internal LOBs with Java using objects

        oracle.sql.BLOB/CLOB, 3-45

    checking if BLOB is temporary, 11-28

    checking if CLOB is temporary, 11-29

    CLOB streaming APIs, 3-62

    creating a temporary BLOB, 11-20

    creating a temporary CLOB, 11-21

    driver to load LOBs, improving

        performance, 6-13

    empty LOBs, 3-68

    encapsulating locators

    freeing a temporary BLOB, 11-36

    freeing a temporary CLOB, 11-37

    inserting a row with empty LOB locator into  
    table, 6-7

    methods/properties for BLOB-buffering, 3-49

    methods/properties for CLOB-buffering, 3-50

    modifying BLOB values, 3-48

    modifying CLOB values, 3-49

    newStreamLob.java, 3-64

    now binds and defines VARCHAR2

- variables, 7-50
- opening and closing BFILEs, 3-56
- opening and closing CLOBs, 3-54
- opening and closing LOBs, 3-52
- OracleBlob, OracleClob do not work in
  - 8.1.x, 3-51
- reading internal LOBs and external LOBs (BFILEs) with Java, 3-45
- reading/examining BLOB values, 3-48
- reading/examining CLOB values, 3-49
- reading/examining external LOB (BFILE) values, 3-50
- referencing LOBs, 3-46
- streaming APIs for LOBs, 3-61
- syntax references, 3-46
- temporary LOB workarounds not needed, 11-11
- trimming LOBs, 3-60
- using OracleResultSet to reference LOBs, 3-46
- using OUT parameter from
  - OraclePreparedStatement to reference LOBs, 3-46
- writing to empty LOBs, 3-69
- JDBC and Empty LOBs, 3-68
- JPublisher
  - building an empty LOB in, 6-8

## L

---

LBS, See Lob Buffering Subsystem (LBS)

length

- an internal persistent LOB, 10-144
- getting BFILE, 12-154
- temporary LOB, 11-106

loading

- 1Mb into CLOB column, FAQ, 6-13
- a LOB with BFILE data, 12-46
- external LOB (BFILE) data into table, 12-42
- LOB with data from a BFILE, 10-32
- temporary BLOB with binary data from BFILE, 11-48
- temporary CLOB or NCLOB with character data from BFILE, 11-52
- temporary LOB with data from BFILE, 11-39

loading XML documents, 9-2

LOB, 5-12

LOB buffering

- BLOB-buffering with JDBC, 3-49
- buffer-enabled locators, 5-24
- disable for temporary LOBs, 11-211
- example, 5-21
- flushing for temporary LOBs, 11-204
- flushing the buffer, 5-22
- flushing the updated LOB through LBS, 5-23
- guidelines, 5-19
- OCI example, 5-25
- OCI functions, 3-16
- OCILobFlushBuffer(), 5-23
- Oracle Objects for OLE (OO4O)
  - methods for internal LOBs, 3-43
- physical structure of buffer, 5-21
- Pro\*C/C++ precompiler statements, 3-32
- Pro\*COBOL precompiler statements, 3-36
- temporary LOBs
  - CACHE, NOCACHE, CACHE READS, 11-6
  - usage notes, 5-21

LOB Buffering SubSystem (LBS)

LOB Buffering Subsystem (LBS)

- advantages, 5-18
- buffer-enabled locators, 5-23
- buffering example using OCI, 5-25
- example, 5-21
- flushing
  - updated LOB, 5-23
- flushing the buffer, 5-22
- guidelines, 5-18
- saving the state of locator to avoid reselect, 5-25
- usage, 5-21

LOB locator

- copy semantics, 2-3
- external LOBs (BFILEs), 2-3
- internal LOBs, 2-3
- reference semantics, 2-3

LOB-Based Web site, building, 14-13

LOBFILE, syntax, 9-2

LOBs, 5-17

- accessing through a locator, 2-7
- attributes and object cache, 5-17
- buffering
  - caveats, 5-18
  - pages can be aged out, 5-23

- buffering subsystem, 5-18
- buffering usage notes, 5-21
- CACHE READS setting, 4-19
- compatibility, 1-8
- datatypes versus LONG, 1-4
- external (BFILEs), 2-2
- flushing, 5-18
- in partitioned tables, 7-26
- in the object cache, 5-17
- index metadata through system views, 6-17
- inline storage, 2-5
- interfaces, See programmatic environments
- interMEDIA, 1-4
- internal
  - creating an object in object cache, 5-17
- internal LOBs
  - CACHE / NOCACHE, 7-8
  - CHUNK, 7-10
  - copy semantics, 2-3
  - ENABLE | DISABLE STORAGE IN ROW, 7-11
  - initializing, 12-108
  - locators, 2-5
  - locking before updating, 10-154, 10-194, 10-204, 10-213, 10-230, 10-242
  - LOGGING / NOLOGGING, 7-9
  - Oracle Objects for OLE (OO4O), modifying methods, 3-41
  - PCTVERSION, 7-7
  - setting to empty, 2-9
  - tablespace and LOB index, 7-7
  - tablespace and storage characteristics, 7-5
  - transactions, 2-2
- locators, 2-5, 5-2
  - cannot span transactions, 7-14
- migration issues, 1-8
- multimedia uses, B-2
- object cache, 5-17
- performing SELECT on, 2-7
- piecewise operations, 5-6
- read consistent locators, 5-2
- reason for using, 1-2
- setting to contain a locator, 2-6
- setting to NULL, 2-9
- tables

- adding partitions, 7-31
- creating, 7-28
- creating indexes, 7-30
- exchanging partitions, 7-30
- merging partitions, 7-31
- moving partitions, 7-31
- partitioning, 7-28
- splitting partitions, 7-31
- unstructured data, 1-2
- updated LOB locators, 5-5
- value, 2-5
- varying-width character data, 7-3
- locators, 2-5
  - accessing a LOB through, 2-7
  - BFILEs, 12-11
    - guidelines, 12-11
    - two rows can refer to the same file, 12-11
  - buffer-enabled, 5-24
  - cannot span transactions, 7-14
  - copying temporary LOB, 11-125
  - external LOBs (BFILEs), 2-5
  - initializing LOB or BFILE to contain, 2-6
  - LOB, cannot span transactions, 5-12
  - multiple, 5-2
  - OCI functions, 3-16, 3-27
  - Pro\*COBOL precompiler statements, 3-35
  - providing in Pro\*COBOL precompiler, 3-33
  - read consistent, 5-2, 5-10, 5-12, 5-23, 5-25, 5-26, 5-28
  - read consistent locators, 5-2
  - read consistent, updating, 5-2
  - reading and writing to a LOB using, 5-15
  - saving the state to avoid reselect, 5-25
  - see if LOB locator is initialized
    - internal persistent LOBs, 10-180
  - selecting, 2-7
  - selecting within a transaction, 5-16
  - selecting without current transaction, 5-15
  - setting column or attribute to contain, 2-6
  - temporary, SELECT permanent LOB INTO, 11-7
  - transaction boundaries, 5-14
  - updated, 5-5, 5-10, 5-22
  - updating, 5-12
- LOGGING

- migrating LONG-to-LOBs, 8-10
- LOGGING / NOLOGGING, 7-9
- LONG API
  - See LONG-to-LOB, 8-2
- LONG versus LOB datatypes, 1-4
- LONG-to-LOB Migration, 8-2
- LONG-to-LOB migration
  - ALTER TABLE, 8-6
  - changes needed, 8-22
  - clustered tables, 8-10
  - examples, 8-24
  - index organized tables, 10-66
  - LOGGING, 8-10
  - multibyte charactersets, 8-5
  - NULLs, 8-11
  - OCI, 8-3, 8-12
  - parameter passing, 8-6
  - performance, 8-42
  - PL/SQL, 8-5
  - rebuilding indexes, 8-9
  - replication, 8-10
  - space requirements, 8-9
  - triggers, 8-10
  - utldtree.sql use for PL/SQL, 8-23
- LONG-to-LOB migration
  - PL/SQL, 8-16

## M

---

- migrating
  - LONG to LOBs, 6-23
  - LONG to LOBs, see LONG-to-LOB, 8-2
  - LONG-to-LOB using ALTER TABLE, 8-6
  - LONG-to-LOBs, constraints maintained, 8-8
  - LONG-to-LOBs, indexing, 8-9
- migration, 1-8
- multi-threaded server (MTS)
  - BFILEs, 4-18, 12-10

## N

---

- national language support
  - NCLOBs, 2-2
- NCLOB parameters allowed as attributes, xlvi
- NCLOBs

- datatype, 2-2
- DBMS\_LOB, offset and amount parameters in
  - characters, 3-7
  - modify using DBMS\_LOB, 3-8
- NewStreamLob.java, 3-64
- NOCOPY restrictions, 11-10
- NOCOPY, using to pass temporary LOB parameters
  - by reference, 9-7
- non-NULL
  - before writing to LOB column make it
    - internal persistent LOBs, 10-273
- IS, 7-43
- VARCHAR2
  - and CLOBs, IS, 7-43
- NULL, 7-43
- null
  - versus zero length, in SQL92 standard, 7-43
- NULL in, 7-43
- NULL LOB, restrictions calling OCI and DBMS\_LOB
  - functions, 2-9

## O

---

- object cache, 5-17
  - creating an object in, 5-17
  - LOBs, 5-17
- OCCI
  - compared to other interfaces, 3-3
  - LOB functionality, 3-22
- OCCIBfile, 3-27
- OCCIBlob
  - copy, amount parameter must be less than length
    - of BFILE, 3-25
  - read, 3-24
  - write, 3-25
- OCCIBlob class, 3-23
- OCCIClob
  - copy, amount parameter must be less than length
    - of BFILE, 3-25
  - read, 3-24
  - read, amount parameter, 3-25
  - write, 3-25
- OCCIClob class, 3-23
- OCI
  - available LOB functions, 3-3

- functions for BFILES, 3-15, 3-27
- functions for temporary LOBs, 3-15, 3-27
- functions to modify internal LOB values, 3-14, 3-26
- functions to open/close internal and external LOBs, 3-16, 3-28
- functions to read or examine internal and external LOB values, 3-15, 3-27
- LOB buffering example, 5-25
- LOB locator functions, 3-16, 3-27
- Lob-buffering functions, 3-16
- locators, 2-8
- NCLOB parameters, 3-13, 3-25
- now binds and defines VARCHAR2 variables with LOBs, 7-50
- OCILobFileGetLength
  - CLOB and NCLOB input and output length, 3-12
- OCILobRead
  - varying-width CLOB and NCLOB input and amount amounts, 3-12
- OCILobWrite
  - varying-width CLOB and NCLOB input and amount amounts, 3-12, 3-25
- offset and amount parameter rules
  - fixed-width character sets, 3-11, 3-24
- seeIfLOBOpen and how to use main(), 3-17
- setting OCILobRead, OCILobWrite to OCI\_UCS2ID, 3-11
- temporary LOBs, 11-9
- temporary lobs can be grouped into logical buckets, 11-6
- using in LONG-to-LOB migration, 8-12
- using to work LOBs, 3-10
- OCI functions on a NULL LOB restriction, 2-9
- OCIBindByName(), 7-14
- OCIBindByPos(), 7-14
- OCIDuration(), 11-6
- OCIDurationEnd(), 11-6, 11-9, 11-31
- OCILobAssign(), 5-20, 11-8
- OCILobFileSetName(), 12-6, 12-11
- OCILobFlushBuffer(), 5-23
- OCILOBFreeTemporary(), 11-31
- OCILobGetLength(), 12-109
- OCILobLoadFromFile(), 12-48
- OCILobRead
  - BFILES, 12-109
- OCILobRead(), 10-99, 10-110, 11-75, 12-109
  - amount, 6-5
  - to read large amounts of LOB data, 10-72
- OCILobWrite(), 11-169
  - to write large amounts of LOB data, 10-84
- OCILobWriteAppend(), 10-204
- OCIObjectFlush(), 12-11
- OCIObjectNew(), 12-11
- OCISetAttr(), 12-12
- OLEDDB, 3-70, 13-2
- OO4O, See Oracle Objects for OLE (OO4O)
- open
  - checking for open BFILES, 12-78
  - checking for open BFILES with FILEISOPEN(), 12-80
  - checking if BFILE is open with ISOPEN, 12-86
  - checking if temporary LOB is, 11-57
  - seeing if a LOB is open, 10-53
- opening
  - BFILES, 12-63
  - BFILES using FILEOPEN, 12-65
  - BFILES with OPEN, 12-70
- opening and closing LOBs
  - using JDBC, 3-52
- ora\_21560
  - DBMS\_LOB.write() to temporary LOB, 11-170
- ORA-17098
  - empty LOBs and JDBC, 3-69
- OraBfile, See Oracle Objects for OLE (OO4O)
- OraBlob, See Oracle Objects for OLE (OO4O)
- Oracle Call Interface, See OCI
- Oracle Objects for OLE (OO4O)
  - available LOB methods/properties, 3-4
  - internal LOB buffering, 3-43
  - methods and properties to access data stored in BLOBs, CLOBs, NCLOBs, and BFILES, 3-39
  - modifying internal LOBs, 3-41
  - opening/closing external LOBs (BFILES), 3-42
  - OraBfile example
  - OraBlob example
  - OraBlob, OraClob, and OraBfile encapsulate locators, 3-38
  - properties for operating on external LOBs

- (BFILES), 3-44
- properties for operating on LOBs, 3-43
- reading/examining internal LOB and external LOB (BFile) values, 3-42
- read-only methods for external LOBs (BFILES), 3-43
- syntax reference, 3-37
- using Clone method to retain locator independent of dynaset, 3-38
- Oracle Provider for OLEDB, see OraOLEDB, 13-2
- OraclePreparedStatement, See JDBC
- OracleResultSet, See JDBC
- oracle.sql.BFILE
  - BFILE-buffering, 3-51
  - JDBC methods to read/examine BFILES, 3-50
- oracle.sql.BLOB
  - for modifying BLOB values, 3-48
  - reading/examining BLOB values, 3-48
  - See JDBC
- oracle.sql.BLOBs
  - BLOB-buffering
- oracle.sql.CLOB
  - CLOB-buffering
  - JDBC methods to read/examine CLOB values, 3-49
  - modifying CLOB values, 3-49
- oracle.sql.CLOBs
  - See JDBC
- ORaOLEDB
  - AppendChunk(), 13-3
- OraOLEDB, 3-70, 13-1, 13-2
  - GetChunk(), 13-3

## P

---

- partitioned index-organized tables
  - for LOBs, 5-29
  - restrictions for LOBs, 5-31
- pattern
  - check if it exists in BFILE using instr, 12-137
  - see if it exists IN LOB using (instr) internal persistent LOBs, 10-137
  - temporary LOBs
    - checking if it exists, 11-99
- PCTVERSION, 7-7
- performance
  - assigning multiple locators to same temporary LOB, impacts, 11-8
  - chunks versus reading, 6-26
  - creating temporary LOBs in called routine (FAQ), 6-27
  - disk striping of LOBs, B-3
  - guidelines
    - frequently asked questions (FAQ), 6-26
    - reading/writing large data chunks, 9-5
    - reading/writing large data chunks, temporary LOBs, 9-8
  - improving BLOB and CLOB, when loading with JDBC driver, 6-13
  - improving loading, when using Veritas, 6-24
  - inlining and when its a good idea to use (FAQ), 6-27
  - LONG-to-LOB migration, 8-42
  - OCI and temporary LOBs, 11-9
    - when using SQL semantics with LOBs, 7-43
  - PIOT, 5-29
  - PL/SQL, 3-2
    - and LOBs, semantics changes, 7-45
    - assigning one LOB to another, 11-127
    - changing locator-data linkage, 7-48
    - CLOB variables in, 7-48
    - CLOB variables in PL/SQL, 7-48
    - CLOB versus VARCHAR2 comparison, 7-50
    - CLOBs passed in like VARCHAR2s, 7-47
    - defining a CLOB Variable on a VARCHAR, 7-46
    - freeing temporary LOBs automatically and manually, 7-49
    - inserting Word document into a BLOB, 14-5
    - OCI and Java LOB interactions, 7-50
    - performance when reassigning locators in temporary LOBs, 11-10
      - using in LONG-to-LOB migration, 8-16
  - polling, 10-72, 10-84, 10-110, 10-204, 11-169
  - populating your data repository, 14-4
  - Pro\*C/C++ precompiler
    - available LOB functions, 3-3
    - LOB buffering, 3-32
    - locators, 3-31
    - modifying internal LOB values, 3-30



- opening and closing internal LOBs and external LOBs (BFILES), 3-32
- providing an allocated input locator pointer, 3-29
- reading or examining internal and external LOB values, 3-30
- statements for BFILES, 3-31
- statements for temporary LOBs, 3-31
- Pro\*COBOL precompiler
  - available LOB functions, 3-3
  - LOB buffering, 3-36
  - locators, 3-35
  - modifying internal LOB values, 3-34
  - providing an allocated input locator, 3-33
  - reading or examining internal and external LOBs, 3-35
  - statements for BFILES, 3-35
  - temporary LOBs, 3-35
- programmatically environments
  - available functions, 3-3
  - compared, 3-3
- programmatically environments for LOBs, 3-2
- putChars(), 6-10

## R

---

- read consistency
  - LOBs, 5-2
- read consistent locators, 5-2, 5-10, 5-12, 5-23, 5-25, 5-26, 5-28
- reading
  - BFILES
    - specify 4 Gb-1 regardless of LOB, 12-109
  - data form a temporary LOB, 11-74
  - data from a LOB
    - internal persistent LOBs, 10-109
  - large amounts of LOB data using streaming, 10-72
  - large data chunks, performance guidelines, 9-5
  - large data chunks, temporary LOBs, 9-8
  - portion of BFILE data using substr, 12-119
  - portion of LOB using substr
    - internal persistent LOBs, 10-120
  - portion of temporary LOB, 11-85
  - small amounts of data, enable buffering, 10-252

- Recordsets, ADO, 13-3
- redo space
  - during LONG-to-LOB migration, prevent generation, 9-10
- reference semantics, 2-3, 10-21
  - BFILES enables multiple BFILE columns for each record, 12-6
- replication, 8-10
- restrictions
  - binding of data, removed for INSERTS and UPDATES, xlix
  - binds of more than 4000 bytes, 7-16
  - cannot call OCI or DBMS\_LOB functions on a NULL LOB, 2-9
  - clustered tables, 8-10
  - indexes, 8-11
  - on LOBs, 4-16
  - partitioned index-organized tables and LOBs, 5-31
  - removed, xlvi
  - replication, 8-10
  - triggers, 8-10
- retrieving data, 14-10
- round-trips to the server, avoiding, 5-18, 5-25
- Rowset, OLEDB, 13-2
- rules for using directory objects and BFILES, 4-1

## S

---

- searching for data, 14-6
- security
  - BFILES, 12-7
  - BFILES using SQL DDL, 12-8
  - BFILES using SQL DML, 12-9
- segment
  - LOB restriction, must be at least 3 blocks, 4-18
- SELECT statement
  - FOR UPDATE, 2-7
  - read consistency, 5-2
- selecting a permanent LOB INTO a temporary LOB locator, 11-7
- semantics
  - copy-based for internal LOBs, 10-21
  - pseudo-reference, 11-8
  - reference based for BFILES, 12-6

- value, 11-8
- SESSION\_MAX\_OPEN\_FILES parameter, 4-2, 12-64, 12-78
- setData
  - setting to EMPTY\_BLOB() using JPublisher, 6-8
- setting
  - internal LOBs to empty, 2-9
  - LOBs to NULL, 2-9
  - overrides for NLS\_LANG variable
- space requirements, LONG-to-LOB migration, 8-9
- spatial cartridge and user-defined aggregates, 7-53
- SQL
  - “Character” Functions, improved, 7-33
  - features where LOBs cannot be used, 7-39
  - functions and operators, returning CLOB values from, 7-41
  - RAW type and BLOBs, 7-43
  - where LOBs cannot be used, 7-39
- SQL DDL
  - BFILE security, 12-8
- SQL DML
  - BFILE security, 12-9
- SQL Loader
  - loading InLine LOB data, 4-8
  - performance for internal LOBs, 4-7
- SQL semantics on LOBs
  - non-supported functionality, 7-35
- SQL\*Loader
  - conventional path load, 9-3
  - direct-path load, 9-3
  - LOBFILE, 9-2
- storing
  - CLOBs Inline, 6-21
  - greater than 4GB LOBs in database (FAQ), 6-27
  - LOB storage clause, when to use with varrays, 6-21
- storing images in a BFILE versus BLOB, 6-18
- stream
  - reading
    - temporary LOBs, 11-75
  - writing, 11-169
- streaming, 10-84, 10-99
  - do not enable buffering, when using, 10-252
  - write, 10-213
- streaming APIs

- NewStreamLob.java, 3-64
  - using JDBC and BFILES, 3-63
  - using JDBC and CLOBs, 3-62
  - using JDBC and LOBs, 3-61
- symbolic links, rules with DIRECTORY objects and BFILES, 4-1
- system owned object, See DIRECTORY object

## T

---

- tablespace
  - LOB index in same, FAQ, 6-16
  - specified with ENABLE STORAGE IN ROW, FAQ, 6-17
  - temporary, 11-6
  - temporary LOB data stored in temporary, 11-5
- temporary BLOB
  - checking if temporary using JDBC, 11-28
  - freeing, using JDBC, 11-36
  - using JDBC to create, 11-20
- temporary CLOB
  - checking if temporary using JDBC, 11-29
  - freeing using JDBC, 11-37
  - using JDBC to create, 11-21
- temporary LOBs
  - character set ID, 11-144
  - checking if LOB is temporary, 11-23
  - data stored in temporary tablespace, 11-5
  - DBMS\_LOB available
    - functions/procedures, 3-9
  - disable buffering
  - explicitly freeing before overwriting it with permanent LOB locator, 11-7
  - features, 11-7
  - inline and out-of-line not used, 11-5
  - JDBC, 11-11
  - lifetime and duration, 11-5
  - locators can be IN values, 11-4
  - locators operate as with permanent LOBs, 11-4
  - memory handling, 11-6
  - OCI and logical buckets, 11-6
  - OCI functions, 3-15, 3-27
  - performance, 11-8
  - Pro\*C/C++ precompiler embedded SQL statements, 3-31

- Pro\*COBOL precompiler statements, 3-35
- reside on server not client, 11-5
- similar functions used to permanent LOBs, 11-5
- SQL DML does not operate on, 11-4
- transactions and consistent reads not supported, 11-5
- trimming, 11-179
- write append to, 11-160
- temporary tablespace
  - for binds of more than 4000 bytes, 7-14
- temporaryClob.java
  - no longer used- deprecated class, 11-38
- TO\_BLOB(), TO\_CHAR(), TO\_NCHAR(), 7-47
- TO\_CLOB()
  - converting VARCHAR2, NVARCHAR2, NCLOB to CLOB, 7-47
- TO\_LOB
  - limitations, 10-66
- TO\_NCLOB(), 7-47
- transaction boundaries
  - LOB locators, 5-14
- transaction IDs, 5-15
- transactions
  - external LOBs do not participate in, 2-3
  - IDs of locators, 5-14
  - internal LOBs participate fully, 2-2
  - LOB locators cannot span, 5-12
  - LOBs locators cannot span, 7-14
  - locators with non-serializable, 5-15
  - locators with serializable, 5-15
  - migrating from, 5-23
- triggers
  - LOB columns with, how to tell when updated (FAQ), 6-3
  - LONG-to-LOB migration, 8-10
- trimming
  - LOB data
    - internal persistent LOBs, 10-229
    - temporary LOB data, 11-179
  - trimming LOBs using JDBC, 3-60
- Trusted Oracle and user-defined aggregates, 7-53

## U

---

UDAGs, see user-defined aggregates

## UNICODE

- VARCHAR2 and CLOBs support, 7-38
- unstructured data, 1-2
- UPDATE statements
  - binds of greater than 4000 bytes, 7-14
- updated locators, 5-5, 5-10, 5-22
- updating
  - any length data (FAQ), 6-2
  - avoid the LOB with different locators, 5-8
  - BFILEs by selecting a BFILE from another table, 12-196
  - BFILEs using BFILENAME(), 12-193
  - by initializing a LOB locator bind variable
    - internal persistent LOBs, 10-277
  - by selecting a LOB from another table
    - internal persistent LOBs, 10-275
  - LOB values using one locator, 5-8
  - LOB values, read consistent locators, 5-2
  - LOB with PL/SQL bind variable, 5-10
  - LOBs using SQL and DBMS\_LOB, 5-6
  - locators, 5-12
  - locking before, 10-154
  - locking prior to, 10-194, 10-230, 10-242
  - with EMPTY\_CLOB()/EMPTY\_BLOB()
    - internal persistent LOBs, 10-271
- UPLOAD\_AS\_BLOB and DAD, 6-30
- use cases
  - full list of internal persistent LOBs, 10-2
- user-defined aggregates (UDAGs) and LOBs, 7-53
- utldtree.sql, 8-23

## V

---

- value of LOBs, 2-5
- VARCHAR2
  - accessing CLOB data when treated as, 7-46
  - also RAW, applied to CLOBs and BLOBs, 7-40
  - defining CLOB variable on, 7-46
  - OCI and JDBC now bind and define variables to SQL, PL/SQL with LOBs, 7-50
- VARCHAR2s
  - on CLOBs, SQL functions and operators
    - for, 7-35
- VARRAYs
  - See creating VARRAYs

- varrays
  - including LOB storage clause to create tables (FAQ), 6-21
- varying-width character data, 2-4
- Veritas with LOBs, 6-24
- views on DIRECTORY object, 12-9
- Visual Basic, See Oracle Objects for OLE(OO4O)

## **W**

---

- Web sites, building LOB-based, 14-13
- write
  - streaming, 11-169
- write appending
  - to temporary LOBs, 11-160
- writing
  - data to a LOB
    - internal persistent LOBs, 10-212
    - data to a temporary LOB, 11-168
  - large data chunks, performance guidelines, 9-5
  - large data chunks, temporary LOBs, 9-8
  - singly or piecewise, 10-204
  - small amounts of data, enable buffering, 10-252

## **X**

---

- XML
  - stored in LOBs, 1-3
- XML, loading, 9-2